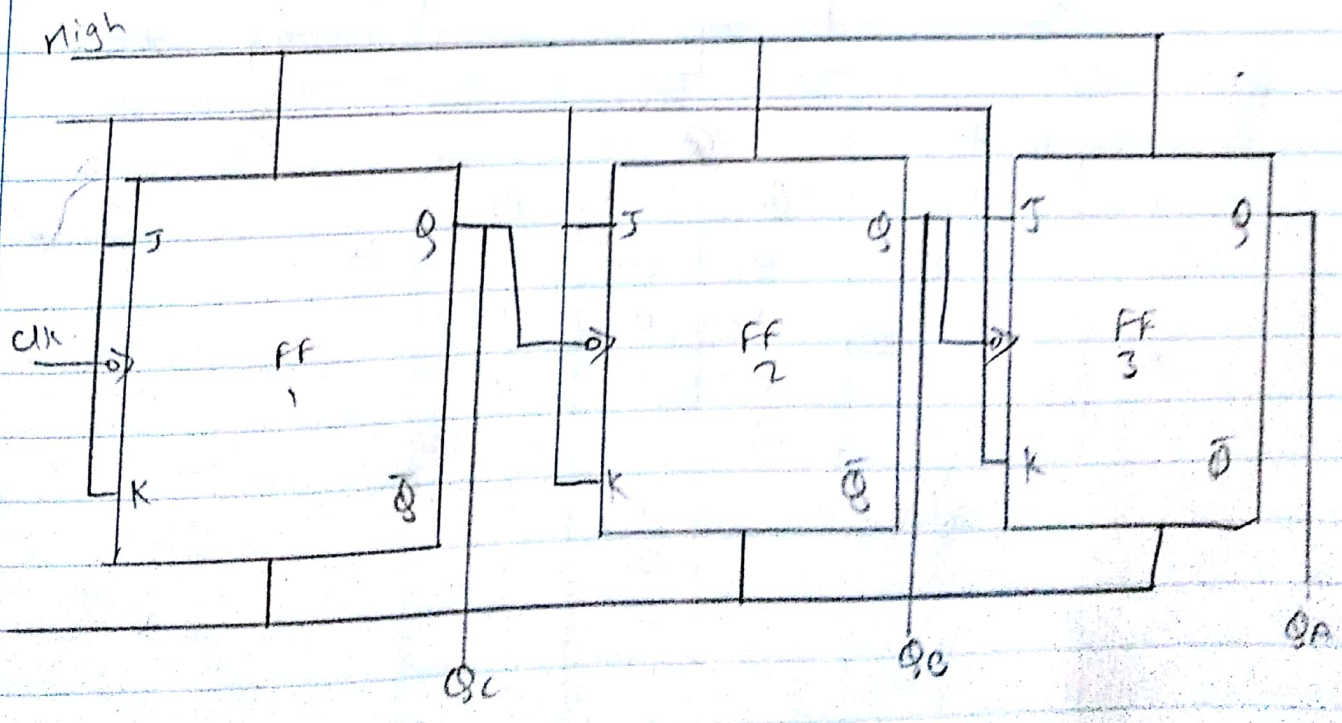
Experiment No. 8.

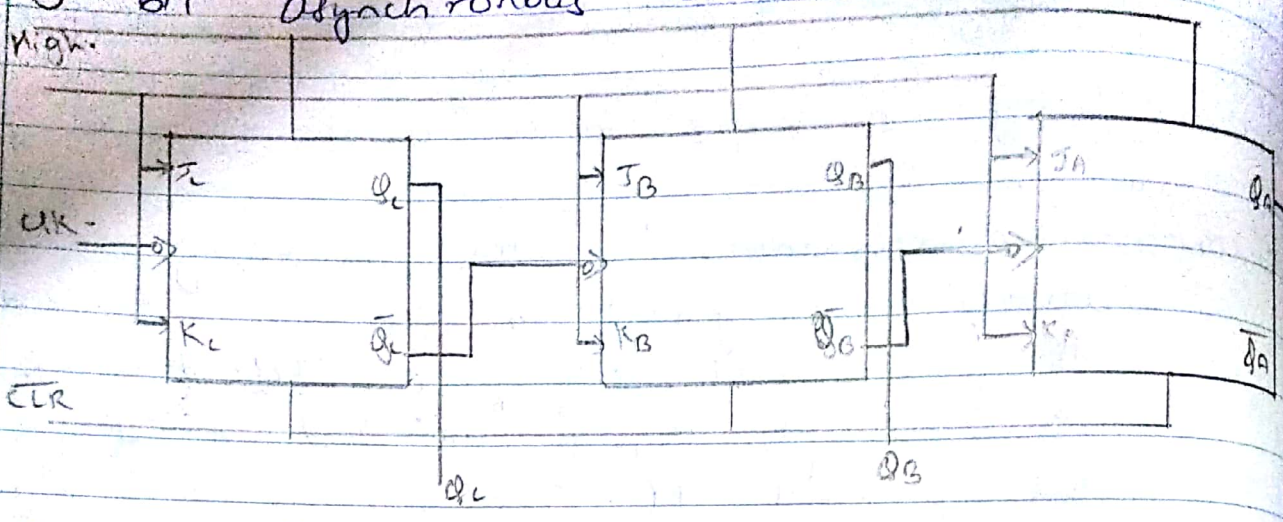
Aim: Design of ripple counter using suitable flip flop.

Theory:- In asynchronous counter commonly called ripple counter, the first flip flop is clocked by the external clock pulse and then each successive flip-flop is clocked by Q or $\bar{Q}$ output the previous FF. Therefore, in an asynchronous ~~CNTR~~ counter the FF are not clocked simultaneously. The output input of MSJK is connected to Vcc because when both inputs are one output is toggled. As MSJK is negative edge triggered at each low transition, the next FF is triggered.

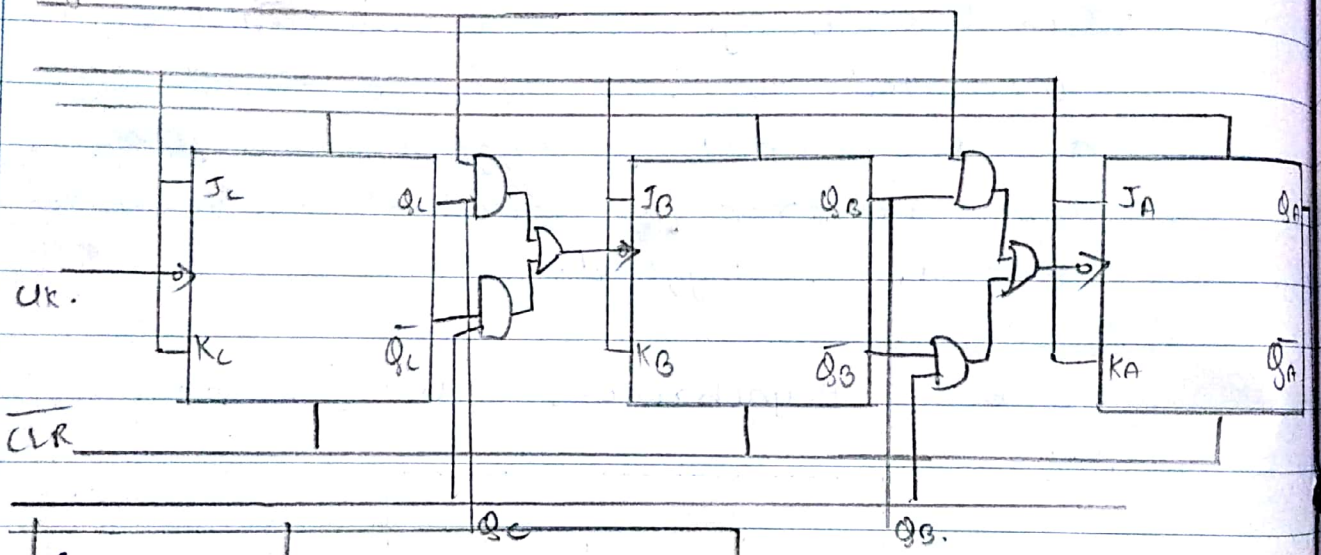
3 bit Asynchronous UP counter.



3 bit Asynchronous Down Counter.



3 bit Asynchronous - Up-Down counter.



Up counting

Counter Status	Count Q_A	Q_B	Q_C
↓ 0	0	0	0
↓ 1	0	0	1
↓ 2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0

Down counter:

Counter status	Count		
	Q_2	Q_1	Q_0
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

Procedure:-

1) Make the connections as shown.
2) Verify the counter operation in different modes.

Conclusion:-

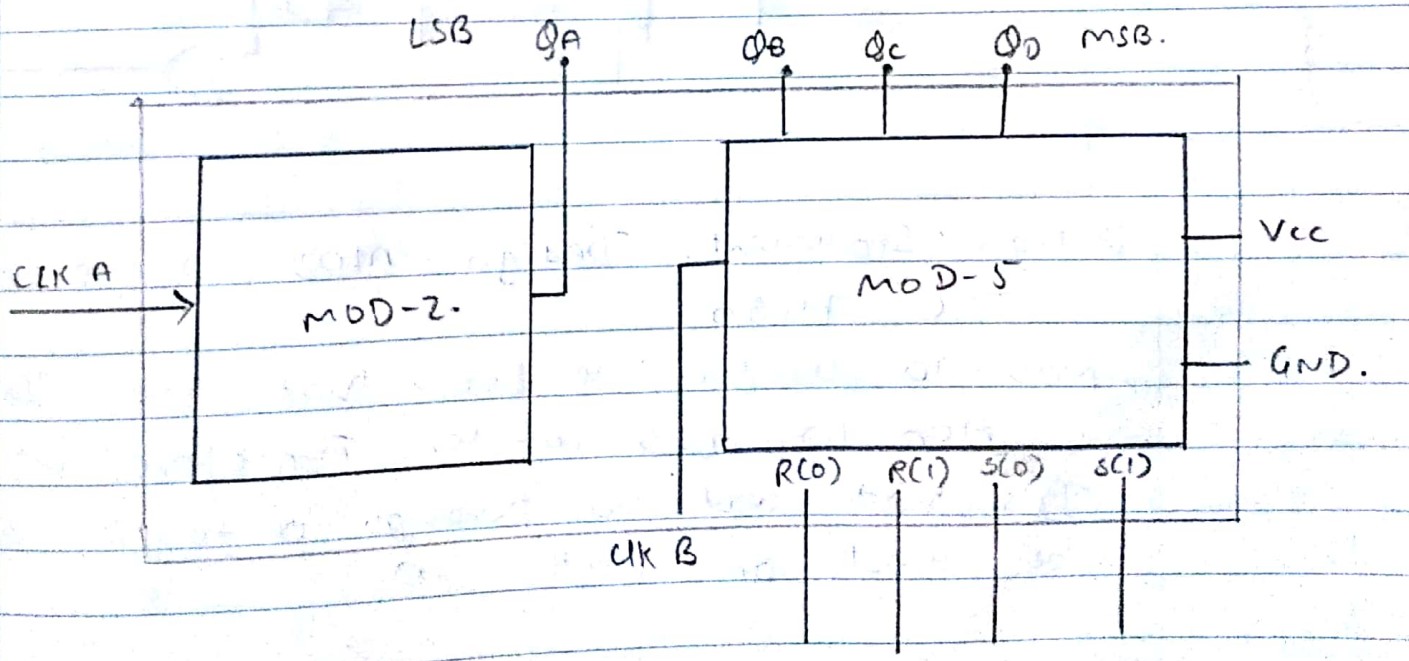
Hence we have studied design & realization of ripple counter using suitable flip flops.

Experiment 9B.

Aim: Realization of mod- N Counter (7490 & 74193).

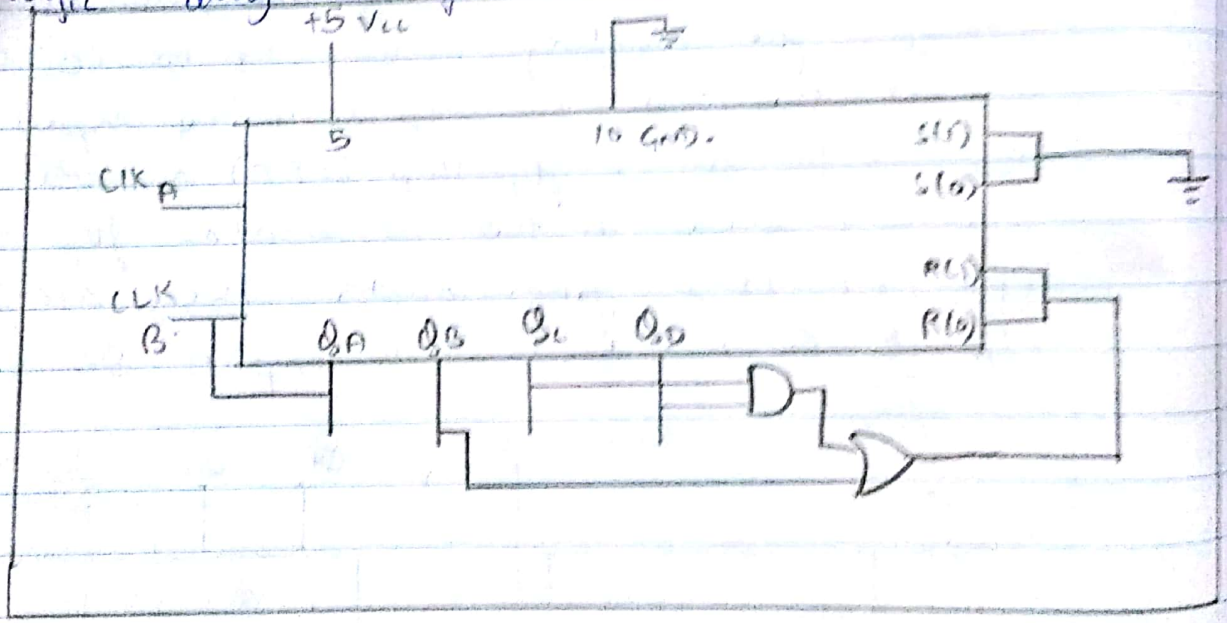
Theory: IC 7490 is a decode binary counter. It consist of four master slave flip flop and additional gating to provide a divide by 2 counter and a three stage binary counter for which the count cycle length is divide by five.

Since the output from the divide by 2 sections is not internally connected to the succeeding stage for counting further up to 10 states connect the MOD-2 output to the input of MOD-5 counter. In fig 2 flip flop (FFA) operates as a MOD-2 counter where as the combination flip flop i.e. FFB, FFC, FFD form counter. There are 2 RESET i/p ie R_0, R_i, & 2 SET inputs S0, S1.



Reset input				Output			
$R_0(1)$	$R_0(2)$	$R_9(1)$	$R_9(2)$	Q_D	Q_C	Q_B	Q_A
1	1	0	X	0	0	0	0
1	1	X	0	0	0	0	0
X	X	1	1	1	0	0	1
X	0	X	X	Counter			
0	X	0	X	counter			
0	X	X	0	counter			
X	0	0	X	counter			

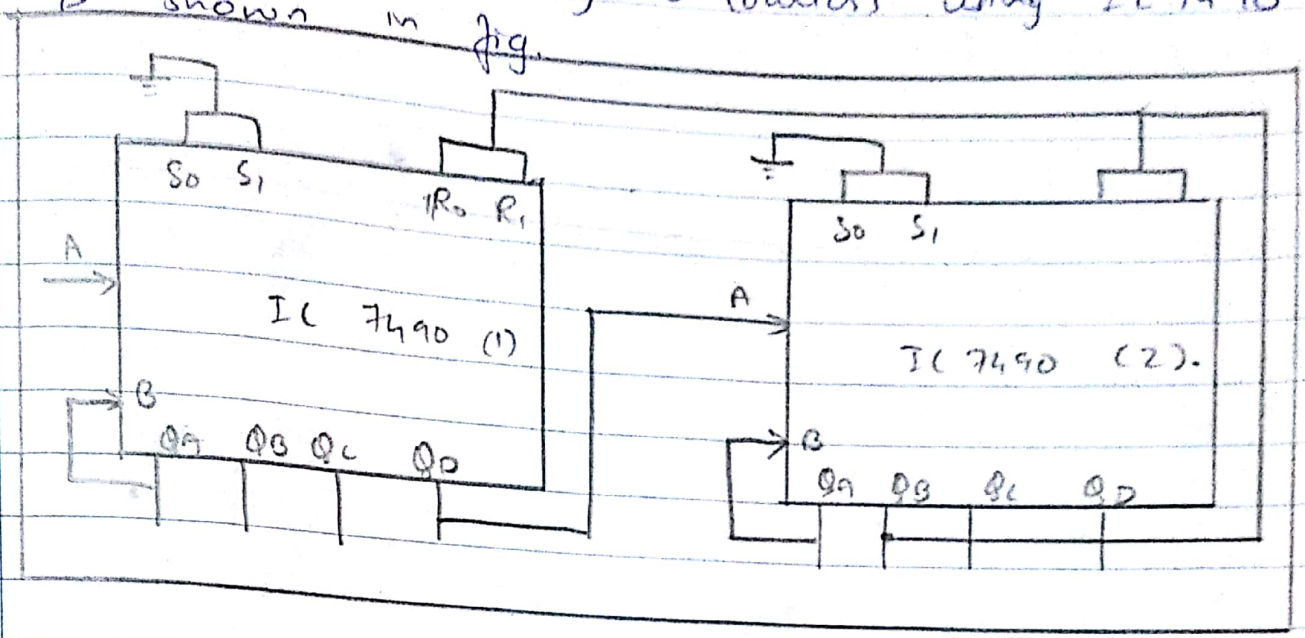
logic diagram for MOD-6



Design Statement: Design MOD-20 counter, using IC 7490

MOD-20 counter: We know that one IC work mod 10 BCD counter. Therefore we need 2 ICs that will go through 0 to 19 and then be reset on state 20.

Diagram of divide by 20 counters using IC 7490 is shown in fig.



Procedure :-

1) Firstly make the connection for MOD-6 counters as shown in diagram.
2) Make connection for MOD-20 counter as shown in diagram.

Conclusion :-

Hence we have studied MOD-N counter using 7490.

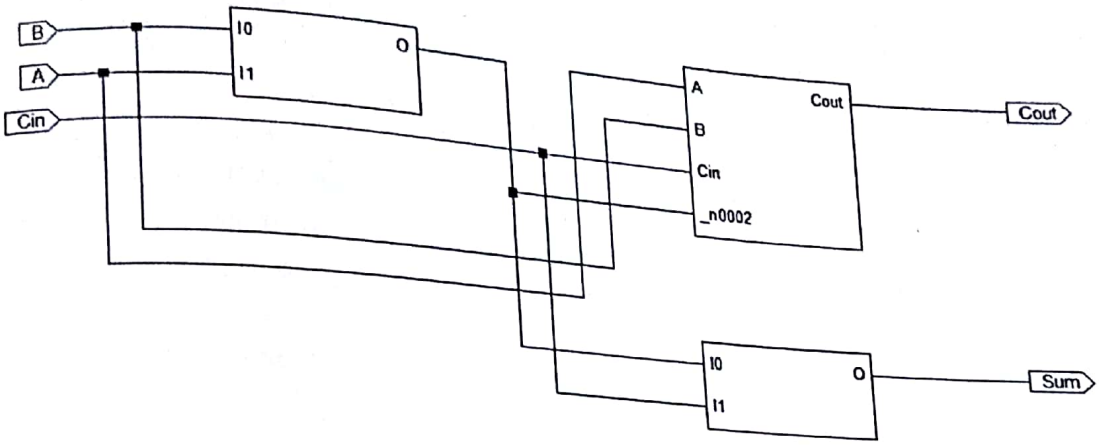
Assignment No- ~~13~~ 12

i. Full adder:-
Code:-

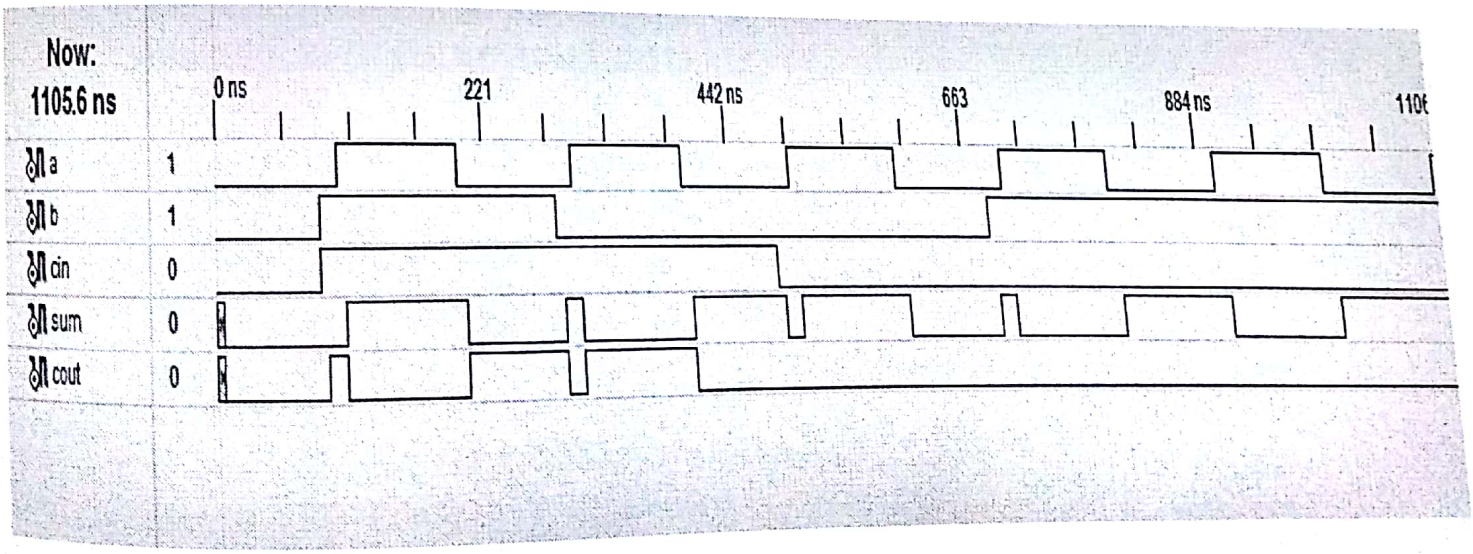
-

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entityfulladder is
   Port ( A : in  STD_LOGIC;
B : in  STD_LOGIC;
Cin : in  STD_LOGIC;
Sum : out  STD_LOGIC;
Cout : out  STD_LOGIC);
endfulladder;

architecture Behavioral of fulladder is

begin
sum<= a xor b xorcin;
cout<= (a xor b) and (cin or ( a and b));

end Behavioral;
```

RTL Schematic:-



Output:-



ii. Multiplexer

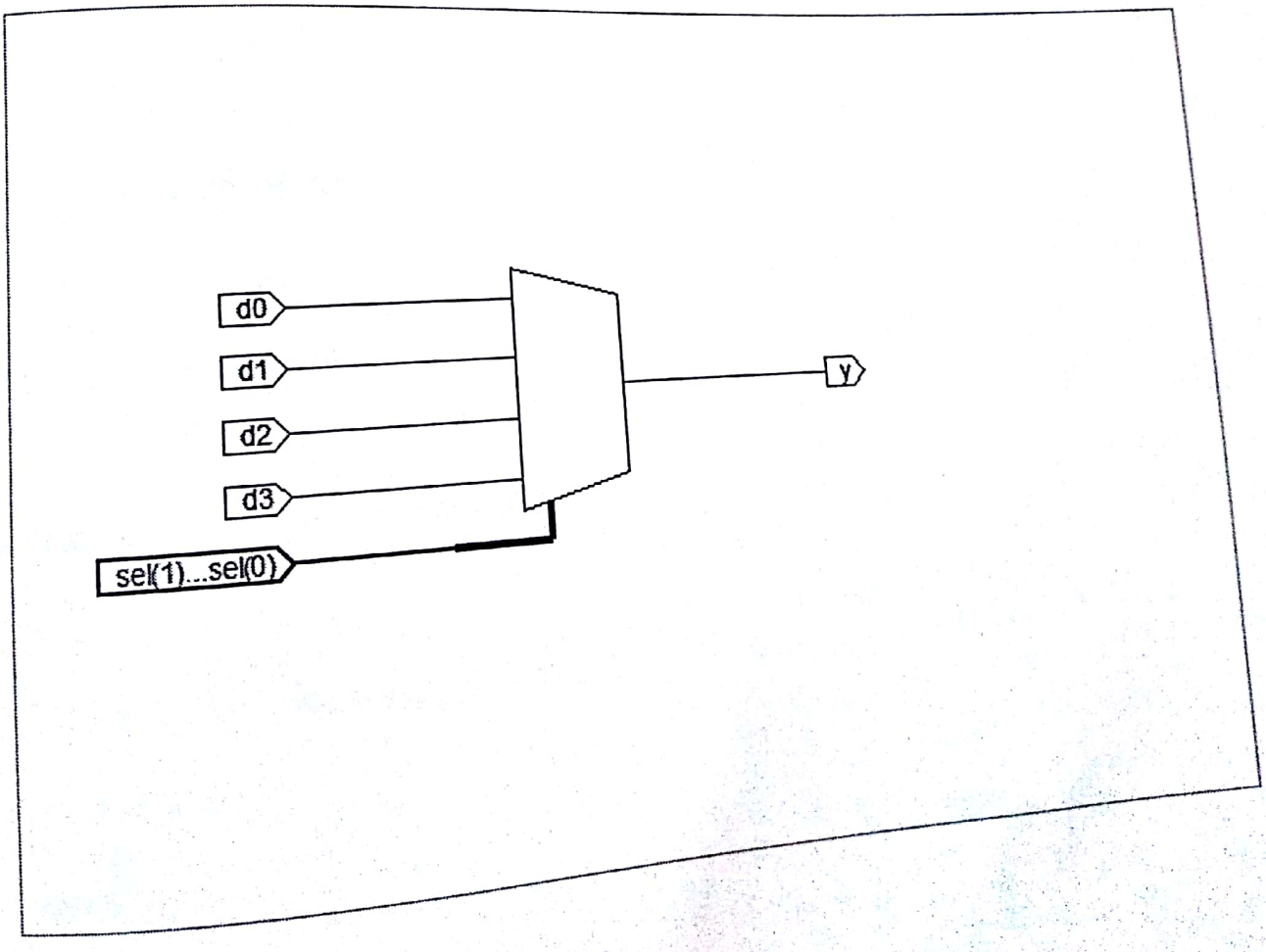
Code:-

i. using If statement

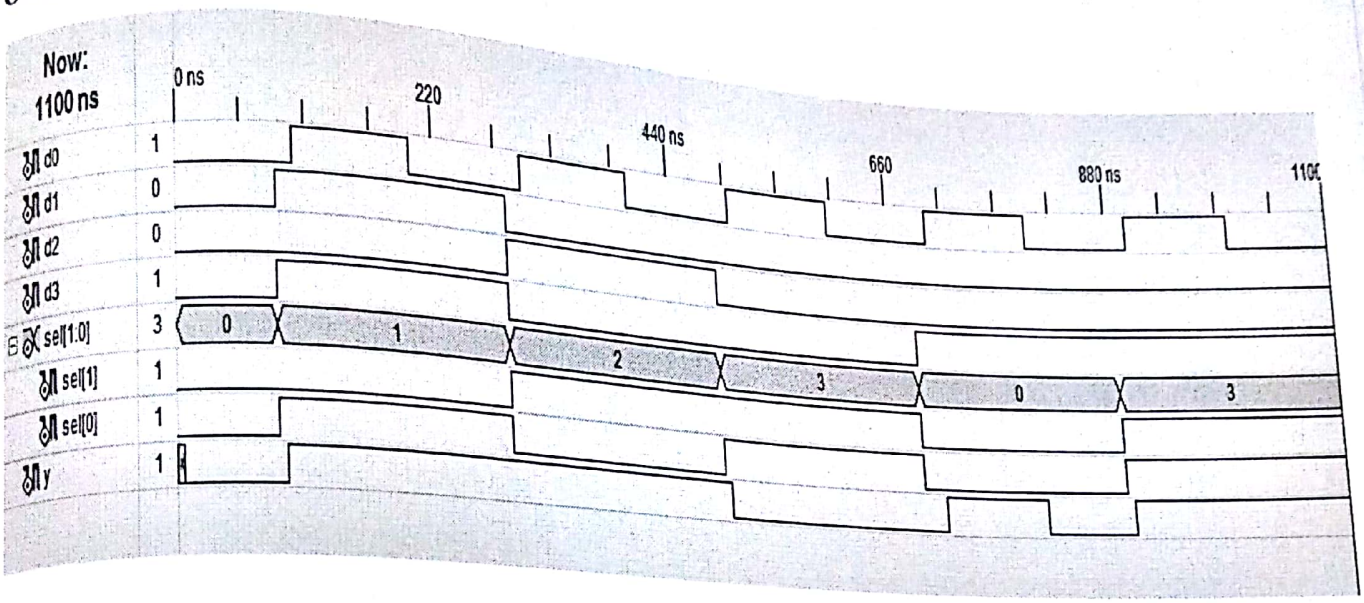
```
- Company:
-- Engineer:
--
-- Create Date:    09:56:39 09/25/2016
-- Design Name:
-- Module Name:    mux3 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux3 is
   Port ( d0 : in  STD_LOGIC;
d1 : in  STD_LOGIC;
d2 : in  STD_LOGIC;
d3 : in  STD_LOGIC;
sel : in  STD_LOGIC_VECTOR (1 downto 0);
y : out  STD_LOGIC);
```

```
end mux3;

architecture Behavioral of mux3 is

begin
process(sel,d0,d1,d2,d3)
begin
if (sel=0) then
y<= d0;
elsif (sel=1) then
y<=d1;
elsif(sel=2) then
y<=d2;
else
y<=d3;
end if;
end process;


end Behavioral;
```

RTL Schematic:-



Output:-



ii.Using CASE statement

- Company:

-- Engineer:

--

-- Create Date: 10:05:52 09/25/2016

-- Design Name:

-- Module Name: muxcase - Behavioral

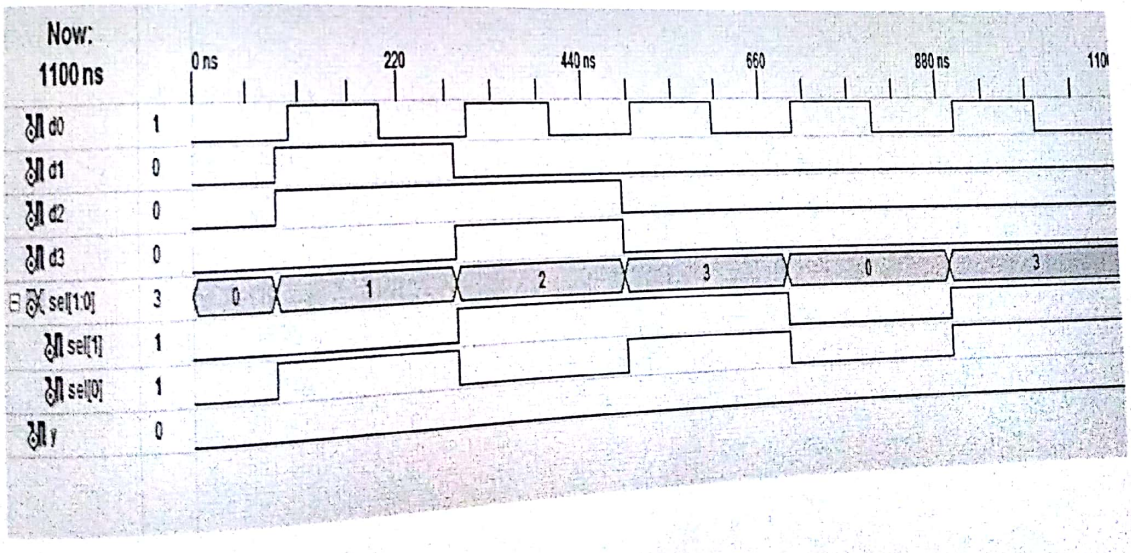
-- Project Name:

-- Target Devices:

```
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entitymuxcase is
   Port ( d0 : in  STD_LOGIC;
d1 : in  STD_LOGIC;
d2 : in  STD_LOGIC;
d3 : in  STD_LOGIC;
```

```
sel : in  STD_LOGIC_VECTOR (1 downto 0);
y : out  STD_LOGIC);
endmuxcase;

architecture Behavioral of muxcase is
begin
process (Sel,D0,D1,D2,D3 )
begin
caseSel is
when"00" => Y <= D0;
when"01" => Y <= D1;
when"10" => Y <= D2;
when others => Y <= D3;
end case;
end process;

end Behavioral;
```

Output:-



Assignment No - ~~14~~ 13

Title:- Asynchronous Counter

Aim: - Design & simulate asynchronous 3- bit counter using VHDL

i. **Up Counter**

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

--- Uncomment the following library declaration if instantiating
--- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter is
  Port ( clock : in  STD_LOGIC;
clear : in  STD_LOGIC;
count : in  STD_LOGIC;
q : out  STD_LOGIC_VECTOR (2 downto 0));
end counter;

architecture Behavioral of counter is
signal value: std_logic_vector (2 downto 0);
begin
```

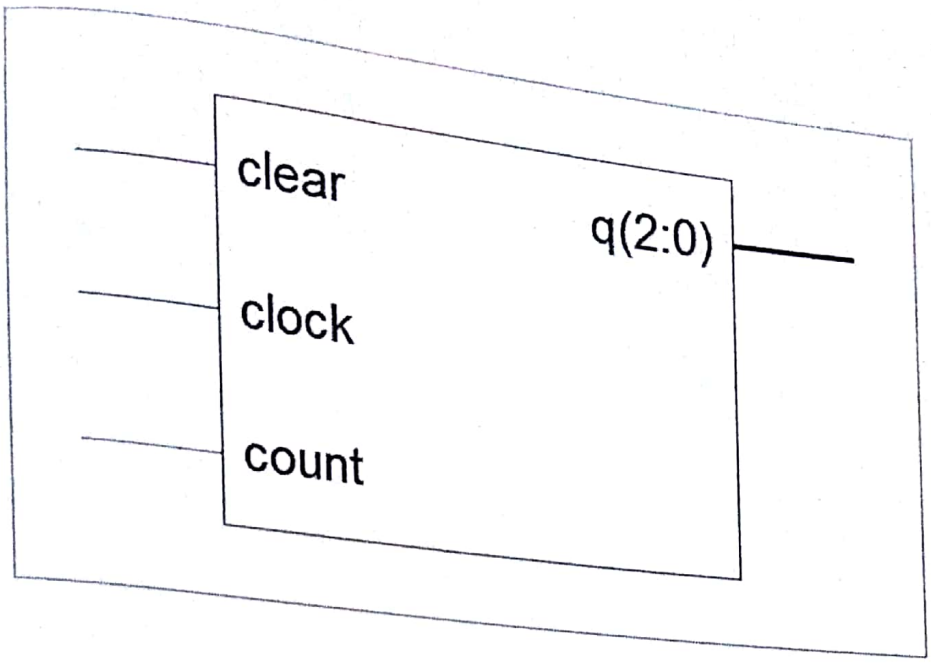
```
process( clock, clear)
        begin
                if (clear='1') then
                        value<= "000";
                elsif ( clock'event and clock='1') then
                        if ( count='1' ) then
                                value<= value + 1;
                        end if;
                end if;
        end process;
q <= value;

end Behavioral;
```

ii. Down counter

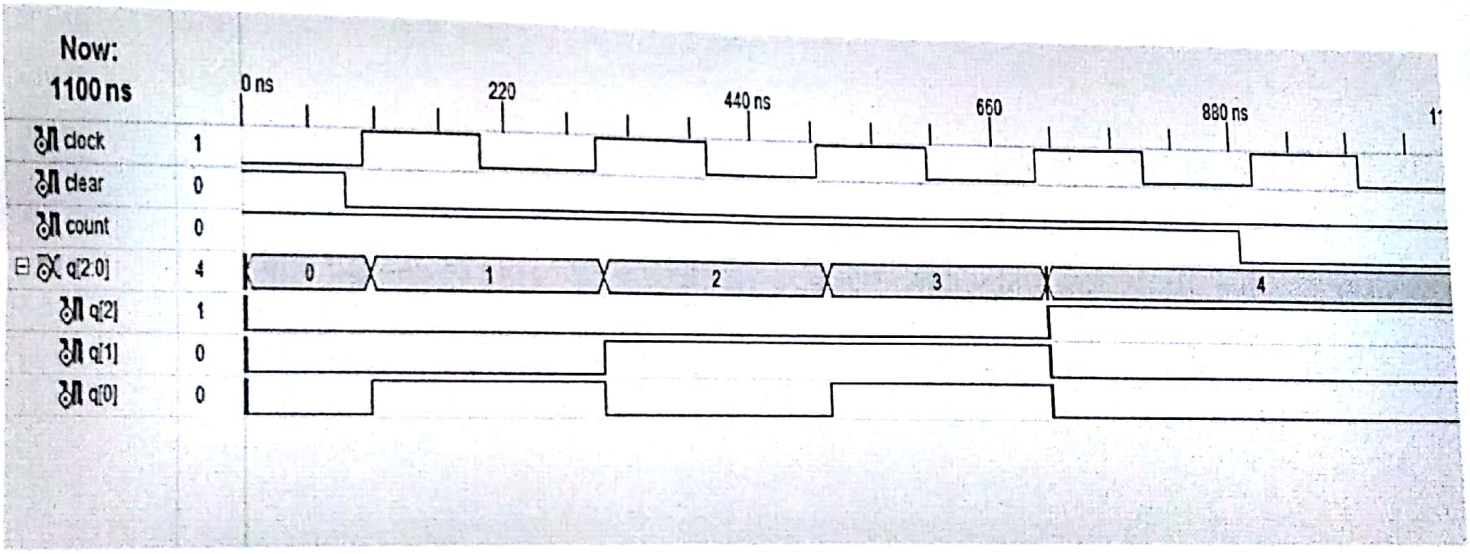
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

```
entity counter is
   Port ( clock : in  STD_LOGIC;
clear : in  STD_LOGIC;
count : in  STD_LOGIC;
q : out  STD_LOGIC_VECTOR (2 downto 0));
end counter;

architecture Behavioral of counter is
signal value: std_logic_vector (2 downto 0);
begin
process( clock, clear)
        begin
                if (clear='1') then
                        value<= "000";
                elsif ( clock'event and clock='1') then
                        if ( count='1' ) then
                                value<= value  - 1;
                        end if;
                end if;
        end process;
q <= value;

end Behavioral;
```

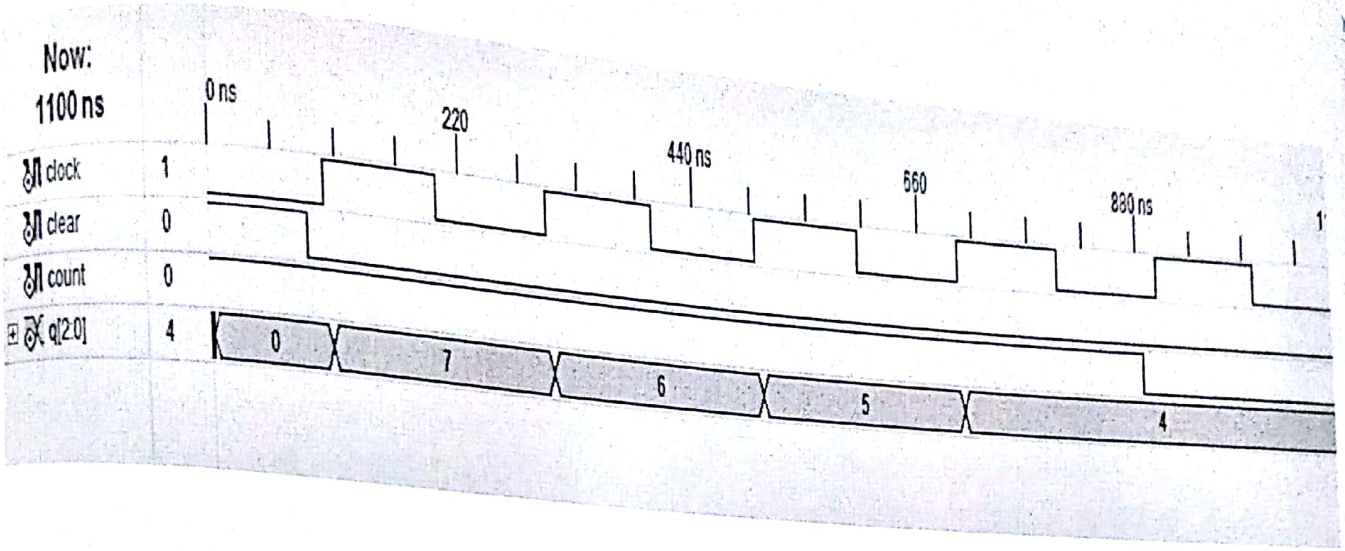
RTL Schematic:-



Up counter output:-



Down counter output:




```
entity counter_2bit is
    port(
       clk : in STD_LOGIC;
       reset : in STD_LOGIC;
       dout : out STD_LOGIC_VECTOR(1 downto 0)
       );
end counter_2bit;

architecture counter_2bit_arc of counter_2bit is
begin

    counting : process (clk,reset) is
    variable m : std_logic_vector (1 downto 0) := "00";
    begin
       if (reset='1') then
          m := "00";
       elsif (rising_edge (clk)) then
          m := m + 1;
       end if;
       dout<= m;
    end process counting;

  end counter_2bit_arc;
```

Experiment No 14.

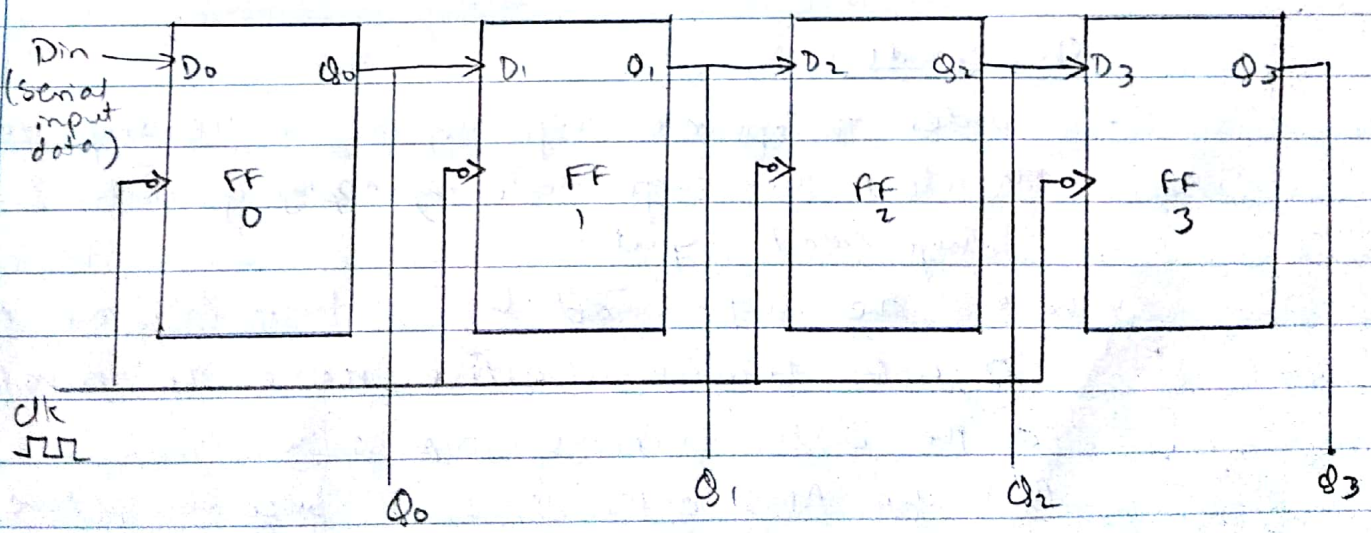
Title: Study of Shift Registers (SISO, PISO, SIPO and PIPO).

Theory:-

① SIPO [Serial in Parallel Out]:-

- In this operation the data is entered serially & taken out in parallel.
- That means first the data is loaded bit by bit. The outputs are disabled as long as the loading is taking place.
- As soon as the loading is complete, and all the flip flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines simultaneously.
- No. of clock cycles reqd. to load a 4 bit word is 4. Hence the speed of operation of SIPO mode is same as SISO.

Diagram:-



② Parallel in Serial Out (PISO):-

- In this mode, the bits are entered in parallel i.e. simultaneously into a shift register, as shown in the figure.
- The circuit shown below is a 4 bit PISO register.
- Output of previous ff is connected to input of next one via a combinational circuit.
- There are 2 modes in which this circuit can work namely shift mode or load mode.

Load mode:-

- In order to load the word $B_3B_2B_1B_0$ into shift register we have to select the load mode by setting shift/$\overline{load}$ input to 0.
- When shift/$\overline{load}$ line is low (0), the AND gates 2, 4 and become active. They will pass B_1, B_2 and B_3 bits to inputs D_1, D_2 & D_3 of the corresponding flip flops. D_0 is directly connected to B_0.
- On the low going edge of clock, the binary inputs $B_0B_1B_2B_3$ will get loaded into the corresponding flip flops. Thus parallel loading takes place.

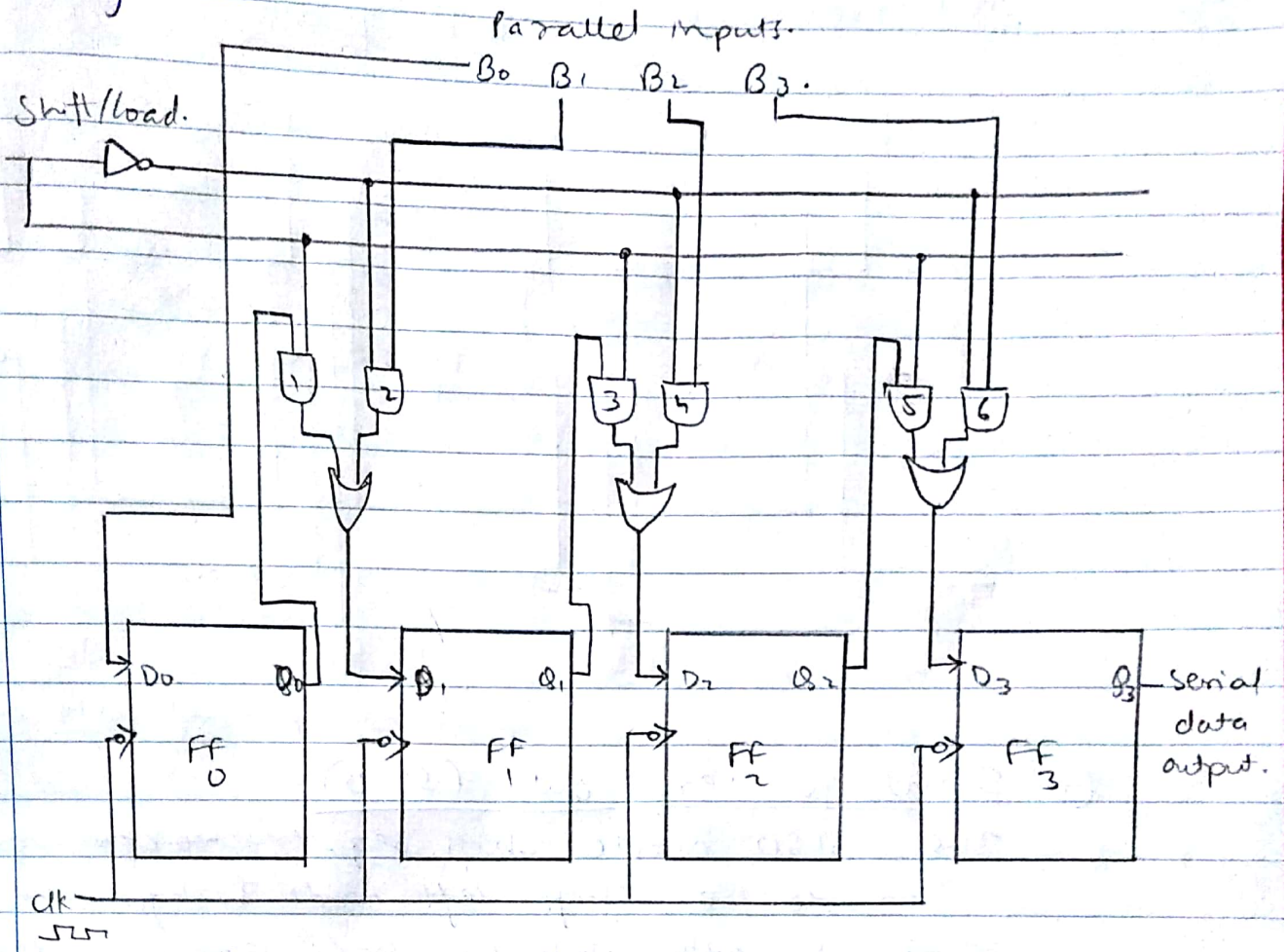
Shift mode:-

- In order to operate shift register in the shift mode we have to select the shift mode by applying logic 1 to the shift/$\overline{load}$ input.
- When the shift/$\overline{load}$ line is high (1), the AND gates 2, 4, 6 become inactive. Hence the parallel loading of the data becomes impossible.
- But the AND gates 1, 3, 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses becomes possible. D0

acts as the data input terminal & at Q_3 we get serial data output.

- Thus the parallel in serial out operation takes place.

Diagram:-

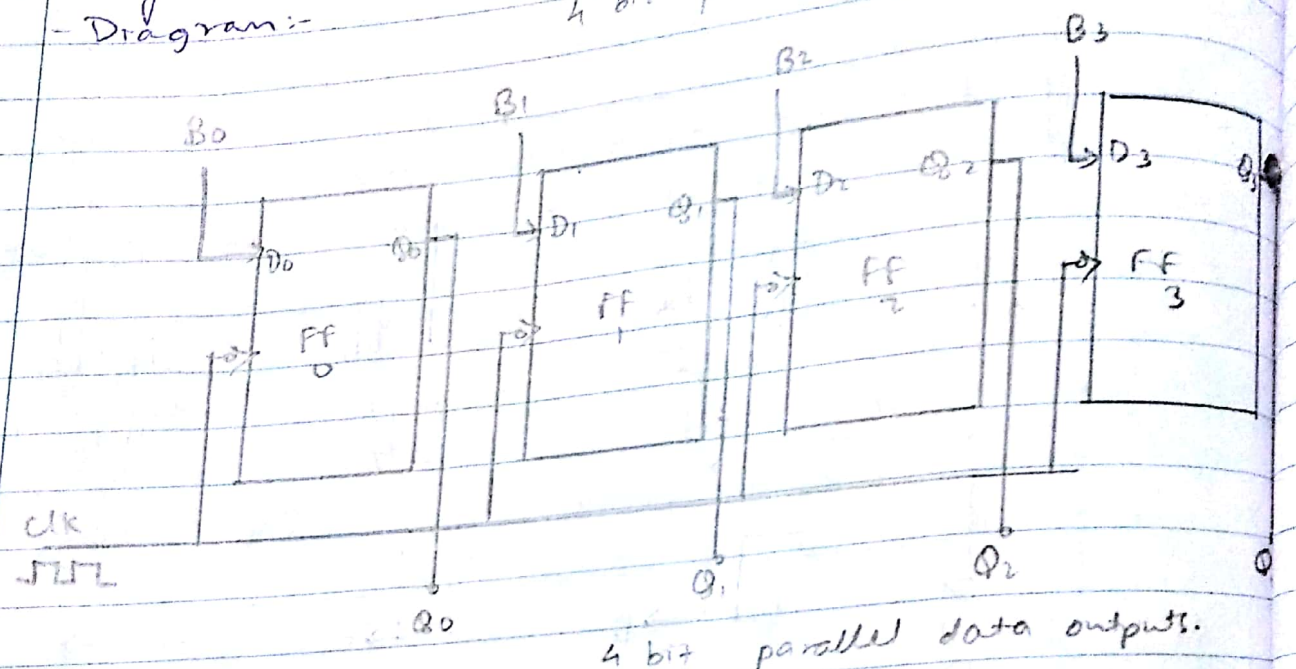


③. Parallel in parallel out (PIPO):-

- The figure demonstrates the parallel in parallel out mode of operation.
- The 4 bit binary input B_0 B_1 B_2 B_3 is applied to the data inputs D_0 D_1 D_2 D_3 respectively of 4 flip flops.
- As soon as negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously.

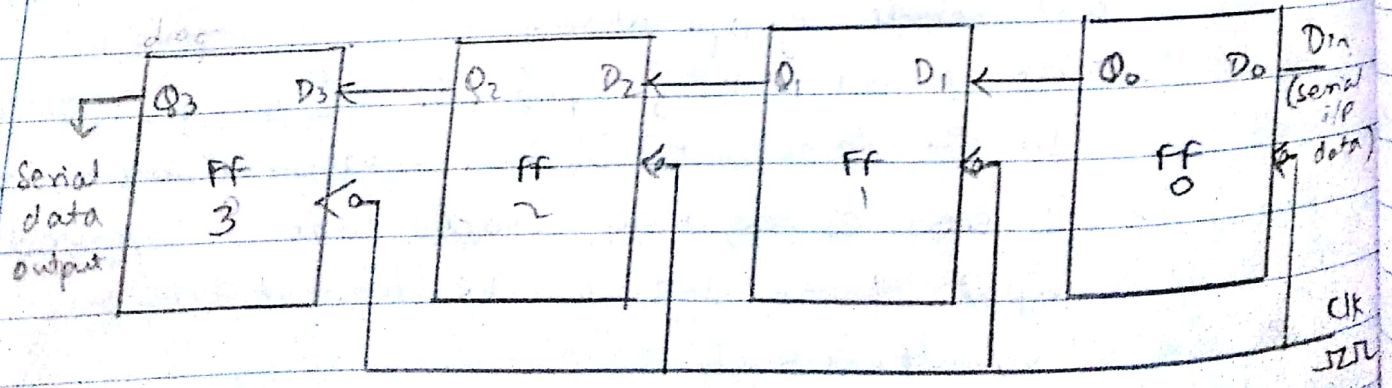
- The loaded bits will appear simultaneously to the output side. Only one clock pulse is essential to load all the bits. Therefore PIPO mode is the fastest mode of operation.

- Diagram:-



(4) Serial in serial out (SISO):-

- The SISO mode shift register can work in 2 mode i.e. shift left mode & shift right mode.
- The SISO with shift left mode is shown below:-
- Let all the flip-flops be initially in reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0$.



- we are going to illustrate the entry of a 4 bit binary number 1111 into register.
- When this is to be done, this number should be applied to 'Din' bit by bit with the MSB bit applied first.
- The D input of FF-0 i.e. D_0 is connected to serial data input (D_{in}). Output of FF0 i.e. Q_0 is connected to the input of next flip flop i.e. D_1 & so on...

Operation:-

- Before application of clock signal, let $Q_3Q_2Q_1Q_0 = 0000$ & apply MSB bit of the number to be entered to D_{in}. So $D_{in} = D_0 = 1$.
- Apply the clock. On the 1st falling edge of clock, the FF-0 is set and stored word in the register is

$$Q_3Q_2Q_1Q_0 = 0001.$$

- Apply next bit to D_{in}. So $D_{in} = 1$.
- As soon as next negative edge of clock hits, FF-1 will set & the stored word changes to

$$Q_3Q_2Q_1Q_0 = 0011.$$

- Apply the next bit to be stored, i.e. 1 to D_{in}.
- Apply the clock pulse. As soon as the third negative clock edge hits, FF-2 will be set & the output gets modified to

$$Q_3Q_2Q_1Q_0 = 0111.$$

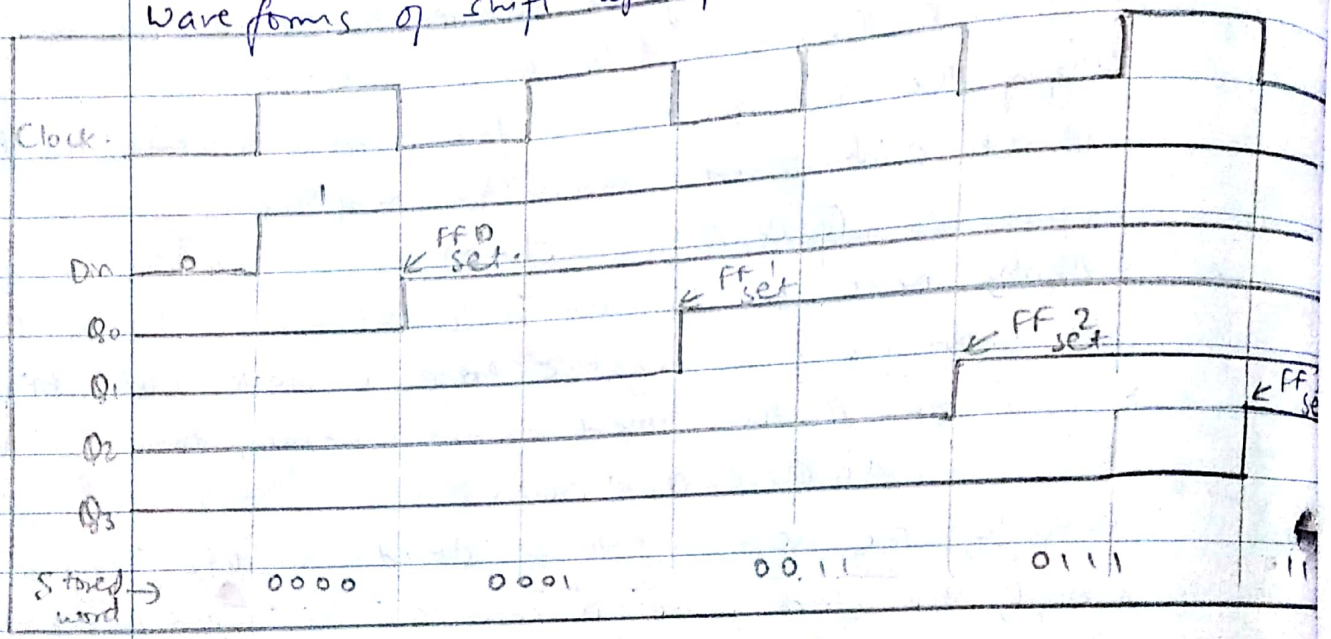
- Similarly with $D_{in} = 1$ and with the 4th negative clock edge arriving, the stored word in the register is $Q_3Q_2Q_1Q_0 = 1111$.

Summary of shift left operation:-

	Clk	Q_3	$Q_2 = D_3$	$Q_1 = D_2$	$Q_0 = D_1$	Serial input $D_{IN} = D_0$
Initially.		0	0	0	0	1
1st	↓	0	0	0	1	1
2nd	↓	0	0	1	1	1
3rd	↓	0	1	1	1	1
4th	↓	1	1	1	1	1

direction of data travel = ←.

Waveforms of shift left operation:-



-Using SISO mode, we needed 4- clock pulses to store 4-bit word. So in general we can conclude that it requires n numbers of clock pulses to store an n bit word using SISO mode. Shift right works totally opposite.

Conclusion:- Hence we studied different types of shift registers.

Experiment No.15

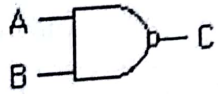
Problem Statement:
Study of TTL Logic Family: Feature, Characteristics and Comparison with CMOS Family

Theory:

TTL FAMILY

The *logic family* refers to the general physical realization of a logical element, such as the TTL, emitter-coupled logic (ECL), or complementary metal -oxide semiconductor (CMOS) logic families. Within each logic family are one or more *logic series* that have distinctive characteristics, relative to other series within the same logic family. For example, in the TTL logic family, there are several logic series: the 74 standard, 74L low-power, 74H high-speed, 74S standard Schottky, 74LS low- power Schottky series, and 74ALS advanced low-power Schottky series.

The TTL family was the most widely used logic family for several years, characterized by its relatively high speed operation. However, it has now been largely replaced by CMOS logic. The physical representation of the binary logic states in these families are high and low voltages, as described in Experiment 1. Assuming positive logic, in the 74LS TTL family LOW (L) voltages in the range 0 V to 0.8 V are considered to be logic 0, and HIGH (H) voltages in the range 2.0 V to 5.5 V are considered to be logic 1. Figure 3.1 illustrates the voltage levels for all possible input combinations to a two-input TTL NAND gate.



A	B	C
L	L	H
L	H	H
H	L	H
H	H	L

Figure Voltage Level Table for a Two-input TTL NAND Gate.

You may wonder why the NAND gate is so popular in the TTL logic families. Perhaps the most important factor in the use of such gates is the presence of transistors. Transistors are *active* devices that tend to restore signal levels and preclude signal deterioration which could cause 1 and 0 become indistinguishable. No degradation of signal levels occurs, even for long chains of TTL NAND gates. In the TTL family the number of transistors required to implement a NAND gate is less than that required to implement

other gates such as AND, OR and NOR. Another factor in favor of NAND gates is the fact that any combinational logic function can be realized using just NAND gates.

TTL CHARACTERISTICS

Each logic family is characterized by several important parameters. These properties, and how they relate to the TTL logic family in particular, are explained below:

Fan-in is the maximum number of inputs to a gate. Although physical considerations limit fan-in, more pragmatic factors, such as limitations on the number of pins possible on IC packages and their standardization predominate. TTL NAND gates typically provide 1, 2, 4, or 8 inputs. If more than eight inputs are required, then a network of NAND gates must be employed.

Fan-out specifies the number of standard loads that the output of a gate can drive withoutimpairing its normal operation. A standard load is defined to be the amount of current required to drive an input of another gate in the same logic family. Due to the nature of TTL gates, two different fanout values are given, one for HIGH outputs and one for LOW outputs. Typically when an input is at logic 1 at most 40μA flows into an input ($I_{IH(max)}$, see Lab 1), and it must be provided (sourced) by the driving output. For logic 0, at most 1.6 mA flows from the input (I_{ILmax}), which the driving output must "sink". By convention the current flowing into an input or output is considered positive while a current flowing out of an input or output is considered negative; hence, I_{ILmax} = -1.6mA. A typical TTL gate can source 400 μA ($I_{OH(max)}$) of current and can sink 16 mA ($I_{OL(max)}$). Hence TTL gates typically have a HIGH (logic level) fanout of

$$|I_{OH(max)}/I_{IH(max)}| = |-400\ \mu A / 40\ \mu A| = 10,$$

and a LOW fanout of

$$|I_{OL(max)}/I_{IL(max)}| = |16\ mA / -1.6\ mA| = 10.$$

Exceeding these fan-out limits may result in incorrect voltage levels at the output, as a gate cannot provide or sink enough current. The lower value of the two fanout values determines the fanout of the gate. In the case of 7400 TTL logic, they are equal, but for some other types of TTL logic the limiting value is the LOW fanout. Some TTL structures have fan-outs of at least 20 for both logic levels.

A **voltage transfer curve** is a graph of the input voltage to a gate versus its output voltage; Figure 3.2 shows the transfer curve for TTL inverter without any fanout. When the input voltage is 0 V, the output is HIGH at 3.3 V. As the input voltage is increased from 0 to 0.7 V, the output remains relatively constant (Region I). Beyond 0.7 V to about 1.2 V, the output decreases more gradually with increasing input voltage (Region II) .

The *threshold voltage*, the voltage on the transfer curve at which $V_{out} = V_{in}$ and occurs in Region III, is found at the intersection of the

transfer curve and the line $V_{out} = V_{in}$. Finally, in Region IV, the output remains constant at 0.2 V as the input voltage is increased.

Noise immunity is a measure of the ability of a digital circuit to avert logic level changes onsignal lines when noise causes voltage level changes. (See Figure 3.3.) One measure of noise immunity is characterized by a pair of parameters: the dc HIGH and LOW *noise margins*, DC1 and DC0, respectively. They are defined as follows:

$$DC1 = V_{OH(min)} - V_{IH(min)}$$

and

$$DC0 = V_{IL(max)} - V_{OL(max)}.$$

The minimum values of DC1 and DC0 determine worst case noise margin.

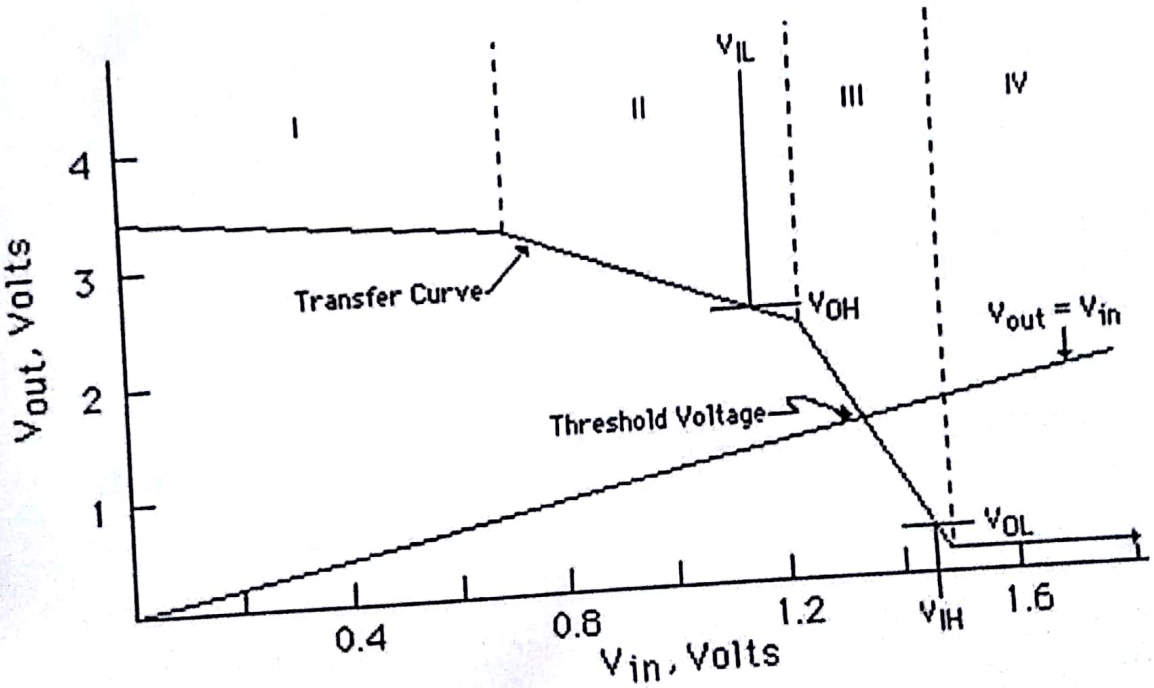


Figure Voltage Transfer Curve for a TTL Gate.

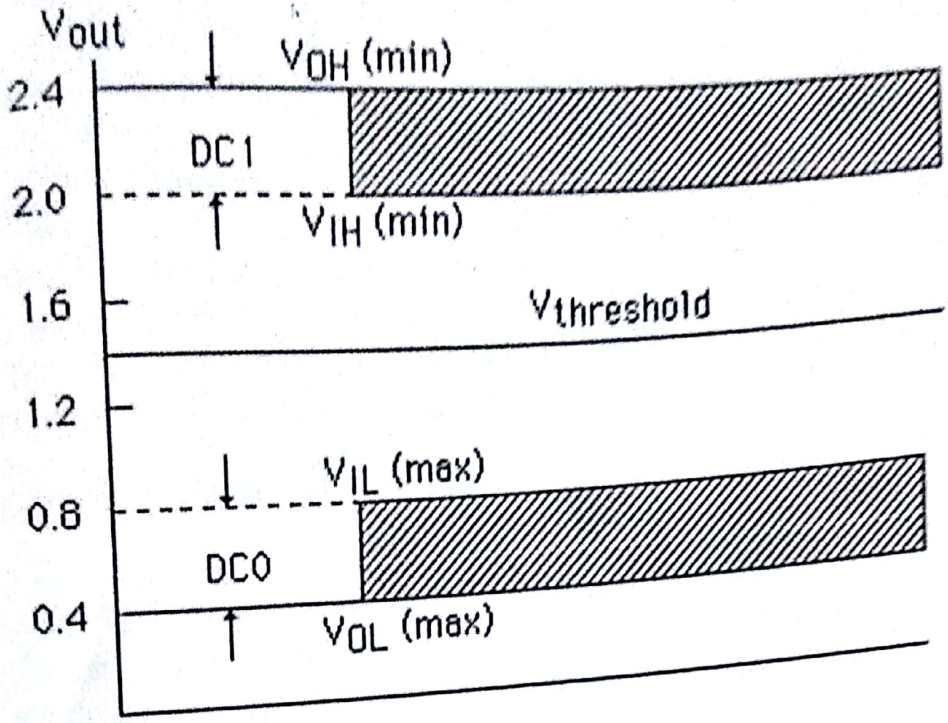


Figure Input and Output TTL Voltage Levels Illustrating DC Noise Margin.

Recall from Experiment 1 that $V_{OL(max)}$ is the largest voltage that can occur on the output of a gate when the output is in the LOW (logic 0) voltage range. $V_{OH(min)}$ is the smallest voltage that can occur on a gate output when the output is in the HIGH (logic 1) voltage range output. In addition, $V_{IH(min)}$ is the smallest input voltage that will be interpreted as a HIGH input and $V_{IL(max)}$ is the greatest input voltage that will be interpreted as a LOW input. Note that V_{OL} and V_{OH} are specified by the manufacturer on the data sheets for the device. V_{IL} and V_{IH} can be determined, given V_{OL} and V_{OH}, from the voltage transfer curve, as indicated in Figure 1 . Illustrated in Figure 2 are the DC0 and DC1 noise margins. For the TTL family, typically

DC0=0.8 - 0.4 = 0.4 V, and DC1 = 2.4 - 2.0 = 0.4 V. (The values for V_{OL}, V_{OH}, V_{IL}, and V_{IH} were obtained from the specification sheet of the 7400 NAND gate.) Any

disturbance with a negative peak of up to DC1 could appear superimposed on the worst case HIGH voltage level, V_{OH}, on an input without disturbing the gate output. A similar statement can be made for DC0.

The **propagation delay time** for a gate is the time required for the output to respond to a change in an input. In all practical gates, a time lag exists between an input change and the corresponding output response. The time interval between the instants when the input and output change states is not a satisfactory measure of the delay time of a logical device for two reasons. First, the input signals to gates and the output signals produced by gates are not the idealized pulses studied in theory. Figure 3.4 illustrates the nonideal input and output signals to a NAND gate. The transitions between HIGH and LOW voltage levels have nonzero *rise* and *fall times*. The time required for a signal to rise from 10% to 90% of its final value is called the rise time, t_r.

The fall time, t_f, is the time required for the signal to fall from 90% to 10% of its initial value. Secondly, the input voltage to a gate has only to reach the threshold voltage level before the device begins to change state. For these reasons, the delay time is measured with respect to a *reference voltage level* V_{ref}, or the threshold voltage.

Consider the NAND gate in Figure 3.4, connected as a NOT gate. The input waveform, V_{in}, is a non-ideal pulse. When the input signal goes HIGH, the output will go LOW after the *turn-ondelay time* t_{PHL}. The figure illustrates the turn-on delay for a non-ideal output pulse. The typical turn-on delay for a standard series TTL NAND gate is 7 ns. When the input signal goes LOW again, the output of the NAND gate goes HIGH after the *turn-off delay time* t_{PLH}. The typical turn- off delay time for a standard series TTL NAND gate is 11 ns. The average *propagation delay time* t_p is then defined by:

$$t_p = (t_{PHL} + t_{PLH}) / 2.$$

The maximum value for both t_{PHL} and t_{PLH} is 15 ns.

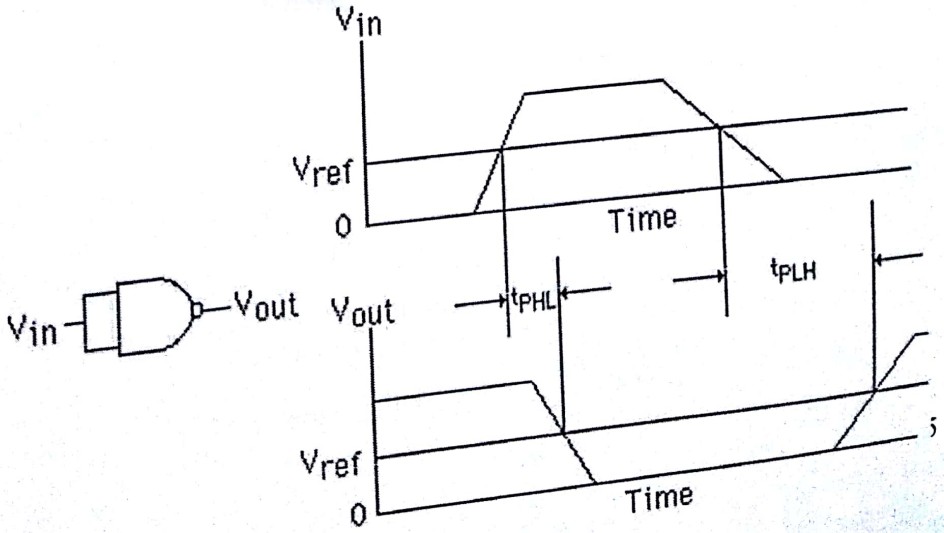


Figure Propagation Delay Times.

A phenomenon associated with TTL devices is **current spiking**. When the output of a TTL device is HIGH, a constant supply current I_{CCH} is drawn from the power supply by the IC.

When the output is LOW, a constant supply current I_{CCL} is drawn from the power supply. For a 7400 NAND gate, I_{CCH} = 4 mA and I_{CCL} = 12 mA per IC. However, when the gate output changes state, a short burst of current is drawn during the transition. The result is current spikes (narrow pulses) in the power supply line as shown in Figure 3.5. The largest spike occurs in the LOW to HIGH transition. To prevent these current spikes from corrupting the power supply and ground and thus appearing as noise, one decoupling capacitor (0.01 μF to 0.1 μF) for each five to ten IC packages are generally connected from power to ground near the IC pins.

Recall from Lab 1, it was mentioned that the power supply should never be connected directly to gate inputs as the gate could be destroyed if the input voltage ever exceeded the supply voltage to the IC--a virtually unlimited amount of current is then allowed to flow backwards through the IC. Current spiking can lead to just such a situation if power were connected directly to a gate's inputs. Thus, another TTL gate should always be used to provide a constant HIGH voltage if required.

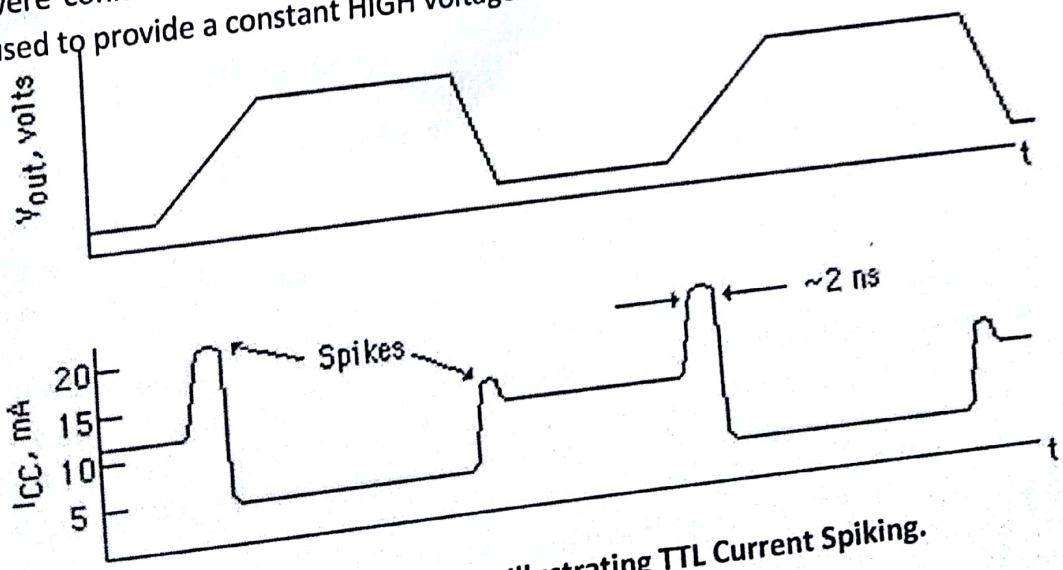


Fig: Waveforms Illustrating TTL Current Spiking.

and prevents oscillation of the gate's outputs, which also causes increased power dissipation and current spiking.

Any loads imposed upon the outputs of gates deteriorate the output signal. **Capacitive loading** occurs if the load impedance to the output of a gate has a capacitive component. This external load capacitance component can be caused by such things as other logic devices or stray wiring capacitance. The effect of capacitive loads is to increase the rise and fall times of signals, subsequently increasing propagation delay times and reducing the maximum operation speed. **Resistive loading** is present when the load impedance contains a resistive component, such asthat associated with other logic devices. The effect of resistive loading is reduced noise margins and output voltages.

TTL gates are available in three different types of output configurations. **Totem-pole output** gates are used in most logic; in this case, the gate by itself drives its output HIGH or LOW depending on the gate's inputs. Connecting the outputs of two or more totem-pole gates together produces undefined output values, may damage the device and should never be done. **Open-collector output** gates can only drive their output LOW; for input combinations where theoutput should be HIGH, an external pullup resistor connected the supply voltage is needed to produce the HIGH. The outputs of open-collector gates to be wired together; the result is to effectively AND all the output signals together. Finally, **three- state** (or **tri-state) output** gates provide, as their name implies, three output states. Like totem-pole output gates, tristate gates can drive their output either HIGH or LOW, as determined by the input combination, but they also have a control input that overrides the effect of the other inputs and places the gate output in a 'third state'. In the third state the internal transistors of the gate are effectively disconnected from the gate output and the output is in an open circuit or high-impedance state. This allows a direct wire connection of many tri- state gate outputs to a common line; however, when this is done, all but at most one of the tri-state output gates connected together must be in the high-impedance state at any given time.

Comparison of TTL and CMOS:

TTL stands for Transistor-Transistor Logic.It is a classification of integrated circuits. The name is derived from the use of two Bipolar Junction Transistors or BJTs in the design of each logic gate. CMOS (Complementary Metal Oxide Semiconductor) is also another classification of ICs that uses Field Effect Transistors in the design.

The primary advantage of CMOS chips to TTL chips is in the greater density of logic gates within the same material. A single logic gate in a CMOS chip can consist of as little as two FETs while a logic gate in a TTL chip can consist of a substantial number of parts as extra components like resistors are needed.

TTL chips tend to consume a lot more power compared to CMOS chips especially at rest. The power consumption of a CMOS chip can vary depending on a few factors. One

major factor in the power consumption of a CMOS circuit is the clock rate, with higher values resulting to higher power consumption. Typically, a single gate in a CMOS chip can consume around 10nW while an equivalent gate on a TTL chip can consume around 10mW of power. That is such a huge margin, which is why CMOS is the preferred chip in mobile devices where power is supplied by a limited source like a battery.

MOS chips are a bit more delicate compared to TTL chips when it comes to handling as it is quite susceptible to electrostatic discharge. People often unwittingly damage their CMOS chips from simply touching the terminals as the amount of static electricity needed to damage CMOS chips are too minute for people to notice.

The prominence of CMOS chips has pushed TTL chips to the background. Instead of being the primary IC of choice, it is now used as components that link the whole circuit as 'glue logic'. CMOS chips that emulate the TTL logic has also gained prominence and is slowly replacing most TTL chips. These chips have similar name to their TTL equivalent so that users can easily identify them.

Summary:

1. TTL circuits utilize BJTs while CMOS circuits utilize FETs.

2. CMOS allows a much higher density of logic functions in a single chip compared to TTL.

3. TTL circuits consumes more power compared to CMOS circuits at rest.

4. CMOS chips are a lot more susceptible to static discharge compared to TTL chips.

5. There are CMOS chips that have TTL logic and are meant as replacements for TTL chips.

Conclusion:-

Hence we have studied TTL Logic Family: Feature, Characteristics and Comparison with CMOS Family