DECEMBER 2019

Q1)

(a)

Functions	Procedure
→ Functions return single value through RETURN statement	Procedures return zero or more values which are returned through parameters passed to procedure and no RETURN statement is used.
→ Function may accept zero or more input arguments with default mode IN, only class and type of parameters need to be specified in function declaration.	Since procedures return values through its passed arguments only IN, INOUT and OUT mode of arguments is allowed. For every argument passed mode, class and type need to be specified in declaration.
→ Input parameter passed to function are essentially a signal or a constant object.	Parameter passed to procedure can be a signal, constant, variable or file object. Default for IN mode: constant and for OUT/INOUT: variable.
→ A function call is always part of an expression or assignment statement	A procedure call is an independent sequential statement
→ Executes in zero simulation time as wait statement are not to be used in functions.	May or maynot simulate in zero simulation time as procedure can have a wait statement.

(b) RTL: A level of description of a digital circuit in which operation of spread over several clock cycles describes circuit behaviour as a flow of data between registers.

(c) CONSTANT N : INTEGER := 32;
CONSTANT PI : INTEGER := 3.14;

(f)

Signal	Variable
→ Signals are VHDL objects used to represent wires and interconnections	Variables represent temporary storage in VHDL
→ Assignment operator for signal is '<='	Assignment operator for variable is ':='
→ Values of signal are updated only after default delta delay or specified user delay.	Values of variables are updated immediately on execution of variable assignment statement
→ Signals require event scheduling and synchronization of signal drivers.	No event scheduling and synchronization is required in case of variables.
→ Use of signals has greater processing overheads and consumes more memory	Use of variable consumes less memory due to no event scheduling.
→ Use of signal is allowed with all styles of modelling	Use of variable is allowed only with sequential style of subprogram

(b)

	Programmable logic array (PLA)	Programmable Array logic (PAL)
Stands for		
Construction	Programmable array of AND, OR gates	Programmable array of AND gates and fixed array of OR gates
Availability	Less available	More available
Flexibility	Provides more programming flexibility	Offers less flexibility but more likely used
Cost	Expensive	Intemediate cost
Number of Functions	Large	Limited
Speed	Slow	High

PLA is used

- To provide control over datapath
- As a counter
- As a decoder
- As a BUS interface in programmed I/O.

PAL is used

- As a counter
- As state machine
- As a decoder
- As a bus interface
- In Combinational Logic Circuit

Ans 2) (a) VHDL Code for 4 Bit Up Counter [Diagram needed]

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

(b) Structural Style of Modelling

It is the simplest style of modelling in VHDL and is based on the lowest level of abstraction of the entity that is to be designed. It is a representation of circuit or entity in terms of interconnected components.

In structural style of modelling architecture body of entity that is to be designed consists of

- component declaration
- component instantiation

Component declaration is similar to entity declaration where each and every component to be used in the architecture description

is declared with its unique name - identifier, its number, and type of input and output ports. Component declaration is done in declarative section of architecture body.
Component instantiation is mapping between actual and formal ports of the entity and components, respectively. Component instantiation is port map written inside executable section of architecture body.
eg. Half Adder using Structural Modelling

```
entity half_adder is
  port (a, b : in bit;
        sum, carry : out bit);
end half-adder;
architecture struct of half-adder is
component and_gate is
    port (x, y : in bit;
          z : out bit);
  end and_gate;
component xor-gate is
    port (l, m : in bit;
          n : out bit);
    end xor_gate;
begin
X1 : xor_gate port map (a, b, sum);
A1 : and_gate port map (a, b, carry);
end struct;
```

Dataflow Modelling

It expresses functionality of circuit ~~is~~ as a flow of data through buses and interconnections. It does not use the structure or components but uses logical expressions to express the flow of data from input to output through signals. It is also the concurrent

style of modelling. It is closer to RTL level simulation. It corresponds to middle level of abstraction.

eg. Half Adder using dataflow modelling

```
entity half_adder is
  port (a, b : in bit;
        sum, carry : out bit);
  end half_adder;
architecture dataflow of half_adder is
begin
  sum <= a xor b;
  carry <= a AND b;
end dataflow;
```

Behavioral Modelling

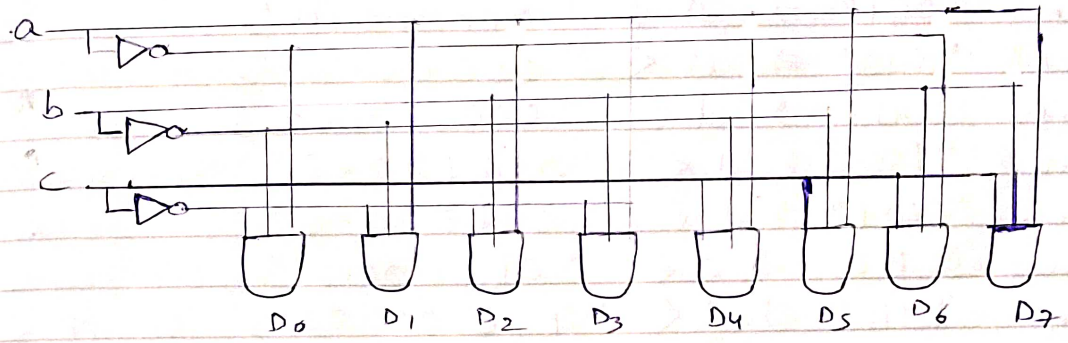
This style of modelling emphasizes on the behavior of the circuit i.e behavior of output with respect to changes in input. This modelling is sequential style of modelling, so the order of statement is important. It is also called algorithmic style of modelling. It corresponds to highest level of abstraction. It uses special construct called PROCESS.

eg. Half adder using behavioral modelling

```
entity half_adder is
  port (a, b : in bit;
        sum, carry : out bit);
  end half_adder;
architecture behave of half_adder is
  begin
    process (a, b)
  begin
  sum <= a xor b;
```

carry <= a AND b;
end process;
end behave;

Ans 3) (a) 3X8 Decoder Circuit



c	b	a	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

VHDL Code

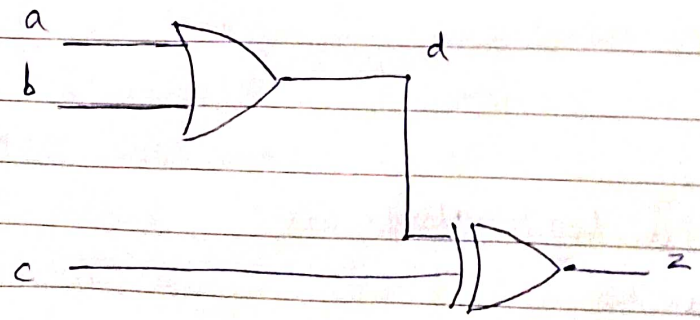
```
library ieee;
use ieee.std_logic_1164.all;
entity decoder38 is
port (a: in std_logic_vector(0 to 2);
      d: out std_logic_vector(0 to 7);
end decoder38;
architecture decoder_arch of decoder38 is
```

```
begin
process (a)
  begin
case a is
when "000" => d <= "00000001";
when "001" => d <= "00000010";
when "010" => d <= "00000100";
when "011" => d <= "00001000";
when "100" => d <= "00010000";
when "101" => d <= "00100000";
when "110" => d <= "01000000";
when "111" => d <= "10000000";
end case;
end process;
end decoder_arch;
```

(b) Delta Delay :

Delta (Δ) is an infinitesimal unit of time which is required for correct execution of concurrent statements that are executed at the same time. Though actual simulation time remains unaffected.

eg. Consider the following circuit



dataflow model is

```
entity dataflow is
port ( a, b, c : in  std_logic;
```

z : out std_logic);
end dataflow
architecture ex of dataflow is
signal d : std_logic
begin
d <= a OR b; -- statement 1
z <= c XOR d; -- statement 2
end ex;

When the value of 'a' and 'b' changes 'd' gets its new value, which in turn triggers statement 2 giving a new value to 'z'. These changes take place in same simulation cycle but infinitesimal delay δ is inserted between the execution of two statements.

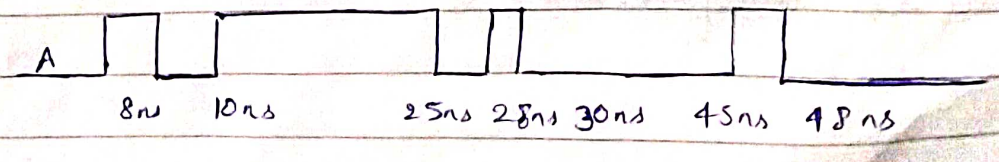
Inertial Delay:

→ In inertial delay model, an input must be stable for a specified pulse rejection limit duration so that this input value is propagated to the output after the specified inertial delay.

→ If input is not stable for specified pulse rejection limit then no output change takes place.

Syntax for Inertial Delay
signal-object <= reject pulse rejection limit inertial expression after inertial delay;

eg. Given signal assignment statement and input waveform
z <= reject 5ns inertial A after 10ns;



Output Waveform is

Transport Delay Model

→ Transport delay model is hardware that do not have inertial delay.

→ This delay represents pure propagation delay i.e. any change in input is transmitted to output after a specified delay even though the change in input is for very small duration

→ To use transport delay model, Keyword "transport" must be used.

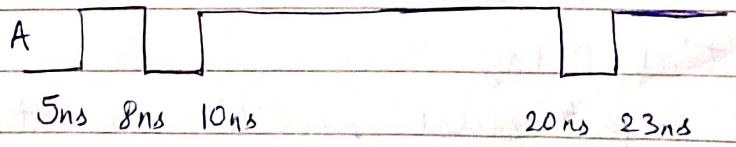
Syntax for transport delay model:

signal-object ⇐ transport expression after delay;

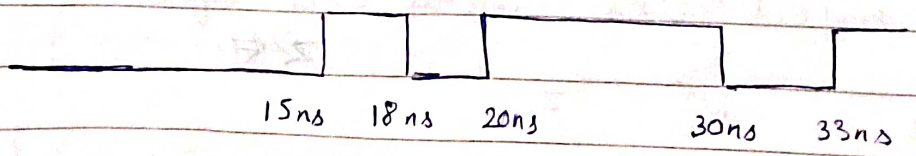
eg.

z ⇐ transport A after 10ns;

Input Waveform

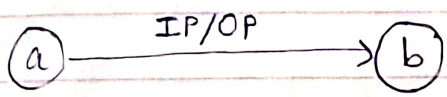
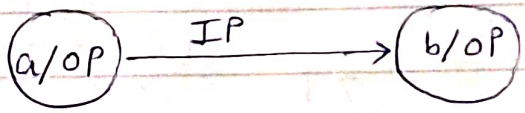


Output Waveform



Output Waveform

4)(a)

Mealy Machine	Moore Machine
→ The output is associated with each transition	The output is associated with each state.
→ Next state depends on present state and input	Next state depends on input
→ Number of states required are less	Number of states required are more to perform the same operation as compared to mealy machine.
→ The output mapping function is complex	The output mapping function is ~~less complex~~ simple
→ It can be represented as 	It can be represented as 

(b) Block Statement is used to group self-contained logical areas. Block statements are used to perform following functions:

(i) to disable some signal using guard expression
(ii) to limit the scope of declaration
(iii) to group logical areas in the program to improve readability.

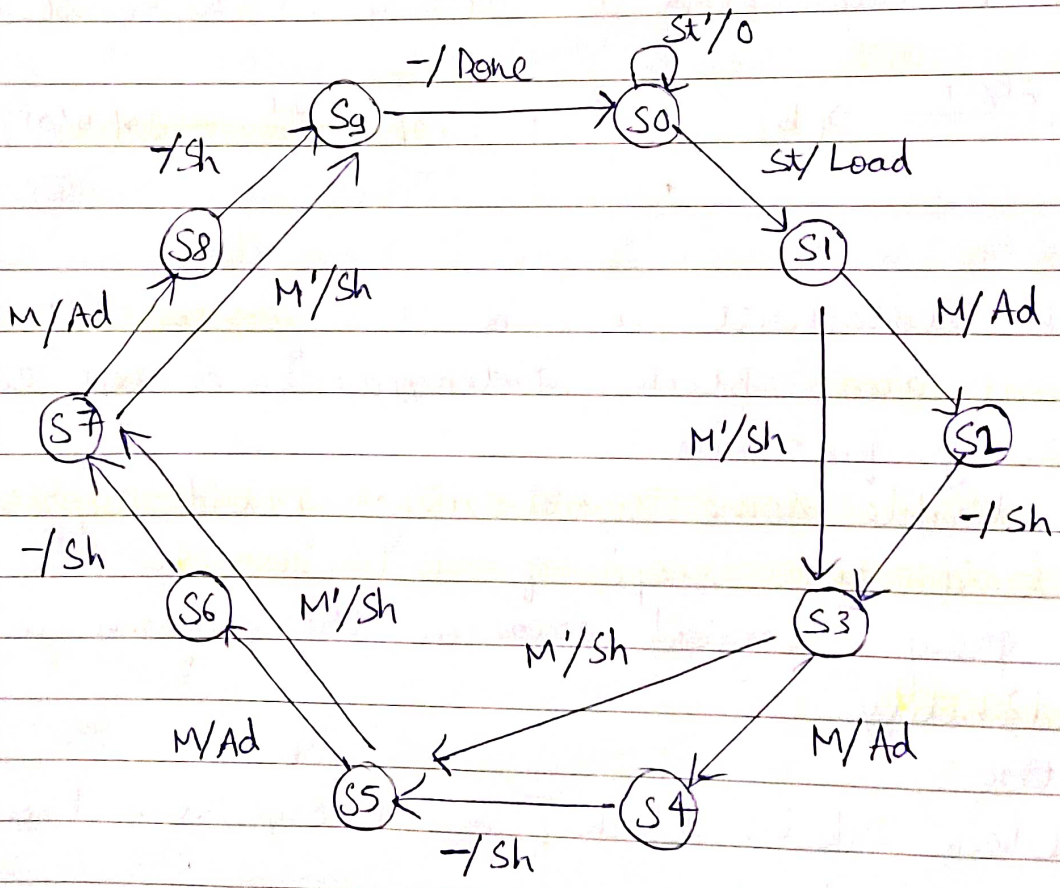
Syntax:

```
block label : block [ guard expression ] is
    block header
    block declarations
begin
    concurrent statements
end block [ block label ];
```

eg.

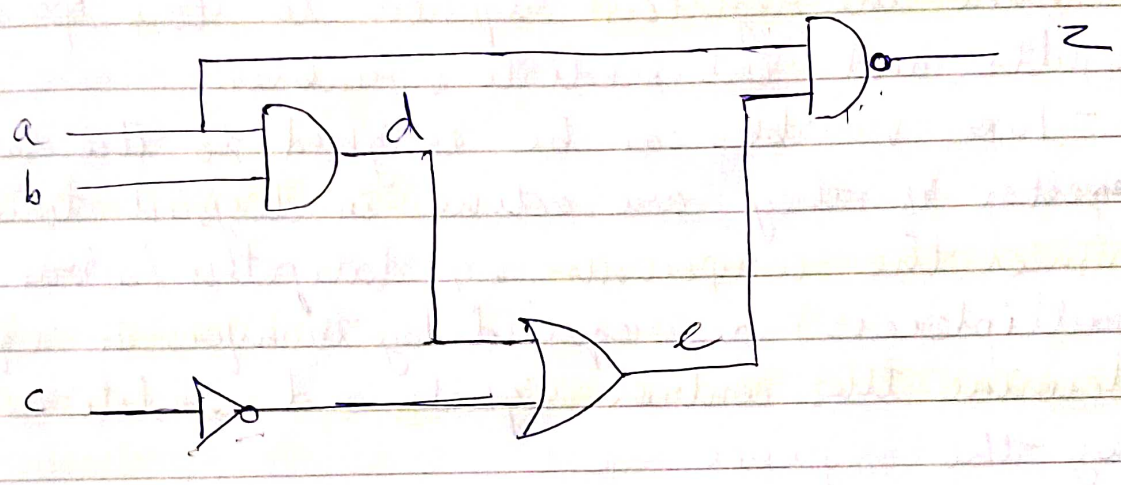
```
blk1 : block
  begin
  blk2 : block
    begin
      concurrent statements;
    end block blk2;
  concurrent statements;
end block blk1;
```

(c) State Graph for Binary Multiplier

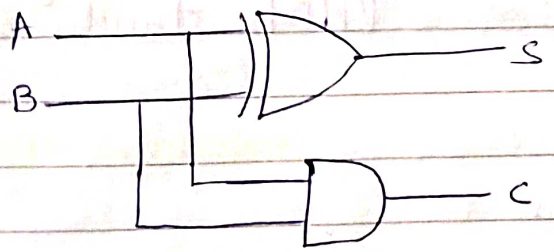


Ans 5)(a) entity dataflow is
port (a, b, c: in std_logic;
z: out std_logic);
end dataflow
architecture ex of dataflow is

signal d : std_logic
signal e : std_logic
begin
d <= a and b;
e <= d or not (c);
z <= not (a and e);
end ex;



Ans 6) (a)



Half Adder

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Code:
library ieee;
use ieee. std_logic_1164. all;
use ieee. std_logic_arith . all;
use ieee. std_logic_unsigned . all;
entity half_adder is
port (a, b : in bit; sum, carry : out bit);
end half_adder;
architecture data of half_adder is
begin
sum <= a xor b;

carry ← a and b;
end data;

(b) Data Subsystem is designed by following these steps

→ Determine the operator (functional units)
Two operators can be assigned to same functional unit if they form part of different group.

→ Determine the registers required to store operands, results and intermediate variables.
- Two variables can be assigned to the same register if they are active in disjoint time intervals.

→ Connect the components by datapaths (wires and multiplexers) as required by transfers in sequence.

→ Determine the control signals and conditions required by the sequence

→ Describe the structure of the data section by a logical diagram, a net list, or a VHDL structural description

Data Subsystem consists
(i) Storage Modules
(ii) Function Modules (operators)
(iii) Datapaths (switches and wires)
(iv) Control Points
(v) Condition Points

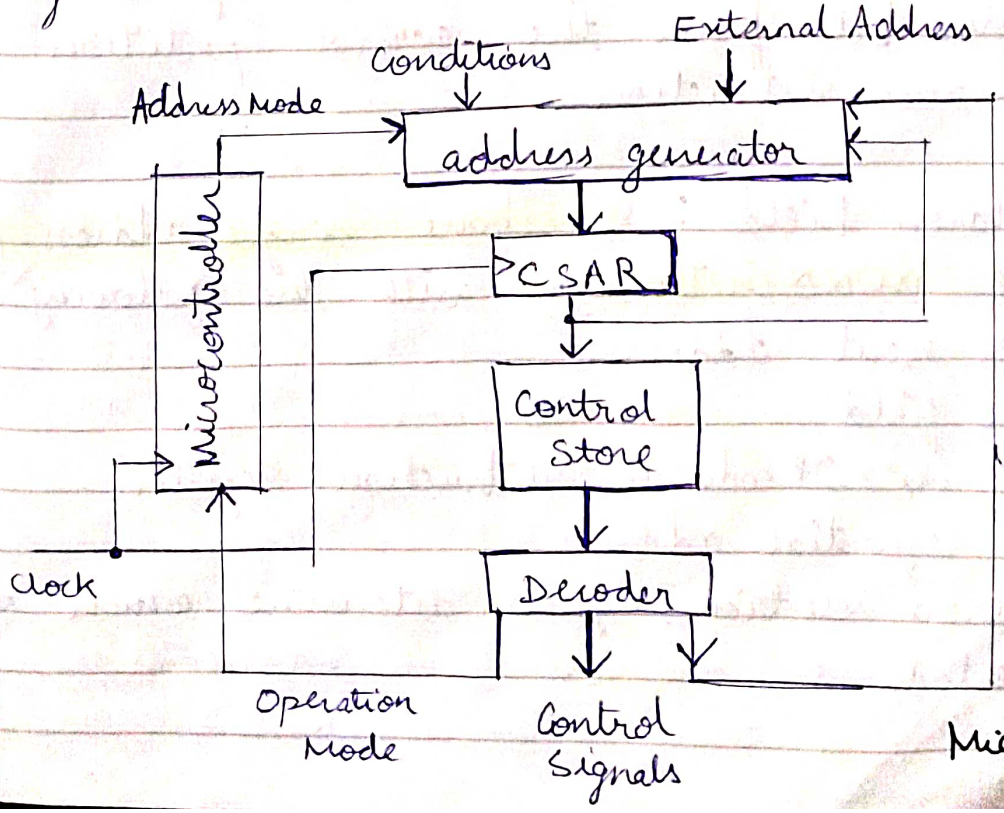
Design of Control Subsystem
→ Determine the register transfer sequence
→ Assign one state to each RT group
→ Determine state transition and output functions
→ Implement the corresponding sequential system.

Control Subsystem

→ Inputs: Control inputs to the system and conditions from the data subsystem
→ Outputs: control signals
→ One state per statement in register transfer sequence.
→ Transition function corresponds to sequencing
→ Output for each state corresponds to control signal

Ans 7) (a) Microprogrammed Controller consists of the following modules:

- Control store (CS) - a memory device (eg. ROM or RAM) containing the microprogram
- Control Store Address Register (CSAR) - a register containing the address of the microinstruction to be executed
- Address Generator - computes the address of the next microinstruction.
- Decoder - Generates control and other signals from the microinstruction.
- Microcontroller - Sequences the operation of the microprogrammed controller.



Microprogrammed Controller

(b) Microinstruction Sequencing:
Depending on the current microinstruction condition flags & the contents of the instruction register, a control memory address must be generated for the next micro instruction.

There are three general techniques based on the format of the address information in the microinstruction
1. Two Address Field
2. Single Address Field
3. Variable Format

1. Two address field: we provide two address field in each microinstruction and multiplexer is provided to select:
→ Address from the second address field
→ Starting address based on the OP code field in the current instruction.
The address selection signals are provided by branch logic module whose input consists of control unit flags plus bits from the control portition of the microinstruction.

2. Single Address Field: we have single address field in the micro instruction with the following options for the next address
→ Address field
→ Based on OP code in instruction register.
→ Next sequential address
The address selection signals determine which option is selected.

3. Variable Format

There are two different microinstruction formats. One bit designates which format is being used. The remaining bits are used to activate control signals. In the second format, some bits drive the branch logic module, and the remaining bits provide the address. With the first format, next address is either the next sequential address or an address derived from the instruction register. With the second format, either a conditional or unconditional branch is specified.

Ans 8) (a)

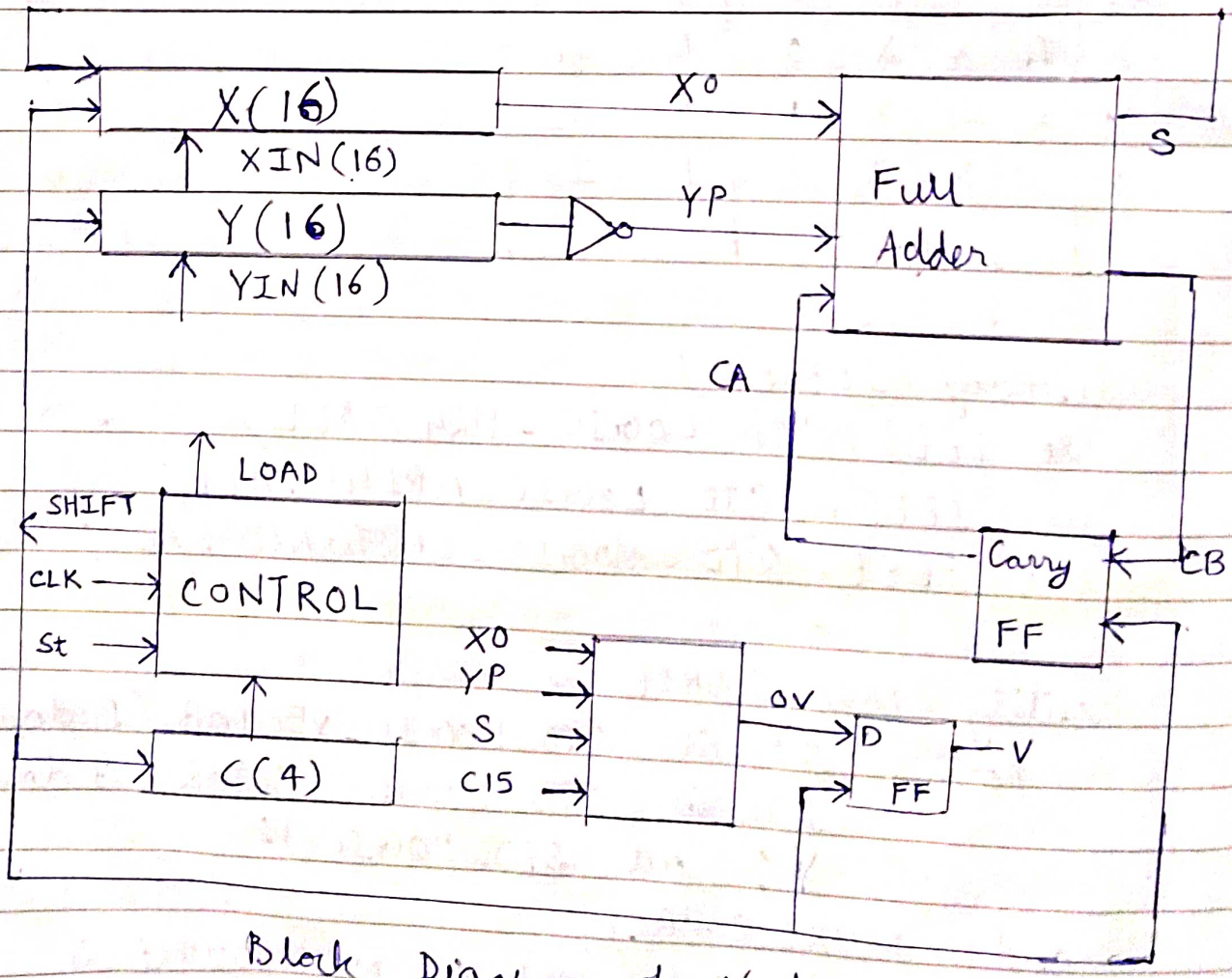
I_0	I_1	I_2	I_3	S0	S1	Y
I_0	X	X	X	0	0	I0
X	I_1	X	X	0	1	I_1
X	X	I_2	X	1	0	I_2
X	X	X	I_3	1	1	I_3

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

entity MUX_SOURCE is
  port (S: in STD_LOGIC_VECTOR (1 downto 0);
        I: in STD_LOGIC_VECTOR (3 downto 0);
        Y: out STD_LOGIC);
end MUX_SOURCE;
architecture Behavioral of MUX_SOURCE is
begin
```

```
process (S, I)
 begin
if ( S<= "00") then
  Y <= I(0);
else if (S <= "01") then
  Y <= I(1);
elsif (S <= "10") then
  Y <= I(2);
 else
  Y <= I(3);
end if;
end process;
end Behavioral;
```

(b)



Block Diagram of 16 bit serial adder with accumulator

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

entity seradder is
port ( N, clk : in std_logic;
A: in std_logic_vector (15 downto 0);
R: out std_logic_vector (15 downto 0);
B: in std_logic_vector (15 downto 0);
Co: out std_logic);
end seradder;
architecture mealy of seradder is
signal si, ci, cip, ld, sh : std_logic := '0';
signal K : std_logic := '0';
signal acc, regb : std_logic_vector (15 downto 0);
signal state : integer range 0 to 2 := 0;
signal nxst : integer range 0 to 2 := 0;
signal cnt : std_logic_vector (4 downto 0) := "00000";
begin
K <= '1' when (cnt = "10000") else '0';
process (clk)
  begin
if (clk = '1' and clk' event) then
 case state is
 when 0 => if (N = '1') then ld <= '1'; sh <= '1';
        acc <= A;
regb <= B; ci <= '0'; state <= state +1;
   else state <= 0; end if;
when 1 => if (K = '1') then sh <= '1'; state <= state+1;
        R <= acc; Co <= cip;
      else sh <= '1';
```

```
acc <= si & acc(15 downto 1);
regb <= regb(0) & regb(15 downto 1)
ci <= cip; cnt <= cnt + 1; end if;
when 2 => if (N = '0') then state <= 0;
    else nxst <= state; end if;
end case;
end if; end process;
si <= acc(0) xor regb(0) xor ci;
cip <= (acc(0) and regb(0)) or (acc(0) and ci) or
  (regb(0) and ci);
end mealy;
```

Ans 9) (a) EDA (Electronic Design Automation) Tools are used for design entry, simulation and synthesis.

→ ~~Design Entry: VHSIC and VHDL are used for~~

→ Design Entry: The information about the circuit to be designed is given in the form of a text file written in VHDL. The code giving the system specification and the logical expression to perform the desired function is written in HDL.

→ Simulation: Once design entry step is complete presynthesis simulation is carried out to check if the circuit works as intended. The simulation tool gives its output in the form of timing waveforms, where the waveforms describe the output values for the given set of inputs.

→ Synthesis: It expresses the circuit in terms of basic components in accordence with the logical expression that are used to describe the circuit design. It optimizes the circuit and checks for errors in design. Netlist, log and schematic are outputs of synthesis.

(c) There are five datatypes in VHDL

→ Scalar Types: They include numeric data types and enumerated data types. The numeric data type consists of integer, floating point (real) and physical types. Bit, Boolean and Character are all enumerated types.

→ Composite Types: Array and record types are composite data types. The value of these types are collection of their elements.

→ Access Types: They are pointers; they provide access to objects of a given data type.

→ File Types: They provide access to object that contain a sequence of values of a given type.

→ Other Types: They include the data types provided by usual external libraries.

2018

Ans 1

(a) Solved

(b) Solved

(c) By using generic we can use same design by creating different instances each having different parameter values. These parameter values which differ in various instances of the same design are called generic parameters.

Syntax:

```
GENERIC ( generic_name : TYPE := value )
```

eg.

```
ENTITY Chinkandi IS
    GENERIC ( prop_delay : TIME := 10 ns );
    PORT ( x, y : IN BIT ; z : OUT BIT );
END Chinkandi ;
```

(d) Process statement is the primary construct used for behavioral modelling. All VHDL statements written inside a process are sequential though the process statement by itself is a concurrent statement.

Syntax:

optional [Process label] : PROCESS (Sensitivity List)
Variable, signal declarations; (-local to the process)
BEGIN
Sequential statement 1;
Sequential statement 2;
.....
Sequential statement n;
END PROCESS [Process label];

(e) Solved

(f) Operator overloading is used to overload the existing built-in operators to be used with user-defined data types.

Syntax:

FUNCTION "operator symbol" (Class param1 : TYPE;
CLASS param 2 : TYPE) RETURN TYPE;
FUNCTION "operator symbol" (Class param1 : TYPE;
CLASS param 2 : TYPE) RETURN TYPE IS
Declarations local to function body

```
BEGIN
Sequential statements ;
RETURN output-val;
END " operator symbol ";
```

eg.

```
library ieee;
use ieee.std_logic_1164.all;
Entity op_overload is
port (x, y1 : in bit vector;
      z : out bit_vector);
end op_overload;
architecture op_overload of op_overload is
function "+" (a, b: bit_vector) return bit_vector is
 variable carry : bit := '0';
 variable sum : bit_vector;
 begin
 for i in range 0 to a'high - 1 loop
  sum(i) = a(i) xor b(i) xor carry;
carry := (a(i) and b(i)) or (a(i) and carry) or
     (carry and b(i));
 end for;
  return sum;
end "+";
begin
    z <= x1 + y1;
end op_overload;
```

Here addition operator '+' is overloaded to add two-bit vectors.

2) (a) Solved

(b) Operators in VHDL are of five types
- → Arithematic Operators
- → Logical Operators
- → Relational Operators
- → Shift and rotate operators
- → Miscellaneous operators

① Arithematic Operators

- Addition (+) and Subtraction (-)

  eg. c <= a+b;
  c <= a-b;

- Concatenation (&)

  eg. "0101" & '1'

- Multiplication (*)

  eg. c <= a * b;

- Division

  eg. c <= a / b;

- Remainder

  eg. A rem B = A - (A/B) * B;

- Modulus

  eg. A mod B = A - B * N

  where N is the smallest value for which the sign of the result is that of the second operand, that is B.

- Unary Operators

  eg. +a , -b

② Logical Operators : AND, OR, NOT, NAND, NOR, XOR, EXNOR

③ Relational Operators :

- Equality (=)

eg.
```
if a = b then
    c <= '0';
else
    c <= '1';
end if;
```

- Inequality (/=)

eg.
```
if a /= b then
    c <= '1';
else
    c <= '0';
end if;
```

- Smaller than (<)

eg.
```
if a < b then
    c <= '1';
end if;
```

- Smaller than or equal to (<=)

eg.
```
if a <= b then
    c <= '1';
end if;
```

- Greater than (>)

eg.
```
if a > b then
    c <= '1';
end if;
```

- Greater than or equal to (>=)

eg.
```
if a >= b then
    c <= '1';
end if;
```

④ Shift and Rotate Operators

- Shift left logic (SLL)

eg. a <= 101001

b ⇐ a SLL 2

- Shift right logic (SRL)

  a ⇐ 101001
  b ⇐ a SRL 2

- Shift left arithmetic (SLA)

  eg. a ⇐ 101001;
  b ⇐ a SLA 2

- Shift Right Arithmetic (SRA)

  eg. a ⇐ 101001
  b ⇐ a SLL 2

- Rotate Left (ROL)

  eg. a ⇐ 101001
  b ⇐ a ROL 2

- Rotate Right (ROR)

  eg. a ⇐ 101001
  b ⇐ a ROR 2

⑤ Miscellaneous Operators

- Absolute (abs)

  eg. abs (-2) = 2

- Exponentiation (**)

  eg. b ⇐ 3 ** 2 = 9

Ans 5) (b)

Concurrent Statements	Sequential Statements
→ Can appear outside of a process block	Can only appear inside of a process block
→ All statements inside an architecture block are concurrent statements	The statements are inside a process block
→ Simple signal assignment statement	Sequential Variable assignment statement
→ Process, Component instance, concurrent signal assignment	If, for, switch-case, signal assignment

Code 1

```
Entity ...
...
End ;
Architecture
Begin
Statement 1;
Statement 2;
Statement 3;
End ... ;
```

Code 2

```
Entity ...
...
End;
Architecture
Begin
Statement 1;
Statement 3;
Statement 2;
End ... ;
```

In sequential code, Code 1 and Code 2 are expected to give different results because difference in order of statement 2 and 3.
In Concurrent Coding, Code 1 and Code 2 are equivalent and are expected to give same result.

Ans 7) (a) State Machine - A state machine is a sequential circuit that advances through a number of states.

(b) State Diagram - A state diagram has following features :

→ Circles represent the states
→ Arrows between the circle represent rules for changing from state to state.
→ The information underneath the line in the circle represents the output value when in each state
→ Arrow coming from nowhere touches initial state.

(c) State Table - State table is a tabular method to represent finite state machines. State table consists of present state, next state and output.

(c) State Table - State table includes the following information
- State Name - A unique identifier for each state
- Input Conditions - The conditions that must be met for a transition to occur
- Next state - The state that system will transition to.

(d) State Assignment - It is a mapping from set of states of an FSM to the set of binary of codes with the objective of minimising the area of combinational circuit required to realise the FSM.

Ans) (a) ASM Chart

→ Algorithmic State Machine (ASM) charts provide less ambiguous description of a sequential system than state diagram.

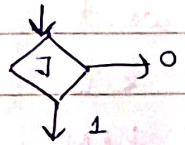
→ ASM chart resembles flow chart, but contain implicit timing information

→ They represent real hardware

→ All transitions within ASM charts must form closed paths (except a reset signal).

Components of ASM Chart

① Decision Box



→ Decision box has two or more branches going out

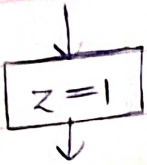
→ Decision is made based on value of one or more input signals (eg. signal J)

→ Decision box must follow and be associated with a state box.

→ Decision is made in the same clock cycle as the other actions of the state

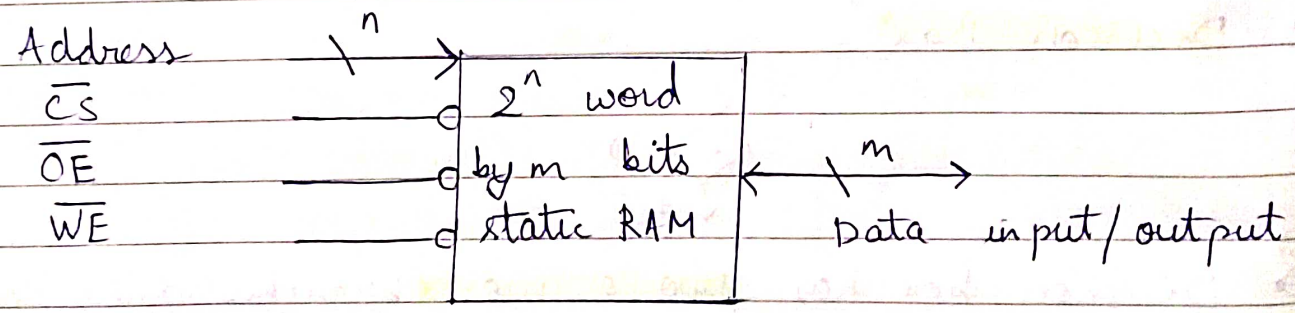
→ Input signal must be available and valid at the start of the clock cycle.

② Conditional Output Box



→ A conditional output box must follow a decision box
→ A conditional output box is attached to a state box through one or more decision boxes.
→ Output signals in conditional output box are asserted in the same clock cycle as those in the state box to which it is attached.
→ output signals can change during that state as a result of changes on the inputs.
→ Conditional output signals are sometimes referred as Mealy outputs since they depend on input signals as well

(b) Static RAM



Block Diagram of SRAM

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	I/O Pins
H	X	X	Not Selected	High-Z
L	H	H	Output Disabled	High-Z
L	L	H	Read	Data Out
L	X	L	Write	Data In

Truth table of SRAM

Block diagram shows a static RAM with n address lines, M data lines, and three control lines. This memory can store a total of 2^n words, each m bits wide. The data lines are bidirectional in order to reduce the number of pins and the package size of the memory chip.

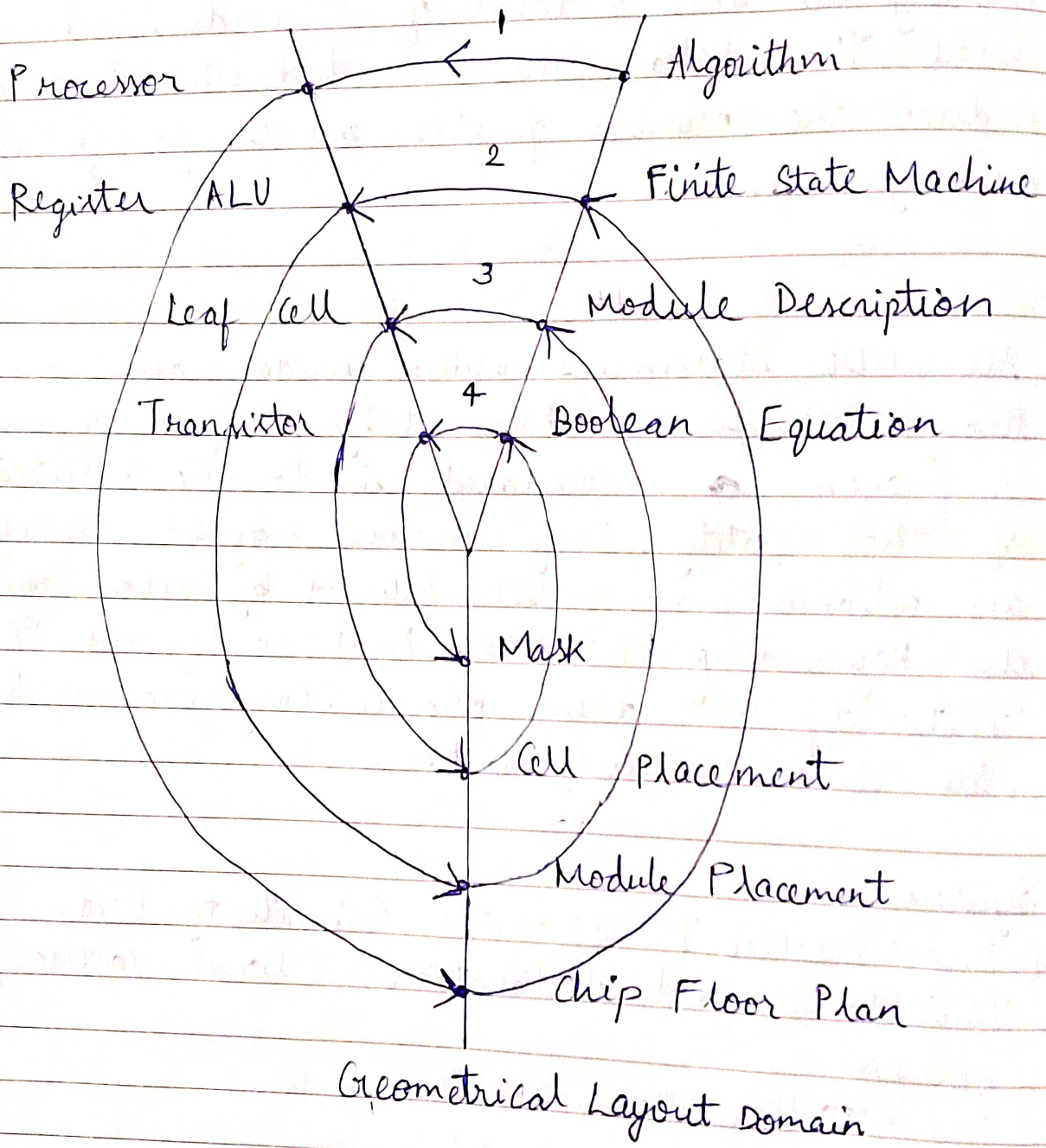
2017

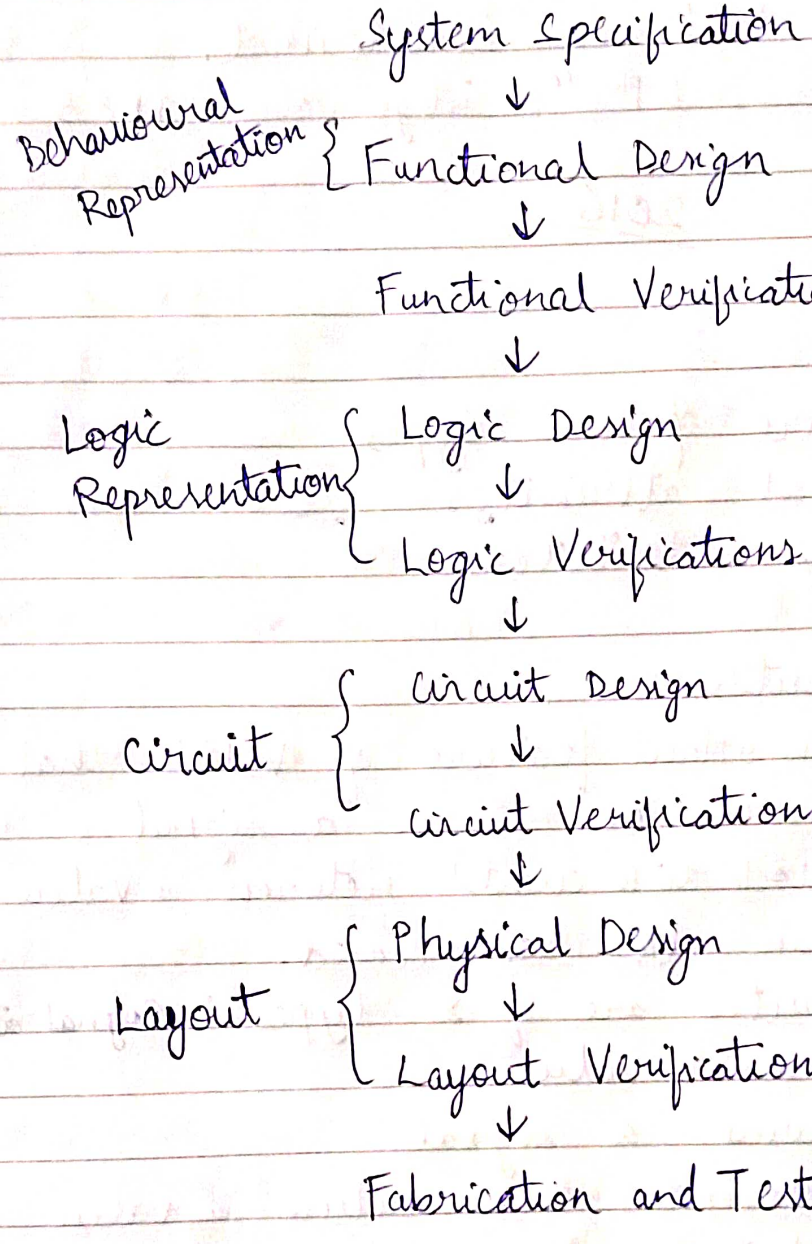
Any 1 (c) All VHDL statements written inside process are sequential though the process statement by itself is concurrent. A process is declared inside the architecture body of the entity. Every process may be identified with an optional process label, followed by colon and then by the keyword PROCESS. Next to keyword PROCESS is sensitivity list of the process. Every process has associated with it a sensitivity list.

Syntax.

```
[Process label]: PROCESS (Sensitivity List)
Variable, signal declarations; (local to the process)
 BEGIN
 Sequential statement 1;
 Sequential statement 2;
 ......
 Sequential statement n;
 END PROCESS [Process label];
```

Ans 3)(a) Gaski Y Chart:





FLOW CHART OF GAJSKI Y CHART

2016

Ans 1)(a) Based Literals : Literals are preceded with a base specification (such as 2, 8, or 16) and enclose the non decimal value with a pair of '#' characters

Given 16 # 23 DF # (integer value 9183)

Bit string literals : They are used to represent binary, octal or hexadecimal numeric data values. They are preceded by 'B' for binary, 'O' for

octal and 'X' for hexadecimal.
Given X" 23 DF " (integer value 9183)

2016

Ans1) (b) and (e)

Attributes are of two types
- Signal attributes
- Array attributes

Signal Attributes
- It is an important feature of VHDL that either returns a value or returns a signal
- clk'event (clock tick event) returns a value true if a change in clk has occured.
- Signal attributes are of 2 types:

→ Signal attributes that return a value
→ Signal attributes that return a signal

* Signal Attributes that returns a value

Attribute	Returns
S'ACTIVE	True if a transaction occured during the current delta, else false
S'EVENT	True if an event occured during the current delta, else false
S'LAST EVENT	Time elapsed since the previous event on S
S'LAST VALUE	Value of S before the previous event on S
S'LAST ACTIVE	Time elapsed since previous transaction on S

* Signal attribute that returns a signal

Attributes	Creates
S'DELAYED[(time)]*	Signal same as S delayed by specified time
S'STABLE[(time)]*	Boolean signal that is true if S had no events for the specified time.
S'QUIET[(time)]*	Boolean signal that is true if S had no transactions for the specified time
S'TRANSACTION	Signal of type bit that changes for every transaction on S.

Array Attributes

Attribute	Returns
A'LEFT(N)	left bound of Nth index range
A'RIGHT(N)	right bound of Nth index range
A'HIGH(N)	largest bound of Nth index range
A'LOW(N)	smallest bound of Nth index range
A'RANGE(N)	Nth index range

Ans 1)(h) Implicit Sequencing - In implicit sequencing concurrent statements are executed in parallel. This means that the order in which they appear does not affect their order of execution. Execution order of concurrent statements is determined by design's hardware architecture.

Explicit Sequencing - It uses sequential statements to specify the order of execution of statements.

Sequential statements are executed in a specific order, and the order in which they appear in the code is the order in which they are executed.

Ans 5(b) Field Programming Gate Array Logic (FPGA)

→ FPGA consists of configurable logic blocks, I/O blocks interconnected by routing channels.
→ They are much more versatile than CPLD
→ They do not contain AND-OR planes
→ They generally use SRAM
→ They use narrow logic resources and higher ratio of flip flops.
→ Logic capacity of FPGAs is much more higher than CPLDs
→ Logic density of FPGAs is higher than CPLDs.

2015

Ans 1)(c) Real time clock (RTC) is a computer clock, usually in form of an integrated circuit that keeps time. RTC must keep time even when the device is powered off.

Benefits of RTC

→ They are more precise than other methods
→ It frees the main system from time critical tasks.
→ It has low power consumptions and improved frequency stability.

Ans 5(b)

Resistor Array	RAM
→ Stores digital data by connecting or disconnecting a grid of resistors in a specific pattern.	Stores data in memory cells made up of transistors and capacitors.
→ Simple	Complex
→ Less expensive	More Expensive
→ Slow	Fast
→ Low Capacity	High Capacity