VHDL

July 2, 2023 – Embedded System

By Prashant Poudel

Prologue

- Initially, digital circuits were manually designed using techniques like Boolean expressions.
- With the increasing device densities the choice of this traditional methods has become limited.
- Thus we need EDA (Electronic Design Automation) tools to design complex circuits like HDL.
- What is HDL?
  - Hardware Description Language

VHDL

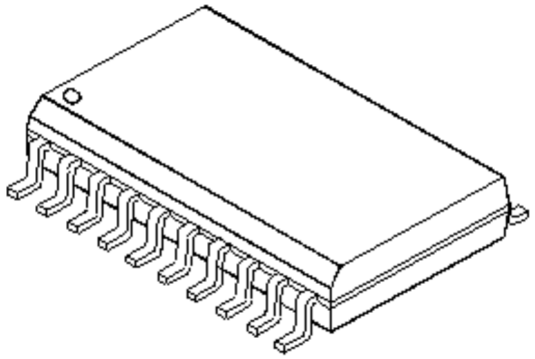
- **V**ery High Speed Integrated Circuit (VHSIC) **H**ardware **D**escription **L**anguage.
- It is used to describe the behavior of an electronic system, which further enables designer to implement the physical system
- It is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling
- Hardware Description Language (HDL) is an industry standard language used to describe hardware from the abstract to concrete level.
- With VHDL, designers can simulate and verify the functionality of a digital circuit before actual implementation.

Basic design units of VHDL

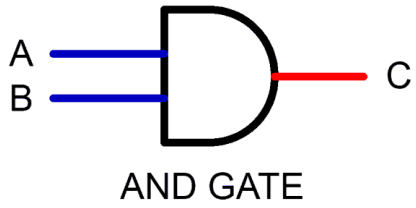
- **Entity:** Defines the interface or external behavior of a design entity.
- **Architecture:** Defines the internal behavior or structure of a design entity.
- **Package:** Contains globally accessible declarations for sharing across design entities.
- **Library:** Organizational unit for managing design units and their dependencies.
- **Configuration:** Specifies the binding of design units to actual entities or architectures.
- **Testbench:** Separate design unit used for simulation and testing purposes.

Entity

- The entity describes the interface to the outside world (connection pins of package).
- Entity in VHDL is used to define the interface or external behavior of a design entity. It specifies the inputs, outputs, and other properties of the entity.

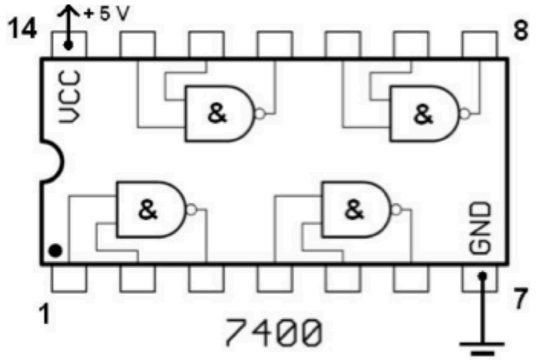


```
entity and_gate is
  port (
    A : in std_logic;
    B : in std_logic;
    C : out std_logic
  );
end and_gate;
```

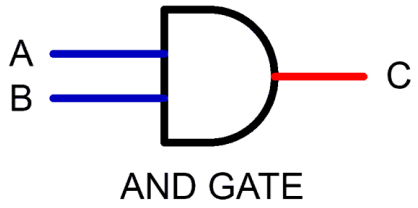


Architecture

- The architecture describes the functionality of the circuit inside the entity (package)
- Architecture in VHDL defines the internal behavior or structure of a design entity. It describes how the inputs are processed to produce the desired outputs.




```
architecture behavioral of and_gate is
begin
  C <= A and B;
end behavioral;
```



Configuration

- In VHDL, a configuration is used to specify the binding of design units to actual entities or architectures. It determines the specific implementation to be used during synthesis or simulation.

```
configuration and_gate_config of and_gate is
  for behavioral
  for and_gate
  end for;
end configuration and_gate_config;
```

In this example:

- The configuration name is **and_gate_config**.
- The configuration is associated with the entity **and_gate** and the architecture **behavioral**.
- The configuration specifies the binding of the **and_gate** entity with its **behavioral** architecture.

Packages and Library

```
package mux_constants is
  constant WIDTH: natural := 4;
  constant SELECT_WIDTH: natural := 2;
end package mux_constants;
```

```
library my_lib;
use my_lib.my_package.all;
```

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

Data Types in VHDL

std_logic type represents a single bit of digital information and has multiple values like:

- '0' (logic low)
- '1' (logic high)
- 'Z' (high-impedance or floating)
- 'X' (unknown)
- 'W' (weak signal)
- 'L' (weak signal, more likely to be low)
- 'H' (weak signal, more likely to be high)
- '-' (don't care)

```
signal input_signal : std_logic;
signal output_signal : std_logic;
```

std_logic_vector represents an array of std_logic values and is commonly used to represent multi-bit signals or buses.

```
signal data_bus : std_logic_vector(7 downto 0);
```

Data Types in VHDL (Contd…)

Integer

```
signal counter : integer := 0;
signal index : natural := 0;
signal delay: positive := 5;
```

Floating

```
signal real_value : float := 3.14;
```

Array

```
type data_array is array (0 to 7) of std_logic_vector(7 downto 0);
signal memory : data_array;
```

Statements in VHDL

1. Concurrent Signal Assignment: Assigns a value to a signal in a concurrent manner, meaning that the assignments happen simultaneously without any defined sequencing.

```
-- Syntax:
signal_name <= value_expression;

-- Example:
signal A: std_logic;
signal B: std_logic;
signal C: std_logic;
C <= A and B;
```

Statements in VHDL (Contd…)

2. Conditional Signal Assignment: Provides a way to conditionally assign a value to a signal based on a given condition. It allows for different signal assignments based on the evaluation of the condition.

```
-- Syntax:
signal_name <= value_expression when condition_expression else
alternative_expression;

-- Example:
signal A, B, C: std_logic;
C <= '1' when (A = '1' and B = '0') else '0';
```

Statements in VHDL (Contd…)

3. Selected Signal Assignment: Allows selecting one of several signal assignments based on the value of a select expression.

```
-- Syntax:
with select_expression select
  signal_name <= value_expression1 when choice1,
                 value_expression2 when choice2,
                 ...
                 value_expressionN when choiceN,
                 value_expression_default when others;

-- Example:
signal A: std_logic_vector(1 downto 0);
signal B: std_logic_vector(3 downto 0);
with A select
  B <= "0000" when "00",
              "0001" when "01",
             "0010" when "10",
             "0100" when "11",
             "1111" when others;
```

Statements in VHDL (Contd…)

4. Process Statement: Represents a block of sequential statements that are executed in the order written within a process.

```
-- Syntax:
process (sensitivity_list)
begin
  -- Sequential statements
end process;

-- Example:
process (clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      counter <= (others => '0');
    else
      counter <= counter + 1;
    end if;
  end if;
end process;
```

Statements in VHDL (Contd…)

5. Signal Assignment (Sequential): Assigns a value to a signal within a sequential block of code, such as a process or subprogram.

```
-- Syntax:
signal_name <= value_expression;

-- Example:
signal A, B, C: std_logic;
process
begin
  A <= '1';
  B <= A and '0';
  C <= A or B;
end process;
```

Statements in VHDL (Contd…)

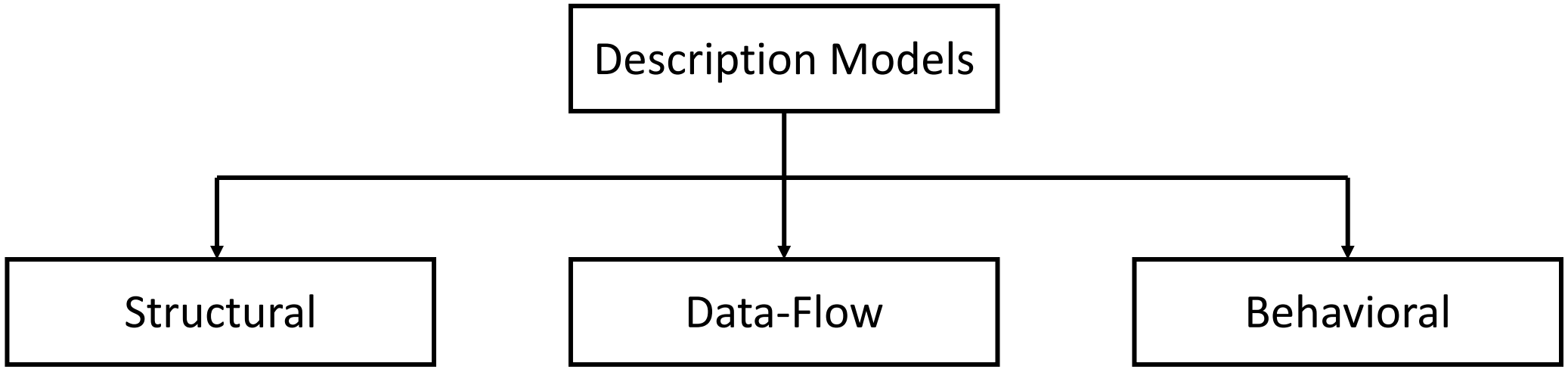
6. If Statement (Sequential):

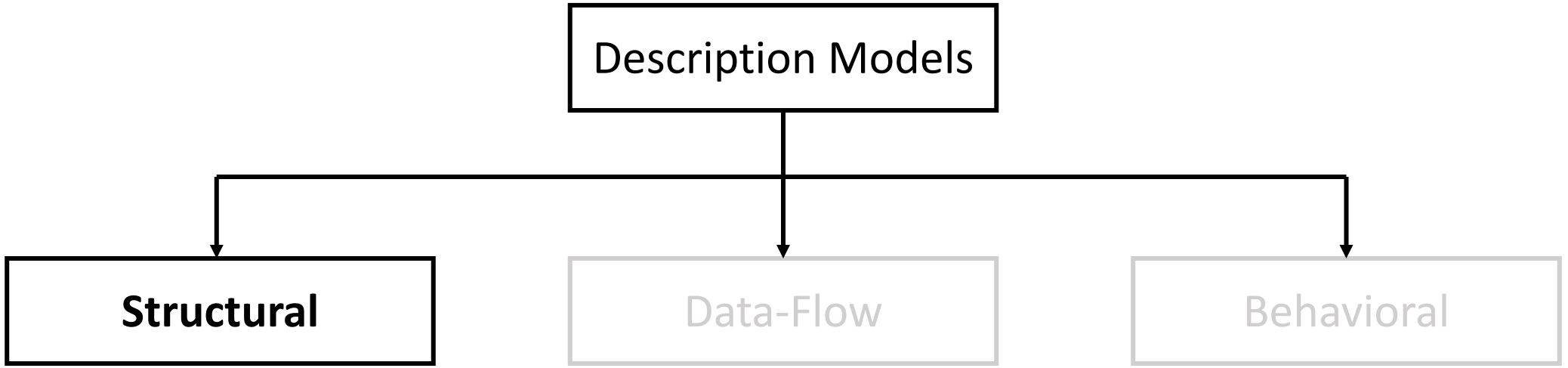
```
-- Syntax:
if condition then
  -- Sequential statements
else
  -- Sequential statements
end if;
```

7. Case Statement (Sequential):

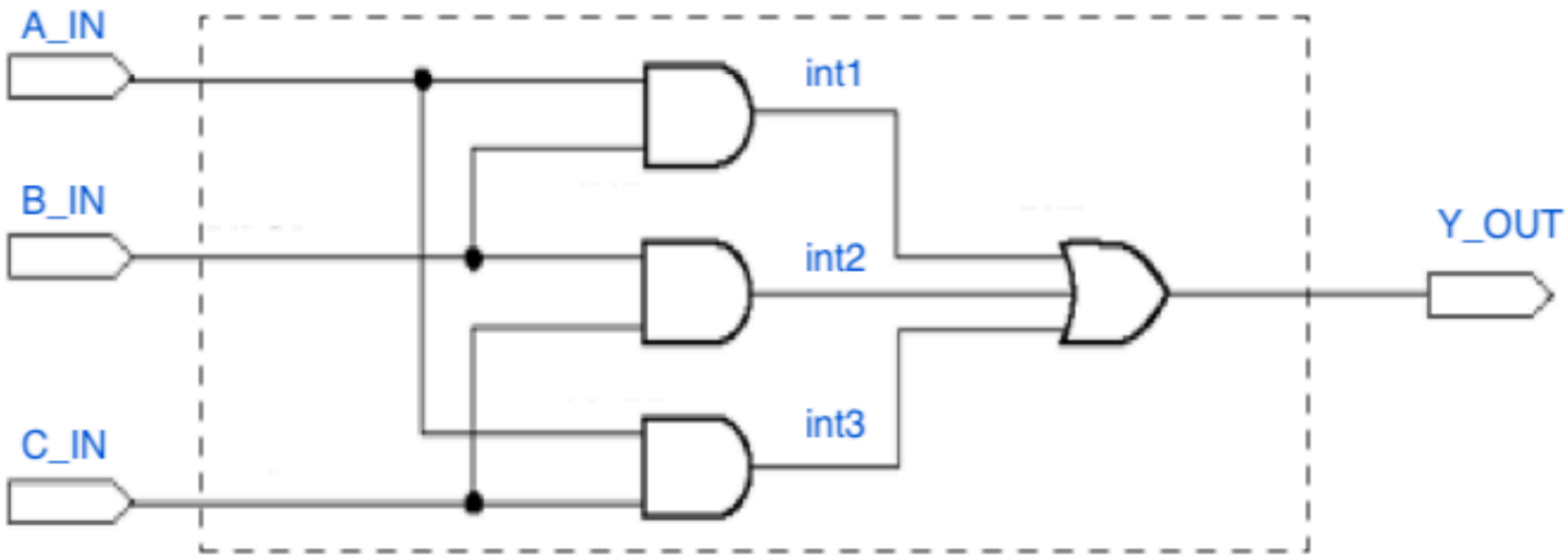
```
-- Syntax:
case expression is
  when choice1 =>
    -- Sequential statements for choice1
  when choice2 =>
    -- Sequential statements for choice2
  ...
  when others =>
    -- Sequential statements for all other choices
end case;
```

VHDL – Description Models



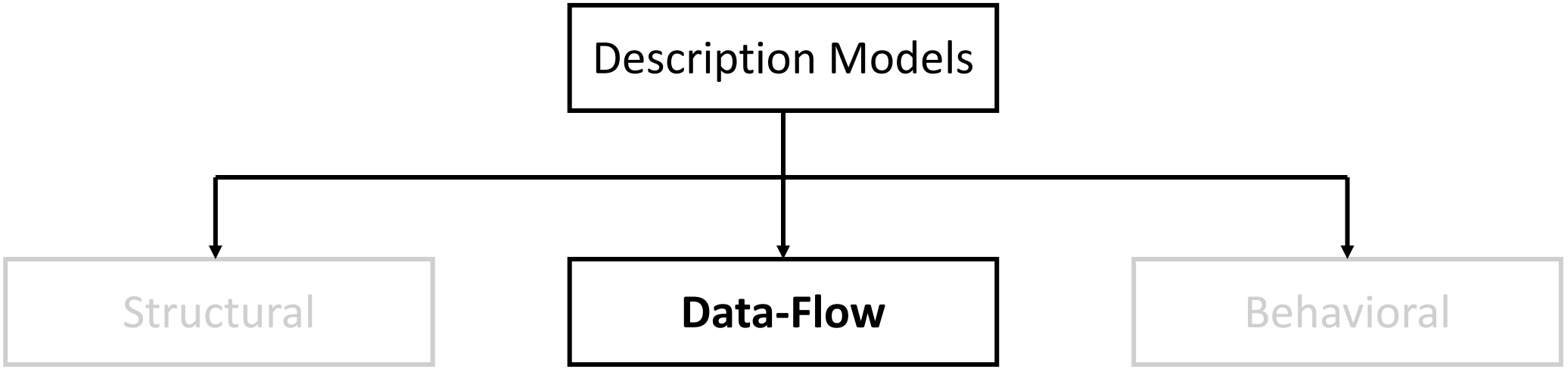


- In structural modeling, an entity is represented as a group of connected components.
- Structural modeling focuses on connecting components without specifying their individual behavior.
- Components are like black boxes, where we don't look inside or know how they work.
- Structural modeling doesn't describe the behavior of the components or the entity as a whole.


A_IN
B_IN
C_IN
int1
int2
int3
Y_OUT

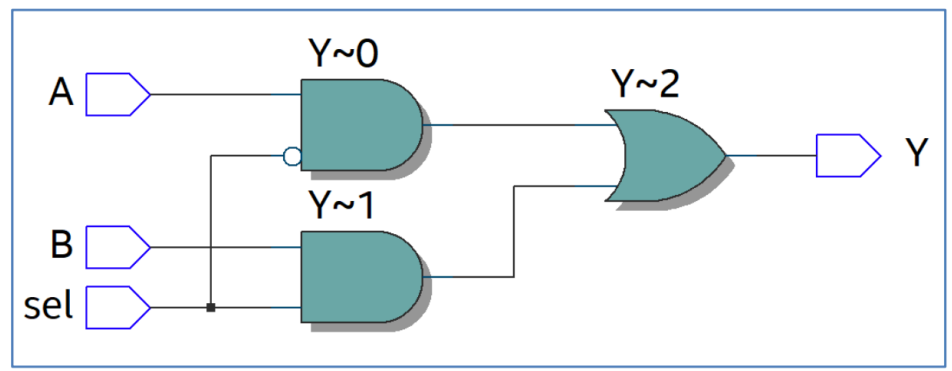
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity example is
  port(
    A_IN  : in std_logic;
    B_IN  : in std_logic;
    C_IN  : in std_logic;
    Y_OUT : out std_logic
    );
end entity;

architecture structural of example is

  signal int1 : std_logic;
  signal int2 : std_logic;
  signal int3 : std_logic;


  component AND_GATE
    port(
      A, B : in  std_logic;
      Y    : out std_logic
      );
  end component;

```

```
27
28
29    component OR_GATE
30      port(
31        A, B, C : in  std_logic;
32        Y       : out std_logic
33        );
34    end component;
35
36
37  begin
38
39
40    A1 : AND_GATE port map (A => A_IN, B => B_IN, Y => int1);
41    A2 : AND_GATE port map (A => A_IN, B => B_IN, Y => int2);
42    A3 : AND_GATE port map (A => A_IN, B => B_IN, Y => int3);
43    O1 : OR_GATE port map (A  => int1, B => int2, C => int3, Y => Y_OUT);
44
45
46  end architecture;
47
```



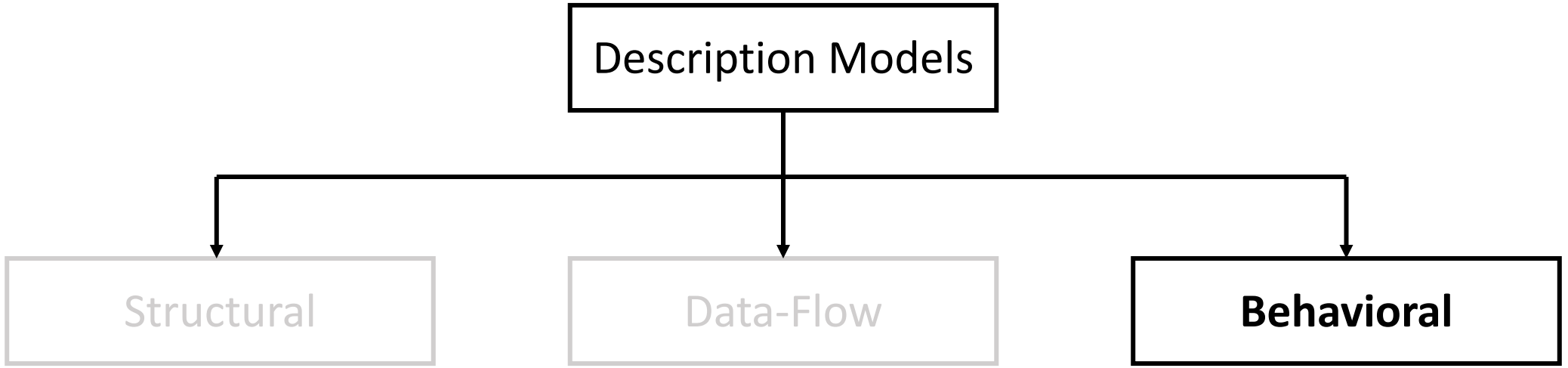
- Data-flow models use concurrent signal assignments to represent the flow of data.
- The value of a signal is assigned based on the current values of other signals using logical and arithmetic operations, conditional statements, and combinational logic.
- Statements are activated when there is a change in the input.

```
architecture top_level of example is

-- Parallel declaration part

begin
  ..........................................................................
   -- Concurrent / parallel statements area



  Y <= ((not sel) and A) or (sel and B);
```




```
  ..........................................................................
end architecture;
```

Concurrent statements area



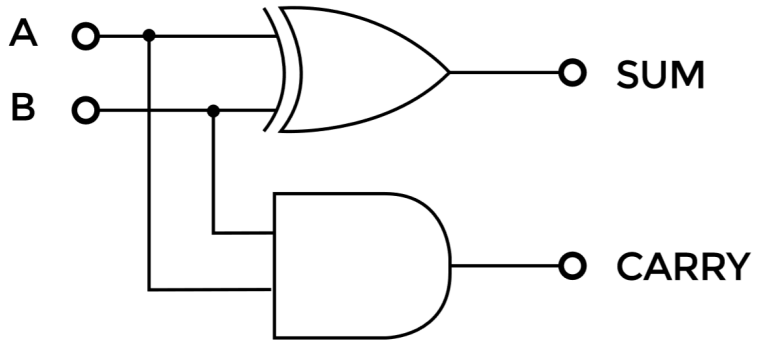
- Behavioral modeling describes the functionality or behavior of a digital system, It focuses on the desired behavior without specifying the internal structure or implementation details.
- Processes or subprograms containing sequential or concurrent statements are used to define the system's behavior.
- Abstraction is a key characteristic, providing a high-level description of the system's behavior.

```
architecture top_level of example is

-- Parallel declaration part

begin
  -- Concurrent / parallel statements area

  process() is

    -- Sequential declaration area

  begin

    -- Sequential VHDL

  end process;

  -- Concurrent / parallel statements area


end architecture;
```

process:
a concurrent
VHDL
statemen

Concurrent
statements
area

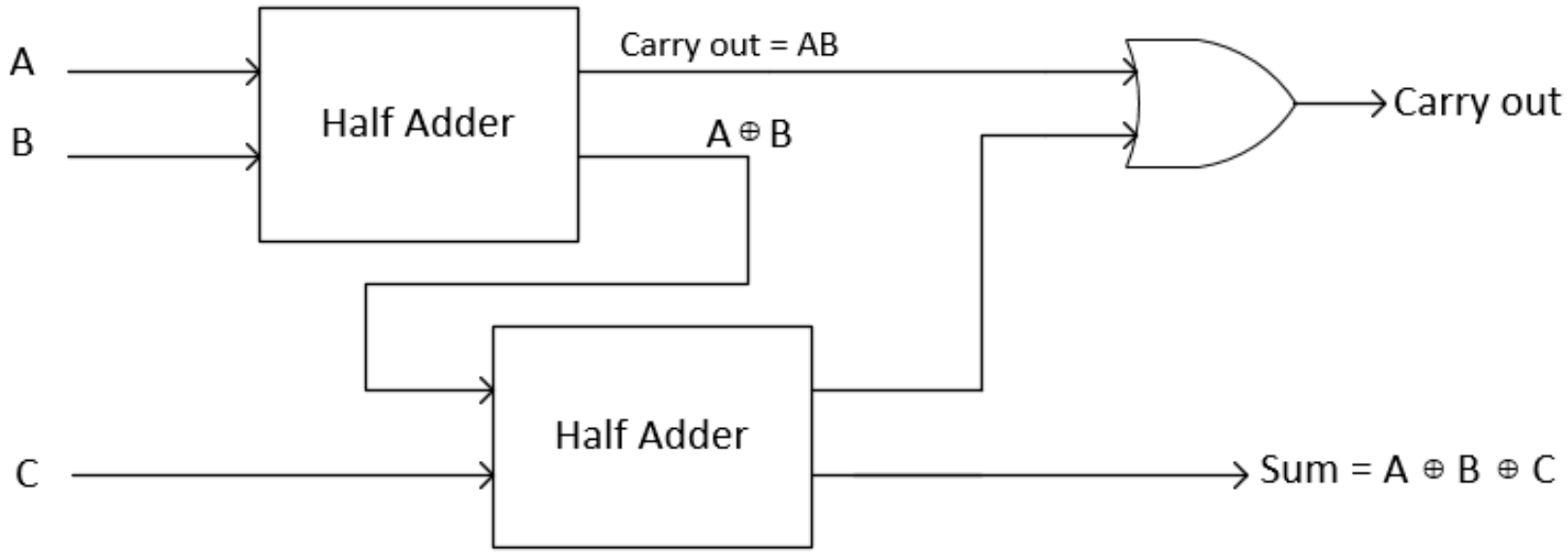
Q. Write a VHDL code for half adder.



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

```
Library ieee;
use ieee.std_logic_1164.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;
```

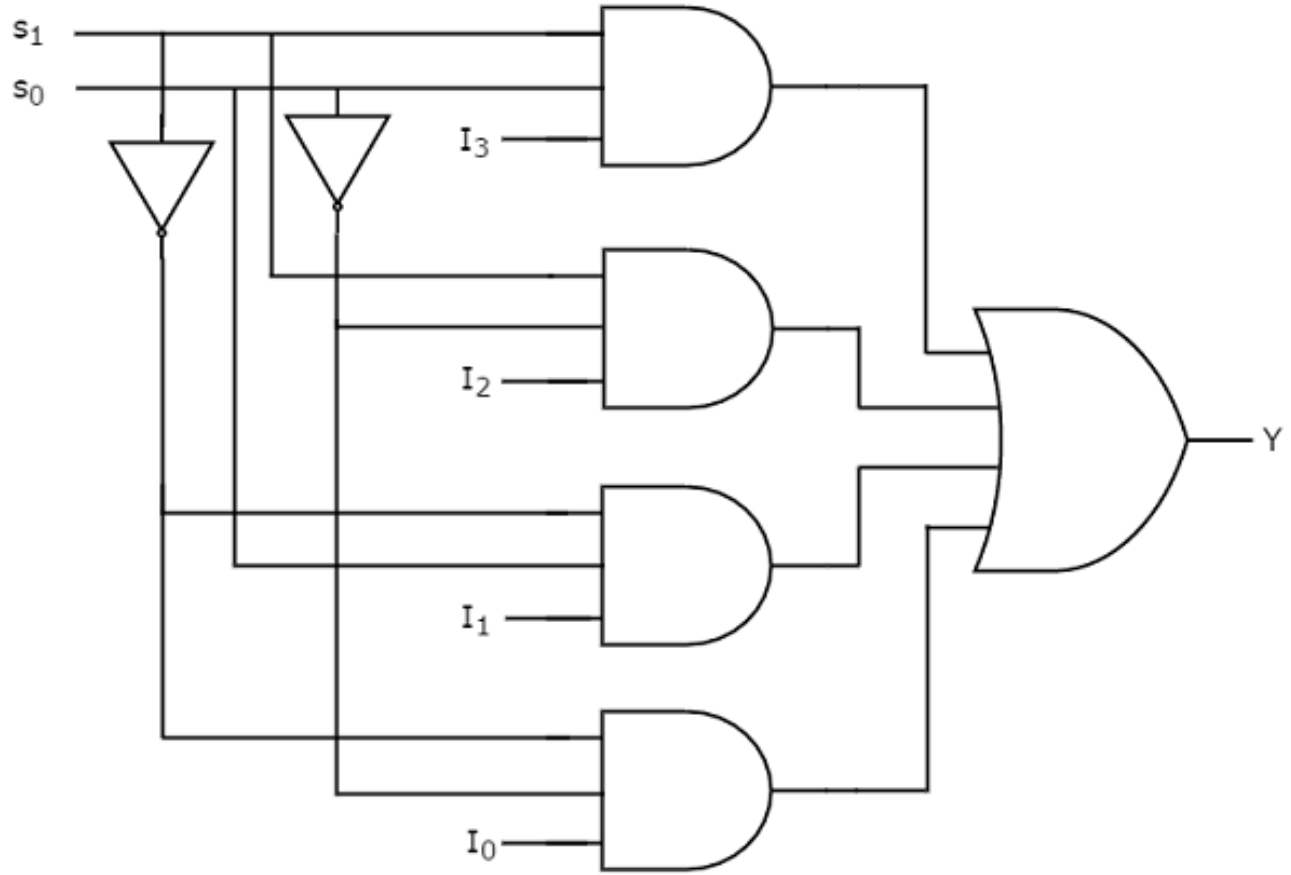
Q. Write a VHDL code for a full adder using 2 half adder as component.



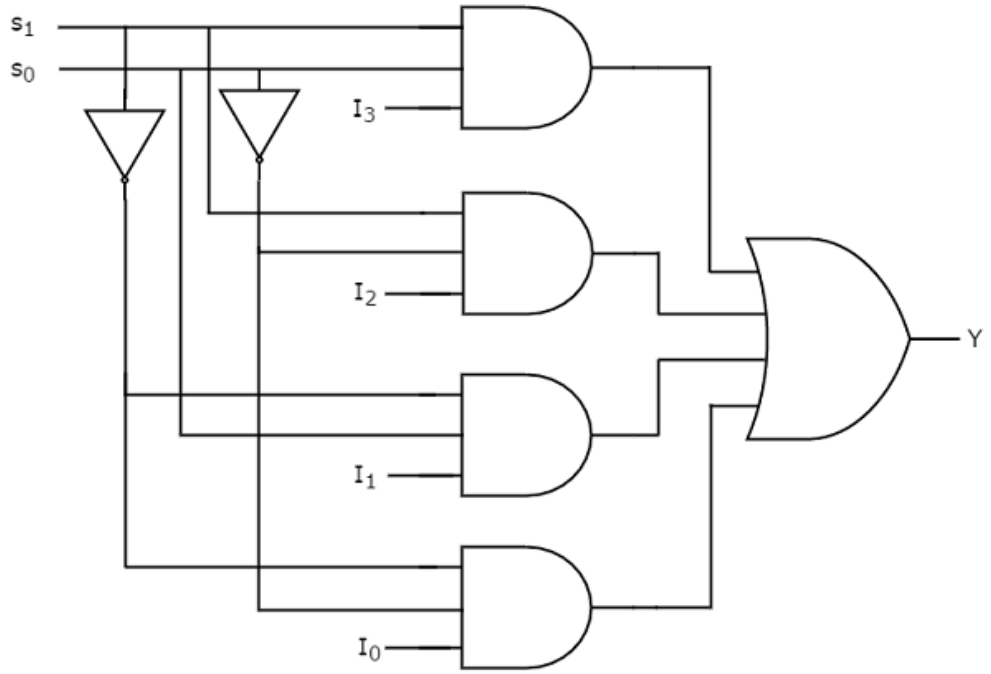
```vhdl
    -- Entity declaration
    Library ieee;
    use ieee.std_logic_1164.all;

    entity FullAdder is
        port (
            A, B, C: in std_logic;
            Sum, Cout: out std_logic
        );
    end entity FullAdder;

-- Architecture definition
architecture Behavioral of FullAdder is
    -- Component declaration for HalfAdder
    component HalfAdder is
        port (
            A, B: in std_logic;
            Sum, Carry: out std_logic
        );
    end component HalfAdder;

    -- Signals declaration
    signal S1, S2, C1, C2: std_logic;
```

```vhdl
begin
    -- Instantiation of the first HalfAdder component
    HA1: HalfAdder
        port map (
            A => A,
            B => B,
            Sum => S1,
            Carry => C1
        );

    -- Instantiation of the second HalfAdder component
    HA2: HalfAdder
        port map (
            A => S1,
            B => C,
            Sum => Sum,
            Carry => C2
        );

    -- Cout assignment
    Cout <= C1 or C2;
end architecture Behavioral;
```

Q. Write a VHDL code for a 4-to-1 multiplexer.

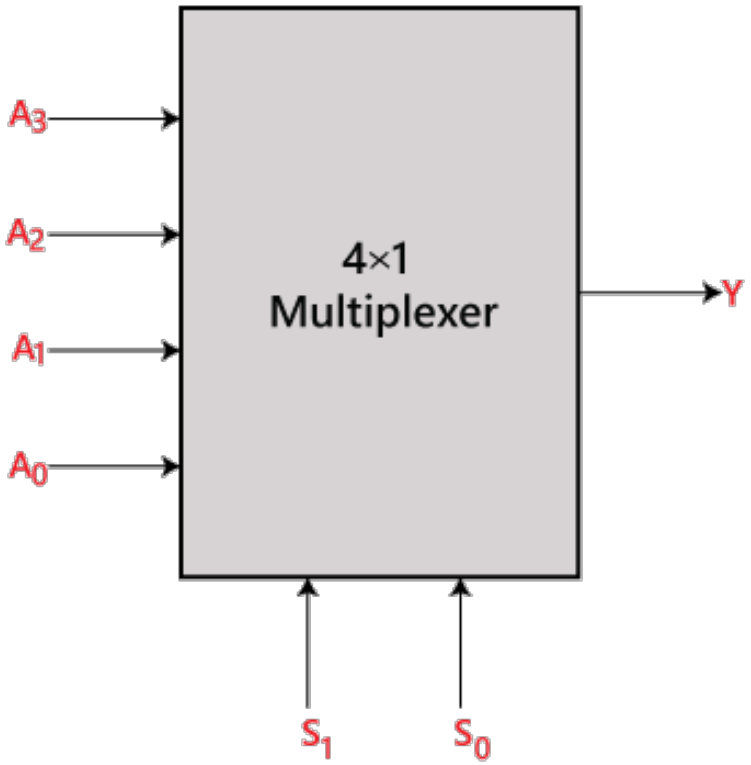


Q. Write a VHDL code for a 4-to-1 multiplexer.




```vhdl
Library ieee;
use ieee.std_logic_1164.all;

entity mux is
   port(S1,S0,I0,I1,I2,I3:in bit; Y:out bit);
end mux;

architecture data of mux is
begin
   Y<= (not S0 and not S1 and I0) or
      (S0 and not S1 and I1) or
      (not S0 and S1 and I2) or
      (S0 and S1 and I3);
end data;
```

Q. Write a VHDL code for a 4-to-1 multiplexer.



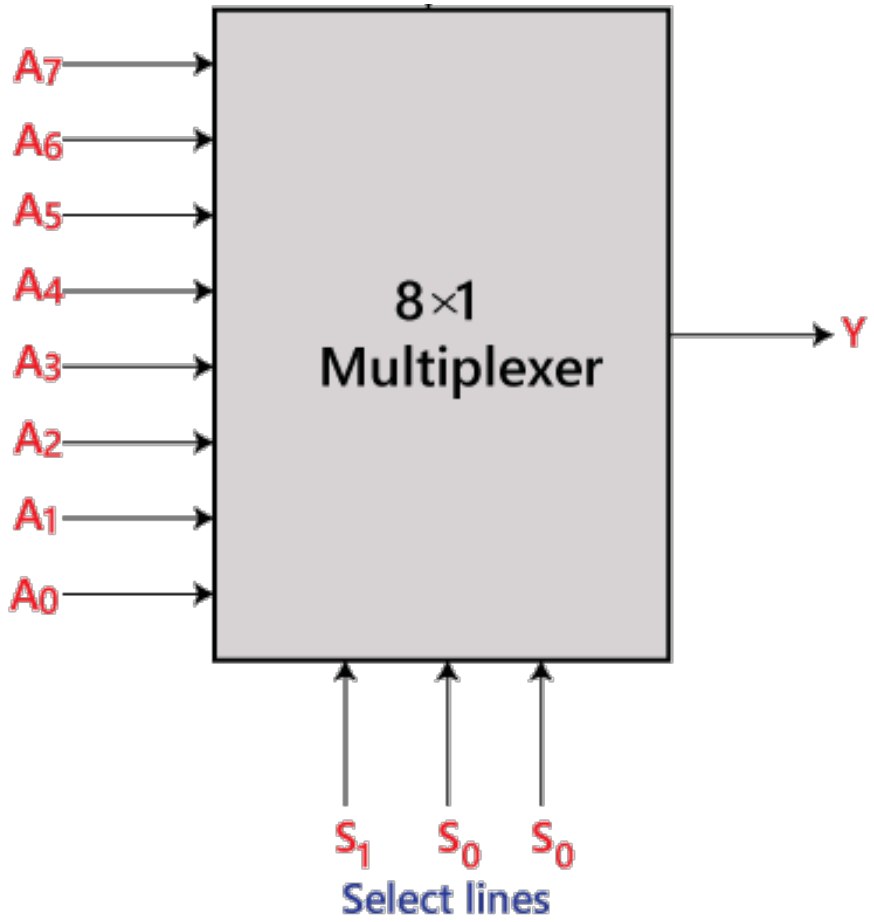
INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

$$Y = S_1' S_0' A_0 + S_1' S_0 A_1 + S_1 S_0' A_2 + S_1 S_0 A_3$$

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

entity MUX4to1 is
    port (
        A0, A1, A2, A3: in std_logic;
        S0, S1: in std_logic;
        Y: out std_logic
    );
end MUX4to1;
```

```vhdl
architecture Behavioral of MUX4to1 is
begin
    process (A0, A1, A2, A3, S0, S1)
    begin
        case (S1 & S0) is
            when "00" =>
                Y <= A0;
            when "01" =>
                Y <= A1;
            when "10" =>
                Y <= A2;
            when "11" =>
                Y <= A3;
            when others =>
                Y <= 'X';
        end case;
    end process;
end Behavioral;
```

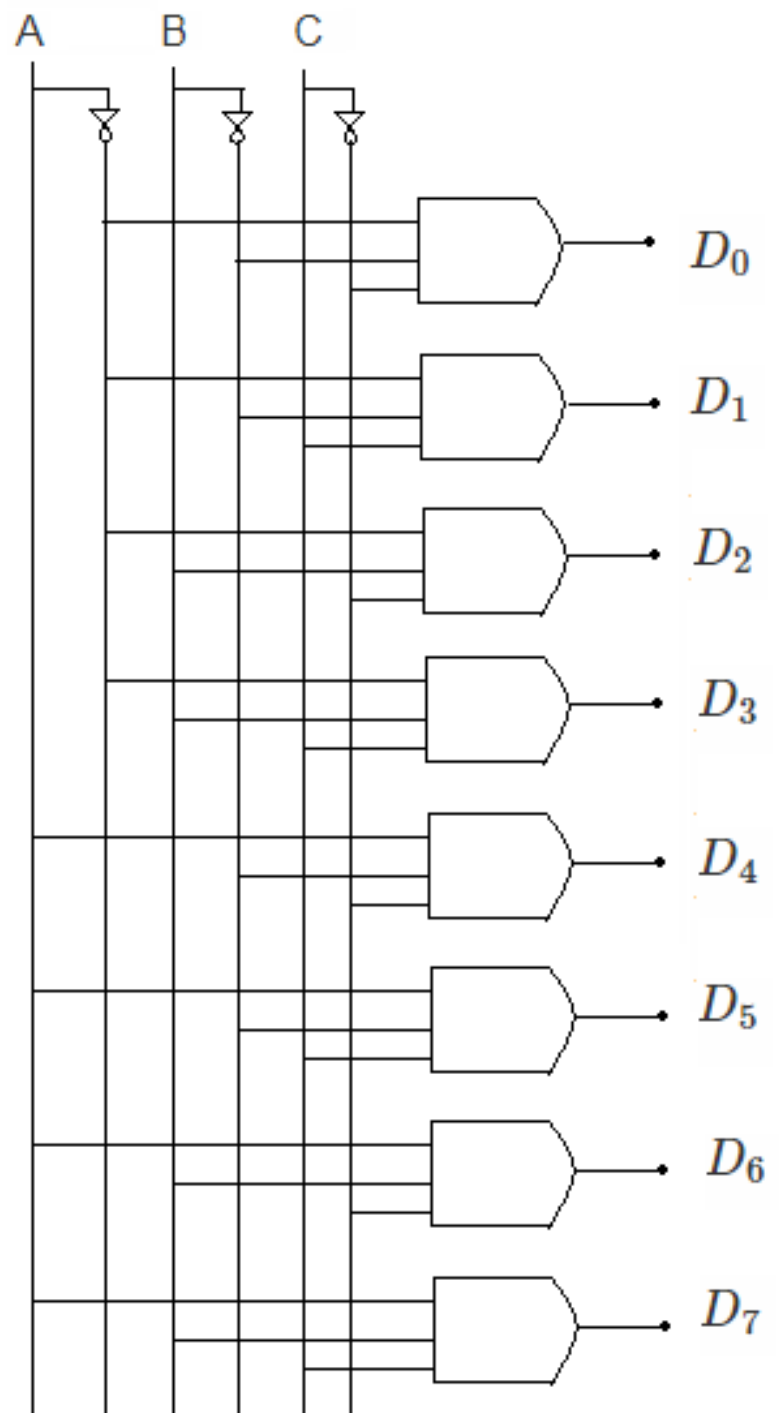
Q. Write a VHDL code for a 8-to-1 multiplexer.



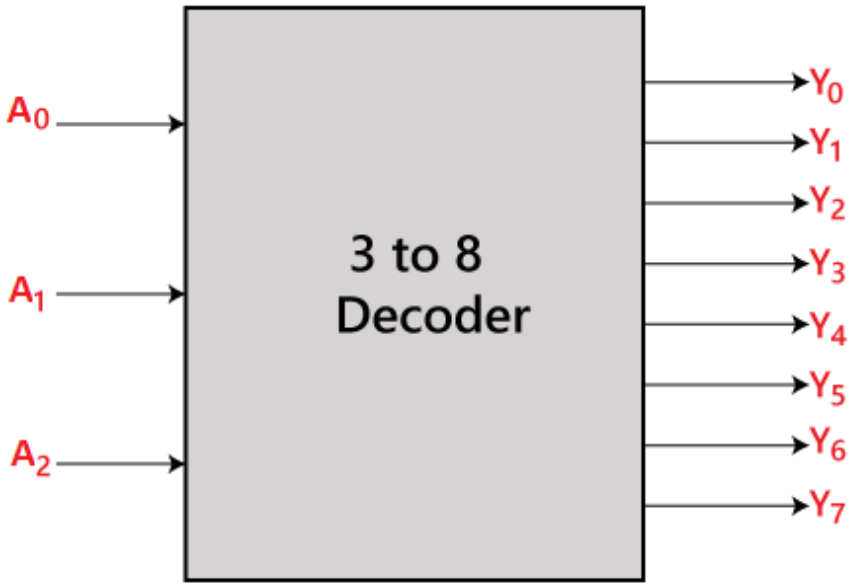
INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

Q. Write a VHDL code for a 3-to-8 decoder.

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

entity dec is
   port(A,B,C:in bit; D0,D1,D2,D3,D4,D5,D6,D7: out bit);
end dec;

architecture data of dec is
begin
   D0<=(not A) and (not B) and (not C);
   D1<=(not A) and (not B) and C;
   D2<=(not A) and B and (not C);
   D3<=(not A) and B and C;
   D4<=A and (not B) and (not C);
   D5<=A and (not B) and C;
   D6<=A and B and (not C);
   D7<=A and B and C;
end data;
```



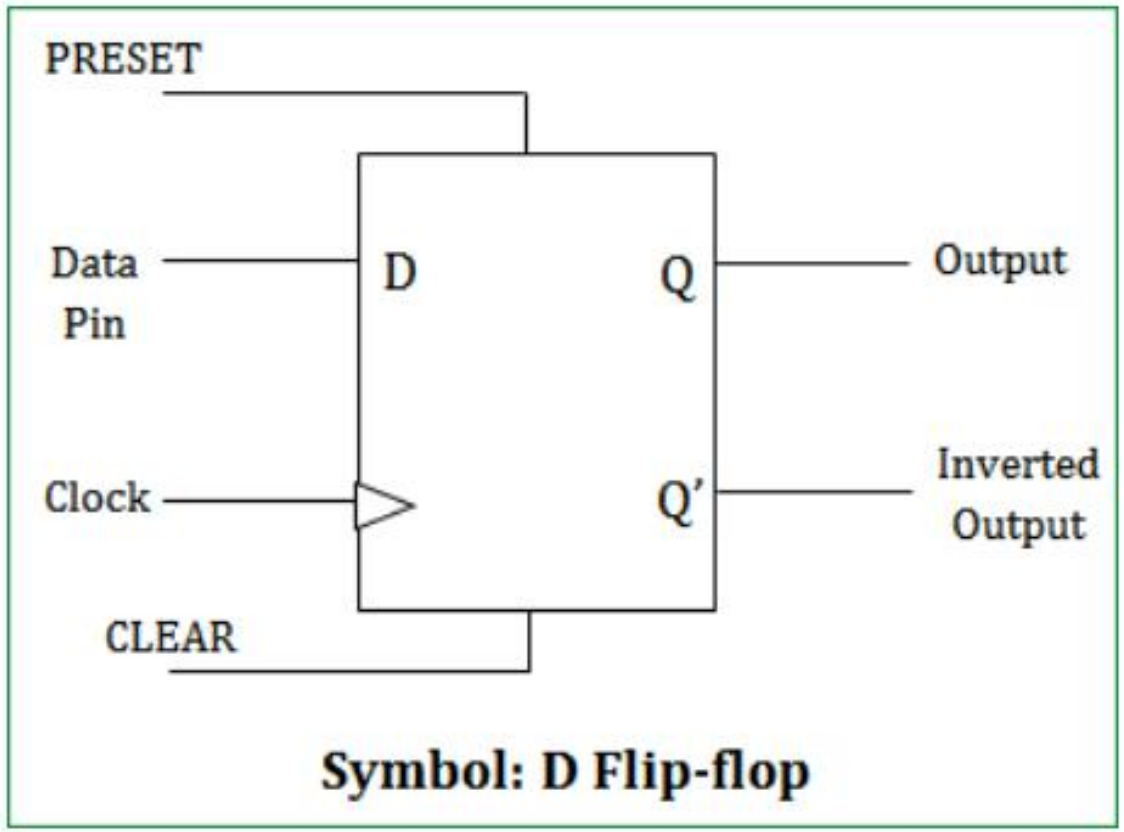
Q. Write a VHDL code for a 3-to-8 decoder.



Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity Decoder3to8 is
    port (
        A, B, C: in std_logic;
        Y: out std_logic_vector(7 downto 0)
    );
end Decoder3to8;
architecture Behavioral of Decoder3to8 is
begin
    process (A, B, C)
    begin
        case (A & B & C) is
            when "000" =>
                Y <= "00000001";
            when "001" =>
                Y <= "00000010";
            when "010" =>
                Y <= "00000100";
            when "011" =>
                Y <= "00001000";
            when "100" =>
                Y <= "00010000";
            when "101" =>
                Y <= "00100000";
            when "110" =>
                Y <= "01000000";
            when "111" =>
                Y <= "10000000";
            when others =>
                Y <= "00000000";
        end case;
    end process;
end Behavioral;
```

Q. Write a VHDL code for a D Flip Flop.



Symbol: D Flip-flop

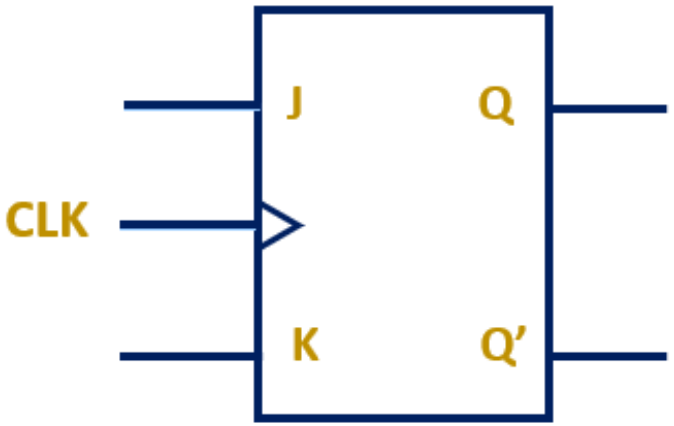
Truth table of D Flip-Flop:

Clock	INPUT	OUTPUT	
	D	Q	Q'
LOW	x	0	1
HIGH	0	0	1
HIGH	1	1	0

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

entity DFF is
    port (
        D, CLK: in std_logic;
        Q: out std_logic
    );
end DFF;

architecture Behavioral of DFF is
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            Q <= D;
        end if;
    end process;
end Behavioral;
```

Q. Write a VHDL code for a JK Flip Flop.

Symbol



Truth Table

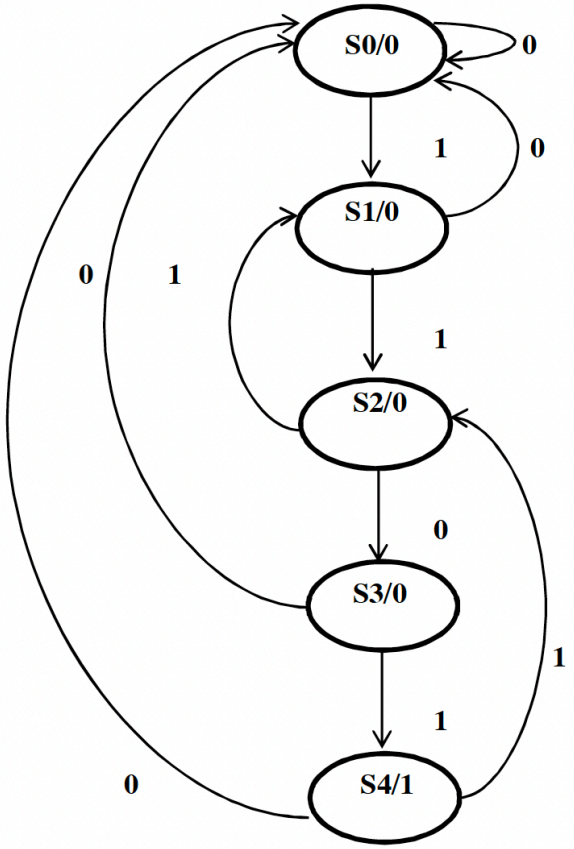
CLK	J	K	Q n+1
↑	0	0	Q n
↑	0	1	0
↑	1	0	1
↑	1	1	Q n'

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

entity JKFF is
    port (
        J, K, CLK: in std_logic;
        Q, Qbar: out std_logic
    );
end JKFF;
```

```vhdl
architecture Behavioral of JKFF is
    signal temp: std_logic;
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            if J = '1' and K = '0' then
                temp <= '1';
            elsif J = '0' and K = '1' then
                temp <= '0';
            elsif J = '1' and K = '1' then
                temp <= not temp;
            end if;
        end if;
    end process;

    Q <= temp;
    Qbar <= not temp;
end Behavioral;
```

Q. Design a sequence detector which detects a “1” when the sequence “1101” is detected. Draw a state diagram and state table. Also write a VHDL code.

Current State(CS)	*Next State(NS) X=0*	*Next State(NS) X=1*	*Output*
S0	S0	S1	0
S1	S0	S2	0
S2	S3	S1	0
S3	S0	S4	0
S4	S0	S2	1

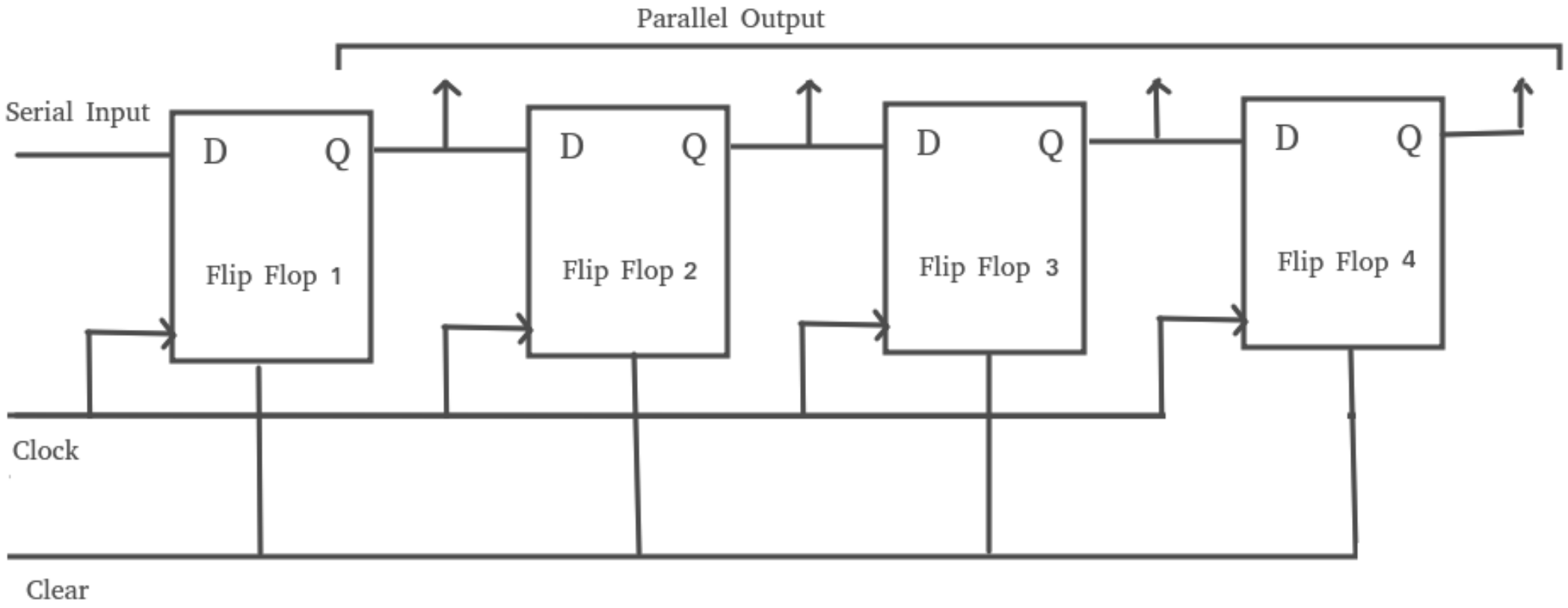


State Diagram

```vhdl
-- Sequence Detector FSM
library ieee;
use ieee.std_logic_1164.all;

entity SequenceDetector is
    port (
        input : in std_logic;
        output : out std_logic
    );
end entity;

architecture Behavioral of SequenceDetector is
    type state_type is (S0, S1, S2, S3, S4);
    signal current_state, next_state : state_type;
begin
    process(current_state, input)
    begin
        case current_state is
            when S0 =>
                if input = '0' then
                    next_state <= S0;
                    output <= '0';
                elsif input = '1' then
                    next_state <= S1;
                    output <= '0';
                end if;
            when S1 =>
                if input = '0' then
                    next_state <= S0;
                    output <= '0';
                elsif input = '1' then
                    next_state <= S2;
                    output <= '0';
                end if;
            when S2 =>
                if input = '0' then
                    next_state <= S3;
                    output <= '0';
                elsif input = '1' then
                    next_state <= S1;
                    output <= '0';
                end if;
            when S3 =>
                if input = '1' then
                    next_state <= S4;
                    output <= '0';
                elsif input = '0' then
                    next_state <= S0;
                    output <= '0';
                end if;
```

```
            when S4 =>
                if input = '1' then
                    next_state <= S2;
                    output <= '1';
                elsif input = '0' then
                    next_state <= S0;
                    output <= '1';
                end if;
        end case;
    end process;

    process(next_state)
    begin
        current_state <= next_state;
    end process;
end architecture;
```

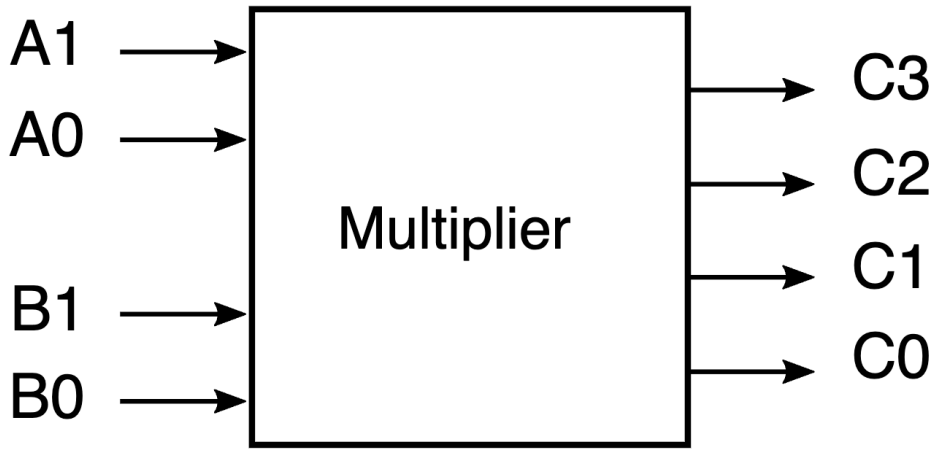
Q. Write VHDL code for 4-bit SIPO Shift register.



```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity ShiftRegister is
    port (
        clk, reset: in std_logic;
        data_in: in std_logic;
        shift: in std_logic;
        data_out: out std_logic_vector(3 downto 0)
    );
end ShiftRegister;
```

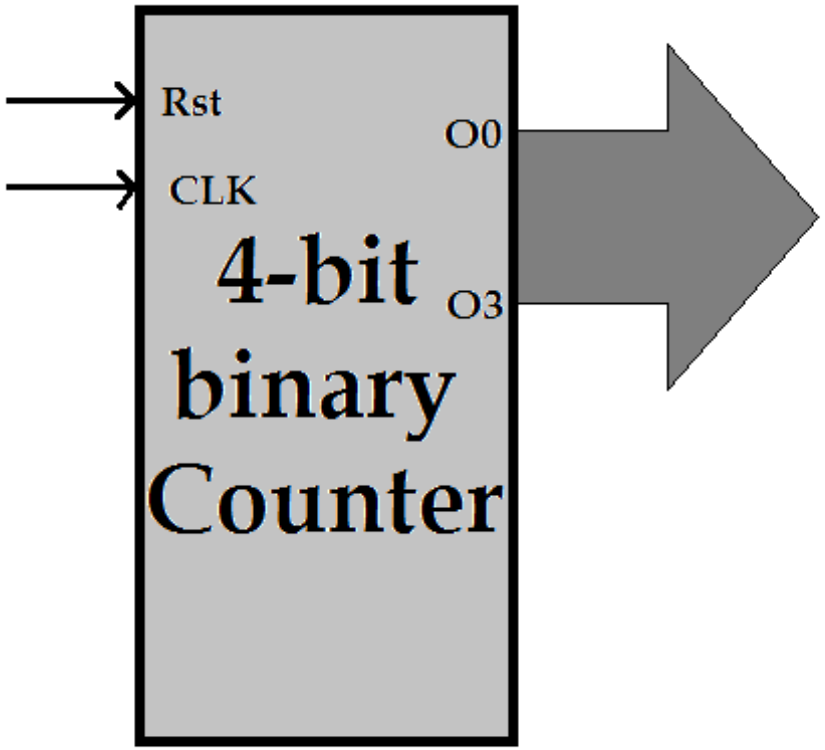
```vhdl
architecture Behavioral of ShiftRegister is
    signal reg: std_logic_vector(3 downto 0);
begin
    process (clk, reset)
    begin
        if reset = '1' then
            reg <= (others => '0');
        elsif rising_edge(clk) then
            if shift = '1' then
                reg <= data_in & reg(3 downto 1);
            end if;
        end if;
    end process;
    data_out <= reg;
end Behavioral;
```

Q. Write VHDL code for Multiplier.



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Multiplier is
    port (
        A, B: in std_logic_vector(1 downto 0);
        Result: out std_logic_vector(3 downto 0)
    );
end Multiplier;
architecture Behavioral of Multiplier is
begin
    process (A, B)
    begin
        Result <= std_logic_vector(unsigned(A) * unsigned(B));
    end process;
end Behavioral;
```

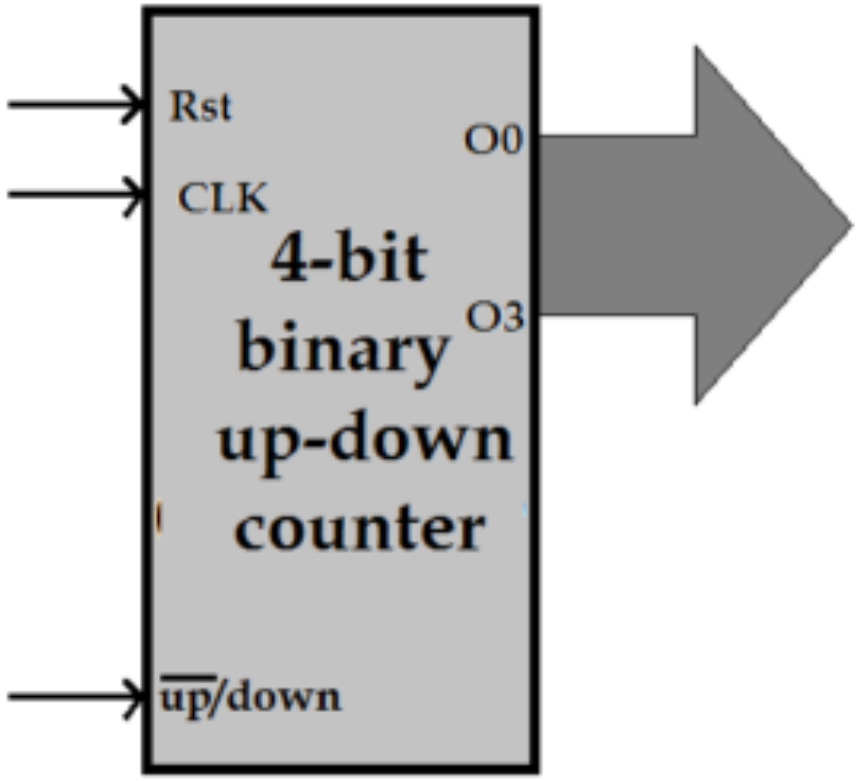
Q. Write a VHDL program a VHDL program to build a 4-bit binary counter.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter is
    Port ( rst, clk : in std_logic;
           o : out std_logic_vector(0 to 3));
end counter;
```

```vhdl
architecture count_arch of counter is
    signal count : std_logic_vector(0 to 3);
begin
    process (clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                count <= "0000";
            else
                count <= count + 1;
            end if;
        end if;
    end process;
    o <= count;
end count_arch;
```

Q. Write a VHDL program a VHDL program to build a up/ down 4-bit binary counter.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter is
    Port (
        rst, clk, up_dwn : in std_logic;
        o : out std_logic_vector(0 to 3)
    );
end counter;

architecture count_arch of counter is
    signal count : std_logic_vector(0 to 3);
begin
    process (clk, rst)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                count <= "0000";
            else
                if up_dwn = '1' then
                    count <= count - 1;
                else
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    o <= count;
end count_arch;
```

Q. Write an algorithm and VHDL code for a custom processor that calculates Least Common Multiple (LCM) of two numbers.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity LCMCalculator is
    port (
        N1, N2: in unsigned(15 downto 0);
        LCM: out unsigned(31 downto 0)
    );
end LCMCalculator;
```

```
architecture Behavioral of LCMCalculator is
    signal A, B, GCD: unsigned(15 downto 0);
begin
    A <= N1;
    B <= N2;
    -- Calculate GCD
    while B /= (others => '0') loop
        GCD <= B;
        B <= A mod B;
        A <= GCD;
    end loop;
    -- Calculate LCM
    LCM <= (N1 * N2) / GCD;
end Behavioral;
```