VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital System Design Lab

School of Electronics Engineering

ECE 2003 – DIGITAL LOGIC DESIGN LAB MANUAL

FALL 2019-20

ECE2003 – DIGITAL LOGIC DESIGN

SYLLABUS

S. No.	List of Challenging Experiments (Indicative)	CO: 7
1	Characteristics of Digital ICs (Hardware)	4 hours
2	Implementation of Combinational Logic Design using MUX/Decoder ICs (Hardware)	4 hours
3	Design and Implementation of various data path elements Adders/Multipliers (Hardware)	4 hours
4	Design and Implementation of various data path elements like Adders/Multipliers and combinational Logic circuits like Multipliers (Mandatory: Verilog Modelling, Simulation and Synthesis. FPGA implementation (optional)	6 hours
5	Design and implementation of simple synchronous sequential circuits like Counters / Shift registers (Hardware)	2 hours
6	Complex state machine design (Simulation and Synthesis)	4 hours
7	Simple processor design (Simulation and Synthesis)	6 hours
	Total Laboratory Hours:	**30 hours**

LIST OF EXPERIMENTS

List of Application Oriented Experiments

Expt. No.	Experiment title
1.	**Study and Verification of Basic Logic Gates & Implementation of Simple Boolean Expression** **(a).** Identify appropriate logic gate required to design a bank safety locker system. The locker opens only if both finger PIN number and finger print of the account holder matches. Otherwise, locker remains in locked state.(Hint: AND) **(b).** Identify an appropriate logic gate required to design a simple intruder detection system. The system consist of two PIR sensors to detect intruders. One of the sensor placed at main door and other placed at back door. If either one of the sensor detects human presence, it activate alarm. (Hint : OR) **(c).** Construct and verify the logic of NOT gate using its truth table. **(d).** Two tanks store certain liquid chemicals that are required in a manufacturing process. Each tank has a sensor that detects when the chemical level drops to 25% of full. The sensors produce a HIGH level of 5 V when the tanks are more than one-quarter full. When the volume of chemical in a tank drops to one-quarter full, the sensor puts out a LOW level of 0 V. Identify an appropriate logic gate required to design this system.(Hint: NAND) **(e).** Construct and verify the logic of 2-input NOR gate using its truth table. **(f).** Construct and verify the logic of 2-input XOR gate using its truth table. **(g).** Construct and verify the logic of 3-input AND gate using its truth table. **(h).** Construct and verify the logic of 3-input NAND gate using its truth table. **(i).** Realize the Boolean expression **Y=A.B'+A.C'** using basic logic gates.
2.	**Design and Implementation of Adder & Subtractor** **(a).** Design a half adder circuits & verify the truth table using basic logic gates. **(b).** Design a half subtractor circuits & verify the truth table using basic logic gates. **(c).** Design a partial simplified Arithmetic Logic Unit (ALU) using basic logic gates, which is capable of performing the 1-bit addition operation with the inclusion of carry as an additional input. (Hint: Full Adder) **(d).** Design a partial simplified Arithmetic Logic Unit (ALU) using basic logic gates, which is capable of performing the 1-bit subtraction operation with the inclusion of borrow as an additional input. (Hint: Full Subtractor)

3. Design and Implementation of 4:1 Mux & 4:2 Decoder

(a). In a communication system, multiple signals are transmitted on a single transmission line. Assume audio, image, data and video are four signal to be transmitted on a single transmission line by using appropriate signal selection line. Implement a digital circuit using basic logic gates to meet this requirement.(Hint: 4x1 Multiplexer)

(b). Control & timing unit of a Microprocessor has a decoding unit to decode the program instructions in order to activate the specific control lines such that different operations in the ALU are carried out as shown below. Assume program instructions are represented in 2-bit format. (Hint: 4:2 Decoder)

Program Instruction		Control signal
0	0	Activate Add operation in ALU
0	1	Activate Subtract operation in ALU
1	0	Activate compare operation in ALU
1	1	Activate logical operation in ALU

4. Design and Implementation of Magnitude Comparator, Parity Generator & Checker

(a). Design a control unit for a sea salt packing machine to pack a salt bag of 2kg each. Current weight of the bag is measured and represented in 2-bit binary number then compared with the reference value (2kg). If the salt bag is less/greater than 2kg control signal generated to add/remove excess salt to/from salt bag. If the salt bag weight is exactly 2kg then control signal generated to make a pack. Design and Implement an appropriate digital circuit using basic logic gates to meet this requirement. (Hint: 2-Bit Magnitude comparator)

(b). A Bluetooth module is interfaced with Arduino Uno board and transmitting its 4-bit data continuously to the receiver section. This 4-bit data comprises of 3-bit temperature sensor data and 1-bit for parity (even). Since this system operating under a noisy environment, there is a possibility of error prone into the actual values of transmitting data. Design a suitable combinational logic circuit in the transmitter and receiver section to detect the data is valid or not.(Hint: Parity Generator and Checker)

5. Design and Implementation SR, JK, D & T Flip-Flops

(a). Construct SR Flip-Flops using basic logic gates and verify its logic using truth table.

(b). Construct JK Flip-Flops using basic logic gates and verify its logic using truth table.

	(c). In a Microprocessor design, signals that flow through different paths arrive at different time. This could cause many problems when these signals have to interact with each other. Identify and implement a suitable flop-flop required to synchronize these signals using basic logic gates. (Hint: D-FF) **(d).** Counters are the digital circuits, which are used to count the number of events. Identify and implement the most suitable Flip-flip required to design a counter unit using basic logic gates. (Hint: T-FF)
6.	**Design and Implementation of 4-Bit Shift Register (SISO, SIPO, PIPO)** **(a).** In modern computer system, it is essential to operating the computing unit in optimized manner. Especially, shift registers are extensively used to perform optimized binary multiplication and division. This is due to the fact that the shift of data bit by one position towards right causes the number to be divided by 2 while the left-shift of the data bit by one place in the shift register multiplies the number by 2. For example, consider a 4-bit shift register with the content 0110, which is equal to 6 in decimal. If the number shifts left by one-bit, then one gets 1100, which is 12 (= 6 × 2) in decimal. Design and implement the suitable 4-bit shift register logic circuit for the given requirement using appropriate flip flop.(Hint: SISO) **(b).** In general, many microprocessors/microcontrollers handle data as bytes (8 bits) or words (16 bit, 32 bit) as a preferred format. However, many external interfacing device prefer to operate on serial format instead of parallel. Whenever any external serial device communicate with microprocessor/microcontrollers, there should be a serial-to-parallel converter unit to convert the serial data into parallel data. Design and implement the suitable 4-bit serial to parallel converter logic circuit for the given requirement using appropriate flip-flop. (Hint: SIPO) **(c).** A free-running analog-to-digital converter is one that updates its digital output(for the given analog signal) as often as it can, not waiting for any prompting from another device. If we were to connect a free-running ADC to a microprocessor or microcontroller, we would need some way to sample the ADC's output at specific times and hold that binary number long enough to register it. Otherwise, the ADC may update its output in the middle of one of the computer's "input" cycles, possibly resulting in corrupted data. We could build such a sample-and-hold circuit out of flip-flops, which could shift register inputs multiple bits of data all at once, and transfers that data to its output lines all at once, at the command of a clock pulse. Design and implement the suitable 4-bit shift register to serve for the sample and hold circuit requirement using appropriate flip-flop. (Hint: PIPO)

7.	**Design and Implementation of 4-Bit Ripple Counter and MOD-12 counter** **(a).** In the packaging department of a cricket ball manufacturing company, the balls roll down on a conveyor and get filled into the empty boxes for shipment. Capacity of each box is 16 balls. Each ball is allowed to pass through IR scanner, which generates one-clock pulse for every ball that crosses the scanner. Design an appropriate counter using Flip Flops to count the clock pulse generated from scanner to indicate whether the box is full or not, so that next empty box can be moved into the position. (Hint: 4-bit Ripple Counter) **(b).** A simple human counter system for an elevator overload indication module constructed using optical IR sensor, buzzer, microcontroller and seven segment LED display. Number of persons entering into elevator is detected by IR sensor, which generates one-clock pulse per person while entering into elevator. By default, it must display value 0 and increment the number on the display by one for every time a human enters into the elevator. Assume the maximum capacity of the elevator as 12 persons and if it exceed it alarm buzzer. Design an appropriate counter using Flip Flops to indicate overload condition. (Hint: MOD-12 Counter)
8.	**Introduction to Verilog HDL and Verification of Simple Boolean Expression** **(a).** To understand the basics of Verilog HDL and concepts of various Verilog modelling such as Gate-level, Dataflow, Behavioural and Switch-level modelling. **(b).** In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point. Design a digital logic circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point. Write a Verilog HDL code in gate level modelling for the proposed design and verify its logic by simulation using Xilinx ISE Simulator.
9.	**Design and Simulate Full Adder & 4-Bit Parallel Adder** A digital circuit designer needs to design a partial simplified Arithmetic Logic Unit (ALU) in Verilog HDL to perform a 4-bit addition operation on two operands, but he had already created a full adder logic for another application. Help him to realize partial simplified ALU design to perform 4-bit addition by instantiating full adder logic. Verify the output logic by simulation using Xilinx ISE Simulator. (Hint: 4-bit Parallel Adder using Full adder)

10.	**Design and Simulate 4-bit Multiplier** Multiplication is one of the essential operation to be carried out by ALU of a 4004 microprocessor. Design and simulate multiplier unit for 4-bit ALU unit with the help of basic logic gates and 4-bit parallel adders. Verify the output logic by simulation using Xilinx ISE Simulator.
11.	**Design and Simulate SR, JK, D & T Flip-Flops** **(a).** Construct SR Flip-Flops using basic logic gates and verify its output logic by simulation using Xilinx ISE Simulator. **(b).** Construct JK Flip-Flops using basic logic gates and verify its output logic by simulation using Xilinx ISE Simulator. **(c).** In a Microprocessor design, signals that flow through different paths arrive at different time. This could cause many problems when these signals have to interact with each other. Identify and implement a suitable flop-flop required to synchronize these signals using basic logic gates. Verify its output logic by simulation using Xilinx ISE Simulator. (Hint: D-FF) **(d).** Counters are the digital circuits, which are used to count the number of events. Identify and implement the most suitable Flip-flip required to design a counter unit using basic logic gates. Verify its output logic by simulation using Xilinx ISE Simulator. (Hint: T-FF)
12.	**Design and Simulate PISO Shift Register and Up/Down Counters** **(a).** In many communication systems, Serial data transmission is preferred for long distance communication due to its economical value in terms of the wires used. This necessitates parallel-to-serial conversion at the sender-end for which shift registers can be used. Design and implement the suitable 4-bit parallel to serial converter logic circuit for the given requirement using appropriate flip-flop. (Hint: PISO) **(b).** A Microcontroller based bidirectional visitor counter is shown in figure-1 to count the number of visitors to the college library. Depending upon the signal from the IR sensors, this system detects the entry and exit of the visitor. Figure-2 shows sensor setup at the library entrance for bidirectional visitor counter. The logic control circuit block shown in figure – 1 keeps the record on of number of visitor entered and number of visitor exited. Microcontroller calculate number of person present in the library by subtracting number of people exited from number of people entered. Then, it displays the number of visitors present in the library on the display device-using microcontroller. For simplicity assume the system can count maximum of 16 people. (Hint: 4-bit up/down counter)

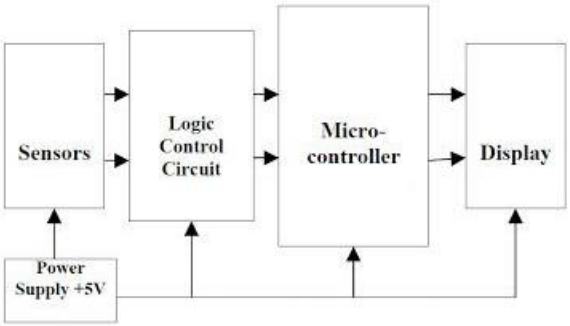


Figure-1: Block diagram of Bi-directional Visitor counter

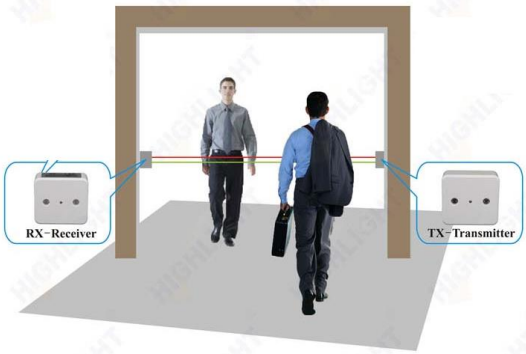


Figure-2 Illustration of sensor setup of Bi-directional visitor counter

13. Design and Simulate Simple Arithmetic Logic Unit

Design and simulate a simple 8-bit microprocessor's ALU unit which performs arithmetic and logical operations (shown in below table) on two 8-bit inputs [7:0]A and [7:0]B. Write the Verilog HDL code in behavioural modelling for the above mentioned design and verify the output using sample test cases.

S. No	ALU_Sel	Operation	Description
1	0000	A+1	Increment A by 1
2	0001	A+B	Addition
3	0010	A-1	Decrement A by 1
4	0011	A-B	Subtraction
5	0100	A*B	Multiplication
6	0101	A==B	Equality
7	0110	A > B	Greater than
8	0111	A < B	Lesser than
9	1000	~ A	Logical NOT
10	1001	A & B	Logical AND
11	1010	A | B	Logical OR
12	1011	~(A & B)	Logical NAND
13	1100	~(A | B)	Logical NOR
14	1101	A ^ B	Logical XOR
15	1110	A >> 1	Right shift by 1
16	1111	A << 1	Left shift by 1

14. Design and Simulate Chocolate Vending Machine using Finite State Machine

Design a simple vending machine, which dispatches a chocolate after deposition of 15 rupees. The machine has only one coin slot to receive coins from customers' one coin at a time. In addition, the machine receives only 10 (D) or 5 (N) rupee coin and it does not give any change if total deposited amount exceeds 15 rupees. Simulate and verify the designed FSM using Verilog HDL.

HARDWARE EXPERIMENTS

Introduction to Digital Laboratory Equipments & IC's

The Breadboard

The breadboard consists of two terminal strips and two bus strips. Each bus strip has two rows of contacts. Each of the two rows of contacts are a node. That is, each contact along a row on a bus strip is connected together (inside the breadboard). Bus strips are used primarily for power supply connections, but are also used for any node requiring a large number of connections. Each terminal strip has 60 rows and 5 columns of contacts on each side of the centre gap. Each row of five contacts is a node.

You will build your circuits on the terminal strips by inserting the leads of circuit components into the contact receptacles and making connections with 22-26 gauge wire. There are wire cutter/strippers and a spool of wire in the lab. It is a good practice to wire +5V and 0V power supply connections to separate bus strips.

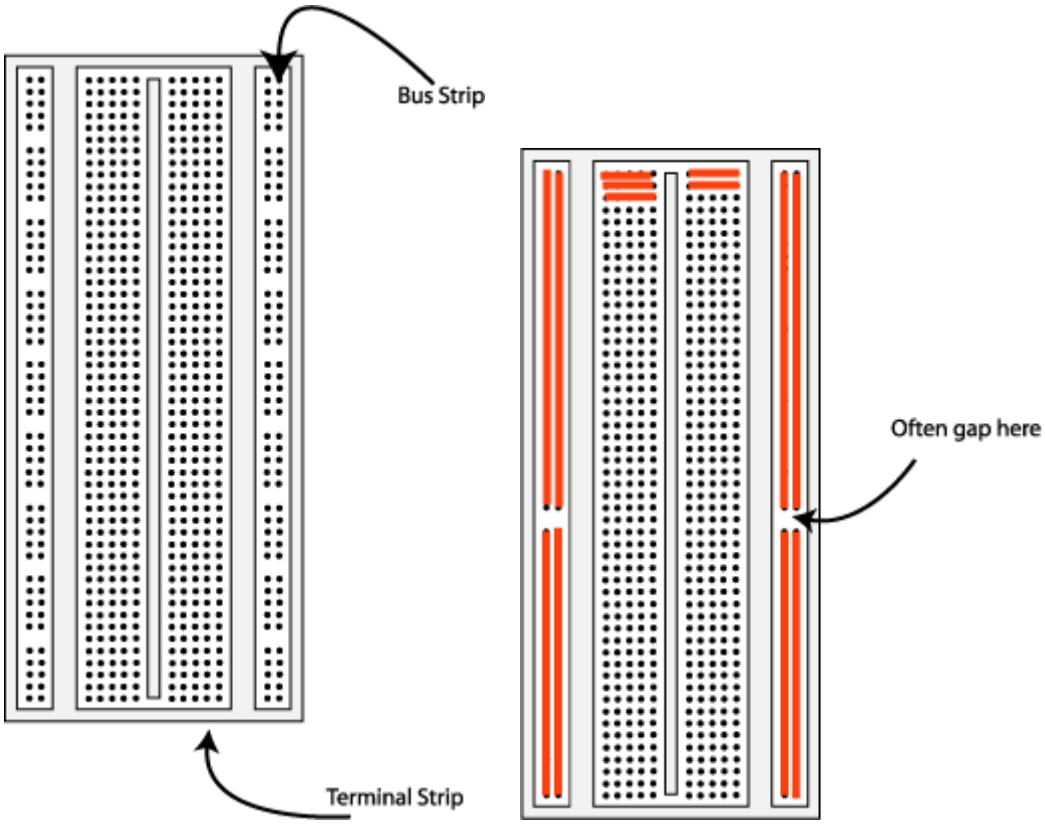


Fig 1. The breadboard. The lines indicate connected holes.

The 5V supply **MUST NOT BE EXCEEDED** since this will damage the ICs (Integrated circuits) used during the experiments. Incorrect connection of power to the ICs could result in them exploding or becoming very hot - with the **possible serious injury occurring to the people working on the experiment! Ensure that the power supply polarity and all components and connections are correct before switching on power.**

Safety Instructions

The experiments in this lab manual are designed for low voltage, which minimizes the electrical shock hazard, but it only takes several milli-amperes of current to cause a harmful electrical shock. Safety must always be first. Below are several general safety rules for all digital experiments and activities in the laboratory.

1. Avoid direct contact with any power source. Turn off all power sources when not needed.
2. When hooking up a circuit, connect to the power source last, while power is off.
3. Before making changes in a circuit, turn off or disconnect the power first.
4. Never work alone in the laboratory. Perform the experiment under instructor or lab technician supervision.
5. When changing a powered up connection, use only one hand. Never touch two points in the circuit that are at different voltages.
6. Know that the circuit and connections are correct before applying power to the circuit. If needed have the instructor review the circuit before applying power.
7. Know the location of the emergency power-off switch at each bench.
8. Keep the work area around the circuit and test equipment neat and free of clutter.
9. Remove all jewellery that can be seen before working on any experiment.

Building the Circuit

Throughout these experiments, we will use TTL chips to build circuits. The steps for wiring a circuit should be completed in the order described below:

1. Make sure the Trainer Kit power is off before you build anything!
2. Connect the +5V and ground (GND) leads of the power supply to the power and ground bus strips on your breadboard.
3. Plug the chips you will be using into the breadboard. Point all the chips in the same direction with pin 1 at the upper-left corner. (Pin 1 is often identified by a dot or a notch next to it on the chip package)
4. Connect +5V and GND pins of each chip to the power and ground bus strips on the breadboard.
5. Select a connection on your schematic and place a piece of hook-up wire between corresponding pins of the chips on your breadboard. It is better to make the short connections before the longer ones. Mark each connection on your schematic as you go, so as not to try to make the same connection again at a later stage.
6. Get one of your group members to check the connections, **before you turn the power on**.
7. If an error is made and is not spotted before you turn the power on. Turn the power off immediately before you begin to rewire the circuit.
8. At the end of the laboratory session, collect you hook-up wires, chips and all equipment and return them to the demonstrator.

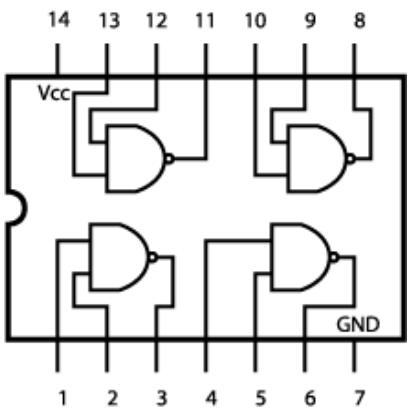
Common Causes of Problems

1. Not connecting the ground and/or power pins for all chips.
2. Not turning on the power supply before checking the operation of the circuit.
3. Leaving out wires.
4. Plugging wires into the wrong holes.
5. Driving a single gate input with the outputs of two or more gates
6. Modifying the circuit with the power on.

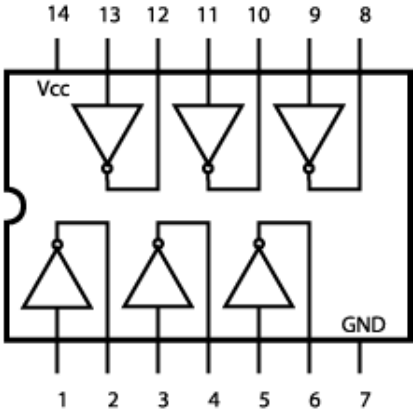
In all experiments, you will be expected to obtain all instruments, leads, components at the start of the experiment and return them to their proper place after you have finished the experiment. Please inform the technician if you locate faulty equipment. If you damage a chip, inform a demonstrator, do not put it back in the box of chips for somebody else to use.

Example Implementation of a Logic Circuit

Build a circuit to implement the Boolean function F = (A'.B').



Quad 2 Input 7400



Hex 7404 Inverter

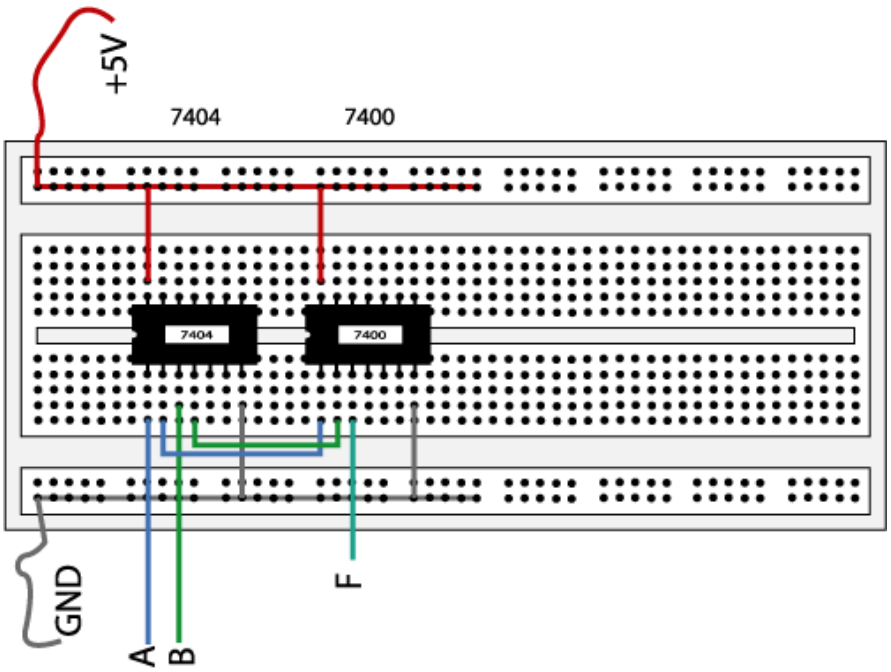


Fig 2. The complete designed and connected circuit

Expt. No. 1	Study and Verification of Basic Logic Gates & Implementation of Simple Boolean Expression

AIM:

(a). Identify appropriate logic gate required to design a bank safety locker system. The locker opens only if both finger PIN number and finger print of the account holder matches. Otherwise, locker remains in locked state (Hint: AND Gate)

(b). Identify an appropriate logic gate required to design a simple intruder detection system. The system consist of two PIR sensors to detect intruders. One of the sensor is placed at main door and other is placed at back door. If either one of the sensor detects human presence then it activate alarm unit. (Hint: OR Gate)

(c). Construct and verify the logic of NOT gate using its truth table.

(d). Two tanks store certain liquid chemicals that are required in a manufacturing process. Each tank has a sensor that detects when the chemical level drops to 25% of full. The sensors produce a HIGH level of five V when the tanks are more than one-quarter full. When the volume of chemical in a tank drops to one-quarter full, the sensor puts out a LOW level of 0 V. Identify an appropriate logic gate required to design this system.(Hint: NAND Gate)

(e). Construct and verify the logic of 2-input NOR gate using its truth table.

(f). Construct and verify the logic of 2-input XOR gate using its truth table.

(g). Construct and verify the logic of 3-input AND gate using its truth table.

(h). Construct and verify the logic of 3-input NAND gate using its truth table.

(i). Realize the simple Boolean expression using basic logic gates. **Y=A.B'+A.C'**

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	2-INPUT AND GATE	IC 7408	1
2.	2-INPUT OR GATE	IC 7432	1
3.	1-INPUT NOT GATE	IC 7404	1
4.	2-INPUT NAND GATE	IC 7400	1
5.	2-INPUT NOR GATE	IC 7402	1
6.	2-INPUT XOR GATE	IC 7486	1
7.	3-INPUT AND GATE	IC7411	1
8.	3-INPUT NAND GATE	IC7410	1
9.	DIGITAL TRAINER KIT	-	1
10.	Connecting wires	-	few

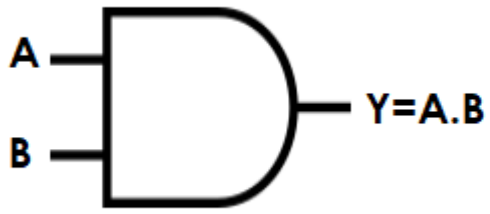
THEORY:

Logic gates are electronic circuits, which perform logical functions on one or more inputs to produce one output. There are seven logic gates which includes AND, OR, NOT, NAND, NOR, XOR and XNOR. When all the input combinations of a logic gate are written in a series and their corresponding outputs written along them, then this input/ output combination is called Truth Table. Following section, explain the operation of each basic logic gate (except XNOR) along with their symbol, truth table and their pin diagram.

(a). AND GATE:

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. The output is low level when any one of the inputs is low. A dot (.) is used to show the AND operation i.e. A.B. IC 7408 is the two Inputs AND gate IC.

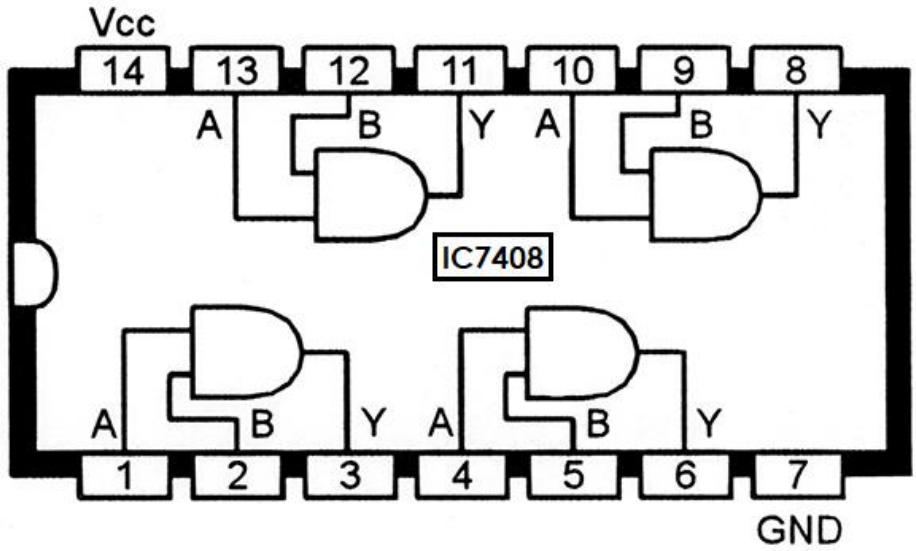
SYMBOL

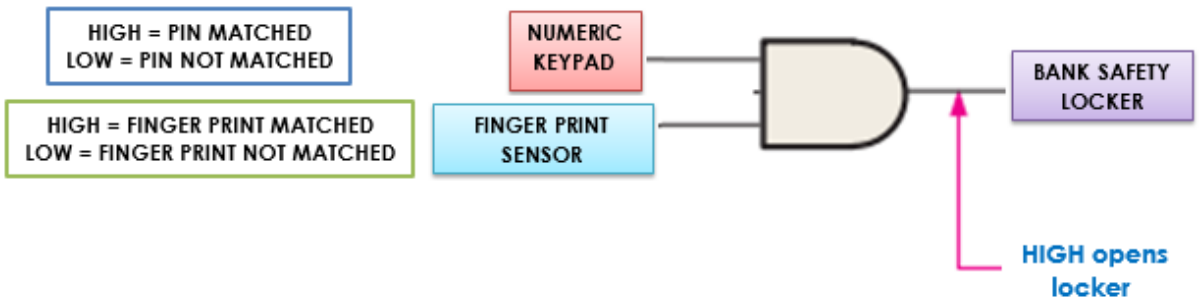


TRUTH TABLE

INPUTS		OUTPUT
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM

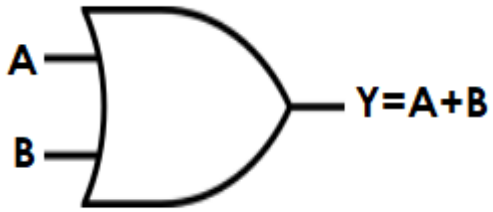




(b). OR GATE:

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation. IC 7432 is the two Input OR gate IC.

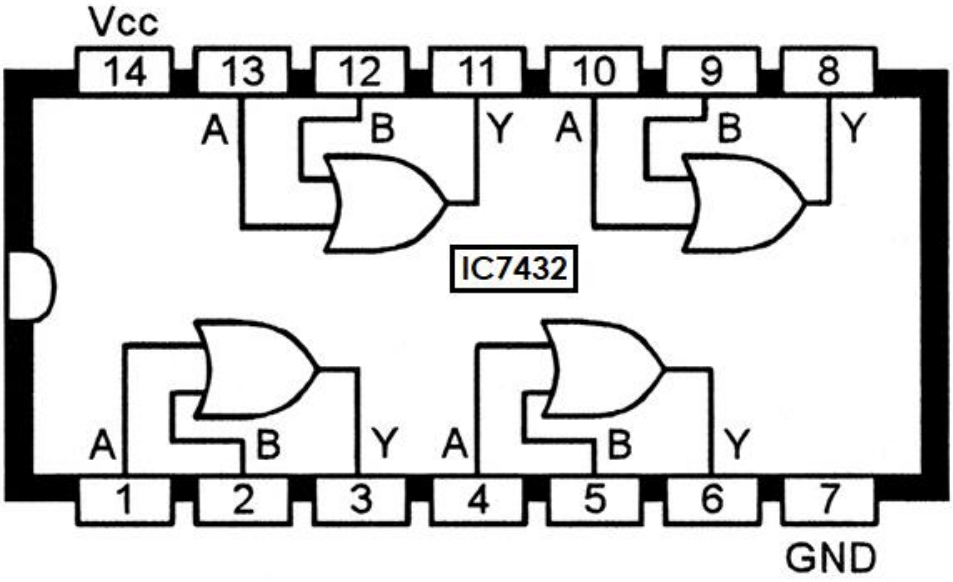
SYMBOL

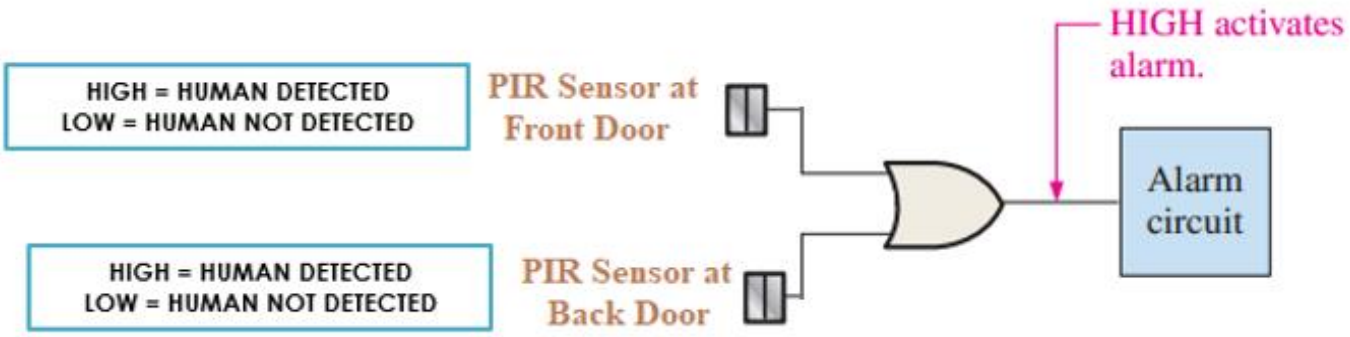


TRUTH TABLE

INPUTS		OUTPUT
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

PIN DIAGRAM

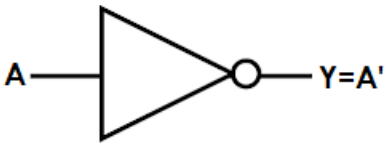




(c). NOT GATE:

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. The output is high when the input is low. The output is low when the input is high. It is also known as an inverter. IC 7404 is a NOT gate IC.

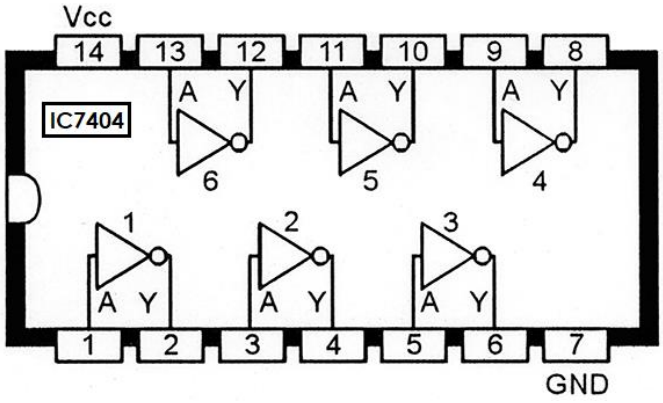
SYMBOL



TRUTH TABLE

INPUT	OUTPUT
A	Y=A'
0	1
1	0

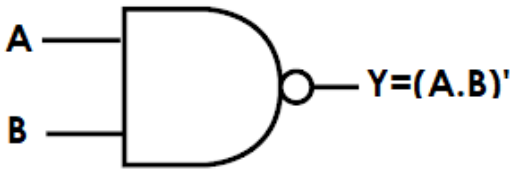
PIN DIAGRAM



(d). NAND GATE:

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The output is low level when both inputs are high. IC7400 is a two input NAND gate.

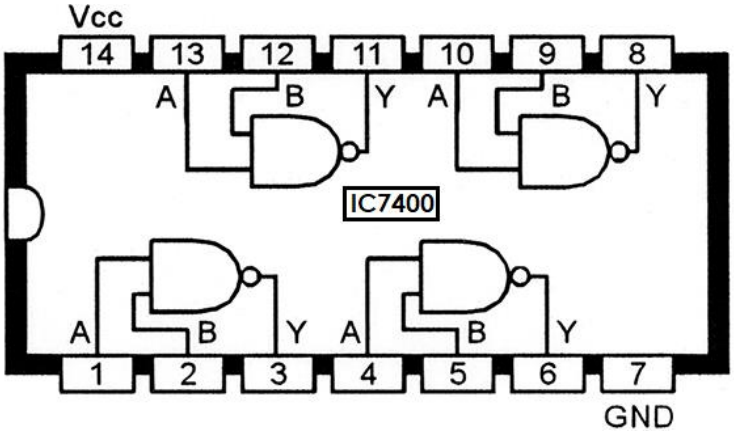
SYMBOL

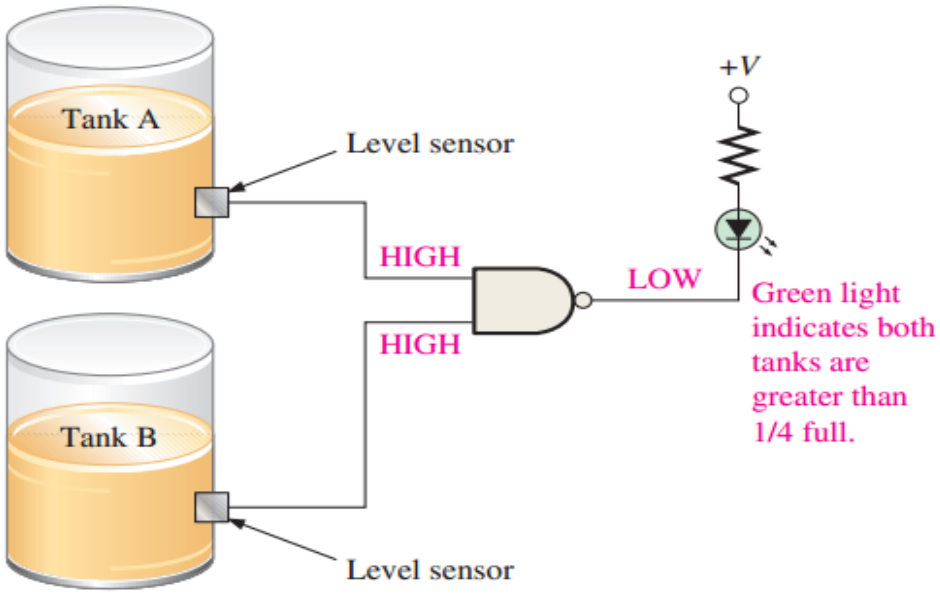


TRUTH TABLE

INPUTS		OUTPUT
A	B	Y=(A.B)'
0	0	1
0	1	1
1	0	1
1	1	0

PIN DIAGRAM

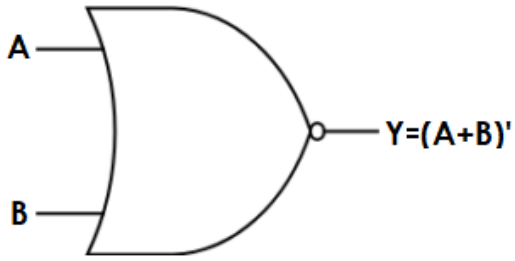




(e). NOR GATE:

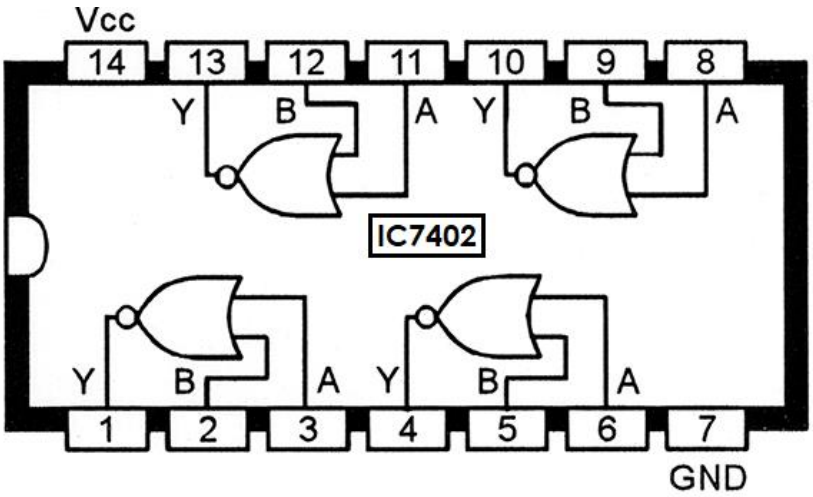
This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The output is high when all inputs are low. IC 7402 is a two input NOR gate.

SYMBOL TRUTH TABLE



INPUTS		OUTPUT
A	B	Y=(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

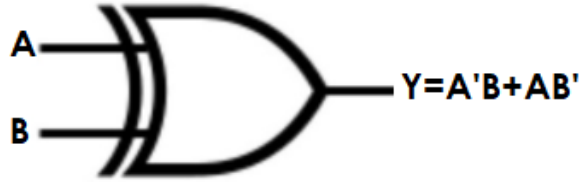
PIN DIAGRAM



(f). XOR GATE:

The 'Exclusive-OR' gate is a circuit which will give the output is high only when odd number of is high. The output is low when both the inputs are low and both the inputs are high. IC 7486 is two input XOR gate.

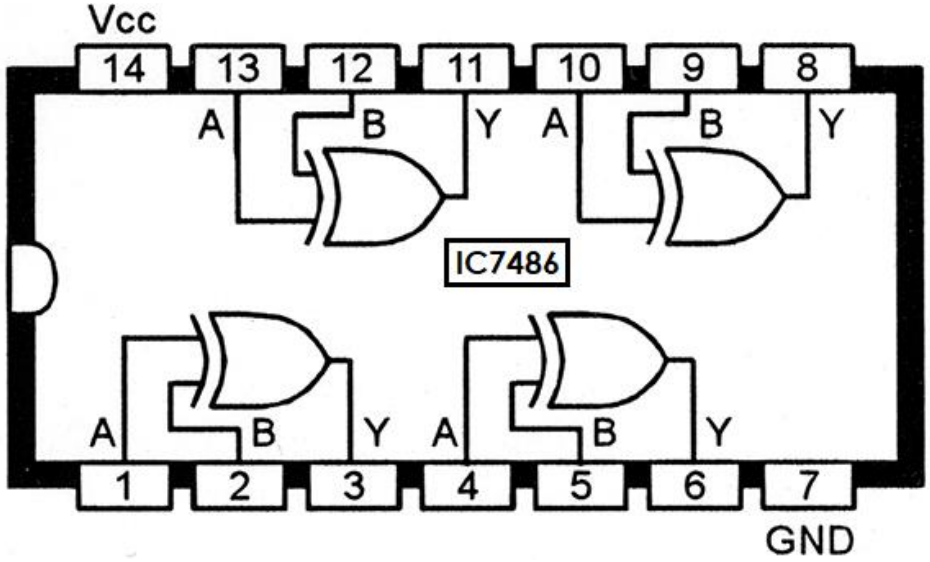
SYMBOL



TRUTH TABLE

INPUTS		OUTPUT
A	B	Y=A'B+AB'
0	0	0
0	1	1
1	0	1
1	1	0

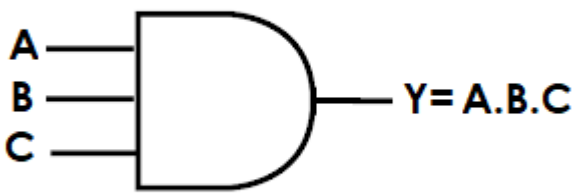
PIN DIAGRAM



(g). 3-INPUT AND GATE:

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. The output is low level when any one of the inputs is low. IC 7411 is a three input AND gate.

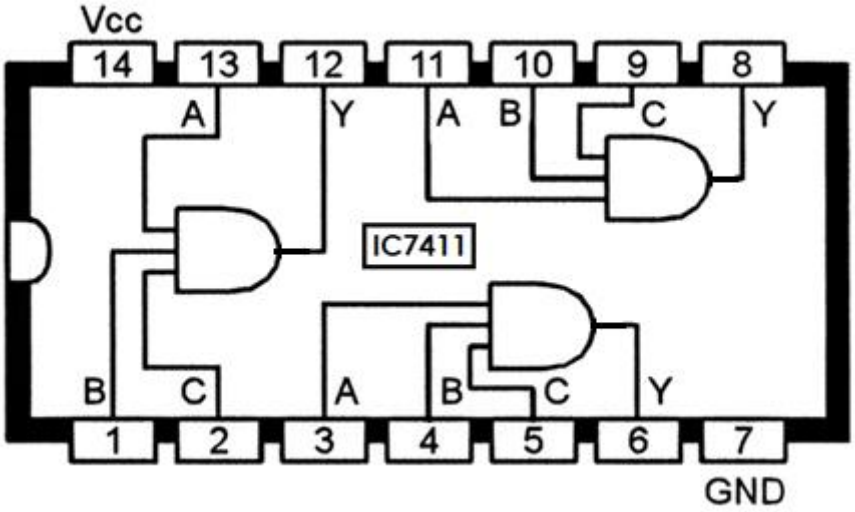
SYMBOL



PIN DIAGRAM

INPUTS			OUTPUT
A	B	C	Y=A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

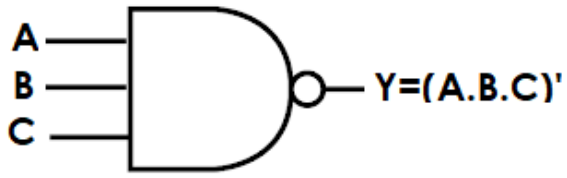
PIN DIAGRAM



(h). 3-INPUT NAND GATE:

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The output is low level when all inputs are high. IC 7410 is a three input NAND gate.

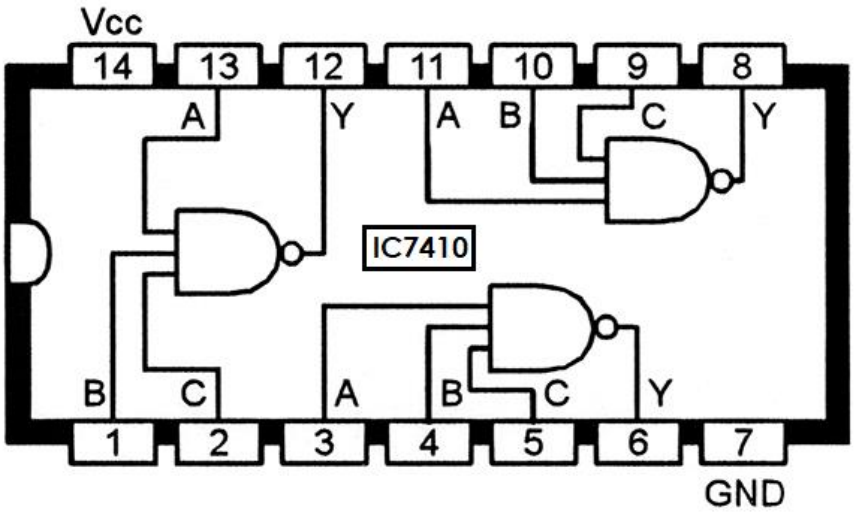
SYMBOL



PIN DIAGRAM

INPUTS			OUTPUT
A	B	C	Y=(A.B.C)'
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

PIN DIAGRAM

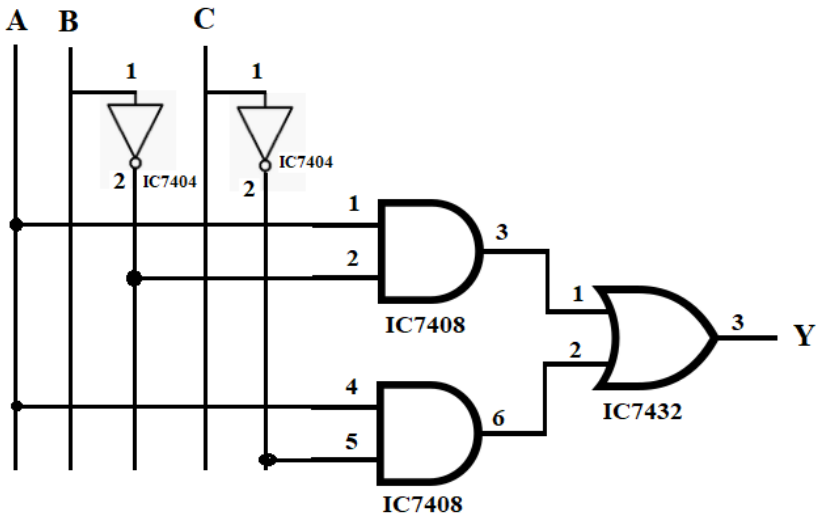


(i). REALIZATION OF SIMPLE BOOLEAN EXPRESSION: Y=A.B'+A.C'

TRUTH TABLE:

INPUTS			INTERMEDIATE OUTPUTS		OUTPUT
A	B	C	AB'	AC'	Y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	0

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table
8. Repeat the process for all other logic gates and logic diagram.

RESULT:

Thus, various basic logic gates and simple Boolean expressions are constructed and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **Why NAND & NOR gates are called universal gates?**
   NAND and NOR gates are called Universal gates because by using which we can design any type of logical expression

2. **Under what conditions the output of three input NOR gate is LOW?**
   If any one of the input is at HIGH state

3. **How many 2-input AND & 2-input OR gates required to realize Y = BD + CE + AB?**
   This expression require 3 2-input AND gate, 2 2-input OR gate required

4. **How many NOT gates are contained in a 7404 NOT IC?**
   Six

5. **How many truth table entries are necessary for a four-input circuit?**
   2^4 = 16 entries

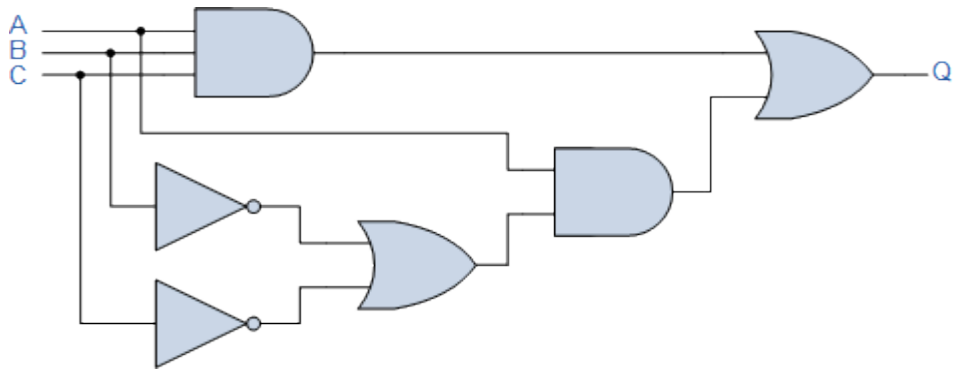
6. **State the logic behind 3-input XOR gate.**
   Give a high output only when odd number of inputs are high

7. **The output will be a LOW for any case when one or more inputs are HIGH in a(n)__**
   NOR gate

8. **What input values will cause 3-input AND logic gate to produce a HIGH output?**
   All inputs are HIGH

9. **How many inputs of 3-input NAND gate must be LOW in order for the output of the logic gate to go HIGH?**
   Atleast one of the input must be LOW

10. **Write a logical expression for the output Q of below logic diagram.**
    Q=ABC+A. (B'+C')



Expt. No. 2	Design and Implementation of Adder & Subtractor

AIM:

(a). Design and construct half adder circuits and verify the truth table using logic gates.

(b). Do design and construct half subtractor circuits and verify the truth table using logic gates.

(c). Design a partial simplified Arithmetic Logic Unit (ALU) using basic logic gates, which is capable of performing the 1-bit arithmetic addition operation with the inclusion of carry as an additional input.

(d). Design a partial simplified Arithmetic Logic Unit (ALU) using basic logic gates, which is capable of performing the 1-bit arithmetic subtraction operation with the inclusion of borrow as an additional input.

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	2-INPUT AND GATE	IC 7408	1
2.	2-INPUT OR GATE	IC 7432	1
3.	1-INPUT NOT GATE	IC 7404	1
4.	2-INPUT XOR GATE	IC 7486	1
5.	DIGITAL TRAINER KIT	-	1
6.	Connecting wires	-	few

(a). HALF ADDER:

THEORY:

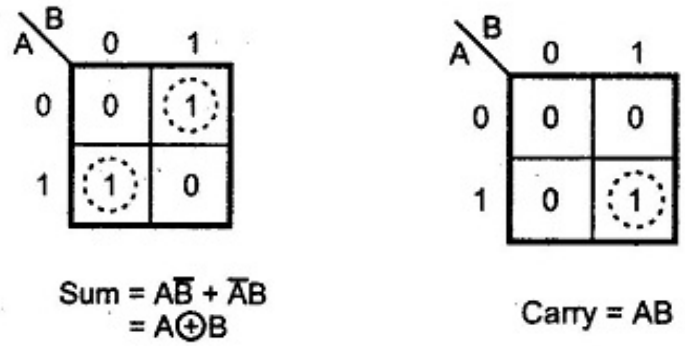
An adder/subtractor is a digital logic circuit in electronics that implements addition/subtraction of numbers. In many computers and other types of processors, adders/Subtractors are used to calculate addresses, similar operations and table indices in the ALU in other parts of the processors. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. Adders are classified into two types: half adder/subtractor and full adder/subtractor. 2-INPUT

Half adder is a combinational arithmetic circuit that adds two one-bit numbers and produces a sum bit (S) and carry bit (C) as the output. If A and B are the input bits, then sum bit (S) is the X-OR of A and B, the carry bit (C) will be the AND of A and B. Half adder is the simplest of all adder circuit, but it has a major disadvantage. The half adder can add only two input bits (A and B) and has nothing to do with the carry if there is any and that's why it is called a half adder.

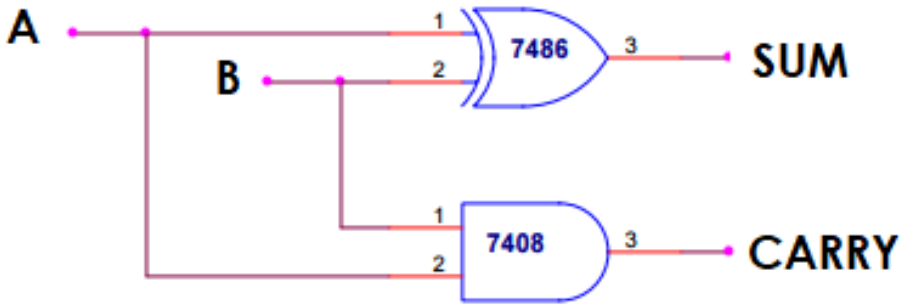
TRUTH TABLE:

INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-MAP:



LOGIC DIAGRAM:



(b). HALF SUBTRACTOR:

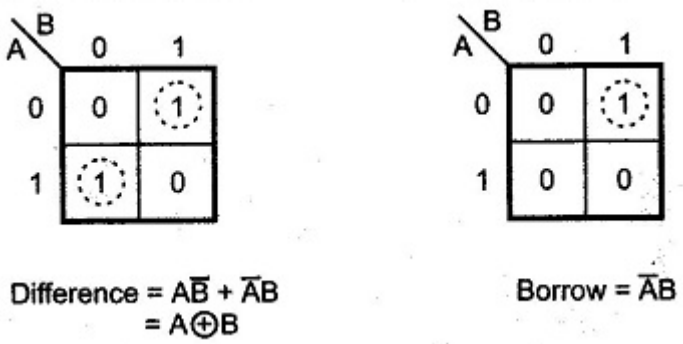
THEORY:

A half-subtractor is mainly used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. As in binary number system, 1 is the largest digit, we only produce borrow when the subtrahend 1 is greater than minuend 0 and due to this, borrow will require.

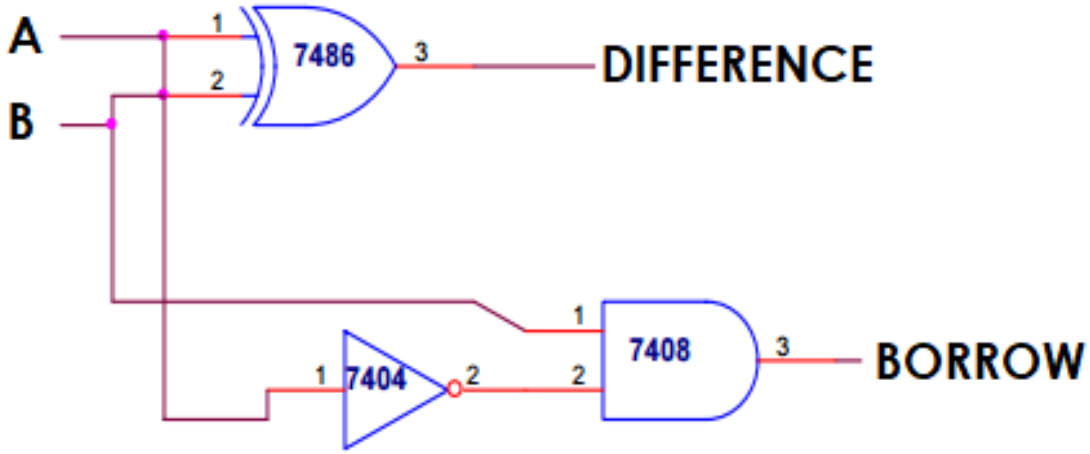
TRUTH TABLE:

INPUTS		OUTPUTS	
A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-MAP:



LOGIC DIAGRAM:



(c). FULL ADDER:

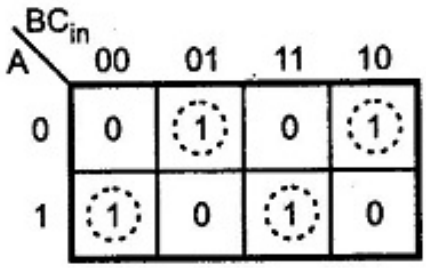
THEORY:

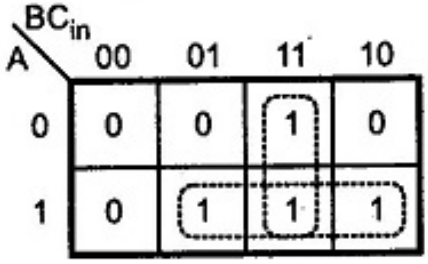
A full adder is a digital circuit that adds three one-bit binary numbers, two operands and a carry bit. The adder outputs two numbers, a sum and a carry bit. Full adders are made from XOR, AND, OR gates in hardware. When a full-adder logic is designed, string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

TRUTH TABLE:

INPUTS			OUTPUTS	
A	B	C_{in}	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-MAP:





SUM = A'B'C_{in} + A'BC_{in}' + ABC_{in}' + ABC_{in} **CARRY = AB + BC_{in} + AC_{in}**

SIMPLIFICATION:

Sum = A'BC'$_{in}$ + AB'C'$_{in}$ + ABC_{in} + A'B'C_{in}

= C'$_{in}$ (A'B+AB') + C_{in} (AB+A'B')

= C'$_{in}$ (A'B+AB') + C_{in} (A'B+AB')' [(x'y+xy')'= (xy+x'y')]

= C_{in} ⊕ (A'B+AB')

S = C_{in} ⊕ (A ⊕ B) [x ⊕ y = x'y+ xy']

Carry = AB+ ACin+ BCin.

= AB+ ACin+ BCin (A +A')

= ABCin+ AB+ ACin+ A'BCin

= AB (Cin+1) + ACin+ A'BCin [Cin+1= 1]

= AB+ ACin+ A'BCin

= AB+ ACin (B+B') + A'BCin

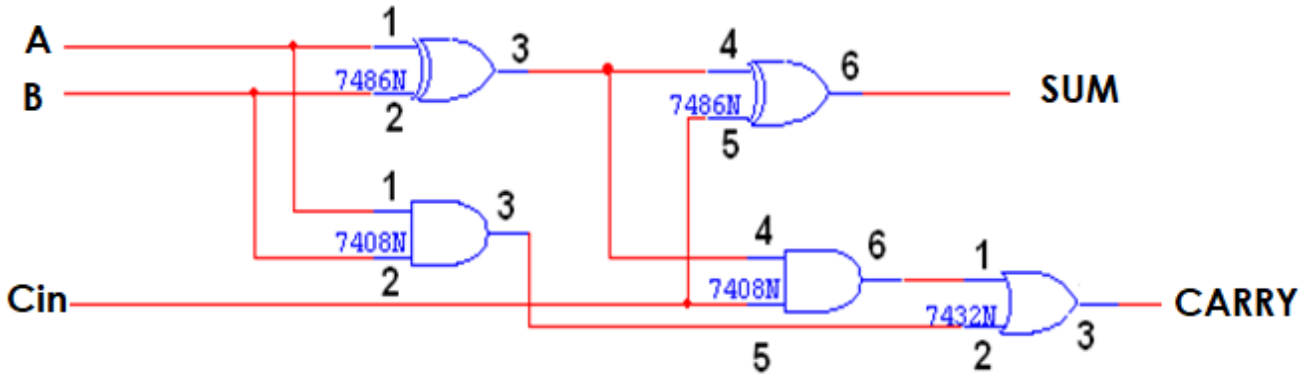
= ABCin+ AB+ A'BCin+ AB'Cin

= AB (Cin+1) + A'BCin+ AB'Cin

= AB+ A'BCin+ AB'Cin [Cin+1= 1]

Carry = AB+ Cin (A'B+AB')

LOGIC DIAGRAM:



(d). FULL SUBTRACTOR:

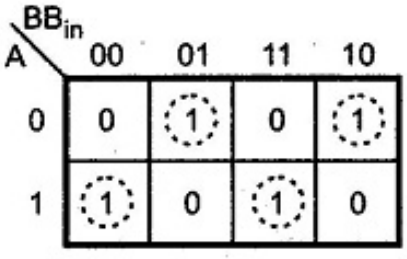
THEORY:

Full subtractor is an electronic device or logic circuit, which performs subtraction of two binary digits. The foremost disadvantage of the half subtractor is, we cannot make a Borrow bit in this subtractor. Whereas in full subtractor design, actually we can make a Borrow bit in the circuit & can subtract with remaining two inputs. Here A is minuend, B is subtrahend & B_{in} is borrow in. There are two outputs that are DIFFERENCE and BORROW output. The BORROW output indicates that the minuend bit requires borrow '1' from the next minuend bit. .

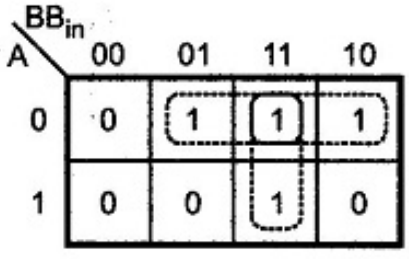
TRUTH TABLE:

INPUTS			OUTPUTS	
A	**B**	**B_{in}**	**DIFFERNECE**	**BORROW**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-MAP:



DIFFERENCE = $A'B'B_{in}+A'BB_{in}'+AB'B_{in}'+ABB_{in}$



BORROW = $A'B+BB_{in}+A'B_{in}$

SIMPLIFICATION:

Difference, D = $A'BB'_{in} + AB'B'_{in} + ABB_{in} + A'B'B_{in}$.

$= B'_{in}(A'B+AB') + B_{in}(AB+A'B')$

$= B'_{in}(A'B+AB') + B_{in}(A'B+AB')'$ $[(x'y+xy')'= (xy+x'y')]$

$= B_{in} \oplus (A'B+AB')$

$D = B_{in} \oplus (A \oplus B)$ $[x \oplus y = x'y+ xy']$

Borrow, Bout = A'B+ BBin+ A'Bin.

= A'B+ BBin+ A'Bin (B +B')

= A'BBin+ A'B+ BBin+ A'B'Bin

= A'B (Bin+1) + BBin+ A'B'Bin [Cin+1= 1]

= A'B+ BBin+ A'B'Bin

= A'B+ BBin (A+A') + A'B'Bin [A+A'= 1]

= A'BBin+ A'B+ ABBin+ A'B'Bin

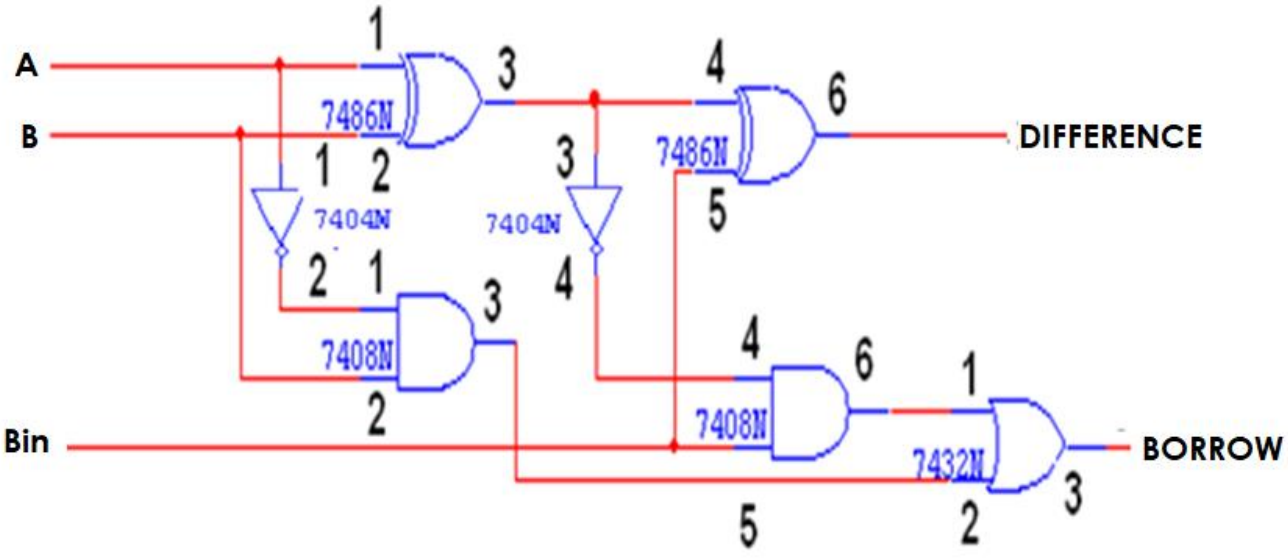
= A'B (Bin+1) + ABBin+ A'B'Bin [Cin+1= 1]

= A'B+ ABBin+ A'B'Bin

= A'B+ Bin (AB+A'B')

Bout = A'B+ Bin (A'B+AB')' $[(x'y+xy')'= (xy+x'y')]$

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus, various data path elements such as half adder, half subtractor, full adder and full subtractor logic circuits are constructed using basic logic gates and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **What is combinational circuit?**
   Digital circuits whose output depends upon the inputs present at that current instant of time

2. **State the purpose of K-Map.**
   It is a method of simplifying Boolean Functions in a systematic mathematical way

3. **What is the main limitation of half-adder?**
   They perform addition of two one-bits, cannot accept a carry bit from a previous stage

4. **State the output of DIFFERENCE and BORROW if full subtractor accepts input A=0, B=1 and B_{in}=1.**
   DIFFERENCE=0, BORROW=1

5. **How many AND, OR and EXOR gates are required for the configuration of full adder?**
   2-AND, 1-OR and 2-XOR

6. **Let A, B and C is the input of a subtractor then the difference output will be____.**
   A XOR B XOR C

7. **Let the input of a half subtractor is A and B then what the difference output will be if A = B?**
   0

8. **The full subtractor can be implemented using _____.**
   2-AND, 1-OR and 2-XOR, 1-NOT

9. **Full adder is used to perform subtraction of _____.**
   Three one-bit numbers

10. **List out atleast three applications of Full adder.**
    - ALU-Arithmetic Logic Unit (one of the circuit is a full adder).
    - To generate memory addresses inside a computer
    - To make the Program Counter point to next instruction as Increment and decrement operators

Expt. No. 3	Design and Implementation of 4:1 Mux & 4:2 Decoder

AIM:

(a) In a communication system, multiple signals are transmitted on a single transmission line. Assume audio, image, data and video are four signal to be transmitted on a single transmission line by using appropriate signal selection line. Implement a digital circuit using basic logic gates to meet this requirement.

(b) Control & timing unit of a Microprocessor has a decoding unit to decode the program instructions in order to activate the specific control lines such that different operations in the ALU are carried out as shown below. Assume program instructions are represented in 2-bit format.

Program Instruction		Control signal
0	0	Activate Add operation in ALU
0	1	Activate Subtract operation in ALU
1	0	Activate compare operation in ALU
1	1	Activate logical operation in ALU

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	3-INPUT AND GATE	IC 7411	2
2.	2-INPUT OR GATE	IC 7432	1
3.	1-INPUT NOT GATE	IC 7404	1
4.	DIGITAL TRAINER KIT	-	1
5.	Connecting wires	-	few

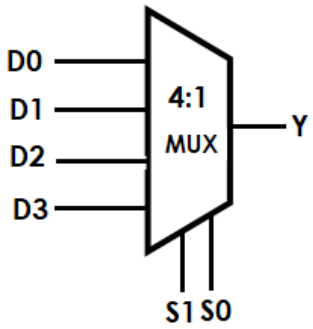
(a). 4:1 MULTIPLEXER:

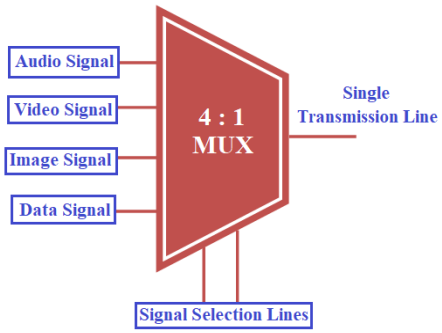
THEORY:

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line, the device for such purpose is called a Multiplexer. A digital multiplexer (MUX) 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. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output. MUX, can be digital circuits made from high-speed logic gates used to switch digital or binary data.

Normally there are 2^n input line and n selection lines whose bit combination determine which input is selected at the output side. A multiplexer can be designed with various inputs according to our needs. In 4:1 MUX, there will be 4 input lines and 1 output line. In addition, to control which input should be selected out of these 4, we need 2 selection lines. The combination of binary numbers given as a selection line will determine the output of the MUX.

BLOCK DIAGRAM:



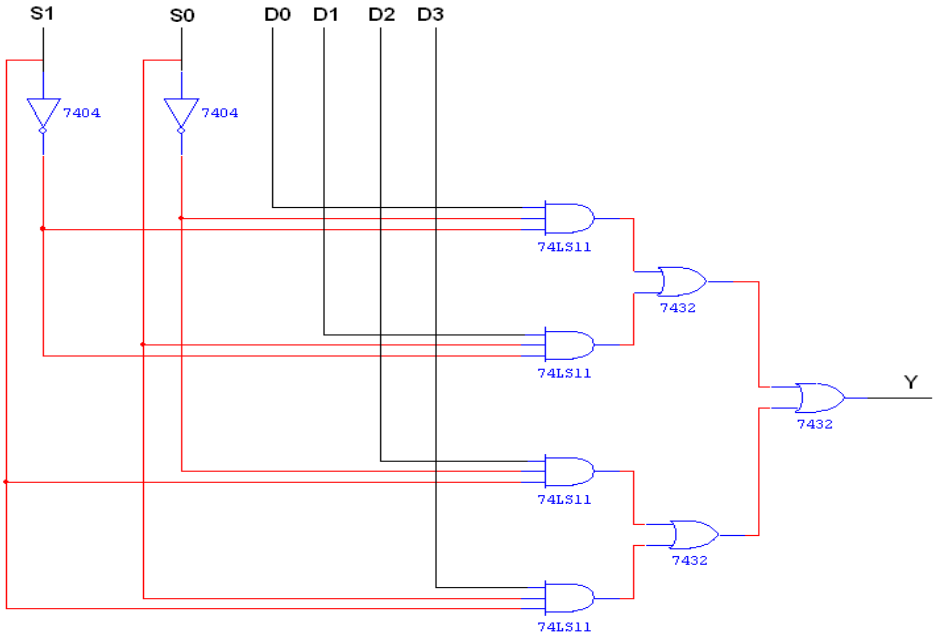


TRUTH TABLE:

SELECTION LINES		DATA INPUTS				OUTPUT
S1	S0	D3	D2	D1	D0	Y
0	0	X	X	X	0	0
0	0	X	X	X	1	1
0	1	X	X	0	X	0
0	1	X	X	1	X	1
1	0	X	0	X	X	0
1	0	X	1	X	X	1
1	1	0	X	X	X	0
1	1	1	X	X	X	1

Y = D0 S1' S0' + D1 S1' S0 + D2 S1 S0' + D3 S1 S0

LOGIC DIAGRAM:



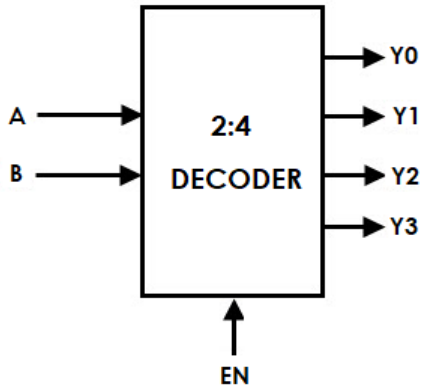
(b). 2:4 DECODER:

THEORY:

Decoder is type of combinational Circuit, which decodes a small bit value into large bit value. It is normally used in combination with encoders, which does exactly the opposite of what a decoder does. A binary decoder is a multi-input, multi-output combinational circuit that converts a binary code of n input lines into a one out of 2^n output code. A decoder can be used for obtaining the parallel data from the serial data received.

In 2:4 decoder, 2 binary inputs are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. The logic circuit design of the 2:4 decoder can be implemented by using NOT and AND gates. A common enable line is connected to each AND gate such that when EN= 0 all the outputs are zero and if EN=1, depends on the inputs A and B, outputs are produced. Each output represents one of the minterms of the 2 input variables.

BLOCK DIGRAM:

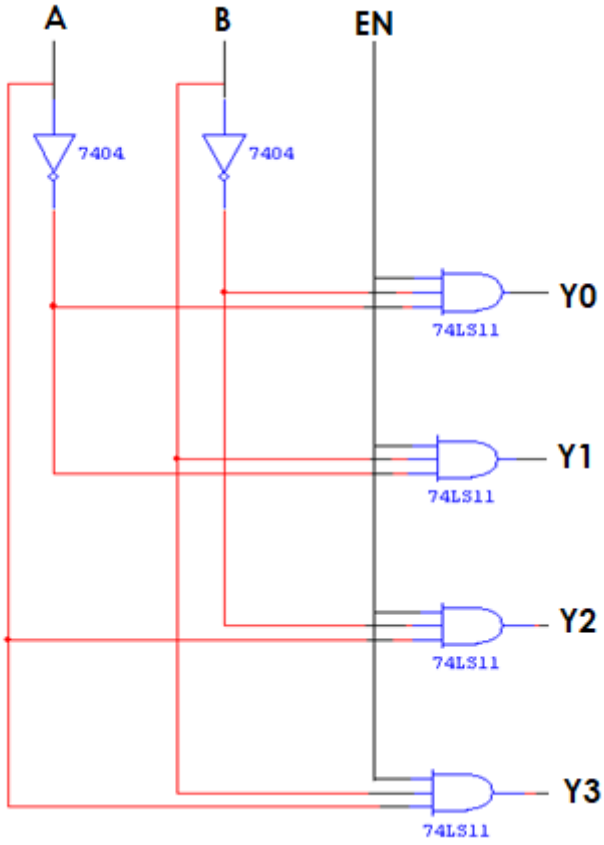


TRUTH TABLE:

INPUTS			OUTPUTS			
E	**A**	**B**	**Y3**	**Y2**	**Y1**	**Y0**
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Y0=EN.A'.B' **Y1=EN.A'.B** **Y2=EN.A.B'** **Y3=EN.A.B**

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus, basic combinational logic circuits 4:1 Multiplexer and 2:4 Decoder are constructed using basic logic gates and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **Why MUX is called as "Data Selector"?**
   This selects one out of many inputs

2. **How many number of selection lines are required to construct 32:1 multiplexer logic?**
   "5" selection lines

3. **What is the function of an enable input on a decoder circuit?**
   Enable input is used to activate the decoder function, when enable is high the circuit works as a decoder. When enable is low, the circuit does not work.

4. **What is the number of output lines for a decoder with four input lines?**
   "16" Output lines

5. **In a multiplexer, the selection of a particular input line is controlled by _____**
   Selection lines

6. **How many NOT gates are required for the construction of an 8-to-1 multiplexer?**
   3

7. **In the given 4-to-1 multiplexer, if selection line S1S0=01 and input D3D2D1D0=1010 then, the output Y is _____.**
   1

8. **What is the use of Enable input E in decoder circuit?**
   To enable and disable the decoder circuit

9. **List atleast three application of Multiplexer.**
   - MUX Applications: parallel to serial converter, data selector in communication system, address and data lines in memory devices.

10. **List atleast three application of Decoder.**
    - Decoder applications: Data transfer in communication system, address decoding and instruction decoding in Microprocessor, Code converters

Expt. No. 4	Design and Implementation of Magnitude Comparator, Parity Generator & Checker

AIM:

(a). Design a control unit for a sea salt packing machine to pack a salt bag of 2kg each. Current weight of the bag is measured and represented in 2-bit binary number then compared with the reference value. If the salt bag is less/greater than 2kg control signal generated to add/remove excess salt to/from salt bag. If the salt bag weight is exactly 2kg then control signal generated to make a pack. Design and Implement an appropriate digital circuit using basic logic gates to meet this requirement. (Hint: 2-Bit Magnitude comparator)

(b). Bluetooth module is interfaced with Arduino Uno board and transmitting its 4-bit data continuously to the receiver section. This 4-bit data comprises of 3-bit temperature sensor data and 1-bit for parity (even). Since this system operating under a noisy environment, there is a possibility of error prone into the actual values of transmitting data. Design a suitable combinational logic circuit in the transmitter and receiver section to detect the data is valid or not. (Hint: Parity or and Checker)

COMPONENTS REQUIRED:

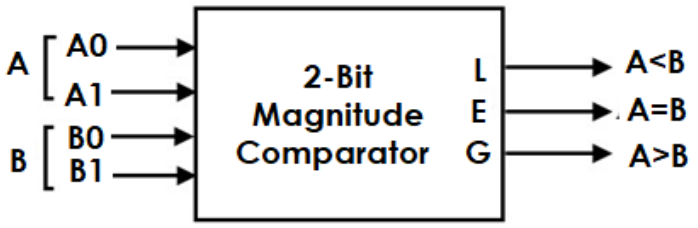
S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	2-INPUT AND GATE	IC 7408	2
2.	2-INPUT OR GATE	IC 7432	1
3.	1-INPUT NOT GATE	IC 7404	1
4.	2-INPUT XOR GATE	IC7486	1
5.	DIGITAL TRAINER KIT	-	1
6.	Connecting wires	-	few

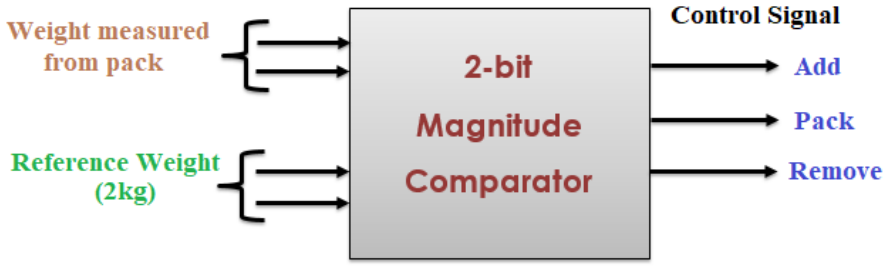
(a). 2-BIT MAGNITUDE COMPARATOR:

THEORY:

Data comparison is needed in digital systems while performing arithmetic or logical operations. The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result of the comparison. A Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B. A 2-bit comparator compares two binary numbers, each of two bits and produces their relation such as one number is equal or greater than or less than the other. The first number A is designated as A = A1A0 and the second number is designated as B = B1B0. This comparator produces three outputs as G (G = 1 if A>B), E (E = 1, if A = B) and L (L = 1 if A<B).

BLOCK DIAGRAM:

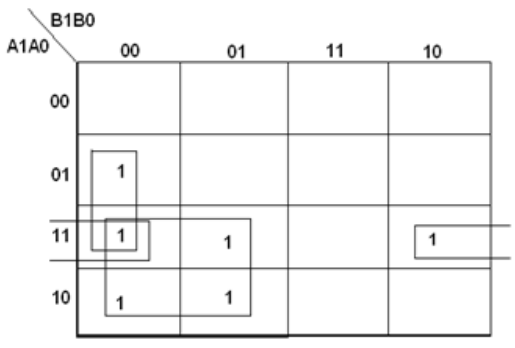




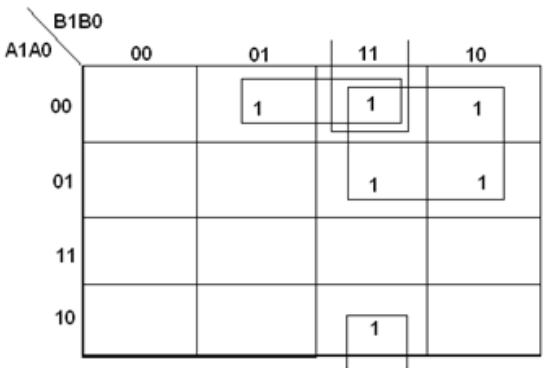
TRUTH TABLE:

INPUTS				OUTPUT		
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

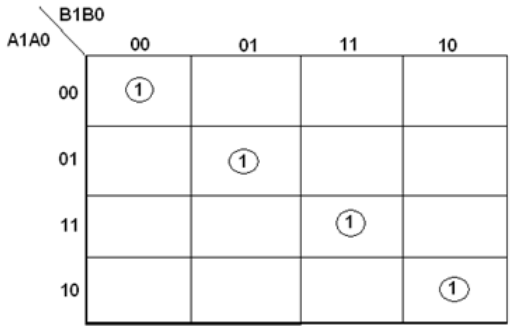
K-MAP:



A>B = A1B1' + A0B1'B0' + A1A0B0'



A<B = A1'B1 + A0'B1B0 + A1'A0'B0



A=B = A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'

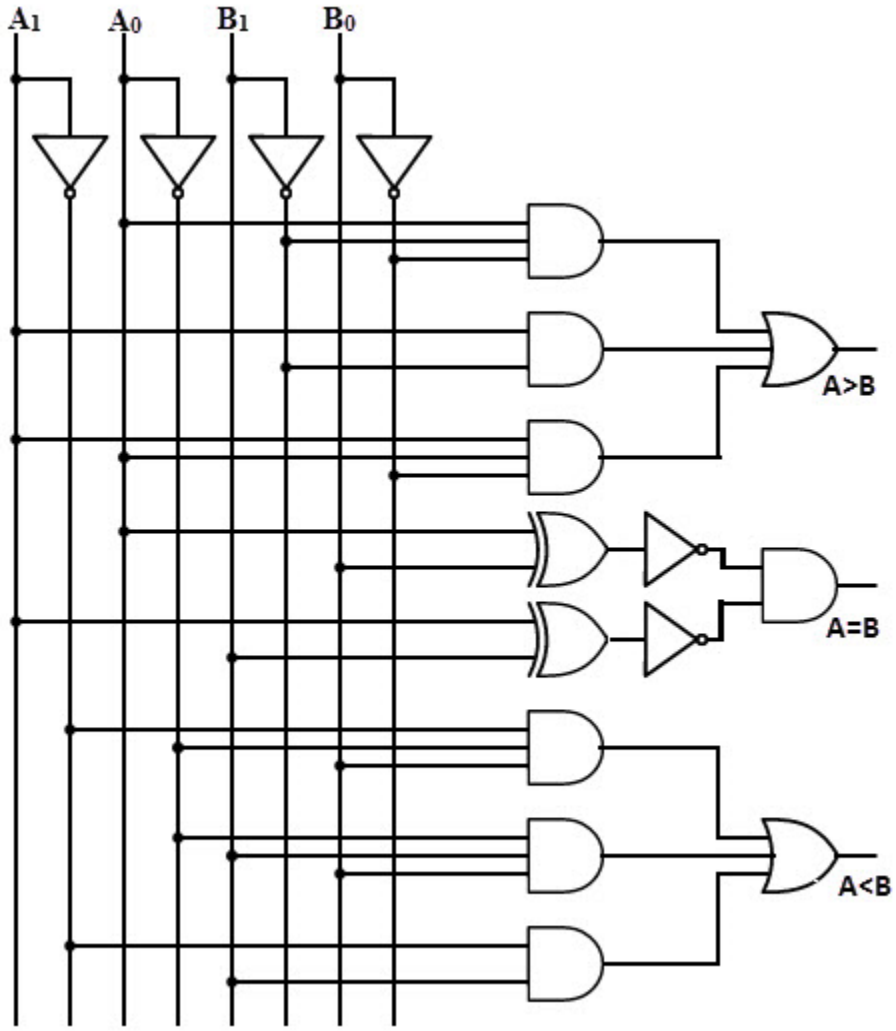
= A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')

= (A0B0 + A0'B0') (A1B1 + A1'B1')

= (A0 Θ B0) (A1 Θ B1)

A=B = (A0 ⊕ B0)' (A1 ⊕ B1)'

LOGIC DIAGRAM:



(b). PARITY (EVEN) GENERATOR & CHECKER:

THEORY:

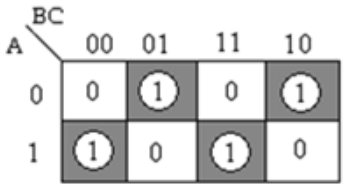
A Parity is a very useful tool in information processing in digital computers to indicate any presence of error in bit information. External noise and loss of signal strength causes loss of data bit information while transporting data from one device to other device, located inside the computer or externally. To indicate any occurrence of error, an extra bit is included with the message according to the total number of 1s in a set of data, which is called parity. If the extra bit is considered 0 if the total number of 1s is even and 1 for odd quantities of 1s in a set of data, then it is called even parity. On the other hand, if the extra bit is 1 for even quantities of 1s and 0 for an odd number of 1s, then it is called odd parity. The message including the parity is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker. The message bits with the parity bit are transmitted to their destination, where they are applied to a parity checker circuit. The parity checker circuit produces a check bit and is very similar to the parity generator circuit. If the check bit is 1, then it is assumed that the received data is incorrect. The check bit will be 0 if the received data is correct.

3-BIT EVEN PARITY GENERATOR:

TRUTH TABLE:

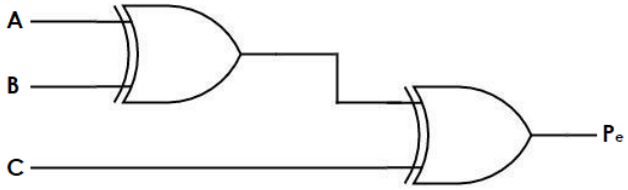
3-BIT MESSAGE INPUT			EVEN PARITY BIT
A	B	C	P_e
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

K-MAP:



$$P_e = A'B'C + A'BC' + A'B'C' + ABC$$
$$= A'(B'C + BC') + A(B'C' + BC)$$
$$= A'(B \oplus C) + A(B \oplus C)'$$
$$P_e = (A \oplus B \oplus C)$$

LOGIC DIAGRAM:

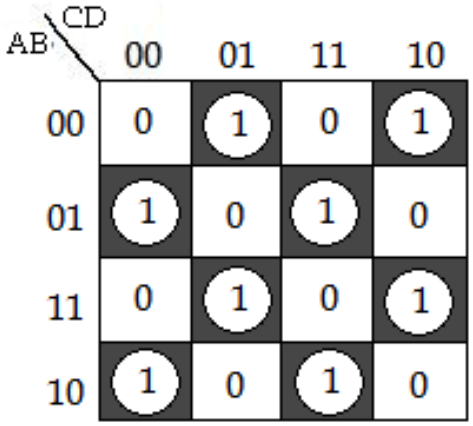


4-BIT EVEN PARITY CHECKER:

TRUTH TABLE:

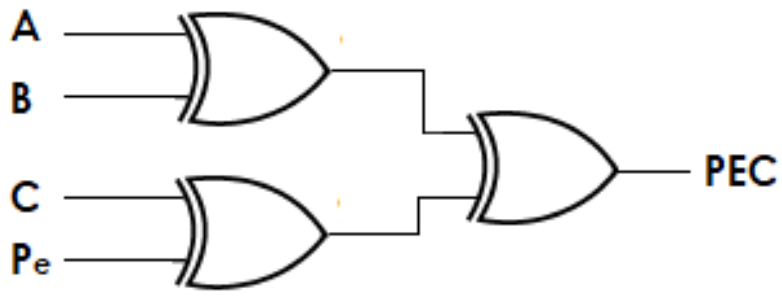
4-BIT RECEIVED MESSAGE				PARITY ERROR CHECK (PEC)
A	B	C	P_e	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

K-MAP:



PEC = A'B' (C'D+ CD') + A'B (C'D'+ CD) + AB (C'D+ CD') + AB' (C'D'+ CD)
= A'B' (C ⊕ D) + A'B (C ⊕ D)' + AB (C ⊕ D) + AB' (C ⊕ D)'
= (A'B'+ AB) (C ⊕ D) + (A'B+ AB') (C ⊕ D)'
= (A ⊕ B)' (C ⊕ D) + (A ⊕ B) (C ⊕ D)'
PEC = (A ⊕ B) ⊕ (C ⊕ D)

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus combinational logic circuits such as 2-bit magnitude comparator, 3-bit even parity checker and 4-bit even parity checker are constructed using basic logic gates and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **What is the difference between identity comparator and magnitude comparator?**
   Identity Comparator is a digital comparator with only one output terminal for when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW). Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B.

2. **Magnitude comparator always produces only ____ output (s) as HIGH among its possible ____ number of outputs.**
   1, 3

3. **Number of inputs required for magnitude comparator is?**
   Two inputs are required to perform comparison operation by magnitude comparator

4. **Which combination of logic gates are ideal realizing "equal" condition in 2-bit magnitude comparator?**
   XNOR, AND

5. **State atleast 2 application of Magnitude comparator**
   - In CPU and microcontrollers
   - Used in password verification and biometric applications

6. **What is a parity bit?**
   A simple form of error detection is achieved by adding an extra bit to the transmitted word. The additional bit is known as parity bits.

7. **The even parity output of binary number (1101010111) is?**
   1

8. **Which logic gate is most appropriate to modify the even parity generator logic circuit into odd parity generator logic circuit?**
   Adding NOT gate at the output end of even parity generator

9. **State main drawback of parity checker based error detection method.**
   Parity checker is capable of identifying only one bit change. If there is more than one bit change on the message signal, then it can't be detected by parity checker

10. **List atleast three application of Parity Checker / Generator.**
    - SCSI and PCI buses use parity to detect transmission errors
    - many microprocessor instruction caches include parity protection
    - In serial communication contexts, parity is usually generated and checked by interface hardware (e.g., a UART)

Expt. No. 5	Design and Implementation SR, JK, T & D Flip-Flops

AIM:

(a). Construct SR Flip-Flops using basic logic gates and verify its logic using truth table.

(b). Construct JK Flip-Flops using basic logic gates and verify its logic using truth table.

(c). In a Microprocessor design, signals that flow through different paths arrive at different time. This could cause many problems when these signals have to interact with each other. Identify and implement a suitable flop-flop required to synchronize these signals using basic logic gates. (Hint: D-FF)

(d). Counters are the digital circuits, which are used to count the number of events. Identify and implement the most suitable Flip-flip required to design a counter unit using basic logic gates. (Hint: T-FF)

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	2-INPUT NAND GATE	IC 7400	1
2.	1-INPUT NOT GATE	IC 7404	1
3.	DIGITAL TRAINER KIT	-	1
4.	Connecting wires	-	few

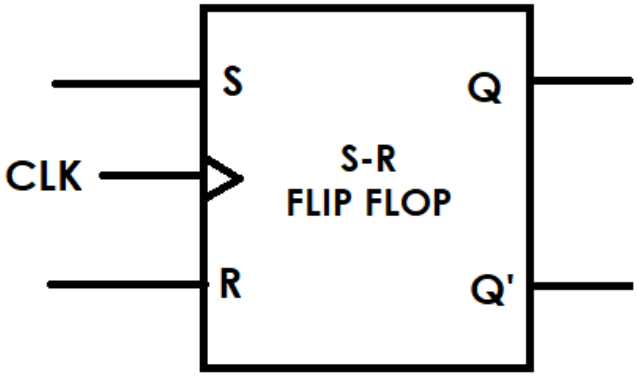
(a). SR (SET RESET) FLIP-FLOP:

THEORY:

The basic 1-bit digital memory circuit is known as a flip-flop. Flip-Flops are synchronous bistable devices (has two outputs Q and Q'). In this case, the term synchronous means that the output changes state only at a specified point on the triggering input called the clock (CLK). It can have only two states, either the 1 state or the 0 state. Flip-flops can be constructed by using NAND or NOR gates.

The SR flip-flop, stands for "Set-Reset" flip-flop. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will "SET" the device (meaning the output = "1"), and is labelled S and one which will "RESET" the device (meaning the output = "0"), labelled R. The reset input resets the flip-flop back to its original state with an output Q. A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it's current state or history. When S and R at HIGH state both outputs tries to get into HIGH state and not of them get into state output state. This state is called intermediate or invalid state.

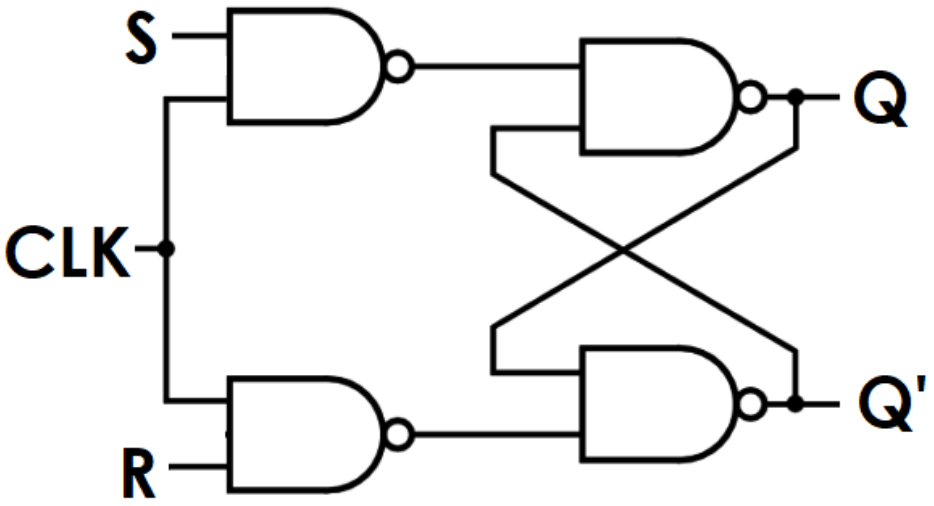
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUTS		PRESENT STATE	NEXT STATE	STATE
	S	R	Q_n	Q_{n+1}	
↑	0 0	0 0	0 1	0 1	NO CHANGE (NC)
↑	0 0	1 1	0 1	0 0	RESET (R)
↑	1 1	0 0	0 1	1 1	SET (S)
↑	1 1	1 1	0 1	x x	INDETERMINATE (*)
↓	x x	x x	0 1	0 1	NO CHANGE (NC)

LOGIC DIAGRAM:

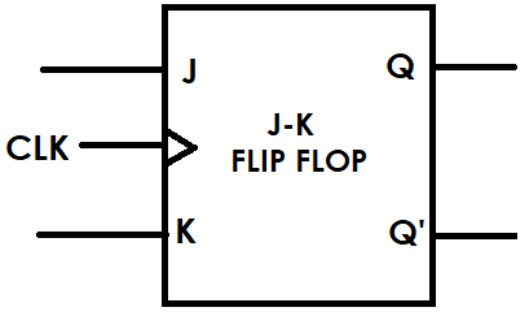


(b). JK (JACK KILBY) FLIP-FLOP:

THEORY:

The JK Flip Flop is the most widely used flip flop. It is considered to be a universal flip-flop circuit. The sequential operation of the JK Flip Flop is same as for the RS flip-flop with the same SET and RESET input. The difference is that the JK Flip Flop does not the invalid input states when S and R are both 1. A JK Flip-Flop can be obtained from the clocked SR Flip-Flop by augmenting two AND gates. If the circuit is in the "SET" condition, the J input is inhibited by the status 0 of Q through the lower NAND gate. Similarly, the input K is inhibited by 0 status of Q through the upper NAND gate in the "RESET" condition. When both J and K are at logic "1", the JK Flip Flop toggle.

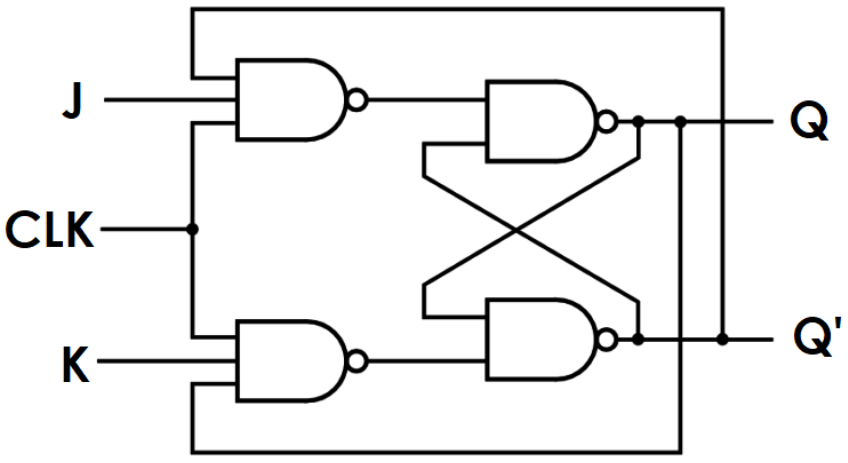
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUTS		PRESENT STATE	NEXT STATE	STATE
	J	K	Q_n	Q_{n+1}	
↑	0 0	0 0	0 1	0 1	NO CHANGE (NC)
↑	0 0	1 1	0 1	0 0	RESET (R)
↑	1 1	0 0	0 1	1 1	SET (S)
↑	1 1	1 1	0 1	1 0	TOGGLE (T)
↓	x x	x x	0 1	0 1	NO CHANGE (NC)

LOGIC DIAGRAM:

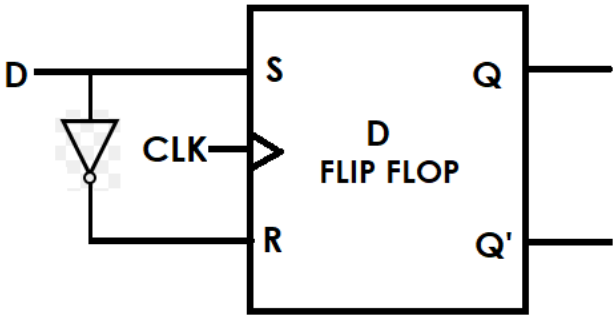


(c). D (DATA/DELAY) FLIP-FLOP:

THEORY:

This flip-flop, called a Data flip-flop because of its ability to 'latch' and remember data, or a Delay flip-flop because latching and remembering data can be used to create a delay in the progress of that data through a circuit. A D flip–flop is constructed by modifying an SR flip – flop. The S input is given with D input and the R input is given with inverted D input. Hence a D flip – flop is similar to SR flip – flop in which the two inputs are complement to each other, so there will be no chance of any intermediate state occurs. The major drawback of SR flip – flop is the race around condition which in D flip – flop is eliminated (because of the inverted inputs). When we don't apply any clock input to the D flip flop or during the falling edge of the clock signal, there will be no change in the output. It will retain its previous value at the output Q. If the clock signal is high and if D input is high, then the output is also high and if D input is low, then the output will become low. Hence the output Q follows the input D in the presence of clock signal.

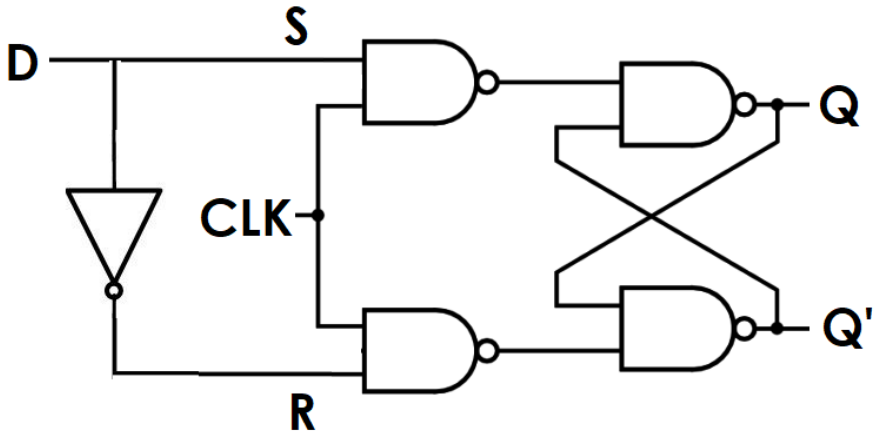
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUT	PRESENT STATE	NEXT STATE	STATE
	D	Q_n	Q_{n+1}	
↑	0	0	0	RESET
↑	0	1	0	
↑	1	0	1	SET
↑	1	1	1	
↓	X	Q_n	Q_n	NO CHANGE

LOGIC DIAGRAM:

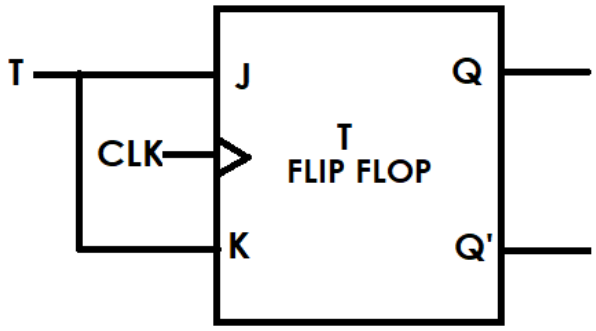


(d). T (TOGGLE) FLIP-FLOP:

THEORY:

T flip – flop is also known as "Toggle Flip – flop". To avoid the occurrence of intermediate state in SR flip – flop, we should provide only one input to the flip – flop called Trigger input or Toggle input (T). Then the flip – flop acts as a Toggle switch. Toggling means 'Changing the next state output to complement of the present state output'. The T (Toggle) Flip-Flop is a modification of the JK Flip-Flop. It is obtained from JK Flip-Flop by connecting both inputs J and K together, i.e., single input. Regardless of the present state, the Flip-Flop complements its output when the clock pulse occurs while input T= 1.

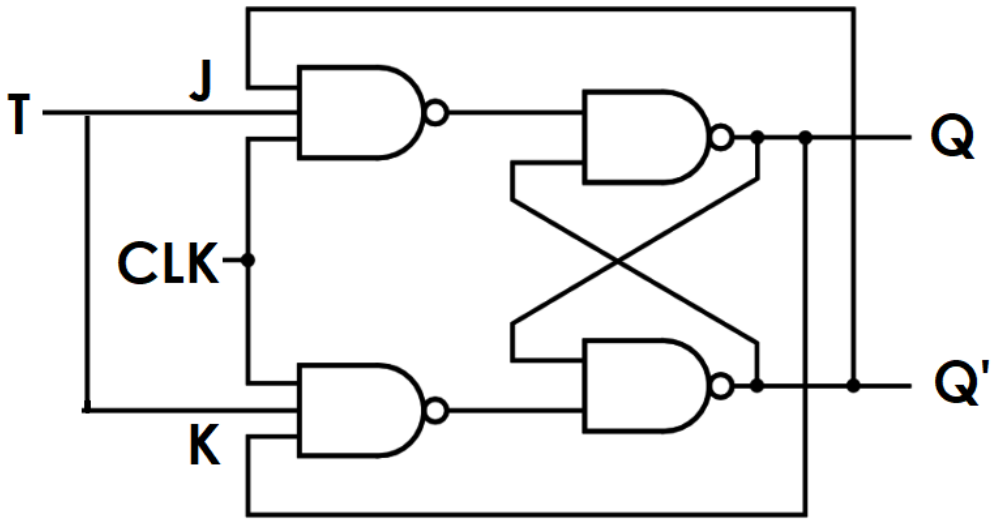
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUT	PRESENT STATE	NEXT STATE	STATE
	T	Q_n	Q_{n+1}	
↑	0	0	0	NO CHANGE
↑	0	1	1	
↑	1	0	1	TOGGLE
↑	1	1	0	
↓	X	Q_n	Q_n	NO CHANGE

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus, basic flip-flops such as SR, JK, D and T are constructed using basic logic gates and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **What is sequential circuits?**
   The logic circuits whose outputs at any instant of time depends only on the present input but also on the past outputs are called Sequential circuits.

2. **A basic S-R flip-flop can be constructed by cross coupling of which basic logic gates?**
   NAND and NOR

3. **What is the main drawback of a S-R flip-flop?**
   The main drawback of s-r flip flop is invalid output when both the inputs are high, which is referred to as Invalid or intermediate State.

4. **Which flip-flop is called universal flip?**
   JK Flip flop. It can be used to realize other Flip-flops such as SR, D and T.

5. **When and which is a flip-flop said to be transparent?**
   D Flip flop is called transparent flip-flip since Q output follows the input.

6. **On a positive edge-triggered flip-flop circuit, the outputs reflect the input condition only when?**
   The clock pulse transitions from LOW to HIGH.

7. **Which type of circuit is faster, combinational or sequential? Why?**
   Combinational circuits are often faster than sequential circuits. Since, the combinational circuits do not require memory elements whereas the sequential circuits need memory devices to perform their operations in sequence.

8. **In T-flip flop toggle operation is performed only when T=?**
   1

9. **How T-flip flop is constructed using JK Flip-flop?**
   By shorting J and K input as a single input T.

10. **List atleast 3 application of flip-flops.**
    Data storage device
    To design counters
    Used in frequency divider circuits
    Used as registers in Microprocessors/ microcontrollers
    For data transfer applications using shift registers

Expt. No. 6	Design and Implementation of 4-Bit Shift Registers (SISO, SIPO, PIPO)

AIM:

(a). In modern computer system, it is essential to operating the computing unit in optimized manner. Especially, shift registers are extensively used to perform optimized binary multiplication and division. This is due to the fact that the shift of data bit by one position towards right causes the number to be divided by 2 while the left-shift of the data bit by one place in the shift register multiplies the number by 2. For example, consider a 4-bit shift register with the content 0110, which is equal to 6 in decimal. If the number shifts left by one-bit, then one gets 1100, which is 12 (= 6 × 2) in decimal. Similarly is the number shifts towards right by one bit, then the register contents will become 0011, which is nothing but 3 (=6/2) in decimal. Design and implement the suitable 4-bit shift register logic circuit for the given requirement using appropriate flip flop.(Hint: SISO)

(b). In general, many microprocessors/microcontrollers handle data as bytes (8 bits) or words (16 bit, 32 bit) as a preferred format. However, many external interfacing device prefer to operate on serial format instead of parallel. Whenever any external serial device communicate with microprocessor/microcontrollers, there should be a serial-to-parallel converter unit to convert the serial data into parallel data. Design and implement the suitable 4-bit serial to parallel converter logic circuit for the given requirement using appropriate flip-flop. (Hint: SIPO)

(c). A free-running analog-to-digital converter is one that updates its digital output(for the given analog signal) as often as it can, not waiting for any prompting from another device. If we were to connect a free-running ADC to a computer (microprocessor or microcontroller), we would need some way to sample the ADC's output at times specified by the computer, and hold that binary number long enough for the computer to register it. Otherwise, the ADC may update its output in the middle of one of the computer's "input" cycles, possibly resulting in corrupted data. We could build such a sample-and-hold circuit out of flip-flops, which could shift register inputs multiple bits of data all at once, and transfers that data to its output lines all at once, at the command of a clock pulse. Design and implement the suitable 4-bit shift register to serve for the sample and hold circuit requirement using appropriate flip-flop. (Hint: PIPO)

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	D-FLIP FLOP	IC7474	2
2.	DIGITAL TRAINER KIT	-	1
3.	Connecting wires	-	few

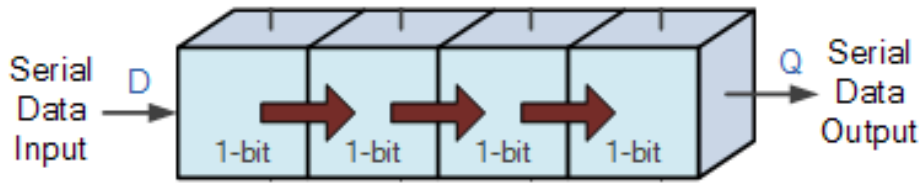
(a). SERIAL-IN SERIAL-OUT (SISO):

THEORY:

Shift Registers are sequential logic circuits, capable of storage and transfer of data. Shift Register is a group of flip flops used to store multiple bits of data. Shift registers are basically a type of register which have the ability to transfer ("shift") data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses. An n-bit shift register can be formed by connecting n flip-flops where each flip flop stores a single bit of data. Shift Registers are used for data storage or for the movement of data and are therefore commonly used inside calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format.

The shift register, which allows serial input (one bit per clock cycle) and produces a serial output is known as Serial-In Serial-Out shift register. In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as serial input. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as serial output. The circuit consists of four D flip-flops which are connected in a serial manner. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop. This type of Shift Register also acts as a temporary storage device or it can act as a time delay device for the data, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc., or by varying the application of the clock pulses.

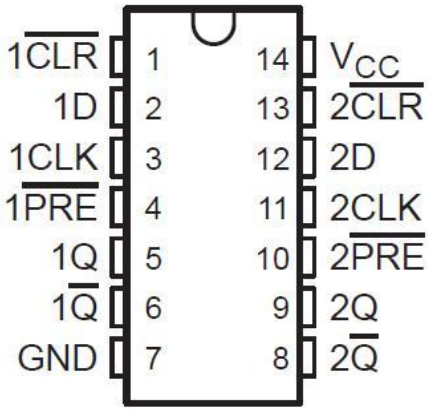
BLOCK DIAGRAM:



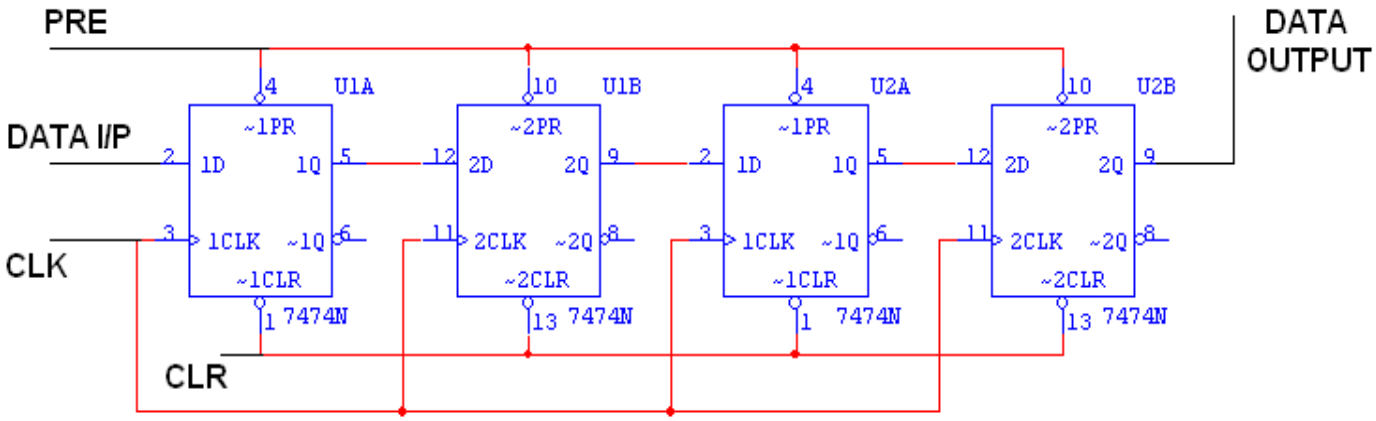
TRUTH TABLE:

CLOCK PULSE NO.	SERIAL DATA INPUT (D)	EACH FLIP FLOP STATE AT Q				SERIAL DATA OUTPUT (Q)
		Q3	Q2	Q1	Q0	
0	0	0	0	0	0	0
1	1	1	0	0	0	0
2	1	1	1	0	0	0
3	0	0	1	1	0	0
4	1	1	0	1	1	1
5	0	0	1	0	1	1
6	0	0	0	1	0	0
7	0	0	0	0	1	1
8	0	0	0	0	0	0

IC7474 PIN DIAGRAM:



LOGIC DIAGRAM:

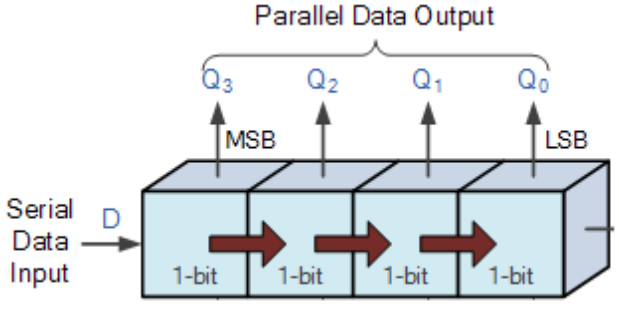


(b). SERIAL-IN PARALLEL-OUT (SIPO):

THEORY:

In the serial in-parallel out shift register, the data is input serially one bit at a time and output in a parallel form. The effect of each clock pulse is to shift the data contents of each stage one place to the right. This data value can now be read directly from the outputs of Q0 to Q3. This means when the data is read in, each read in bit becomes available simultaneously on their respective output line. Then the data has been converted from a serial data input signal to a parallel data output. They are used in communication lines where demultiplexing of a data line into several parallel lines is required because the main use of SIPO register is to convert serial data into parallel data.

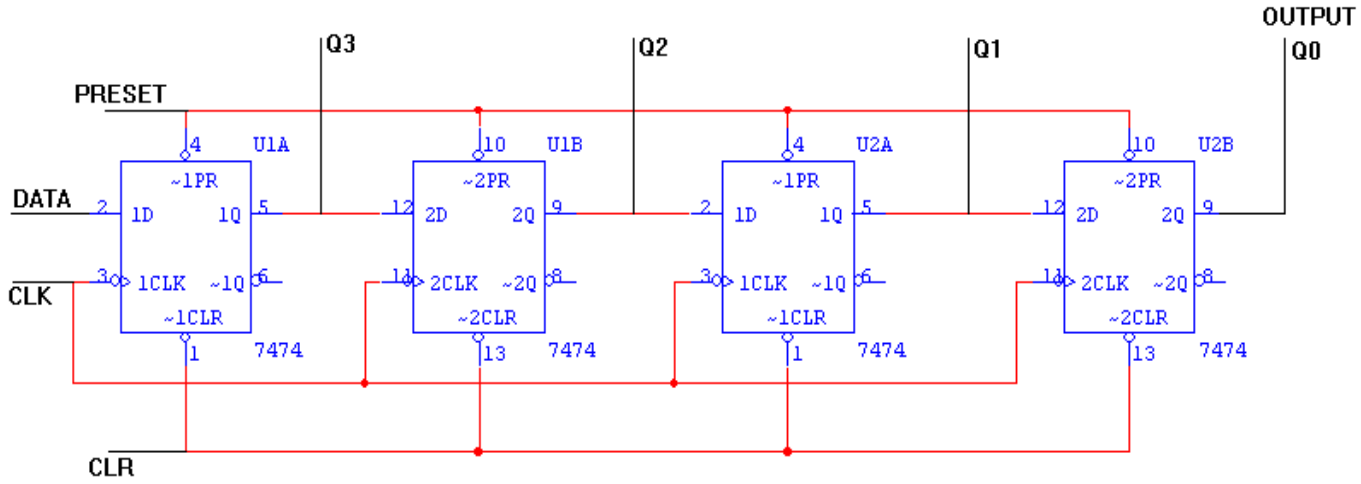
BLOCK DIAGRAM:



TRUTH TABLE:

CLOCK PULSE NO.	SERIAL DATA INPUT (D)	PARALLEL DATA OUTPUT			
		Q3	Q2	Q1	Q0
0	0	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	0	0	1	1	0
4	1	1	0	1	1
5	0	0	1	0	1
6	0	0	0	1	0
7	0	0	0	0	1
8	0	0	0	0	0

LOGIC DIAGRAM:

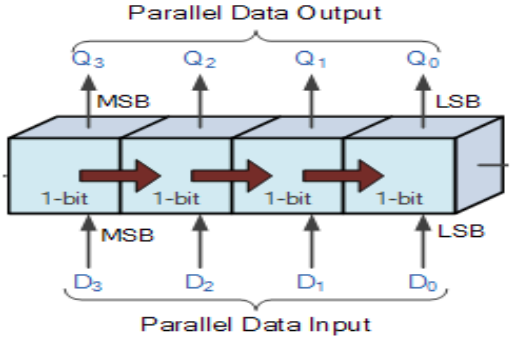


(c). PARALLEL-IN PARALLEL-OUT (PIPO):

THEORY:

The parallel-in parallel-out shift register receives the data input in parallel batches on every clock pulse and the data is shifted and output in parallel. In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required. Data is given as input separately for each flip-flop and in the same way, output also collected individually from each flip-flop. The data is presented in a parallel format to the parallel input pins D0 to D3 and then transferred together directly to their respective output pins Q0 to Q3 by the same clock pulse. This type of shift register also acts as a temporary storage device or as a time delay device similar to the SISO.

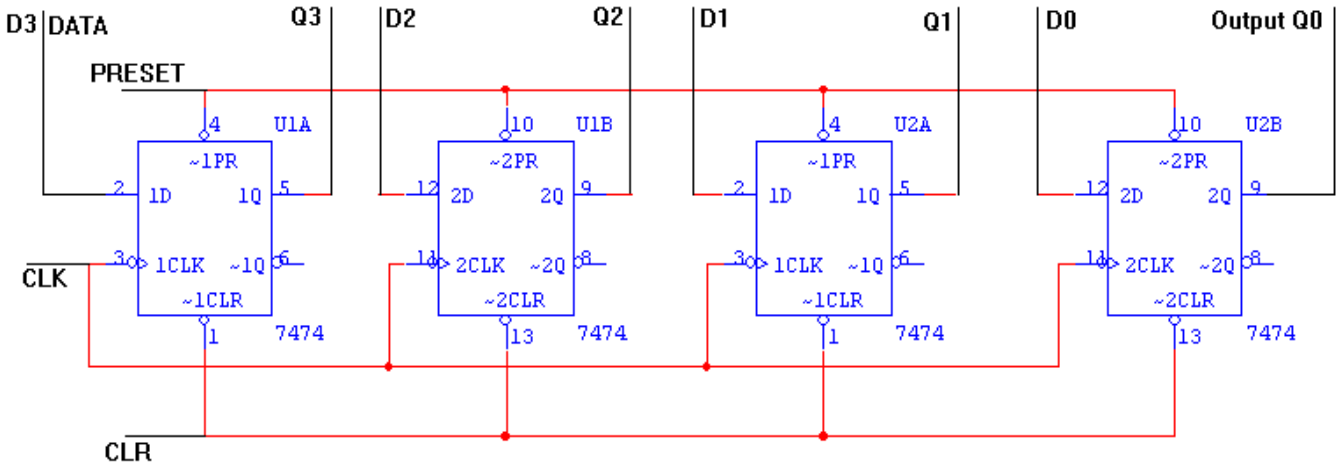
BLOCK DIAGRAM:



TRUTH TABLE:

CLOCK PULSE NO.	PARALLEL DATA INPUT				PARALLEL DATA OUTPUT			
	D3	D2	D1	D0	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1
2	1	0	1	0	1	0	1	0
3	1	1	0	0	1	1	0	0
4	1	0	0	1	1	0	0	1

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus, 4-bit shift registers such as SISO, SIPO and PIPO are constructed using D-flip flop (IC7474), and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **List out different type shift registers.**
   SISO, SIPO, PIPO, PISO

2. **Which flip-flop is most commonly used to design shift registers?**
   D Flip-flop

3. **How much storage capacity (in bit(s)) does each stage in a shift register represent?**
   1-bit

4. **Number of flip-flops required to construct 8-Bit serial-in parallel-out shift register?**
   8

5. **How can parallel data be taken out of a shift register simultaneously?**
   Use the Q output of each FF

6. **How many clock pulses will be required to completely load serially a 5-bit shift register?**
   5

7. **The group of bits 11001 is serially shifted (right-most bit first) into a 5-bit parallel output shift register with an initial state 01110. After three clock pulses, the register contains _____.**
   00101

8. **A serial in parallel out, 4-bit shift register initially contains all 1s. The data nibble 0111 is waiting to enter. After three clock pulses, the register contains _____.**
   1111

9. **An 8-bit serial in/serial out shift register is used with a clock frequency of 2 MHz to achieve a time delay (t_d) of ____.**
   4 μs

10. **List atleast 3 applications of shift registers.**
    - Used for data transfer, manipulation and data storage
    - PISO shift register is used for converting parallel to serial data
    - SIPO used for converting serial to parallel data in communication lines
    - SISO & PIPO shift registers are used for generating time delay in digital circuits

Expt. No. 7	Design and Implementation of 4-Bit Ripple counter and Mod-12 counter

AIM:

(a). In the packaging department of a cricket ball manufacturing company, the balls roll down on a conveyor and get filled into the empty boxes for shipment. Capacity of each box is 16 balls. Each ball is allowed to pass through IR scanner, which generates one-clock pulse for every ball that crosses the scanner. Design an appropriate counter using Flip Flops to count the clock pulse generated from scanner to indicate whether the box is full or not, so that next empty box can be moved into the position.

(b). A simple human counter system for an elevator overload indication module constructed using optical IR sensor, buzzer, microcontroller and seven segment LED display. Number of persons entering into elevator is detected by IR sensor, which generates one-clock pulse per person while entering into elevator. By default, it must display value 0 and increment the number on the display by one for every time a human enters into the elevator. Assume the maximum capacity of the elevator as 12 persons and if it exceed it alarm buzzer. Design an appropriate counter using Flip Flops to indicate overload condition.

COMPONENTS REQUIRED:

S. No.	COMPONENT	SPECIFICATION	QUANTITY
1.	2-INPUT NAND GATE	IC7404	1
2.	JK FLIP-FLOP	IC7476	2
3.	DIGITAL TRAINER KIT	-	1
4.	Connecting wires	-	few

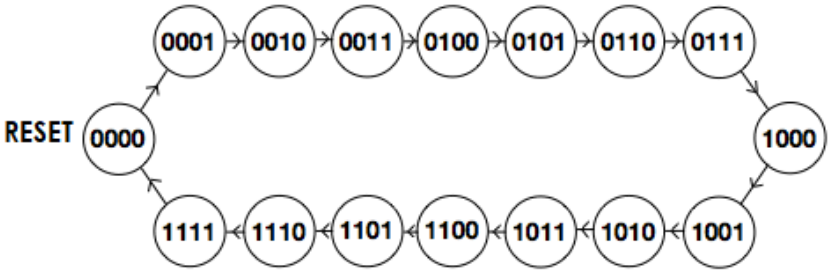
(a). 4-BIT RIPPLE COUNTER:

THEORY:

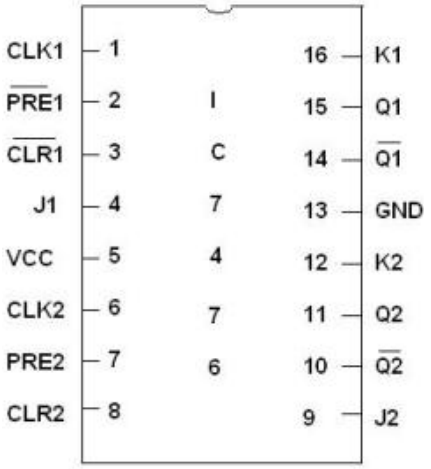
A counter is basically used to count the number of clock pulses applied to a flip-flop. It can also be used for Frequency divider, time measurement, frequency measurement, distance measurement and also for generating square waveforms. Counters are digital circuits made using flip-flops. There are two types of counters Synchronous and Asynchronous. Synchronous counter, as the name suggests have all the flip-flops working in sync with clock pulse as well as each other. Here clock pulse is applied to every flip flop. Whereas in Asynchronous counter clock pulse is applied only to the initial flip flop whose value would be considered as LSB. Instead of the clock pulse, the output of first flip-flop acts as a clock pulse to the next flip flop, whose output is used as a clock to the next in line flip-flop and so on. This counter is also known as ripple counter since the clock pulse ripples through the flip-flops.

It is also known as MOD-16 counter. An n-MOD ripple counter contains n number of flip-flops and the circuit can count up to 2^n values before it resets itself to the initial value. In 4-bit ripple counter, n value is 4 so, 4 JK flip flops are used and the counter can count up to 16 pulses (0000 to 1111). All the JK flip-flops are configured to toggle their state on a downward transition, and output of each flip-flop is fed into the next flip-flop's clock. So, when each bit changes from 1 to 0, it "carries the one" to the next higher bit.

SEQUENCE DIAGRAM:



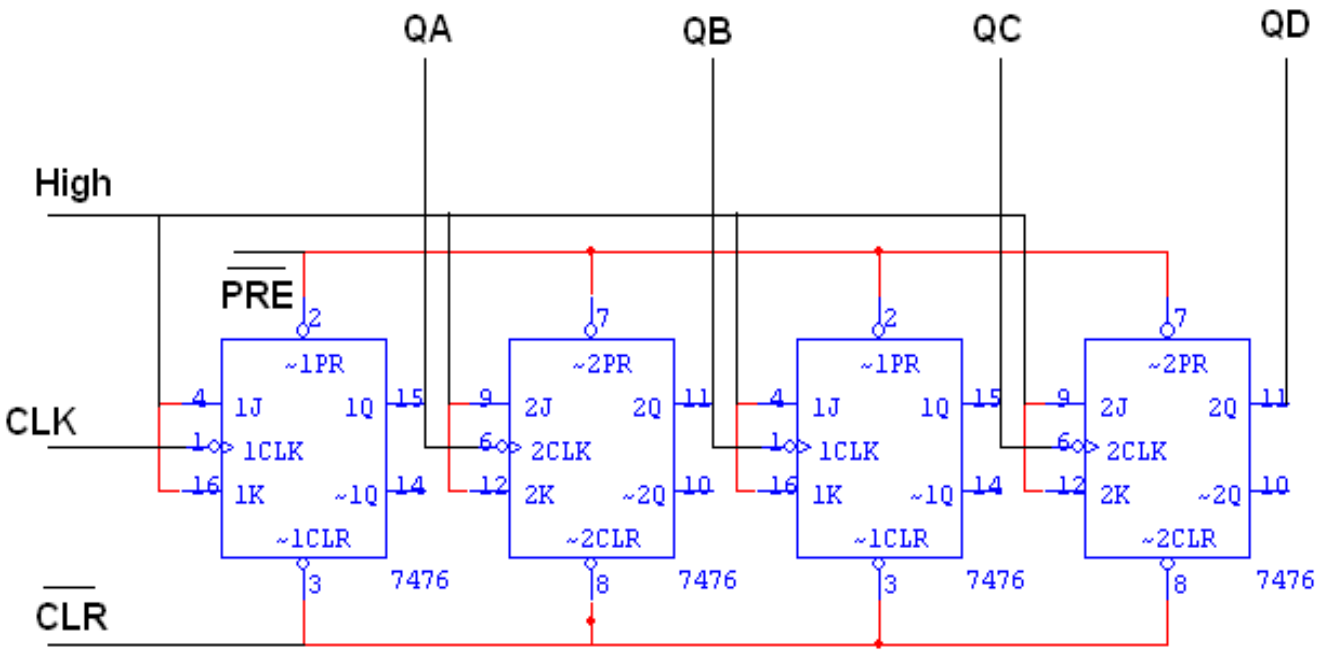
IC7476 PIN DIAGRAM:



`TRUTH TABLE:

CLOCK PULSE NO.	COUNTER OUTPUT AT Q			
	QD	QC	QB	QA
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

LOGIC DIAGRAM:

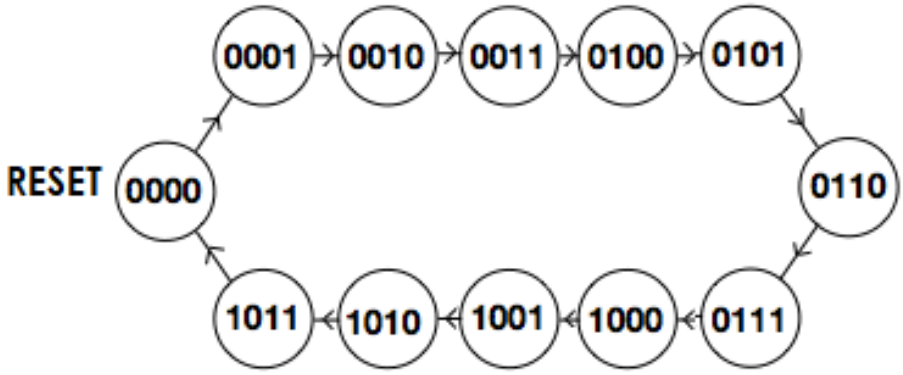


(b). MOD-12 COUNTER:

THEORY:

A modulo 12 (MOD-12) counter circuit, known as divide-by-12 counter, can be made using 4 JK-type flip-flops. The circuit design is such that the counter counts from 0000 to 1011, and then on the 12th count it automatically resets to begin the count again. The trick is to start with a MOD-16 counter and then look for the binary sequence 1100, which is 12 in decimal. Since this binary sequence is unique, we look for the sequence of 1's and feed them into an NAND gate. The LOW output from the NAND gate is then used to control the RESET function on all four flip-flops. Remember that binary is read from right to left (LSB to MSB), whilst the counter output is left to right. Students are likely to make mistakes here, and end up using the incorrect outputs to feed the NAND gate.

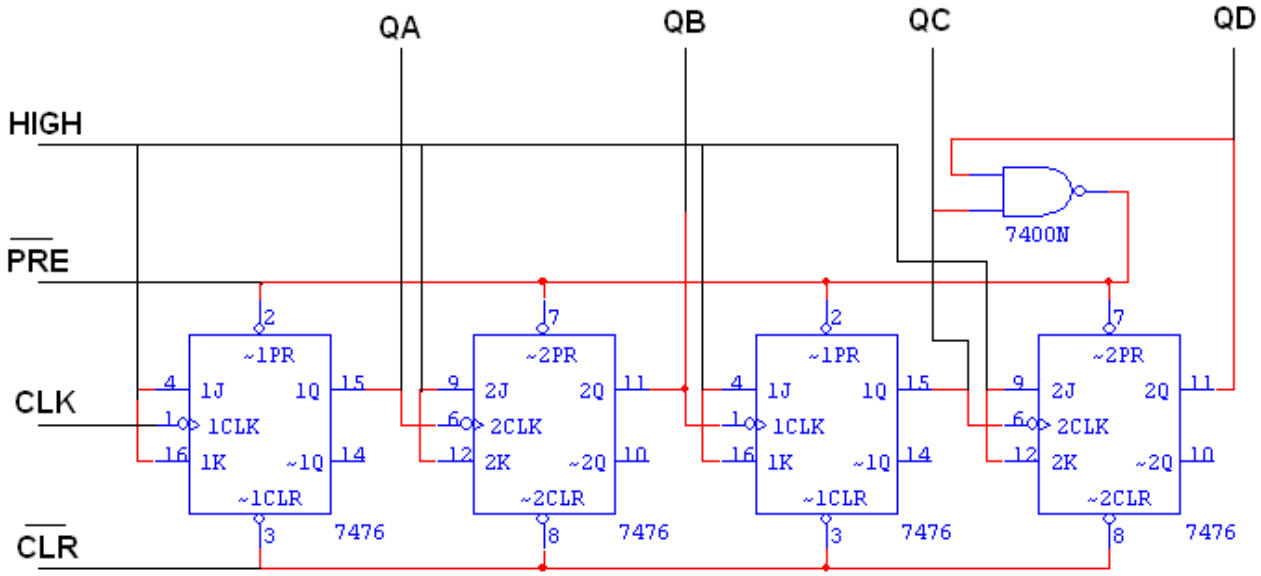
SEQUENCE DIAGRAM:



TRUTH TABLE:

CLOCK PULSE NO.	COUNTER OUTPUT AT Q			
	QD	QC	QB	QA
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

LOGIC DIAGRAM:



PROCEDURE:

1. Place the IC on IC trainer kit
2. Connect the Vcc and ground to respective pins of IC trainer kit
3. Connections are given as per logic diagram.
4. Connect the inputs to the input switches provided in the IC trainer kit
5. Connect the outputs to the switches of output LED's
6. Apply various combinations of input according to the truth table
7. Observe the condition of output LED's and verify the truth table

RESULT:

Thus, 4-bit ripple counter and MOD-12 counters are constructed using JK-flip flop (IC7476), and their truth tables are verified.

VIVA QUESTIONS WITH ANSWERS:

1. **To construct the digital counter circuit, the J-K flip-flop must be configured with J=__ and K=___.**
   1, 1

2. **Number of flip-flops required to construct 8-bit counter is ____ and it can count upto _____ decimal values.**
   8, 255

3. **Number of flip-flops needed to construct MOD-46 counter is _____ and it can count from ____ to ____ in decimal values.**
   6, 0 to 45

4. **One of the major drawbacks to the use of asynchronous counters**
   High-frequency applications are limited because of internal propagation delays

5. **A 5-bit asynchronous binary counter is made up of five flip-flops, each with a 12 ns propagation delay. The total propagation delay is ____.**
   60 ns

6. **Mod-24 counter reaches reset state(0000) for every _____ clock pulses**
   24

7. **A 4-bit ripple counter is holding the count $(1001)_2$. What will the count be after 31 clock pulses?**
   $(1000)_2$

8. **Synchronous (parallel) counters eliminate the delay problems encountered with asynchronous (ripple) counters by**
   Appling input clock pulses simultaneously to each stage

9. **To configure a down counting operation on 4-bit ripple counter each flip-flop is triggered by _____ output of the preceding flip-flops**
   Inverter (Q')

10. **List atleast 3 applications of digital counters.**
    - Measurement of Time, frequency, distance, speed
    - Waveform generation
    - Frequency Division
    - Digital Computers

SOFTWARE EXPERIMENTS

Getting Started With Xilinx ISE Software

INTRODUCTION:

This tutorial will give the procedural steps involved in implementing any digital design with the Xilinx ISE software. Step-by-step instructions will be given to guide the reader through generating a project, creating a design file, compiling the project, simulating the designed file and downloading the design to an FPGA board. This tutorial will show how to:

- ✓ Part I : Creating a new project in Xilinx ISE 14.1
- ✓ Part II : Implement a design using Verilog HDL
- ✓ Part III : Simulate the Verilog design using the ISim + Verilog test fixture

Before proceeding with the tutorial please make sure that your computer machine installed with "Xilinx ISE 14.1". Some of the following procedures may be different depending on the version of ISE.

XILINX ISE SOFTWARE:

Xilinx ISE (**I**ntegrated **S**oftware **E**nvironment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. The Web Edition is a free version of Xilinx ISE that can be downloaded at no charge. It provides synthesis and programming for a limited number of Xilinx devices. In particular, devices with a large number of I/O pins and large gate matrices are disabled.

XILINX FPGA SPARTAN-3 DEVELOPMENT BOARD:

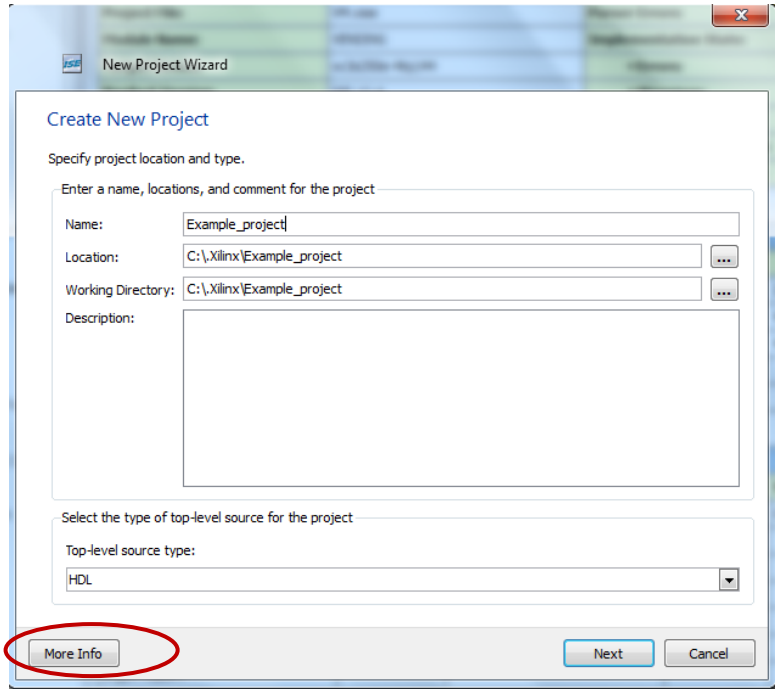
The FPGA development board what we chosen for implementing our design is **VE-XILINX-S3EX** developed by Frontline Electronics. This development board is build on the Xilinx XC3S250E Spartan-3E FPGA development platform. The Spartan 3E series of FPGA is a very low cost, high-performance logic solution for high-volume, consumer-oriented applications. This Xilinx Spartan-3E FPGA development board with USB Interface is most popular. This board provides a basic development platform for the Spartan 3E device with all I/O available to the user such as toggle switches, LED, LCD, RTC, 7-segment display, 4x4 Matrix keypad etc,. The device may be programmed in-circuit through the USB/JTAG port from the PC using top view programmer. These FPGA development board enable our customers improvise their design cycle and meet desired goals.

PART I: CREATING A NEW PROJECT IN XILINX ISE 14.1

0. To start the ISE Design Suite, double-click the Project Navigator icon on your desktop, or select Start > All Programs > Xilinx ISE Design Suite > Xilinx Design Suite 14 > ISE Design Tools > Project Navigator.

1. Create a new project. To create a new project use either click on New Project tab or select File → New Project and change the Name and Location to whatever you like. [**Note:** Xilinx does not allow spaces in path or file names! For example "C:\ECE 3700 will not work, same for the file name! Use the under_score for spaces if you need to]

The selected Top Level Source Type is HDL because first we will discuss the procedure for verilog HDL design. The dialog box for the project wizard looks like:

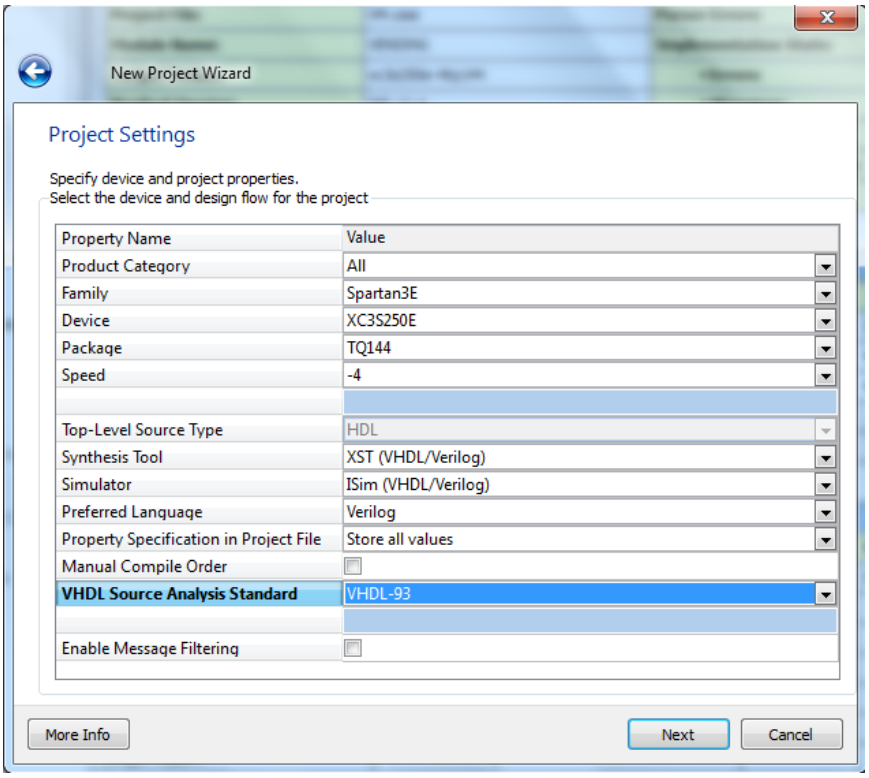


2. Click NEXT and Device Properties page appears, Select the following options in the fields of project settings on the Device Properties page:

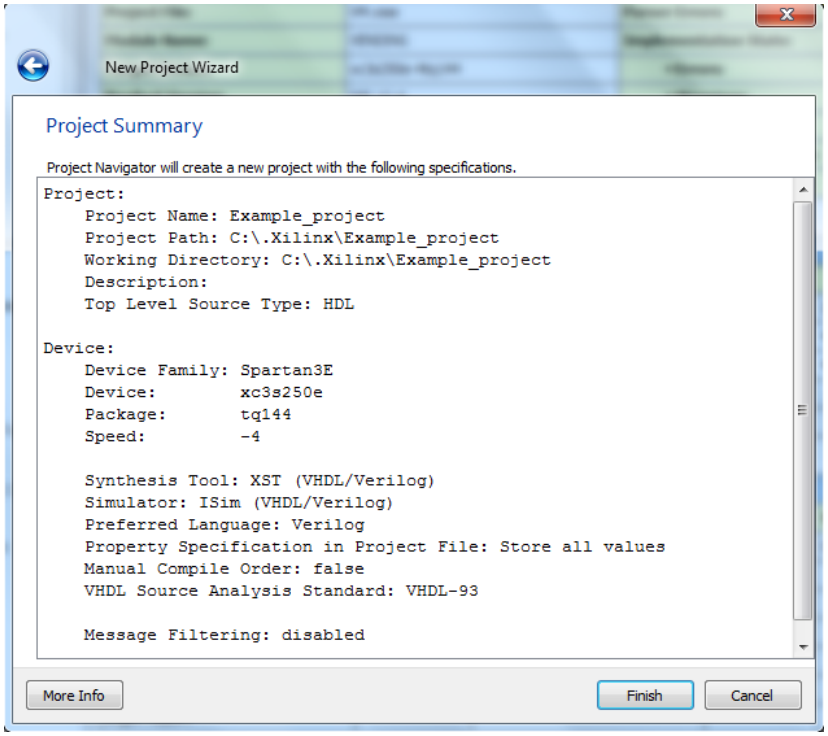
- Product Category: All
- Family: Spartan3E
- Device: XC3S250E
- Package: TQ144
- Speed: -4
- Synthesis Tool: XST (VHDL/Verilog)
- Simulator: ISim (VHDL/Verilog)
- Preferred Language: Verilog

Other properties can be left at default values. This is a details of FPGA device used for implementation

[**Note:** If you fail to set the correct options in this part, you will not be able to implement your design and program it on the FPGA device]



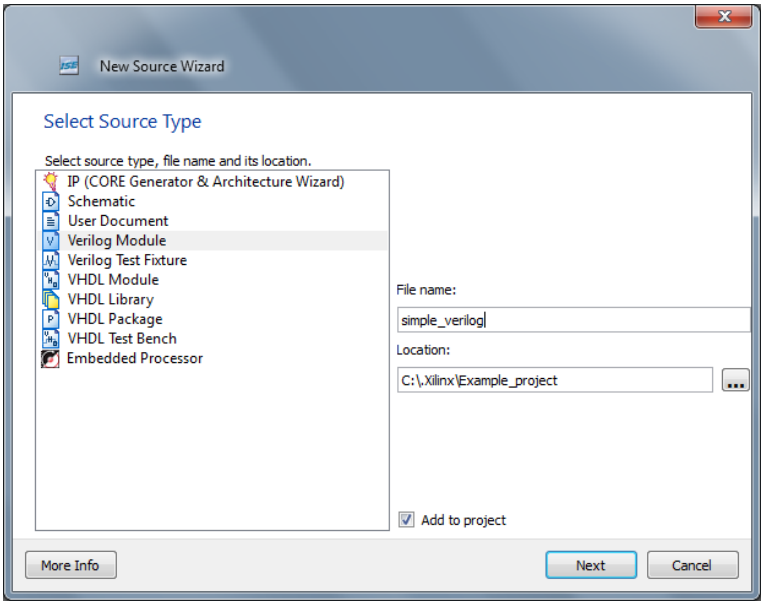
3. Click NEXT and review the project summary page and then click FINISH, it is always good to double check the project summary to prevent the problems while implementing the design.



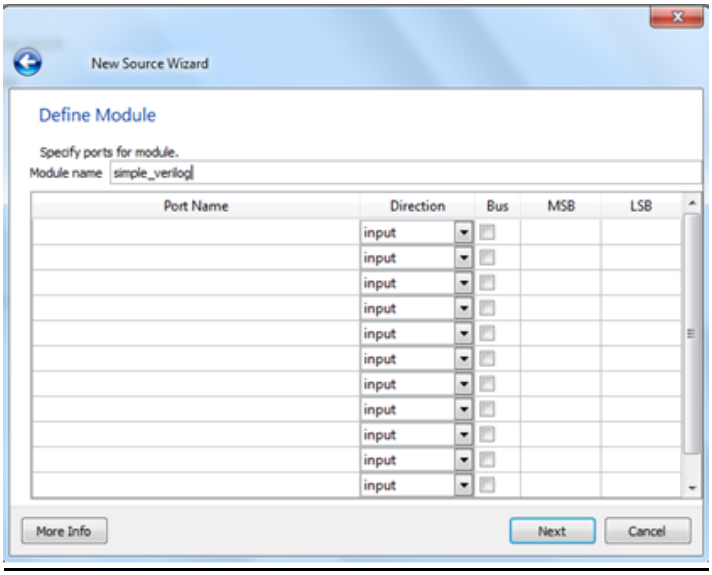
PART II: IMPLEMENT A DESIGN USING VERILOG HDL

1. Now we will explore the implementation of the function F=(A& (!B)) | (B & C) using a "Verilog" module. First we need to add a new source file on the created project. For adding a new source file choose Project → New Source or right click on the Hierarchy section of the design windows to get the new source wizard dialog box and then choose "Verilog Module" and give it a file name.

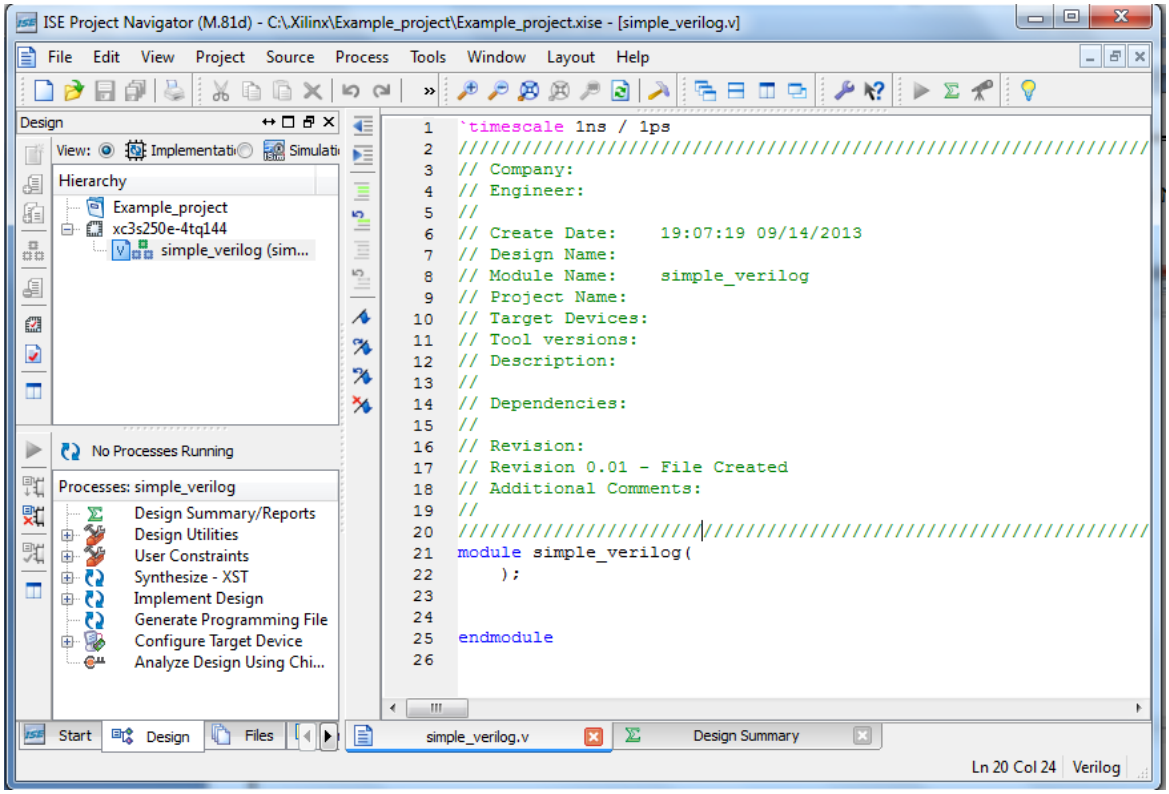
   [**Note:** Remember to only use "Verilog Module" for Implementation and "Verilog Test Fixture" for simulation purposes]



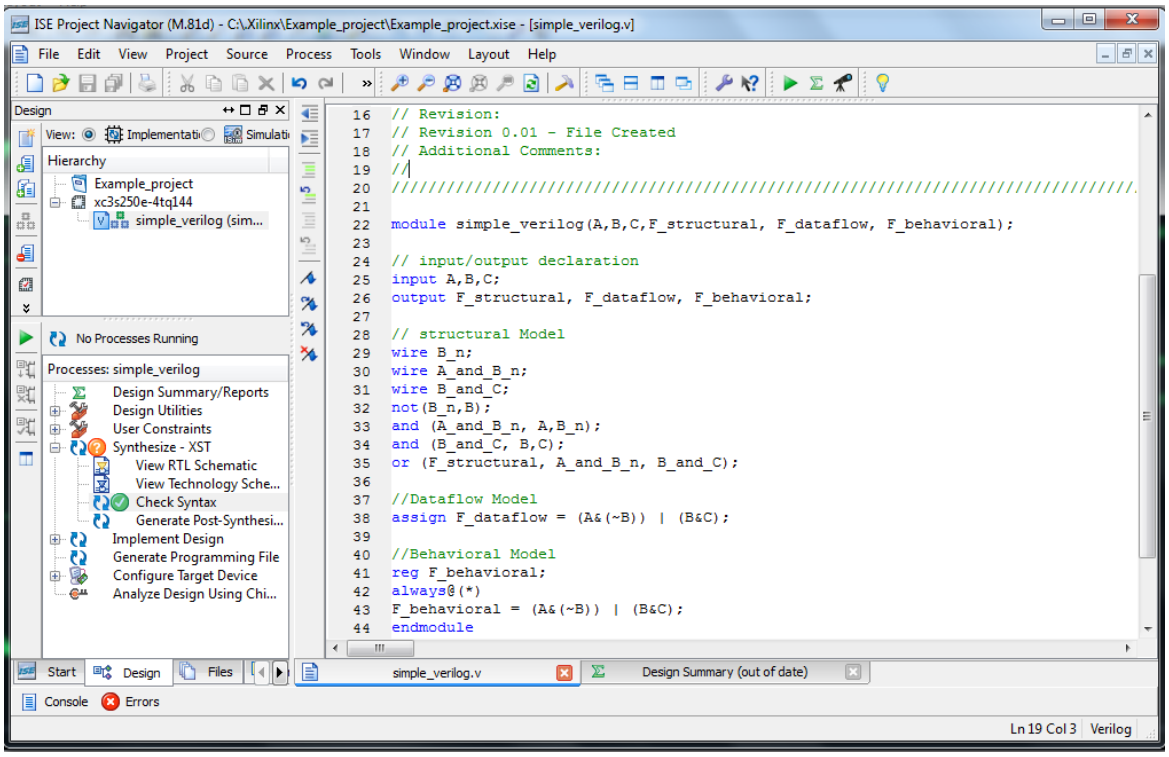
2. Click NEXT and you should see the define module box. Here you can setup I/O names with correct polarity and a choice for buses and the width. Note that you do not have to add anything here right away and you can always add the I/O definitions to the module's header when it is created. So here no I/O names are specified to proceed further click NEXT. After you're done click NEXT and then observe the summary page for a quick review of your I/O list.



3. Now you can notice that the Verilog module file is added to the hierarchy section as a part of the project. You can also notice that a tab for the Verilog file opened in the ISE main pane. Type the Verilog HDL code for F=(A& (!B)) | (B & C) in the main pane.



4. For implementing the function F we can write the Verilog code in any one of following three modelling.



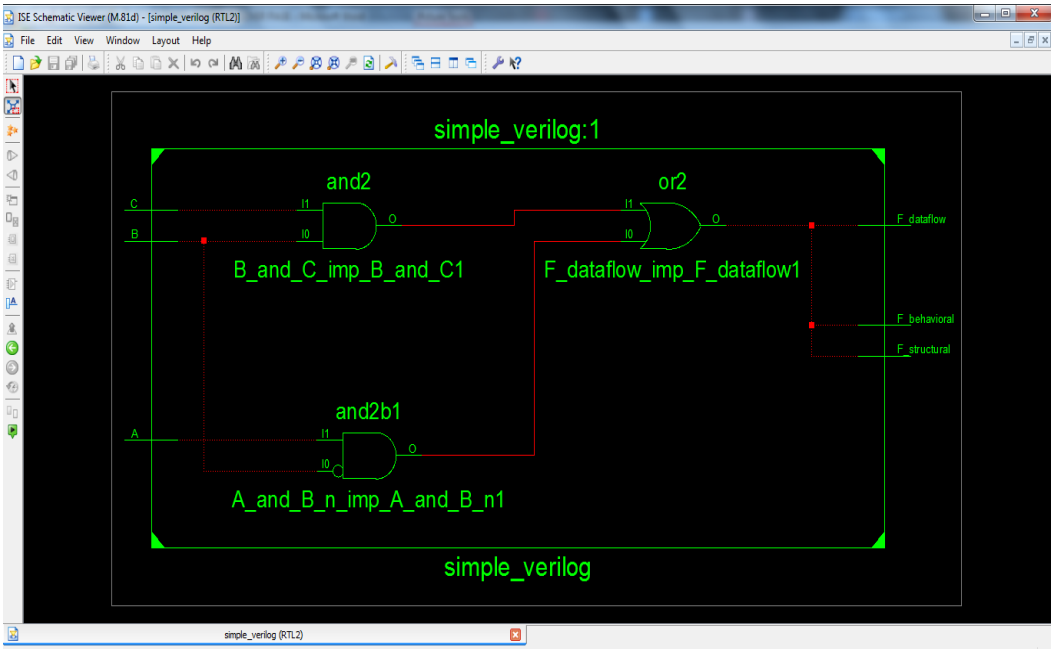
The Structural model: This is done using gate primitives that are automatically taken from the Xilinx libraries by calling their name and passing parameters.

The Data flow model: Using the "assign" keyword to assign the results of the function expression to the output.

The Behavioural model: Using "Always" keyword we can implement the same function using behavioural modelling. The difference is that it only wakes up and assign the output when any of the inputs are changes, hence the (*) which means, "any change". One thing to notice is that any output being assigned inside of an always block needs to be declared as a "reg" for synthesis purposes, and that you cannot use the "assign" keyword inside of such block.

5. Next process is to perform check syntax operation available under synthesis-XST option in the process window. Once the green tick mark present in check syntax option then it indicate no error in the typed Verilog code. We will get Red Cross mark on the check syntax option for the indication of error in the program. To solve the errors in the program select the errors tab in the transcript window, look at the errors in the program and solve one by one until check syntax option gets green tick mark.

6. Once check syntax operation got over, make a double click on "View RTL Schematic" option under the synthesis-XST process. To verify the logic diagram created for the written logic in Verilog code, select the created Verilog module (In our case it is "simple_verilog") and add it to the selected element window, then click on create schematic option. Again, double click on the RTC diagram to see the internal parts of the created logic.



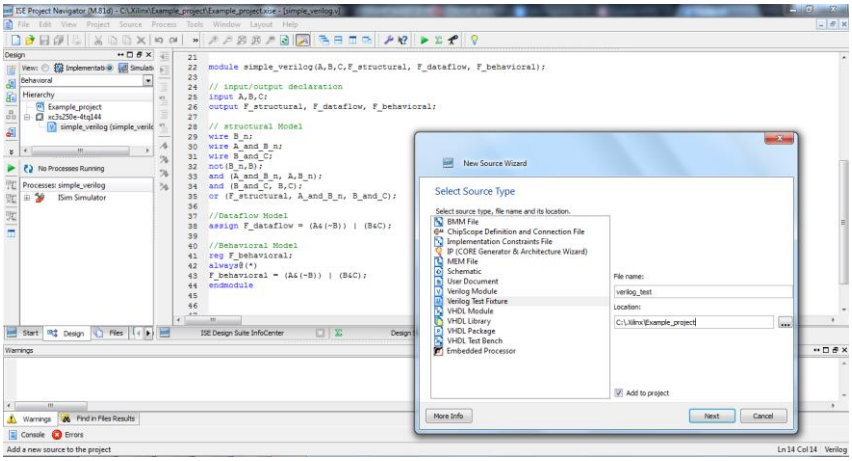
Now the circuit is ready for simulation or implementation on the Xilinx FPGA board. But, before doing implementation of the design it is better to verify the logic using behavioural simulation process (optional). In the next part will discuss about how the same function F=(A&(!B)) | (B&C) can be implemented using schematic logic.

PART III: SIMULATE THE SCHEMATIC / VERILOG DESIGN USING ISIM & VERILOG TEST FIXTURE

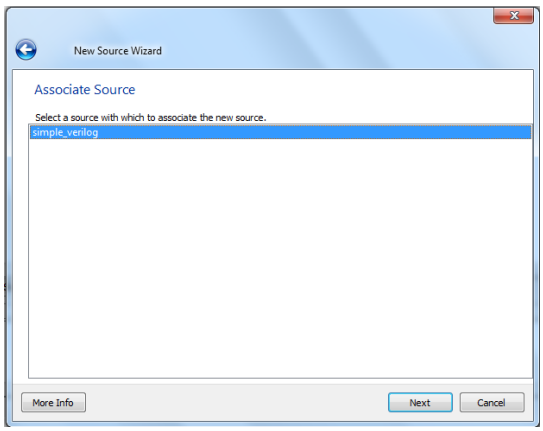
Now that you have a saved Verilog module and schematic, you need to simulate its behaviour. The ISIM is the build in simulator of the XILINX ISE software, which is essentially a Verilog simulator. In order to simulate the circuit you need:

Test bench: It is a file that becomes a top module to your design and applies inputs to your circuit, and potentially checks that the outputs are correct. This will be another Verilog file written slightly differently than circuit implementation. The test bench will instantiate one copy of your circuit, and call it UUT for "Unit Under Test". You will then write the Verilog statements that set the inputs to your circuit (the UUT), and looks at the outputs produced by your circuit.

1. First you need to ensure that the ISE is changed to "Simulation" from implementation. Go to the top left pane and change the "View" field to simulation. The design window will then change slightly with different options. Referring back to the same step in creating a "New Source" create a "Verilog Test Fixture" to create a Verilog file that will contain the test code.



2. Click NEXT and choose which design you want to associate the test bench with. This is very important as you will have multiple modules or schematics in the future and you need to be sure which design is under test using the test bench. In this case I will just choose the "simple_verilog" module to be tested.



3. Click NEXT and after observing the summary click FINISH. Now a new piece of Verilog code generated for you. This Verilog code instantiates the "simple_verilog" module as the UUT, and includes some other stuff related to how the UUT is connected to the test bench. It looks like this:

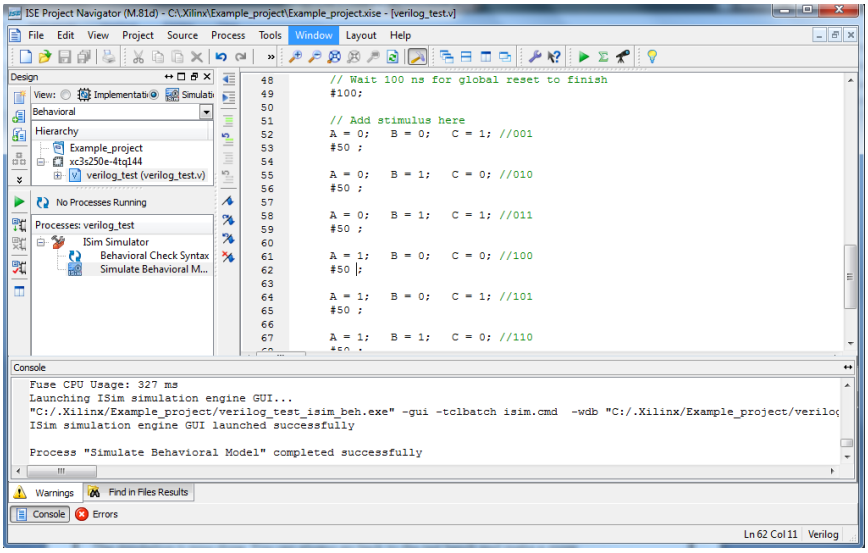
```
24
25  module verilog_test;
26     // Inputs
27     reg A;
28     reg B;
29     reg C;
30     // Outputs
31     wire F_structural;
32     wire F_dataflow;
33     wire F_behavioral;
34
35     // Instantiate the Unit Under Test (UUT)
36     simple_verilog uut (
37        .A(A),
38        .B(B),
39        .C(C),
40        .F_structural(F_structural),
41        .F_dataflow(F_dataflow),
42        .F_behavioral(F_behavioral)
43     );
44
45     initial begin
46        // Initialize Inputs
47        A = 0;
48        B = 0;
49        C = 0;
50
51        // Wait 100 ns for global reset to finish
52        #100;
53
54        // Add stimulus here
55     end
56  endmodule
57
58
```

4. You can now write your test bench code as an initial block right before the end module. Basically you set the values of your inputs, and tell the simulator how long to wait between each change on the inputs. The results will eventually be plotted on a waveform for you. Verilog syntax for setting a variable is very simple, and the #50 notation just means for the simulation to wait for 50 ticks of the simulation clock before moving on to the next statement. A very simple test bench for this circuit looks like the following.

   Add the lines between initial and end to drive the inputs with different values so that we can see what the circuit result is. Note that each statement in Verilog ends with a semicolon, and you can put multiple statements on a line if you like.

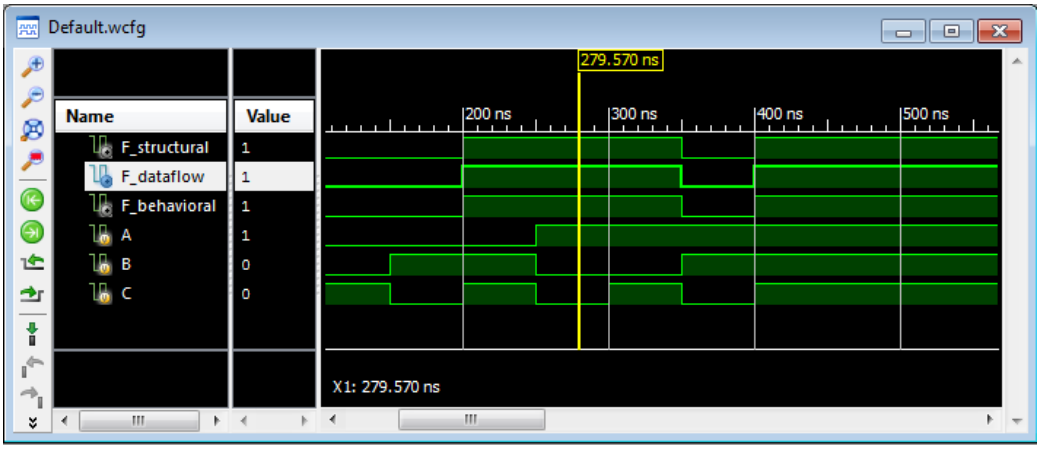
```
39        .C(C),
40        .F_structural(F_structural),
41        .F_dataflow(F_dataflow),
42        .F_behavioral(F_behavioral)
43     );
44
45     initial begin
46        // Initialize Inputs
47        A = 0;   B = 0;   C = 0; //000
48        // Wait 100 ns for global reset to finish
49        #100;
50
51        // Add stimulus here
52        A = 0;   B = 0;   C = 1; //001
53        #50 ;
54
55        A = 0;   B = 1;   C = 0; //010
56        #50 ;
57
58        A = 0;   B = 1;   C = 1; //011
59        #50 ;
60
61        A = 1;   B = 0;   C = 0; //100
62        #50 ;
63
64        A = 1;   B = 0;   C = 1; //101
65        #50 ;
66
67        A = 1;   B = 1;   C = 0; //110
68        #50 ;
69
70        A = 1;   B = 1;   C = 1; //111
71        #50 ;
72        end
73  endmodule
```

5. We usually want to test all possible inputs to be able to draw a better conclusion on whether the circuit is functioning correctly. After you're satisfied with the input setting of your test bench, make sure to save. Always observe the console window to look out for errors after saving. Now you are ready to simulate your Verilog circuit.



Observe that the test bench Verilog file is now the top module to your "simple_verilog" module in the simulation design view. It is very important to have the test bench file selected for simulation or things will go wrong. After selecting and highlighting the test bench file in the design windows, double-click the "Simulate Behavioural Model" to see the waveform generated by the ISim.

6. The output will be displayed as waveforms as shown. Note that the simulator is by default set up to simulate for 1000ns, at the beginning of the simulation. I had to zoom out a little to see this view. The values reported for A, B, S, and F are the values seen at the blue bar. You can pick up (with the mouse) and move the blue bar to see the values at different points in the simulation.



By looking at the waveform, we can see that all three different forms of expressing the function in Verilog (structural, functional, and behavioural) are all holding the same behaviour throughout the simulation. You can click on the waveform in different places (the yellow line is where in the range of time in the waveform it was clicked) and you can see values quickly for all I/O in the "Name" and "Value" sections to the left of the waveform. The simulation is now done.

Expt. No. 8	Introduction to Verilog HDL and Verification of Simple Boolean Expression

AIM:

(a). To understand the basics of Verilog HDL and concepts of various Verilog modelling such as Gate-level, Dataflow, Behavioural and Switch-level modelling.

(b). In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point. Design a digital logic circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point. Write a Verilog HDL code in gate level modelling for the proposed design and verify its logic by simulation using Xilinx ISE Simulator.

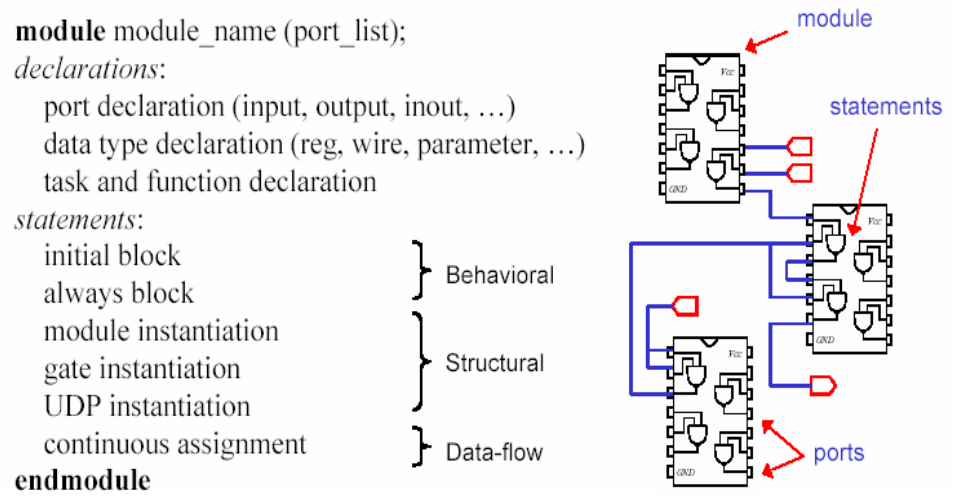
H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

(a). INTRODUCTION TO VERILOG HDL:

THEORY:

The fundamental characteristics of a digital system are defined by the concepts of entity, connectivity, concurrency and timing. The sequential model used in traditional programming languages (i.e. "C") cannot capture the characteristics of digital hardware and There is a need for special languages (i.e., HDLs) that are designed to model digital hardware. Hardware Descriptive Language (HDL) is a language used to describe digital system. The goal of an HDL is to describe and model digital systems faithfully and accurately. A hardware description language looks much like a programming language such as C. The two main HDLs are Verilog and VHDL for digital system design. VERILOG HDL – VERIfiable LOGic HDL: Programming language used in the design and verification of digital circuits at the register-transfer level of abstraction. Verilog is similar to C/Pascal programming language, case-sensitive and its file extension is (.v). Verilog versions: Verilog-95, 2001, 2005, System Verilog 2009. Structure of Verilog HDL is shown below.



Logic of the digital circuits can be realized in Verilog HDL using four different modelling approaches. This includes gate-level, dataflow, behavioural and switch-level. However, our focus would be to understand first three kinds of Verilog HDL modelling.

Gate-level modelling:

At gate level, the circuit is described in terms of gates (e.g., and, nand, or). All logic circuits can be designed by using basic gates. Verilog supports basic logic gates as predefined primitives. There are three classes of basic gates.

- Multiple-input gates: and, or, nand, nor, xor, xnor
- Multiple-output gates: buffer, not
- Tristate gates: bufif0, bufif1, notif0, notif1

These primitives are instantiated like modules except that they are predefined in Verilog and do not need a module definition. Logic gates can be used in design using gate instantiation. Basic syntax for multiple-input gates is:

Multiple_input_Gate_type [instance_name] (output, input1, input2,……..inputn);

Basic syntax for multiple-output gates is:

Multiple_output_Gate_type [instance_name] (output1, output2,……..outputn, input);

Half-adder example for gate-level modelling:

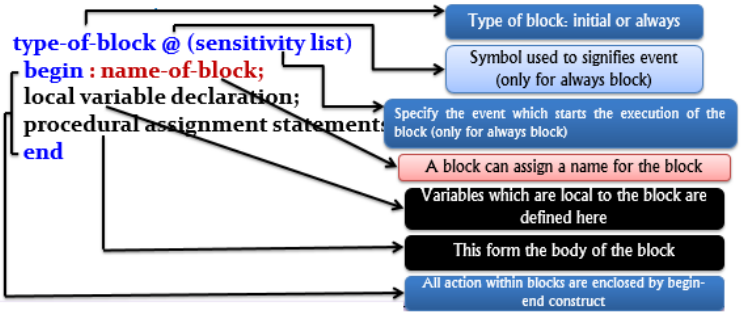
```
module halfadder(s,c,a,b);
input a,b;
output s,c;
xor(s,a,b);
and(c,a,b);
endmodule
```

Behavioural modelling:

Behavioral model enables you to describe the system at a higher level of abstraction. All we need to do is to describe the behavior of our design

- Action → How the model of our circuit should behave?
- Timing control → At what time do what thing & At what condition do what thing

In Verilog procedural block are the basic of behavior modeling. We can describe a one logic in one procedural block. Procedural block types: (i) initial and (ii) always. All initial & always statement execute concurrently starting at time t=0. A module may contain any number of initial & always statement. Structure of the procedural block is shown below.



Half-adder example for behaviour-level modelling:

```
module halfadder(s,c,a,b);
input a,b;
output s,c;
reg s,c;
always @ (a or b)
begin
        s=a^b;
        c=a&b;
end
endmodule
```

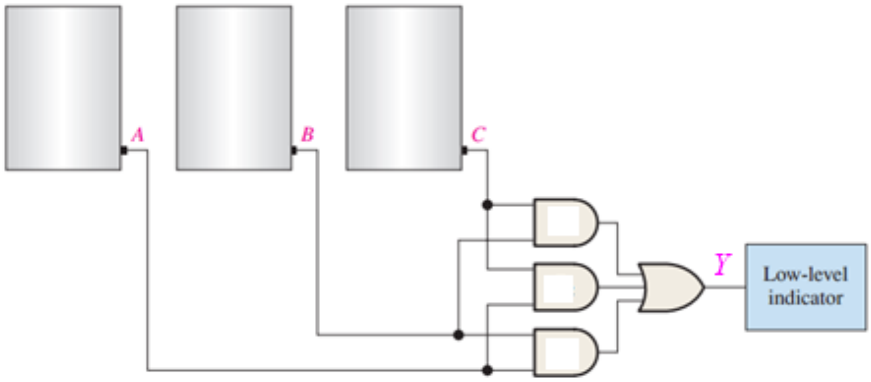
Data-flow modelling:

Data flow level description of a digital circuit is at higher level, it makes the circuit description more compact as compared to design through gate primitives. Design implement using data flow modeling uses a continuous assignment statement and Verilog provides different operators to perform a function. The assignment statement start with the keyword assign and results are assigned to nets. In general dataflow modelling, various logical and arithmetic operators used to realize the required logic. Some of the logical operators are & (bitwise AND), | (bitwise OR), ~(bitwise NOT), ^ (bitwise XOR etc.,

Half-adder example for dataflow modelling:

```
module halfadder(s,c,a,b);
 input a,b;
 output s,c;
assign s=a^b;
assign c=a&b;
endmodule
```

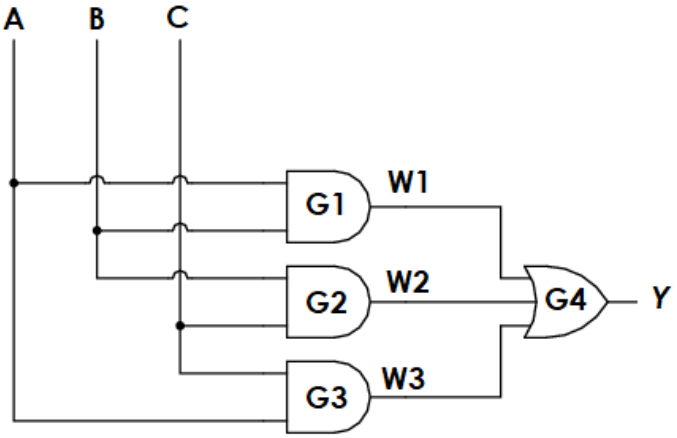
(b). VERIFICATION OF BOOLEAN EXPRESSION USING VERILOG HDL: Y = AB+BC+AC



TRUTH TABLE:

INPUTS			INTERMEDIATE OUTPUTS			OUTPUT
A	B	C	AB	AC	BC	Y = AB+AC+BC
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

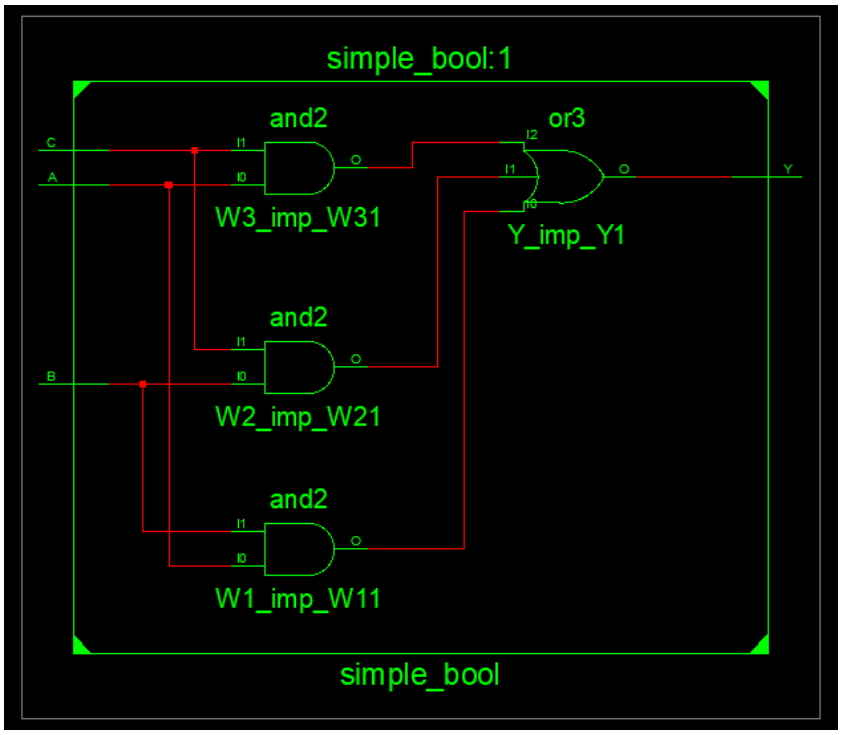
LOGIC DIAGRAM:



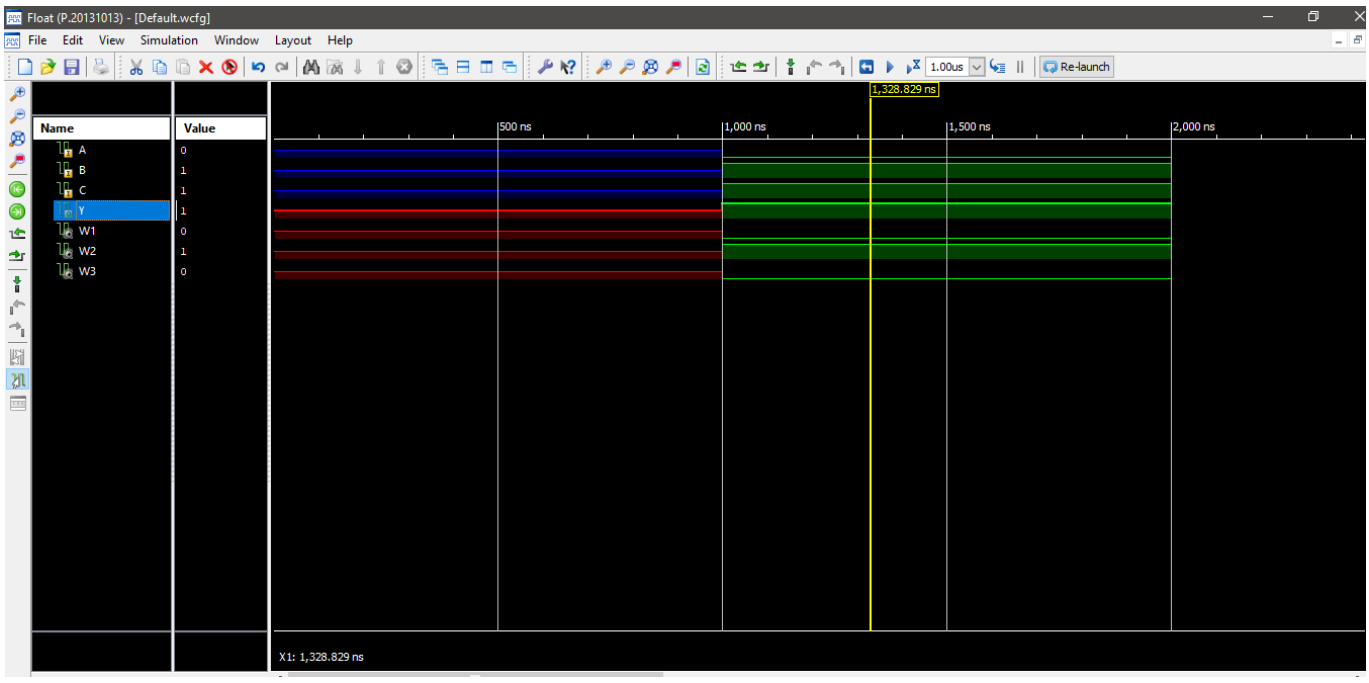
VERILOG HDL CODE (Gate-level):

```
module simple_bool(Y,A,B,C);
input A,B,C;
output Y;
wire W1,W2,W3;
and G1(W1,A,B);
and G2(W2,B,C);
and G3(W3,A,C);
or G4(Y,W1,W2,W3);
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, the basics of Verilog HDL studied and simple Boolean expression realized in Verilog HDL gate level modelling and its truth table is verified by simulation using Xilinx ISE Simulator.

VIVA QUESTIONS WITH ANSWERS:

1. **State the purpose of Hardware Descriptive Language (HDL).**
   HDL is a language used to describe digital system.

2. **Name two most commonly used Hardware Descriptive Languages.**
   VHDL and Verilog HDL

3. **In Verilog HDL module, port list section describes list of ____ and _____ of the design.**
   Inputs, outputs

4. **State four types of program modelling in Verilog HDL?**
   Gate-level, Behavioural, Dataflow and Switch-level

5. **In behavioural modelling logic of the design is described in _____ block.**
   Procedural block

6. **Write a gate-level primitive instantiation statement to realize 4-input NAND Gate (Assume A, B, C, D are input and Y is the output.**
   nand(Y,A,B,C,D);

7. **In data-flow modelling the continuous assignment statement start with the keyword _____ and results are assigned to _____.**
   Assign, nets

8. **Which Verilog modelling enables you to describe the system at a higher level of abstraction?**
   Behavioural modelling

9. **Verilog HDL keywords can't be assigned as a module name. True or false?**
   True

10. **Basic syntax for multiple-output gates is?**
    Multiple_output_Gate_type [instance_name] (output1,……..output n, input);

Expt. No. 9	Design and Simulate Full Adder & 4-Bit Parallel Adder

AIM:

A digital circuit designer needs to design a partial simplified Arithmetic Logic Unit (ALU) in Verilog HDL to perform a 4-bit addition operation on two operands, but he had already created a full adder logic for another application. Help him to realize partial simplified ALU design to perform 4-bit addition by instantiating full adder logic. Verify the output logic by simulation using Xilinx ISE Simulator. (Hint: 4-bit Parallel Adder using Full adder)

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

PROCEDURE:

1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the simulate behavioural model to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per truth table.

4-BIT PARALLEL ADDER USING FULL ADDER:

THEORY:

An adder is a digital logic circuit in electronics that implements addition of numbers. In many computers and other types of processors, adders/Subtractors are used to calculate addresses, similar operations and table indices in the ALU also in other parts of the processors. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. A full adder is a digital circuit that adds three one-bit binary numbers, two operands and a carry bit. The adder outputs two numbers, a sum and a carry bit. Full adders are made from XOR, AND, OR gates.

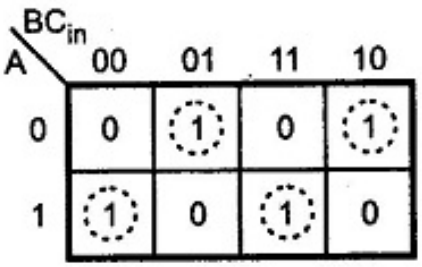
Parallel Adder is a digital circuit capable of finding the arithmetic sum of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel. It consists of full adders connected in a chain where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain. A "n" bit parallel adder requires "n" full adders to perform the operation. So for the two-bit number, two adders are needed while for four bit number, four adders are needed and so on. Consider the example that two 4-bit binary numbers $B_3\ B_2\ B_1\ B_0$ and $A_3\ A_2\ A_1\ A_0$ are to be added with a carry input C_{in}. This can be done by cascading four full adder circuits as shown in logic diagram section. The least significant bits A_0, B_0, and C_{in} are added to the produce sum output S_0 and carry output C_1. Carry output C_1 is then added to the next significant bits A_1 and B_1producing sum output S_1 and carry output C_2. C_2 is then added to A_2 and B_2 and so on. Thus finally producing the four-bit sum output $S_3\ S_2\ S_1\ S_0$ and final carry output C_{out}.

FULL ADDER:

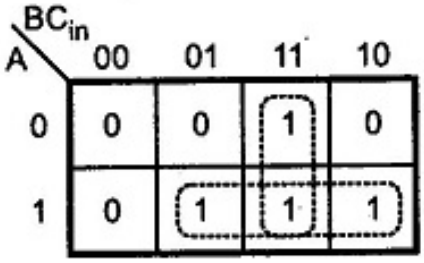
TRUTH TABLE:

INPUTS			OUTPUTS	
A	**B**	**C_{in}**	**SUM**	**CARRY**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-MAP:

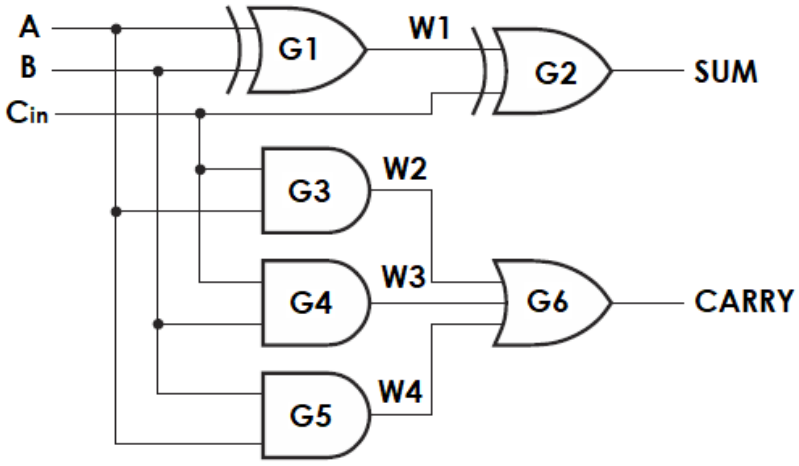


SUM = A'B'C_{in} + A'BC_{in}' + ABC_{in}' + ABC_{in}



CARRY = AB + BC_{in} + AC_{in}

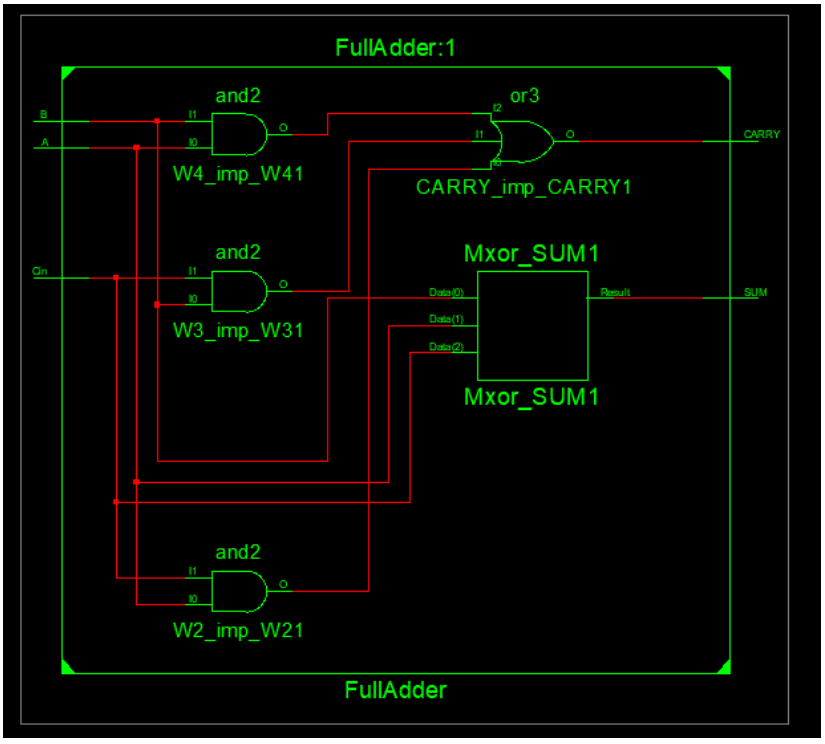
LOGIC DIAGRAM:



VERILOG HDL CODE - FULL ADDER (Gate-level):

```
// Module Name:    FullAdder
module FullAdder(SUM, CARRY, A, B, Cin);
input A,B,Cin;
output SUM, CARRY;
wire W1,W2,W3;
xor G1(W1,A,B);
xor G2(SUM,W1,Cin);
and G3(W2,A,Cin);
and G4(W3,B,Cin);
and G5(W4,A,B);
or G6(CARRY,W2,W3,W4);
endmodule
```

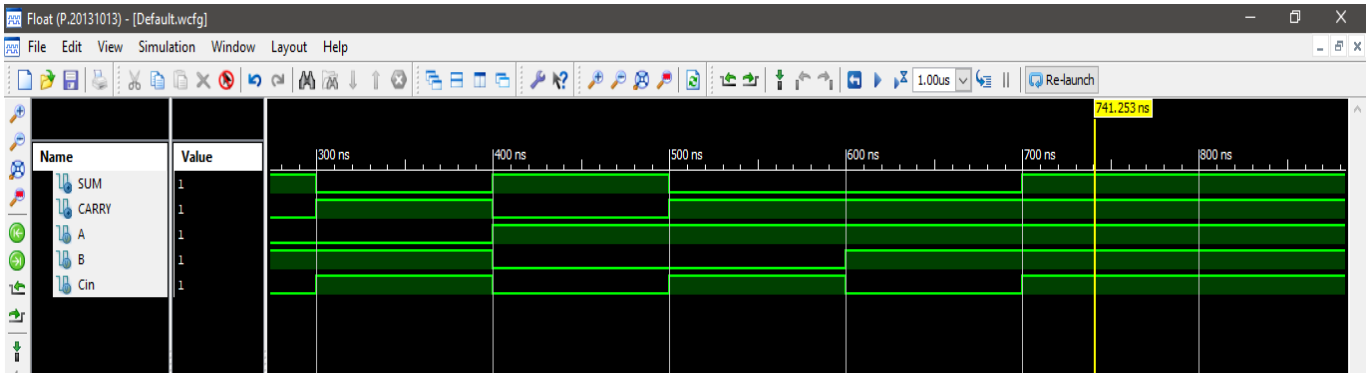
RTL SCHEMATIC:



TEST BENCH:

```
module FullAdder_tb;
    // Inputs
    reg A;
    reg B;
    reg Cin;
    // Outputs
    wire SUM;
    wire CARRY;
    // Instantiate the Unit Under Test (UUT)
    FullAdder uut (
        .SUM(SUM),
        .CARRY(CARRY),
        .A(A),
        .B(B),
        .Cin(Cin)
    );
    initial begin
        // Initialize Inputs
        A = 0; B = 0; Cin = 0;
        #100;
        A = 0; B = 0; Cin = 1;
        #100;
        A = 0; B = 1; Cin = 0;
        #100;
        A = 0; B = 1; Cin = 1;
        #100;
        A = 1; B = 0; Cin = 0;
        #100;
        A =1; B = 0; Cin = 1;
        #100;
        A = 1; B = 1; Cin = 0;
        #100;
        A = 1; B = 1; Cin = 1;
        #100;
    end
endmodule
```

SIMULATION OUTPUT:



4-BIT PARALLEL ADDER:

TEST CASES:

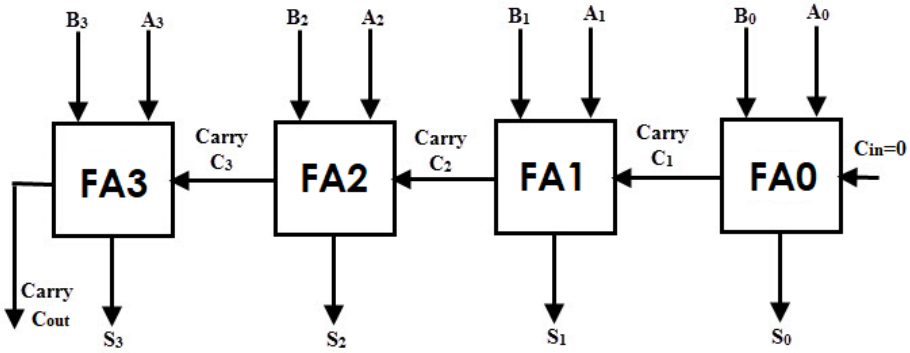
Case-1: 5 + 5 = (?)

Carry in		1	0	1	0
Augend (A_x)		0	1	0	1
Addend (B_x)	+	0	1	0	1
Carry out		0	1	0	1
Sum (S_x)		1	0	1	0

Case-2: 15 + 12 = (?)

Carry in		1	0	0	0
Augend (A_x)		1	1	1	1
Addend (B_x)	+	1	1	0	0
Carry out		1	1	0	0
Sum (S_x)		1	0	1	1

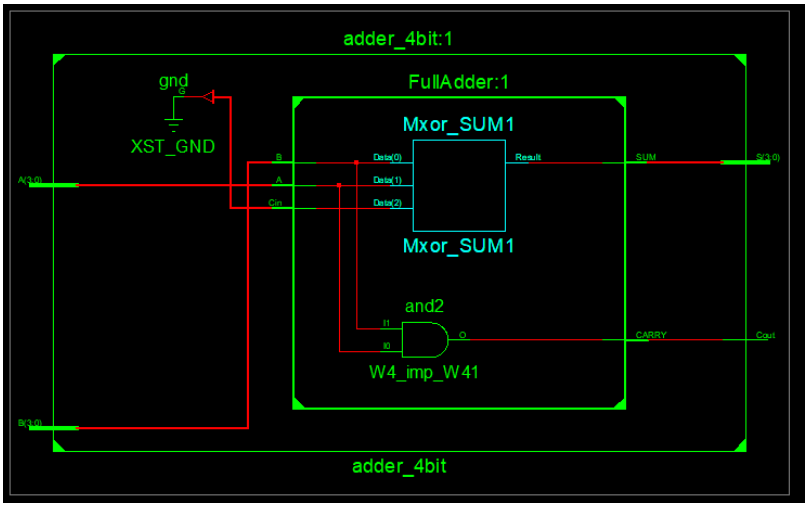
LOGIC DIAGRAM:



VERILOG HDL CODE – 4-BIT PARALLEL ADDER (Gate-level):

```
module adder_4bit (S, Cout,A,B);
input [3:0] A ;
input [3:0] B ;
output [3:0]S ;
output Cout ;
wire [3:1]C;
FullAdder FA0(S[0],C[1],A[0],B[0],1'b0);
FullAdder FA1(S[1],C[2],A[1],B[1],C[1]);
FullAdder FA2(S[2],C[3],A[2],B[2],C[2]);
FullAdder FA3(S[3],Cout,A[3],B[3],C[3]);
endmodule
```

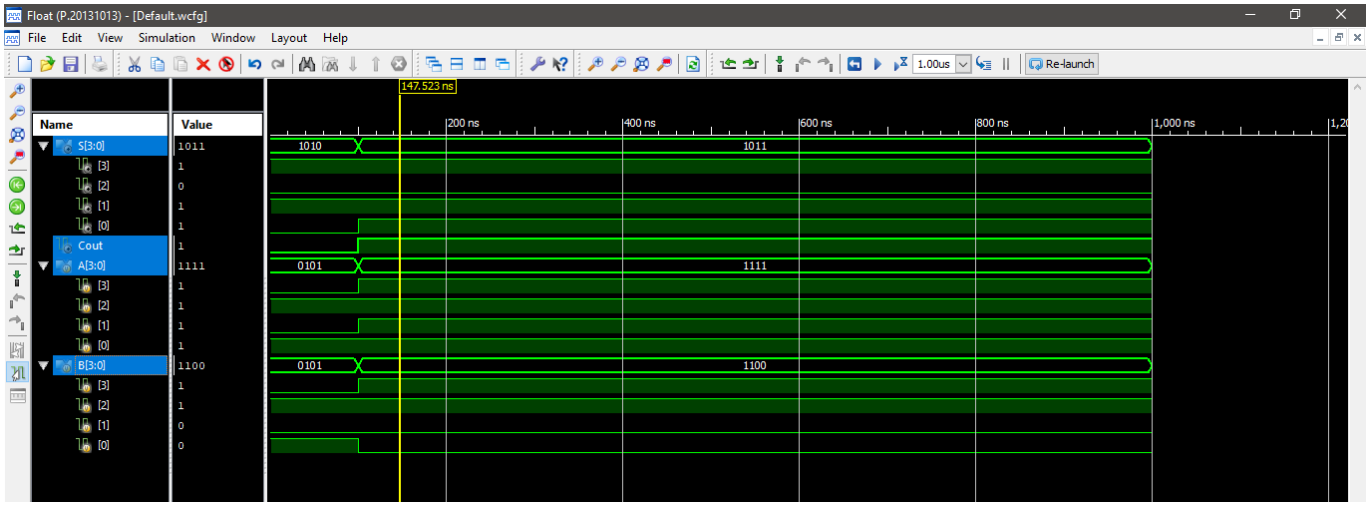
RTL SCHEMATIC:



TEST BENCH:

```
module adder_4bit_tb;
    // Inputs
    reg [3:0] A;
    reg [3:0] B;
    // Outputs
    wire [3:0] S;
    wire Cout;
    // Instantiate the Unit Under Test (UUT)
    adder_4bit uut (
        .S(S),
        .Cout(Cout),
        .A(A),
        .B(B)
    );
    initial begin
        // Initialize Inputs
        A = 5; B = 5; #100;
        A = 15; B = 12; #100;
    end
endmodule
```

SIMULATION OUTPUT:



RESULT:

Thus, 4-bit parallel adder is constructed using four full adder using instantiation and its logical functions is verified by simulation using Xilinx ISE Simulator.

VIVA QUESTIONS WITH ANSWERS:

1. **What is the main limitation of parallel adder?**
   Carry propagation delay

2. **What is test bench in Verilog HDL?**
   Test bench is another Verilog file used to verify the correctness or soundness of a design or model

3. **Why Cin of 4-bit parallel adder set to be 0?**
   For the LSB bit addition no carry bit is generated. So we assign Cin as 0.

4. **Number of full adder required to design 16-bit parallel adder is ___**
   16 full adder (or 15 full adder 1 half adder)

5. **What is Sum ouput expression for full adder?**
   SUM= A xor B xor C

6. **What is the purpose of test bench in Verilog HDL?**
   Test bench used to verify the functionality of the design under test by applying appropriate test pattern to the inputs and checks the outputs of the design.

7. **While writing test bench, original Verilog module inputs are reassigned as _____ and outputs are reassigned as _____.**
   Reg, wire

8. **If two 16-bit inputs are added using 16-bit parallel adder then size of the final sum output will be _____.**
   16-bit

9. **Write a syntax for module instantiation.**
   Module_name Instance_name (Port_Association_List)

10. **While using module instantiation, Instance name should be unique but the size and order of each port can be different. True or False?**
    False

Expt. No. 10	Design and Simulate 4x4 Binary Multiplier

AIM:

Multiplication is one of the essential operation to be carried out by ALU of a 4004 microprocessor. Design and simulate multiplier unit for 4-bit ALU unit with the help of basic logic gates and 4-bit parallel adders. Verify the output logic by simulation using Xilinx ISE Simulator.

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

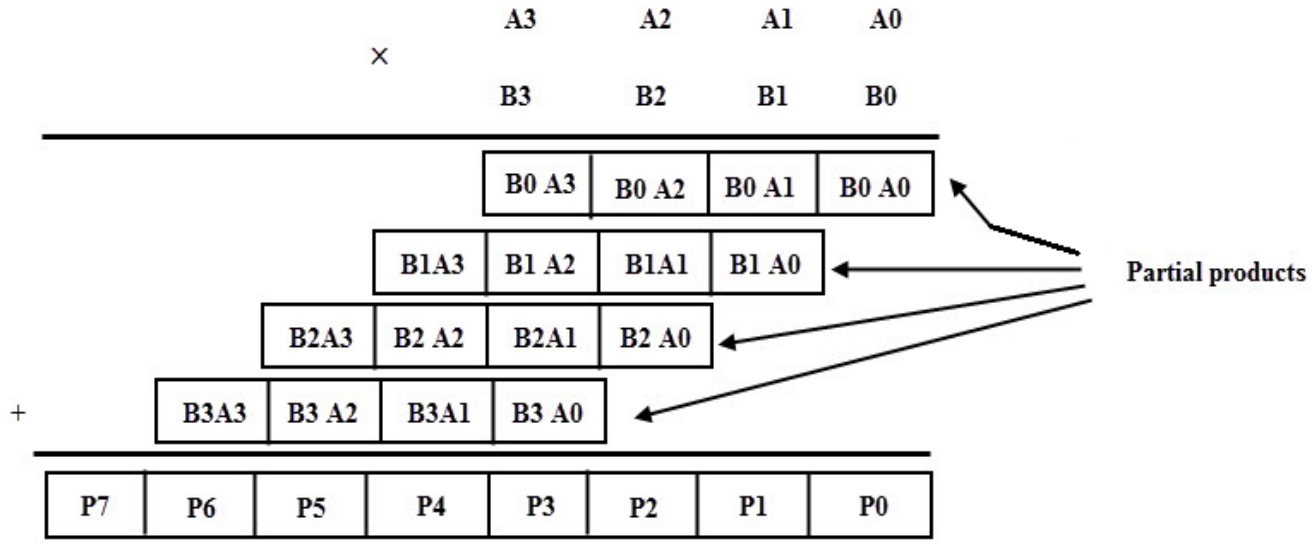
PROCEDURE:

1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the simulate behavioural model to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per truth table.

4 x 4 BINARY MULTIPLIER:

THEORY:

A binary multiplier is a combinational logic circuit used in digital systems to perform the multiplication of two binary numbers. These are most commonly used in various applications especially in the field of digital signal processing to perform the various algorithms. Similar to the multiplication of decimal numbers, binary multiplication follows the same process for producing a product result of the two binary numbers. Let us consider two unsigned 4 bit numbers multiplication in which the multiplicand, A is equal to A3 A2 A1 A0 and the multiplier B is equal to B3 B2 B1B0. The partial products are produced depending on each multiplier bit multiplied by the multiplicand. Each partial product consists of four product terms and these are shifted to the left relative to the previous partial product as shown in figure. All these partial products are added to produce the 8 bit product.



The logic circuit for the 4×4 binary multiplication can be implemented by using three 4-bit parallel adders along with AND gates. The first partial product is obtained by multiplying B0 with A3 A2 A1A0, the second partial product is formed by multiplying B1 with A3 A2 A1A0, likewise for 3rd and 4th partial products. So these partial products can be implemented with AND gates as shown in logic diagram. These partial products are then added by using 4-bit parallel adder. The three most significant bits of first partial product with carry (considered as zero) are added with second partial term in the first full adder. Then the result is added to the next partial product with carry out and it goes on until the final partial product, finally it produces 8-bit sum, which indicates the multiplication value of the two binary numbers.

TEST CASES:

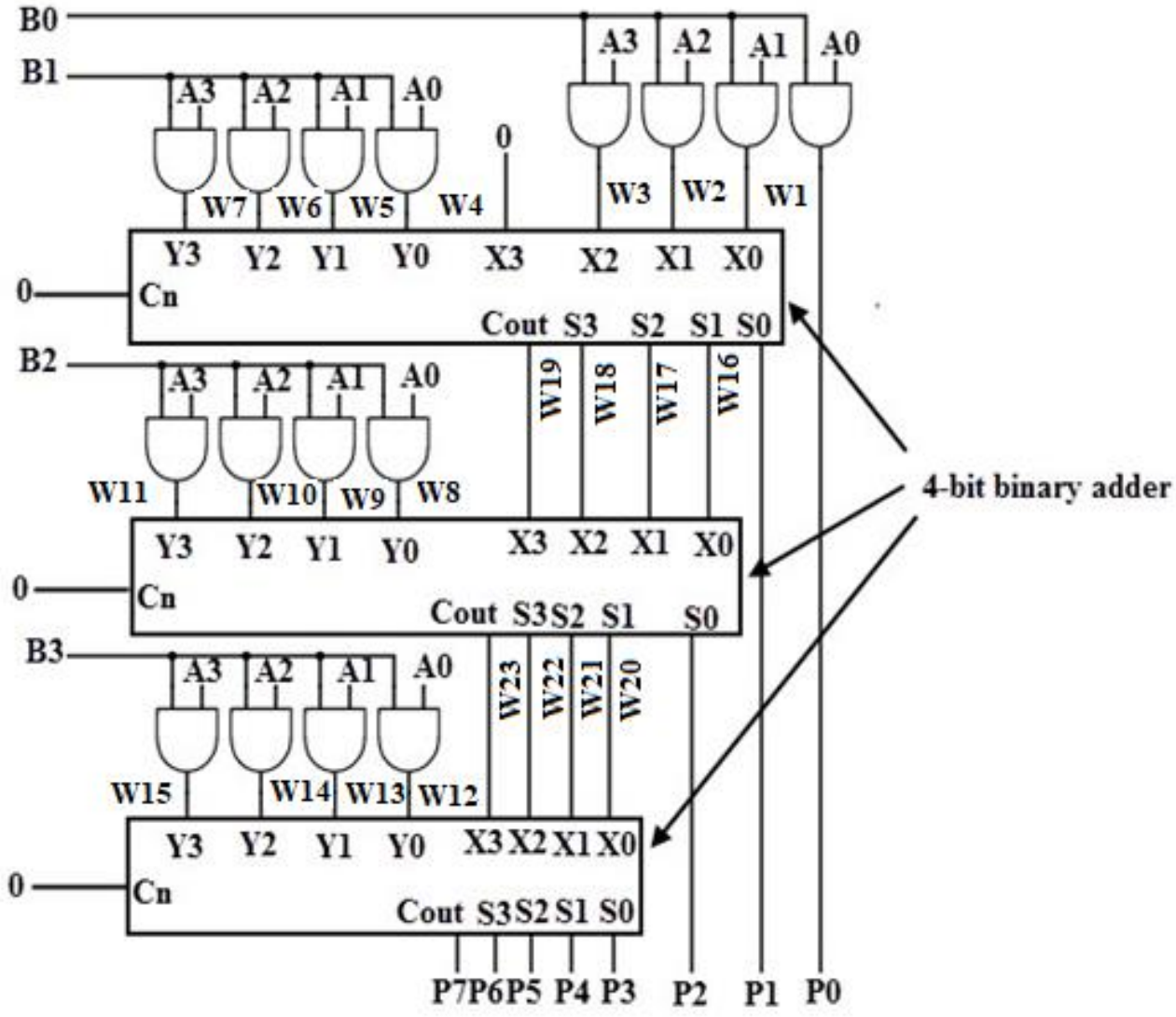
Case-1: $(10)_{10}$ x $(11)_{10}$ =(?)$_{10}$

```
    1 0 1 0   ──►  Multiplicand
  × 1 0 1 1   ──►  Multiplier
  ---------
    1 0 1 0   ──►  Partial product 1
  1 0 1 0     ──►  Partial product 2
  0 0 0 0     ──►  Partial product 3
1 0 1 0       ──►  Partial product 4
  ---------
  1 1 0 1 1 1 0
  ---------
```

Case-2: $(13)_{10}$ x $(11)_{10}$ =(?)$_{10}$

```
        1 1 0 1    (13)10   Multiplicand M
      × 1 0 1 1    (11)10   Multiplier Q
      -----------
        1 1 0 1
      1 1 0 1                Partial products
    0 0 0 0
  1 1 0 1
  -----------
1 0 0 0 1 1 1 1    (143)10  Product P
```

LOGIC DIAGRAM:



VERILOG HDL CODE – 4x4 BINARY MULTIPLIER (Gate-level):

```
//instantiate 4-bit adder and full adder module from previous experiment
module multiplier_4x4(P,A,B);
//inputs
input [3:0]A,B;
//outputs
output [7:0]P;

//wires
wire [23:1]W;

//andgate instantiations
and a1(P[0],A[0],B[0]);
and a2(W[1],A[1],B[0]);
and a3(W[2],A[2],B[0]);
and a4(W[3],A[3],B[0]);

and a5(W[4],A[0],B[1]);
and a6(W[5],A[1],B[1]);
and a7(W[6],A[2],B[1]);
and a8(W[7],A[3],B[1]);

and a9(W[8],A[0],B[2]);
and a10(W[9],A[1],B[2]);
and a11(W[10],A[2],B[2]);
and a12(W[11],A[3],B[2]);

and a13(W[12],A[0],B[3]);
and a14(W[13],A[1],B[3]);
and a15(W[14],A[2],B[3]);
and a16(W[15],A[3],B[3]);

//full adders instatiations
adder_4bit PA1({W[18],W[17],W[16],P[1]}, W[19], {1'b0,W[3],W[2],W[1]},
{W[7],W[6],W[5],W[4]});

adder_4bit PA2({W[22],W[21],W[20],P[2]},W[23], {W[19],W[18],W[17],W[16]},
{W[11],W[10],W[9], W[8]});

adder_4bit PA3({P[6],P[5],P[4],P[3]}, P[7], {W[23],W[22],W[21],W[20]},
{W[15],W[14],W[13],W[12]});

endmodule
```

TEST BENCH FOR 4x4 BINARY MULTIPLIER:

```
module multiplier_4x4_tb;
    reg [3:0] A;
    reg [3:0] B;
    wire [7:0] P;
    multiplier_4x4 uut (.P(P), .A(A),  .B(B));
    initial begin
        A = 13; B = 11; #100;
        A = 10; B = 11; #100;
    end
```

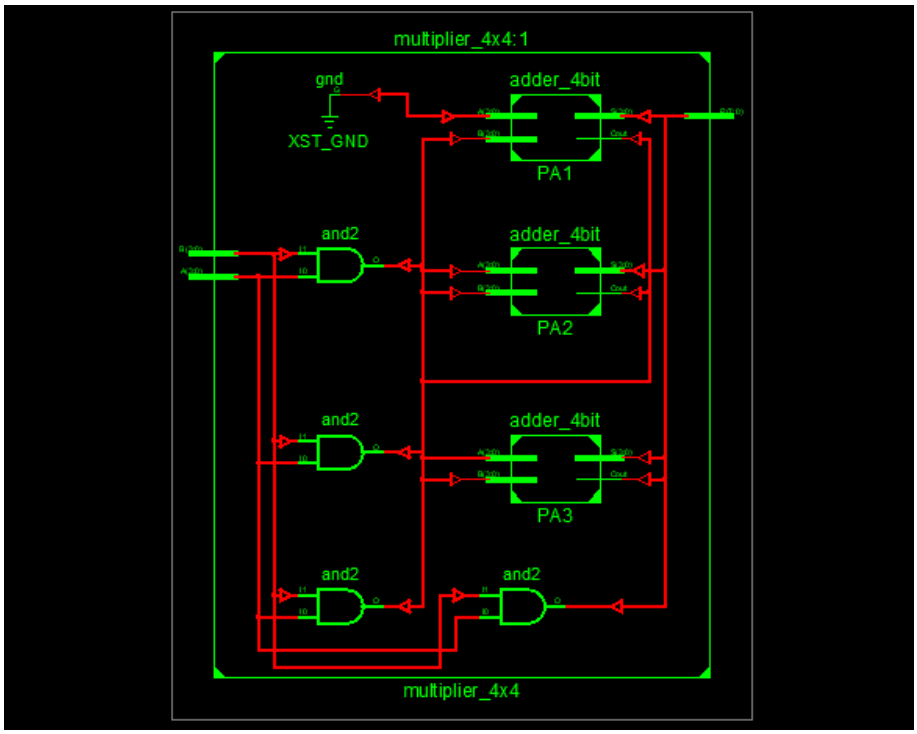
VERILOG HDL CODE – 4-BIT PARALLEL ADDER (Gate-level):

```
// Module Name:    adder_4bit
module adder_4bit (S, Cout,A,B);
input [3:0] A ;
input [3:0] B ;
output [3:0]S ;
output Cout ;
wire [2:1]C;
FullAdder FA0(S[0],C[1],A[0],B[0],1'b0);
FullAdder FA1(S[1],C[2],A[1],B[1],C[1]);
FullAdder FA2(S[2],C[3],A[2],B[2],C[2]);
FullAdder FA3(S[3],Cout,A[3],B[3],C[3]);
endmodule
```

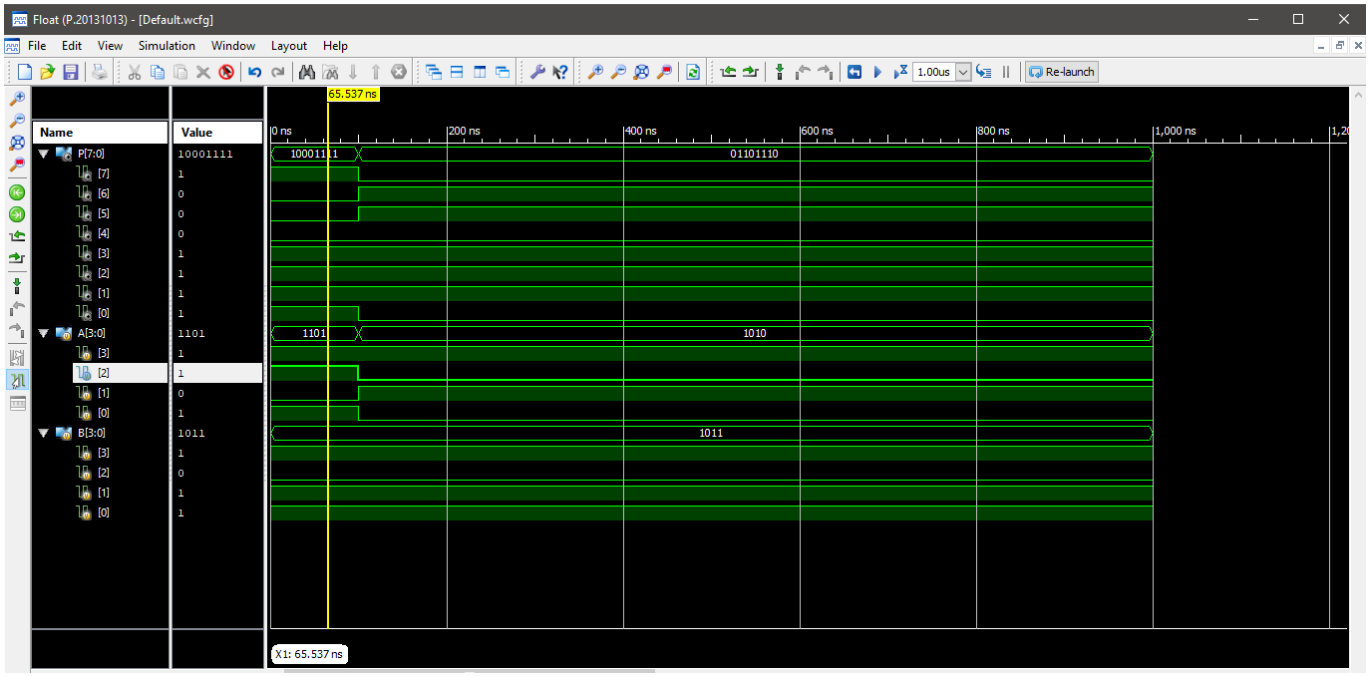
VERILOG HDL CODE - FULL ADDER (Data-Flow):

```
// Module Name:    FullAdder
module FullAdder(SUM, CARRY, A, B, Cin);
input A,B,Cin;
output SUM, CARRY;
assign SUM = A ^ B ^ Cin;
assign CARRY = (A&B) | (B&Cin) | (Cin&A);
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, multiplication unit of 4-bit ALU is designed using basic logic gates and 4-bit parallel adder and its logical operation is verified by simulation using Xilinx ISE Simulator.

VIVA QUESTIONS WITH ANSWERS:

1. **For an M x N binary multiplier how many AND gates and 4-bit adders are needed?**
   (M x N)AND gates and (N-1) 4-bit adders

2. **The result obtained on binary multiplication of 1010 * 1100 is _____.**
   1111000

3. **For 4x3 binary multiplier how many AND gates and 4-bitadders are needed?**
   12 AND gates and 3 4-bit adders

4. **Number of partial product resulted in 4x2 multiplication.**
   2

5. **What is the use of AND gate in binary multiplier?**
   Used to compute individual bits multiplication in partial product result

6. **Number of bits required to represent the final product output of 5x4 multiplier.**
   9

7. **What is the purpose 4-bit adder in binary multiplier?**
   Used to add all partial products to generate a final product output

8. **Application of multipliers?**
   - Most commonly used in digital signal processing to perform the various algorithms.
   - Commercial applications like computers, mobiles, high-speed calculators and some general purpose processors also require binary multipliers.

9. **Which operator is used in Verilog HDL to combine multiple bits into single number?**
   Concatenate operator { }

10. **How to represent decimal number 12 as a binary number in Verilog HDL?**
    4'b1100

Expt. No. 11	Design and Simulate SR, JK, D and T Flip-Flops

AIM:

(a). Construct SR Flip-Flops using basic logic gates and verify its output logic by simulation using Xilinx ISE Simulator.

(b). Construct JK Flip-Flops using basic logic gates and verify its output logic by simulation using Xilinx ISE Simulator.

(c). In a Microprocessor design, signals that flow through different paths arrive at different time. This could cause many problems when these signals have to interact with each other. Identify & implement a suitable flop-flop required to synchronize these signals using basic logic gates. Verify its output logic by simulation using Xilinx ISE Simulator. (Hint: D-FF)

(d). Counters are the digital circuits, which are used to count the number of events. Identify and implement the most suitable Flip-flip required to design a counter unit using basic logic gates. Verify its output logic by simulation using Xilinx ISE Simulator. (Hint: T-FF)

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

PROCEDURE:

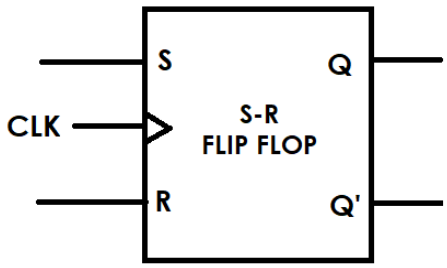
1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the simulate behavioural model to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per truth table.

(a). SR(SET RESET) Flip-Flop:

THEORY:

The SR flip-flop, stands for "Set-Reset" flip-flop. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will "SET" the device (meaning the output = "1"), and is labelled S and one which will "RESET" the device (meaning the output = "0"), labelled R. The reset input resets the flip-flop back to its original state with an output Q. A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it's current state or history. When S and R at HIGH state both outputs tries to get into HIGH state and not of them get into state output state. This state is called intermediate or invalid state.

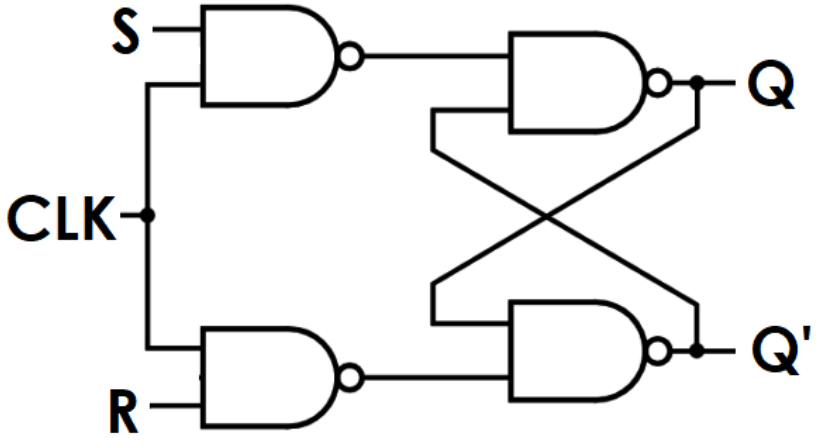
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUTS		PRESENT STATE	NEXT STATE	STATE
	S	R	Q_n	Q_{n+1}	
↑	0 0	0 0	0 1	0 1	NO CHANGE (NC)
↑	0 0	1 1	0 1	0 0	RESET (R)
↑	1 1	0 0	0 1	1 1	SET (S)
↑	1 1	1 1	0 1	x x	INDETERMINATE (*)
↓	x x	x x	0 1	0 1	NO CHANGE (NC)

LOGIC DIAGRAM:



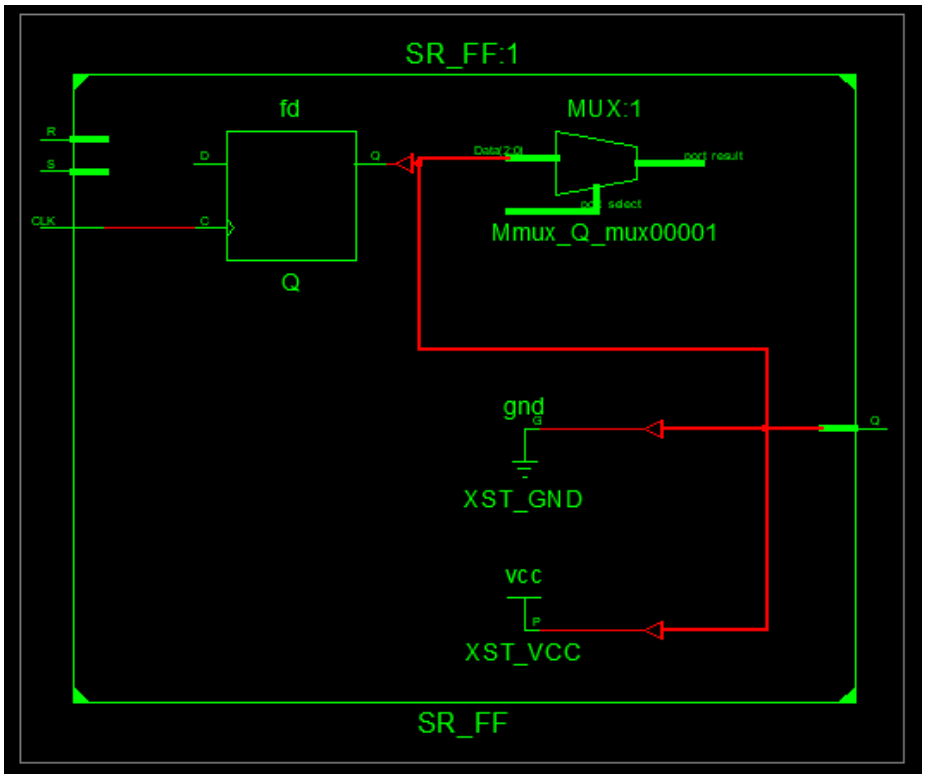
VERILOG HDL CODE (Behavioural):

```
module SR_FF(Q,QB,S,R,CLK);
input S,R,CLK;
output Q,QB;
reg Q,QB;
always @(posedge CLK)
begin
case({S,R})
2'b00:Q=Q;
2'b01:Q=0;
2'b10:Q=1;
2'b11:Q=1'bx;
endcase
QB=~Q;
end
endmodule
```

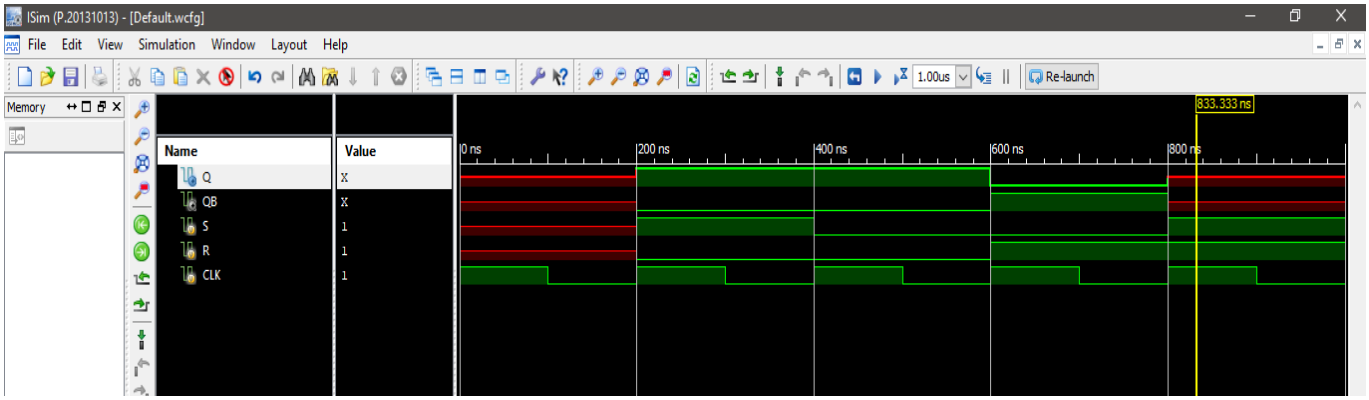
TEST BENCH:

```
module SR_FF_TB;
	// Inputs
	reg S;
	reg R;
	reg CLK;
	// Outputs
	wire Q;
	wire QB;
	// Instantiate the Unit Under Test (UUT)
	SR_FF uut (
		.Q(Q),
		.QB(QB),
		.S(S),
		.R(R),
		.CLK(CLK)
	);

	always #100 CLK=~CLK;
	initial begin
		// Initialize Inputs
		CLK=1;
		#200 S=1; R=0;
		#200 S=0; R=0;
		#200 S=0; R=1;
		#200 S=1; R=1;
	end
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:

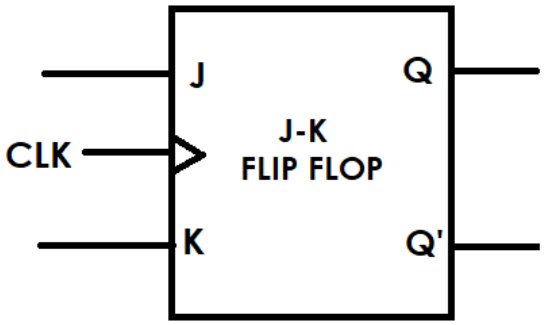


(b). JK (JACK KILBY) FLIP-FLOP:

THEORY:

The JK Flip Flop is the most widely used flip flop. It is considered a universal flip-flop circuit. The sequential operation of the JK Flip Flop is same as for the RS flip-flop with the same SET and RESET input. The difference is that the JK Flip Flop does not the invalid input states when S and R are both 1. A JK Flip-Flop can be obtained from the clocked SR Flip-Flop by augmenting two AND gates. If the circuit is in the "SET" condition, the J input is inhibited by the status 0 of Q through the lower NAND gate. Similarly, the input K is inhibited by 0 status of Q through the upper NAND gate in the "RESET" condition. When both J and K are at logic "1", the JK Flip Flop toggle.

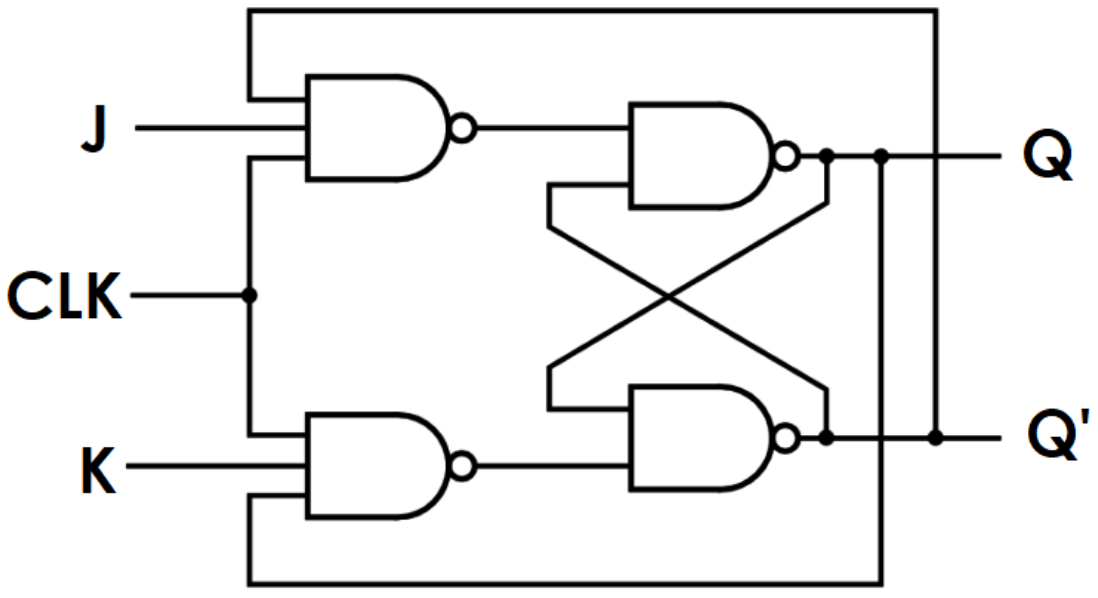
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUTS		PRESENT STATE	NEXT STATE	STATE
	J	K	Q_n	Q_{n+1}	
↑	0 0	0 0	0 1	0 1	NO CHANGE (NC)
↑	0 0	1 1	0 1	0 0	RESET (R)
↑	1 1	0 0	0 1	1 1	SET (S)
↑	1 1	1 1	0 1	1 0	TOGGLE (T)
↓	x x	x x	0 1	0 1	NO CHANGE (NC)

LOGIC DIAGRAM:



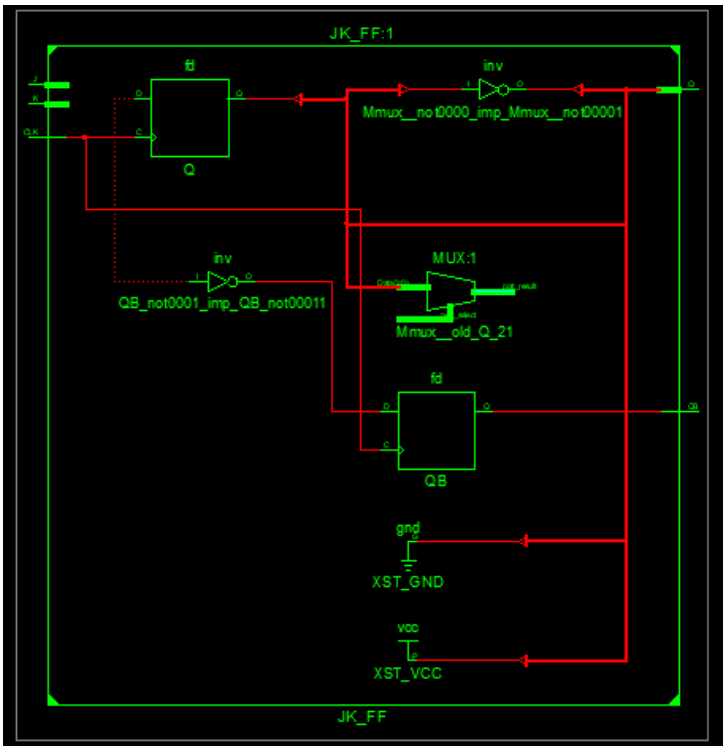
VERILOG HDL CODE:

```
module JK_FF(Q,QB,J,K,CLK);
input J,K,CLK;
output Q,QB;
reg Q,QB;
always @(posedge CLK)
begin
case({J,K})
2'b00:Q=Q;
2'b01:Q=0;
2'b10:Q=1;
2'b11:Q=~Q;
endcase
QB=~Q;
end
endmodule
```

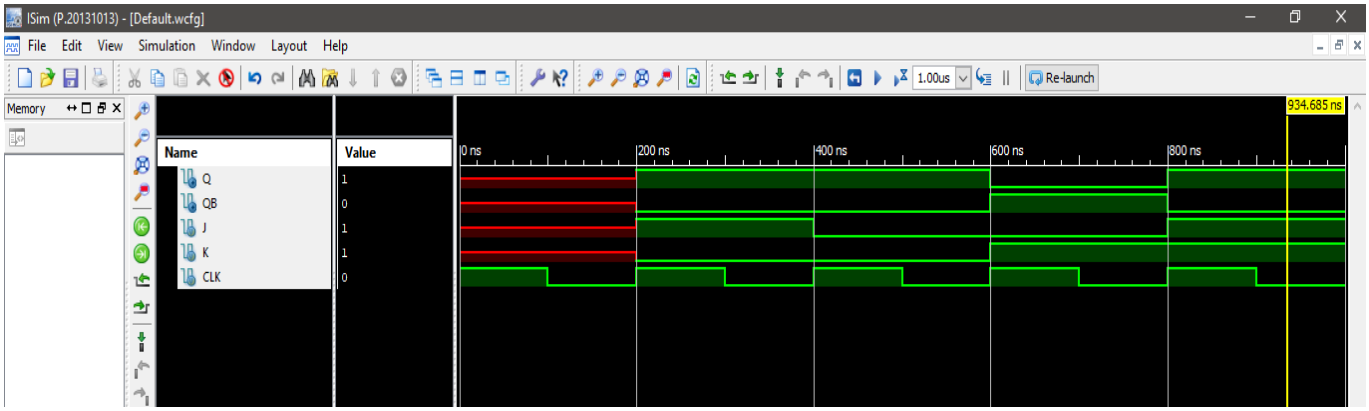
TEST BENCH:

```
module JK_FF_TB;
	// Inputs
	reg J;
	reg K;
	reg CLK;
	// Outputs
	wire Q;
	wire QB;
	// Instantiate the Unit Under Test (UUT)
	JK_FF uut (
		.Q(Q),
		.QB(QB),
		.J(J),
		.K(K),
		.CLK(CLK)
	);
always #100 CLK=~CLK;
	initial begin
		// Initialize Inputs
		CLK=1;
		#200 J=1;K=0;
		#200 J=0; K=0;
		#200 J=0; K=1;
		#200 J=1; K=1;
	end
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:

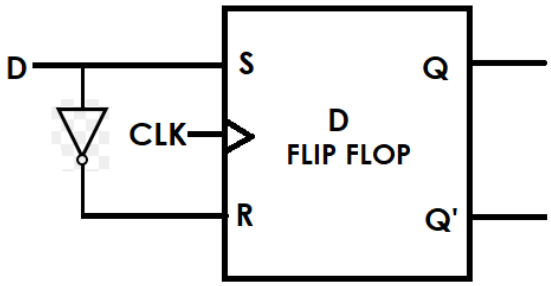


(c). D (DATA/DELAY) FLIP-FLOP:

THEORY:

This flip-flop, called a Data flip-flop because of its ability to 'latch' and remember data, or a Delay flip-flop because latching and remembering data can be used to create a delay in the progress of that data through a circuit. A D flip–flop is constructed by modifying an SR flip – flop. The S input is given with D input and the R input is given with inverted D input. Hence, there will be no chance of any intermediate state occurs. The major drawback of SR flip – flop is the race around condition which in D flip – flop is eliminated. When we don't apply any clock input to the D flip flop or during the falling edge of the clock signal, there will be no change in the output. It will retain its previous value at the output Q. If the clock signal is high (rising edge to be more precise) and if D input is high, then the output is also high and if D input is low, then the output will become low. Hence the output Q follows the input D in the presence of clock signal.

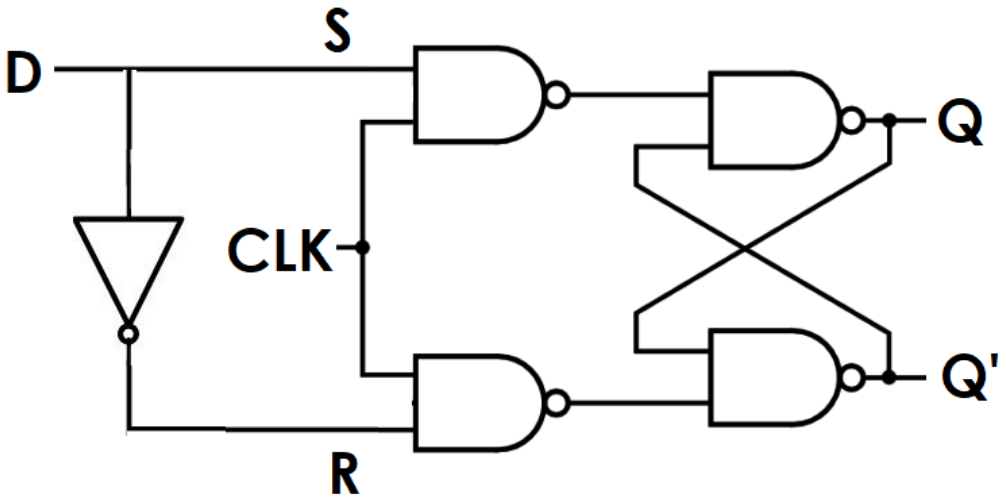
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUT	PRESENT STATE	NEXT STATE	STATE
	D	Q_n	Q_{n+1}	
↑	0	0	0	RESET
↑	0	1	0	
↑	1	0	1	SET
↑	1	1	1	
↓	X	Q_n	Q_n	NO CHANGE

LOGIC DIAGRAM:



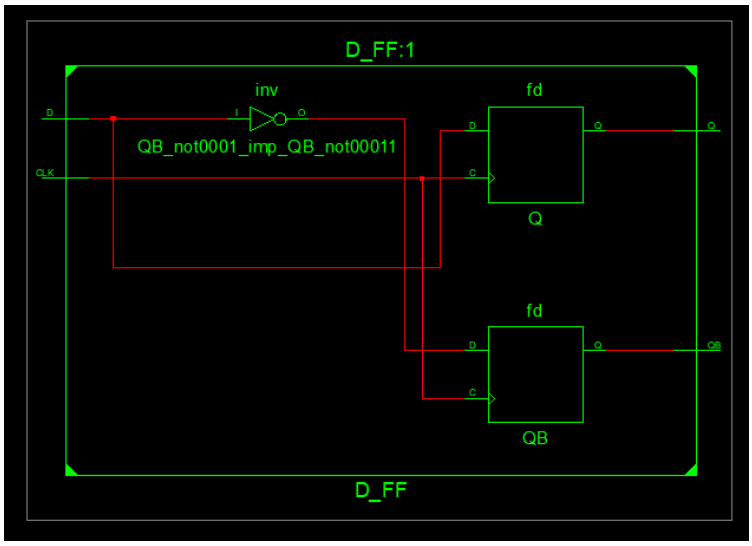
VERILOG HDL CODE:

```
module D_FF(Q,QB,D,CLK);
input D,CLK;
output Q,QB;
reg Q,QB;
always @(posedge CLK)
begin
 Q=D;
 QB=~Q;
 end
endmodule
```

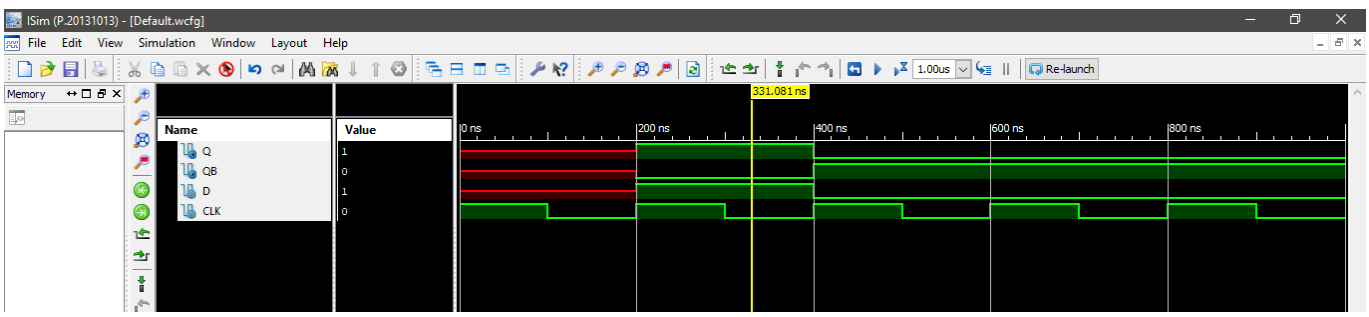
TEST BENCH:

```
module D_FF_TB;
    reg D;
    reg CLK;
    wire Q;
    wire QB;
    D_FF uut (
        .Q(Q),
        .QB(QB),
        .D(D),
        .CLK(CLK)
    );
    always #100 CLK=~CLK;
    initial begin
        // Initialize Inputs
        CLK=1;
        #200 D=1;
        #200 D=0;
        end
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:

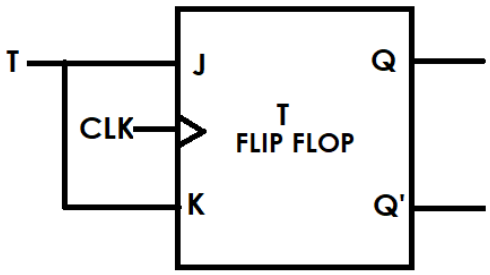


(d). T (TOGGLE) FLIP-FLOP:

THEORY:

T flip – flop is also known as "Toggle Flip – flop". To avoid the occurrence of intermediate state in SR flip – flop, we should provide only one input to the flip – flop called Trigger input or Toggle input (T). Then the flip – flop acts as a Toggle switch. Toggling means 'Changing the next state output to complement of the present state output'. The T (Toggle) Flip-Flop is a modification of the JK Flip-Flop. It is obtained from JK Flip-Flop by connecting both inputs J and K together, i.e., single input. Regardless of the present state, the Flip-Flop complements its output when the clock pulse occurs while input T= 1.

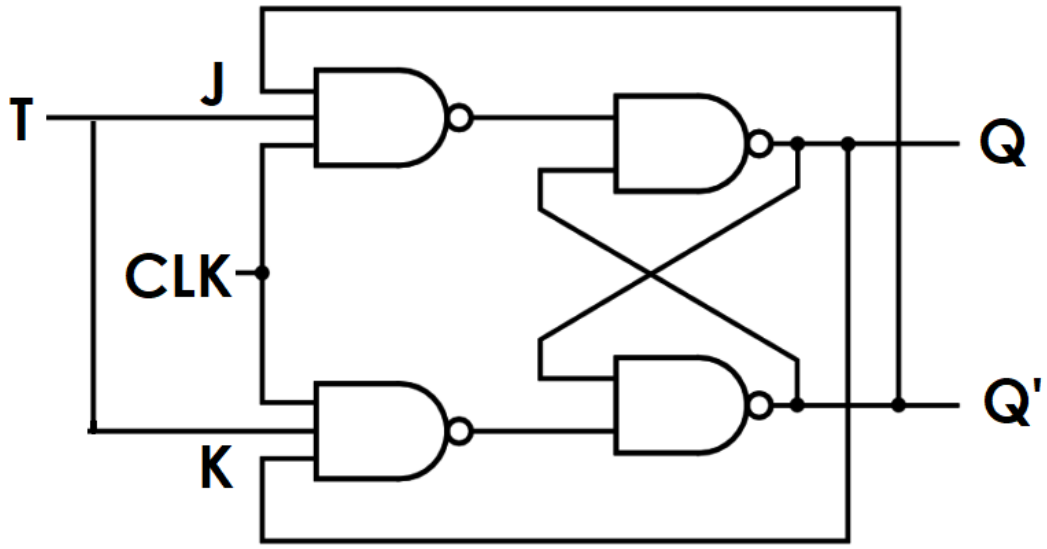
BLOCK DIAGRAM:



TRUTH TABLE:

CLK	INPUT	PRESENT STATE	NEXT STATE	STATE
	T	Q_n	Q_{n+1}	
↑	0	0	0	NO CHANGE
↑	0	1	1	
↑	1	0	1	TOGGLE
↑	1	1	0	
↓	X	Q_n	Q_n	NO CHANGE

LOGIC DIAGRAM:



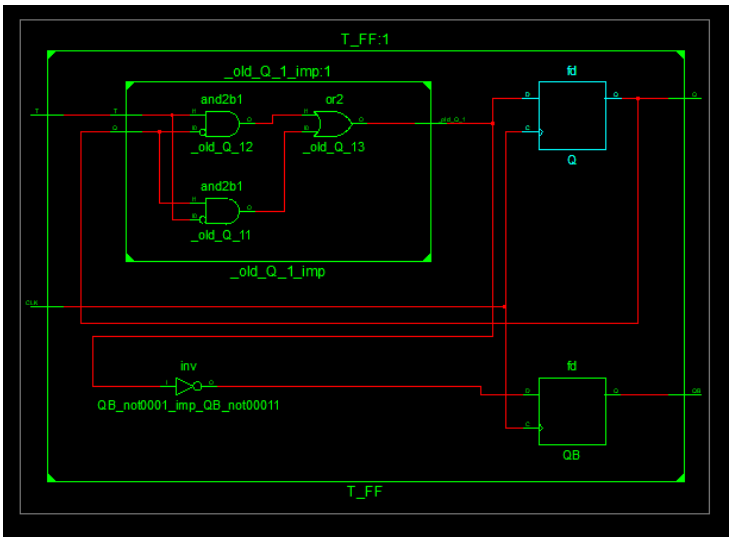
VERILOG HDL CODE:

```
module T_FF(Q,QB,T,CLK);
input T,CLK;
output Q,QB;
reg Q=0,QB;
always @(posedge CLK)
begin
case(T)
1'b0:Q=Q;
1'b1:Q=~Q;
endcase
QB=~Q;
end
endmodule
```

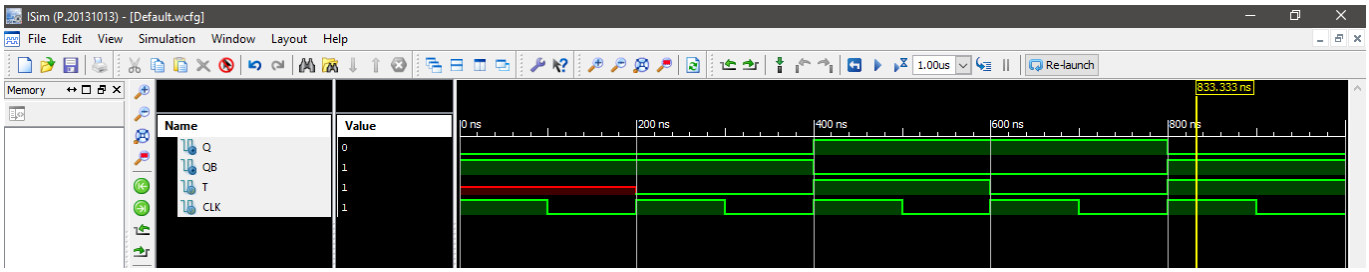
TEST BENCH:

```
module T_FF_TB;
        // Inputs
        reg T;
        reg CLK;
        // Outputs
        wire Q;
        wire QB;
        // Instantiate the Unit Under Test (UUT)
        T_FF uut (
                .Q(Q),
                .QB(QB),
                .T(T),
                .CLK(CLK)
        );
always #100 CLK=~CLK;
        initial begin
                // Initialize Inputs
                CLK=1;
                #200 T=0;
                #200 T=1;
                #200 T=0;
                #200 T=1;
                end
    endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, various types of flip-flops such as SR, JK, D and T are designed under Verilog HDL in behavioural modelling and its logical operation is verified by simulation using Xilinx ISE Simulator.

VIVA QUESTIONS WITH ANSWERS:

1. **What is sequential circuits?**
   The logic circuits whose outputs at any instant of time depends only on the present input but also on the past outputs are called Sequential circuits

2. **A basic S-R flip-flop can be constructed by cross-coupling of which basic logic gates?**
   NAND and NOR

3. **What is the main drawback of S-R flip-flop?**
   The main drawback of s-r flip-flop is invalid output when both the inputs are high, which is referred to as Invalid or intermediate State.

4. **Which flip-flop is called universal flip?**
   JK Flip flop. It can be used to realize other Flip-flops such as SR, D and T.

5. **When and which is a flip-flop said to be transparent?**
   D Flip flop is called transparent flip-flip since Q output follows the input

6. **On a positive edge-triggered flip-flop circuit, the outputs reflect the input condition only when?**
   The clock pulse transitions from LOW to HIGH

7. **Which type of circuit is faster, combinational or sequential? Why?**
   Combinational circuits are often faster than sequential circuits. Since, the combinational circuits do not require memory elements whereas the sequential circuits need memory devices to perform their operations in sequence.

8. **In T-flip flop toggle operation is performed only when T=?**
   1

9. **How T-flip flop is constructed using JK Flip-flop?**
   By shorting J and K input as a single input T.

10. **List atleast three applications of flip-flops.**
    - Data storage device
    - To design counters
    - Used in frequency divider circuits
    - Used as registers in Microprocessors/ microcontrollers
    - For data transfer applications

Expt. No. 12	Design and Simulate SIPO Shift Register & Up/Down Counters

AIM:

(a). In many communication systems, Serial data transmission is preferred for long distance communication due to its economical value in terms of the wires used. This necessitates parallel-to-serial conversion at the sender-end for which shift registers can be used. Design and implement the suitable 4-bit parallel to serial converter logic circuit for the given requirement using appropriate flip-flop. (Hint: PISO)

(b). A Microcontroller based bidirectional visitor counter is shown in figure-1 to count the number of visitors to the college library. Depending upon the signal from the IR sensors, this system detects the entry and exit of the visitor. Figure-2 shows sensor setup at the library entrance for bidirectional visitor counter. The logic control circuit block shown in figure – 1 keeps the record on of number of visitor entered and number of visitor exited. Microcontroller calculate number of person present in the library by subtracting number of people exited from number of people entered. Then, it displays the number of visitors present in the library on the display device-using microcontroller. For simplicity assume the system can count maximum of 16 people.

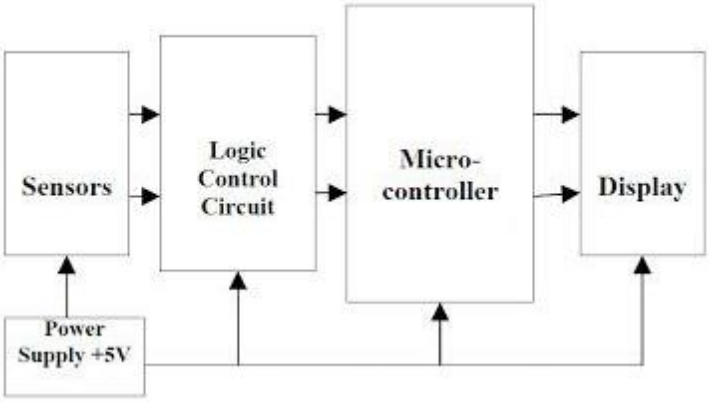


Figure-1: Block diagram of Bi-directional visitor counter

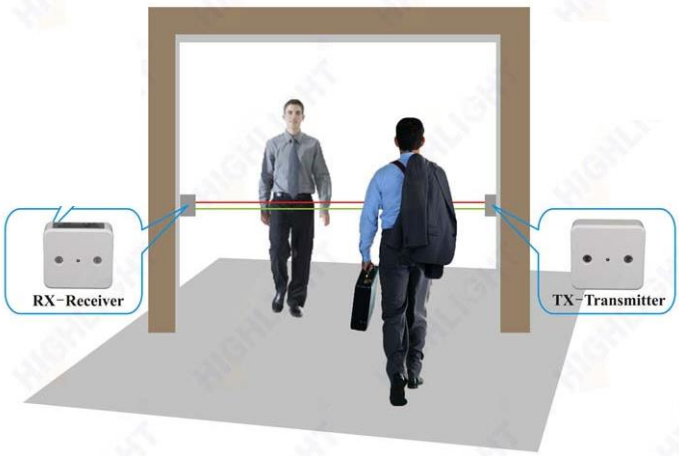


Figure-2 Illustration of sensor setup of Bi-directional Visitor counter

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

PROCEDURE:

1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the simulate behavioural model to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per truth table.

(a). PARALLEL-IN SERIAL-OUT (PISO):

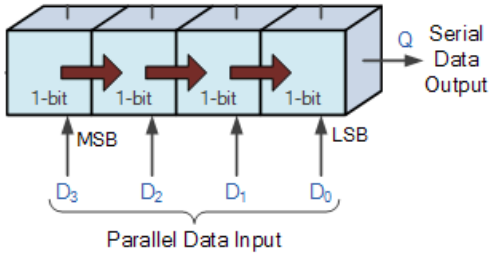
THEORY:

The parallel in-serial out shift register receives the data input in parallel batches on every clock pulse and the data is shifted and output serially. The data is loaded into the register in a parallel format in which all the data bits enter their inputs simultaneously, to the parallel input pins D0 to D3 of the register. The data is then read out sequentially in the normal shift-right mode from the register at Q representing the data present at D0 to D3. This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this type of data register a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data. This type registers used to perform parallel to serial data conversion in data communication systems.

There are four data-input lines, D0,D1,D2,D3 and a SHIFT / LOAD' input, which allows four bits of data to load-in parallel into the register. When SHIFT / LOAD is LOW, gates G1through G4 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with D = 1 will set and those with D = 0 will reset, thereby storing all four bits simultaneously.

When SHIFT / LOAD' is HIGH, gates G1 through G4 are disabled and gates G5 through G7 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the SHIFT / LOAD' input. Notice that FF3 has a single AND to disable the parallel input, D3. It does not require an AND/OR arrangement because there is no serial data in.

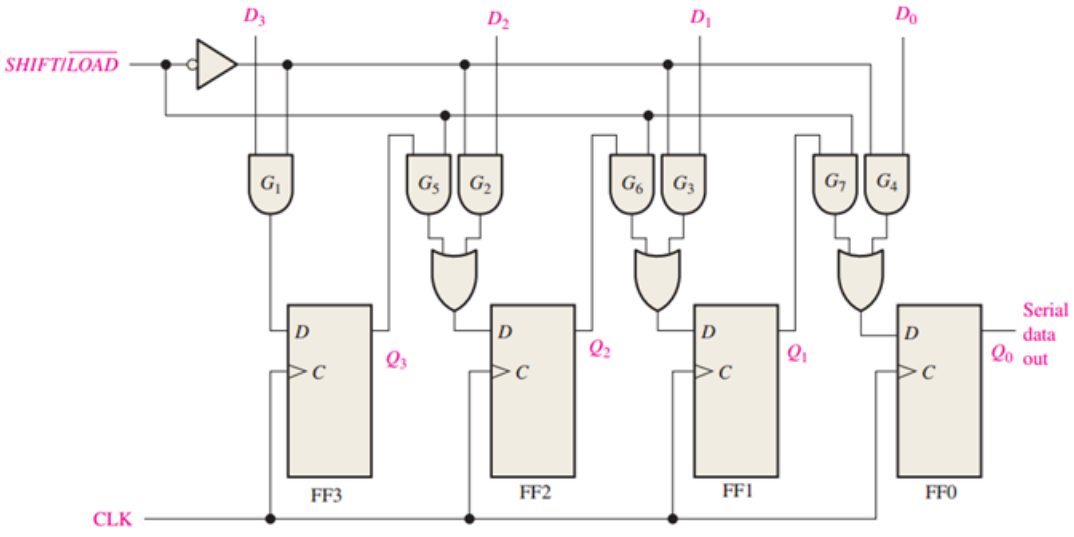
BLOCK DIAGRAM:



TRUTH TABLE:

CLOCK PULSE NO.	SHIFT/ LOAD'	PARALLEL DATA INPUT/FLIP-FLOP STATE				SERIAL DATA OUTPUT (Q)
		D3/Q3	D2/Q2	D1/Q1	D0/Q0	
1	0	1	1	0	1	0
2	1	1	1	0	1	1
3	1	0	1	1	0	0
4	1	0	0	1	1	1
5	1	0	0	0	1	1
6	1	0	0	0	0	0
7	1	0	0	0	0	0
8	0	1	0	0	1	0
9	1	1	0	0	1	1
10	1	0	1	0	0	0
11	1	0	0	1	0	0
12	1	0	0	0	1	1
13	1	0	0	0	0	0

LOGIC DIAGRAM:



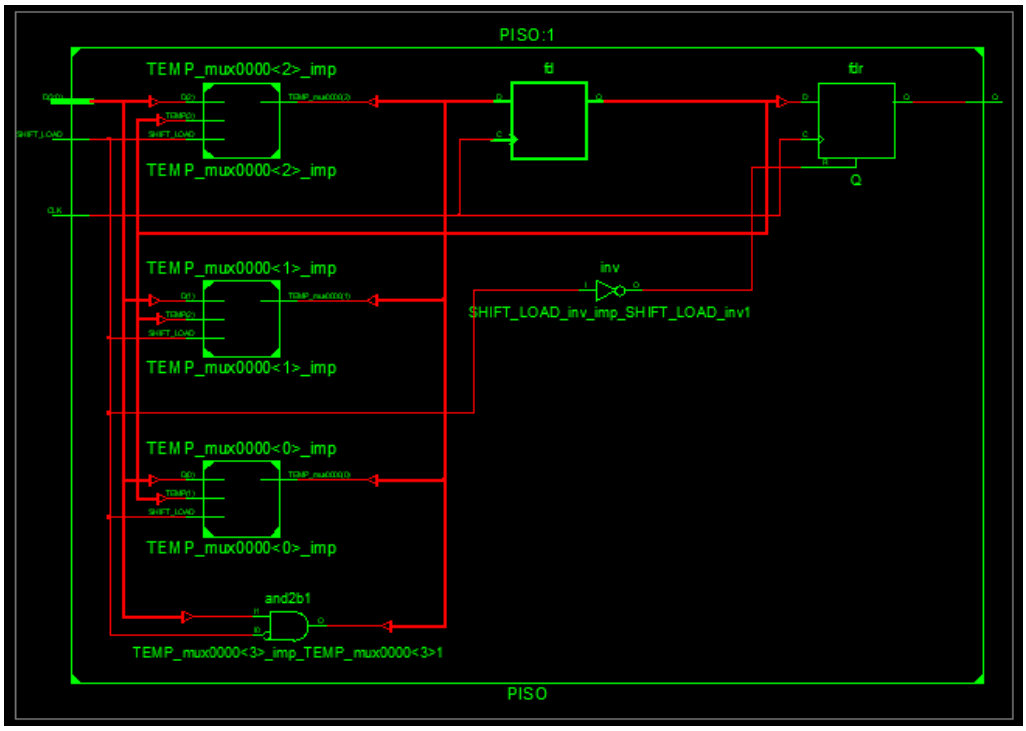
VERILOG HDL CODE (Behavioural):

```
module PISO(CLK,D,Q,SHIFT_LOAD);
        input CLK, SHIFT_LOAD;
        input [3:0]D;
        output Q;
        reg Q;
        reg [3:0]TEMP;
        always@(posedge CLK)
        begin
        if(SHIFT_LOAD==1'b0)
                begin
                Q<=1'b0;
                TEMP<=D;
                end
        else
                begin
                Q<=TEMP[0];
                TEMP<= TEMP>>1'b1;
                end
        end
        endmodule
```

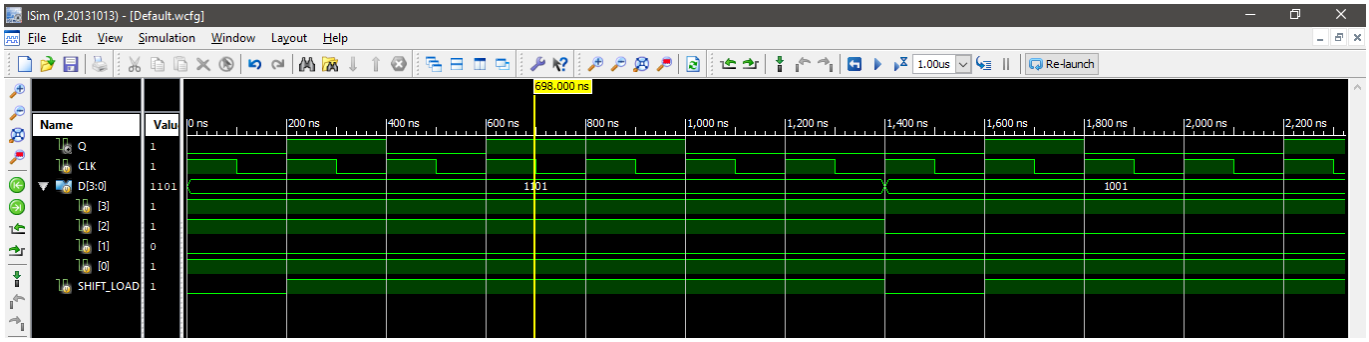
TEST BENCH:

```
        module PISO_tb;
                reg CLK, SHIFT_LOAD;
                reg [3:0] D;
                wire Q;
                PISO uut (.CLK(CLK),
                        .D(D),
                        .Q(Q),
                        .SHIFT_LOAD(SHIFT_LOAD) );
        initial CLK=1'b1;
        always #100 CLK=~CLK;
                initial begin
                        D=4'b1101;  SHIFT_LOAD = 1'b0;
                        #200 ;
                        SHIFT_LOAD = 1'b1;
                        #1200;
                        D=4'b1001; SHIFT_LOAD = 1'b0;
                        #200 ;
                        SHIFT_LOAD = 1'b1;
                        #1000 $stop;
        end
        endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



(b). UP/DOWN COUNTER:

THEORY:

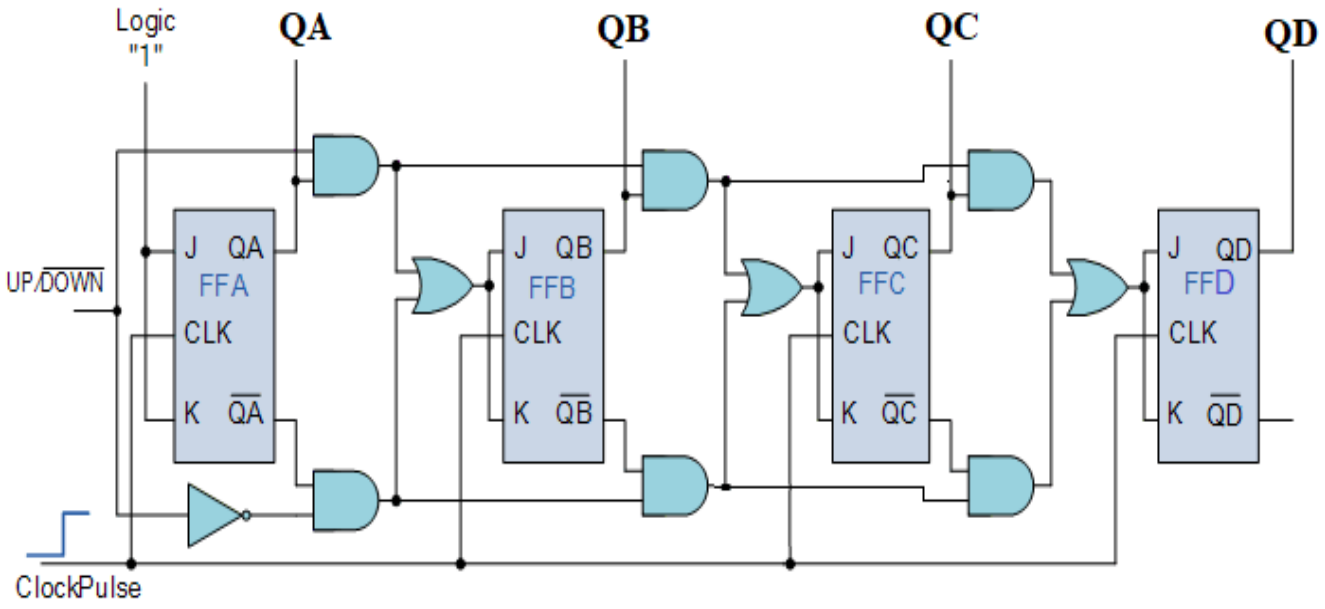
A counter is basically used to count the number of clock pulses applied to a flip-flop. It can also be used for Frequency divider, time measurement, frequency measurement, distance measurement and also for generating square waveforms. Counters are digital circuits made using flip-flops. There are two types of counters Synchronous and Asynchronous. Synchronous counter, as the name suggests have all the flip-flops working in sync with clock pulse as well as each other. Here clock pulse is applied to every flip-flop. Whereas in Asynchronous counter clock pulse is applied only to the initial flip flop whose value would be considered as LSB. Instead of the clock pulse, the output of first flip-flop acts as a clock pulse to the next flip flop, whose output is used as a clock to the next in line flip-flop and so on. This counter is also known as ripple counter since the clock pulse ripples through the flip-flops.

The circuit shown in logic diagram section is a simple 4-bit Up/Down synchronous counter using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero (0000) to seven (1111) and back to zero again. Then the 3-Bit counter advances upward in sequence (0 to 15) or downwards in reverse sequence (15 to 0). Generally, most bidirectional counter chips can be made to change their count direction either up or down at any point within their counting sequence. This is achieved by using an additional input pin, which determines the direction of the count, either Up or Down.

TRUTH TABLE:

CLOCK PULSE NO.	COUNTER OUTPUT AT Q			
	QD	QC	QB	QA
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

LOGIC DIAGRAM:



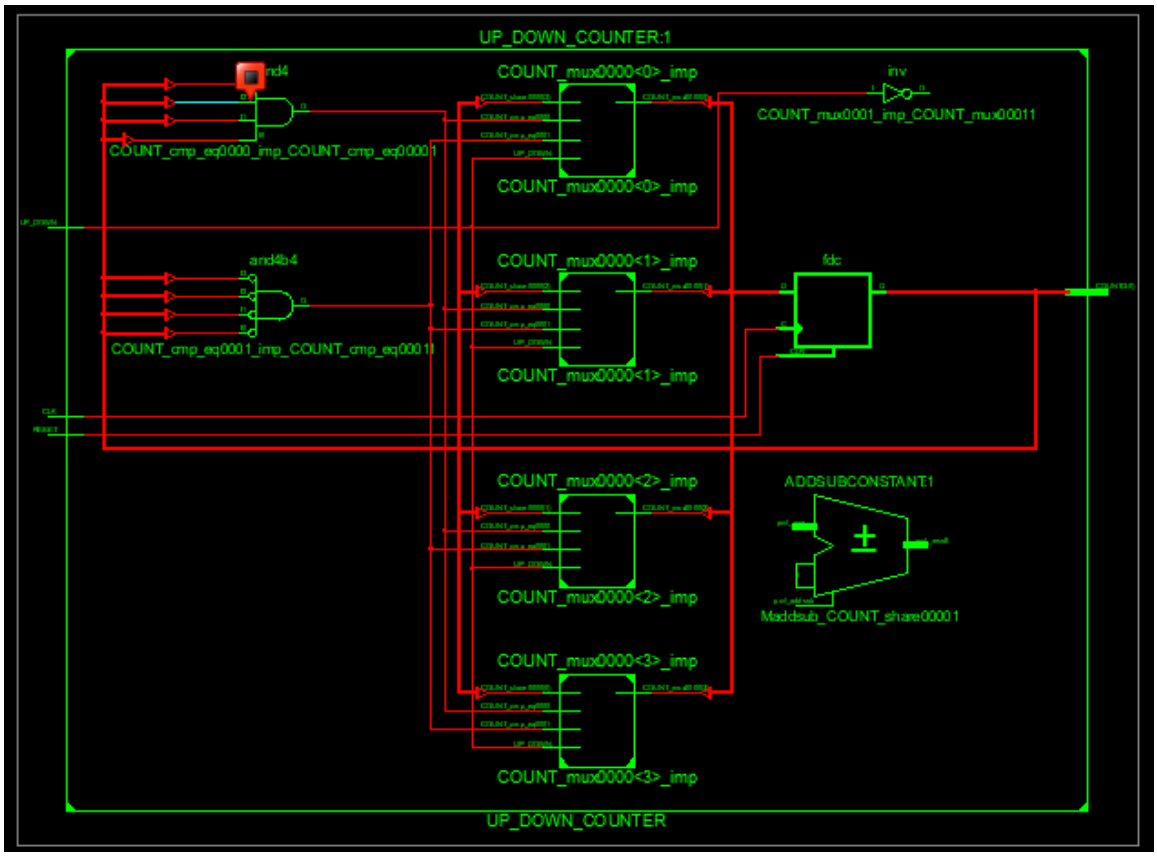
VERILOG HDL CODE (Behavioural):

```
module UP_DOWN_COUNTER(COUNT, CLK, UP_DOWN, RESET);
   input CLK, UP_DOWN, RESET;
   output [3 : 0] COUNT;
      reg [3:0] COUNT = 0;
     always @(posedge CLK or posedge RESET)
    begin
      if(RESET==1)
         COUNT<=0;
      else
         if(UP_DOWN == 1)   //Up mode selected
            if(COUNT == 15)
               COUNT <= 0;
            else
               COUNT <= COUNT + 1; //Incremend Counter
         else  //Down mode selected
            if(COUNT == 0)
               COUNT <= 15;
            else
               COUNT <= COUNT - 1; //Decrement counter
   end
 endmodule
```

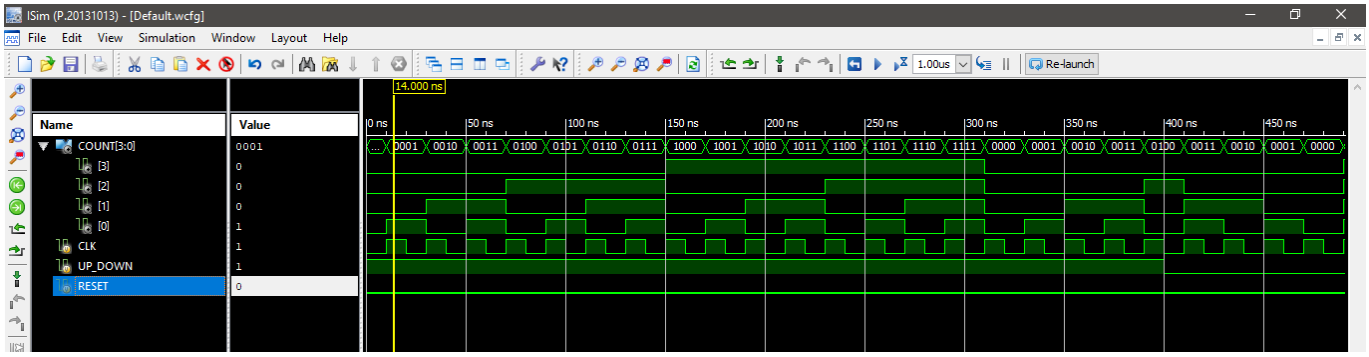
TEST BENCH:

```
module UP_DOWN_COUNTER_TB;
        reg CLK, RESET;
        reg UP_DOWN;
        reg;
        wire [3:0] COUNT;
        // Instantiate the Unit Under Test (UUT)
        UP_DOWN_COUNTER uut (
                .COUNT(COUNT),
                .CLK(CLK),
                .UP_DOWN(UP_DOWN),
                .RESET(RESET));
initial CLK = 0;
   always #10 CLK = ~CLK;
        initial begin
       RESET = 0; UP_DOWN = 1;
       #400;
       UP_DOWN = 0;
       #400;
         end
     endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, 4-Bit Parallel-In Serial-Out shift registers and 4-Bit Up/Down Counters are designed under Verilog HDL in behavioural modelling and its logical operation is verified by simulation using Xilinx ISE Simulator.

VIVA QUESTIONS WITH ANSWERS:

1. **How much storage capacity does each stage in a shift register represent?**
   1-bit

2. **How many clock pulses will be required to completely load the parallel input of a 5-bit PISO shift register?**
   1

3. **What type of register would have a complete binary number shifted in one bit at a time and have all the stored bits shifted out one at a time?**
   Serial-in parallel-out

4. **The bit sequence 1100 is loaded simultaneously into a 4-bit PISO shift register using LOAD option. What will be the value of serial out bit after six clock pulses?**
   0 (bit)

5. **List at least two applications of PISO shift register.**
   Used as parallel to serial converters in Microprocessor/Microcontrollers
   Used for data transfer in communication systems

6. **Number of basic logic gates and Flip-flops required to construct 3-bit UP/DOWN counter are _____**
   Four 2-input AND Gate, two 2-input OR Gate, three JK-Flip flops

7. **If 4-bit UP/DOWN counter operating under UP mode, then each flip-flop is triggered by the _____ output of the preceding flip-flop.**
   Q or Normal

8. **Once an UP/DOWN counter begins its count sequence, it can't be altered. True or False?**
   False

9. **If 4-bit UP/DOWN counter operating under DOWN mode and currently counter holding a value 1100 then what will be the value hold by counter after 5 clock pulses?**
   0111

10. **List at least two applications of UP/DOWN Counters.**
    - Used as visitor counter
    - Used for identifying number of free lot in parking area

Expt. No. 13	Design and Simulate Simple Arithmetic Logic Unit

AIM:

Design and simulate a simple 8-bit microprocessor's ALU unit which performs arithmetic and logical operations (shown in below table) on two 8-bit inputs [7:0]A and [7:0]B. Write the Verilog HDL code in behavioural modelling for the above mentioned design and verify the output using sample test cases.

S. No	ALU_Sel	Operation	Description
1	0000	A+1	Increment A by 1
2	0001	A+B	Addition
3	0010	A-1	Decrement A by 1
4	0011	A-B	Subtraction
5	0100	A*B	Multiplication
6	0101	A==B	Equality
7	0110	A > B	Greater than
8	0111	A < B	Lesser than
9	1000	~ A	Logical NOT
10	1001	A & B	Logical AND
11	1010	A | B	Logical OR
12	1011	~(A & B)	Logical NAND
13	1100	~(A | B)	Logical NOR
14	1101	A ^ B	Logical XOR
15	1110	A >> 1	Right shift by 1
16	1111	A << 1	Left shift by 1

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

PROCEDURE:

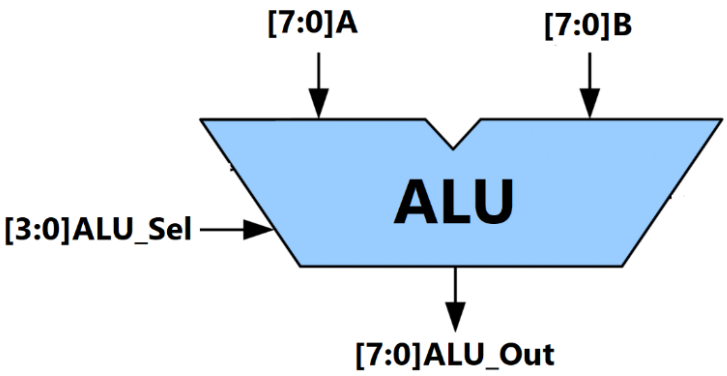
1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the "simulate behavioural model" to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per sample test case.

ARITHMETIC AND LOGIC UNIT (ALU):

THEORY:

An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs. An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR. Size of the input operand is based on size of ALU. For example, 8-bit ALU accepts two 8-bit inputs to perform arithmetic and logical operation on it. There are two operand inputs to an ALU. The operation performed on them depends on ALU-opcode (Selection lines). While the operations are performed on both the operands, there are some operations performed on only one operand. The output is result of operation, and there might be a few more outputs like carry-out, overflow-out, auxiliary-carry etc., coming out of ALU. The output result is placed in a storage register and settings that indicate whether the operation was performed successfully.

BLOCK DIAGRAM:



SIGNAL DESCRIPTION OF ALU:

SIGNAL DESCRIPTION OF ALU	
[7:0]A	8-Bit input signal A
[7:0]B	8-Bit input signal B
[3:0] ALU_Sel	8-Bit ALU Selection lines
[7:0]ALU_Out	8-Bit output signal

TEST CASE:

[7:0] A = 10101111 [7:0] B=01001011

S. No.	ALU_Sel	OPERATION	DESCRIPTION	[7:0]ALU_Out
1	0000	A+1	Increment A by 1	10110000
2	0001	A+B	Addition	11111010
3	0010	A-1	Decrement A by 1	10101110
4	0011	A-B	Subtraction	01100100
5	0100	A*B	Multiplication	01000101
6	0101	A==B	Equality	00000000
7	0110	A > B	Greater than	00000001
8	0111	A < B	Lesser than	00000000
9	1000	~ A	Logical NOT	01010000
10	1001	A & B	Logical AND	00001011
11	1010	A | B	Logical OR	11101111
12	1011	~(A & B)	Logical NAND	11110100
13	1100	~(A | B)	Logical NOR	00010000
14	1101	A ^ B	Logical XOR	11100100
15	1110	A >> 1	Right shift by 1	01010111
16	1111	A << 1	Left shift by 1	01011110

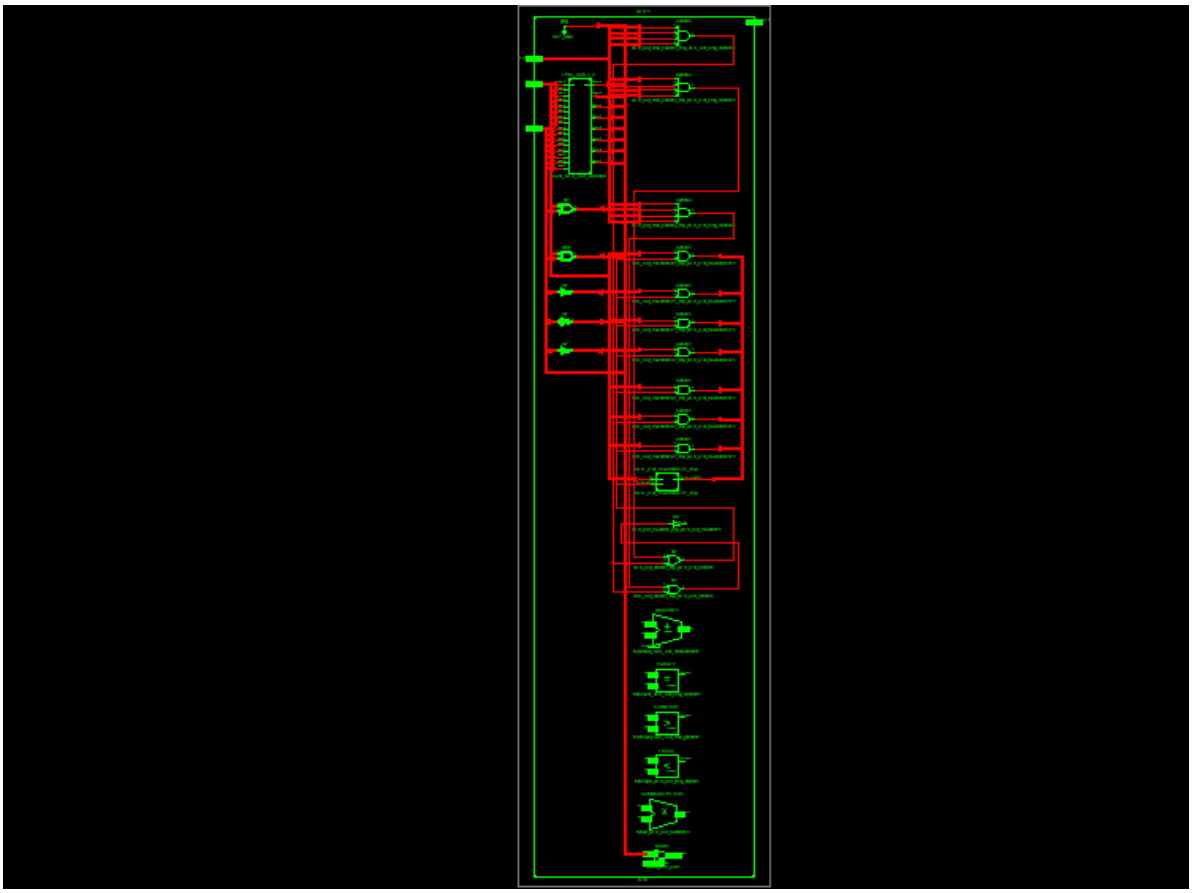
VERILOG HDL CODE (Behavioural):

```
module ALU(ALU_Out,A,B,ALU_Sel);
input[7:0]A,B;
input[3:0] ALU_Sel;
output[7:0]ALU_Out;
reg [7:0]ALU_Out;
always@(ALU_Sel)
begin
    case(ALU_Sel)
        4'b0000: ALU_Out=A+1;   //Increment by 1
        4'b0001: ALU_Out=A+B;   //Addition
        4'b0010: ALU_Out=A-1;   //Decrement by 1
        4'b0011: ALU_Out=A-B;   //Subtrattion
        4'b0100: ALU_Out=A*B;    //Multiplication
        4'b0101: ALU_Out=A==B;  //Equality
        4'b0110: ALU_Out=A>B;   //Greater than
        4'b0111: ALU_Out=A<B;   //Lesser than
        4'b1000: ALU_Out=~A;    //Logical NOT
        4'b1001: ALU_Out=A&B;   //Logical AND
        4'b1010: ALU_Out=A|B;   //Logical OR
        4'b1011: ALU_Out=~(A&B);//Logical NAND
        4'b1100: ALU_Out=~(A|B);//Logical NOR
        4'b1101: ALU_Out=A^B;   //Logical XOR
        4'b1110: ALU_Out=A>>1;  //Right Shift by 1
        4'b1111: ALU_Out=A<<1;  //dLeft Shift by 1
        default:ALU_Out=8'bXXXXXXXX;
    endcase
end
endmodule
```

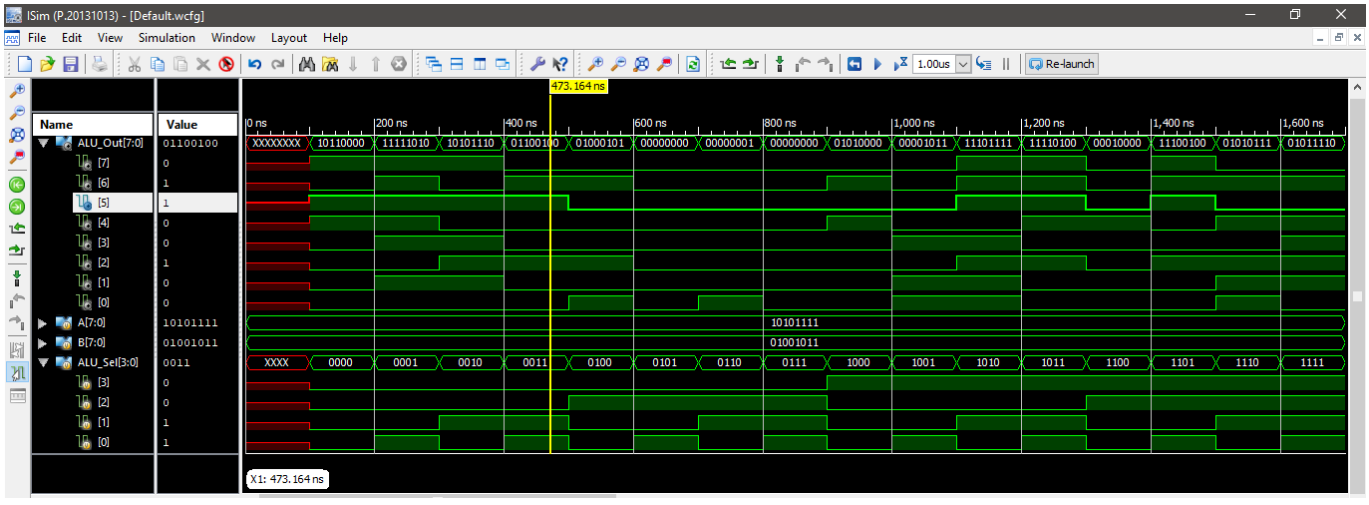
TEST BENCH:

```
module ALU_TB;
    // Inputs
    reg [7:0] A;
    reg [7:0] B;
    reg [3:0] ALU_Sel;
    // Outputs
    wire [7:0] ALU_Out;
    // Instantiate the Unit Under Test (UUT)
    ALU uut (
        .ALU_Out(ALU_Out),
        .A(A),
        .B(B),
        .ALU_Sel(ALU_Sel)
    );
    initial
    begin
        // Initialize Inputs
        A = 8'b10101111;
        B = 8'b01001011;
        #100; ALU_Sel = 0;
        #100; ALU_Sel = 1;
        #100; ALU_Sel = 2;
        #100; ALU_Sel = 3;
        #100; ALU_Sel = 4;
        #100; ALU_Sel = 5;
        #100; ALU_Sel = 6;
        #100; ALU_Sel = 7;
        #100; ALU_Sel = 8;
        #100; ALU_Sel = 9;
        #100; ALU_Sel = 10;
        #100; ALU_Sel = 11;
        #100; ALU_Sel = 12;
        #100; ALU_Sel = 13;
        #100; ALU_Sel = 14;
        #100; ALU_Sel = 15;
        #100;
        $stop;
  end
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, the simple 8-bit ALU of a Microprocessor is designed using Verilog HDL in behavioural modelling and its functionality is verified using test case.

VIVA QUESTIONS WITH ANSWERS:

1. **ALU in Microprocessor/Microcontroller is used for?**
   To perform arithmetic and logical operation such as +,-*,&,|,^,~, etc.,

2. **Size of the ALU decided based on size of _____.**
   Input operands

3. **Assume if A=10101010 and B=11011011, then the result of A>B is?**
   00000000

4. **Assume if A=10101010, then the result of A>>3 is?**
   00010101

5. **Consider a laptop with 64-bit Intel core-i5 processor then, its ALU size is?**
   64-Bit

6. **ALU is the place where the actual executions of instructions take place during the processing operation. True or False?**
   True

7. **Write an expression for XNOR logic using Verilog operator ALU_Out=?**
   ALU_Out= ~(A^B)

8. **The ALU gives the output of the operations and the output is stored in the _____?**
   Registers

9. **In ALU, all operations are performed only on two input operands. True or false?**
   False

10. **A single CPU, FPU or GPU may contain multiple ALUs. True or False?**
    True

Expt. No. 14	Design and Simulate Chocolate Vending Machine using Finite State Machine

AIM:

Design a simple vending machine, which dispatches a chocolate after deposition of 15 rupees. The machine has only one coin slot to receive coins from customers' one coin at a time. In addition, the machine receives only 10 (D) or 5 (N) rupee coin and it doesn't give any change if total deposited amount exceeds 15 rupees. Simulate and verify the designed FSM using Verilog HDL.

H/W & S/W REQUIRED:

S. NO.	COMPONENT	SPECIFICATION
1.	XILINX ISE 14.1	14.1 VERSION
2.	PERSONAL COMPUTER	-

PROCEDURE:

1. Launch Xilinx ISE 14.1 and create a new project by selecting File → New Project.
2. Create a Verilog source file for the project by clicking on Project → New Source from the menu.
3. Type the Verilog HDL program for the given logic
4. To check the syntax errors in the design, double-click on "check syntax" option under process window.
5. Then run "Synthesize XST" option to convert your Verilog HDL code into logic circuit and visualize under "RTL Schematic" option.
6. Create a Verilog test fixture module to verify the functionality of the design by selecting Project → New Source.
7. Based on the logic chosen for implementation all possible set of inputs must be provided with proper delay values in test bench program.
8. Select "Simulation" option in project window, then double click on the simulate behavioural model to open an ISIM Simulator.
9. The output the designed logic will be displayed in the forms of waveforms.
10. Place the mouse cursor on the waveforms area; simultaneously verify the state value of the inputs and outputs as per state transition table.

VENDING MACHINE USING FINITE STATE MACHINE:

THEORY:

A finite state machine is a form of abstraction and it models the behaviour of a system by showing each state it can be in and the transitions between each state. A state machine will change state dependent upon its current state also current factors influencing the system, namely, inputs. One state is designated the initial (or start) state. Beginning with this state, the inputs are sampled periodically, and the machine changes state dependent on both the inputs and its present state. This state may be observed and treated as an output; the present input may also influence the output.

To understand the workings of a finite state machine, you must understand how it changes state. In a given state, a given output produces a change of state (perhaps to the same state). We can represent the finite state machine, then, as a function mapping an input and state to a state. A state transition table is essentially a truth table in which some of the inputs are the current state, and the outputs include the next state, along with other outputs. In a digital circuit, the inputs will be binary values, the transition function will be implemented by using the inputs and the outputs of the flip-flops at a given clock pulse to affect the state of the flip-flops, and the new state will be the new state of the flip-flops after the clock pulse.

An alternative representation of a finite state machine is that of a state diagram. Here, the states are designated by labelled circles. An arrow is drawn from one state to another whenever the machine is able to transition in one-step from the state at the arrow's tail to the state at its head. The arrow is labelled with the input that prompts the transition. State diagrams are very useful for producing the state transition table and ensuring that all possible input transitions are accounted for.

There are two types of FSMs.

- Mealy State Machine: A Finite State Machine is said to be Mealy state machine, if outputs depend on both present inputs & present states.
- Moore State Machine: A Finite State Machine is said to be Moore state machine, if outputs depend only on present states.

In general, the number of states required in Moore state machine is more than or equal to the number of states required in Mealy state machine. Vending Machine is a practical example where FSM is used. The ticket dispatcher unit at the stations, the can drinks, chocolate dispatcher at the shops are some examples of Vending machines.

DESIGN SPECIFICATION:

A simple digital circuit is to be designed for the coin acceptor of a chocolate vending machine. Assume that the chocolate cost 15 INR. The coin acceptor takes only INR of 5 and 10. Exact change must be provided. The coin acceptor does not return extra money. Valid combinations including order of coins are one 5 and one 10, three 5, or one 10 and one 5. Two 10 are valid, but the acceptor does not return money. This digital circuit can be designed by using the finite state machine approach. We can represent the functionality of the digital circuit with a finite state machine.

- **input : 2-bit, coin[1:0]** - no coin(2'b00), INR 5 coin(2'b01), INR of 10 coin(2'b10), coin(2'b11) is invalid
- **input: 1-bit, reset –** to make the vending machine back to initial state S0
- **input: 1-bit, clk –** clock pulse provided to operate sequential circuits (Flip-flops) for FSM
- **output: 1-bit, out** - release chocolate door when out = 1 'b1
- **states: 4 states** – S0 = No coin; S5 = INR of 5; S10 = INR of 10; S15 = INR of 15

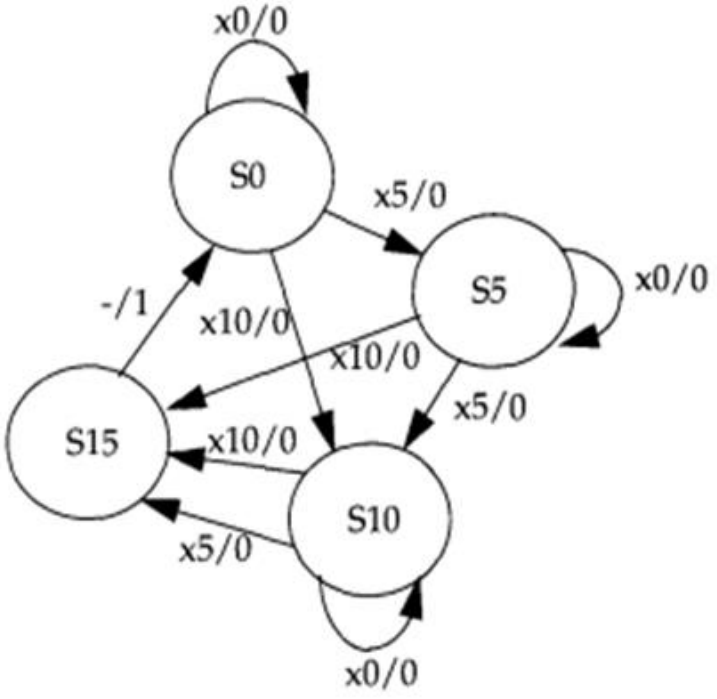
STATE TABLE:

Present state	Next state				Output (out)			
	Coin=00	Coin=01	Coin=10	Coin=11	Coin=00	Coin=01	Coin=10	Coin=11
S0	S0	S5	S10	S0	0	0	0	X
S5	S1	S10	S15	S0	0	0	0	X
S10	S10	S15	S15	S0	0	0	0	X
S15	S0	S0	S0	S0	1	1	1	X

MEALY MODEL - STATE DIAGRAM:

State	*Money*
S0	*0*
S1	*5*
S2	*10*
S3	*15*

Input	*coin[1:0]*
x0	*2'b00*
x5	*2'b01*
x10	*2'b10*
x11	*don't care*



VERILOG HDL CODE (Behavioural):

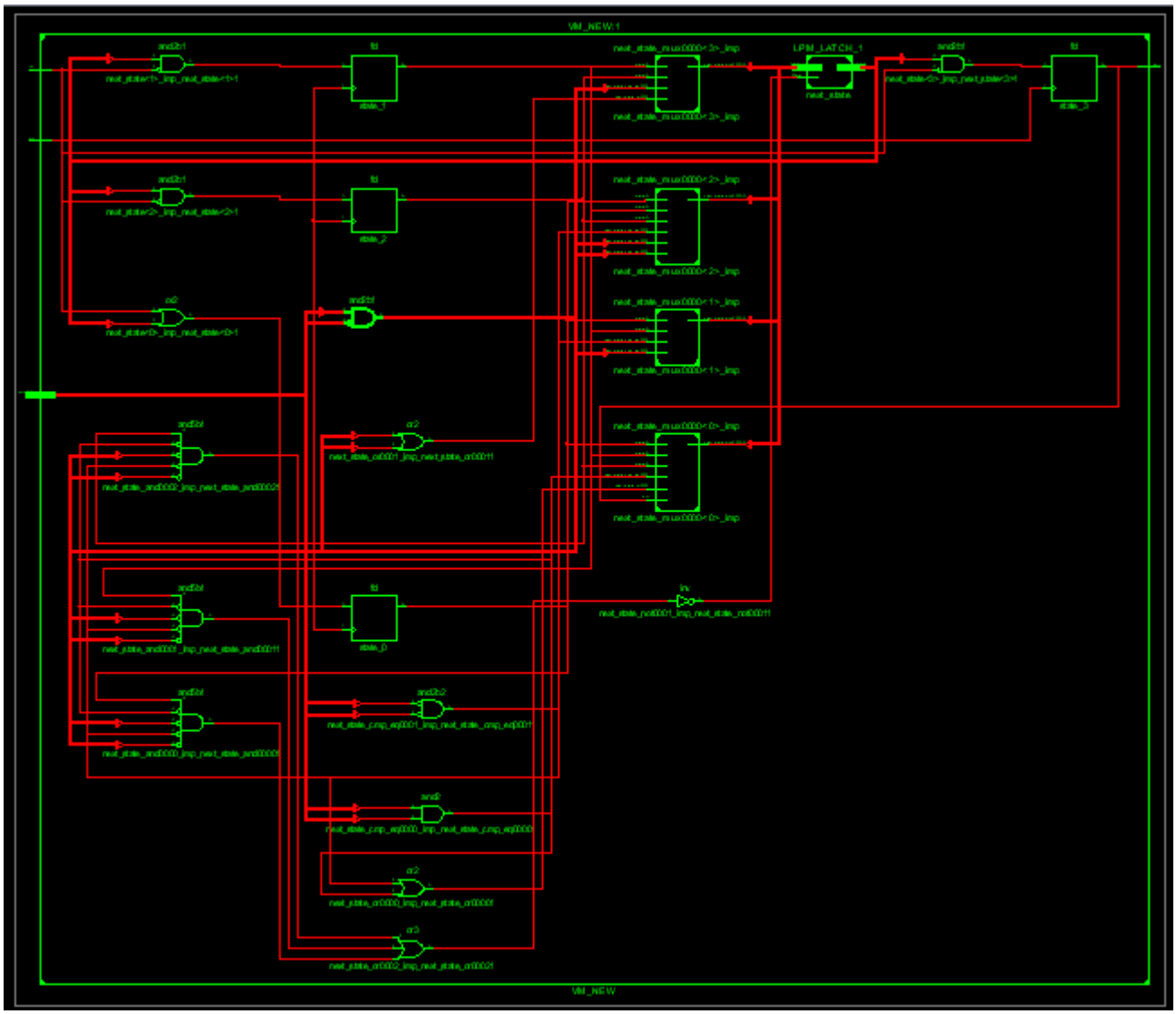
```
module VENDING_MACHINE (out,coin,clk,rst);

 input [1:0]coin;
 input clk,rst;
 output out;
 reg out;
 reg [1:0]state,next_state;

   parameter s0=2'd0,s5=2'd1,s10=2'd2,s15=2'd3;

   parameter x0=2'd0,x5=2'd1,x10=2'd2,x15=2'd3;

   always @ (posedge clk)
     begin
    if(rst)
      state=s0;
    else
      state=next_state;
     end

   always @ (state,coin)
     begin
    case(state)

      s0:begin
         if(coin==x0)
           begin
          next_state=s0; out=0;
           end

         else if(coin==x5)
           begin
          next_state=s5; out=0;
           end

         else if(coin==x10)
           begin
          next_state=s10; out=0;
           end

           else if(coin==x15)
           begin
          next_state=s0; out=1'bx;
           end

      end // case: s0
```

```
 s5:begin
        if(coin==x0)
          begin
         next_state=s5; out=0;
          end
        else if(coin==x5)
          begin
         next_state=s10; out=0;
          end
        else if(coin==x10)
          begin
         next_state=s15; out=0;
          end
          else if(coin==x15)
          begin
         next_state=s0; out=1'bx;
          end
      end // case: s5

      s10:begin
        if(coin==x0)
          begin
         next_state=s10; out=0;
          end
        else if(coin==x5)
          begin
         next_state=s15; out=0;
          end
        else if(coin==x10)
          begin
         next_state=s15; out=0;
          end
          else if(coin==x15)
          begin
         next_state=s0; out=1'bx;
          end
      end // case: s10

      s15:
        begin
          out=1;
          next_state=s0;
        end
        endcase
      end // always @ (state,coin)
endmodule
```

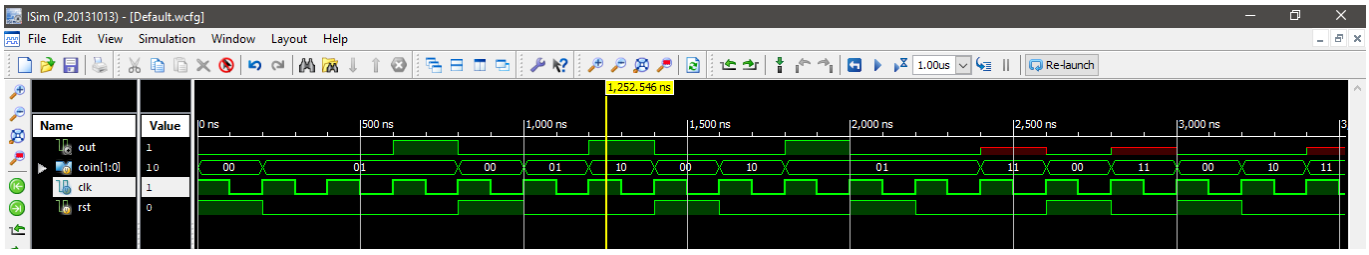
TEST BENCH:

```
module VENDING_MACHINE_TB;
    // Inputs
    reg [1:0] coin;
    reg clk;
    reg rst;
    // Outputs
    wire out;
    // Instantiate the Unit Under Test (UUT)
    VENDING_MACHINE uut (
        .out(out),
        .coin(coin),
        .clk(clk),
        .rst(rst));
initial clk=1;
always #100 clk = ~clk;
    initial begin
        //CASE:1 5+5+5=15
        coin = 0; rst = 1; #200;
        coin = 1; rst = 0; #200;
        coin = 1; rst = 0; #200;
        coin = 1; rst = 0; #200;
        //CASE:2 5+10=15
        coin = 0; rst = 1; #200;
        coin = 1; rst = 0; #200;
        coin = 2; rst = 0; #200;
        //CASE:3 10+5=15
        coin = 0; rst = 1; #200;
        coin = 2; rst = 0; #200;
        coin = 1; rst = 0; #200;
        //CASE:4 5+15=15 No change
        coin = 1; rst = 1; #200;
        coin = 1; rst = 0; #200;
        coin = 3; rst = 0; #200;
        //CASE:5 15=15
        coin = 0; rst = 1; #200;
        coin = 3; rst = 0; #200;
        //CASE:6 10+15=15 No change
        coin = 0; rst = 1; #200;
        coin = 2; rst = 0; #200;
        coin = 3; rst = 0; #200;
        end
endmodule
```

RTL SCHEMATIC:



SIMULATION OUTPUT:



RESULT:

Thus, the chocolate vending machine controllers is designed using Finite state machine and realized with the help of Verilog HDL in behavioural modelling and the its logical function is verified using state transition table.

VIVA QUESTIONS WITH ANSWERS:

1. **What is Finite State Machine?**
   A finite state machine is a form of abstraction and it models the behaviour of a system by showing each state it can be in and the transitions between each state.

2. **A Finite State Machine is said to be _____ state machine, if outputs depend on both present inputs & present states.**
   Mealy

3. **A Finite State Machine is said to be _____ state machine, if outputs depend only on present states.**
   Moore

4. **What is the use of state diagram?**
   State diagrams are very useful for producing the state transition table and ensuring that all possible input transitions are accounted for.

5. **What is the purpose transition table?**
   A state transition table is essentially a truth table in which some of the inputs are the current state, and outputs include the next state, along with other outputs.

6. **Finite state machine will initially state with a state called initial state. True or false?**
   True

7. **In FSM, transition from one state to another is represented by _____ symbol.**
   Arrow

8. **The major difference between Mealy and Moore machine is**
   Mealy and Moore machine vary over how the outputs depends on prior one (transitions) and on the latter one (states).
9. **Number of states required in Mealy state machine is more than or equal to the number of states required in Moore state machine. True or False?**
   False

10. **How states are represented in FSM?**
    States are designated by labelled circles