

B.E.

Sixth Semester Examination, May-2009

Digital System Design (CSE-308-Q)

Note : Attempt any *five* questions.

Q. 1. (a) What is the data object? Explain its all categories.

Ans. Data Object : A data object holds a value of a specified type. It is created by means of an object declaration as an example :

```
variable COUNT : INTEGER;
```

This results in the creation of a data object called COUNT, which can hold integer values. The object COUNT is also declared to be of variable class.

Every data object belongs to one of the following four classes :

1. Constant : An object of constant class can hold a single value of a given type. This value is assigned to the constant before simulation starts and the value cannot be changed during the course of simulation. For a constant declared within a subprogram, the value is assigned to the constant everytime the subprogram is called.

2. Variable : An object of variable class can also hold a single value of given type, however in this case, different values can be assigned to the variable at different times using a variable assignment statement.

3. Signal : An object belonging to the signal class holds a list of values, which include the current value of the signal and a set of possible future values that are to appear on the signal. Future value can be assigned to a signal using a signal assignment statement.

4. File : An object belonging to the file class contains a sequence of values, values can be read or written to the file using read procedure and write procedure, respectively.

An object declaration is used to declare an object, its type and its class.

Q. 1. (b) Explain all types of overloading.

Ans. Overloading are Basically Two Types :

1. Subprogram Overloading : Sometimes it is convenient to have two or more subprogram with the same name. In such a case, the subprogram name is said to be overloaded; also, the corresponding subprograms are said to be overloaded. For example :

```
function COUNT(ORANGES : INTEGER) return INTEGER;
```

```
function COUNT(APPLE : BIT) return INTEGER;
```

Both functions are overloaded since they have the same name, COUNT. When a call to either function is made it is possible to identify the exact function to which call is made from the type of actual passed because they have different parameter type.

For Example : the function call

```
COUNT(20)
```

Refers to the first function, since 20 is the type of INTEGER while the function call.

```
COUNT(1)
```

Here is another example :

```
function TO-CHARACTER (ARG : STD_LOGIC)
```

```
return CHARACTER
function TO_CHARACTER(ARG : STD_LOGIC_VECTOR)
return CHARACTER.
```

Here, both function are overloaded. The function call

```
TO_CHARACTER('0')
```

2. Operator Overloading : Operator overloading is one of the most useful features in the language. When the standard operator symbol is made to behave differently based on the type of its operand. The operator is said to be overloaded.

The need for operator overloading arises from the fact that the predefined operators in the language are defined only for operands of certain predefined types. For example, the and operation is defined for arguments of type BIT and BOOLEAN and for one-dimensional array of BIT and BOOLEAN only.

What if the argument here of type MVL, in such a case, it is possible to argument the 'and' operation as a function that operates on argument of type MVL, the and operator is then said to be overloaded, the operator in expression;

S_1 and S_2

Where S_1 and S_2 are of type MVL, would refer to the and operation that was defined by the model. Write as a function the operator in the expression

clk 1 and clk 2

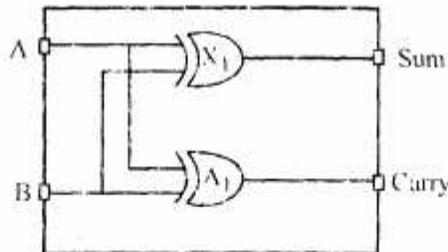
Where clk 1 and clk 2 are the type of bit. Would refer to predefined and operator.

Q. 2. (a) Explain entity with syntax and example.

Ans. Entity : A hardware abstraction of this digital system is called an 'entity'. An entity X, when used in another entity Y, becomes a component for the entity Y.

Therefore, a component is also an entity depending on the level at which you are trying to model

Entity Declaration : The entity declaration specifies the name of the entity being modeled and lists the set of interface ports, ports are signals through which the entity communicates with the other models in its external environment.



Here is an example of an entity declaration for the half-adder circuit shown.

```
entity HALF_ADDER is
port (A : B; in BIT; SUM, CARRY; out BIT);
end HALF_ADDER;
```

This is a command line.

The entity, called HALF_ADDER, has two input port, A and B (the mode in specifies input port) and two output ports, SUM and CARRY. BIT is a proved type.

Q. 2. (b) In behavioural style of modeling, explain transport and inertial delay.

Ans. Behavioural Style of Modeling : The behavioural modeling of an entity as a set of statement that are executed sequentially in the specified order.

This set of sequential statements. For example, consider the following behavioural model for the DECODER 2 × 4 entity.

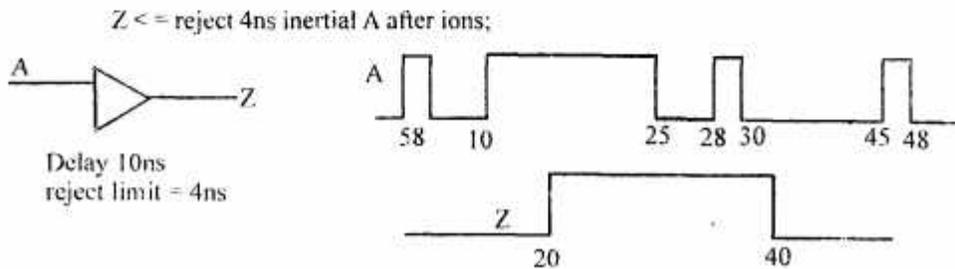
architecture DEC_SEQUENTIAL of DECODER 2 × 4 is

```
begin
    process (A, B, ENABLE)
        variable ABAR, BBAR; BIT;
    begin
        ABAR := not A;
        BBAR := not B;
        if ENABLE='1' then
            Z(3) <= not (A and B);
            Z(0) <= not (ABAR and BBAR);
            Z(2) <= not (A and BBAR);
            Z(1) <= not (ABAR and B);
        else
            Z <= "1111";
        end if;
    end process;
end DEC_SEQUENTIAL;
```

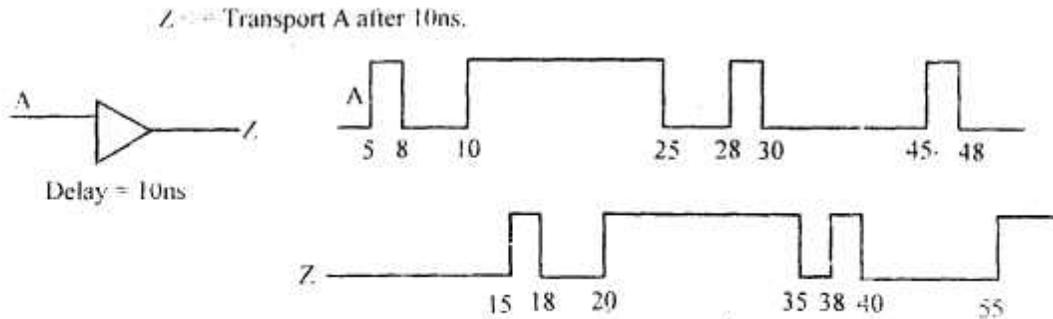
Inertial Delay : Inertial delay models the delays often found in switching circuits. An input value must be stable for a specified pulse rejection limit duration before the value is allowed to propagate to the output. The syntax of such a specification in the signal assignment is :

```
Signal-object <= [[reject pulse-rejection-limit+] inertial]
expression
after inertial_delay_value;
```

Figure shown a simple example of a non-inverting buffer with an inertial delay of 10ns and a pulse rejection limit of 4ns.



Transport Delay Model : Transport delay models the delays in hardware that do not exhibit any inertial delay. This delay represents pure propagation delay : that is, any change on input are transported to the output, no matter how small the keyword is transport



Q. 3. (a) Explain component declaration and component installation in brief.

Ans. Component Declaration : A component instantiated is a structural description must first be declared using a component declaration. A component declaration declares the name and the interface of a component. The interface specifies the mode and the type of ports. The syntax of a simple form of component declaration is :

```
[component component-name ]is]
[port (list-of-interface-ports);]
end component [component-name]
```

The component-name may or may not refer to the name of an entity already existing in a library.

Component Instantiation : A component instantiation statement defines a subcomponent of the entity in which it appears. It associates the signal in the entity with the port of subcomponent.

A format of a component instantiation is :

```
Component-label : component-name [portmap (association-list)];
```

The component label can be any legal identifier and can be considered as the name of the instance. The component name must be the name of a component declared earlier using a component declaration.

The association-list- associates signal in the entity, called actuals, with the port of components called formats an actual may be signal.

Q. 3. (b) Differentiate between a process and wait statement. Can they be used simultaneously in a program?

Ans. Process Statement : A process statement contains sequential statements that describe the functionality of a portion of an entity in sequential terms. The syntax of a process statement is :

```
[process-label : ] process [(sensitivity-list)] [is]
[process-item-declarations]
begin
    sequential-statements; these are →
    variable statement :
    signal assignment status
    wait statement;
end process [process label];
```

Wait Statement :

When a process has a sensitivity list, it always suspends after executing the last sequential statement in the process.

The wait statement provides an alternate way to suspend the execution process. There are three basic forms of the wait statement.

- wait on sensitivity_list;
- wait until boolean_expression;
- wait for time_expression;

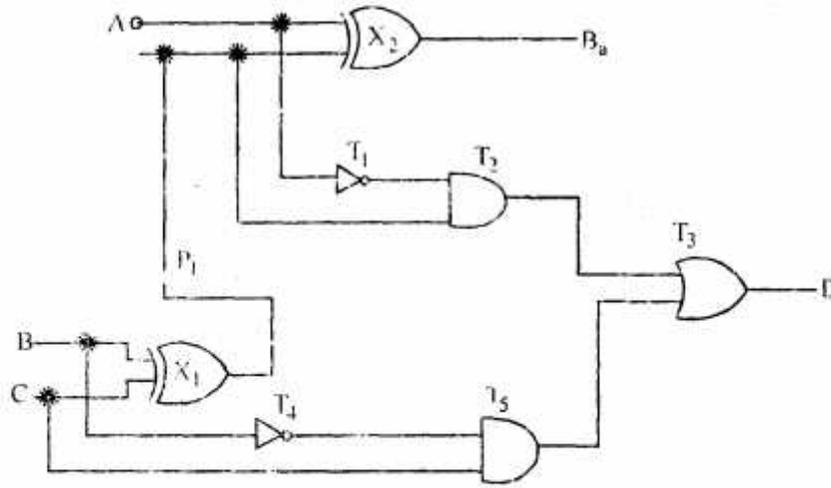
They may also be combined in a single wait statement.

Wait on sensitivity list until boolean expression time expression.

Yes, process statement and wait statement can be used simultaneously in a program.

Q. 4. (a) Write down VHDL code for full subtractor.

Ans. VHDL Code for Full Subtractor :



$$D = [\bar{A}(B \oplus C)] \oplus (\bar{B}C)$$

$$B_a = [A \oplus (B \oplus C)]$$

```

entity FULL_SUBTRACTOR is
Port (A, B, C : in BIT; D, Ba : OUT BIT);
end FULL_SUBTRACTOR
architecture FS_MIXED of FULL_SUBTRACTOR is
component XOR2
Port (B, C : in BIT; P1 : OUT BIT)
end component;
signal S1 : BIT;
begin
    X1 : XOR2 port map (A, B, C)
    Variable T1, T2, T3, T5, T4, P1 BIT;

```

```
begin
    T1 := NOT A ;
    T2 := T1 and A;
    T4 := NOT B;
    T5 := T4 and C;
    T3 := T5 or T4;
    D <= T3 or T5 or T4;
end process;

Bn <= A XOR P1;
end FS_MIXED;
```

Q. 4. (b) Write down VHDL code for look ahead carry circuit.

Ans. Look-ahead Carry VHDL Code:

```
C_1_addr.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY C_1_addr IS
PORT
(
    X-in : IN STD_LOGIC_VECTOR (7 DOWN TO 0);
    Y-in : IN STD_LOGIC_VECTOR (7 DOWN TO 0);
    Carry_in : IN STD_LOGIC;
    SUM : OUT STD_LOGIC_VECTOR (7 DOWN TO 0);
    Carryout : OUT STD_LOGIC
);
END C_1_addr;
ARCHITECTURE behavioral OF C_1_addr IS
SIGNAL h_sum : STD_LOGIC_VECTOR (7 DOWN TO 0);
SIGNAL Carry_generate : STD_LOGIC_VECTOR (7 DOWN TO 0);
SIGNAL Carry_propagate : STD_LOGIC_VECTOR (7 DOWN TO 0);
SIGNAL Carry_in_internal : STD_LOGIC_VECTOR (7 Down To 1);
BEGIN
    h_sum <= x_in XOR Y_in
    Carry_generate <= X-in AND Y-in;
```

```
Carry_propagate <= x-in OR y-in;
PROCESS (carry-generate, carry_propagate, carry_in_internal)
BEGIN
    Carry_in_internal (1) <= Carry_generate (0) OR
    (Carry_propagate (0) AND carry_in);
    inst : FOR i IN 1 to 6 LOOP
        Carry_in_internal (i+1) <= carry_generate (i) OR
        (carry_propagate (i) AND carry-in-internal (1));
    END Loop;
    Carry-out <= carry_generate (7) OR (Carry_propagate (7) AND carry_in_internal (7));
END PROCESS;
    Sum (0) <= h-sum (0) XOR Carry_in;
    Sum (7 DOWN TO 1) <= h_sum (7 DOWN TO 1) XOR
    Carry_in_internal (7 DOWN TO 1);
--End behavioural;
```

Q. 5. (a) Write down VHDL code for Modulo-5 counter in data flow modelling.

Ans. VHDL Code for Modulo-5 Counter :

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity pulse-5clk is
port
'clk, reset : in std_logic; go, stop; in std_logic
    pulse : out std_logic;
pulse : pulse-5 clk;
architecture fan-arch of pulse-5 clk is
type fsm-s1
OR
Library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
entity COUNT-5 is
Port
(PIN : in std_logic_vector (7 down to 0);
    clk : in std_logic;
    LOAD : in std_logic;
```

```

        POUT : Out std_logic_vector (4 down to 0)
    );
end counts;
architecture behaviour of counts is
begin
    clk_proc : process (CLK)
        variable COUNT : unsigned (4 DOWN TO 0)
            := "0000";
        begin
            if (CLK_EVENT AND CLK = '1') then
                if LOAD = '1' then
                    COUNT := DIW;
                else COUNT := COUNT + 1;
                end if
            end if
            POUT <= COUNT after 500 ps;
        end process clk_proc;
    end behavior
end process.

```

Q. 5. (b) Write down VHDL code for 4 bit up/down counter in structural modelling.

Ans. Structural Modeling for 4 Bit Up/Down Counter :

```

entity UP_DOWN is
    Port (CLK, CNT_UP, CNT_DOWN : in BIT;
          Q0, Q1, Q2, Q3 : Buffer BIT);
end UP_DOWN;
architecture COUNTER of UP_DOWN is
    component JK_FF
        Port (J, K, CLK : in BIT; Q, QN : buffer BIT);
    end component;
    component AND_2
        Port (A, B : in BIT; C : OUT BIT);
    end component;
    component OR_2
        port (A, B : in BIT; C : OUT BIT);
    end component;
    signal S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12 : BIT;

```

```
begins
    JK1 : JK_FF port map ('1', '1', CLK, Q0, S1);
    A1 : AND2 port map (CNT_UP, Q0, S2);
    A2 : AND2 port map (S2, S3, S4);
    JK2 : JK_FF Port map ('1', '1', S4, Q1, S5);
    A3 : AND2 portmap (Q1, CNT_UP, S7);
    A4 : AND2 portmap (S5, CNT_DOWN, S6);
    Q2 : OR2 portmap (S7, S6, S8);
    JK3 : JK-FF port map ('1', '1', S8, Q2, S9);
    A5 : AND2 port map (Q2, CNT_UP, S11);
    A6 : OR2 port map (S10, S10, S12);
    JK4 : JK-FF port map ('1', '1', S12, Q2, Open)
end caster;
```

Q. 6. (a) Explain the description of processor implementation in VHDL.

Ans. Processor Implementation in VHDL : The performance of the software system is dramatically affected by how well software designer understand the basic hardware technologies at work in a system."

Simple Implementation Scheme : In the first section, it is simple implementation scheme of a MIPS subset the basic hardware of the microcontroller, data path and its control is developed by step by step and implemented in VHDL.

Multicycle Implementation : Establishing that the efficiency of a long-cycle implementation is not likely to be very good the processor speed is improved by using multicycle implementation. Then instruction are allowed to take different member of clock cycles and functional unit can be shared within the execution of single instruction.

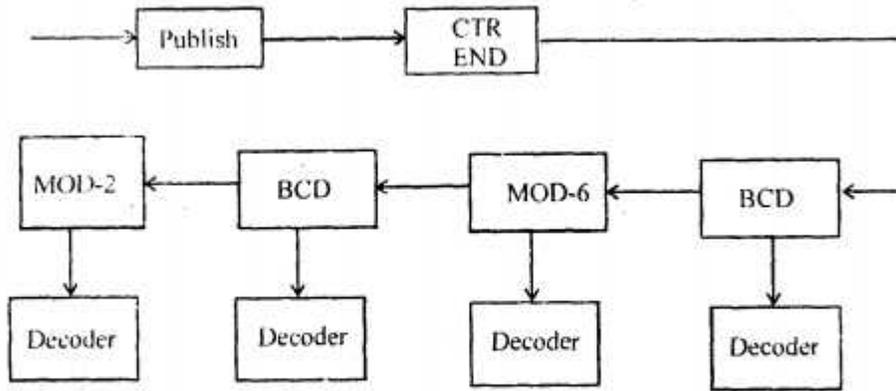
Enhancing Performance by Pipelining : In order to enhance to performance and to get very fast processor another implementation technique called. Called pipelining in introduced. Multiple instruction are overlapped in execution. So that some stages are asking parallel.

Q. 6. (b) Implement a simple micro computer using VHDL.

Ans. Design of Simple Microcomputer Using VHDL :

```
Library ieee;
use ieee.std_logic_1164.all;
ENTITY Fig is
PORT
    (Clk : IN STD_LOGIC);
```

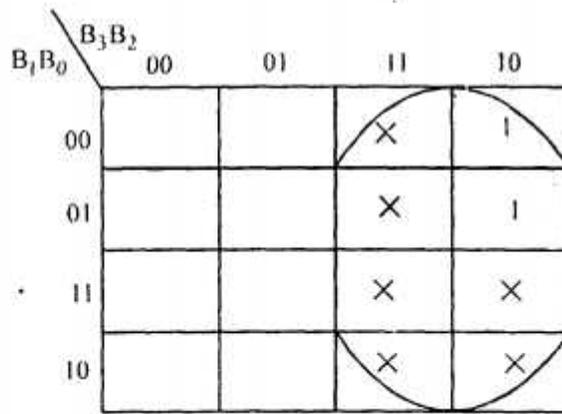
```
Col: IN STD_LOGIC_VECTOR (3 DOWN 0);
rol: OUT STD_LOGIC_VECTOR (3 DOWN 0);
d: OUT STD_LOGIC_VECTOR (3 DOWN 0);
dav: OUT STD_LOGIC;
END fig;
ARCHITECTURE VHDL of fig is
SIGNAL freeze: STD_LOGIC;
SIGNAL data: STD_LOGIC_VECTOR (3 DOWN TO 0)
BEGIN:
    PROCESS (Clk)
        VARIABLE Ring: STD-LOGIC-VECTOR (3 DOWN TO 0);
        BEGIN
            IF (Clk: EVENT AND Clk = '1') THEN
                If Freeze: '0' THEN
                    CASE ring is:
                        WHEN "1110" => Ring := "1101"
                            :
                    END Case:
                END if;
                dav <= freez;
            END if;
            row <= ring
            WHEN "1110" => data (3 DOWN TO 2) <= "00";
            :
        ENDCASE
        CASE col IS
            WHEN "1110" => Data (1 DOWN TO 0) <= "00"; freeze <= '1';
            :
        ENDCASE;
        If freeze = '1' Then d <= data;
        ELSE d <= 'zzz'
        END if
    END process.
```



Q. 7. (a) Implement BCD to Binary converter using PAL.

Ans. BCD to Binary Converter Using PAL :

B_3	B_2	B_1	B_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1



$$B_3 = b_3$$

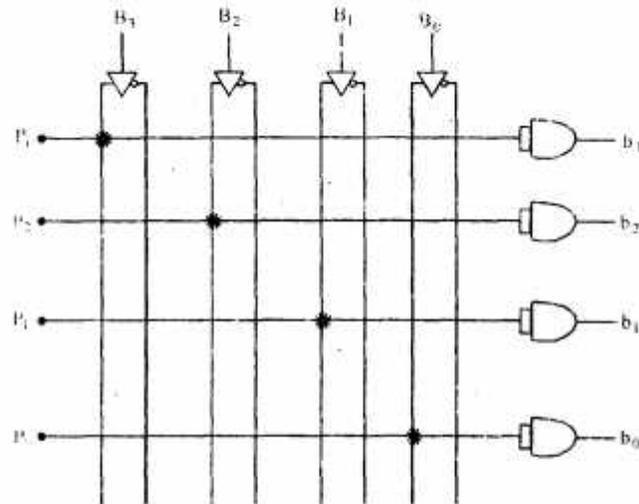
Similarly we will get
(by taking care of don't cares)

$$b_2 = B_2$$

$$b_1 = B_1$$

$$b_0 = B_0$$

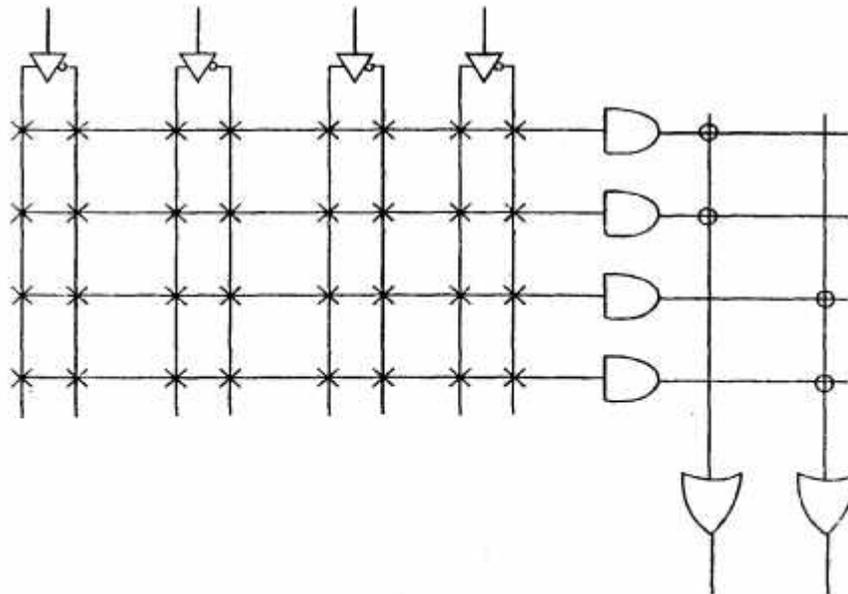
Designing by PAL :



Q. 7. (b) Differentiate between PAL & PLA.

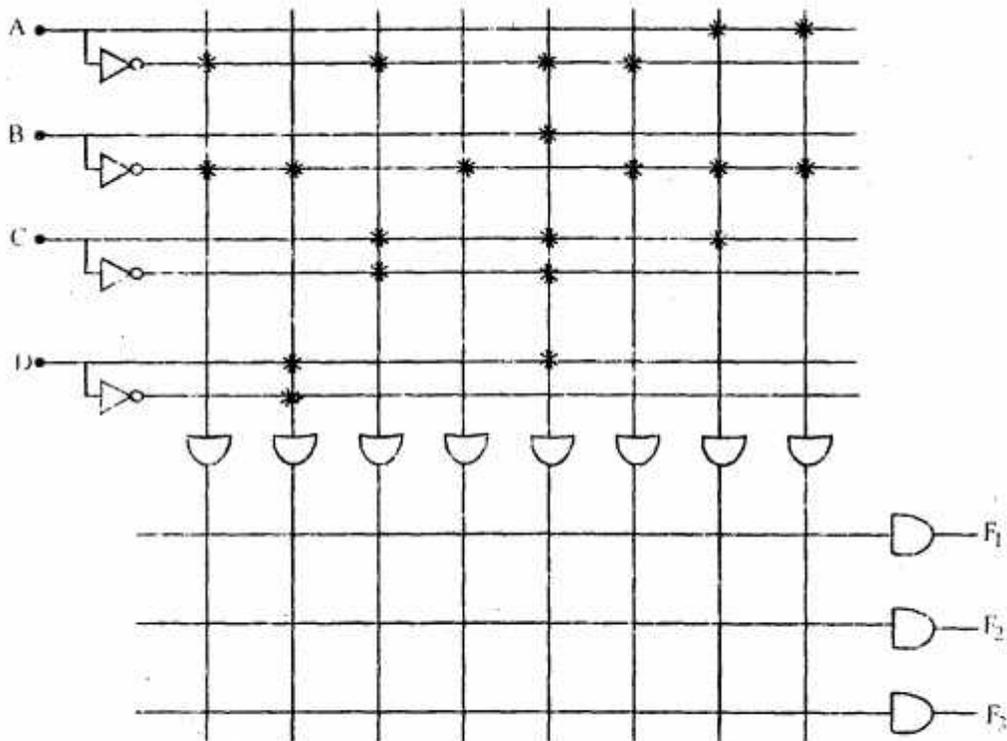
Ans. PLA & PAL : PAL concept-implemented by monolithic memories constraint topology of the OR Array- i.e., the OR array cannot be fully programmed.

A given column of the OR array has access to only a subset of the possible product terms.



PLA also consist of both AND and OR

Gate and both can be programmed as in fig.



Q. 8. Write short notes on the :

(a) GAL

(b) Generics.

Ans. (a) GAL (Gate Array Logic) : The basic component used in VLSI design to the gate array. A gate array consist of a pattern of gates fabricated in an area of silicon, that is repeated thousand to hundred thousand gates are fabricated within a single IC chip depending on the technology used.

The design with gate arrays requires that the customer provide the manufacturer to desired interconnection pattern. The first few levels of fabrication process are command and independent of the final logic function.

Additional fabrication steps are required to interconnected the gate according to specification given by the designer.

A field programmable gate array logic (FPGAL) is a VLSI circuit that can be programmed in the user's location. A typical FPGA consist of an array of hundred or thousand logic blocks surrounded by the programmable input and output blocks and connected together via programmable interconnection.

There is a wide variety of internal configuration within this group of devices, the performance of each devices type depends on the circuit contained in their logic blades and efficiency of their programmed interconnection.

(b) Generics : It is often useful to pass certain types of information into a design description from its environment. Examples of such information are rise and fall delays and the size of interface ports.

This is accomplished by using generics. Generics of any entity are declared along with its ports in the entity a declaration.

An example of a generic N-input and gate is :

```
entity AND_GATE is
  generic (N : NATURAL);
  Port (A : in BIT_VECTOR (1 to N) : Z : OUT BIT);
end AND_GATE;
architecture GENERAL_EX of: AND_GATE is
  begin
    process (A)
      variable AND_OUT : BIT;
    begin
      AND_OUT := '1';
      for K in 1 to N loop
        AND_OUT := AND_OUT and A (K);
      exit when AND_OUT = '0';
    end loop;
    Z <= AND_OUT;
  end process;
end GENERAL_EX;
```

A generic declares a constant object in and can be used in entity declaration and its corresponding architecture bodies.

The value of a generic must be determined at elaboration time.