further Processing such as place & Route, timing Analysis & Programming of the PLD.

Synthesis tools, such as Xilinx, Vivado or Synopsys Design Compiler, are used to perform this step. These tools take into account the specific PLD Architecture & optimize the design accordingly.

Simulation → Simulation is a crucial step in the PLD design flow. where the Digital ckt is simulated to verify its functionality & Performance. During Simulation the following steps are involved.

1. Testbench Prepration → Creating test cases for the design.
2. Simulator Selection → choosing the appropriate Simulator tools.
3. Simulation Run → Running the design & testbench in the Simulator.
4. Result Analysis → Analysing the Simulation results & verifying the design functionality.

The benifits of Simulation include.

ES240146276

1) Helping to detect design errors.
2) optimise Performance.
3) Verifying the designs functionality.

Sequential circuit →

VHDL Code for JK flip flop using Case Statement →

```
library ieee;
Use ieee.std_logic_1164.all;
entity JKflipflop is
Port ( J, K, clk, rst : in std_logic;
       Q: out std_logic);
end JKflipflop;

Architecture behavior of JKflipflop is
Signal state: std_logic;
Signal value: std_logic_vector(1 downto 0);
  begin
    process (clk, rst)
  begin
     value <= J & K;
  if rst = '1' then state <= '0';
    elsif clk'event and clk = '1'
            then
      Case value is
```

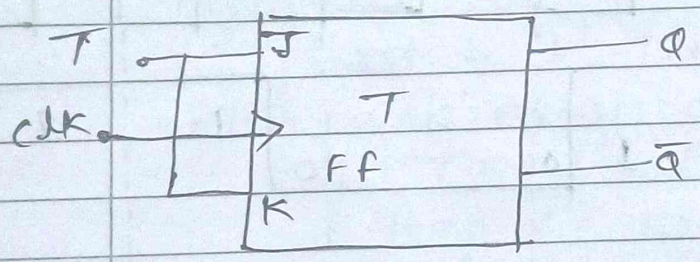
```
when "11" => State <= Not State;
when "10" => State <= '1';
when "01" => State <= '0';

when others => State <= not State;
end Case;
end if;
end Process;
Q <= State;

end behavior;
```

- VHDL Code for JK-Flipflop using if-else Statement

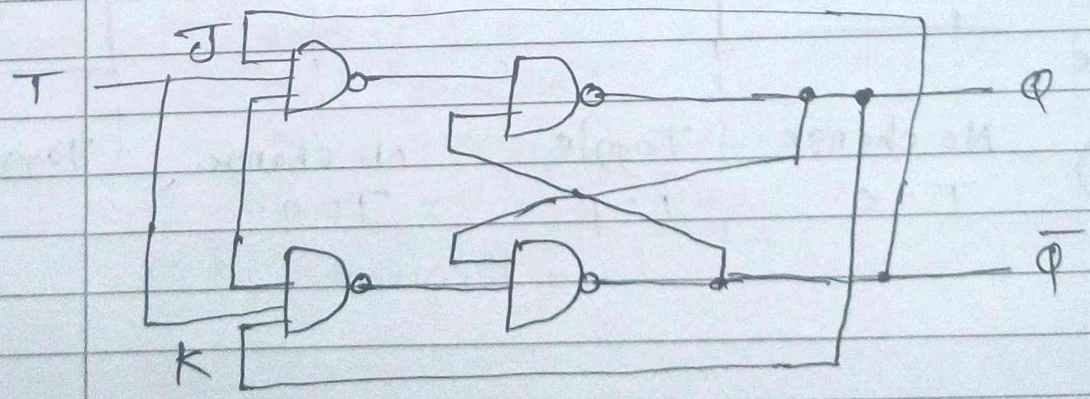
```
library ieee;
use ieee std_logic_1164.all;
entity JKFF is
Port (J, K, clk, rst: in std_logic;
                Q : out std_logic);
end JKFF;

Architecture behavior of JKFF is
Signal temp : std_logic;
  begin
Process (clk, rst)
      begin
  if rst = '1' then temp <= '0';
    elsif clk'event and clk = '1' then
if (J = '1' and K = '1') then temp <= not
                              temp;
```

```
elsif (J = '1' and K = '0') then temp <= '1';
elsif (J = '0' and K = '1') then temp <= '0';
   else temp <= temp;
end if;
end if;
end process;

Q <= temp;

end behavioral;
```

## flip flop



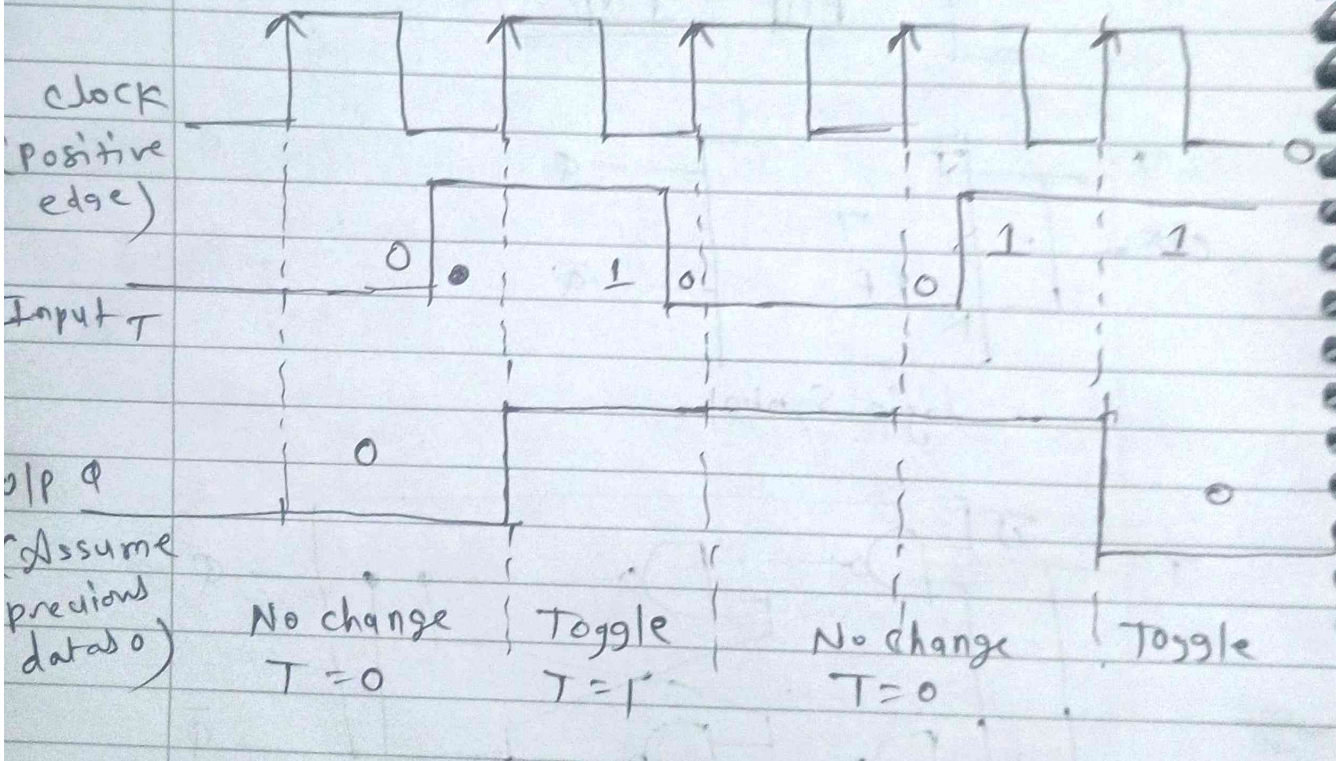

logic Symbol




logic Symbol & internal Structure of T flip flop

It Can designed using J-K flip-flop by Connecting J and K inputs together to a single I/p T as shown in fig.

| Input clk | T | output Q | State |
|---|---|---|---|
| ↑ | 0 | Q | No change |
| ↑ | 1 | $\overline{Q}$ | Toggle |
| 0 | X | Q | No change |

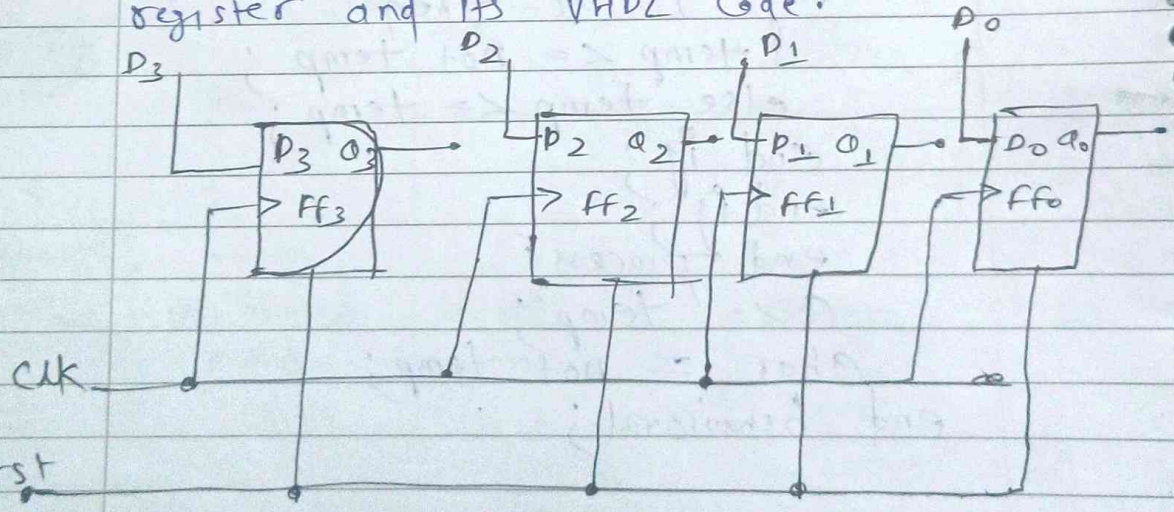


VHDL Code for T-Flip Flop using if-else Statement →

```
library ieee;
use ieee. std_logic_1164. all;
entity Tff is
Port (T, clk, rst : in std_logic);
      Q, Qbar : out std_logic);
end Tff;

Architecture behavior of Tff is
 signal temp : std_logic;
 begin
   process (clk, rst)
 begin
   if rst = '1' then temp <= '0';
 elsif clk' event and clk = '1' then
    if T = '1' then
      temp <= not temp;
     else temp <= temp;
    end if;
    end if;
    end process;
    Q <= temp;
   Qbar = not temp;
 end behavioral;
```

Registers → Registers are the digital circuits which we used to store n-bits information and each bit is stored in a flip flop. Generally, registers are built with D-flip flops. Registers can also be designed using S-R and J-K flip flops. A registers may output data either in Serial form or in parallel form. Serial output means that data is transferred out of the register, one bit at a time.

Parallel output means that the entire data ~~is transferred~~ stored in the register is available in Parallel form and can be transferred out at the same time. The circuit diagram for a 4 bit register and its VHDL Code.




4 bit Register

11

## VHDL Code for Register

```
library ieee;
use ieee.std_logic_1164.all;
entity register is
Port (D : in std_logic_vector
        (3 downto 0);
      clk, rst : in std_logic;
Q : out std_logic_vector (3 downto 0));
end register;

Architecture behavior of register is
begin
    Process (clk, rst)
    begin
    if rst = '1' then
    Q <= "0000";
    elsif clk'event and clk = '1' then
        Q <= D;
    end if;
    end Process;
    end behavior;
```

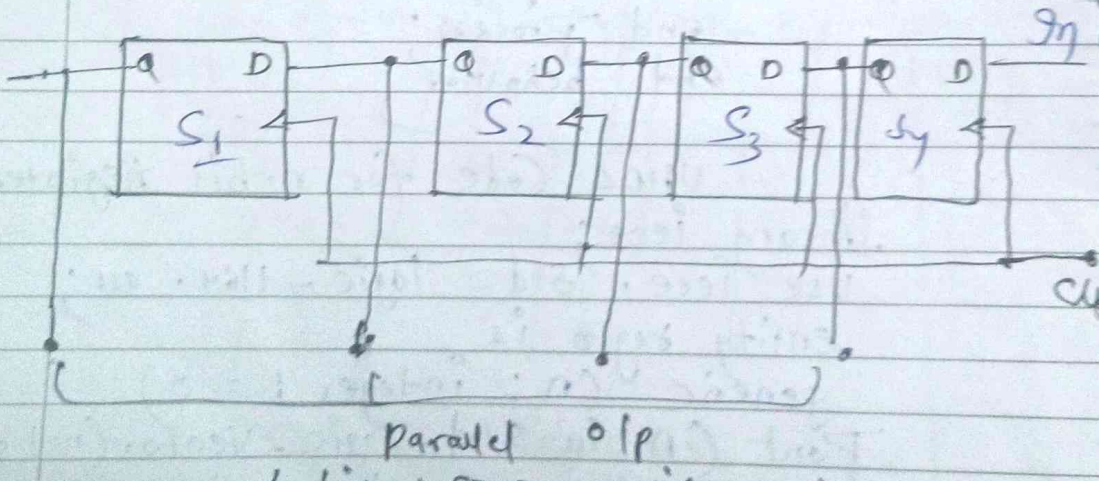
## VHDL Code for n-bit Register

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
generic (n : integer := 8);
Port (D : in std_logic_vector(n-1 downto 0);
      clk, rst : in std_logic;
```

Signal <= , generic variable :=

```
q : out std_logic_vector (n-1 downto 0));
end reg8;

architecture behavior of reg8 is
begin
   Process (clk, rst)
   begin
   if rst = '1' then
      q <= (other => '0');
   elsif clk'event and clk = '1' then
      q <= D;
   end if;
   end process;
   end behavior;
```

4 bit SIPO shift register using D flip flop




4-bit SIPO shift registers.

```
library ieee;
use ieee, std_logic_1164.all;
entity shift4 is
Port (In, clk : in std_logic);
      Q : out std_logic_vector(1 to 4)
end shift4;

Architecture behavior of shift4 is
Signal state : std_logic_vector(1 to 4)
  begin
    Process (clk)
  begin
if clk'event and clk = '1' then
  state(4) <= In;
  state(3) <= state(4);
  state(2) <= state(3);
  state(1) <= state(2);
  end if;

  end process;
  Q <= state;
end behavior;
```

# 4 bit Binary Counter

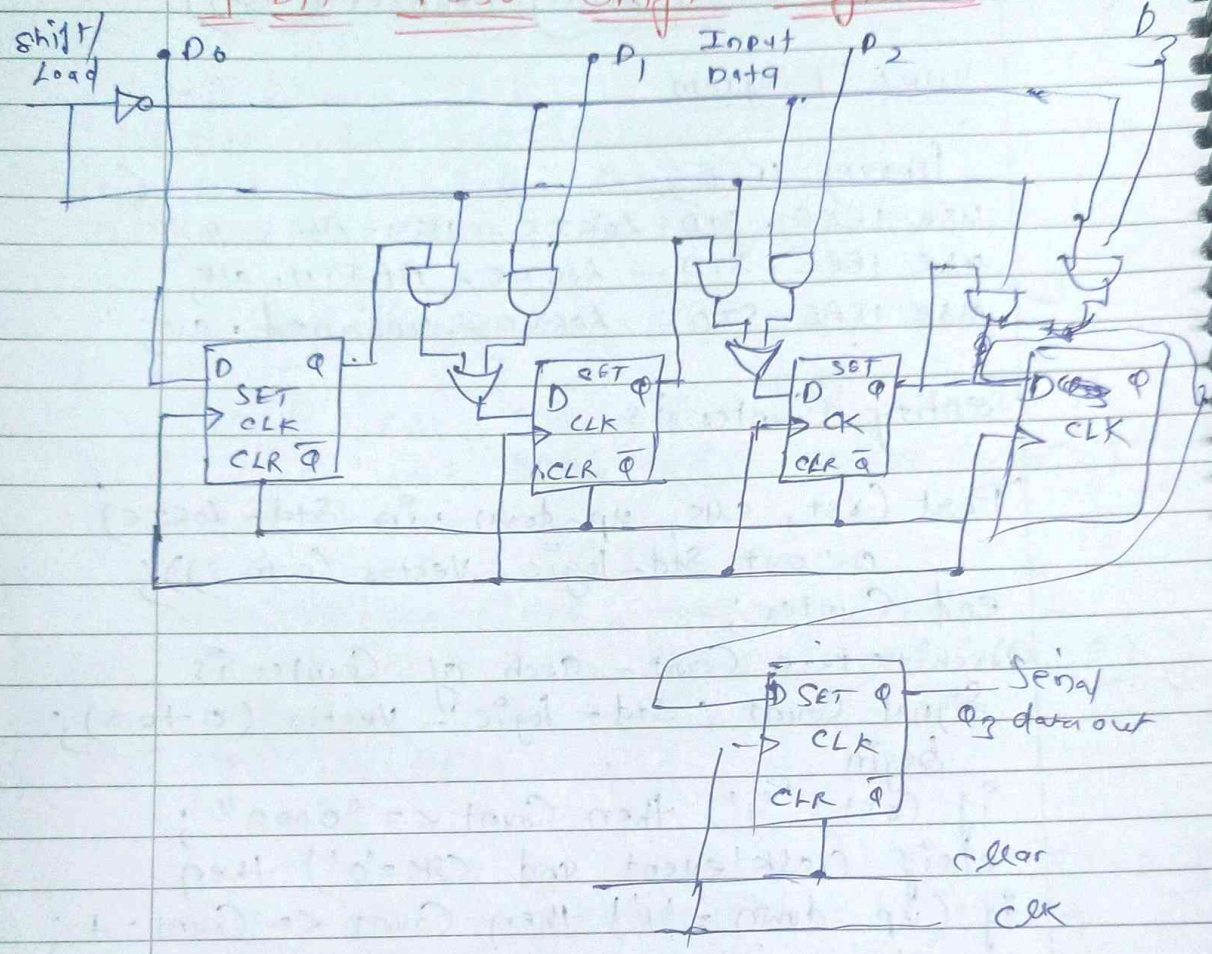
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Counter is
Port (rst, clk : in std_logic;
      O : out std_logic_vector(0 to 3));
end Counter;

architecture Count_arch of Counter is
Signal Count : std_logic_vector(0 to 3);
  begin
Process (rst, clk)
  begin
    if (rst = '1') then Count <= "0000";
elsif (clk'event and clk = '1') then
      Count <= Count + 1;
end if;
end process;
    O <= Count;
  end Count behavioral;
```

# The up-down binary Counter (4bit)

## VHDL Program

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_unsigned.all;

entity Counter is

Port (rst, clk, up_down : in Std_LOGIC)
      O : out Std_logic_Vector (0 to 3));
end Counter;
Architecture Count_arch of Counter is
  Signal Count : Std_logic_Vector (0 to 3);
    begin
  if (rst = '1') then Count <= "0000";
  elsif (clk'event and clk='0') then
  if (up_down = '1') then Count <= Count - 1;
     else
     Count <= Count + 1;
     end if;
     end if;
     end process;
      O <= Count;
      end Count_arch;
```

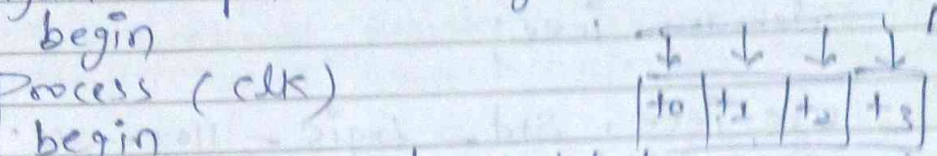
# 4 bit PISO Shift Register →



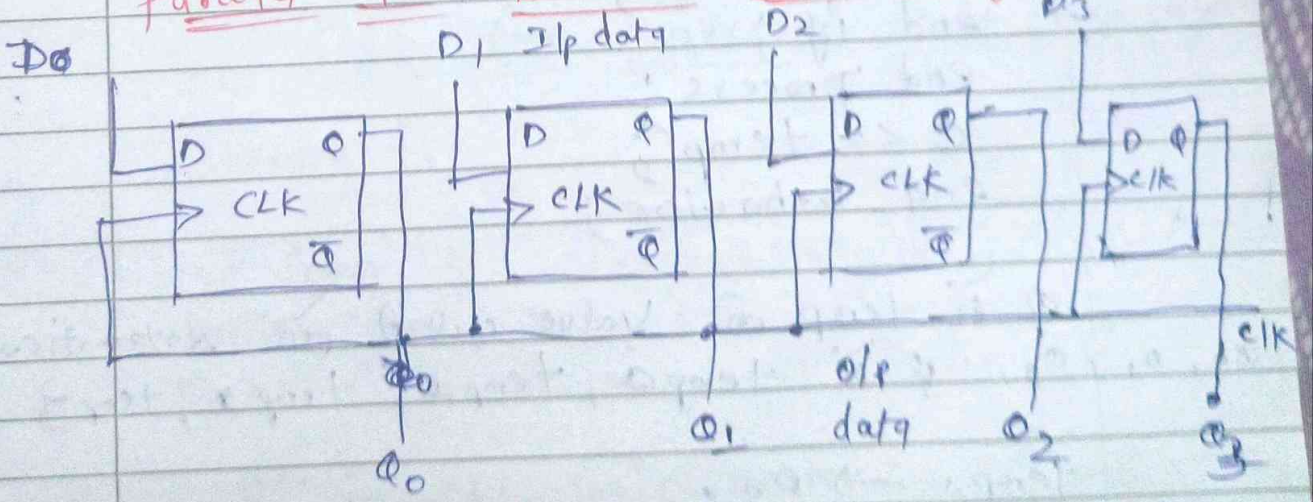

## VHDL Code for PISO shift register

```
library ieee;
use ieee.std_logic_1164.all;
  entity shiftreg is
Port ( D: in std_logic_vector(0 to 3);
  SL, clk : in std_logic;
        Q : out std_logic);
end shiftreg;
```

```
architecture behaviour of shiftreg is
signal temp : std_logic_vector(0 to 3);
  begin
Process (clk)
 begin
if clk'event and clk='1'
        then
if SL = '0' then temp <= D;
  else
temp (1) <= temp (0);
temp (2) <= temp (1);
temp (3) <= temp (2);
 end if;
 end process;

Q <= temp (3);
 end behaviour;
```




## Parallel in Parallel out Shift Register




Parallel In - Parallel out shift Register

# VHDL Code for PIPO Shift Register

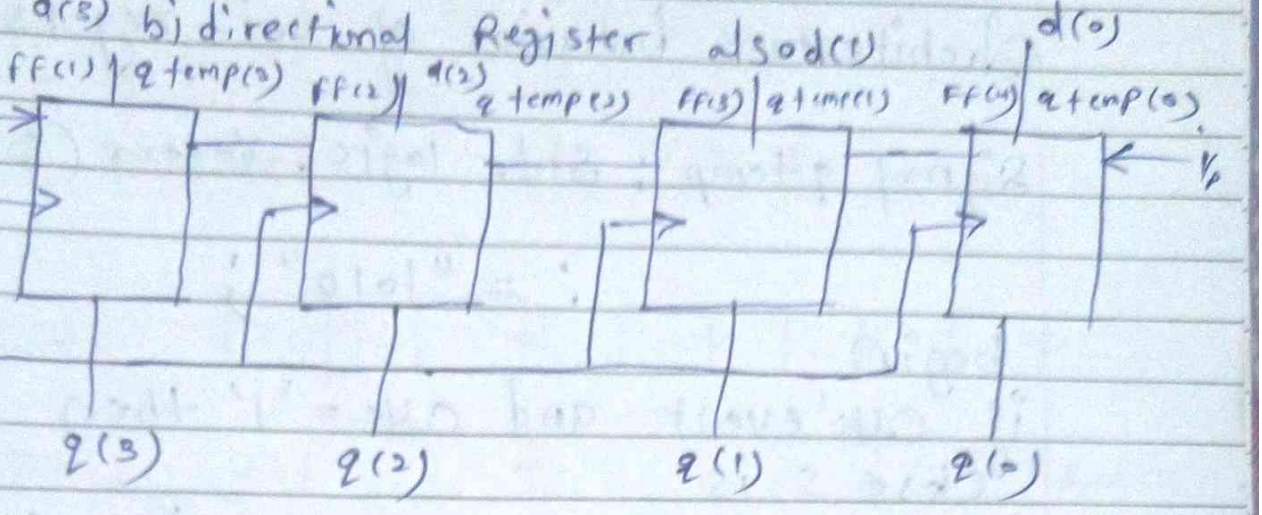
```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg is
Port (D : in std_logic_vector (0 to 3);
      clk : in std_logic;
      Q : out std_logic_vector (0 to 3));
end shift_reg;

Architecture behavior of shift_reg is
signal temp : std_logic_vector (0 to 3);
  begin
Process (clk)
  begin
if clk'event and clk = '1' then
   temp <= D;
  end if;
  end process;
  Q <= temp;
  end behavior;
```

Q में temp की value जाएगी तो Automatically $Q_0$, $Q_1$, $Q_2$, $Q_3$ में temp 0, temp 1, temp 2, temp 3

temp 0 → D0
temp 1 → D1
temp 2 → D2
temp 3 → D3

# Universal Shift Register →

d(3) bidirectional Register also d(0)




| $S_1$ | $S_0$ | State |
|---|---|---|
| 0 | 0 | No change |
| 0 | 1 | Parallel in |
| 1 | 0 | shift left |
| 1 | 1 | shift right |

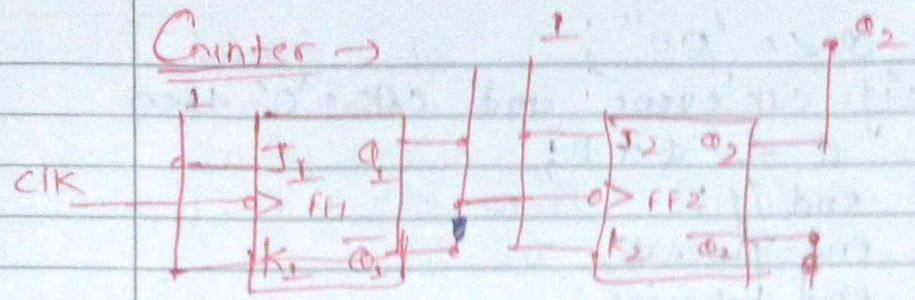
Universal Shift Register

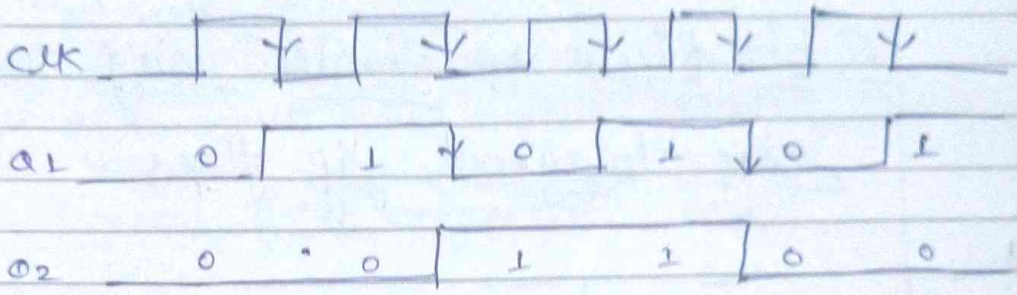
## VHDL Code for universal Shift Register

```
library ieee;
use ieee.std_logic_1164.all;
entity uni_reg is
port (clk, ir, il : in std_logic;
      s : in std_logic_vector(1 downto 0);
      d : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0));
```

```
end uni_reg;
Architecture behavior of uni_reg is
Signal qtemp : Std_logic_vector(3 downto 0)
                := "1010";
begin
if clk'event and clk = '1' then
Case s is
when "00" => qtemp <= qtemp;
when "01" => qtemp <= d;
when "10" => qtemp <= qtemp(2 downto 0)
                & il;
when other => qtemp <= in & qtemp
                (3 downto 1);
end Case;
end if;
end Process;
q <= qtemp;
end bchavior;
```

Counter →



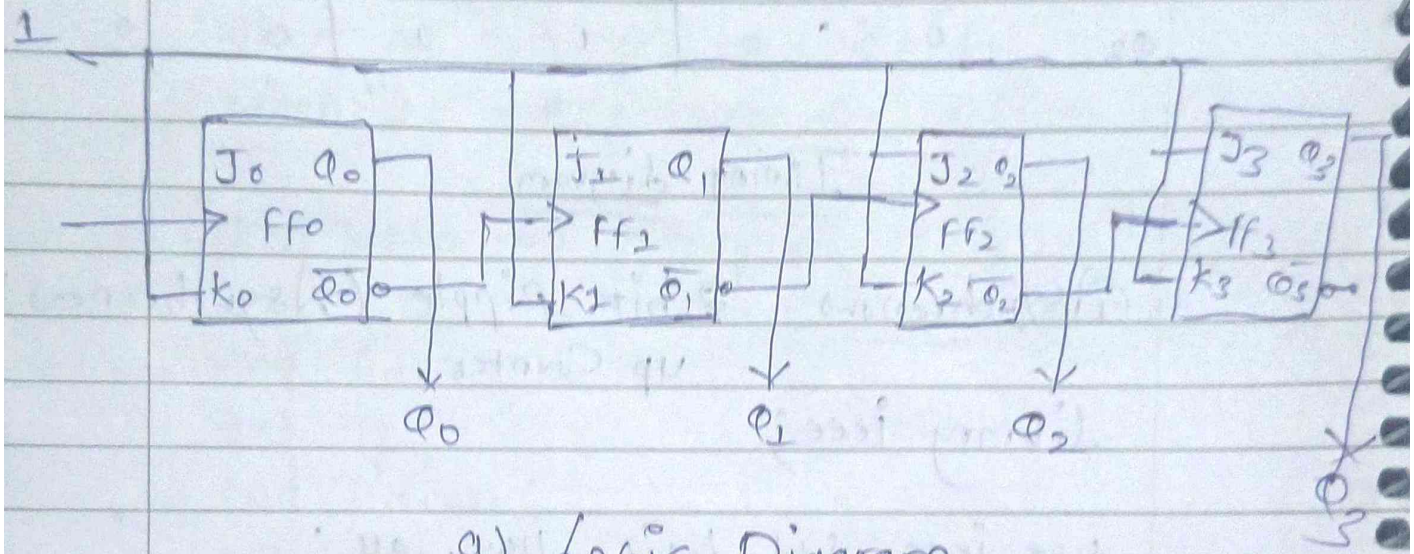

Timing diagram

Asynchronous 2 bit Ripple (Asynchronous) up Counter

```
library ieee;
use ieee. Std-Logic-1164. all;
entity upCounter is
Port (clk, reset : in Std-logic;
      Q : buffer Std-logic-vector
          (1 down to 0));
end upCounter;
Architecture behavior of upCounter is
 begin
     Process (clk, reset)
       begin
    if reset = '1' then
```

```
        Q <= "00";
    elsif clk'event and clk = '0' then
        Q <= Q + 1;
    end if;
    end process;
    end behavior;
```

## 4 bit Ripple up Counter using Positive edge triggered flip flops




a) Logic Diagram