

Department of Electronics Engineering
IIT (ISM), Dhanbad

LAB MANUAL

**DIGITAL SYSTEM DESIGN
LAB**

ECC 204

Conducted in:

**Part I:
Digital Electronics Lab**

**Part II:
VLSI Lab**

General Instructions

DO'S

1. Be regular to the lab.
2. Follow proper dress code.
3. Maintain silence.
4. Know the theory behind the experiment before coming to the lab.
5. Identify the different leads or terminals or pins of the IC before making connection.
6. Know the Biasing Voltage required for different families of IC's and connect the power supply voltage and ground terminals to the respective pins of the IC's.
7. Know the current and voltage rating of the IC's before using them in the experiment.
8. Avoid unnecessary talking while doing the experiment.
9. Handle the IC Trainer Kit properly.
10. Mount the IC Properly on the IC ZIF Socket.
11. While doing the interfacing, connect proper voltages to the interfacing kit.
12. Keep the Table clean.
13. Take a signature of the In-charge before taking the kit/components.
14. After the completion of the experiments switch off the power supply and return the apparatus.
15. Arrange the chairs/stools and equipment properly before leaving the lab.
16. Know the basic theory related to the experiment before starting any new experiment

DON'TS

1. Do not exceed the voltage Rating.
2. Do not interchange the IC's while doing the experiment.
3. Avoid loose connections and short circuits.
4. Do not throw the connecting wires to floor.
5. Do not come late to the lab.
6. Do not operate IC trainer kits unnecessarily.
7. Do not panic if you don't get the output.

Contents

S. No.	Name of Experiments	Page no.
PART I : Digital Electronics		
01	Design and hardware implementation of: a. 2-bit Adder/Subtractor with XOR as well as NAND gates, b. 4:1 Multiplexer using universal gates and realization of Full Adder using Multiplexers, c. BCD Adder using two binary adders (IC based) and other gates, d. 3:8 Decoder and realization of Full Adder	3 6 9 12
02	Realization of R-S, D and J-K latches and D Flip-Flop	14
03	Realization of Mod-8 Up-Down Ripple Counter	17
04	Realization of synchronous Mod-3 and Mod-2 counters	20
05	Realization of higher Mod counter by cascading lower Mod counters	22
PART II: Digital System Design using HDL and EDA		
06	Modeling different types of gates: (a) 2-input NAND (b) 2-input OR gate (c) 2-input NOR gate (d) NOT gate (e) 2-input XOR gate (f) 2-input XNOR gate	24
07	Modeling (a) Half-adder (b) Full-adder	26 27
08	Modeling a “D flip-flop”	29
09	Modeling a “D Latch”	31
10	Modeling a (a) 2-to-1 Multiplex (b) 2-to-4 Decoder (c) Tri-State Buffer <i>[Do It Yourself]</i>	32
11	Modeling a 4-to-1 Multiplexer <i>[Do It Yourself]</i>	32
12	Modeling a 4-bit PARALLEL ADDER <i>[Do It Yourself]</i>	32
13	Modeling a 4-bit adder-subtractor circuit <i>[Do It Yourself]</i>	33

Experiment-1(a)

Aim: Design and hardware implementation of 2-bit Adder/Subtractor with XOR as well as NAND gates.

Apparatus required: Bread board, Connecting wires, IC 7400, IC 7486, IC 7404, IC 7408, IC trainer kit

Theory:

Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

$$S = A \oplus B$$

$$C = AB$$

Half-Subtractor: A combinational logic circuit that performs the subtraction of two data bits, A and B, is called a half-subtractor. Subtraction will result in two output bits; one of which is the difference bit, and the other is the borrow bit. The Boolean functions describing the half-subtractor are:

$$\text{Difference} = A \oplus B$$

$$\text{Borrow} = A'B$$

Truth Table and observation Table

Full Adder

Half Subtractor			
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Adder			
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Boolean Expressions using K-map:

Half Adder

$$\text{Sum} = A \oplus B$$

	B	00	01
A	00		1
	01	1	

$$\text{Carry} = AB$$

	B	00	01
A	00		
	01		1

Half Subtractor

$$\text{Difference} = A \oplus B$$

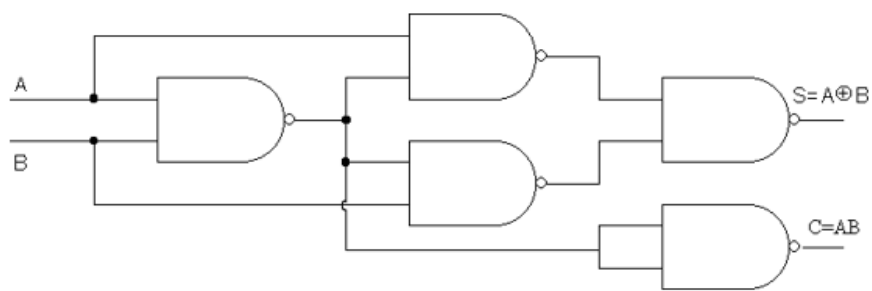
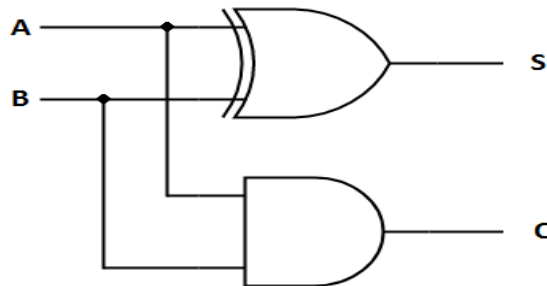
	B	00	01
A	00		1
	01	1	

$$\text{Borrow} = A'B$$

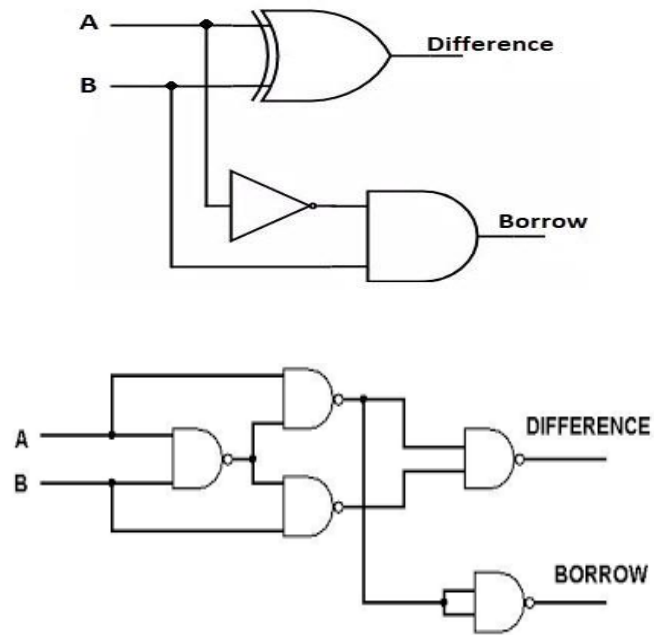
	B	00	01
A	00		1
	01		

Digital Circuit Diagram:

Half Adder



Half Subtractor:



Procedure:

1. Connect the trainer kit to ac power supply.
2. Connect the all ICs pins for given logic functions to be realized.
3. Connect the inputs to logic sources and output to logic indicator.
4. Apply various input combinations and observe output for each one.
5. Verify the truth table for each input/ output combination.
6. Repeat the process for all logic functions.
7. Switch off the ac power supply.

Experiment 1(b)

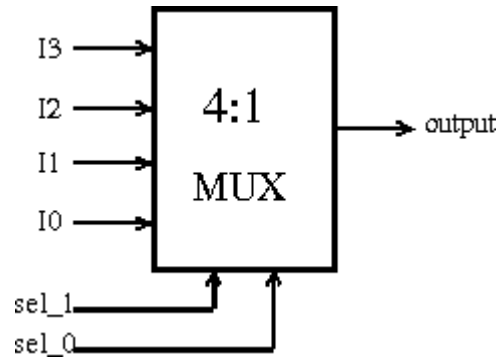
Aim: Design and hardware implementation of 4:1 Multiplexer using universal gates and realization of Full adder using Multiplexers.

Apparatus and Components Required: Connecting wires, IC Trainer Kit, IC 7400, IC 7410, IC 7420.

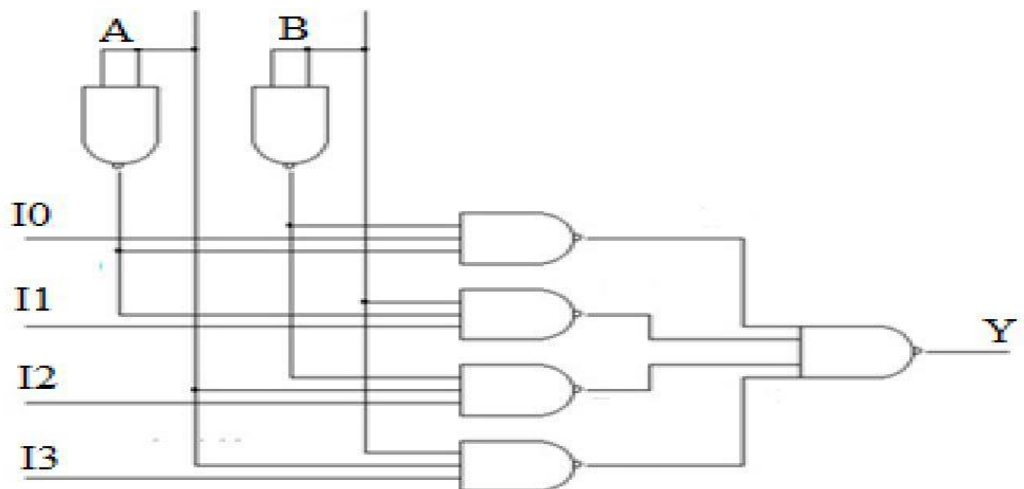
Theory: Multiplexers are very useful components in digital systems. They transfer a large number of information units over a smaller number of channels, (usually one channel) under the control of selection signals. Multiplexer means many to one. A multiplexer is a circuit with many inputs but only one output. By using control signals (select lines) we can select any input to the output. Multiplexer is also called as data selector because the output bit depends on the input data bit that is selected. The general multiplexer circuit has 2^n input signals, n control/select signals and 1 output signal. Simple block diagram of 4:1 MUX is given below.

Truth Table:

INPUT						OUTPUT
A	B	I0	I1	I2	I3	Y(V)
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1



Digital Circuit Diagram of MUX using NAND Gates :



Full adder implementation by using 4:1 Mux:

Two 4:1 Mux can be used to realize carry and adder of full adder separately. The truth table of full adder is given by:

$$S = \sum(1,2,4,7)$$

$$C = \sum(3,5,6,7)$$

INPUTS			OUTPUTS	
A	B	Cin	S	C
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

Let A, B be the selection line of 4:1 MUX on observing the truth table

On observing the truth table following cases can be deduced for Sum of full adder:

When A=0, B=0; S=Cin

When A=0, B=1; S=Cin'

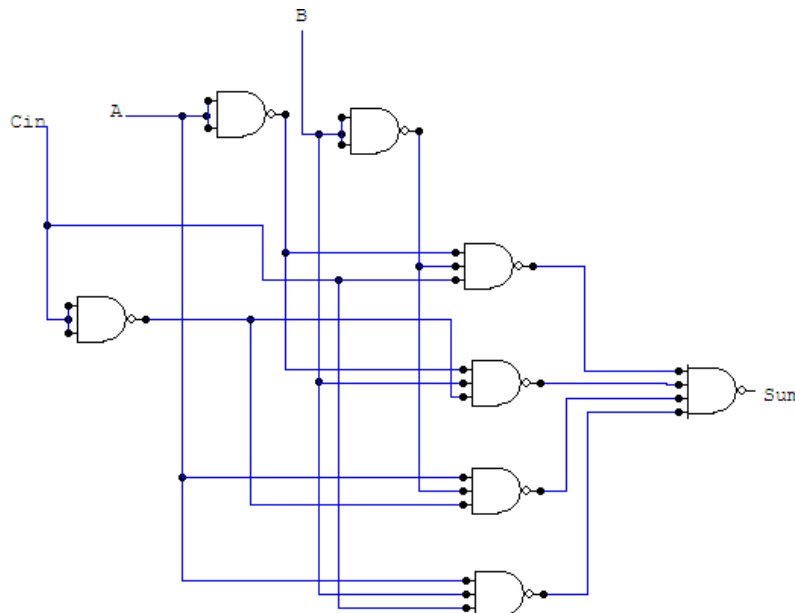
When A=1, B=0; S=Cin'

When A=1, B=1; S=Cin

The SOP expression for Sum is given by

$$S = A'B'Cin + A'BCin' + AB'Cin' + ABCin$$

The Circuit Diagram for Sum is given by



Similar method can be also applied for the Carry of full adder

When $A=0, B=0; C=Cin$

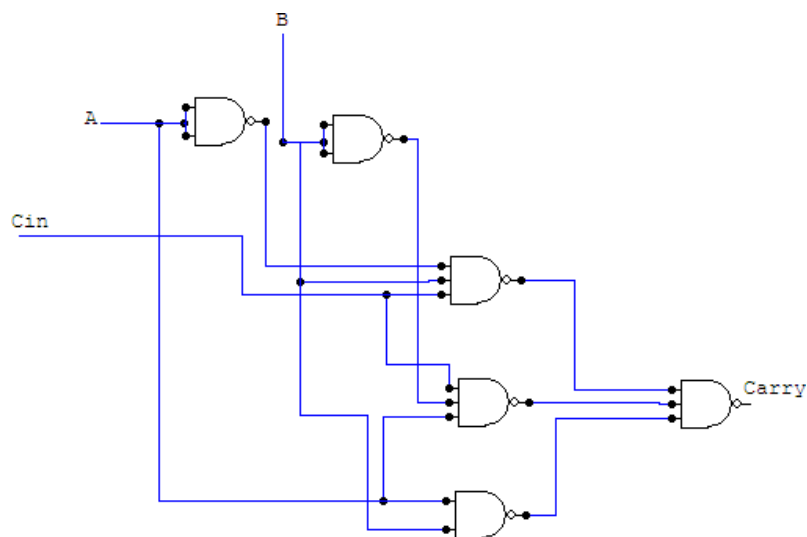
When $A=0, B=1; C=Cin$

When $A=1; B=0; C=Cin$

When $A=1; B=1; C=1$

The SOP expression for carry of full adder given by $C=A'BCin+AB'Cin+AB$

The Circuit diagram of Carry implementation of full adder by 4:1 Mux given by:



Procedure:

1. Connect the trainer kit to ac power supply.
2. Connect the NAND gates IC for given logic functions to be realized.
3. Connect the inputs to logic sources and output to logic indicator.
4. Apply various input combinations and observe output for each one.
5. Verify the truth table for each input/ output combination.
6. Repeat the process for all logic functions.
7. Switch off the ac power supply.

Experiment 1-c

Aim: Design and hardware implementation of BCD adder using two binary adder (IC 7483) and other gates.

Apparatus and Components required: Bread Board, IC trainer kit, Connecting wires, IC 7483, IC 7400

Theory: The full form of BCD is Binary-Coded Decimal. Since this is a coding scheme relating decimal and binary numbers, four bits are required to code each decimal number. The code is also known as 8-4-2-1 code. This is because 8, 4, 2, and 1 are the weights of the four bits of the BCD code. The weight of the LSB is 2^0 or 1, that of the next higher order bit is 2^1 or 2, that of the next higher order bit is 2^2 or 4, and that of the MSB is 2^3 or 8. Therefore, this is a weighted code. 0 to 9 is the legal number available in BCD for numbers greater some adjustment to be carried out for conversion into BCD.

Binary to BCD:

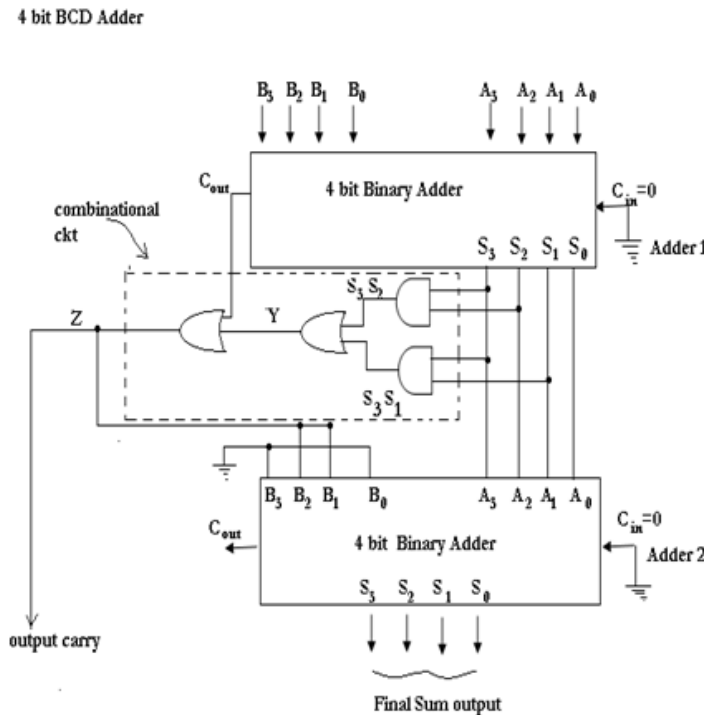
Binary Number				BCD				
Y3	Y2	Y1	Y0	A	B	C	D	E
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and should appear in the form listed in the columns. A BCD adder that adds 2 BCD digits and produce a sum digit in BCD. The 2 decimal digits, together with the input carry, are first added in the top 4-bit adder to produce the binary sum. Binary Coded Decimal is a method of using binary digits to represent the decimal digits 0 through 9. The valid BCD numbers are (0000 to 1001) BCD. Each digit of the decimal number will be represented by its four-bit binary equivalent. Following, three cases arises in BCD addition

1. The resulting BCD number equal to less than (1001) BCD.
2. The resulting BCD number greater than (1001) BCD.
3. Carry is generated in the BCD addition.

The two BCD inputs to be added are applied at inputs A and B of the first binary adder IC 7483. The sum output of the first binary adder is given to the B input of the second binary adder. The A input of the binary adder is given (0110) BCD when a carry is generated from the first adder or when sum from the first binary adder is greater than (0110) BCD, else A input is (0000) BCD. The following Boolean expression is used to find whether (0110) BCD or (0000) BCD needs to be applied to the A input, $C_{out} = C_{out1} + S_4 (S_3 + S_2)$. Where S_4, S_3, S_2, S_1 are the sum of the BCD from the first binary adder with S_4 as the MSB and S_1 as the LSB. C_{out1} is the carry output from the first binary adder.

Digital Circuit Diagram of BCD Adder (only NAND Gate to be used in Lab):



Truth Table:

BCD SUM				CARRY
S4	S3	S2	S1	C
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Procedure:

1. Connect the trainer kit to ac power supply.
2. Connect the all ICs pins for given logic functions to be realized.
3. Connect the inputs to logic sources and output to logic indicator.
4. Apply various input combinations and observe output for each one.
5. Verify the truth table for each input/ output combination.
6. Repeat the process for all logic functions.
7. Switch off the ac power supply.

Questions:

1. What is the difference between Decimal adder and BCD adder?
2. What is the operating temperature and supply voltage range for IC7483?
3. Why do we add 6 in BCD addition?

Experiment 1-d

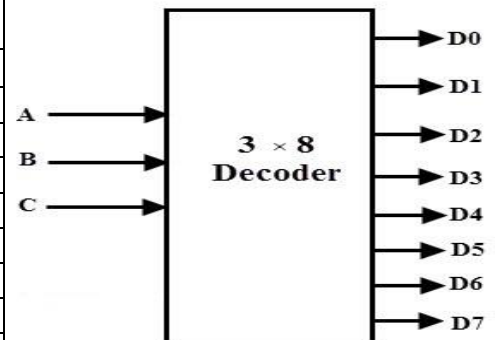
Aim: Design and hardware implementation of 3:8 Decoder (using NAND gates) and implementation of Full Adder subsequently.

Apparatus and Components Required: Bread Board, IC trainer Kit, IC 7400, Connecting wires.

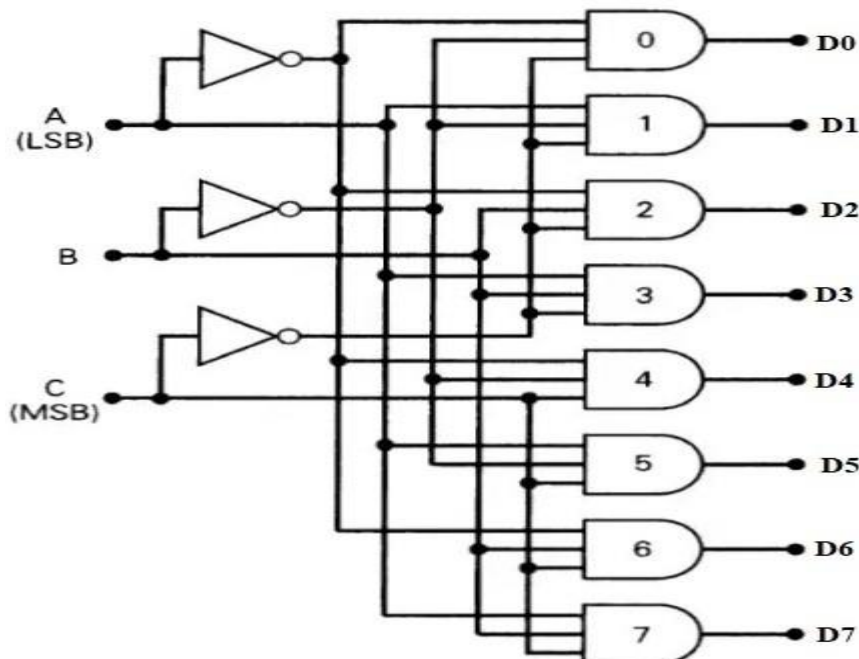
Brief Theory: A decoder is a combinational circuit that converts binary information from n input lines to up to 2^n output lines. These decoders are called n -to- m line decoders such that: $m \leq 2^n$. A 3-to-8 line decoder has three inputs and eight outputs. The decoder decodes the input binary code represented by the three bits and generates all eight min-terms of the inputs. Only one output is one while the other seven are zeros. The logical block diagram given by:

Truth Table:

Input			Output							
A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

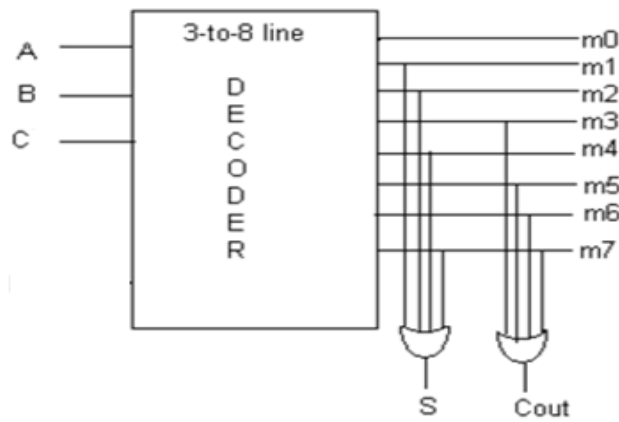


Circuit Diagram (to be designed by using only NAND gates in Lab):



Full Adder realization:

Full Adder can be implemented using 3 to 8 decoder in following ways:



Procedure:

1. Using truth table obtain the logical expression for this decoder.
2. Draw the circuit diagram for the obtained reduced function using minimum number of gates.
3. Implement the reduced circuit using digital ICs on a bread board.
4. Observe the output and record it in the observation table and check it with the truth table.
5. After implementing decoder, realize logical expression of sum and carry for a full adder using 3 to 8 line decoder.

Questions:

1. Draw logic diagram of a 2 to 4 line decoder using NOR gates only.
2. Construct a 5×32 decoder with four 3×8 decoders with enable input and one 2×4 decoder.
3. What is the difference between decoder and demultiplexer? Can a decoder be used as demultiplexer? How / Why?

Experiment 2

Aim: Realization of R-S, D, J-K latches and D flip-flop using NAND gates only.

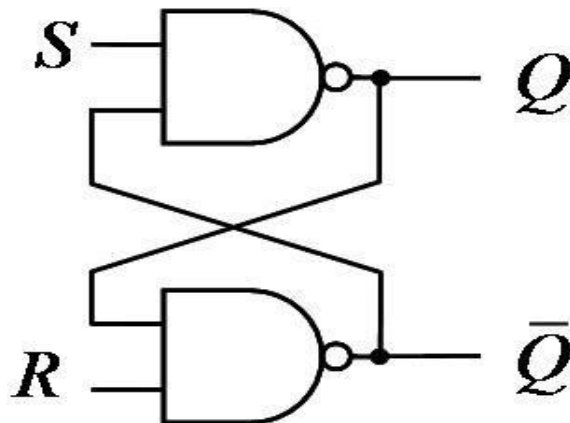
Apparatus and Components required: Bread board, IC trainer kit. Connecting wires, IC 7400, IC 7410

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

1. S-R Latch:

An **S-R latch** consists of two cross-coupled NOR gates. An S-R flip-flop can also be design using cross-coupled NAND gates as shown. A clocked S-R flip-flop has an additional clock input so that the S and R inputs are active only when the clock is high. When the clock goes low, the state of flip-flop is latched and cannot change until the clock goes high again. Therefore, the clocked S-R flip-flop is also called “enabled” S-R flip-flop.

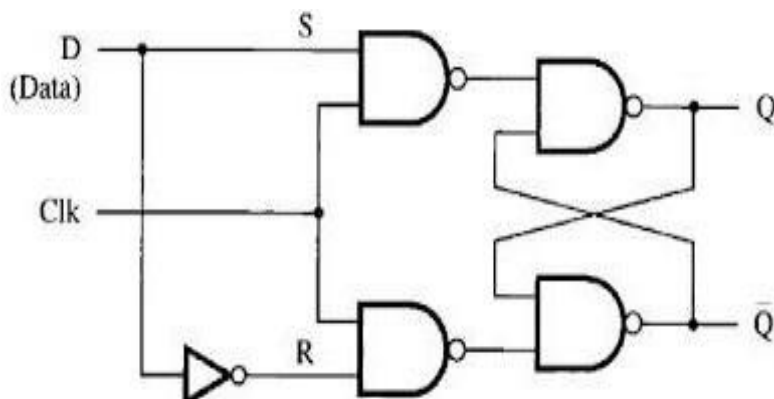
S	R	Q	Q''
0	0	Prev	prev
0	1	0	1
1	0	1	1
1	1	Undefined	Undefined



2. D Latch:

A **D latch** combines the S and R inputs of an S-R latch into one input by adding an inverter. When the clock is high, the output follows the D input, and when the clock goes low, the state is latched.

Clk	D	Q	Q''
0	x	Prev	Prev
1	0	0	1
1	1	1	0

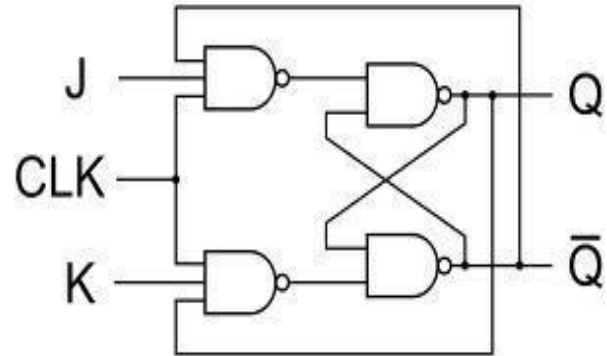


3. JK Latch:

The **JK latch** is a less often used alternative that instead of setting or resetting the value, toggles the value on an input.

Latch:

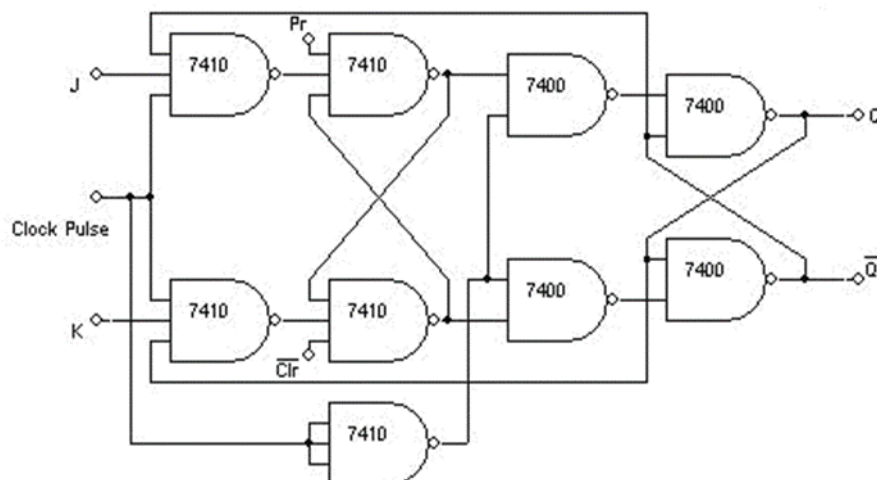
J	K	Q	Q''
0	0	Prev	Prev
0	1	0	1
1	0	1	0
1	1	Toggles	Toggles



4. JK Flip Flop:

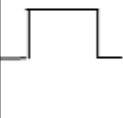

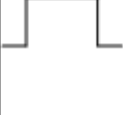
The JK flip-flop is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1". Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle". Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF". To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC's the much-improved Master- Slave JK Flip-flop was developed. The master-slave flip-flop eliminates all the timing problems by using two SR flip-flops connected together in a series configuration. One flip-flop act as the "Master" circuit, which triggers on the leading edge of the clock pulse while the other acts as the "Slave" circuit, which triggers on the falling edge of the clock pulse. This results in the two sections; the master section and the slave section being enabled during opposite half-cycles of the clock signal.

Digital circuit Diagram of Master-Slave J-K Flip Flop



Function Table of Master Slave Flip-Flop:

The input signals J and K are connected to the gated "master" SR flip-flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip-flop is the inverse (complement) of the "master" clock input, the "slave" SR flip-flop does not toggle. The outputs from the "master" flip-flop are only "seen" by the gated "slave" flip-flop when the clock input goes "LOW" to logic level "0". When the clock is "LOW", the outputs from the "master" flip-flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip-flop now responds to the state of its inputs passed over by the "master" section. Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip-flop are fed through to the gated inputs of the "slave" flip-flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip-flop edge or pulse-triggered.

		Inputs			Output
PR	CLR	CLK	J	K	Q
0	0	X	X	X	Race Condition
0	1	X	X	X	1
1	0	X	X	X	0
1	1	X	0	0	No change
1	1		0	1	0
1	1		1	0	1
1	1		1	1	Toggle

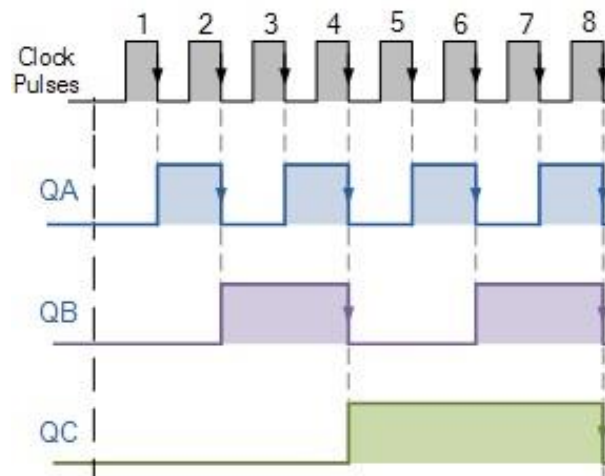
Sequence table of Up counter (M=0):

Clock pulse	Q_c	Q_B	Q_a
1	1	1	1
2	1	1	0
3	1	0	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1
8	0	0	0

Sequence table for down counter (M=1):

Clock pulse	Q_c	Q_B	Q_a
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Timing Diagram:



Procedure:

1. Make the connections as per the above circuit diagram.
2. Switch on the main power supply and the trainer kit.
3. Note down sequence table for mod 8 ripple counter.

Questions:

1. What is the difference between ripple counter and synchronous counter?
2. How many states will there be in a 4-bit ripple counter?
3. What is major drawback of a ripple counter?

Experiment 4

Aim: Realization of Mod-2 and Mod-3 Synchronous counter.

Apparatus and Components required: Bread Board, Connecting wires, IC 7476, IC 7400

Theory:

A *counter* is a sequential logic circuit that goes through a prescribed sequence of states upon the application of input pulses. The prescribed sequence can be a binary sequence or any other sequence. A counter that goes through 2^N (N is the number of flip-flops in the series) states is called a *binary counter*. The modulus of a counter is the number of different states it is allowed to have. Counter modulus is normally 2^N unless controlled by a feedback circuit which limits the number of possible states (an example being the decimal counter). Counters are very widely used in almost all computers and other digital electronic systems. There are two major categories of counters: asynchronous counters and synchronous counters.

Asynchronous Counters

Counters arranged so that the output of one flip-flop generates the clock input of the next higher stage are generally called *asynchronous counters* (or ripple counter). In other words, in asynchronous counters, the CLK inputs of all flip-flops (except the first one) are triggered not by the incoming pulses but rather by the transition that occurs in other flip-flops. Therefore, the change of state of a particular flip-flop is dependent upon the present state of other flip-flops.

Synchronous Counters

Synchronous counters eliminate the cumulative flip-flop delay seen in ripple counter. Each flip-flop is clocked by the same clock signal. Each gate selectively controls when each more significant bit flip-flop is to change state (toggle) on the next clock transition. Such control (enable) can be realized by setting, for example, the J and K inputs of a J-K flip-flop. Because of this control, the addition of a common clock will synchronize data transfer and all flip-flops will change state simultaneously. The important feature of a synchronous counter is that the transitions of the individual flip-flops are synchronized to a master clock signal.

To Design Synchronous counter:

1. Decide the number and type of flip-flop (FF)
2. Write excitation table for flip-flop
3. State diagram and circuit excitation table
4. Obtain Boolean expression
5. Draw circuit diagram

Mod-3 Synchronous Counter

No. of FF = 2

No. of bits = 2

Mod-2 Synchronous Counter

No. of FF = 1

No. of bits = 1

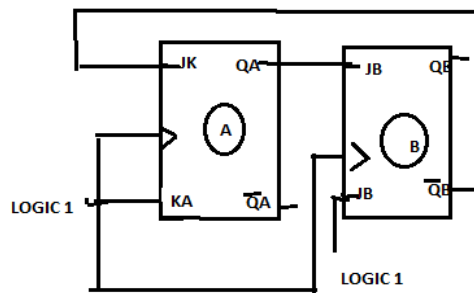
Sequence table: Mod-3

Count	Q _B	Q _A
0	0	0
1	0	1
2	1	0

Sequence table: Mod-2

Count	Q
0	0
1	1

Circuit Diagram of Mod-3 Synchronous counter:



Procedure:

1. Make the connections as per the above circuit diagram.
2. Switch on the power supply and the trainer kit.
3. Note down the state of the counter and verify it with the sequence table for mod-2 and mod-3 Synchronous counter.

Questions:

1. What is the difference between synchronous counter and asynchronous counter?
2. What is the frequency of each stage of the flip-flops?
3. Realize this circuit using D-flip-flop?
4. Design the circuit diagram of mod-4 synchronous counter

Experiment 5

Aim: Realization of higher Mod counter by cascading lower Mod counters.

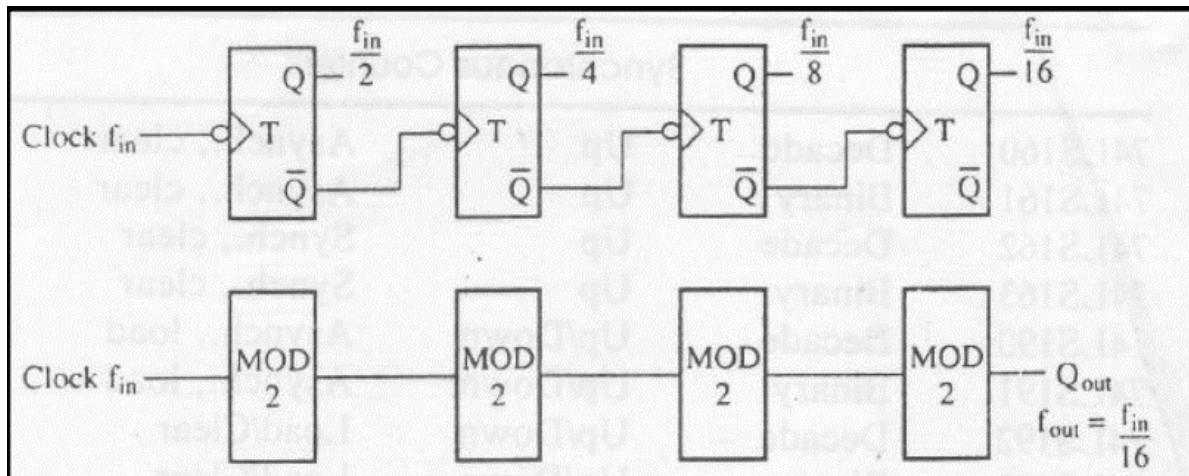
Apparatus and Components required: Bread Board, Connecting wires, IC 74LS90

Theory:

Counter circuits can be cascaded to increase both the modulus of the count sequence and the frequency division. Large counter applications requiring several stages of cascaded counters include digital time clocks, frequency dividers, and synchronization circuits.

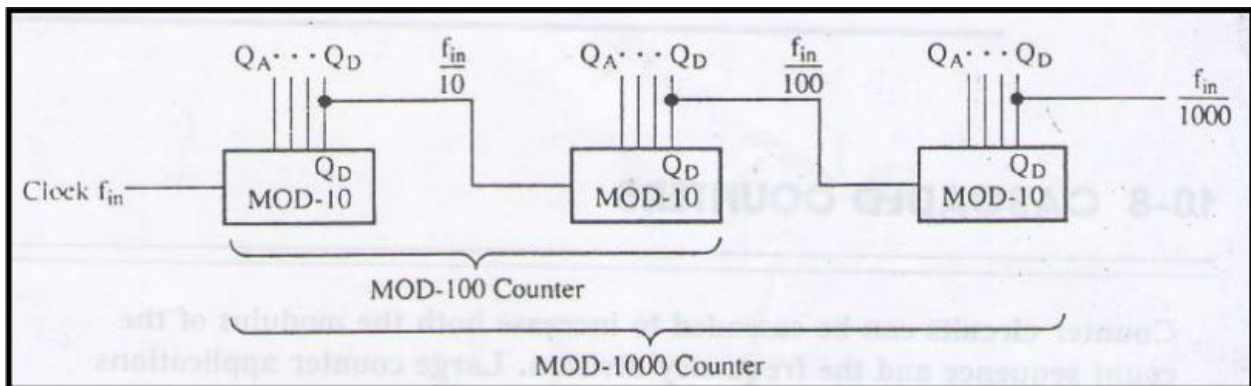
The simplest example of cascaded counter stages is an asynchronous counter. The individual toggle flip-flop stages of an asynchronous counter are MOD-2 counters. MOD-2 counters are cascaded by routing the output of one stage into the clock input of the next stage. With each cascaded stage, the modulus of the counter increases. The final modulus of the counter is equal to the modulus of the individual stages multiplied together. Thus, a 4-bit asynchronous counter has a modulus of $2 \times 2 \times 2 \times 2 = 16$. The output frequency from the final stage is equal to the input frequency divided by the modulus.

Block Diagram of Mod-16 counter using Cascaded MOD-2 Counters:



The 74LS90 IC counter is an example of a counter circuit that requires cascading in order to obtain a decade counter. The decade counter is formed by cascading a MOD-2 counter with a MOD-5 counter. The final modulus is 2×5 , or 10. Several 74LS90 counters could be cascaded together to obtain MOD-10, MOD-100, and MOD-1000 counters. The most significant output bit, QD, is used as the cascaded clock input to the next stage. The modulus of the cascaded counter is not limited to a multiple of the full modulus of the counters in the cascaded circuit. It can be fine-tuned to be any integer value by pre-setting or clearing the count stages as required.

Block Diagram of Mod-1000 counter using Cascaded Mod-10 Counters:



Procedure:

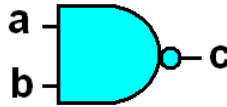
1. Make the connections as per desired counter circuit.
2. Switch on the power supply and the trainer kit.
3. Note down the state of the counter and verify it.

Experiment 6 (a)

Aim: Using DATA FLOW type of architectural modeling, write a VHDL code to describe the functionality of a “2-input NAND” gate. Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

Software Package used: Xilinx ISE Design Suite: System Edition 13.2

Symbol:



VHDL code:

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
```

```
ENTITY nand2_gate IS
```

```
PORT ( a, b : IN STD_LOGIC;
```

```
        c : OUT STD_LOGIC );
```

```
END nand2_gate;
```

```
ARCHITECTURE nand2_gate_arch OF nand2_gate IS
```

```
BEGIN
```

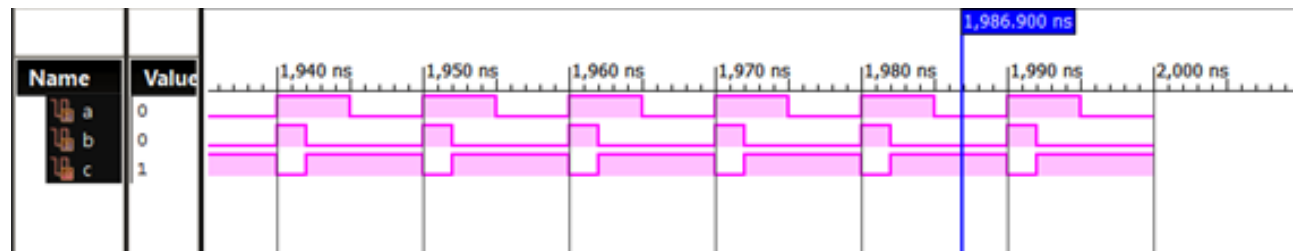
```
c <= a NAND b;
```

```
END nand2_gate_arch;
```

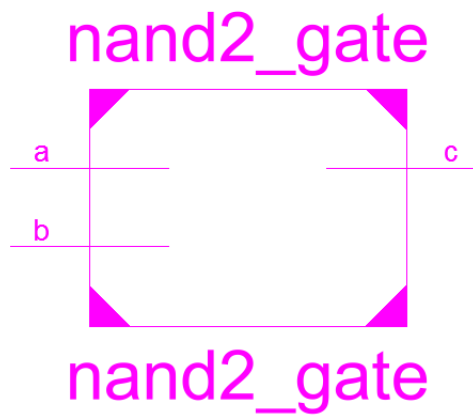
Stimuli for input ports:

Clock Parameters	Port a	Port b
Signal Name	/nand2_gate/a	/nand2_gate/b
Value Radix	Binary	Binary
Leading Edge Value	1	1
Trailing Edge Value	0	0
Starting at Time Offset	0	0
Cancel after Time Offset	<blank>	<blank>
Duty Cycle (%)	50	20
Period	10 ns	10 ns

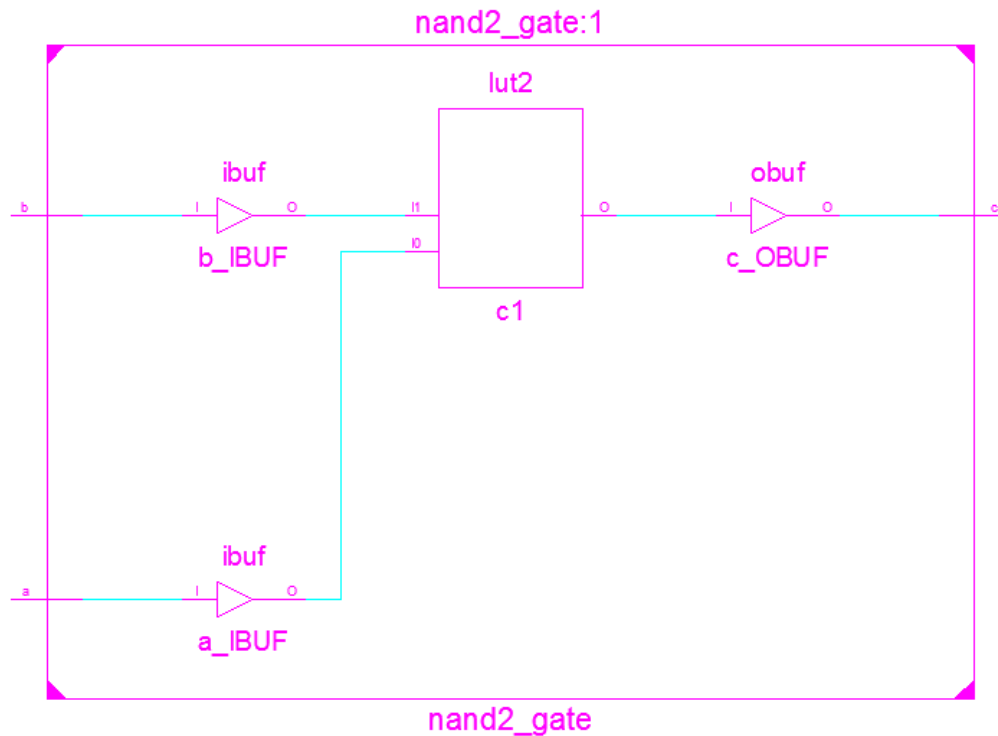
Waveform:



RTL Schematic:



Technology Schematic:



Experiment 6 (b – f):

In a similar way, using DATA FLOW type of architectural modeling, write VHDL codes to describe the functionality of the following gates:

- b) 2-input OR gate
- c) 2-input NOR gate
- d) NOT gate
- e) 2-input XOR gate
- f) 2-input XNOR gate

Compile and simulate the codes to obtain the timing waveform. Draw the RTL and Technology schematic of the above gates.

Experiment 7 (a)

Aim: Using DATA FLOW type of architectural modeling, write a VHDL code to describe the functionality of a “HALF ADDER” circuit. Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

Software Package used: Xilinx ISE Design Suite: System Edition 13.2

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ha is
    Port ( i1_ha : in  STD_LOGIC;
          i2_ha : in  STD_LOGIC;
          sum_ha : out STD_LOGIC;
          carry_ha : out STD_LOGIC);
end ha;

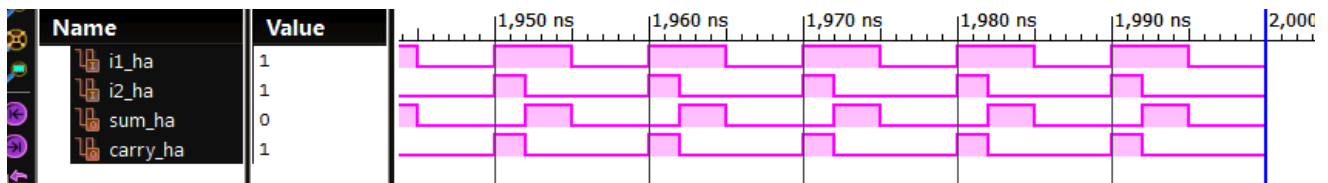
architecture ha_arch of ha is

begin
    sum_ha <= i1_ha xor i2_ha;
    carry_ha <= i1_ha and i2_ha;
end ha_arch;
```

Stimuli for input ports:

Clock Parameters	Port i1_ha	Port i2_ha
Signal Name	/ha/i1_ha	/ha/i2_ha
Value Radix	Binary	Binary
Leading Edge Value	1	1
Trailing Edge Value	0	0
Starting at Time Offset	0	0
Cancel after Time Offset	<blank>	<blank>
Duty Cycle (%)	50	20
Period	10 ns	10 ns

Waveform:



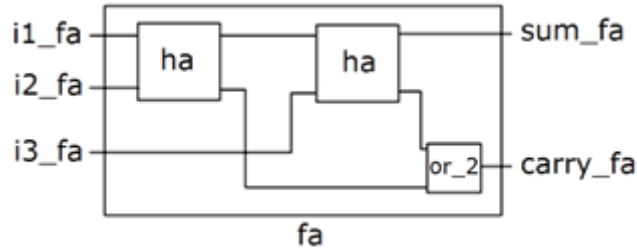
Draw the RTL and Technology schematic of your design

Experiment 7 (b)

Aim: Use two HALF ADDERS and one OR gate as components to model a FULL ADDER using “Component Instantiation”. Compile and simulate the codes to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

Software Package used: Xilinx ISE Design Suite: System Edition 13.2

Block diagram:



VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fa is
    Port ( i1_fa, i2_fa, i3_fa : in  STD_LOGIC;
          sum_fa, carry_fa : out STD_LOGIC);
end fa;

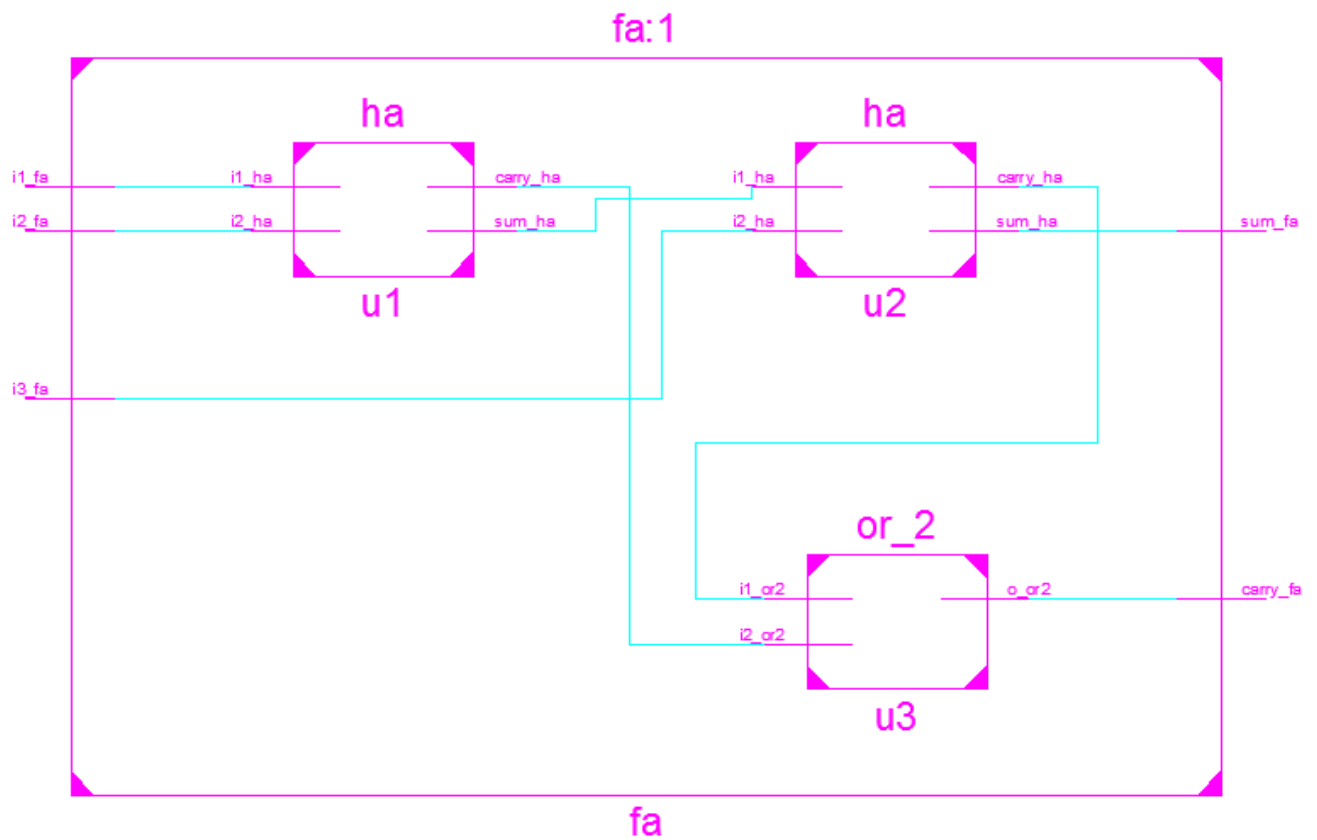
architecture fa_arch of fa is
    component ha
        port(i1_ha, i2_ha : in std_logic;
            sum_ha, carry_ha : out std_logic);
    end component;
    component or_2
        port(i1_or2, i2_or2 : in std_logic;
            o_or2: out std_logic);
    end component;
    signal temp1,temp2,temp3 : std_logic;
begin
    u1:ha port map(i1_ha => i1_fa, i2_ha => i2_fa, sum_ha => temp1, carry_ha => temp2);
    u2:ha port map(i1_ha => temp1, i2_ha => i3_fa, sum_ha => sum_fa, carry_ha => temp3);
    u3:or_2 port map(i1_or2 => temp3, i2_or2 => temp2, o_or2 => carry_fa);
end fa_arch;
```

Stimuli for input ports:

Clock Parameters	Port i1_fa	Port i2_fa	Port i3_fa
Signal Name	/fa/ i1_fa	/fa/ i2_fa	/fa/ i2_fa
Value Radix	Binary	Binary	Binary
Leading Edge Value	1	1	1
Trailing Edge Value	0	0	0
Starting at Time Offset	0	0	0
Cancel after Time Offset	<blank>	<blank>	<blank>
Duty Cycle (%)	50	20	50
Period	10 ns	20 ns	40 ns

Observe the waveform

RTL:

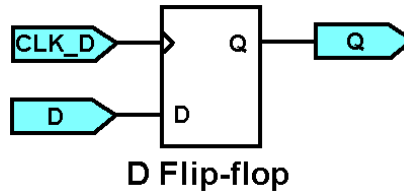


Experiment 8

Aim: Use VHDL to describe the functionality of a “D flip-flop”. Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

Software Package used: Xilinx ISE Design Suite: System Edition 13.2

Block diagram:



VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

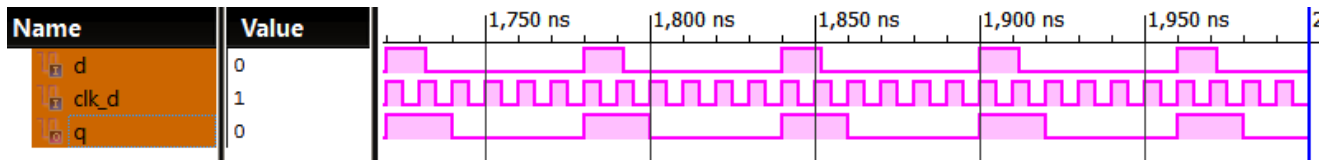
entity D_FF is
    port ( D : in std_logic;
          CLK_D : in std_logic;
          Q : out std_logic);
end D_FF;

architecture D_FF_arch of D_FF is
begin
    process (CLK_D)
    begin
        if (CLK_D'event and CLK_D = '1') then
            Q <= D;
        end if;
    end process;
end D_FF_arch;
```

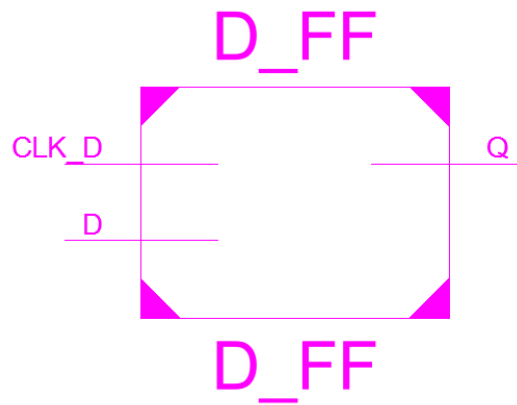
Stimuli for input ports:

Clock Parameters	Port D	Port CLK_D
Signal Name	/D_FF/ D	/ D_FF / CLK_D
Value Radix	Binary	Binary
Leading Edge Value	1	1
Trailing Edge Value	0	0
Starting at Time Offset	0	0
Cancel after Time Offset	<blank>	<blank>
Duty Cycle (%)	20	50
Period	60 ns	10 ns

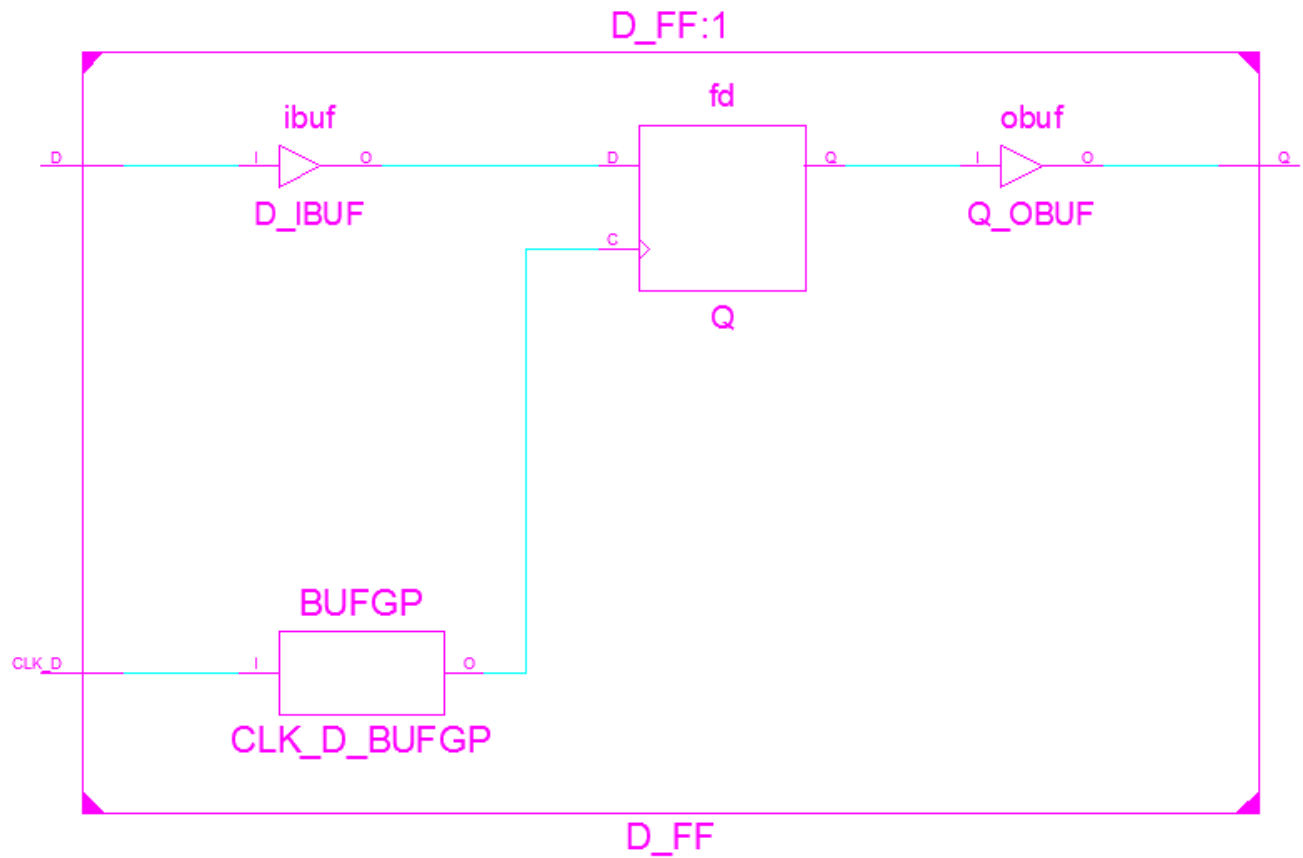
Waveform:



RTL:



Technology Schematic:



Experiment 9

Aim: Use VHDL to describe the functionality of a “D Latch”. Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

Software Package used: Xilinx ISE Design Suite: System Edition 13.2

VHDL code:

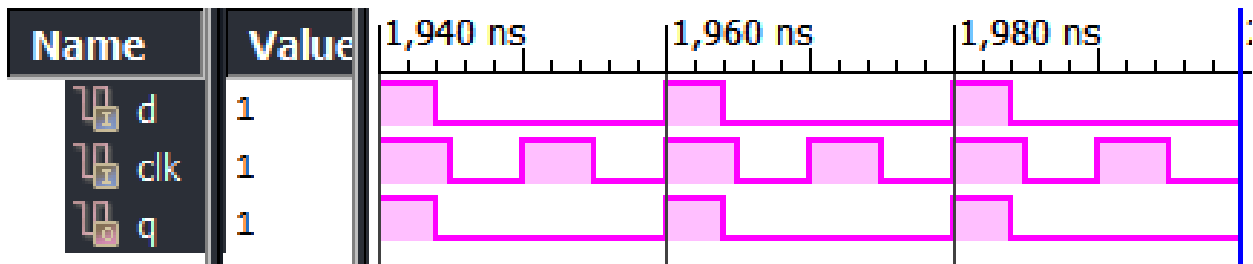
```
entity D_latch is
  Port ( D : in  STD_LOGIC;
        CLK : in  STD_LOGIC;
        Q : out  STD_LOGIC);
end D_latch;

architecture arch_D_latch of D_latch is
begin
  process (CLK, D)
  begin
    if (CLK = '1') then
      Q <= D;
    end if;
  end process;
end arch_D_latch;
```

Stimuli for input ports:

Clock Parameters	Port D	Port CLK
Signal Name	/D_latch/ D	/ D_latch / CLK
Value Radix	Binary	Binary
Leading Edge Value	1	1
Trailing Edge Value	0	0
Starting at Time Offset	0	0
Cancel after Time Offset		
Duty Cycle (%)	20	50
Period	20 ns	10 ns

Waveform:



Experiment 10

[Do It Yourself]

Use 'WHEN' statement to model a:

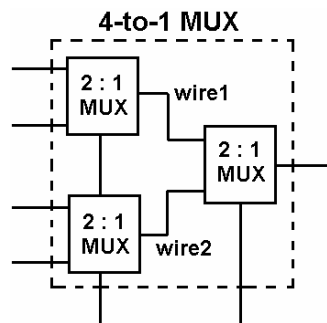
- (a) 2-to-1 Multiplex
- (b) 2-to-4 Decoder
- (c) Tri-State Buffer

Compile and simulate the codes to obtain the timing waveform. Draw the RTL and Technology schematics of the above circuits.

Experiment 11

[Do It Yourself]

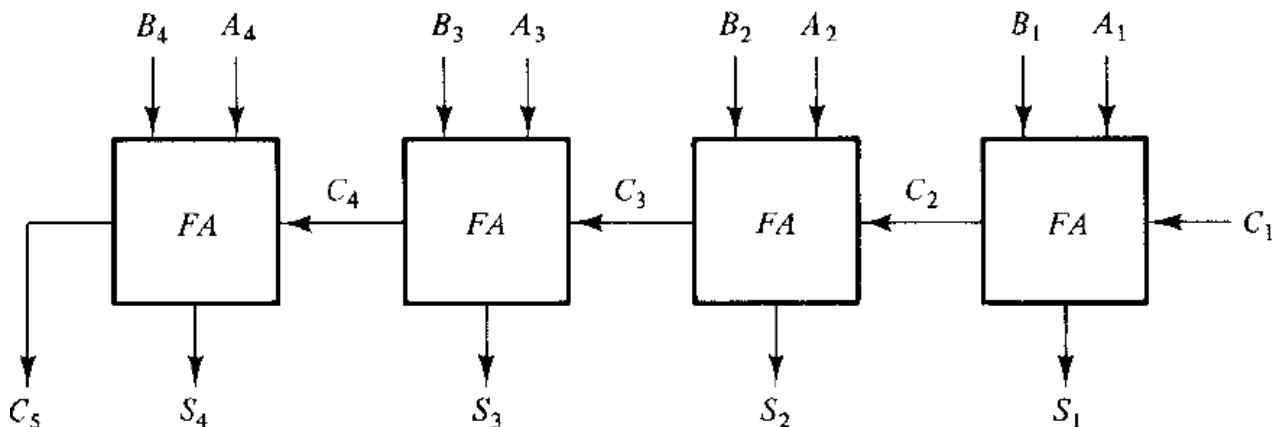
Use a 2-to-1 MUX as a component to model a 4-to-1 Multiplexer using "Component Instantiation". Draw the RTL and Technology schematic of your design.



Experiment 12

[Do It Yourself]

Use four FULL ADDERS as components to model a 4-bit PARALLEL ADDER using "Component Instantiation". Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.



Experiment 13

[Do It Yourself]

Use “Component Instantiation” to model a 4-bit adder-subtractor circuit shown below. Compile and simulate the code to obtain the timing waveform. Draw the RTL and Technology schematic of your design.

