1. Prezentați structura unui program principal VHDL comentând liniile generice de cod cu explicațiile corespunzătoare. Includeți în structura generică a programului tot ceea ce se poate proiecta prin VHDL.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

-- Declararea entității principale

entity Main is

Port ( input1 : in STD\_LOGIC;

input2 : in STD\_LOGIC;

output : out STD\_LOGIC);

end Main;

-- Declarații pentru arhitectura entității

architecture Behavioral of Main is

signal internal\_signal : STD\_LOGIC;

begin

-- Proces principal

process (input1, input2)

begin

-- Logica principală

internal\_signal <= input1 AND input2;

output <= internal\_signal;

end process;

end Behavioral;

library IEEE;: Include biblioteca IEEE pentru utilizarea standardelor VHDL.

entity Main is: Declară entitatea principală a programului.

Port ( input1 : in STD\_LOGIC; ... );: Specifică porturile entității, precizând tipul și direcția acestora.

architecture Behavioral of Main is: Specifică arhitectura entității, în acest caz, o arhitectură comportamentală.

signal internal\_signal : STD\_LOGIC;: Declarează un semnal intern pentru utilizare în cadrul procesului.

process (input1, input2) ... end process;: Definește procesul principal, care conține logica principală a programului.

internal\_signal <= input1 AND input2;: Realizează o operație logică pe intrările input1 și input2.

output <= internal\_signal;: Atribuie rezultatul operației logice la portul de ieșire

1. Prezentați structura unui program de test VHDL comentând liniile generice de cod cu explicațiile corespunzătoare. Includeți în structura generică a programului tot ceea ce se poate face prin programul de test.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

-- Declararea entității pentru test

entity TestBench is

end TestBench;

-- Declarații pentru arhitectura entității de test

architecture Behavior of TestBench is

signal test\_input1 : STD\_LOGIC := '0';

signal test\_input2 : STD\_LOGIC := '1';

signal test\_output : STD\_LOGIC;

begin

-- Instanțierea entității principale

Main\_inst: entity work.Main

Port map (input1 => test\_input1, input2 => test\_input2, output => test\_output);

-- Proces de generare a stimulilor de test

process

begin

-- Setarea unor valori de test pentru intrări

test\_input1 <= '0';

test\_input2 <= '1';

-- Așteptare pentru observarea rezultatelor

wait for 10 ns;

-- Modificarea valorilor de test

test\_input1 <= '1';

test\_input2 <= '0';

wait;

end process;

end Behavior;

1. Prezentați pe scurt ce presupune scrierea unui program VHDL pentru proiectarea unei unități de comandă bazată pe un automat cu stări de tip Mealy.

Proiectarea unei unități de comandă în limbajul de programare VHDL, care utilizează un automat cu stări de tip Mealy, presupune definirea și gestionarea stărilor sistemului, precum și logica asociată semnalelor de intrare și ieșire. Principalele componente ale unui automat cu stări de tip Mealy includ:

-Stări: Specificarea stărilor posibile ale sistemului, de exemplu, StareA, StareB, și StareC.

-Semnale de intrare: Definirea semnalelor care influențează tranzițiile între stări, cum ar fi semnalul de intrare `input\_signal`.

- Semnale de ieșire: Precizarea semnalelor generate în funcție de starea curentă și semnalele de intrare, de exemplu, semnalul de ieșire `output\_signal`.

În cadrul procesului principal al arhitecturii VHDL, se utilizează instrucțiunea `case` pentru gestionarea tranzițiilor între stări în funcție de starea curentă și semnalul de intrare. De exemplu, în StareA, dacă `input\_signal` este '1', atunci `output\_logic` devine '1', și starea următoare devine StareB; în caz contrar, `output\_logic` devine '0', iar starea următoare devine StareC.

Pentru celelalte stări, se implementează logica specifică pentru fiecare stare, stabilind condițiile de tranziție și valorile semnalelor de ieșire asociate.

1. Realizați o scurtă prezentare a limbajului de dezvoltare hardware Chisel incorporat în Scala.

Chisel definit ca și Constructing Hardware in a Scala Embedded Language este un limbaj de descriere hardware în Scala. Chisel oferă abordarea funcțională și orientată pe obiect a limbajului Scala pentru a construi și genera descrieri hardware eficiente. Caracteristicile principale sunt:

-Scala Integration: Chisel beneficiază de toate caracteristicile limbajului Scala, inclusiv tipuri de date, funcții, și modularitate.

-Constructe Hardware: Chisel oferă un set de constructe de nivel înalt pentru proiectarea hardware-ului, inclusiv registre, multiplexoare, și flip-flop-uri.

-Generatori Parametrizați: Chisel permite definirea de generatoare de hardware parametrizate, ceea ce facilitează crearea de componente hardware reutilizabile.

-Simulare și Testare: Chisel oferă un mediu de simulare și testare integrat în Scala, facilitând dezvoltarea și validarea modelelor hardware.