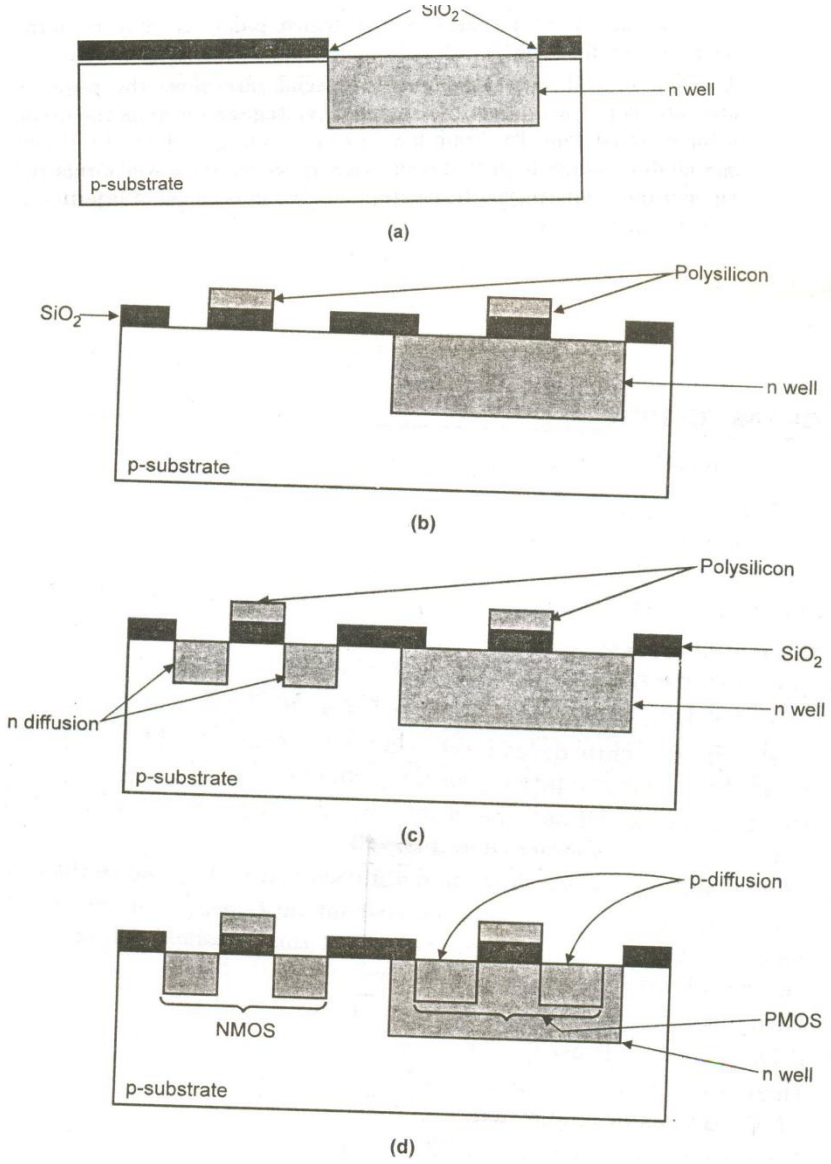
Q.3)

a) N-Well Fabrication Process **(2 Marks for Description and 2 Marks For Diagram)**

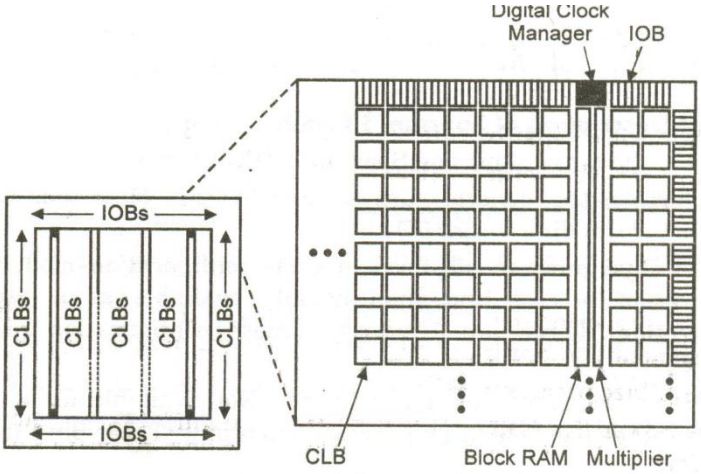
The steps for the n-well process are given below. It is shown in Fig. ..

1) Start with the lightly doped p-type substrate (wafer). In this p-type substrate, we need to create n-type well, for the p-channel devices.
2) The first mask defines the n-well as shown in Fig. (a). p-channel transistors will be fabricated in this well.
3) Ion implantation or deposition and diffusion is used to produce the n-well.
4) This diffusion of n-well must be carried out with special care since the n-well doping concentration and depth will affect the threshold voltages as well as the breakdown voltages of n transistors.
5) Then SiO_2 layer is deposited as shown in Fig.
6) Polysilicon layer is used to form the gate as shown in Fig b).
7) Then p-substrate is diffused with the n-diffusion, which is used to form NMOS transistor as shown in Fig. (c).
8) The n-well is used to form the PMOS transistor after the p-diffusion.

n-well CMOS circuits are superior to p-well because of the lower substrate bias effects on the transistor threshold voltage and lower parasitic capacitances associated with source and drain regions.



b) FPGA Architecture **(2 Marks for Diagram and 2 Marks for Description)**

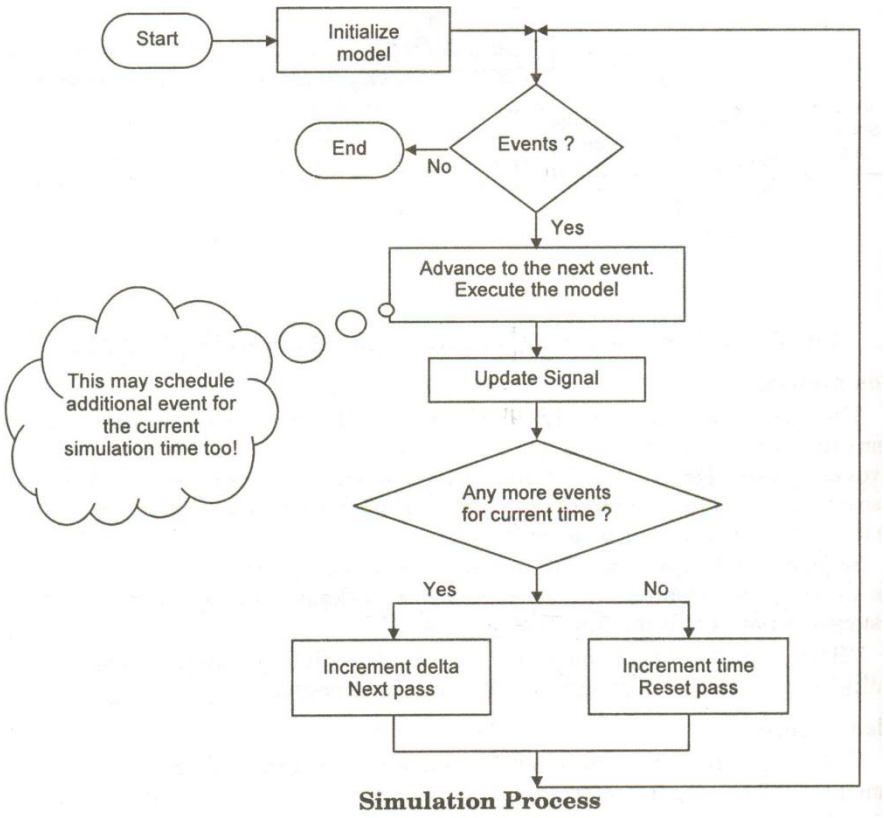


Spartan-3 Family Architecture

1) **CLBs :** It contains RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
2) **IOBs :** They control the flow of data between the I/O pins and the internal logic of the device. Double Data Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
3) Block RAM provides data storage in the form of 18-K bits dual port blocks.
4) Multiplier blocks accept two 18 bit binary numbers as inputs and calculate the product.
5) **Digital clock manager :** Digital clock manager blocks provide self calibrating, fully digital solutions for distributing, delaying, multiplying, dividing and phase shifting clock signals.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

c) Simulation Process **(2 Marks for Diagram and 2 Marks for Description)**



Simulation Process

At time zero, all gate outputs are set to an unknown value. The logic simulator prepares two queues, evaluation and event.

The evaluation queue keeps tracks of logic cells whose outputs are changing and the new values for each output, i.e. all the signals on the left hand side (target signal) of assignments are stored in evaluation queue. Also processes to be executed are stored in evaluation queue.

The event queue keeps tracks of logic cells whose inputs have changed, i.e. all the right hand side signals of the assignment are stored in the event queue. Signals to be updated are stored in the event list.

e.g. While simulating Z <= X *and* Y statement, Z is stored in evaluation queue whereas X and Y are stored in event queue.

When simulation time is incremented, on receiving simulation commands, a signal is updated. (Signals from event queue). Then all target signals and processes sensitive to that signals are placed on evaluation queue and its value is evaluated. Now, this updating of signal may be in the event queue of another signal, so now, that signal is called from evaluation queue for updating.

Each resumed process is executed until it suspends. Effects of the logic changes that have occurred as a result of process execution are evaluated. Signal values are updated only after the process suspends. Then simulation time set to the next event in queue, or halted if simulation time is exhausted. One loop around this cycle is known as a delta cycle.

The simulation cycle is then continuously repeated during which processes are executed and signals are updated till there is no event on any signal in the event queue.

Till this, although real time is going on but simulator keeps its simulating time frozen. When the simulation becomes stable i.e. there are no events in event queue, then only simulator increments its simulation time. Thus real time and simulation time differs. Whatever real time required for the simulator to execute one simulation cycle is called as delta delay.

d) Any other relevant example of modelling can be considered (Design 2 Marks and VHDL code 2 Marks)

Design a BCD to seven segment decoder for single digit LED display with VHDL statements.

Solution :

Step I :

Let us draw schematic for BCD to seven segment display.

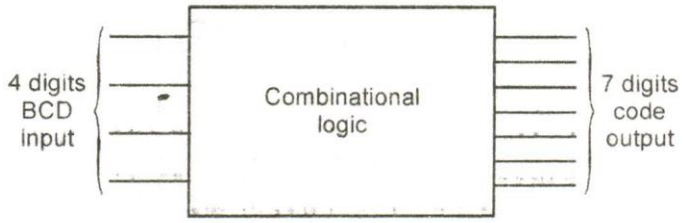
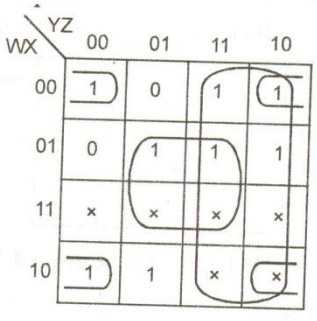


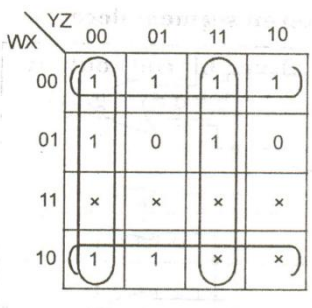
Fig. 7.7 : BCD to Seven Segment Decoder

Step II : Let us write truth table.

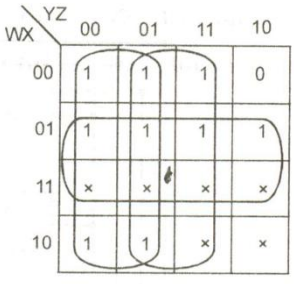
Decimal Digit	4 bit Binary code	Seven bit code output a b c d e f g
0	0 0 0 0	1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 1	1 0 1 1 1 1 1
7	1 0 0 0	1 1 1 0 0 0 0
8	1 0 0 1	1 1 1 1 1 1 1
9	1 0 1 0	1 1 1 1 0 1 1
-	1 0 1 1	×
-	1 1 0 0	×
-	1 1 0 1	×
-	1 1 1 0	×
-	1 1 1 1	×



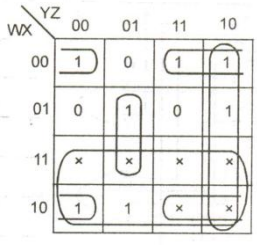
$a = w + y + x z + \bar{x}\ \bar{z}$



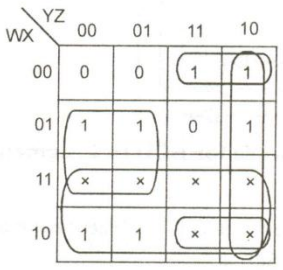
$b = \bar{x} + \bar{y}\ \bar{z} + y z$



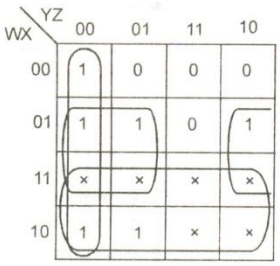
$c = \bar{y} + z + x$



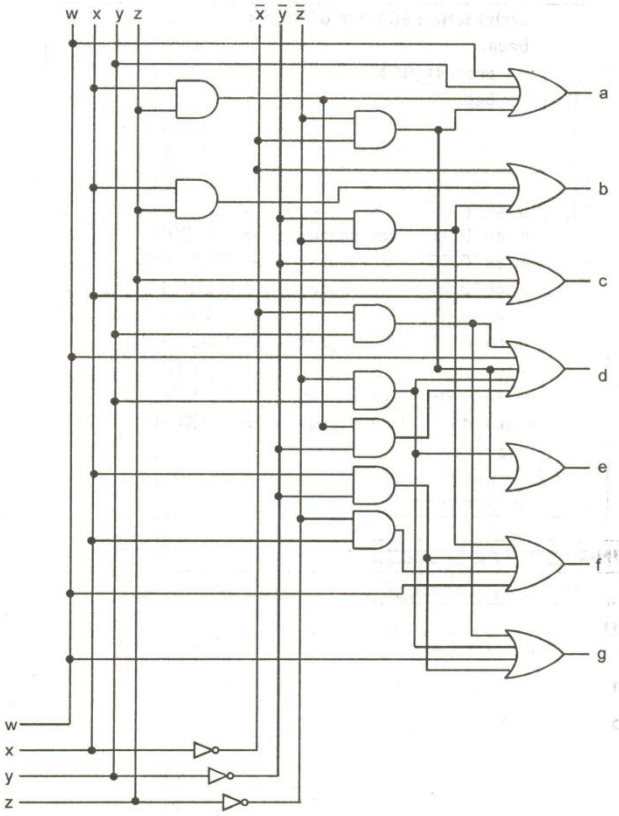
$d = w + \bar{x}\,.\,y + y.\,\bar{z} + \bar{z}\,\bar{x} + \bar{y}\ z x$



$g = w + y\bar{z} + x\bar{y} + \bar{x}y$



$f = w + \bar{y}\,\bar{z} + x\bar{y} + x\bar{z}$



VHDL code for BCD to 7 segment encoder is

```
entity BCD is
    Port (BCDin : in std_logic vector (3 down to 0)
        Segout : out std_logic_vector (6 down to 0);

    end BCD;
```

```
architecture BCDarch of BCD is
begin
    process (BCDIN)
    begin
    case BCDIN is
when "0000"  ⇒  segout  ⇐ "1111110"
when "0001"  ⇒  segout  ⇐ "0110000"
when "0010"  ⇒  segout  ⇐ "1101101"
when "0011"  ⇒  segout  ⇐ "1111001"
when "0100"  ⇒  segout  ⇐ "0110010"
when "0101"  ⇒  segout  ⇐ "1011011"
when "0110"  ⇒  segout  ⇐ "1011011"
when "0111"  ⇒  segout  ⇐ "1110000"
when "1000"  ⇒  segout  ⇐ "1111111"
when "1001"  ⇒  segout  ⇐ "1111011"
when others  ⇒  segout  ⇐ "000000"
end case;
    end process;
end BCDarch;
```

e) Configuration 2 Marks and Attributes 2 Marks

Configuration

(a) A configuration statement is used to bind a component interface to an entity-architecture pair. A configuration can be considered as port or list of design.

(b) It describes which behaviour to use for each entity, much like part list describing which part is to be used in which part of design.

(c) Thus configuration statement maps component instantiations to entities.

(d) **Example :** In hierarchical fashion, architecture netlist appears at topmost position in entity i.e. mux U_1 and U_2 are two component instances of inverter initiated in the netlist.

```
Configuration mux C1 of mux is
    For netlist
        For U1, U2 :
            Inverter USE Entity work myinv;
        End for;
        For U3, U4, U5, U6 : and gate use Entity work . myand (version 1)
        END for;
        For U7; or gate USE entity work. myor (version 1)
        END for;
        END for;
    END muxc1
```

Attributes

Predefined attributes have number of important applications.

Attributes can be used to detect clock edges perform timing checks with 'ASSERT' statement return information about unconstrained types and much more.

Types of Attributes : Following are types of attributes :

(a) 'Function' kind Attributes

(b) 'Type' kind Attributes

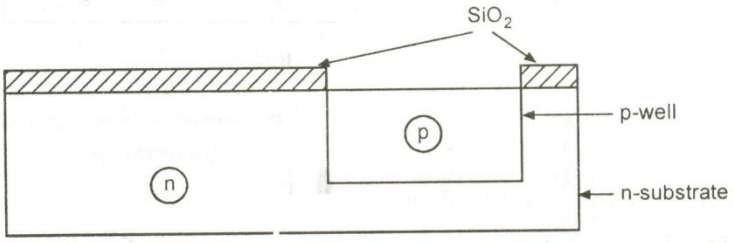
(c) 'Value' kind Attributes

(d) 'Signal' kind Attributes

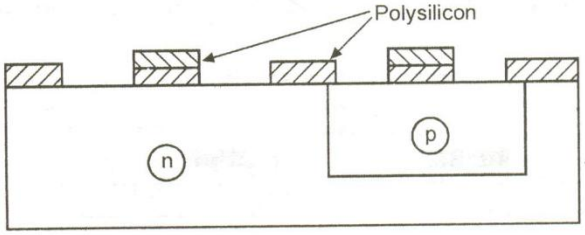
(e) 'Range' kind Attributes

4a i) Procedure 1 marks and diagrams 3 Marks

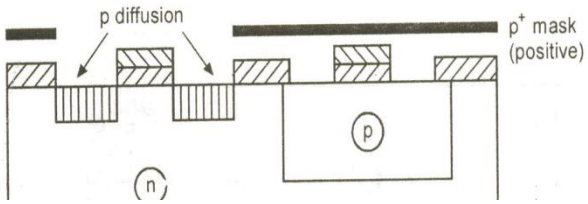
- Mask 1 : It defines the areas in which the deep p-well diffusions are to take place.
- Mask 2 : It defines the thin oxide regions. The areas where the thick oxide is to be stripped and thin oxide grown to accommodate p and n transistors and diffusion wires.
- Mask 3 : It is used to pattern the polysilicon layer which is deposited after the thin oxide.
- Mask 4 : A p-plus mask is used to define all areas where p-diffusion is to take place.
- Mask 5 : It is performed using the negative form of the p-plus mask and with mask 2, it defines those areas where n-type diffusion is to take place.
- Mask 6 : Contact cuts are now defined.
- Mask 7 : The metal layer pattern is defined by this mask.
- Mask 8 : An overall passivation (over glass) layer is now applied and mask 8 is needed to define the openings for access to bonding pads.

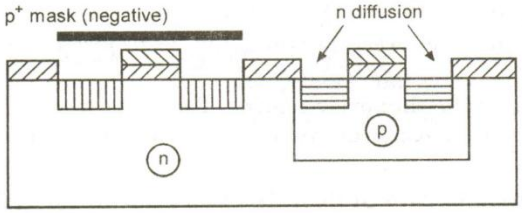


(a)



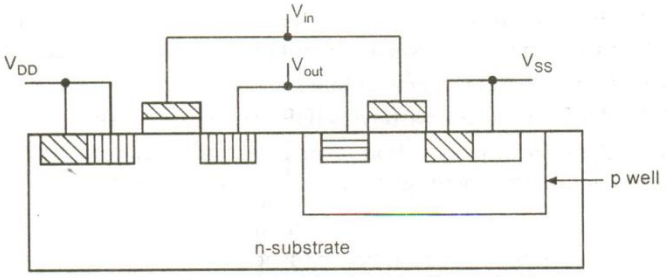
(b)





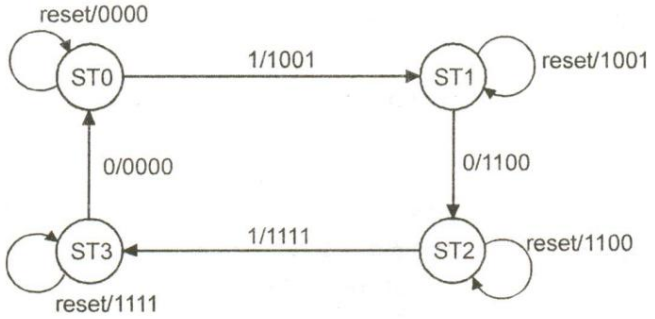
(d)

CMOS p-well process steps



ii) Design 2 Marks and VHDL code 2 Marks (Any other relevant example can be considered)

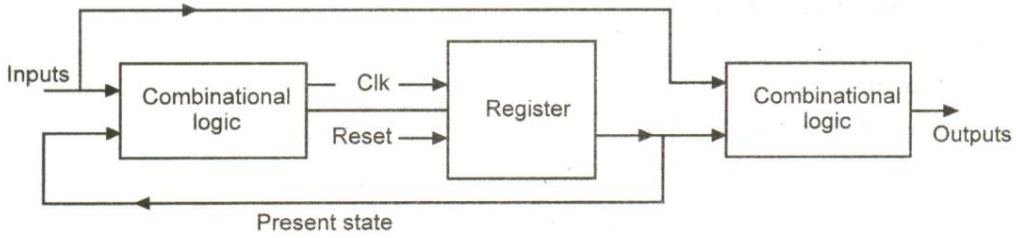
Mealy machine state diagram is shown in Fig. 3.10. The output signals are dependent on both present state and all input signals. It means the output signals change immediately if the input signals change or if the state is changed.



Mealy machine's state diagram

The difference between the Moore and Mealy machine is the combinational output signal process. In Mealy machine, the output signal in combinational process should be a function of the state vector and all of the inputs. The clocked process in Mealy and Moore machines is identical.

The block diagram of Mealy machine is shown in Fig. 3.11.



```
Entity MealySM is
port (clock, reset, input : in std_logic;
        output : out std_logic_vector (3 downto 0));
end MealySM;

architecture Mealy_arch of MealySM is
   type state_type is (ST0, ST1, ST2, ST3);
   signal state : state_type;
begin
    P0 : process (clock, reset)
    begin
        if reset = '1' then
            state <= ST0;
        elsif clock'event and clock = '1' then
            case state is
                when ST0 ⇒  if input = '1' then
                                state < = ST1;
                            end if;
                when ST1 ⇒  if input = '0' then
                                state < = ST2;
                            end if;
                when ST2 ⇒  if input = '1' then
                                state < = ST3;
                            end if;
                when ST3 ⇒  if input = '0' then
                                state < = ST0;
                            end if;
            end case;
        end if;
    end process P0;
P1 : process (state, input)
begin
    case state is
        when ST0 ⇒  if input = '1' then
                        output < = "1001",
                    else
                        output < = "0000";
                    end if;
        when ST1 ⇒  if input = '0' then
                        output < = "1100",
                    else
                        output < = "1001";
                    end if;
```

```
        when ST2 ⇒   if input = '1' then
                         output < = "1111",
                     else
                         output < = "1100";
                     end if;
        when ST3 ⇒   if input = '0' then
                         output < = "0000",
                     else
                         output < = "1111";
                     end if;
        end case;
    end if;
  end process p1;
end Mealy_arch;
```

Synthesis meaning 1 Mark, Input and Output 1 Mark diagram 1 mark , Constraints 1 Mark

Synthesis is an automatic method of conversion of higher level of abstraction into the lower level of abstraction.

(b) As now we have defined synthesis, it is now time to decide whether VHDL is the synthesizer or simulator ?

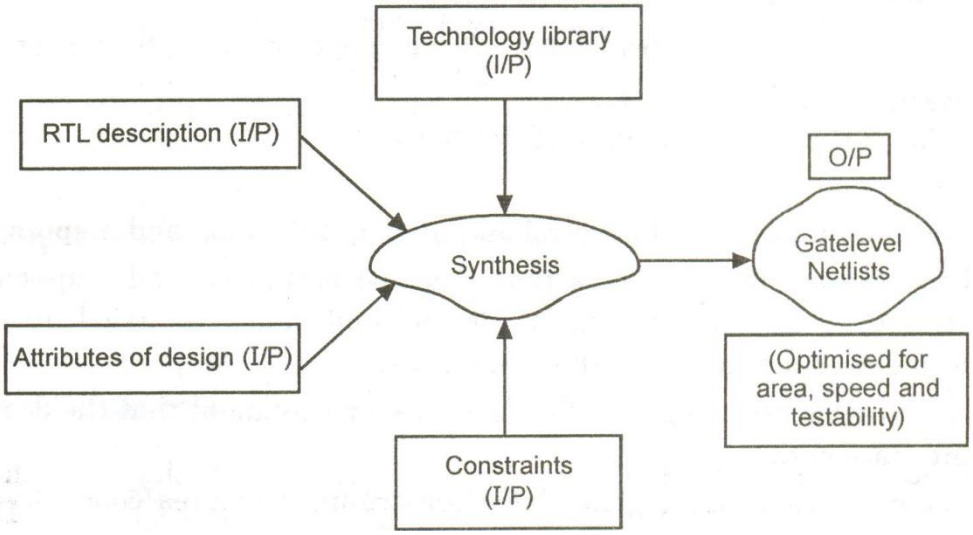
As per our definition we have to implement our logic on chip with the help of synthesis. And for synthesis we require additional tools like EDR tools, DSP tools along with language structures. Hence VHDL language is a **synthesizer and not a simulator**.

Input for the synthesis process are

1) RTL description
2) Technology library
3) Constraints
4) And finally attributes of design.

Output of the synthesis process is optimized gate level netlist obtained from the above described inputs.

Thus this process can be seen as :



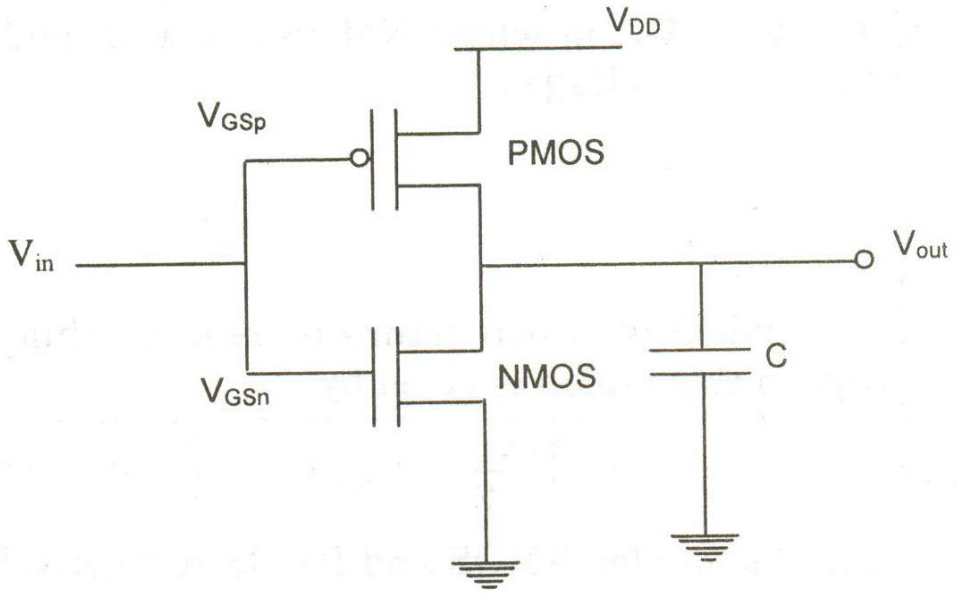
Various types of constraints are

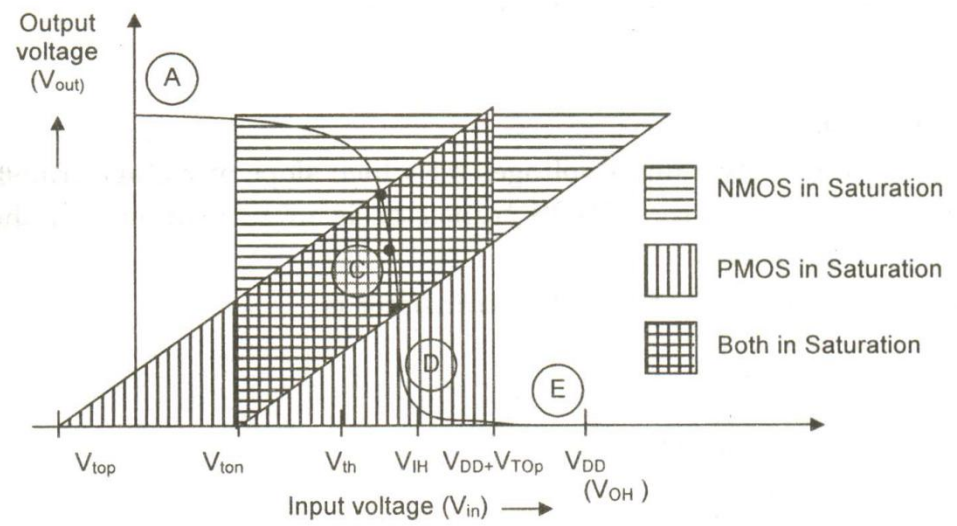
(a) Area constraints
(b) Clock constraints
(c) Timing constraints (Delay constraints)
(d) Testing constraints
(e) Packaging constraints

iv) 1 mark for each difference (Any four difference points) , any other relevant difference can be considered

SRAM FPGA	Antifuse FPGA
1) The physical dimensions of an SRAM cell are of the order of magnitude larger than those of an antifuse element.	1) Antifuse elements take less space compared to SRAM cells. Antifuse elements can be placed very densely.
2) SRAM FPGAs can be reprogrammed.	2) Once programmed, an antifuse element cannot be erased or reprogrammed.
3) SRAM cells are volatile.	3) They are non-volatile.
4) SRAM based FPGAs typically use larger logic cells with fewer inputs and outputs.	4) They have more inputs and outputs with small number of gates in logic cells.
5) Standard fabrication process.	5) A complex fabrication process.

4b) i) CMOS inverter diagram 2 Marks, Transfer curve diagram with explanation 4 Marks





Voltage transfer curve of CMOS inverter

From voltage transfer curve, we define five voltage points.

- V_{OH} is defined as the maximum output voltage when output level is at logic 1 ($V_{OH} = V_{DD}$).
- V_{OL} is the minimum output voltage when the output level is logic 0 ($V_{OL} = 0$).
- V_{IL} is the maximum input voltage which can be read as logic 0.
- V_{IH} is the minimum input voltage which can be read as logic 1.
- V_{th} is the input voltage at input voltage = output voltage.
- V_{top} = threshold voltage of PMOS
- V_{ton} = threshold voltage of NMOS.

Calculation of V_{OH}

- When $V_{in} < V_{ton}$, the NMOS transistor is cut off and PMOS is in linear region. The drain current of NMOS transistor $I_{Dn} = 0$, so $I_{Dp} = 0$.

$$I_{Dn} = I_{Dp} = 0$$

So drain source voltage of PMOS is also zero.

$$V_{DSp} = V_{DD} - V_{out}, \text{ but } V_{DSp} = 0$$

$$V_{out} = V_{DD} = V_{OH}$$

Calculation of V_{OL}

When the input voltage exceeds $V_{DD} + V_{top}$, the PMOS transistor is cut off and NMOS transistor is on. So $I_{Dp} = I_{Dn} = 0$ and $V_{DSn} = 0$

Hence, $V_{out} = V_{DSn} = 0 = V_{OL}$

Calculation of V_{IL}

V_{IL} is the smaller of two input voltages for which slope of voltage transfer curve is –1. When input is low, the NMOS transistor is in saturation and the PMOS transistor is in linear region.

$$V_{out} = (V_{in} - V_{Top}) + \sqrt{(V_{in} - V_{Top})^2 - 2\left(V_{in} - \frac{V_{DD}}{2} - V_{Top}\right)V_{DD} - \frac{k_n}{k_p}(V_{in} - V_{Ton})}$$

Calculation of V_{IH}

Since V_{IH} is the higher value of input voltage for which the slope of the voltage transfer curve is equal to zero. Also the NMOS transistor is in linear region and the PMOS transistor operates in saturation.

$$V_{IH} = \frac{V_{DD} + V_{Top} + \frac{k_n}{k_p}(2V_{out} + V_{Ton})}{1 + \frac{k_n}{k_p}}$$

Calculation of V_{th}

The inverter threshold voltage V_{th} is defined as that voltage for which $V_{in} = V_{out}$. When input and output are same, the PMOS and NMOS transistors are in saturation.

$$V_{th} = \frac{V_{Ton} + \sqrt{\frac{k_p}{k_n}}(V_{DD} - V_{Top})}{1 + \sqrt{\frac{k_p}{k_n}}}$$

4 b II) explanation with diagram 6 marks

All 'types' of objects defined above can be declared using type declaration.
A type declaration defines the name of type and the range of type.
A type declaration can be seen as

```
Type . type_name is type Mark;
```

Type marks are different methods of specifying a type.
These types can be classified into four broad categories as :

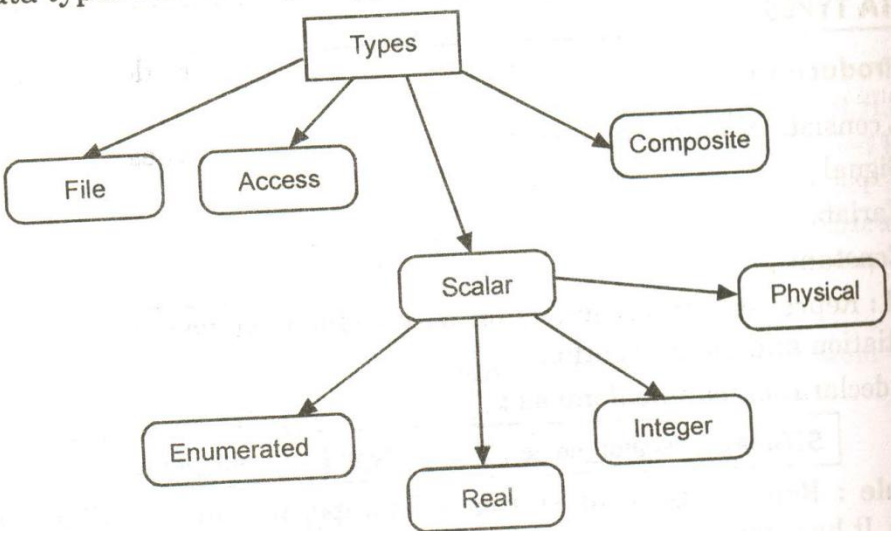
(1) Scalar types
(2) Composite types
(3) Access types
(4) File types

Scalar type includes all simple data types such as integer, real, physical, enumerated. **Access types** are equivalent to pointers in typical programming languages.

Composite data types include arrays and records.

Finally the last type **file types** give designer ability to declare file objects with designer defined file types.

These data types can be diagrammatically defined as :



(a) Integer types :

These are mathematical integer and range is from – 2, 147, 483, 647 to 12, 147, 483, 647

(b) Real types :

Real types are used to declare objects that emulate mathematical real number. The minimum range of real numbers is from –1. ® E + 38 to 1.0 E + 38

(c) Enumerated types :

This is very powerful tool for the abstract modeling.

All the values of the enumerated types are user defined. These values can be identifiers or single character literals.

An identifier is like name e.g. x, abc and black.

Whereas character literals are single character enclosed in quotes, such as 'x', '1' and '0'.

Example :

A typical example for enumerated type is

TYPE COLOR is (Red, Yellow, Blue, Green, Orange). Each identifier in type has specific position in the type. The first identifier has a position number of 0, the next position number of 1 and so on.

Hence 3 means green and 4 means orange.

(d) Physical types :

Physical types are used to represent physical quantities such as distance, current time and so on.

Base values are provided first then successive units are defined in terms of this base unit.

The smallest unit representable is one base unit, the largest is determined by the range specified in the physical type declaration.

An example of physical quantity current is shown as

Example (1)

```
TYPE current is RANGE 0 to 1000000000
UNITS
    na;                    --- nano amperes
    ua  = 1000 na; -- microamps;
    ma = 10000 μa --- milliamps
    a = 1000 ma ---- amps
END UNITS;
```

The type definition begins from name of type of physical quantity. First unit declared in section UNITS is 'na' i.e nanoamperes.

ua, ma, a are next units that are defined in terms of previous units.

As VHDL is dealing with digital circuits, here timing is important. Therefore for physical type, time is important example to deal with.