Write a VHDL code for 4:1 MUX using structural Modeling style

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Entity declaration for the 2-to-1 MUX
ENTITY mux2to1 IS
   PORT(
      a, b, sel : IN  std_logic;
      y         : OUT std_logic
   );
END mux2to1;

ARCHITECTURE behavior OF mux2to1 IS
BEGIN
   y <= a WHEN sel = '0' ELSE b;
END behavior;

-- Entity declaration for the 4-to-1 MUX
ENTITY mux4to1 IS
   PORT(
      d0, d1, d2, d3 : IN  std_logic;
      s0, s1         : IN  std_logic;
      y              : OUT std_logic
   );
END mux4to1;

ARCHITECTURE structural OF mux4to1 IS

   SIGNAL y0, y1 : std_logic;  -- Internal signals

   COMPONENT mux2to1
      PORT(
         a, b, sel : IN  std_logic;
         y         : OUT std_logic
      );
   END COMPONENT;

BEGIN
   -- Instantiate the first level of 2-to-1 MUXes
   U1: mux2to1 PORT MAP (a => d0, b => d1, sel => s0, y => y0);
   U2: mux2to1 PORT MAP (a => d2, b => d3, sel => s0, y => y1);
   -- Instantiate the second level of 2-to-1 MUX
   U3: mux2to1 PORT MAP (a => y0, b => y1, sel => s1, y => y);

END structural;
```

Write a VHDL code for 8:1 MUX using structural Modeling style

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Entity declaration for the 2-to-1 MUX
ENTITY mux2to1 IS
   PORT(
      a, b, sel : IN  std_logic;
      y         : OUT std_logic
   );
END mux2to1;

ARCHITECTURE behavior OF mux2to1 IS
BEGIN
   y <= a WHEN sel = '0' ELSE b;
END behavior;

-- Entity declaration for the 8-to-1 MUX
ENTITY mux8to1 IS
   PORT(
      d0, d1, d2, d3, d4, d5, d6, d7 : IN  std_logic;
      s0, s1, s2                     : IN  std_logic;
      y                              : OUT std_logic
   );
END mux8to1;

ARCHITECTURE structural OF mux8to1 IS

   SIGNAL y0, y1, y2, y3 : std_logic;
   SIGNAL y4, y5         : std_logic;

   COMPONENT mux2to1
      PORT(
         a, b, sel : IN  std_logic;
         y         : OUT std_logic
      );
   END COMPONENT;

BEGIN
   -- Instantiate the first level of 2-to-1 MUXes
   U1: mux2to1 PORT MAP (a => d0, b => d1, sel => s0, y => y0);
   U2: mux2to1 PORT MAP (a => d2, b => d3, sel => s0, y => y1);
   U3: mux2to1 PORT MAP (a => d4, b => d5, sel => s0, y => y2);
   U4: mux2to1 PORT MAP (a => d6, b => d7, sel => s0, y => y3);
```

```
    -- Instantiate the second level of 2-to-1 MUXes
    U5: mux2to1 PORT MAP (a => y0, b => y1, sel => s1, y => y4);
    U6: mux2to1 PORT MAP (a => y2, b => y3, sel => s1, y => y5);

    -- Instantiate the third level of 2-to-1 MUX
    U7: mux2to1 PORT MAP (a => y4, b => y5, sel => s2, y => y);

END structural;
```

VHDL code for an 8-to-1 multiplexer (MUX) using the behavioral modeling style

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux8to1 IS
    PORT(
        d0, d1, d2, d3, d4, d5, d6, d7 : IN  std_logic;
        s                              : IN  std_logic_vector(2 DOWNTO 0);
        y                              : OUT std_logic
    );
END mux8to1;

ARCHITECTURE behavior OF mux8to1 IS
BEGIN
    PROCESS(d0, d1, d2, d3, d4, d5, d6, d7, s)
    BEGIN
        CASE s IS
            WHEN "000" => y <= d0;
            WHEN "001" => y <= d1;
            WHEN "010" => y <= d2;
            WHEN "011" => y <= d3;
            WHEN "100" => y <= d4;
            WHEN "101" => y <= d5;
            WHEN "110" => y <= d6;
            WHEN "111" => y <= d7;
            WHEN OTHERS => y <= '0';  -- Default case
        END CASE;
    END PROCESS;
END behavior;
```

Write a VHDL code to simulate 1 × 8 Demultiplexer

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY demux1x8 IS
   PORT(
      d      : IN  std_logic;
      sel    : IN  std_logic_vector(2 DOWNTO 0);
      y0, y1, y2, y3, y4, y5, y6, y7 : OUT std_logic
   );
END demux1x8;

ARCHITECTURE behavior OF demux1x8 IS
BEGIN
   PROCESS(d, sel)
   BEGIN
      -- Default values
      y0 <= '0';
      y1 <= '0';
      y2 <= '0';
      y3 <= '0';
      y4 <= '0';
      y5 <= '0';
      y6 <= '0';
      y7 <= '0';

      CASE sel IS
         WHEN "000" => y0 <= d;
         WHEN "001" => y1 <= d;
         WHEN "010" => y2 <= d;
         WHEN "011" => y3 <= d;
         WHEN "100" => y4 <= d;
         WHEN "101" => y5 <= d;
         WHEN "110" => y6 <= d;
         WHEN "111" => y7 <= d;
         WHEN OTHERS =>
            y0 <= '0';
            y1 <= '0';
            y2 <= '0';
            y3 <= '0';
            y4 <= '0';
            y5 <= '0';
            y6 <= '0';
            y7 <= '0';
      END CASE;
```

```
    END PROCESS;
END behavior;
```

VHDL program for which output will be 1 when the sequence 1101 is detected considering overlapping condition

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

entity sequence_detector is
    port (
        clk: in std_logic;              -- Clock input
        reset: in std_logic;            -- Reset input
        input_bit: in std_logic;        -- Input bit stream
        output_detected: out std_logic -- Output indicating sequence detection
    );
end entity sequence_detector;

architecture behavioral of sequence_detector is
    -- Define states
    type state_type is (A, B, C, D);
    signal current_state, next_state: state_type;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            current_state <= A;  -- Initial state
        elsif rising_edge(clk) then
            current_state <= next_state;  -- State transition on clock edge
        end if;
    end process;

    process (current_state, input_bit)
    begin
        case current_state is
            when A =>
                if input_bit = '1' then
                    next_state <= B;
                else
                    next_state <= A;
                end if;

            when B =>
                if input_bit = '1' then
```

```
                next_state <= C;
            else
                next_state <= A;
            end if;

        when C =>
            if input_bit = '0' then
                next_state <= D;
            else
                next_state <= C;
            end if;

        when D =>
            if input_bit = '1' then
                next_state <= B;  -- Detects "1101" and loops back to detect more occurrences
            else
                next_state <= A;
            end if;

        when others =>
            next_state <= A;
    end case;
end process;

-- Output assignment
process (current_state)
begin
    output_detected <= '0';
    case current_state is
        when D =>
            output_detected <= '1';  -- Output is '1' when "1101" sequence is detected
        when others =>
            output_detected <= '0';
    end case;
end process;
end architecture behavioral;
```