PQ Solutions

2017 ①

a i) The process in the ~~above~~ behavioural description uses the ~~Sequential~~ If statement

ii)
```
architecture Behavioural of JK_FlipFlop is
    -- signal declaration
    signal qtemp, qbartemp : std_logic := '0';
    begin
    Q <= qtemp
    Qbar <= qbartemp

  process (clk, reset)
  begin
  If (reset = '1') then
    qtemp <= '0';
    qbartemp <= '1';
  endif
  wait until rising_edge (clk); -- wait statement used here
  if (j = '0' and k = '0') then
    NULL;
  elseif (j = '0' and k = '1') then
    qtemp <= '0';
    qtempbar <= '1';
  elseif (j = '1' and k = '0') then
    qtemp <= '1';
    qtempbar <= '0';
  else
    qtemp <= not qtemp
    qtempbar <= not qtempbar
  endIf
  end process
end Behavioural
```

iii) The form of wait statement used is the 'wait until condition'. ~~It~~ This ~~provides~~ form of wait statement provides a condition ~~or conditions~~ which serve as a list of ~~the sensitivity~~ signals which will become the sensitivity channels of the process. Whenever an event occurs on a signal to which the process is sensitive, the process is reactivated.

iv) wait until condition

v) In the process statement, the NULL statement does nothing and ~~execution simply mo~~ the program simply moves to executing the next condition.

vi)
```
if condition then
    statements
endif
```

(b) The ~~Introearth~~ One of the major differences between procedures and functions is that the ~~major~~ invocation of a procedure is a statement while the invocation of of a function is an expression.

Also, functions are intended to be used strictly for computing values and not for changing the value of objects associated with the function formal parameters, therefore all parameters must be mode in and must be of class signal or constant. Procedures on the other hand are permitted to change the value of the objects associated with the procedure formal parameters meaning its parameters ~~can~~ may be of mode in, out and inout.

Another notable difference is that the function, unlike the procedure must declare a return type.

2017 $\boxed{2}$

a i ~~Signals are used to define +~~

Firstly, each data pathway has a type associated with it. The type defines a range of values which ~~can~~ may be passed over the pathway, that is, ~~i.e.~~ a data pathway whose type is integer cannot contain a value of type bit, or a real number or any type other than the integer

Secondly, all communications between processes takes place over these data ~~pathway~~ pathways. The data ~~[illegible]~~ pathways are directed, that is, one side of the pathway generates a value and another side receives the value.

(b) i

```
P1: Process
Variable B: Integer := 1;
 Begin
 Loop 1:
 for A in 1 to 12 loop
     B := 400
     Loop 2: loop
       ~~if (B < (A**2))~~
       next Loop 1 when B < (A**2)
       ~~end if~~
       B := B/A
     end loop Loop 2
 end loop Loop 1
 wait;
 end process;
```

Scanned with CamScanner

ii) next loop-label when condition

(c) In VHDL, the 'wait' statement works by activation and suspension. When a wait statement is executed inside a process, it suspends the process and the conditions for its reactivation are set. There are three different kinds of condition ~~tha~~ and they are : timeout, condition and signal sensitivity. These kinds of conditions can be mixed together in the wait statements or ~~used~~ the default may be used. They forms are shown below:

wait ; -- indefinite suspension
wait on signal-list;
wait until condition;
wait for time-expression; -- times out

2017 (3)

(a) variable_name := expression;

Variable assignment replaces the value of a variable object with new value obtained by evaluating the expression on the ~~left~~ right hand side of the assignment

2017

3ai i) Component
ii) Port
iii) Signal

i) Component is the ~~can be seen~~ building block of hardware description. All gates, chips and boards can be seen as components

ii) Port is the component's point of connection to the world, the point through which data flows in and out of the component

iii) A signal is a path from one component to another component, the path along which data flows between components.

(b) i

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

        ~~(to be cont)~~

~~architecture~~ entity 4b_full_adder is
  PORT (A, B : IN BIT_VECTOR (3 downto 0);
           CIN : IN BIT; COUT OUT bit;
           S : OUT bit_Vector (3 downto 0));
end 4b_full_adder

architecture Behavioural of 4b_full_adder is
signal c: bit_vector (3 downto 0);
begin
    process (A, B, CIN)
        begin
          ~~for i in 0 to 3 loop~~
```

Scanned with CamScanner

```
        if A(i) = '0' and B(i) = '0' then C(i) = '1';
            if CIN = '1' THEN S(i) <= '1';
            else S(i) <= '0';
            end if
        elseif ((A(0) = '0' AND B(0) = '1' and CIN = '0')
    OR (B(0) = '0' and A(0) = 1 and CIN = '0'))
          Then S(0) <= '1'; C(1) <= '0'
        elseif ((A(0) = '0' and B(0) = '1' and CIN = '1')
    OR (A(0) = '1' and B(0) = '0' and CIN = '1'))
          then S(0) <= '0'; C(1) <= '1'
        elseif CIN = '0' then C(1) <= '1'; S(0) = '0';
            else C(1) <= '1'; S(0) <= '1';
        end if
                    : Repeat if block for FA 1, 2 and 3
    end process
end Behavioural
```

⑤ ii

```
library ieee;
use ieee.std_logic_1164.all
entity Ab_full_adder is
    port ( A, B: in bit_vector (3 downto 0);  -- positional association
        SEL: in bit
```

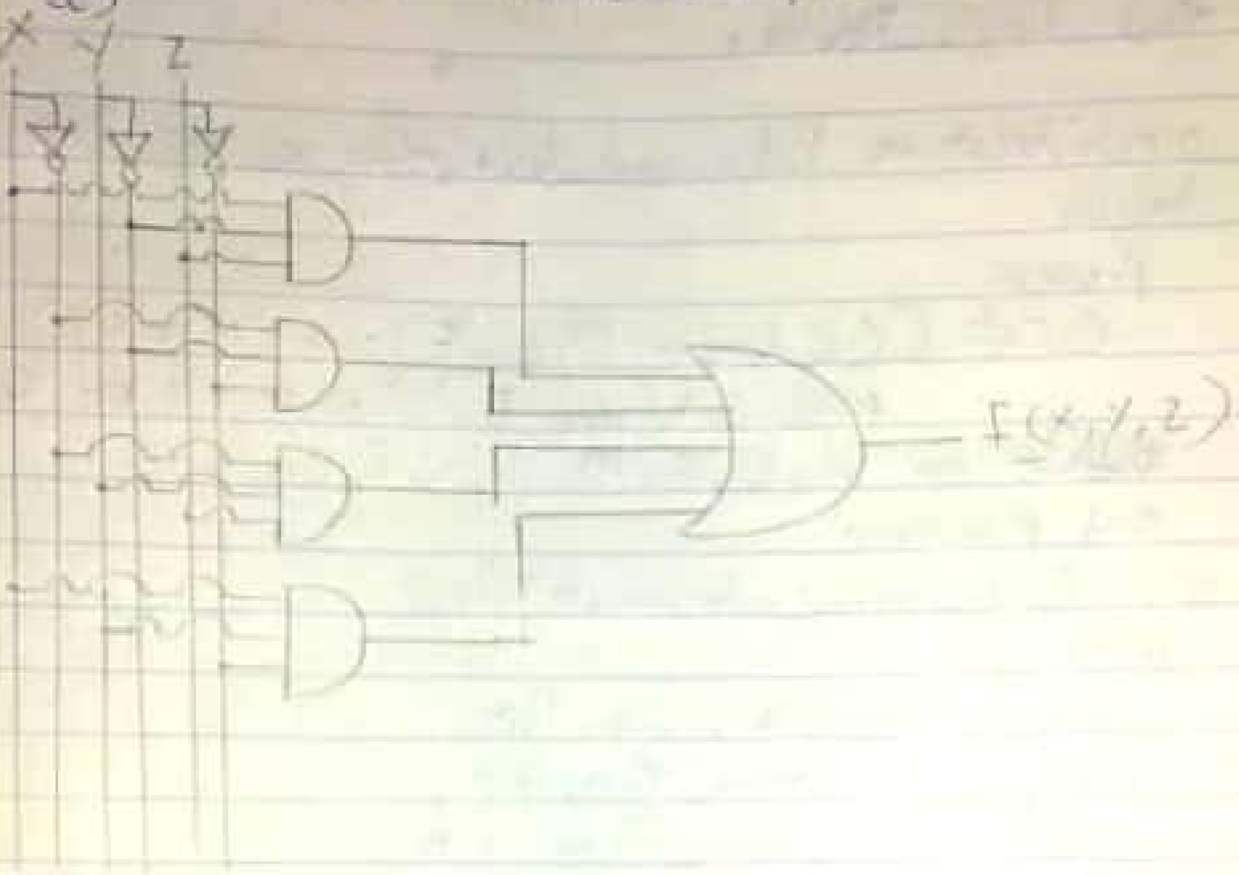
Scanned with CamScanner

```
    COUT : out bit;
      X : out bit_vector(3 downto 0)); -- positional association
end 4b-full-adder
architecture structural of 4b_full_adder is
   component fa_1b -- 1 bit full adder component
     port (~~X~~, Y, CIN : IN bit;
          COUT : OUT bit);
   end component
   signal C : bit-vector (3 downto 0) -- positional association
   begin
   FA0 : fa_1b PORT MAP (A(0), B(0), CIN, S(0), C(1));
   FA1 : fa_1b PORT MAP (A(1), B(1), C(1), S(1), C(2));
   FA2 : fa_1b PORT MAP (A(2), B(2), C(2), S(2), C(3));
   FA3 : fa_1b PORT MAP (A(3), B(3), C(3), S(3), COUT);
end structural
```

2016/2017 Question 4 a

$f(x, y, z) = x\bar{y}z + \bar{x}\bar{y}\bar{z} + \bar{x}yz + xy\bar{z}$

(i)



(ii) Entity Declaration for logic circuit Above

```
entity Logic_Circuit is
  port(X : in Bit; Y: in Bit; Z: in Bit; Out : out Bit);
end Logic_Circuit;
```

(iii)

```
architecture Behavioral_description of Logic_Circuit is
 begin
  Process
    Out <= (X and (not Y) and Z) or ((not X) and (not Y)
           and (not Z)) or ((not X) and Y and Z)
             or (X and Y and (not Z));
   Wait on X, Y, Z;
  end process;
end Behavioral_description;
```

17/2018 Question 1

) $D_i B_{out} = A - B - B_{in}$

A	B	B_{in}	D_i	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

D_i

B_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

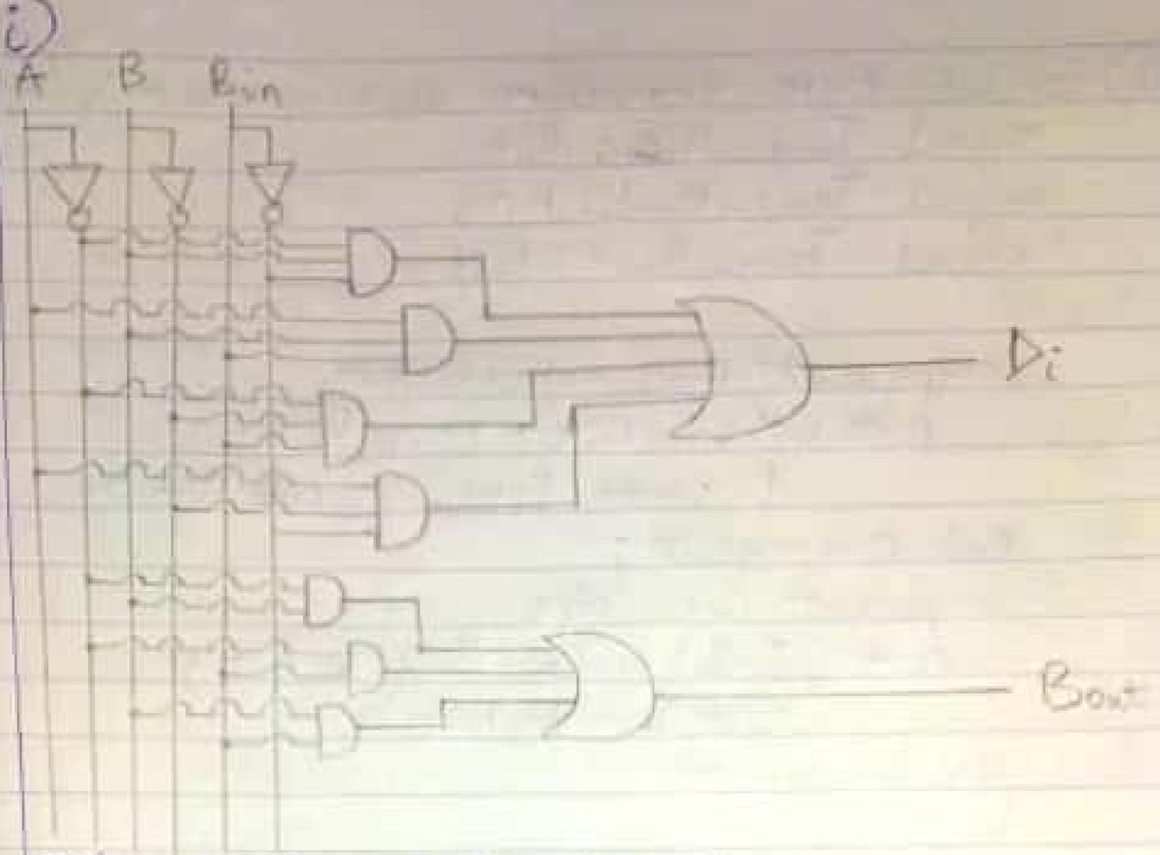
B_{out}

$D_i = \bar{A} B \bar{B}_{in} + A B B_{in} + \bar{A}\bar{B} B_{in} + A \bar{B} \bar{B}_{in}$

$B_{out} = \bar{A} B + \bar{A} \bar{B}_{in} + B B_{in}$

Scanned with CamScanner

i)



ii)

```
entity Subtractor is
  port (A: in Bit; B: in Bit; B_in : in Bit;
        Di : out Bit; Bout : out Bit);
end Subtractor;
```

iii)

```
architecture Behavior of Subtractor is
  begin
    Process
      D_i <= ((not A) and B and (not B_in))
          or (A and B and B_in)
          or ((not A) and (not B) and B_in)
          or (A and (not B) and (not B_in))
      B_out <= ((not A) and B)
            or ((not A) and B_in)
            or (B and B_in);
    Wait on A, B, B_in;
   end process;
 end Behavior;
```

```
iv) architecture Structure of subtractor is
      signal Temp_diff : Bit;
      signal Temp-B_1 : Bit;
      signal Tem_B_2 : Bit;

      component Half_subtractor
          port (X : in Bit; Y : in Bit;
                D : out Bit; B : out Bit);
      end component;
      component Or_gate
          port (In 1 : in Bit; In 2 : in Bit;
                Out 1 : out Bit);
      end component;
      begin
         U0 : Half_subtractor
              port map (
                 X => A, Y => B,
                 D => Temp_diff, B => Temp_B 1);
         U1 : Half_subtractor
              port map (
                 X => Temp_diff, Y => Bin,
                 D => D_i, B => Temp_B 2);
         U2 : Or_gate
              port map (
                 In 1 => Temp_B 1, In 2 => Temp_B_2
                 Out 1 => B_out );
      end structure;

b) entity Logic_Device is
      port (D : in array (11 downto 0) of Bit;
            OE : in Bit; CLK : in Bit;
            AD : inout array (11 downto 0) of Bit;
            A : out array (11 downto 0) of Bit;
            INT : out ('1', '0', 'Z');
            AS : out Bit);
   end Logic Device;
```

Scanned with CamScanner

c) · object-class identifier-list: subtype-indication
signal-kind := expression

The general form for object declaration is given above. The object class is either "constant", "variable", or "signal". The identifier list is one identifier or multiple identifiers separated by commas (,). The subtype indication specifies the type of the objects declared by the object declaration; it may name a type with an additional constraint or just a type. The signal kind is either bus or register; the signal kind is optional and may only appear in a signal declaration.

CPE522

1. CHARACTERISTICS OF PROCESS STATEMENTS (PD 2018, 3a)

(i) They define specific behaviour to be executed when a process (an operation) becomes active.

(ii) It consists of a "label", "declarations" and "statements" with the label being optional. The declarations section defines the local data environment needed by the process while the statements section is the program that define the behaviour of the process.

(iii) Process statements always have a beginning and an end between which the "statements" are written. As mentioned above, the "statements" are the programs to be executed that defines the behaviour of the process.

(b) Considering the process statement,

```
P1:   PROCESS
      Begin
          S <= transport 1 after 1ns
```

```
        S <= transport 1 after 1ns
        S <= transport 1 after 1ns, 2 after 3ns, 4 after 4ns,
                7 after 5ns;
        S <= transport 2 after 2ns;
        S <= transport 3 after 5ns, 4 after 4ns, 4 after 5ns, 6 after 10ns;
       wait;
    End PROCESS;
```

4) The above process statement adds transactions ([illegible]) to the driver of signal S. The first statement adds just one transaction to the driver (1, 1ns). The second statement overwrites the ~~first statement~~ transaction of the previous assignment because ~~of the same~~ the transaction (1, 1ns) of the second statement is scheduled the same time as the transaction of the previous assignment. The third statement ~~being~~ adding the transaction (2, 2ns) to the driver of signal S overwrites all last three transactions of the previous assignment as

it is scheduled before them all namely (2, 3ns), (4, 4ns) and (7, 5ns). In the last assignment, the second transaction overrides ~~[illegible]~~ the first transaction, so what is added to the driver of signal S becomes (4, 4ns), (4, 5ns) and (6, 10ns). Altogether, the following integers are added to the ~~[illegible]~~ driver of signal S, thus constituting the effect on signal S.

(ii) S —

(1, 1ns)	(2, 2ns)	(4, 4ns)	(4, 5ns)	(6, 10ns)

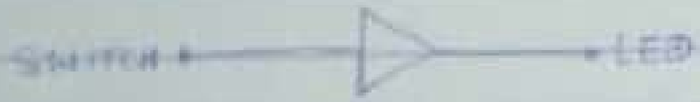
1(a)

VHDL PROGRAM ~~[illegible]~~ CONTROLLING A LED USING A SWITCH

(i) When the switch is toggled to 1, the LED is turned ON but when toggled to zero (0), it is turned OFF. This clearly indicates that the device is a buffer having just a single input and a single output, with the signal output the same as ~~[illegible]~~ the value of the logic input. It is ~~indicated~~ represented by the truth table below:

SWITCH	LED
0	OFF (0)
1	ON (1)

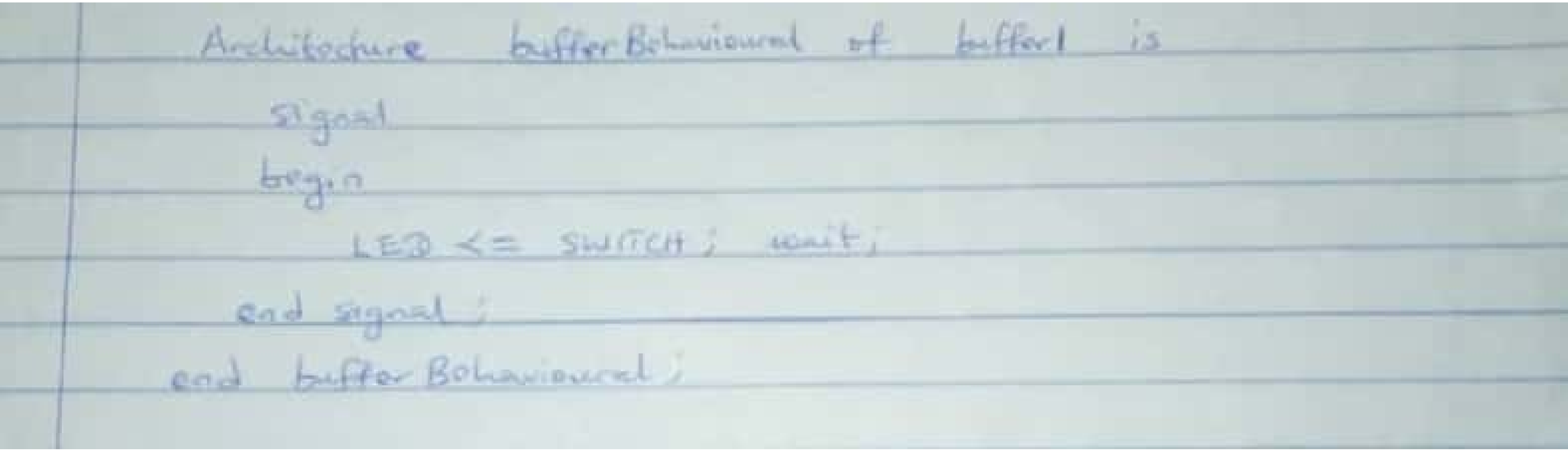
where SWITCH is input, LED is output

(ii) 

SKETCH OF THE DEVICE TO CONTROL LED

(iii) ARCHITECTURE DESCRIPTION OF DEVICE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity buffer1 is
    Port ( SWITCH : in STD_LOGIC;
           LED : out STD_LOGIC);
end [illegible];
```



```
Architecture  bufferBehavioural  of  buffer1  is
    signal
    begin
        LED <= SWITCH ;  wait;
   end signal ;
end  bufferBehavioural ;
```

4(a) CONSIDERING PROCESS STATEMENT OF Qn4 (2019, NO. 4)

(I) Process Statement Using the While Loop

~~P1: Process~~
~~Variable B: Integer :=1; variable A: Integer :=1;~~
~~Begin~~
~~B A:=1~~

```
P1: Process
    Variable B: Integer :=1;
    Variable A: Integer range 1 to 12;
    begin
        A:=1;
Loop1: while (A ≤ 12) loop
            B:=400;
            Loop 2: loop
                if B < (A**2) then
                    Exit loop2;
                end if;
                B:= B/A;
            End loop Loop2;  A:= A+1;
    End loop Loop1;
    wait;
End Process
```

(ii) SYNTAX OF THE WHILE STATEMENT

```
while condition loop
    sequential statements
end loop;
```

(iii) General form of the Process Statement Used

```
Process label:
    process
        declaration
    begin
        statements
    end process.
```

4(b)

* FEATURES THAT DIFFERENTIATE FUNCTIONS AND PROCEDURES

(i) Functions are intended to be used strictly for computing value, not for changing the value of any objects associated with the function formal parameters. On the other hand, procedures are permitted to change the values of the object associated with the procedure formal parameter

(ii) All parameters of functions must be of mode in and must be of class signal or constant. ~~[illegible]~~ Parameters of procedure however may be in mode in, out and inout.

4(c)

GENERAL FORM OF A VHDL FUNCTION

```
function functionname (paramname : paramtype := v
              return returntype is
          declaration
    begin
          statement
          return value;
    end function;
```

NOTE => A function can have as many parameters as well as no parameters.

4(c)(i)

VHDL FUNCTION DESCRIPTION FOR A 3-INPUT XOR FUNC-TION

```
        function   xOR_logic_gate (a, b, c : Bit) return Bit is
              variable  xORvalue: Bit := 0;
        begin
              xORvalue := a xor b xor c ;
              return xORvalue;
        end xOR_logic_gate;
```

4(c)(i)

VHDL FUNCTION DESCRIPTION FOR A 3 INPUT XOR FUNC-
TION

```
function xor_logic_gate (a, b, c : Bit) return Bit is
    variable xorvalue : Bit := 0;
begin
    xorvalue := a xor b xor c;
    return xorvalue;
end xor_logic_gate;
```

2018/2019 Question 1

a i)

```
entity Logic_Device is
  port ( W: in Bit; O: in Bit; E: in Bit;
         M: in Bit; S: in Bit; V: in Bit;
         F: in Bit; C: in Bit; OUT: out Bit);
end Logic_Device;

architecture Behavioral_description of Logic_Device is
begin
  Process
    OUT <= ((((W or O) and E) and (M or S))
            and (V and F)) and (M or C);
  wait on W, O, E, M, S, V, F, C;
  end process;
end Behavioral_description;
```

```
  Process
(ii)  signal Location_One: Bit;
      signal Location_Two: Bit;
      signal Location_Three: Bit;
      signal Location_Four: Bit;
      signal Location_Five: Bit
      locatio
  begin
      Location_One <= OUT
      Location_Two <= OUT
      Location_Three <= OUT
      Location_Four <= OUT
      Location_Five <= OUT
      wait on OUT;
  end process;
```

5(a) ~~package Logic is~~
~~constant true : Bit~~
~~constant False : Bit~~
~~end Logic~~

```
    entity WALLOP is
b(i)  Port (A : in Bit; B: in Bit; OUT : out Bit);
      signal true : Bit bus := '1'
      signal false : Bit bus := '0'
    end WALLOP;
(ii)
```

```
c)  entity WIDGET is
      port( A: in Bit, B: in Bit; OUT_1: out Bit;
              OUT_2: out Bit);
    end WIDGET;
```

2) ~~SULE — PETER — JAMES SULE —~~ (i)

SULE	PETER	JAMES	$\overline{\text{SULE}}$	$\overline{\text{JAMES}}$	P1	Q1	OKAY
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	1	1	1	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	1	0	1	1
1	1	1	0	0	0	0	0

(ii)

JAMES \ SULE PETER	00	01	11	10
0		1	1	
1		1		

OKAY = PETER · $\overline{\text{JAMES}}$
+ $\overline{\text{SULE}}$ · PETER

iii)
```
entity Logic_Circuit is
   port ( SULE : in Bit; PETER : in Bit;
              JAMES : in Bit; OKAY : out Bit);
end Logic_Circuit;

architecture Behavior of Logic_Circuit is
   begin
      process
         OKAY <= ((not SULE) and PETER)
                    or ( PETER and (not JAMES))
      end process;
end Behavior;
```

18/19

No. 3

3i. **Schematic circuit capture**

Schematic capture is the process of creating a schematic diagram for an electronic circuit using various tools designed for the job. This can be done from as simple as using a pen and paper to using schematic capture software, including highly expensive electronic design automation suites or packages that can do everything from schematic capture, layout and simulation.

Schematic capture is part of circuit analysis and design; it is the process of "taking" the schematic design from an engineer's head and entering it into a computer or putting it into a piece of paper. In simpler terms, it means that an engineer is designing a circuit that will serve a specific purpose by making use of industry standards and conventions to put the design into a visual state, either via hand drawing or by entering it into a software made for the purpose.

3ii. **Design Process**

1. Write a design description in the VHDL language. This description can be a combination of structural and functional elements . This description is used with both the Synopsys VHDL Compiler and your VHDL simulator.
2. Provide VHDL-language test drivers for your VHDL simulator. These drivers supply test vectors for the simulation and gather output data.
3. Simulate the design by using your VHDL simulator to verify the accuracy of the description.
4. Synthesize the VHDL description with VHDL Compiler. VHDL Compiler performs architectural optimizations, then creates an internal representation of the design.
5. Use the Synopsys Design Compiler to produce an optimized gate-level description in the target ASIC library. You can optimize the generated circuits to meet the timing and area constraints you want. This optimization step must follow the translation step (step 4) to produce an efficient design.
6. Use the Synopsys Design Compiler to output a gate-level VHDL description. This netlist-style description uses ASIC components as the leaf-level cells of the design. The gate-level description has the same port and module definitions as the original high-level VHDL description.
7. Pass the gate-level VHDL description from step 6 through your VHDL simulator. You can use the VHDL simulation drivers from Step 2 because module and port definitions are preserved through the translation and optimization processes.
8. Compare the output of the gate-level simulation (step 7) against the output of the original VHDL description simulation (step 3) to verify that the implementation is correct.

3iii Structural design decomposition

Hierarchical decomposition or partitioning splits the system specification into simpler sub-systems. These units are to be defined according to the corresponding degree of re-utilization, and those that may share the same operators may be regrouped.

When the hierarchical decomposition is done with respect to the regularity of the system design, the whole synthesis is simplified. Regularity implies that sub-systems or specific designs will be re-used more than once and therefore the total amount of designs needed will be reduced. As a result, regularity allows an improvement in productivity, in general.

3iv **Design Levels**

Behavioral Level

This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other. Functions, Tasks and Always blocks are the main elements. There is no regard to the structural realization of the design.

Register-Transfer Level

Design using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times. Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

Gate Level

A gate level description consists of a network of gates and registers instanced from a technology library, which contains technology-specific delay information for each gate.

Within the logic level the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates). *Using gate level modeling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.*

Question4i

VHDL is used for the following purposes:

For Describing hardware

As a modeling language

For a simulation of hardware

For early performance estimation of system architecture

For the synthesis of hardware

A list of advantages of VHDL is given below:

It supports various design methodologies like Top-down approach and Bottom-up approach.

It provides a flexible design language.

It allows better design management.

It allows detailed implementations.

It supports a multi-level abstraction.

It provides tight coupling to lower levels of design.

It supports all CAD tools.

It strongly supports code reusability and code sharing.

Question4ii

Architecture body

The architecture body specifies how the circuit operates and how it is implemented. As discussed earlier, an entity or circuit can be specified in a variety of ways, such as behavioral, structural (interconnected components), or a combination of the above.

The architecture body looks as follows,

```
      architecture architecture_name of NAME_OF_ENTITY is
    -- Declarations
      -- components declarations
       -- signal declarations
       -- constant declarations
       -- function declarations
       -- procedure declarations
       -- type declarations
:
    begin
    -- Statements
:
    end architecture_name;
```

The header line of the architecture body defines the architecture name, e.g. behavioral, and associates it with the entity. The architecture name can be any legal identifier. The main body of the architecture starts with the keyword begin and gives the Boolean expression of the function. The "<= " symbol represents an assignment operator and assigns the value of the expression on the right to the signal on the left. The architecture body ends with an end keyword followed by the architecture name.

Question 4iii

```
-- Structural modeling of 4:1 mux
library ieee;
use ieee.std_logic_1164.all;
entity MUX4_1 is
```

```
port ( Sel0,Sel1 : in std_logic; A, B, C, D : in std_logic; Y : out std_logic );
end MUX4_1;

architecture structural of MUX4_1 is
component inv
port (pin : in std_logic; pout :out std_logic);
end component;
component and3
port (a0,a1,a2: in std_logic; aout:out std_logic);
end component;
component or4
port (r0,r1,r2,r3:in std_logic; rout:out std_logic);
end component;

signal selbar0,selbar1,t1,t2,t3,t4: std_logic;
begin
INV0: inv port map (Sel0, selbar1);
INV1: inv port map (Sel1, selbar1);
A1: and3 port map (A, selbar0, selbar1, t1);
A2: and3 port map (B, Sel0, selbar1, t2);
A3: and3 port map (C, selbar0, Sel1, t2);
A4: and3 port map (D, Sel0, Sel1, t4);
O1: or4 port map (t1, t2, t3, t4, Y);
end structural;
```