VHDL

VHDL is a hardware description language. It describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be attained (implemented). VHDL stands for **VHSIC H**ardware **D**escription **L**anguage. VHSIC is itself an abbreviation for **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits.

The two main immediate applications of VHDL are in the field of Programmable Logic Devices (including CPLD – Complex Programmable Logic Devices and FPGAs – Field Programmable Gate Arrays) and in the field of ASICs (Application Specific Integrated Circuits). Once the VHDL code has been written, it can be used either to implement the circuit in a programmable device (from Altera, Xilinx, Atmel, etc.) or can be submitted to a foundry for fabrication of an ASIC chip. Currently, many complex commercial chips (microcontrollers for example) are designed using such an approach.

Contrary to regular computer programs which are sequential, VHDL statements are inherently concurrent (parallel). For that reason, VHDL is usually referred to as a code rather than a program. In VHDL, only statements placed inside a process, function or procedure are executed sequentially.

Modeling a Component in VHDL:

To model a component in VHDL, the following are required:

- Import the appropriate packages (libraries), e.g. std_logic_1164.
- The **entity** corresponds to the interface of a component.
- The **architecture** describes its behavior.

Entity and Architecture:

An entity definition which is used to describe the external interface for the logic device, i.e. inputs and outputs to the device.

The general form of the entity description is:

```
Entity component_name is
     Input and outputs.
     Physical and other parameters.
End component_name;
```

An architecture definition which is used to describe the functionality of the digital component.

The general form of the architecture description is:

```
Architecture identifier of component_name is
    Declarations
Begin
    Functionality of the component in terms of input
    lines and other parameters
End identifier;
```

Each entity declaration must have port definition to define the input and output of the components, the form of the port statements is as follows:

```
Port (interface_name: mode data_type; … );
```

Where:

interface_name: the name of the interface

mode: the mode of the interface, the modes available are

in: the interface is input

out: the interface is output

inout: the interface is bidirectional

buffer: like the inout

data_type: the type of interface like, bit, bit_vector, integer and more.

Types of VHDL modeling:

There are three types of VHDL models used to write VHDL statements

1. Data flow
2. Structural
3. Behavioral

1 – Data flow modeling:

In this type of modeling the component that need to be modeled is written from its logic equation. For example the following code is the data flow model of an and gate.

```
entity and_2 is
    port (a, b: in bit; c: out bit);
end and_2;
architecture data_flow of and_2 is
begin
    x1: c <= a and b;
end data_flow;
```

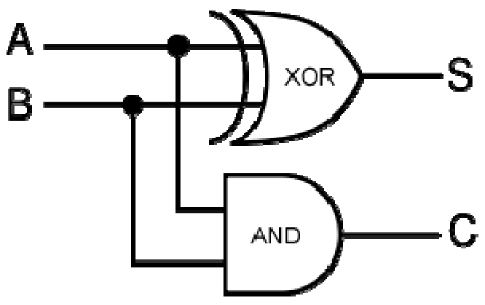
All logic gate can be modeled using this approach by using the following logical operation

and, or, not, nand or xor

Example: Use a VHDL to implement the operation of a half adder.

Solution: The truth table of a H.A. is

A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1



$$S = A\underline{B} + \underline{A}B = A\oplus B$$

$$C = AB$$

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
    port (A, B: in std_logic; S, C: out std_logic);
end half_adder;
```

```
architecture data_flow of half_adder is
begin
    S <= A xor B;
    C <= A and B;
end data_flow;
```

The Delay:

For the purpose of the simulation the designed component will be delayed for a period amount of time to simulate a real component. The form of a delay is:

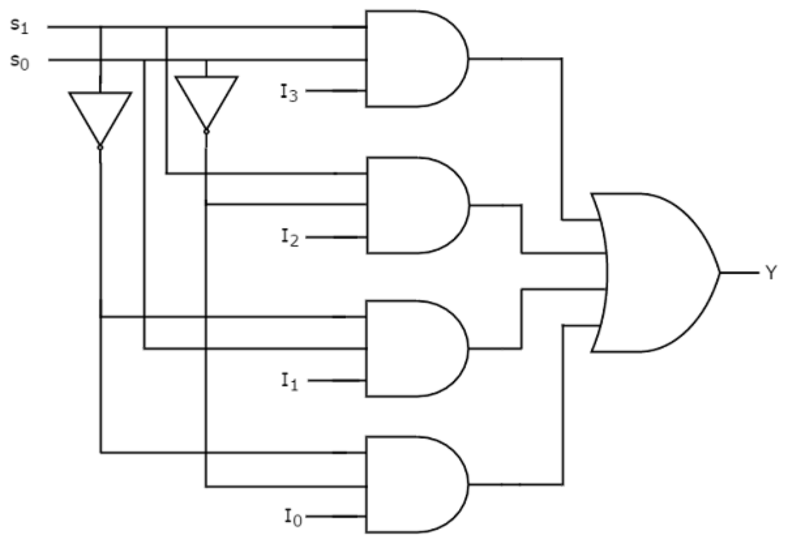
`after` `xx` **`ys`** **,** where xx is any number and y is n, p, u, …

The following example is an inverter gate with a delay of 15 nanosecond.

```
library ieee;
use ieee.std_logic_1164.all;
entity not_1 is
    port (a: in std_logic; c: out std_logic);
end not_1;
architecture data_flow of not_1 is
begin
    x1: c <= not a after 15 ns;
end data_flow;
```

Example: Write a VHDL program to implement a 4-to-1 multiplexer by using data flow model

Solution: The circuit diagram of the required multiplexer is shown below



To write a VHDL data flow model to any circuit, it is required to get the Boolean equation for any gate available in the circuit diagram which means there is a need to know the circuit diagram or the truth table of the circuit.

```
library ieee;
use ieee.std_logic_1164.all;
entity MUX_4_1 is
    port (I: in std_logic_vector (3 downto 0); s: in
    std_logic_vector (1 downto 0); Y: out std_logic);
end MUX_4_1;
architecture data_flow of MUX_4_1 is
    signal m: std_logic_vector (5 downto 0);
begin
    x1: m(0) <= not s(0);
    x2: m(1) <= not s(1);
    x3: m(2) <= m(0) and m(1) and I(0);
    x4: m(3) <= s(0) and m(1) and I(1);
    x5: m(4) <= m(0) and s(1) and I(2);
    x6: m(5) <= s(0) and s(1) and I(3);
    x7: Y <= m(2) or m(3) or m(4) or m(5);
    end data_flow;
```

Concurrent Signal Assignment:

There are three types of concurrent signal assignment

1. Signal Assignment
2. Conditional Signal Assignment
3. Selected Signal Assignment

1 – Signal Assignment:

c <= a and b after 10 ns;

2 – Conditional Signal Assignment:

This type of assignment used when there is a need to execute a condition inside the program. The form of the conditional assignment is shown below:

```
    signal_name <= expression_1 when condition_1 else
                   expression_2 when condition_2 else
                   expression_3;
```

For example:

```
    C <= a when s = "00" else
         b when s = "01" else
         d when s = "10" else
         f;
```

3 – Selected Signal Assignment

This type of assignment used when there is multi value to select among them according to a condition inside the program. The form of the conditional assignment is shown below:

```
with expression select
    signal_name <= expression_1 when choice_1,
                   expression_2 when choice_2,
                   expression_3 when choice_3,
                   expression_n when others;
```

For example:

```
with s select
    c <= a when “00”,
      <= b when “01”,
      <= d when “10”,
      <= f when others;
```

Example: Write a VHDL program to implement an 8-to-1 multiplexer by using 1) conditional signal assignment 2) selected signal assignment.

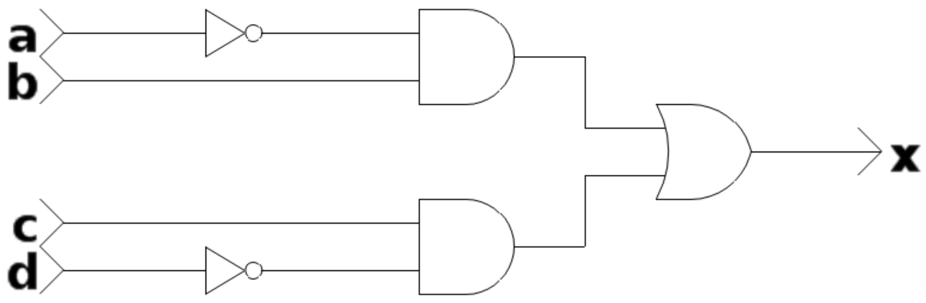
Solution:

```
    library ieee;
    use ieee.std_logic_1164.all;
    entity MUX_8_1 is
        port (I: in std_logic_vector (7 downto 0); s: in
        std_logic_vector (2 downto 0); o: out std_logic);
    end MUX_8_1;
    architecture data_flow of MUX_8_1 is
    begin
1)  o <= I(0) when s = “000” else
      <= I(1) when s = “001” else
      <= I(2) when s = “010” else
      <= I(3) when s = “011” else
      <= I(4) when s = “100” else
```

```
        <= I(5) when s = "101" else
        <= I(6) when s = "110" else
        <= I(7);
2)   with s select
          o <= I(0) when "000",
               I(1) when "001",
               I(2) when "010",
               I(3) when "011",
               I(4) when "100",
               I(5) when "101",
               I(6) when "110",
               I(7) when others;
    end data_flow;
```

2 – Structural Modeling:

In a structural model, the required circuit is implemented from its connected gates, by using by using a keyword called **port map**, to implement the following circuit in VHDL:



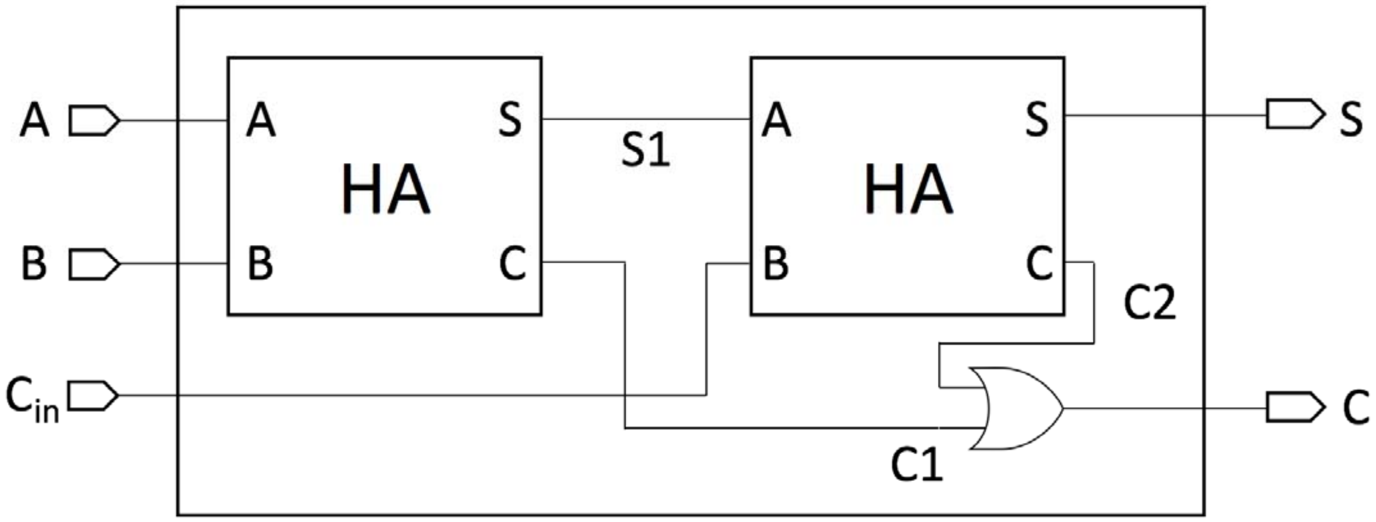
Before writing the VHDL, look at the labels s0, s1, s2 and s3, those labels are added to the circuit because it is impossible to put output port in a place where an input port is required. To prevent this problem, add to the circuit the signals that can behave as input and output at the same time.

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

```
entity cct is
    port (a, b, c, d: in std_logic; x: out std_logic);
end cct;
architecture struct of cct is
    component not_1
        port (a: in std_logic; c: out std_logic);
    end component;
    component and_2
        port (a, b: in std_logic; c: out std_logic);
    end component;
    component or_2
        port (a, b: in std_logic; c: out std_logic);
    end component;
    signal s0, s1, s2, s3: std_logic;
begin
x1: not_1 port map (a, s0);
x2: not_1 port map (d, s1);
x3: and_2 port map (s0, b, s2);
x4: and_2 port map (c, s1, s3);
x5: or_2 port map (s2, s3, x);
end struct;
```

Example: Assume there are a half adder already defined in the library, use it to implement a full adder using structural model.

Solution: The circuit diagram of the required full adder is shown below.



```
library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
    port (A, B, Cin: in std_logic; S, C: out std_logic);
end full_adder;



architecture struct of full_adder is
    component half_adder
        port (A, B: in std_logic; S, C: out std_logic);
    end component;
    component or_2
        port (a, b: in std_logic; c: out std_logic);
    end component;
    signal S1, C1, C2: std_logic;
begin
k1: half_adder port map (A, B, S1, C1);
k2: half_adder port map (S1, Cin, S, C2);
```

```
    k3: or_2 port map (C1, C2, C);
    end struct;
```

3 – Behavioral Modeling

In this type of modeling the required designed circuit is described by its behavioral not by its internal connected component. The main component of a VHDL behavioral model is a keyword called process, the general form of the process is:

```
X: process (a, b, c, …, z)
```

Where: X is a label for different processes

a, b, c, …, z is a sensitivity list.

The sensitivity list used to recall the process if any one of their variable change its value. If the sensitivity list ignored then there must be at least one wait statement written inside the process code.

```
    process
    begin
        wait;
        wait for 10 ns;
        wait until A = '1';
    end process;
```

In the behavioral model, the following can be used:

1 – Conditional Statements

```
    A - if condition then
             statements;
        end if;


    B - if condition then
             statements;
        elsif condition then
             statements;
        else
```

```
            statements;
        end if;
    C - case condition is
            When val1 =>
                Statements;
            When val2 =>
                Statements;
            .
            .
            .
            When others =>
                Statements;
        End case;
```

2 – Looping Statements

```
    A - for var in initial to final [step increment] loop
            Statements;
        end loop;

    B - do
            Statements;
        While condition;
```

Example: Write a VHDL program to implement 3 – to – 8 decoder using behavioral modeling.

Solution:

```
library ieee;
use ieee.std_logic_1164.all;
entity decoder_3_to_8 is
    port( I: in std_logic_vector(2 downto 0);
          O: out std_logic_vector(7 downto 0));
end decoder_3_to_8;
architecture behave of decoder_3_to_8 is
begin
    process (I)
    begin
        case I is
            when "000" =>
                O <= "00000001";
            when "001" =>
                O <= "00000010";
            when "010" =>
                O <= "00000100";
            when "011" =>
                O <= "00001000";
            when "100" =>
                O <= "00010000";
            when "101" =>
                O <= "00100000";
            when "110" =>
                O <= "01000000";
            when others =>
                O <= "10000000";
        end case;
        end process;
    end behave;
```

Example: Write a VHDL program to implement a D – flip flop.

Solution:

```
    library ieee;
    use ieee.std_logic_1164.all;
    entity D_FF is
        port (D, CK: in std_logic; Q, Qbar: out std_logic);
    end D_FF;
    architecture behave of D_FF is
    begin
        process (D, CK)
            variable temp: std_logic;
        begin
            if CK = '1' and CK'event then
                temp := D;
            end if;
            Q <= temp;
            Qbar <= not temp;
        end process;
    end behave;
```

Notes:

- Variable is like signal but with one difference, the signal take the new value after the end of all statements while the variable take it immediately.
- := called Variable Assignment Statement.

Example: Write a VHDL program to implement a JK flip-flop with asynchronous reset using behavioral model.

Solution:

```
library ieee;
use ieee.std_logic_1164.all;
entity JK_FF is
    port (J, K, CK, RES: in std_logic; Q, Qbar: out
    std_logic);
end JK_FF;
architecture behave of JK_FF is
begin
    process (RES, CK, J, K)
        variable JK: std_logic_vector (0 to 1);
        variable temp: std_logic;
    begin
        JK := J&K;
        if RES = ‘0’ then
            temp := ‘0’;
        elsif CK = ‘1’ and CK’event then
            case JK is
                when “01” =>
                    temp := ‘0’;
                when “10” =>
                    temp := ‘1’;
                when “11” =>
                    temp := not temp;
                when others =>
            end case;
        end if;
        Q <= temp;
```

```
            Qbar <= not temp;
        end process;
    end behave;
```

Example: Write a VHDL program to implement a 4 – bit up counter with asynchronous reset and load using behavioral model.

Solution:

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;
    entity counter_4_bit is
        port (D: in std_logic_vector(0 to 3); CK, RES, LD: in
        std_logic; Q: out std_logic_vector(0 to 3));
    end counter_4_bit;
    architecture behave of counter_4_bit is
    begin
        process (RES, CK, LD, D)
            variable count: std_logic_vector (0 to 3);
        begin
            if RES = '0' then
                count := "0000";
            elsif LD = '0' then
                count := D;
            elsif CK = '1' and CK'event then
                count := count + "0001";
            end if;
            Q <= count;
        end process;
    end behave;
```

Example: Write a VHDL program to implement a 4-bit binary to seven segment decoder using behavioral model

Solution:

```
    library ieee;
    use ieee.std_logic_1164.all;
    entity bin7seg is
        port(inp: in std_logic_vector(0 to 3);
            outp: out std_logic_vector(0 to 6));
    end bin7seg;
    architecture behave of bin7seg is
    begin
        process(inp)
        begin
            case inp is
                when "0000" =>
                    outp <= "1000000";
                when "0001" =>
                    outp <= "1111001";
                when "0010" =>
                    outp <= "0100100";
                when "0011" =>
                    outp <= "0110000";
                when "0100" =>
                    outp <= "0011001";
                when "0101" =>
                    outp <= "0010010";
                when "0110" =>
```

```
                    outp <= "0000010";
                when "0111" =>
                    outp <= "1111000";
                when "1000" =>
                    outp <= "0000000";
                when "1001" =>
                    outp <= "0010000";
                when "1010" =>
                    outp <= "0001000";
                when "1011" =>
                    outp <= "0000011";
                when "1100" =>
                    outp <= "1000110";
                when "1101" =>
                    outp <= "0100001";
                when "1110" =>
                    outp <= "0000110";
                when "1111" =>
                    outp <= "0001110";
            end case;
        end process;
    end behave;
```

Example: Write a VHDL program to display the output of the 4-bit binary counter in a seven segment display

Solution:

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

```
entity counter7seg is
    port(clk,    LD,    RES:    in    std_logic;    D:    in
    std_logic_vector(0 to 3); outp: out std_logic_vector(0
    to 6));
end counter7seg;
architecture struct of counter7seg is
component counter_4_bit
    port (D: in std_logic_vector(0 to 3); CK, RES, LD: in
    std_logic; Q: out std_logic_vector(0 to 3));
end component;
component bin7seg
    port(inp: in std_logic_vector(0 to 3);
    outp: out std_logic_vector(0 to 6));
end component;
signal q: std_logic_vector(0 to 3);
begin
    x1: counter_4_bit port map(D, CLK, RES, LD, q);
    x2: bin7seg port map(q, outp);
end struct;
```

Example: Write a VHDL program to implement a 32x4 RAM using behavioral model

Solution:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity RAM32x4 is
```

```
    port(CK, RD, WR: in std_logic; data_in: in
    std_logic_vector(0 to 3); addr: in std_logic_vector(0
    to 4); data_out: out std_logic_vector(0 to 3));
end RAM32x4;
architecture behave of RAM32x4 is
    type RAM is array(0 to 31) of std_logic_vector(0 to 3);
    signal myRAM: RAM;
begin
    process(CK, RD, WR)
    begin
    if (CK = '1' and CK'event) then
        if (RD = '1' and WR = '0') then
            data_out <= myRAM(to_integer(unsigned(addr)));
        end if;
        if (WR = '1' and RD = '0') then
            myRAM(to_integer(unsigned(addr))) <= data_in;
        end if;
    end if;
    end process;
end behave;
```

Example: Write a VHDL program to implement a 16x4 ROM using behavioral model

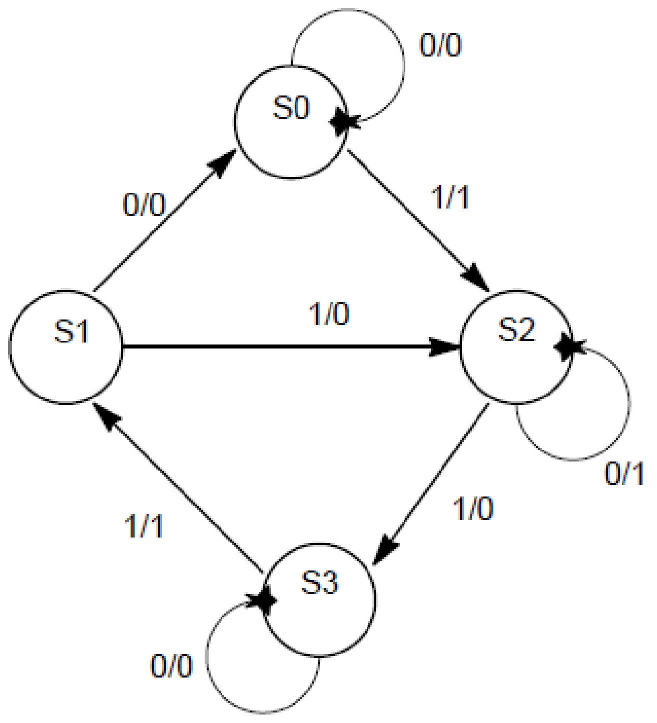
Solution:

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

```
entity ROM16x4 is
    port(CK, RD: in std_logic; addr: in std_logic_vector(0
    to 3); data_out: out std_logic_vector(0 to 3));
end ROM16x4;
architecture behave of ROM16x4 is
    type ROM is array(0 to 15) of std_logic_vector(0 to 3);
    signal myROM: ROM;
    myROM(0) <= "1101";
    myROM(1) <= "1001";
    myROM(2) <= "0001";
    …
    myROM(14) <= "0101";
    myROM(15) <= "1011";
begin
    process(CK, RD)
    begin
    if (CK = '1' and CK'event) then
        if (RD = '1') then
            data_out <= myROM(to_integer(unsigned(addr)));
        end if;
    end if;
    end process;
end behave;
```

Example: Write a VHDL program to implement the following state machine using behavioral model

Solution:



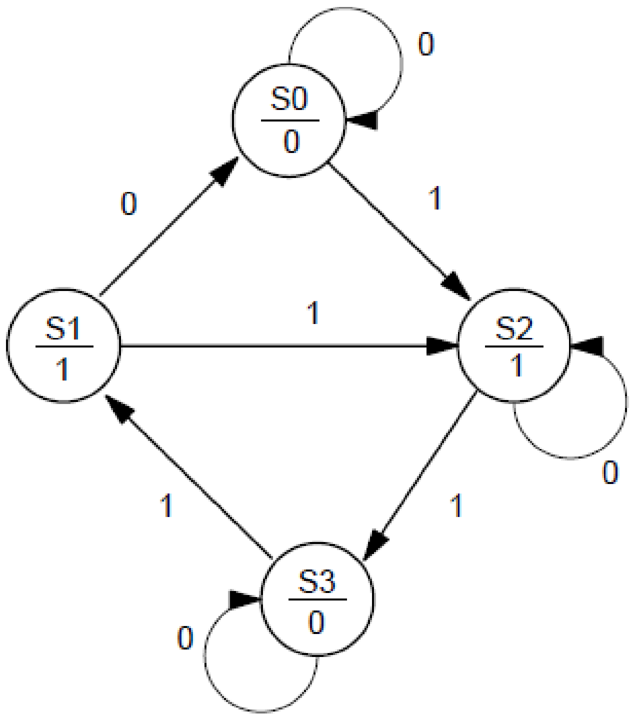
```
library ieee;
use ieee.std_logic_1164.all;
entity mealy_cct is
    port(ck, x, res: in std_logic; z: out std_logic);
end mealy_cct;
architecture behave of mealy_cct is
    type state_type is (S0, S1, S2, S3);
    signal p_state, n_state: state_type;
begin
    process(ck, res)
    begin
    if (res = '1') then


        p_state <= S0;
    elsif (ck = '1' and ck'event) then
        p_state <= n_state;
```

```
end if;
end process;
process(p_state, x)
begin
case p_state is
    when S0 =>
        if x = ‘0’ then
            n_state <= S0;
            z <= ‘0’;
        else
            n_state <= S2;
            z <= ‘1’;
        end if;
    when S1 =>
        if x = ‘0’ then
            n_state <= S0;
            z <= ‘0’;
        else
            n_state <= S2;
            z <= ‘0’;
        end if;
    when S2 =>
        if x = ‘0’ then
            n_state <= S2;
            z <= ‘1’;
        else
            n_state <= S3;
            z <= ‘0’;
```

```
                    end if;
                when S3 =>
                    if x = '0' then
                        n_state <= S3;
                        z <= '0';
                    else
                        n_state <= S1;
                        z <= '1';
                    end if;
            end process;
end behave;
```

Example: Write a VHDL program to implement the following state machine using behavioral model



Solution:

```
    library ieee;
```

```
use ieee.std_logic_1164.all;
entity moore_cct is
    port(ck, x, res: in std_logic; z: out std_logic);
end moore_cct;
architecture behave of moore_cct is
    type state_type is (S0, S1, S2, S3);
    signal p_state, n_state: state_type;
begin
    process(ck, res)
    begin
    if (res = ‘1’) then
        p_state <= S0;
    elsif (ck = '1' and ck'event) then
        p_state <= n_state;
    end if;
    end process;
    process(p_state, x)
    begin
    case p_state is
        when S0 =>
            z <= ‘0’;
            if x = ‘0’ then
                n_state <= S0;
            else
                n_state <= S2;
            end if;
        when S1 =>
            z <= ‘1’;
```

```
                    if x = ‘0’ then
                        n_state <= S0;
                    else
                        n_state <= S2;
                    end if;
                when S2 =>
                    z <= ‘1’;
                    if x = ‘0’ then
                        n_state <= S2;
                    else
                        n_state <= S3;
                    end if;
                when S3 =>
                    z <= ‘0’;
                    if x = ‘0’ then
                        n_state <= S3;
                    else
                        n_state <= S1;
                    end if;
            end process;
end behave;
```

Example: Write a VHDL program to implement N-bit register with clear, load and increment ability using behavioral model

Solution:

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

```
use ieee.numeric_std.all;
entity REG_N is
     generic (N: integer:= 8);
     port (CK, CLR, LD, INC: in std_logic;
          INP: in std_logic_vector (0 to N - 1);
          OUTP: out std_logic_vector (0 to N - 1));
end REG_N;
architecture behave of REG_N is
     signal temp: std_logic_vector(0 to N – 1);
begin
     process(CK, LD, CLR, INC)
     begin
     if (CLR = ‘1’) then
          temp <= (others => ‘0’);
     elsif (CK = '1' and CK'event) then
          if (LD = ‘1’) then
               temp <= INP;
          end if;
          if (INC = '1') then
               temp <= std_logic_vector(unsigned(temp) + 1);
          end if;
     end if;
     end process;
     OUTP <= temp;
end behave;
```