Co-5: Design Custom/ASIC design using FPGA kit through VHDL code.

Learning Outcomes (LO)

Students will able to learn

LO-1 : Various ECAD Tools.

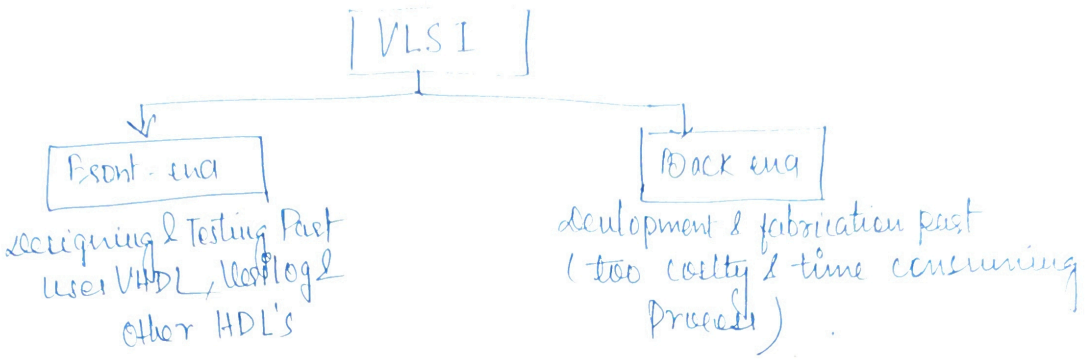
LO-2 : Front End and Back End Design of VLSI Circuits.

LO-3: Custom ASIC Designs.

LO-4 : VHDL Code of following

1) Logic Gates.
2) Flip –Flops
3) Shift Register.

VLSI DESIGNING:-



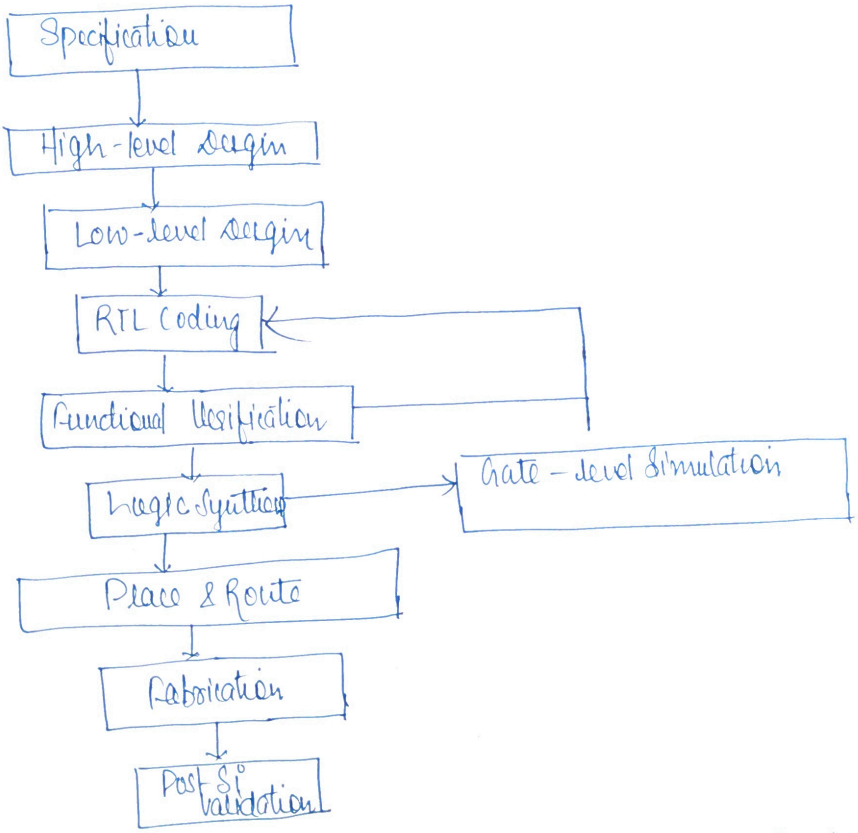


Fig: Block diagram of VLSI design. Front End & Back End.

The above stages are roughly divided into 2 halves.

→ Front end (first half) VLSI design

→ Back end design.

Front End VLSI Design :-

All stages from specification to functional verification are normally considered as a part of front end and engineers working on any of these are front end VLSI design engineers.

Back End VLSI Design :-

All stages from logic synthesis till fabrication are considered as a back end engineer.

1) **SPECIFICATION :-**

In the design process we define important parameters of the system that has to be designed into specification.

2) **High level design :-**

In this various details the design architecture are defined. In this stage, details about the different functional blocks & interface communication protocols between them are defined.

3) **Low- level design :-**

This phase is known as Microarchitecture phase.

In this phase lower level design details about each functional blocks implementation are designed.

This include details like modules, state machine, internal registers decoder, etc.

RTL CODING :-

In RTL Coding, the micro design is modelled in a hardware description language like Verilog / VHDL using constructs of language.

Synthesizable constructs are used so that the RTL model can be input to a Synthesis tool to map the design to actual gate level.

FUNCTIONAL VERIFICATION :-

Process of verifying the functional characteristics of design by generating different input stimuli & checking for correct behaviour of design implementation.

LOGIC SYNTHESIS :-

Synthesis is a process in which Synthesis tools like Compiler takes in RTL, target technology and constraints as input maps the RTL to target technologies.

PLACEMENT & ROUTING :-

Gate level net list from Synthesis tool is taken and imported into place & route tool in Verilog netlist format.

Synthesis & next in routed. After this each block is routed, Output of P&R tool is a GDS file. All these gate & flip-flop are placed, clock tree which is used by foundry for fabricating IC.

GATE LEVEL SIMULATION :-

The placement & Routing tool generates an SDF (Standard Delay File) that contains timing information of the gates.

This is back annotated along with gate level netlist & some functional patterns are used to verify the design functionally.

A static timing analysis tool like prime time can also be used for performing static timing analysis checks.

FABRICATION :-

Once the gate level simulations verify the functional correctness of gate level design after placement & Routing phase, the design is ready for manufacturing.

The final GDS file (a binary database) is normally send to a foundry which fabricates the silicon. Once fabricated proper packaging is done.

POST SILICON FABRICATION :-

Once the chip is back from fabrication it need to be test in real enviornment and tested before using in market.

This phase involves testing in LAB using hardware board & software / firmware that programs the chip.

Since the speed of simulation with RTL is very slow compared to testing in LAB with real silicon, there is always a possibility to find a bug in silicon validation & hence is very important.

Custom Mode Design :

=> Custom design is a methodology for designing integrated circuits by specifying the layout of each individual transistors and the interconnection between them

=> Custom design includes various forms

- Full Custom
- Semi Custom

Full Custom :-

Full Custom design methodology for designing integrated circuits by specifying the layout of each individual resistor & transistors and interconnection between them.

=> Full Custom design include various form of semi-custom design.

=> Full custom design potentially maximizes the performance of chip minimizes its area, but it extremely labor intensive to implement. Full custom design is limited to ICs that are able to fabricate in extremely high value volumes notably certain microprocessor and small number of ASICs.

=> Full Custom design requires all the components to be designed & verified right from transistor level.

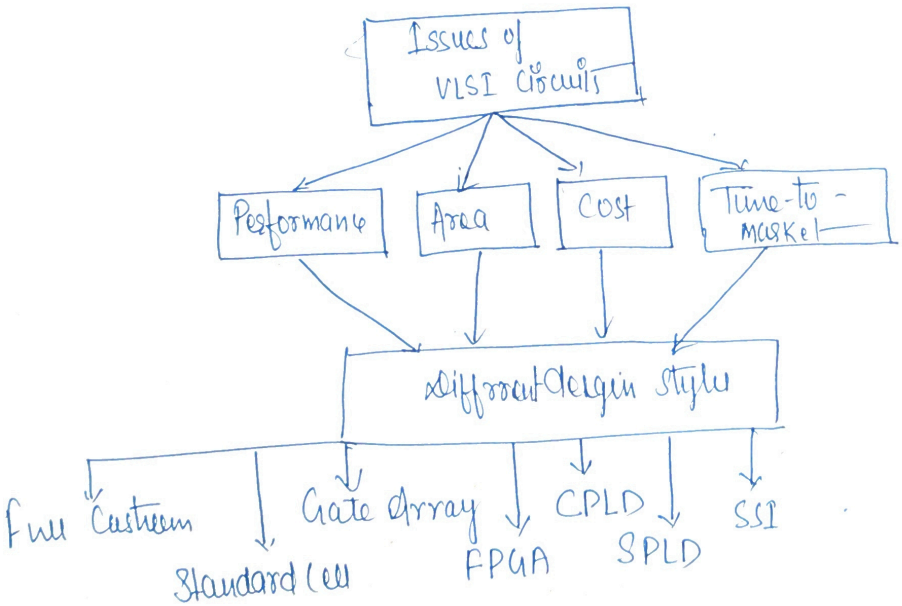
=> In Full Custom design
* Some (possibly all) logic cells are customized and mask layers are customized.

=> In Full Custom design all layers are optimized for an embedded systems particular digital implementation.
* placing transistor
* sizing transistor

* Routing wires.

Advantages:
* High design & Huge design effort
* Each circuit elements carefully manufactured
* Full custom is used for large Volume product
Typically used for high-Volume applications
* Excellent performance, small size, low power

Disadvantages:
* High NRE cost
* long time to come in market



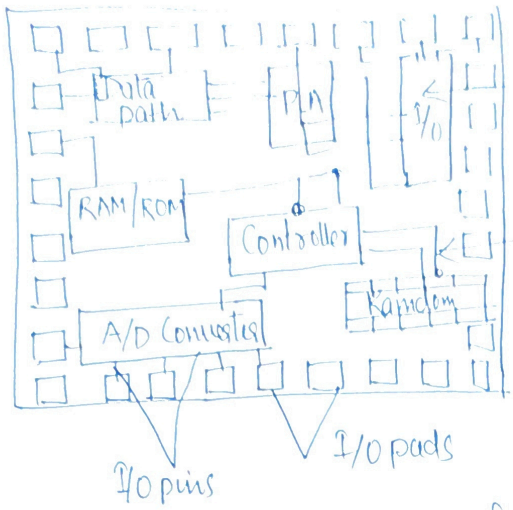


Fig: Block Digram of Full Custom design

SEMI - CUSTOM ASIC DESIGN :-

In Semi- Custom all logic cells are predesigned (defined in cell library) and some (possibly all) of the mask layers are customized.

Semi- Custom ASIC DESIGN
- Standard - cell based
- Gate - Array based ASICs.

Standard cell based :-

Characteristics of standard cell based design

⇒ The chip layout can be created automatically by CAD TOOLS because of regular arrangements of logic gates (cells) in rows.

⇒ Standard cells can be placed anywhere on a silicon

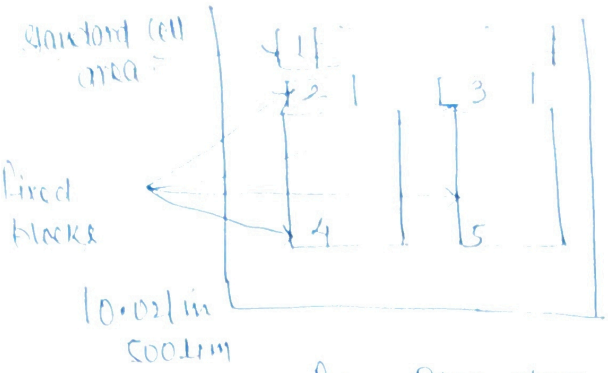
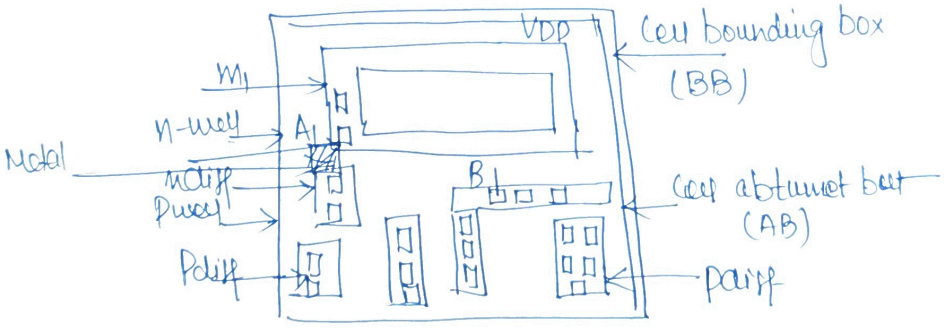


Fig : Block diagram of standard cell

(3)

⇒ Standard-cells area can be used in communication with designer pre-designed cells (microcontroller, microprocessor), known as mega-cells.

⇒ Standard-cells are designed to fit horizontally together to form rows.



⇒ 25 microns wide (lamda is 0.25)

AB : abutment box

BB : Bounding box

Power supplies : VDD, GND

Each different shaded & labled pattern represents a diffrent layer

Connections : A1, B1, Y

GATE-ARRAY :-

⟹ Gate Array is an approach to design & manufacture of application specific integrated circuits (ASIC(s)) using a prefabricated chip with components that are later interconnected into logic device (eg NAND gate, flip-flop).

⟹ Gate Array have also known as Uncommitted logic Array (ULAs) & Semi-Custom chips.

⟹ Gate Array ranks second after Gate Array design (GA) ranks second after the FPGA, in terms of fast prototyping capability.

⟹ Gate Array requires a two-step manufacturing process.

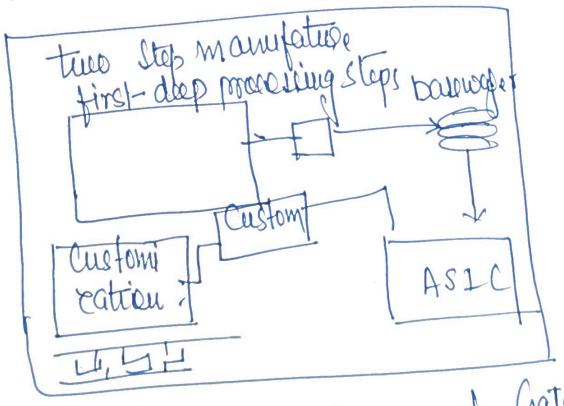
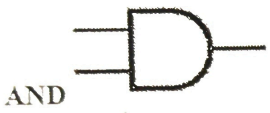


Fig : Block diagram of Gate Array.

Design of various Logic Gates using VHDL

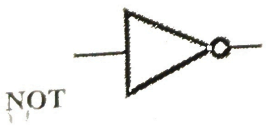
LOGIC GATES:

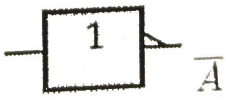
A logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic normally performed is Boolean logic and is most commonly found in digital circuits. Logic gates are primarily implemented electronically using diodes or transistors, but can also be constructed using electromagnetic relays (relay logic), fluidic logic, pneumatic logic, optics, molecules, or even mechanical elements.



INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

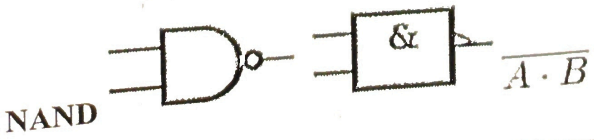
INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1





INPUT	OUTPUT
A	NOT A
0	1
1	0

In electronics a NOT gate is more commonly called an inverter. The circle on the symbol is called a *bubble*, and is generally used in circuit diagrams to indicate an inverted (active-low) input or output.



INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR

$\overline{A+B}$

INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

XOR

$A \oplus B$

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$\overline{A \oplus B}$ or $A \odot B$

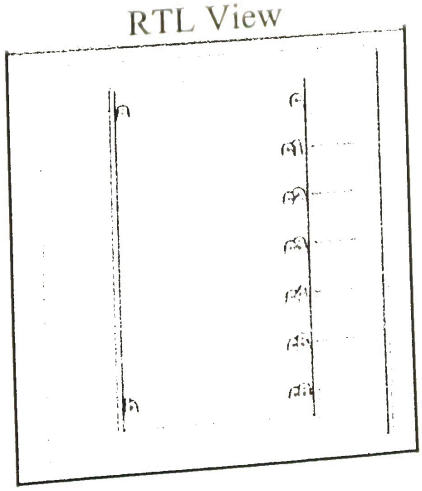
INPUT		OUTPUT
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Program:

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

entity all_ga is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : out  STD_LOGIC;
           c1 : out STD_LOGIC;
           c2 : out STD_LOGIC;
           c3 : out STD_LOGIC;
           c4 : out STD_LOGIC;
           c5 : out STD_LOGIC;
           c6 : out  STD_LOGIC);
end all_ga;

architecture Behavioral of all_ga is

begin
c <= a and b;
c1 <=  a or b;
c2 <= a nand b;
c3 <= a nor b;
c4 <= a xor b;
c5 <= a xnor b;
c6 <= not b;

end Behavioral;
```

OUTPUT:

RTL View

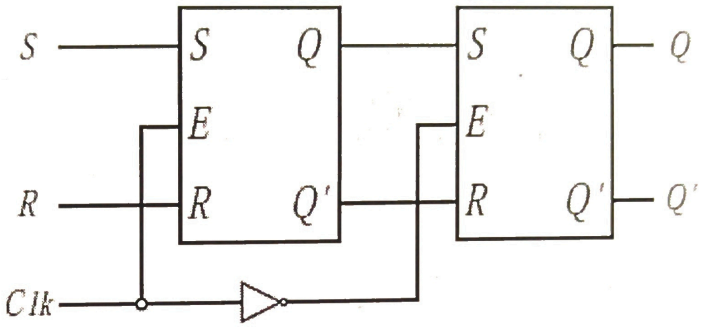


Aim : Design a RS & JK flip-flop

RS & JK FLIP-FLOP

RS flip-flops are useful in control applications where we want to be able to set or reset the data bit. However, unlike SR latches, SR flip-flops change their content only at the active edge of the clock signal. Similar to SR latches, SR flip-flops can enter an undefined state when both inputs are asserted simultaneously.

The truth table and circuit diagram are as follows:



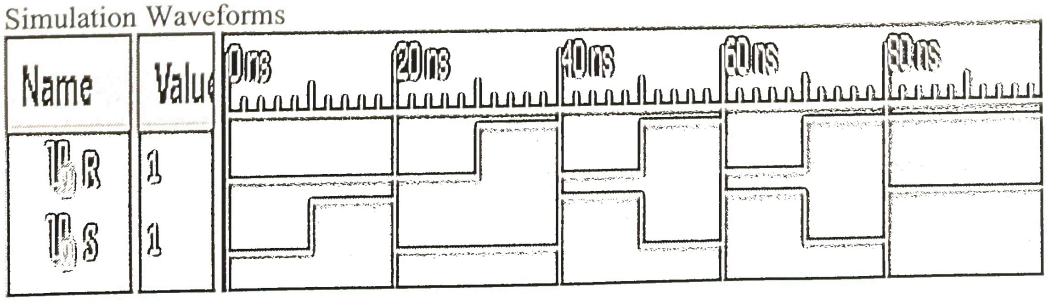
S	R	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	×	×
1	1	1	×	×

Program:

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

```
entity SR-FF is
PORT( S,R,CLOCK,CLR,PRESET: in std_logic;
                            Q, QBAR: out std_logic);
end SR-FF;

Architecture behavioral of SR-FF is
begin
P1: PROCESS(CLOCK,CLR,PRESET)
variable x: std_logic;
begin
if(CLR='0') then
 x:='0';

elsif(PRESET='0')then
 x:='1';

elsif(CLOCK='1' and CLOCK'EVENT) then

if(S='0' and R='0')then
 x:=x;
 elsif(S='1' and R='1')then
 x:='Z';

elsif(S='0' and R='1')then
 x:='0';

else
 x:='1';

end if;
end if;

   Q<=x;
   QBAR<=not x;
end PROCESS;
end behavioral;
```

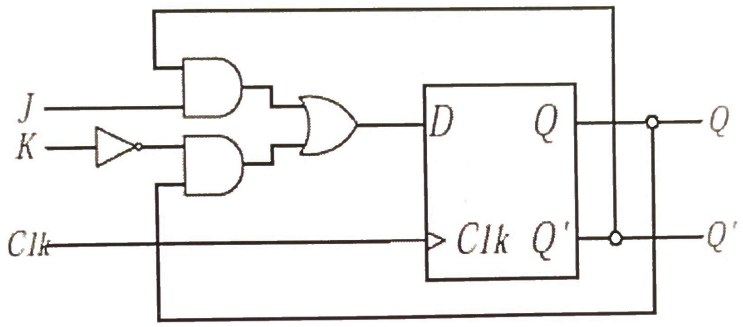
Simulation Waveforms



JK FLIP-FLOP

JK flip-flops are very similar to SR flip-flops. The J input is just like the *S* input in that when asserted, it sets the
flip-flop. Similarly, the K input is like the R input where it clears the flip-flop when asserted. The only difference is when both inputs are asserted. For the SR flip-flop, the next state is undefined, whereas, for the JK flip-flop, the next state is the inverse of the current state. In other words, the JK flip-flop toggles its state when both inputs are asserted.

The truth table and circuit diagram are drawn below:



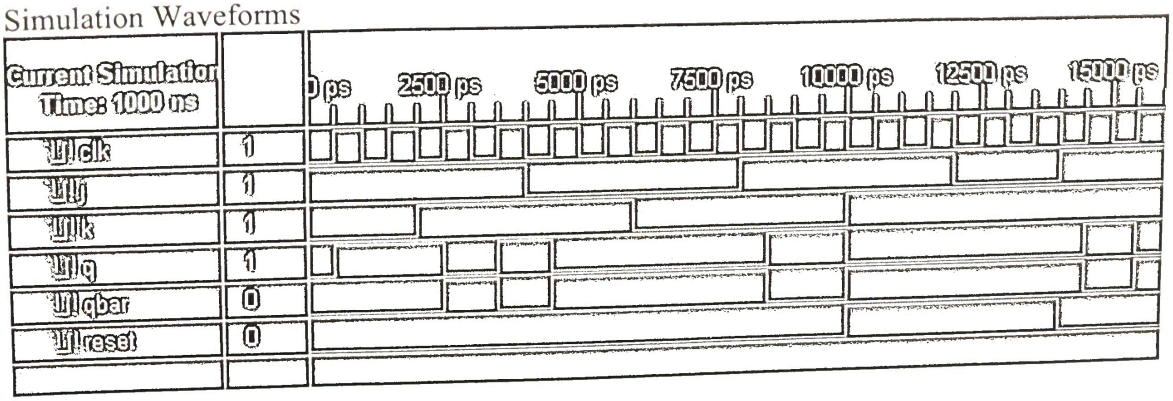
J	K	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Program:

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

entity JK-FF is
PORT( J,K,CLK,PRST,CLR: in std_logic;
                Q, QB: out std_logic);

end JK-FF;
```

```
Architecture behavioral of JK-FF is
begin
P1: PROCESS(CLK,CLR,PRST)
variable x: std_logic;
begin
if(CLR='0') then
 x:='0';

elsif(PRST='0')then
 x:='1';

elsif(CLK='1' and CLK'EVENT) then
    if(J='0' and K='0')then
       x:=x;
   elsif(J='1' and K='1')then
       x:= not x;

   elsif(J='0' and K='1')then
       x:='0';
   else
       x:='1';

   end if;
 end if;
   Q<=x;
   QB<=not x;
 end  PROCESS;
 end behavioral;
```

Simulation Waveforms



Quiz Questions with answer.

Aim:- To Design a 8 bit shift register

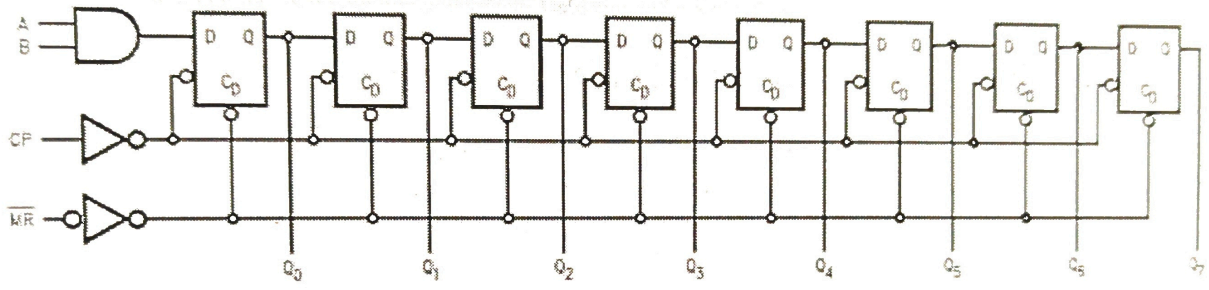
8-bit Shift Register

Shift Register is a type of sequential circuit formed by combination of flip-flops and is capable of shifting data from left to right or vice-versa. Shift register basically performs two functions:

i. Shifting of data(Transfer of data)
ii. Storage function

The circuit diagram for 8-bit Shift Register is given as:

Logic Diagram



The vhdl program for 8-bit shift-left register with a positive-edge clock, serial in, and serial out.

Program:

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

entity shift is
port(C, SI : in std_logic; SO : out std_logic);
end shift;

architecture archi of shift is
signal tmp: std_logic_vector(7 downto 0);

begin
process (C)
begin
if (C'event and C='1') then
for i in 0 to 6 loop
```

```
    tmp(i+1) <= tmp(i);
  end loop;
  tmp(0) <= SI;
  end if;
 end process;

 SO <= tmp(7);
end archi;
```

Simulation Waveforms