Circuite Logice Programabile

LABORATOR 1

INTRODUCERE ÎN MEDIUL INTEGRAT XILINX ISE. PROIECTAREA UNUI SUMATOR PE UN BIT

INTRODUCERE

Softwarele CAD (Computer Aided Design) de proiectare cu circuite logice programabile (FPGA) permit în prima etapă alegerea unei metode de descriere a proiectului: captură schematică, descriere VHDL sau diagrame de stare, în următoarea etapă se poate face o simulare funcțională în urma operației de sinteză şi de timp în urma operației de translatare. În cele din urmă, dacă specificațiile de bază sunt întrunite se poate face implementarea şi testarea proiectului în hardware, vezi diagramele de mai jos.

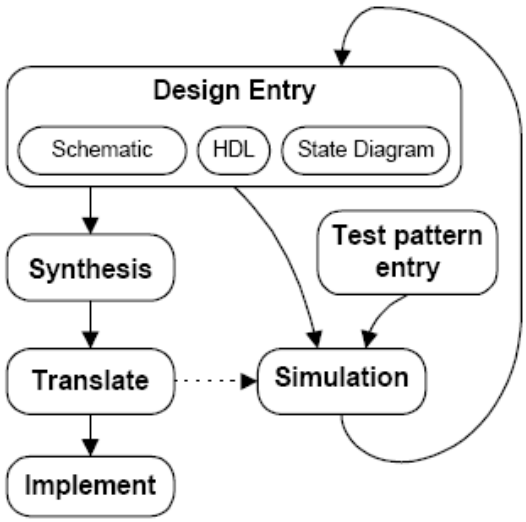


Diagrama I

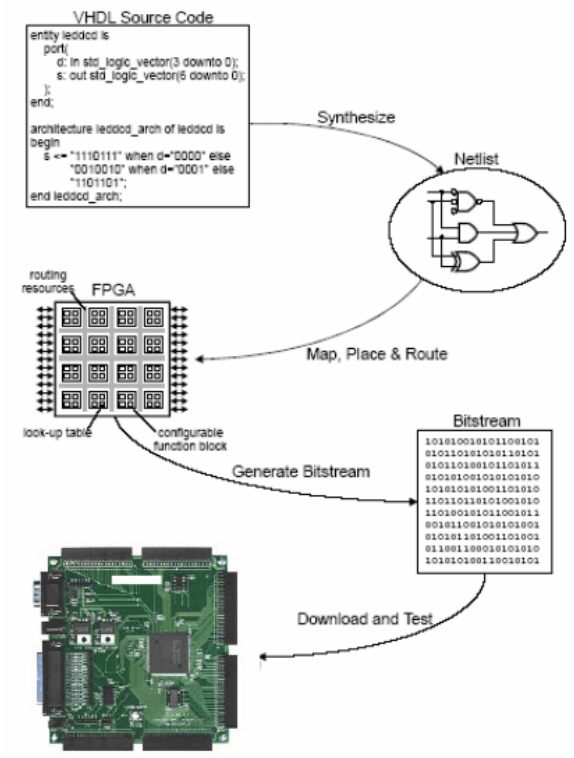


Diagrama II

SCOPUL LUCRĂRII

ISE – Integrated Software Environment – este un mediu integrat care permite descrierea şi implementarea unei multitudini de blocuri logice. Implementarea se face cu FPGA Spartan sau Virtex sau cu CPLD-uri din seria 9500. Circuitele integrate de tip FPGA sau CPLD sunt, de asemenea fabricate de compania Xilinx. La nivel didactic, pentru testarea corectitudinii blocurilor logice proiectate, dispunem de o placa de test D2SB, figura 1, produsă de firma Digilent, bazată pe modulul FPGA de tip Spartan II cu codul XC2S200E.

Pentru a putea exploata mai eficient resursele plăcii D2SB la aceatsa se conectea ctează placa cu circuite periferice DIO4, figura 2. Placa DIO4 este produsă de firma Digilent şi conţine majoritatea dispozitivelor de intrare/ieşire date prezente în orice sistem digital. Placa de dezvoltare este dotată cu comutatoare, taste, LED-uri şi afişoare 7 segmente cu LED-uri.

Figura 1. Modulul de dezvoltare D2SB

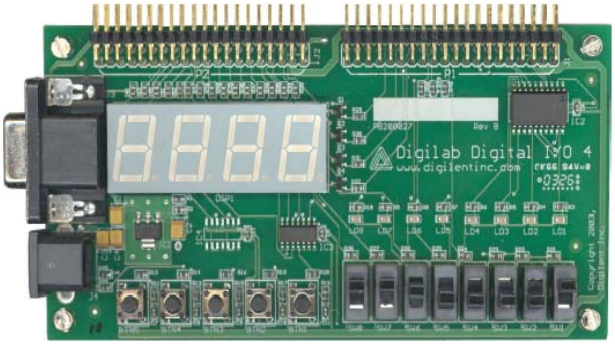


Figura 2. Placa cu dispozitive de intrare/ieşire DIO4

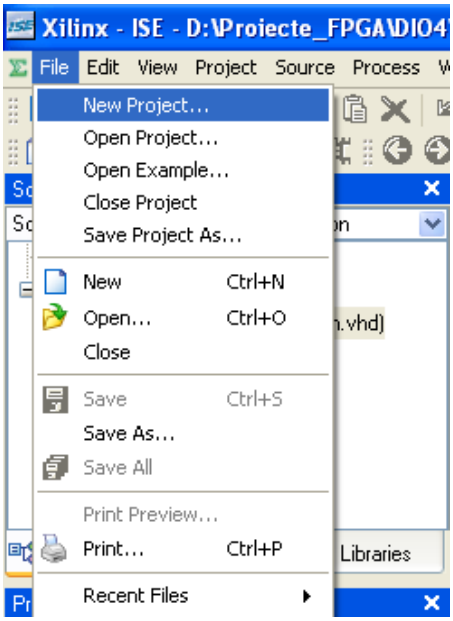
Scopul lucrării constă în implementarea unui sumator elementar descrisin VHDL, pentru familiarizarea studenților cu proiectele de tip HDL.

Desfăşurarea lucrării

Pasul 1: Crearea proiectului.

Se lansează în execuție ISE prin intermediul icoanei 

.

Se creează un nou proiect, 

cu următoarele specificații:

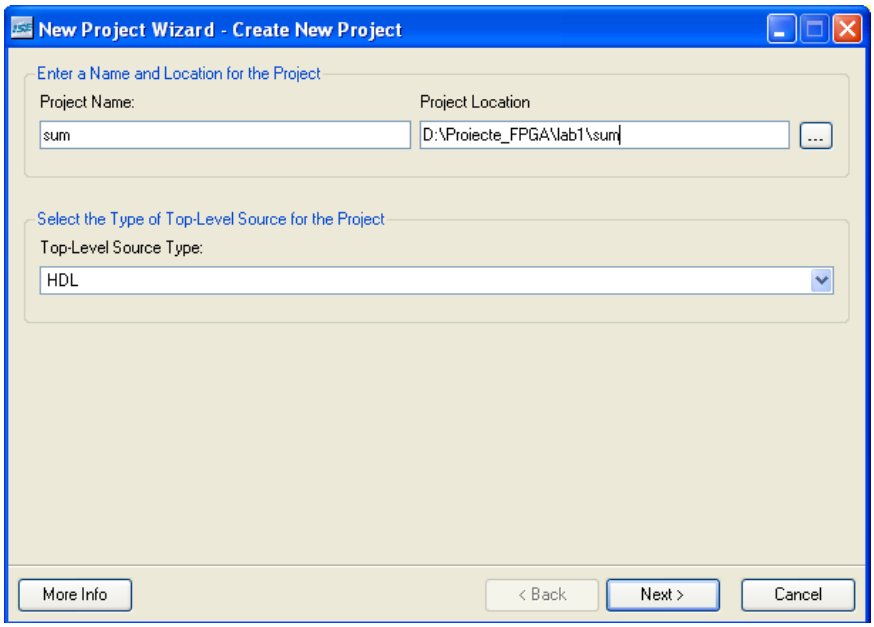


Figura 3

ISE creează câte un folder pentru fiecare proiect. Folderul va avea acelaşi nume ca şi proiectul. Din acest motiv, mai întâi se specifică folderul lab1, iar apoi numele proiectului, sum. După ce ați particularizat şi completat toate informațiile conform figurii 3, apăsați butonul Next. Va apare următoarea fereastră:

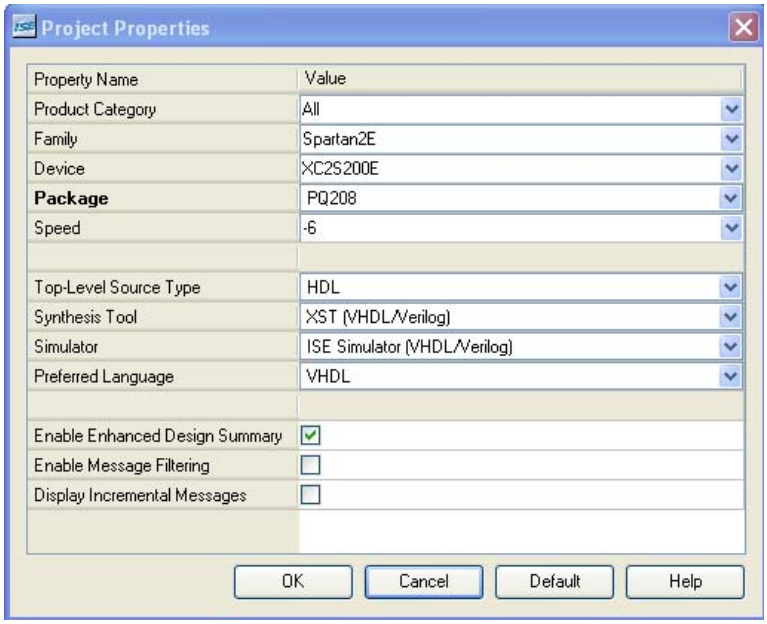


Figure 4

Modulul în jurul căruia este construită placa de dezvoltare D2SB aparține familiei Spartan2, are codul XC2S200E (200 însemnă ca modulul conține echivalentul a 200000 de porți logice) şi împachetarea este de tip PQ208.

Ca şi în cazul proiectelor software, un proiect este compus din mai multe fişiere. Funcționalitatea proiectelor hardware se poate specifica fie prin intermediul schemelor logice fie prin intermediul unui limbaj de descriere hardware, cum ar fi VHDL sau Verilog. Aşa cum în cazul unui proiect scris în C există o funcție care se execută prima, şi anume main, în cazul proiectelor hardware rolul lui „main" este jucat de modulul din vârful ierarhiei, şi anume Top Level Module. Descrierea modulul din vârful ierarhiei poate fi de tip schematic sau HDL (HDL=Hardware Description Language = Limbaj de descriere hardware). Pe parcursul acestui laborator se va folosi numai descrierea de tip HDL. Mai mult despre structura ierarhică a proiectelor hardware, în laboratoarele următoare.

În cazul sumatorului de un bit, descrierea funcționalității se face prin intermediul unui singur fişier HDL şi automat aceasta va fi în vârful ierarhiei. Din acest motiv alegem Top Level Module Type de tip HDL.

Celelalte două câmpuri se setează la valorile din figura 4 for fi detaliate în următoarele laboratoare, iar apoi se apasă butonul Next.

Acțiunile aferente următoarelor două ferestre, Create a New Source şi Add Existing Sources, sunt opționale, şi din acest motiv se apasă Next pentru fiecare în parte, fără să se completeze nimic. In final, apare o fereastră de informare, pentru care se apasă Finish. După executarea tuturor acțiunilor descrise mai sus trebuie să se obțină următorul ecran:

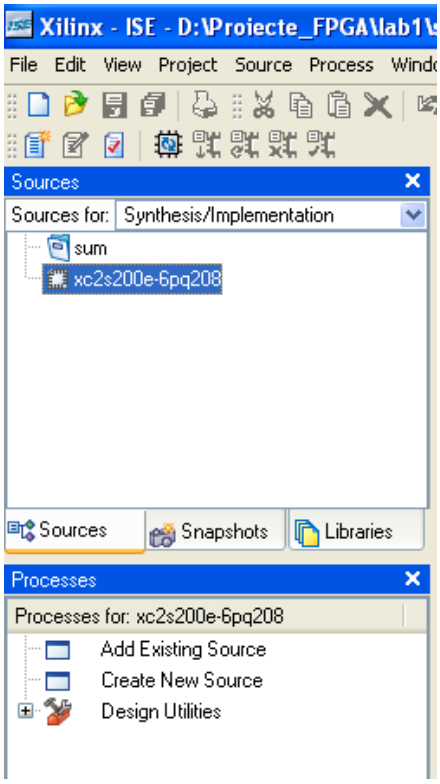


Figura 5

Pasul 2: Crearea fişierului VHDL care va conţine descrierea sumatorului de un bit.

Se face clic dreapta pe numele circuitului (xc2s200e), şi din meniul contextual apărut se selectează New Source. În fereastră New Source, se selectează tipul VHDL Module iar apoi se completează numele fişierului (figura 6). În continuare se apasă Next, apare o fereastră care permite definirea semnalelor prin care modulul descris se conectează cu exteriorul (figura 7)

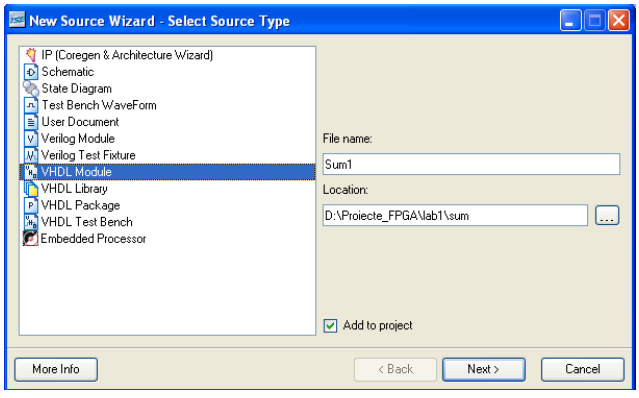


Figura 6

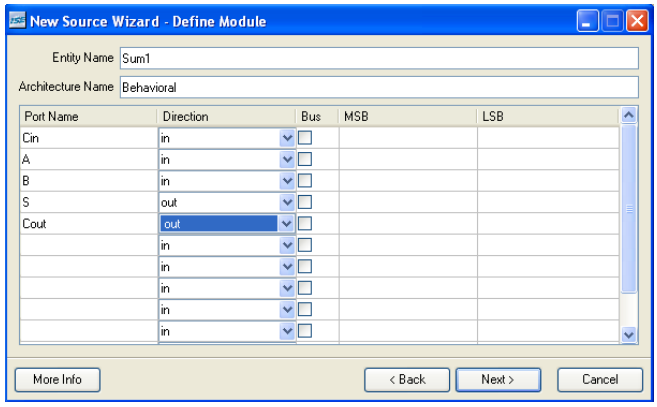


Figura 7

Intrările sumatorului sunt semnalele A, B, şi Cin iar ieşirile S şi Cout. Se completează fereastra din figura 7 cu aceste informaţii, apoi se apasă Next. Ca urmarea, apare o fereastră de informare care sumarizează informaţiile introduse şi permite revenirea în caz că există o greşeală. Dacă totul este corect, se apasă butonul Finish.

Ca efect al apăsării lui Finish, sum.vhd este construit şi adăugat la proiect. După cum se remarcă ISE a creat automat entitatea din descrierea furnizată prin intermediul figurii 7.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Sum_s4 is
    Port ( Cin : in std_logic;
           A : in std_logic;
           B : in std_logic;
           S : out std_logic;
           Cout : out std_logic);
end Sum_s4;
.....
```

În corpul arhitecturii între *begin* şi *end Behavioral* scrieţi expresiile logice S şi Cout.

S <= A xor B xor Cin;

Cout <= (A and B) or (B and Cin) or (A and Cin);

Pasul 3: Crearea fişierului de constrângeri şi specificarea acestora.

Pentru a verifica funcţionarea sumatorului intrărilor A, B şi Cin li se asignează pini FPGA care în exterior sunt conectaţi la comutatoare iar ieşirilor S şi Cout li se asignează pini FPGA conectate la LEDuri. Această asignare se numeşte constrângere. Există mai multe tipuri de constrângeri, aceasta fiind una dintre ele.

Din documentaţiile plăcilor D2SB şi DIO4 rezultă următorul tabel:

RESURSE D2SB ŞI DIO 4	PINI FPGA
"MCLK"	"P182";
"D2SB_BTN"	"P187";
"D2SB_LED"	"P154";
"DIO4_LEDG"	"P45";
"DIO4_LED<0>"	"P111";
"DIO4_LED<1>"	"P109";
"DIO4_LED<2>"	"P102";
"DIO4_LED<3>"	"P100";
"DIO4_LED<4>"	"P98";
"DIO4_LED<5>"	"P96";

"DIO4_LED<6>"	"P94";
"DIO4_LED<7>"	"P89";
"DIO4_BTN<0>"	"P3";
"DIO4_BTN<1>"	"P206";
"DIO4_BTN<2>"	"P44";
"DIO4_BTN<3>"	"P43";
"DIO4_BTN<4>"	"P42";
"DIO4_SW<0>"	"P23";
"DIO4_SW<1>"	"P21";
"DIO4_SW<2>"	"P18";
"DIO4_SW<3>"	"P16";
"DIO4_SW<4>"	"P11";
"DIO4_SW<5>"	"P9";
"DIO4_SW<6>"	"P7";
"DIO4_SW<7>"	"P5";
"DIO4_AN<0>"	"P41";
"DIO4_AN<1>"	"P40";
"DIO4_AN<2>"	"P36";
"DIO4_AN<3>"	"P35";
"DIO4_SSG<0>"	"P22";
"DIO4_SSG<1>"	"P20";
"DIO4_SSG<2>"	"P17";
"DIO4_SSG<3>"	"P15";
"DIO4_SSG<4>"	"P10";
"DIO4_SSG<5>"	"P8";
"DIO4_SSG<6>"	"P6";
"DIO4_SSGDP"	"P4";

Astfel se vor atribui următorii pini FPGA pentru A,B,Cin, S şi Cout: Cin – P23 (SW1), A – P21 (SW2), B – P18 (SW3), conform tabelului anterior. Lui S i se asignează P111 (LD1), iar lui Cout P109 (LD2).

Mai întâi se creează fişierul de constrângeri. Se face clic dreapta pe numele circuitului (xc2s200e) sau pe numele fişierului VHDL sum1.vhd şi din meniul contextual apărut se selectează New Source. În fereastră New Source, se selectează tipul Implementation Constraints File iar apoi se completează numele fişierului. Fie numele acestui fişier sum1cf. Se apasă Next, în următoarea fereastră Next, iar apoi în fereastra de informare se apasă Finish. Dacă totul a decurs conform celor explicate anterior se va obține situația din figura 8.

Se observă ca procesele posibile pentru un anumit fişier sunt diferite în funcție de fişierul selectat în fereastra Sources in Project. Dacă în fereastra Sources in Project se selectează sum1cf.ucf, în fereastra Processes for ... apar procesele din figura 8.

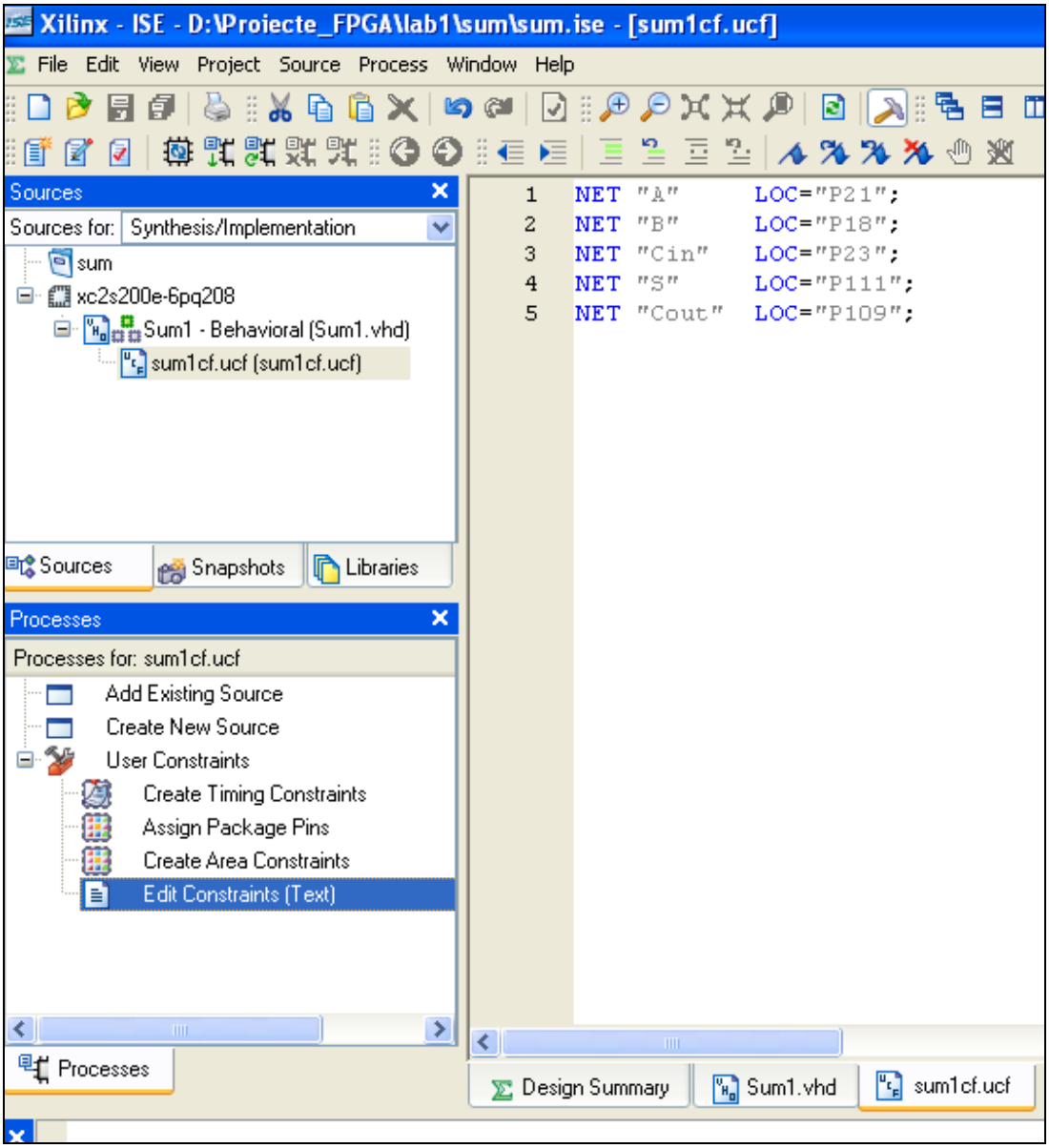


Figura 8

Înainte de a crea fişierul de constrângeri selectaţi Edit -> Preferences -> Editor şi apăsaţi radio butonul Constraints Editor. Pentru crearea/editarea constrângerilor, pornind de la configuraţia din figura 8, se face dublu clic pe procesul Edt Constraints (Text), pentru a lansa în execuţie editorul de constrângeri şi se introduc liniile de configurare din figura 8, după care se salvează.

Pasul 4: Crearea fişierului de configurare şi verificarea funcţionalităţii.

Plecând de la configuraţia din figura 8, se face click pe numele proiectului *Sum1- Behavioral.*. după care în fereastra *Processes* se face dublu clic pe procesul Generate Programming File, vezi figura 9.

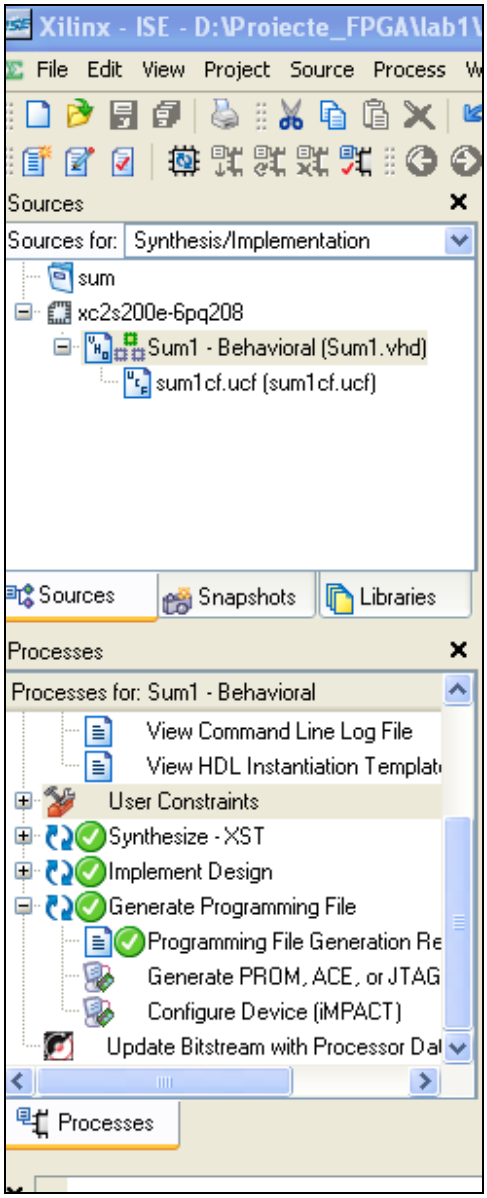


Figura 9

Pasul 5: Configurarea FPGA Spartan 3 de pe placa de dezvoltare

Odată generat fişierul bit sub procesul Generate Programming File faceți dublu clic pe opțiunea Configure Device (iMPACT) aflată sub procesul Generate Programming File. Se va obține fereastra din figura 10, se face clic pe Finish.

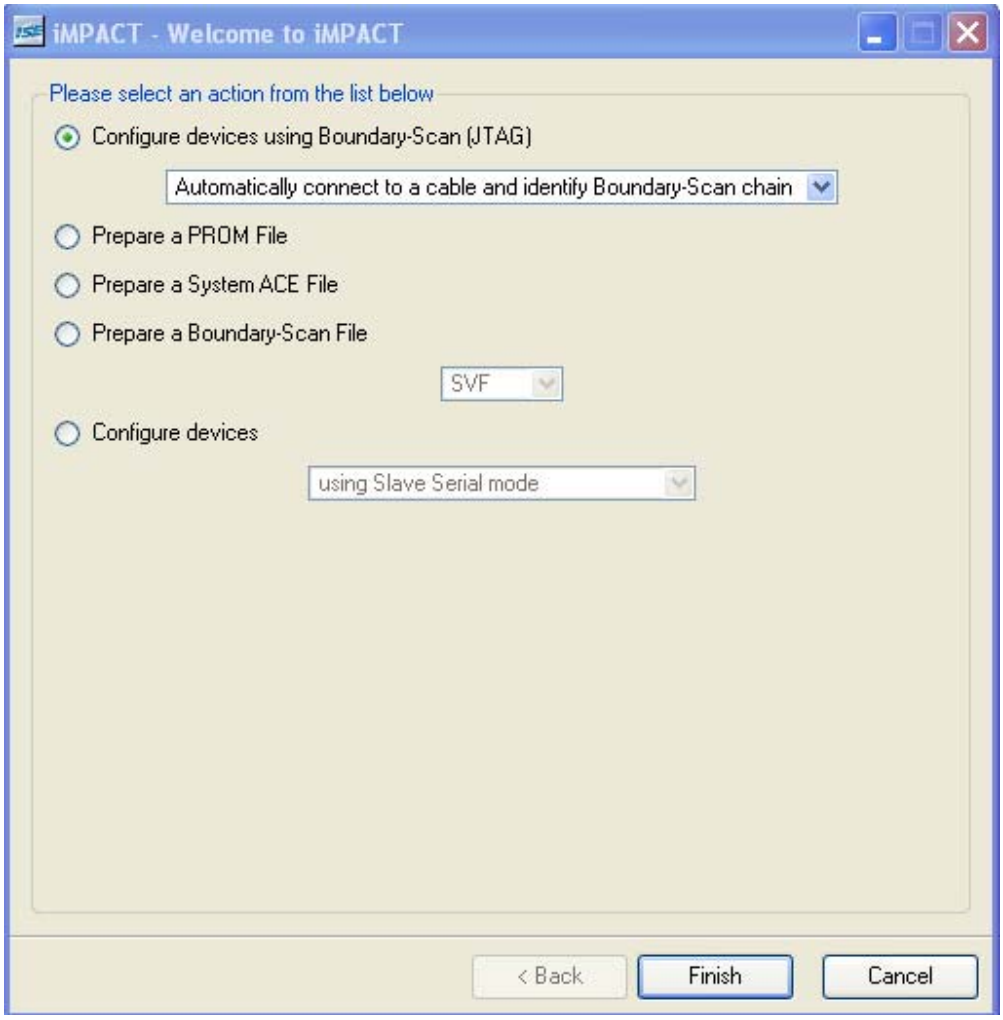


Figura 10

După care se obține figurafereastra din figura 11, după ce se face în care se selectează fişiereul de configurare de tip bit, în cazul nostru *sum1.bit*

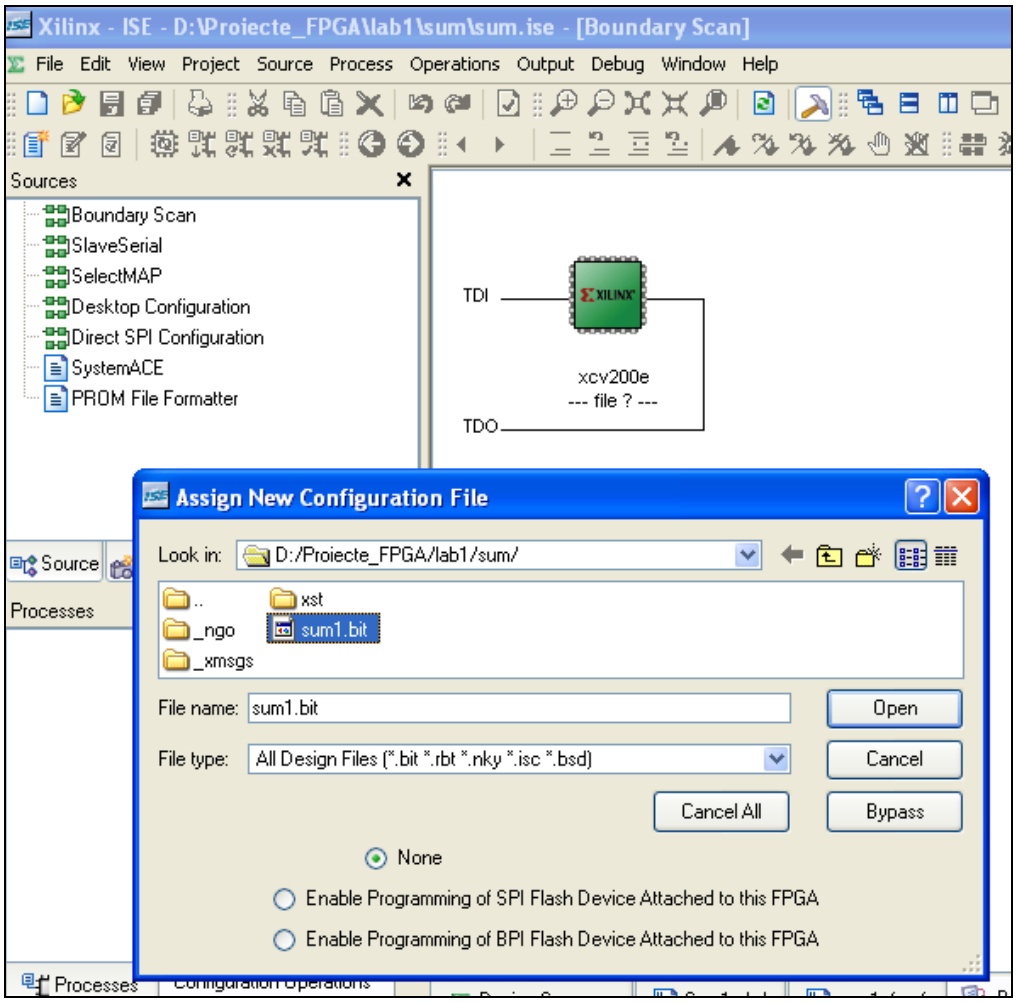
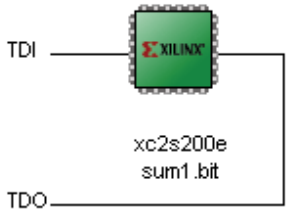


Figura 11



Se face clic dreapta pe dispoyitivul Xilinx şi se alege opțiunea *Program*...în fereastra care apare se face clic pe OK. Apare o fereastră *Progree Dialog* după care ar trebui să se obțină mesajul 

în caz contrar, chemați profesorul!!!

Pasul 6: (opțional, pentru notă mai mare ca 5)

1. Pentru ca datele sa fie transferate imediat de la intrare la ieşire este necesar ca latchurile prezente la fiecare LED să primească semnal de validare, astfel conform documentației plăcii D2SB se recomandă ca semnalul Ledg să fie ținut în 1 logic. Pinul FPGA aferent acestui semnal este P45.
   Introduceți linia corespunzătoare în dul VHDL şi în fişierul de constrângeri UCF, astfel încât afişarea rezultatului la LED-uri să aibă loc imediat ce au fost modificate valorile de la intrarea sumatorului.

2. Modificați proiectul existent astfel încât să se implementeze un sumator pe doi biți.
   Intrările vor fi:
   • Cin,
   • A1, A0 primul operand, A0 LSB
   • B1, B0 al doilea operand, B0 fiind LSB

   Ieşirile vor fi:
   • S1, S0 suma, S0 LSB
   • Cout

   Intrările şi ieşirile se vor conecta la comutatoare şi Leduri astfel:
   Cin – SW0, A0-SW1, A1-SW2, B0-SW3, B1-SW4, S0-LD0, S1-LD1, Cout-LD2

Circuite Logice Programabile

LABORATOR 2

DESCRIEREA IN SCHEMATIC, MODALITATI DE REALIZARE A UNUI TESTBENCH, SIMULAREA FUNCȚIONALĂ

SCOPUL LUCRĂRII

În această lucrare se urmăreşte folosirea modalităminunții de descriere a unui proiect utilizând simbolurile schematice. De asemenea va fi prezentată şi modalitatea de realizare a unui testbench, precum şi modalități de simulare folosind Simulatorul mediului ISE. Pentru exemplificarea tehnicilor enumerate anterior se va face proiectarea unui decodificator binar - 7 segmente.

IINTRODUCERE TEORETICĂ

În tabelul 1 este prezentat tabelul de adevăr pentru decodificatorul binar 7 segmente.

Tabel1

Intrările				Numărul zecimal	Afişajul	seg *a*	seg *b*	seg *c*	seg *d*	seg *e*	seg *f*	seg *g*
I_3	I_2	I_1	I_0									
0	0	0	0	0		1	1	1	1	1	1	0
0	0	0	1	1		1	1	0	0	0	0	0
0	0	1	0	2		1	0	1	1	0	1	1
0	0	1	1	3		1	1	1	0	0	1	1
0	1	0	0	4		1	1	0	0	1	0	1
0	1	0	1	5		0	1	1	0	1	1	1
0	1	1	0	6		0	1	1	1	1	1	1
0	1	1	1	7		1	1	0	0	0	1	0
1	0	0	0	8		1	1	1	1	1	1	1
1	0	0	1	9		1	1	0	0	1	1	1
Restul combinatiilor						×	×	×	×	×	×	×

În figura 1 este prezentată schema convențională a decodificatorului 7-segmente.

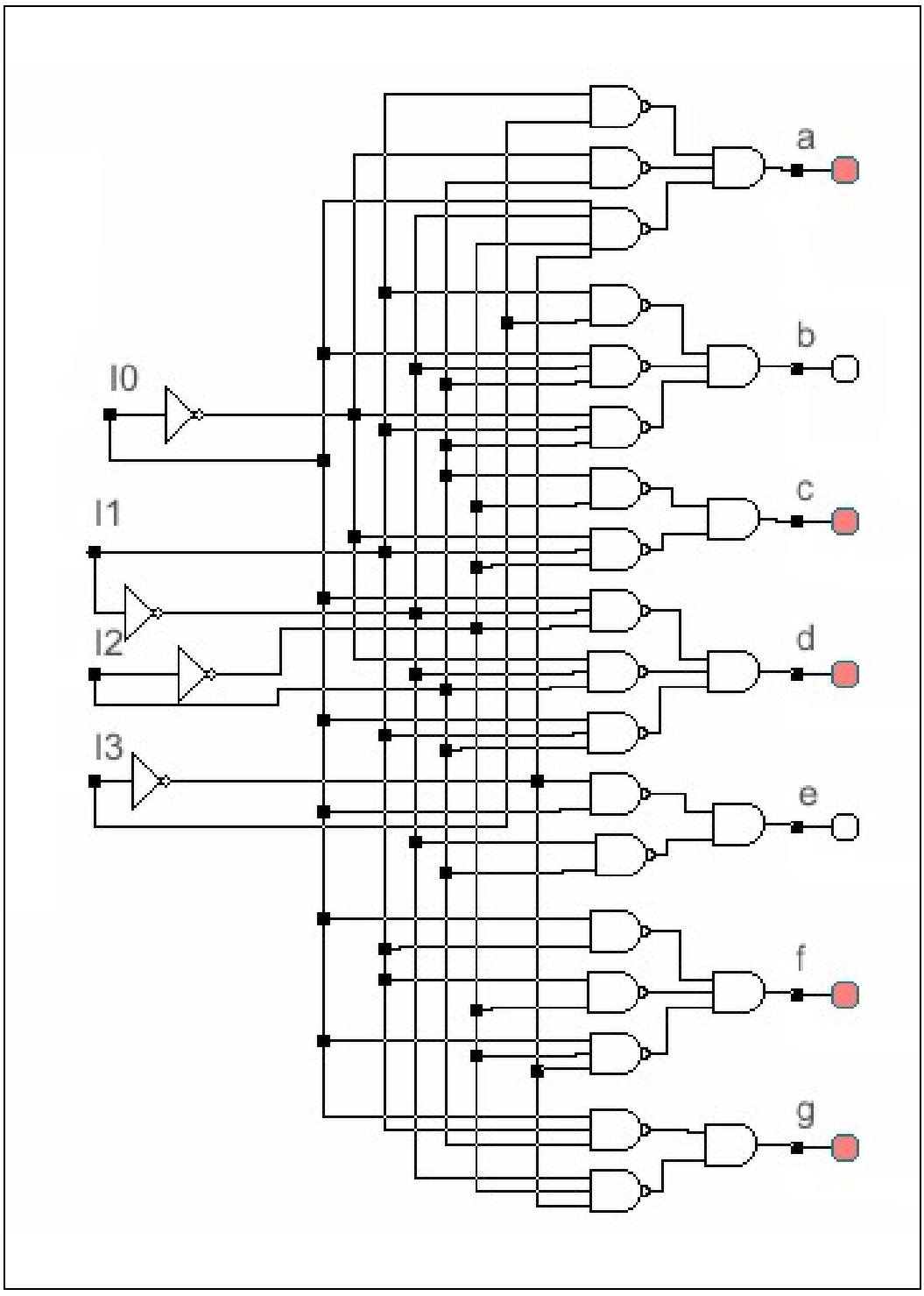


Figura 1

Mai jos este prezentat codul VHDL pentru un decodificator binar 7 segmente.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY BCD2Seven_Segment_Decoder IS
  PORT ( BCD:  IN std_logic_vector(3 DOWNTO 0);
     Segments: OUT std_logic_vector(1 TO 7));
END BCD2Seven_Segment_Decoder;

ARCHITECTURE Behavioral OF BCD2Seven_Segment_Decoder IS
BEGIN
  process(BCD)
  BEGIN
    CASE BCD IS
    WHEN "0000" => Segments <= "1111110";
    WHEN "0001" => Segments <= "1100000";
    WHEN "0010" => Segments <= "1011011";
    WHEN "0011" => Segments <= "1110011";
    WHEN "0100" => Segments <= "1100101";
    WHEN "0101" => Segments <= "0110111";
    WHEN "0110" => Segments <= "0111111";
    WHEN "0111" => Segments <= "1100010";
    WHEN "1000" => Segments <= "1111111";
    WHEN "1001" => Segments <= "1100111";
    WHEN OTHERS => Segments <= "0000000";
    END CASE;
  END PROCESS;
END Behavioral;
```

Desfășurarea lucrării

Pasul 1: Crearea proiectului.

Se va crea un proiect cu numele *dcd7seg,* atenție la directorul de lucru.
Se va selecta ca *Top-Level Source* modul de lucru *Schematic*, vezi figura 2.

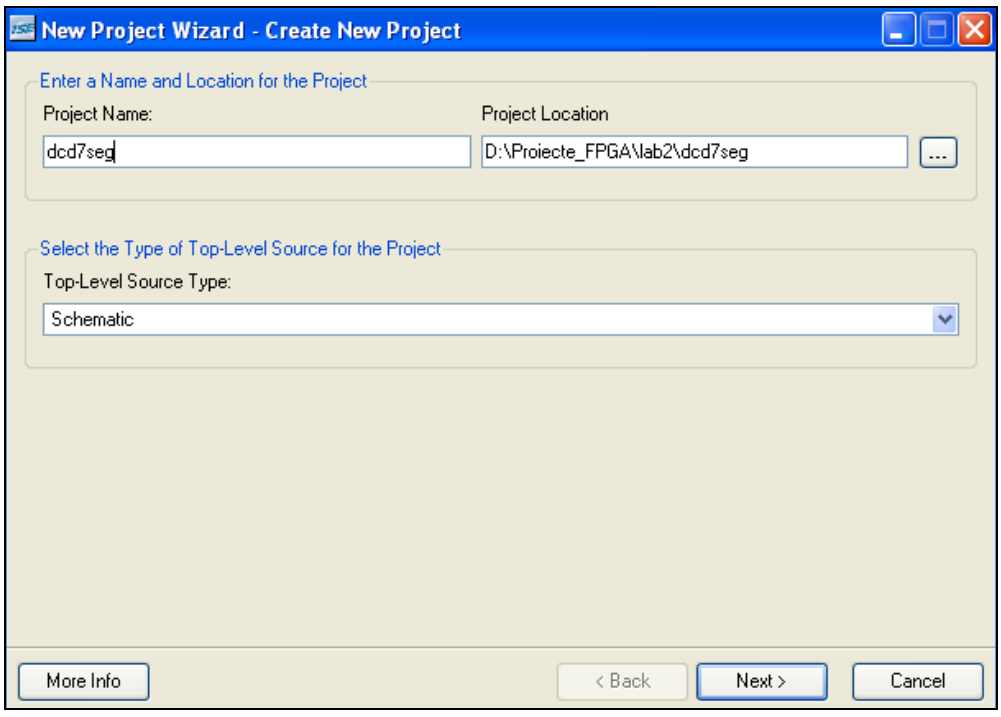


Figura 2.

Se apasă *Next* şi se continuă cu setările din fereastra prezentată în figur 3.

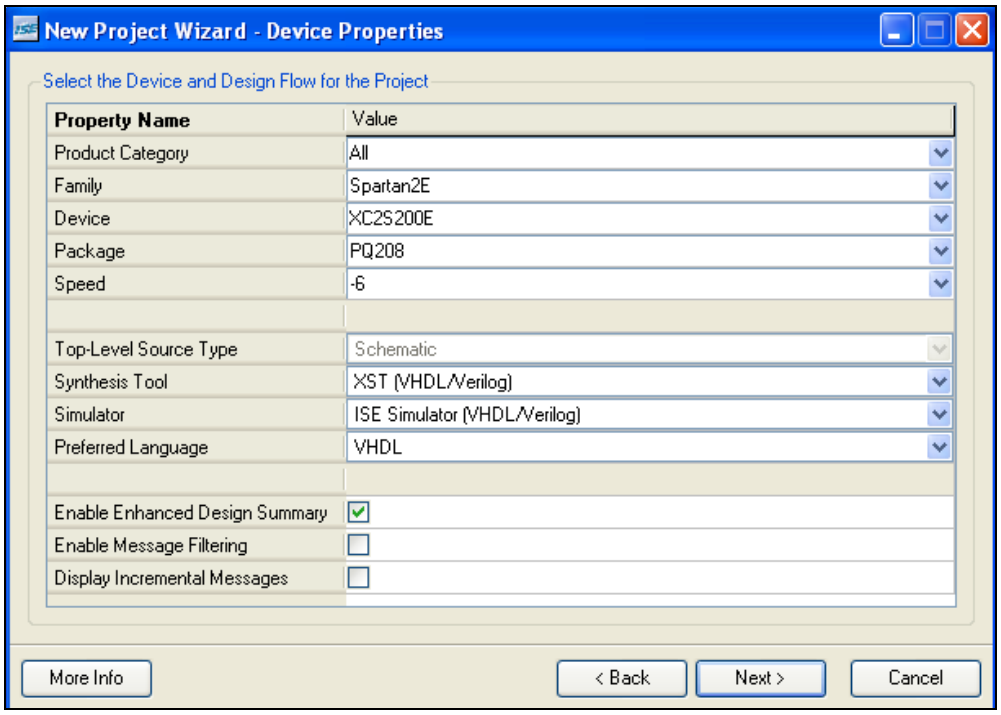


Figura 3

După ce proiectul a fost creeat, conform laboratorului 1 se adaugă un fişier sursă nou, *Project →New Source*, se alege un nume pentru fişierul sursă *dcd7seg* şi se alege să fie de tipul *Schematic*, vezi figura 4.

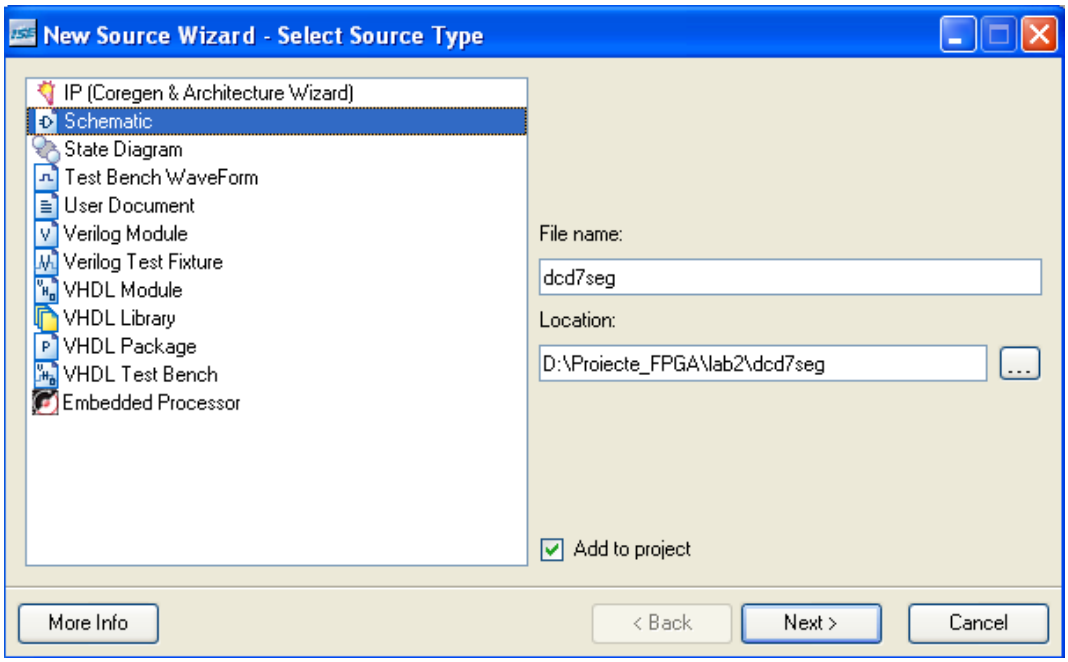


Figura 4

Pasul 2: Proiectarea în schematic

Conform schemei din figura 1 şi a ecuațiilor booleene prezentate anterior se trece la proiectarea în schematic a decodificatorului 7 segmente.

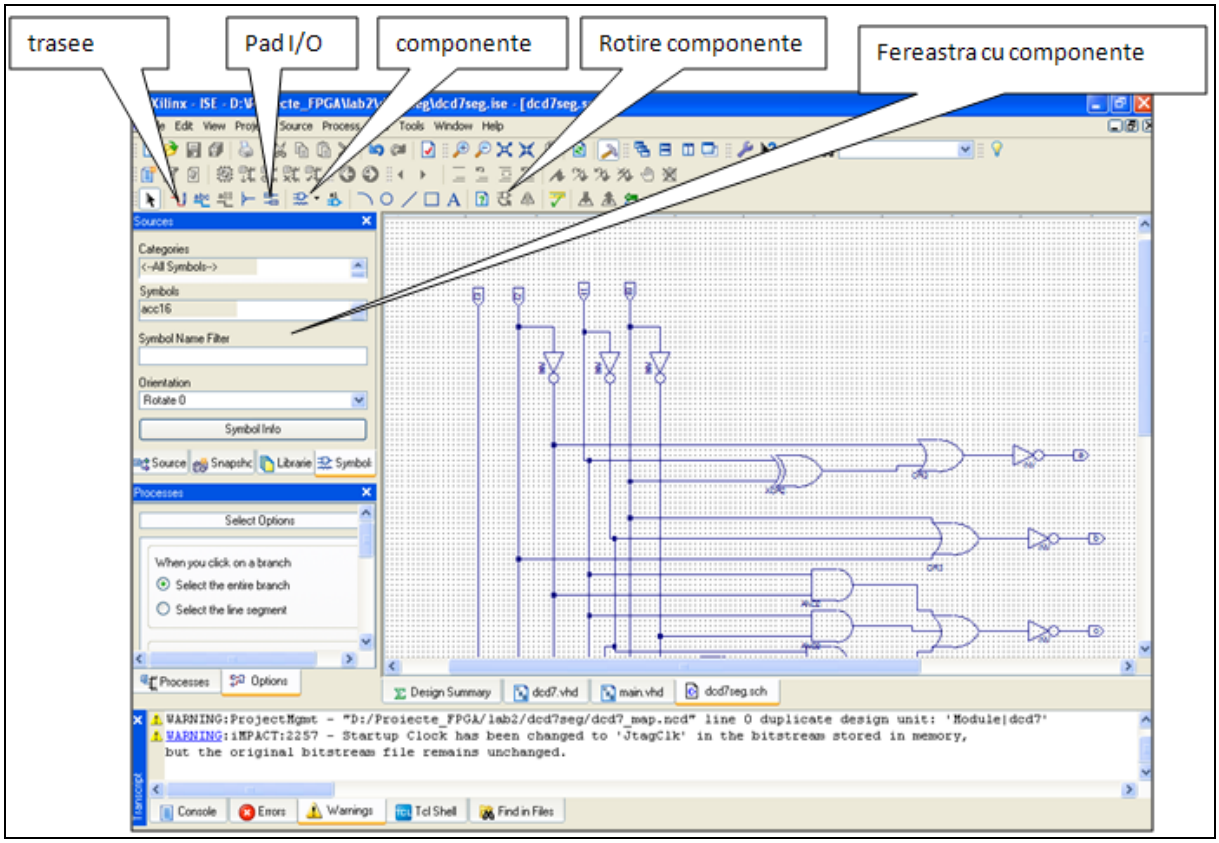


Figura 5

Conform figurii 5, editorul schematic are în partea stângă fereastra *Source* de unde pot fi alese componentele. În câmpul *Symbol Name..* se introduce denumirea prescurtată a componentei, ex. AND, INV, OR, XOR..etc. Odată introdusă denumirea se dă click pe butonul *componente* din bara de meniuri. Componentele pot fi poziționate corespunzător cu ajutorul butonului de rotire, vezi figura 5. Odată aduse toate componentele în zona de lucru, prin intermediul butonului de adăugare *trasee*, se pot conecta între ele. După etapa de conectare urmează etapa de adăugare a *pad*-urilor de intrare/ieşire, acest lucru se realizează prin apăsarea butonului *Pad I/O* din bara de meniuri. Cu dublu click pe pad se poate schimba numele acestora conform figurii 5. Rolul acestor pad-uri este de a realiza conectivitate cu pini circuitului. După terminarea etapei de proiectare în schematic, se salvează proiectul după care editorul poate fi închis.

Pasul 3: Crearea fişierului de constrângeri

În figura 6 este prezentat modul de conectare al afişajului 7 segmente. Se poate observa notarea segmentelor şi modul de conectare împreună a anozilor pentru fiecare caracter al afişajului. Astfel că pentru a fi activat, fiecare segment trebuie pus în 0 logic. Aceasta este

rațiunea pentru care **în schematic trebuie folosite porți inversoare pe fiecare ieşire**. Prin intermediul anozilor fiecare catod poate fi activat individual, vezi detalii despre placa DIO4.

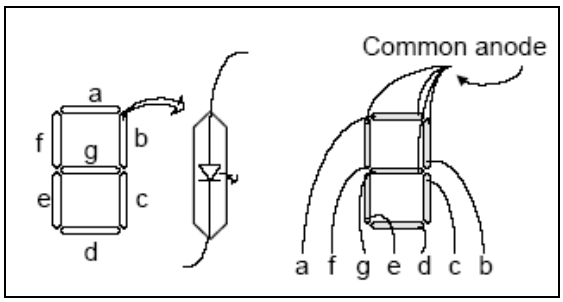


Figura 6.

Asocierea celor şapte segmente şi a celor patru intrări cu comutatoarele şi afişajul de pe placă se face după cum urmează:

a ... g → SSG<0> ... SSG<6>
I0 ... I3 → SW0 ... SW3

În tabelul 1 din laboratorul 1 este indicat pinul corespunzător circuitului FPGA pentru SSG şi SW. Introducerea constrângerilor se poate face fie prin editarea fişierului UCF, conform procedurii din laboratorul 1, fie prin apelare utilitarului *Xilinx PACE,* vezi figura 8.

```
 4  NET "a"  LOC = "P22"  ;
 5  NET "b"  LOC = "P20"  ;
 6  NET "c"  LOC = "P17"  ;
 7  NET "d"  LOC = "P15"  ;
 8  NET "e"  LOC = "P10"  ;
 9  NET "f"  LOC = "P8"  ;
10  NET "g"  LOC = "P6"  ;
11  NET "I0"  LOC = "P23"  ;
12  NET "I1"  LOC = "P21"  ;
13  NET "I2"  LOC = "P18"  ;
14  NET "I3"  LOC = "P16"  ;
```
Figura 7.

Apelare utilitarului menționat anterior se face din fereastra *Processes, User Constraints, Assign Package.* Constrîngerile de LOC ale pinilor se fac conform celor din figura 8.
NU uitați mai înaite trebuie să adaugați fişierul UCF la proiect şi apoi să-l editați.

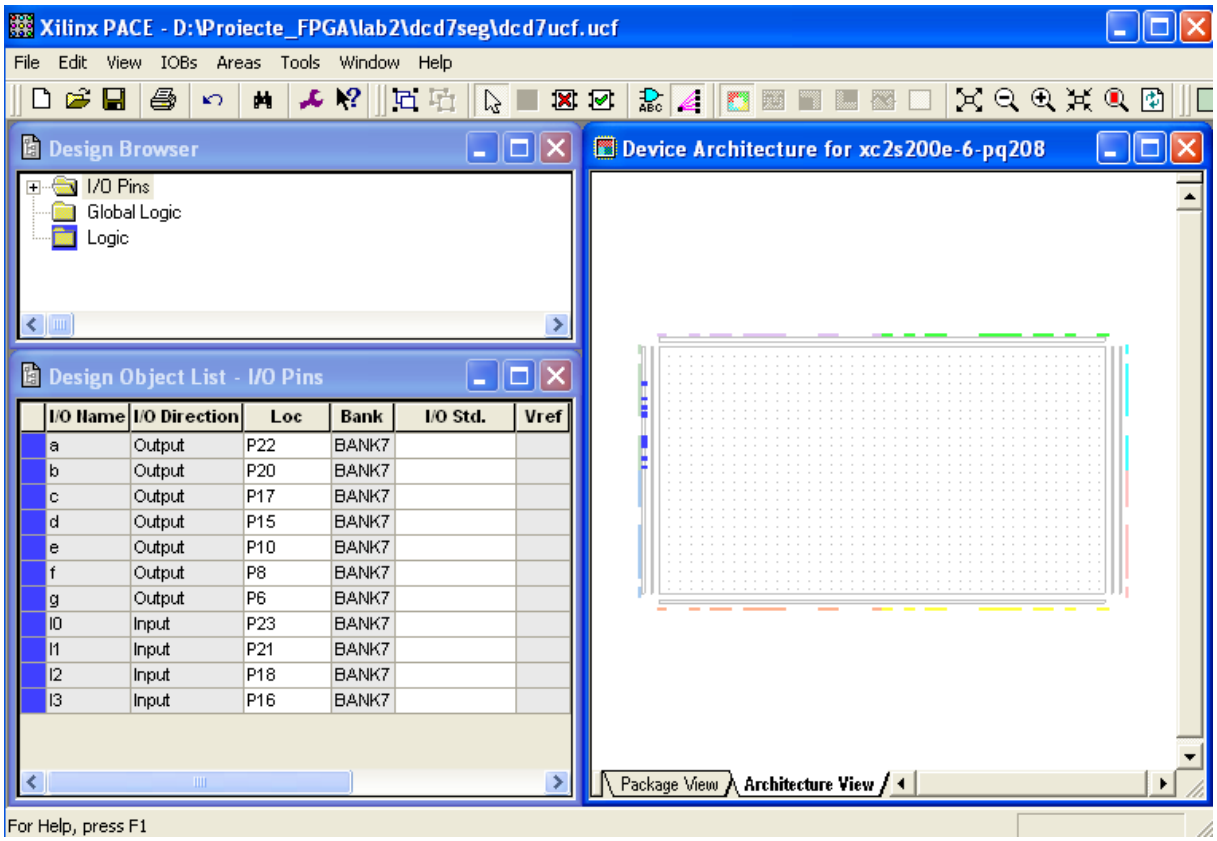


Figura 8.

Pasul 4: Crearea testbench-ului şi simularea funcțională a proiectului

Pentru simularea funcțională a proiectului se creează un aşa numit test bench care va genera stimuli pentru intrarile decodificatorului. Pentru creearea testbenchului se selectează *Project, New Source,* conform figurii 9 numele fişierului sursă va fi dcd7_tb, iar tipul va fi *Testbench Waveform.* Mai de parte să dă click *next, next, finish.*

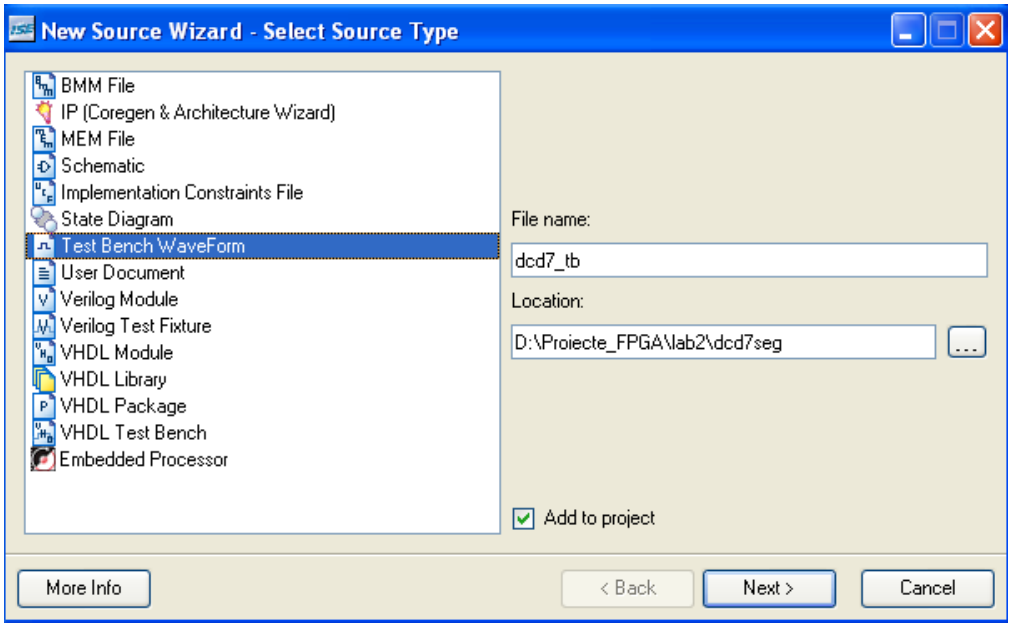


Figura 9

Se va deschide o fereastră ca şi cea din figura 10. Întrucât decodificatorul este combinațional (nu are nevoie de clock) se selectează opțiunea *Combinatorial,* de asemenea în câmpul *Initial Length...* se selectează 2000 ns, aceasta va fi durata simulării, după care se apasă tasta *Finish.*

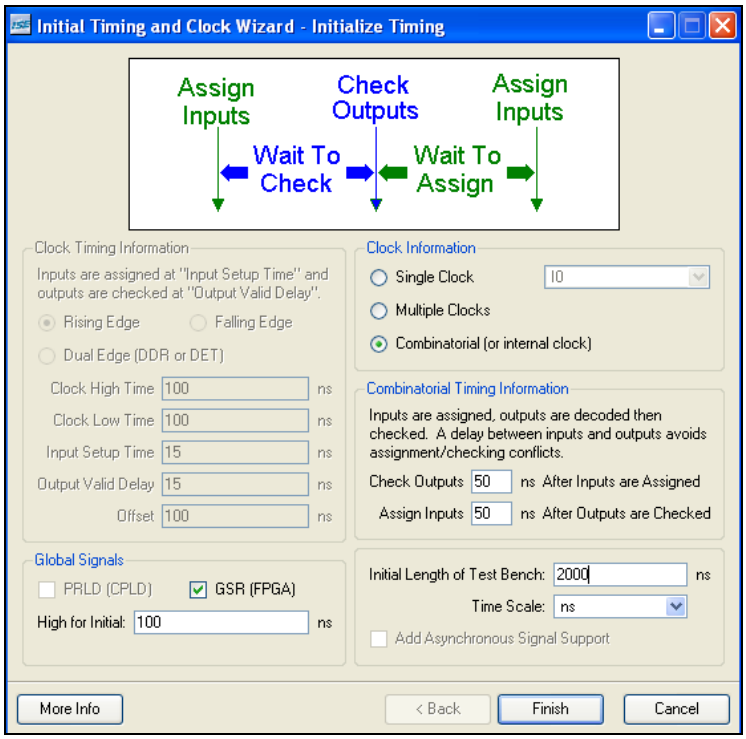


Figura 10.

Se va deschide o fereastră ca şi cea din figura 11. Dacă se dă click în zona albastră din dreptul fiecărui semnal de intrare se poate schimba starea semnalului. Forma semnalului va fi stabilită conform celei din figura 11, astfel vor fi acoperite toate starile pe care le poate avea semnalul de la intrarea decodificatorului.

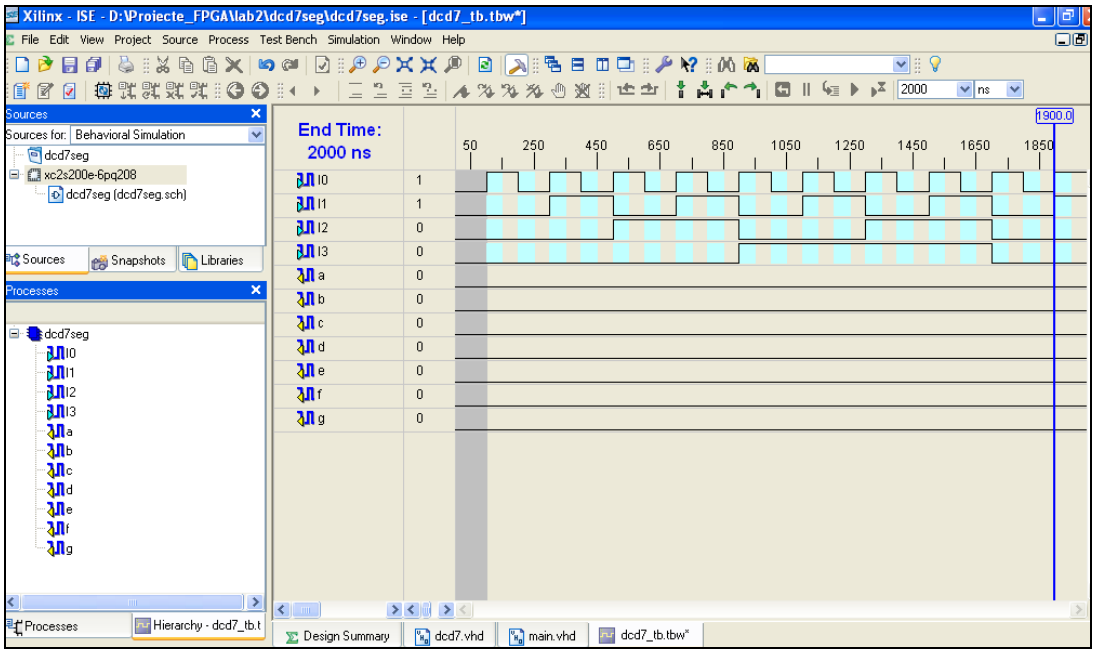


Figura 11.

Se face o salvare a testbenchului după care din fereastra *Source* se selectează *Behavioral Simulation,* vezi figura 12.

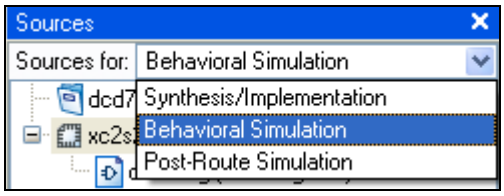


Figura 12.

Pentru simularea funcțională a proiectului se verifică ca în fereastra *Source* să fie selectate *Behavioral Simulation* şi *dcd7_tb.tbw*, după care din fereastra *Processes* cu click pe + se expandează şi se face dublu click pe *Simulate Behavioral Model.* Simulatorul ISE va porni automat şi va rula pînă la sfârşitul prioadei alese 2000 ps, fereastra rezultată trebuie să fie ca şi cea din figura 13.

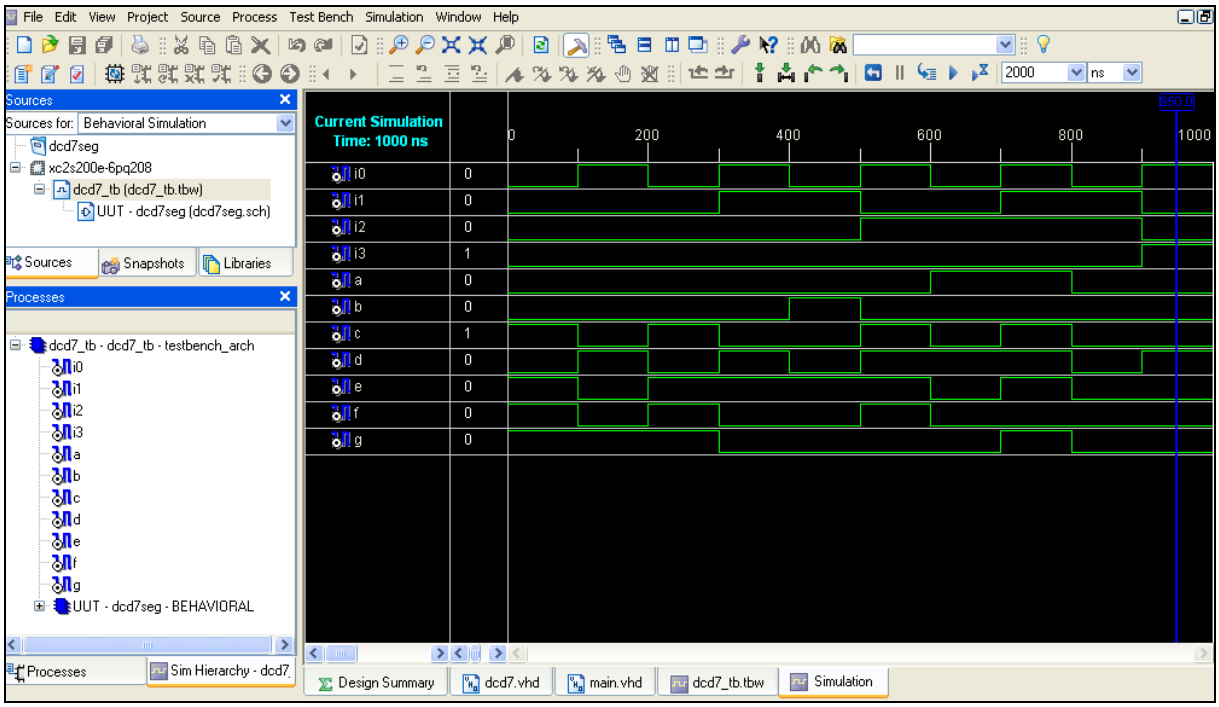


Figura 13.

Verificați ca formele de undă să corespundă cu tabelul de adevăr prezentat în partea teoretică a laboratorului.

Pasul 5: Implementarea şi testarea proiectului

Implementarea se realizează conform procedurii din laboratorul 1.
După implementare se face testarea proiectului în placă, se stabilesc valori binare, conform tabelului 1, prin intermediul comutatorelor SW1-4 de pe placă şi se urmăreşte afişajul 7 segmente.

Activități suplimentare

Dezactivați şi activați pe rând, prin intermediul comutatoarelor rămase nefolosite, fiecare caracter al afişajului 7 segmente.
Reproiectati în VHDL decodificatorul binar 7 segmente.

Circuite Logice Programabile

LABORATOR 3

DESCRIEREA ÎN VHDL A CIRCUITELOR COMBINAȚIONALE, PROIECTAREA IERARHICĂ, ANALIZAREA FIŞIERELOR RAPORT

SCOPUL LUCRĂRII

Obiectivul acestei lucrări este familiarizarea cu limbajul VHDL într-o manieră cât mai practică şi anume descriind circuite combinaționale simple, cu observații asupra topicii şi structurii limbajului. De asemenea se va învăța şi maniera de proiectare ierarhică prin reprezentarea modulelor VHDL ca şi blocuri în editorul schematic. Se vor studia fişierele raport generate în fiecare fază de proiectare.

IINTRODUCERE TEORETICĂ

Considerații teoretice. Noțiunile introductive limbajului VHDL

Folosirea logicii sintetizate a devenit o practică industrială în ultimii ani. Două limbaje majore din acest punct de vedere sunt Verilogul şi VHDL-ul (VHSIC Hardware Description Language).Avantajele sunt numeroase:

- Portabilitatea proiectelor pentru diferite tehnologii în care sunt realizate cipurile, în care urmează să se facă implementarea;
- Maniera de descriere a proiectului sub formă de cod îi conferă acestuia claritate mai bună decât dacă ar fi fost descris sub formă de schemă;
- Timp de proiectare redus;
- Opțiuni de optimizare a proiectului, cum ar fi cele de arie sau/şi viteză.
- Folosirea diferitelor construcții specifice VHDL-ului cum ar fi: package-urile şi bibliotecile (library), permit reutilizarea lor în alte proiecte sau împărțirea lor între membrii grupului de proiectare.

Pentru o mai bună imagine asupra structurii unui cod VHDL, vezi figura 1.

Limbajul VHDL are reputația de a fi unul foarte complex, totuşi subsetul de instrucțiuni VHDL care sunt sintetizabile este mai redus şi aceasta este partea care urmează să fie abordată. Pe parcursul lucrării vom mai întâlnii noțiunea de instrucțiune sintetizabilă sau cod sintetizabil, ce înseamnă aceasta? După ce am scris codul VHDL al proiectului, acesta trebuie compilat pentru a depista eventuale erori de sintaxă, se poate face pe urma o verificare a proiectului (simulări funcționale sau în domeniu timp) iar una dintre fazele finale ale proiectării este sintetizarea proiectului, adică convertirea codului VHDL într-un set de celule primitive sau componente (CLB-uri în cazul FPGA-ului) care pot fi asamblate în tehnologia către care este direcționat proiectul.

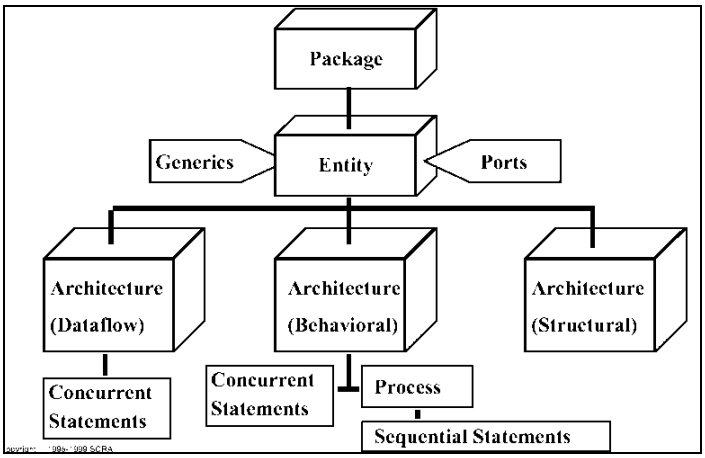


Figura 1 Principalele părți (blocuri) care alcătuiesc un model VHDL complet

Iată câteva construcții de bază pe care le vom folosii pentru sinteză:

- atribuire de semnal : a **<=** b;
- atribuire de variabilă : a **:=** b;
- comparații: **=** (egal), **/=** (diferit), **<>** (mai mare, mai mic), <= (mai mare sau egal), **>=** (mai mic sau egal);
- operatori logici: (and, xor, or, nand, nor, xnor, not);
- instrucțiunea **if:** if (a = b) then ...
- end if | elsif…
- instrucțiunea **for** (folosită pentru bucle când se doreşte crearea matricilor de elemente);
- alte construcții sunt: **“when else”, “case”, “wait”,** de asemenea si operatorul “**:=**” pentru atribuire de variabile.

Alte detalii despre sintaxă vor fi introduse pe parcurs pe măsură ce vor fi descrise circuitele logice (pentru o descriere mai detailată vezi cursul).

Generalități cu privire la limbajul VHDL:

- VHDL-ul nu este *case sensitive* (nu ține cont de litere mici sau majuscule);
- “;” este folosit pentru a indica terminarea unei instrucțiuni;
- “-- “ indică începutul unui comentariu;
- VHDL-ul este un limbaj foarte tipizat (număr mare de tipuri de operanzi), există foarte puține conversii automate între tipuri diferite de operanzi, majoritatea operațiilor trebuie făcute cu operanzi de acelaşi tip.

În continuare se vor prezenta câteva exemple pentru a introduce subsetul de instrucțiuni VHDL sintetizabile. La unele dintre exemple se vor prezenta mai multe variante de implementare pentru a înțelege mai bine versatilitatea limbajului.

Exemplele sunt:

- Multiplexor 2 - intrări şi 1 – ieşire;
- Codificator de prioritate cu 8-intrări;
- Decodificator 3-intrări şi 8-ieşiri;
- Sumator cu transport succesiv pe 8-biți.

În figura 2 se prezintă cadrul declarativ în care se integrează fiecare proiect descris în VHDL.

```
library IEEE;
use …   --numele bibliotecilor
        --şi package-urilor folosite în
model
entity nume_model is
port
(
        --lista intrărilor şi ieşirilor
);
end nume_model;

architecture nume_arhitectură of
nume_model is
begin
        ...
        --instrucțiuni VHD concurențiale
        ....
end architecture_name ;
```

Figura 2.

Multiplexor 2 - intrări şi 1 – ieşire

Figura 3 prezintă descriere VHDL folosind instrucțiunea "**when else"** a unui mux.2-1, cu intrări şi ieşiri de date pe 8-biți.

Declarația **library** se foloseşte pentru a face referire la un grup, unități de proiect anterior definite în VHDL (alte entități sau proceduri/funcții sunt cele cunoscute şi sub numele de package-uri).

Declarația **use** specifică ce entități şi package-uri sunt folosite din librăria apelată. Package-ul **std_logic1164** , este foarte frecvent apelat şi face referire la tipul de date definit în exemplul nostru.

Limbajul VHDL recunoaşte implicit un tip de date care suportă două valori, '1' şi '0' care sunt insuficiente pentru modelarea şi sintetizarea unor aplicații. Standardul 1164 defineşte 9 valori logice, dar numai 4 dintre acestea fiind sintetizabile: '1', '0', 'Z' (înaltă impedanță), '-' (stare indiferentă). Pe parcursul aplicațiilor vor mai apărea tipurile **std_logic** pentru operanzii definiți ca mărimi pe un singur bit şi **std_logic_vector** pentru mărimile definite ca bus-uri de date.

Declarația **entity** defineşte interfața externă a modelului, lista de porturi defineşte semnalele externe. Caracteristicile unui semnal sunt: numele (s, zero,..), mod (in, out sau inout) şi tip (std_logic sau std_logic_vector). În cazul tipurilor care definesc busuri de date se specifică şi lățimea busului:

Ex. std_logic_vector(7 downto 0) – ordine descrescătoare; std_logic_vector(0 to 7) – ordine crescătoare, ambele sunt semnale pe 8 biți. Ordinea de definire afectează declarațiile de atribuire cum ar fi y <= "11110000"; pentru ordine descrescătoare y(7) este '1', pentru ordine crescătoare y(0) = '1'.

Dacă blocul **entity** al unui proiect defineşte interfața dintre proiect şi mediul extern , blocul **architecture** reprezintă descrierea funcțională a proiectului. Se pot definii mai multe

arhitecturi pentru aceeaşi entitate, dar numai una poate fi simulată şi sintetizată la un moment dat. Directiva **configuration** se foloseşte pentru a stabili o pereche entitate – arhitectură. Numele arhitecturii (*behavior of mux2to1,* vezi Tabelul T5.3.2) este specificat de utilizator.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux2_1 is
port(
        s: in std_logic;
        zero,one: in std_logic_vector(7 downto 0);
        y: out std_logic_vector(7 downto 0));
end mux2_1;
architecture behavior of mux2_1 is
begin
        y <= one when (s = '1') else zero;
end behavior;
```

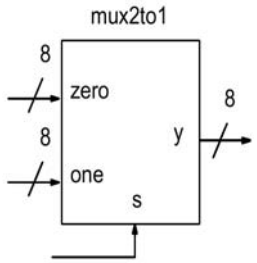


Figura 3. Cod VHDL şi diagrama bloc pentru multiplexor 2-1

Instrucțiunea "**when…else"** este o instrucțiune de atribuire (de valori) condițională a semnalelor. Această instrucțiune este cunoscută ca fiind una concurențială (*concurrent statement*), în discuție mai intrând şi instrucțiunile secvențiale (*sequential statement*), deosebirile vor fi discutate mai târziu.
În continuare se va definii un alt tip de arhitectură pentru mux.2-1, vezi figura4.

```
architecture behavior of
mux2to1_8 is

begin


 comb: process (s, zero, one)

 begin

          y<=zero;

          if (s = '1') then
```

Figura 4. Arhitectură mux.2-1 utilizând directiva **process**

Arhitectura din Tabelul figura 4 foloseşte blocul (directivă) **process** pentru a descrie funcționarea multiplexorului. Acest bloc poate fi asimilat ca fiind o instrucțiune concurențială. În cadrul acestui bloc sunt permise numai instrucțiunile secvențiale.

Atribuirile de valori semnalelor se desfăşoară secvențial, astfel că, dacă se fac mai multe atribuiri unui semnal în cadru unui astfel de bloc, doar ultima atribuire va fi luată în considerare.

Câteva instrucțiuni secvențiale sunt: **“if…else”, “case”, “for…loop”**. Lista semnalelor care urmează cuvântului cheie **process** este aşa numita listă a senzitivităților (*sensitivity list*), rularea instrucțiunilor din blocul proces în timpul unei simulări, va avea loc numai când unul dintre aceste semnale îşi schimbă valoarea (în acest caz se spune că are loc un eveniment).

Codificator de prioritate cu 8-intrări

În blocul **entity** al modelului (vezi figura 5) s-au definit porturile de intrare şi de ieşire, iar în blocul **architecture** prin intermediul declarației **process** s-a definit funcționarea modelului.

În cadrul procesului ordinea secvențială a instrucțiunilor afectează prioritatea atribuiri semnalelor, astfel în cadrul descrieri noastre intrarea *y1* va avea prioritatea cea mai mare.

Remarcăm în cod linia ce conține *vec <= B”000”* , *B* se referă la forma binară a valorii atribuite operandului.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity priority is
port (
        y1, y2, y3, y4, y5, y6, y7: in std_logic;
        vec: out std_logic_vector(2 downto 0));
end priority;
— Architecture body
architecture behavior of priority is
begin
 process (y1,y2,y3,y4,y5,y6,y7)
 begin
        if (y7 = ’1’) then vec <= ”111”;
        elsif (y6 = ’1’) then vec <= ”110”;
        elsif (y5 = ’1’) then vec <= ”101”;
        elsif (y4 = ’1’) then vec <= ”100”;
        elsif (y3 = ’1’) then vec <= ”011”;
        elsif (y2 = ’1’) then vec <= ”010”;
        elsif (y1 = ’1’) then vec <= ”001”;
        else vec <= B”000”;
        end if;
  end process;
end behavior;
```

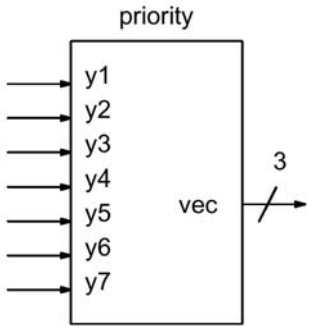


Figura 5. Cod VHDL şi diagrama bloc pentru codificator prioritar cu 8-intrări

Decodificator 3-intrări şi 8-ieşiri

Un alt exemplu de modelare a unei ahitecturi folosind blocul **process** este cel care se refera la decodificatorul 3-8 (vezi figura 6.a). În cadrul acestui model procesul este descris folosind instrucțiunea secvențială "**case…when**".
În cazul în care vreuneia dintre ieşiri nu i se atribuie nici o valoare, sintetizatorul va presupune ca ieşirea respectivă trebuie să-şi păstreze valoarea curentă şi astfel se va genera un latch pentru ieşirea respectivă, cea ce înseamnă a irosi din resursele cipului în care urmează a fi implementat modelul şi de asemenea întârzieri în plus.

Figura 6.b prezintă o altă variantă de cod VHDL pentru decodificator, de data aceasta se folosesc instrucțiuni concurențiale, care după cum se poate observa se derulează în afara vreunui bloc proces.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dec3to8 is port (
        sel: in std_logic_vector (2
        downto 0); — selector
        en: in std_logic; — enable
        y: out std_logic_vector (7
        downto 0)); — ieşirile sunt
        active pe zero
 end dec3to8;

architecture behavior of dec3to8 is
begin
  process (sel,en)
  begin
        y <= ”11111111”;
        if (en = ’1’) then
        case sel is
        when ”000” => y(0) <= ’0’;
        when ”001” => y(1) <= ’0’;
        when ”010” => y(2) <= ’0’;
        when ”011” => y(3) <= ’0’;
        when ”100” => y(4) <= ’0’;
        when ”101” => y(5) <= ’0’;
        when ”110” => y(6) <= ’0’;
        when ”111” => y(7) <= ’0’;
        when others => null;
        end case;
        end if;
  end process;
end behavior;
```

Figura 6.a Instrucțiuni secvențiale

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dec3to8_alt is
port (
        sel: in std_logic_vector(2 downto 0); — selector
        en: in std_logic; — enable
        y: out std_logic_vector(7 downto 0)); — ieşirile
        sunt active pe zero
end dec3to8_alt;
architecture behavior of dec3to8_alt is
begin
        y(0) <= ’0’ when (en = ’1’ and sel = ”000”) else ’1’;
        y(1) <= ’0’ when (en = ’1’ and sel = ”001”) else ’1’;
        y(2) <= ’0’ when (en = ’1’ and sel = ”010”) else ’1’;
        y(3) <= ’0’ when (en = ’1’ and sel = ”011”) else ’1’;
        y(4) <= ’0’ when (en = ’1’ and sel = ”100”) else ’1’;
        y(5) <= ’0’ when (en = ’1’ and sel = ”101”) else ’1’;
        y(6) <= ’0’ when (en = ’1’ and sel = ”110”) else ’1’;
        y(7) <= ’0’ when (en = ’1’ and sel = ”111”) else ’1’;
end behavior;
```

Figura 6.b Instrucțiuni concurențiale

Sumator cu transport succesiv pe 8-biți

În subcapitolul anterior am văzut cum se modelează un sumator, în mediul Xilinx folosind editorul schematic. În acest subcapitol vom prezenta codul VHDL pentru sumatorul cu transport succesiv pe 8–biți. Codul din tabelul T5.3.6 introduce instrucțiunea **loop** cu ajutorul căreia se generează logica specifică sumatorului.
În cod se defineşte un semnal temporar signal "*c: std_logic_vector(7 downto 0);",* pentru a păstra valoarea semnalului carry care se propagă intern.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity adder8 is port (
        a,b: in std_logic_vector (7 downto 0);    -- semnalele de intrare
        cin: in std_logic;                                    -- transport la intrare
        sum: out std_logic_vector(8 downto 0);-- semnale de ieşire
        cout: out std_logic);                          -- transport la ieşire
end adder8;
architecture behavior of adder8 is
        signal c: std_logic_vector(8 downto 0);
begin
 process (a,b,cin,c)
 begin
        c(0) <= cin;
        for i in 0 to 7 loop
        sum(i) <= a(i) xor b(i) xor c(i);
        c(i+1) <= (a(i) and b(i)) or (c(i) and (a(i) or b(i)));
        end loop;
        cout <= c(8);
 end process;
end behavior;
```

Figura 7. Cod VHDL pentru sumatorul cu transport succesiv pe 8 biți

Până la acest punct al lucrării au fost examinate câteva exemple, prin intermediul cărora s-a făcut şi o introducere în limbajul VHDL. S-a putut observa că modelarea unui circuit în VHDL se poate face în mai multe variante (se pot folosii atât instrucțiuni concurențiale cât şi/sau instrucțiuni secvențiale), rămâne la latitudinea proiectantului ce variantă de descriere va folosii.

Desfăşurarea lucrării

Câteodată este mai uşor de vizualizat proiectele folosind o reprezentare în schematic a acestora. Orice modul descris în VHDL poate avea şi o reprezentare în schematic sub forma unui bloc. Mai multe module descrise în VHDL pot fi conectate împreună folosind simbolurile schematice create.

În continuare, plecând de la codul VHDL al multiplexorului cu două intrări de date pe 8 biți şi una de selecție, vezi figura 3, se va crea un simbol schematic al acestuia.
Folosind simbolul schematic creat se va extinde magistrala de date intrare şi ieşire astfel se va proiecta un multiplexor cu 2 intrări de date pe 16 biți şi o intrare de selecție. Acesta va fi simulat şi implementat. În ultima etapă se vor analiza fişierele raport.

Pasul 1: Crearea proiectului.

Se va crea un proiect cu numele *mux_sch,* atenție la directorul de lucru.
Se va selecta ca *Top-Level Source* modul de lucru *Schematic*, vezi laborator 2.
După ce proiectul a fost creat, conform laboratorului 2, se adaugă un fişier sursă nou, *Project →New Source*, se alege un nume pentru fişierul sursă *mux2_1* şi se alege să fie de tipul *VHDL*, vezi laborator 1. Porturile pot fi adăugate utilizând *wizardul* sau dacă se trece peste această etapă, pot fi adăugate direct în codul VHDL generat la creearea fişierului. Codul VHDL al multiplexorului cu două intrări de date şi una de selecție va fi cel din figura 3. În acest moment proiectul trebuie să arate ca şi în figura 8.

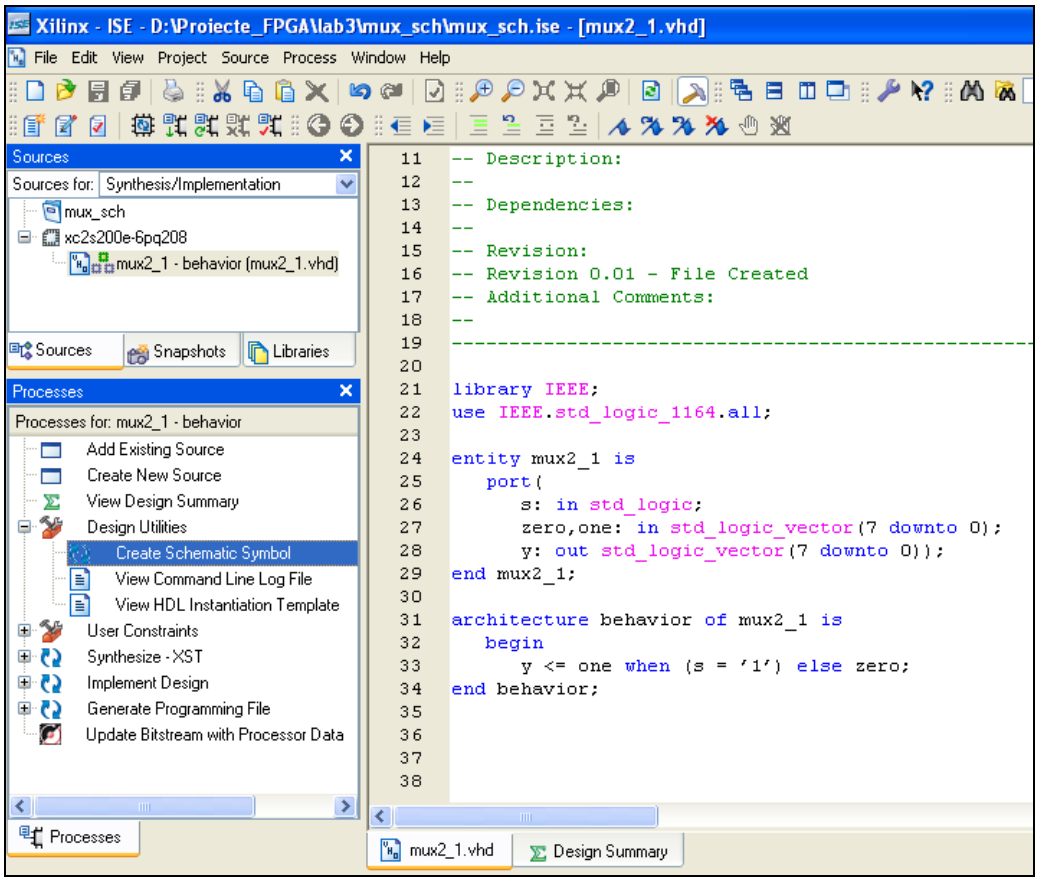


Figura 8

Pasul 2: Proiectarea ierarhică, crearea simbolurilor în schematic

În fereastra *Process,* sub meniul *Design Utilities* se face dublu click pe *Create Schematic Symbol,* aceasta va avea ca rezultat crearea unui simbol schematic şi adăugarea lui în biblioteca de simboluri din editorul schematic.
În continuare se va adăuga o nouă sursă proiectului, *Project, New Source,* de tipul schematic şi având numele *mux_top.* Odată adăugat fişierul schematic, dublu click pe acest, trebuie să se deschdă editorul schematic iar în fereastra *Sources* → *Symbols* trebuie să regăsim simbolul schematic al modulului VHDL *mux2_1.*
Aducem două instanțe ale simbolului creat în pagina de editare, vezi laboratorul 2. După care utilizând cursorul pentru trasee, vezi bara cu unelete , se conectează între ele modulelele conform figurii 9.

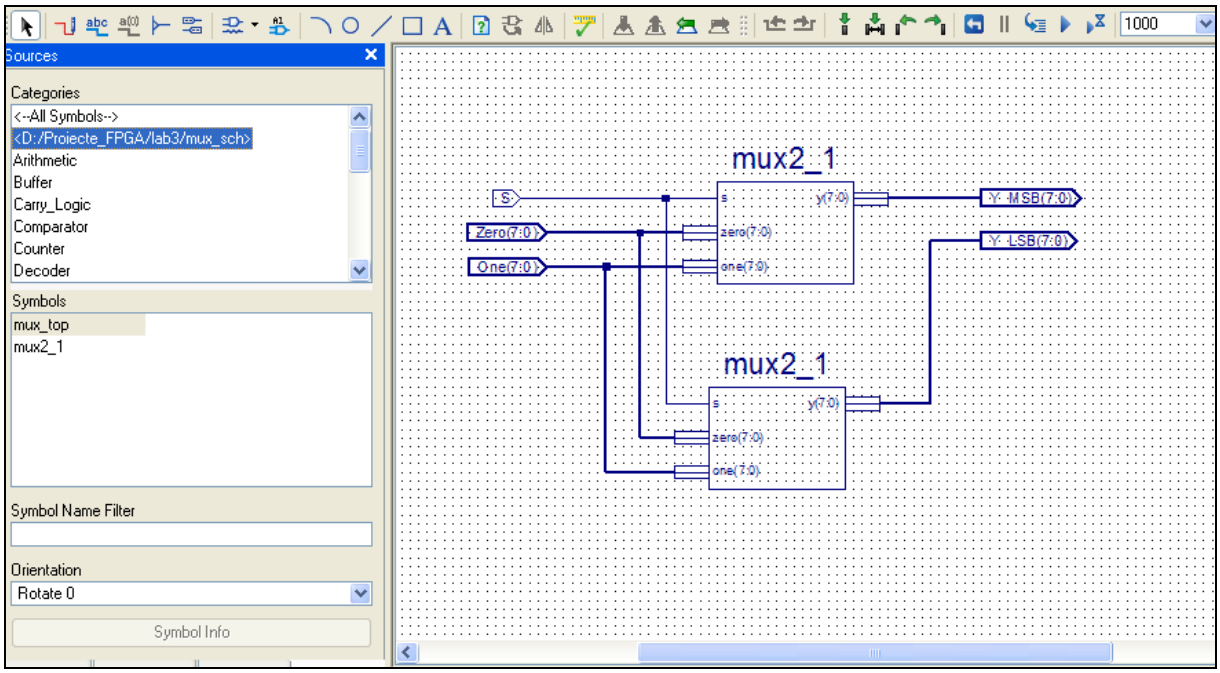


Figura 9

Editorul schematic recunoaşte automat că este vorba de magistrale pe 8 biți şi conectează corespunzător componentele. Se poate observa că magistraelele de date intrare şi ieşire sunt conectate împreună. După definirea traseelor se inserează padurile de intrare şi ieşire folosind notațiile *one şi* zero pentru intrarile de date, *Y* pentru ieşire şi *S* pentru intrarea de selecție, vezi figura 9. Atenție la stabilirea numelor padurilor de magistrală se păstrează dimensiune (0:7), vezi figura 10.
Fiecare instanță a multiplexorului *mux2_1*, poate fi redenumită, dublu click pe instanță şi în câmpul *InstName* se scrie *MuxA,* respectiv *MuxB.*
În acest moment se poate face o salvare şi se poate ieşii din editorul schematic.

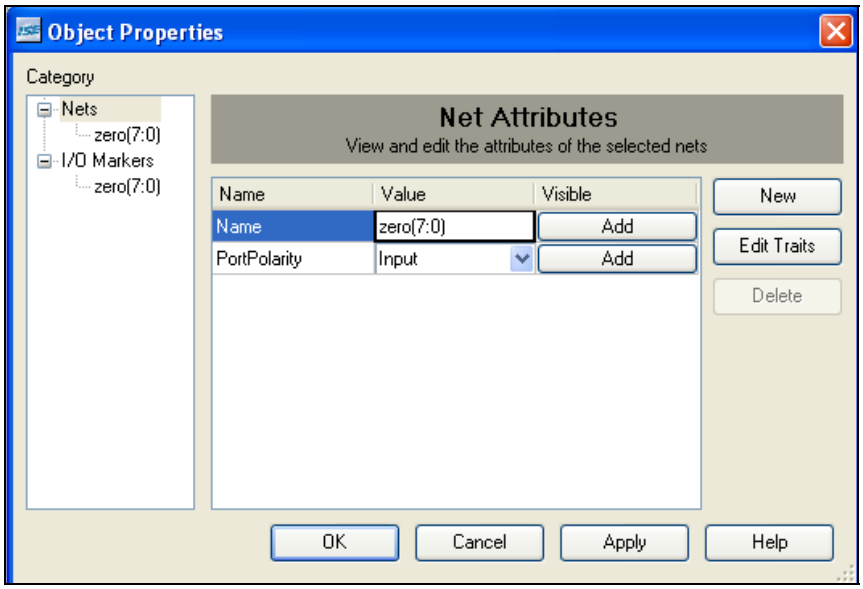
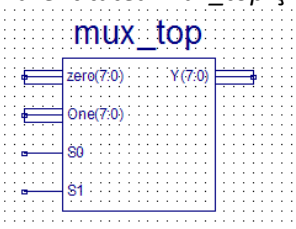


Figura 10

În această etapă în fereastra *Sources* se poate observa entitatea *top* de tip schematic *mux_top.sch* sub care se află instanțele *MuxA, MuxB* ale multiplexorului descris în VHDL.
Dacă se selectează entitatea *mux_top* şi se alege din nou opțiune *Create Schematic Symbol*, se va obține simbolul 

, selectând acest simbol şi alegând din bară opțiunea *Push*  , se va merge practic la ierahia inferioară reprezentată de instanțele *MuxA, B*.
Echivalentul în VHDL a reprezentării schematice a *mux_top.sch* se poate vedea alegând opțiune *View HDL Functional Model.* Aşa ar arăta entitatea noastră *TOP* dacă proiectul ar fi fost de tip *HDL* şi nu schematic.

Pasul 3: Crearea testbench-ului şi simularea funcțională a proiectului

Pentru crearea testbenchului se selectează entitate *Top*, click dreapta *New Source,* se selectează tipul de sursă *Test Bench...*şi se alege numele *mux_top_tb*, *Next,* se selectează entitatea top la care se asociază testbenchul *mux_top*, *Next,* Finish. În fereastra nou apărută se selectează *Combinatorial* şi *Initial Length... 2000ns, Finish.* La fel ca şi în lucrarear anterioară se selectează starea semnalelor de intrare conform modelului din figura 11. Semnalul de selecție primeşte stări alternative 0, 1 logic, iar la intrările de date, lui *Zero* i se lasă valoare *0,* iar intrări numite *One* i se dă valoarea 1 fiecărui bit al magistralei (click pe + pentru a expanda magistrala). Se salvează şi se închide editorul.

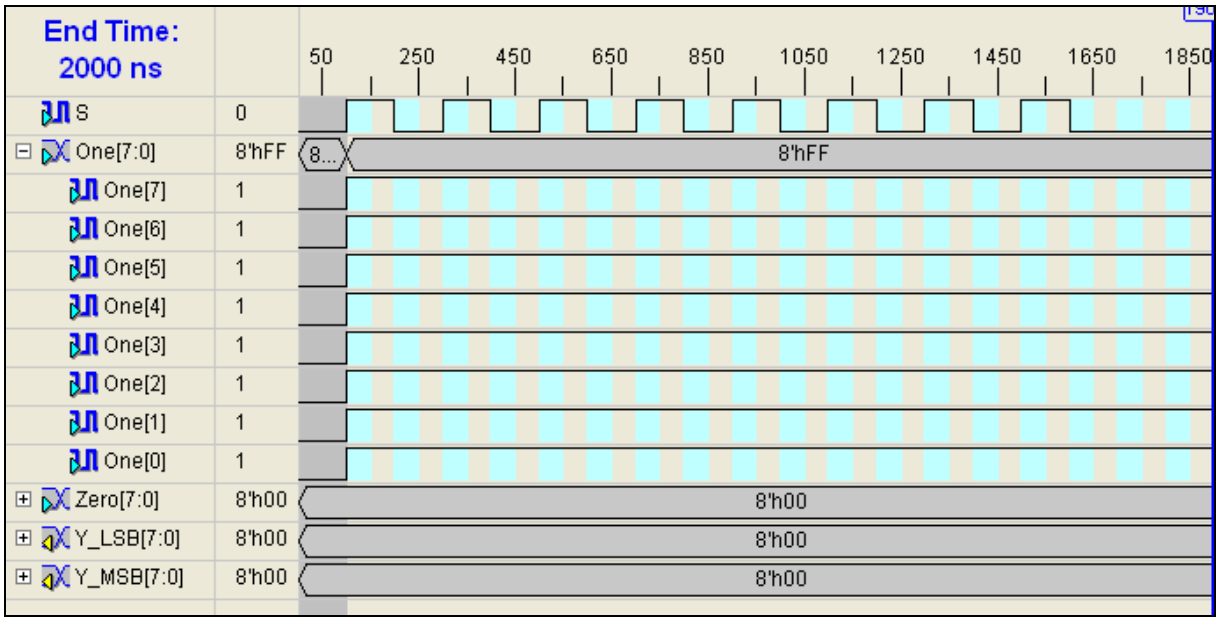


Figura 11

În fereastra *Sources, Sources for* se selectează *Behavioral Simulation, Xilinx Ise Simulator, Simulate Behavioral Model.*
În figura 12, sunt prezentate rezultatele simulării.

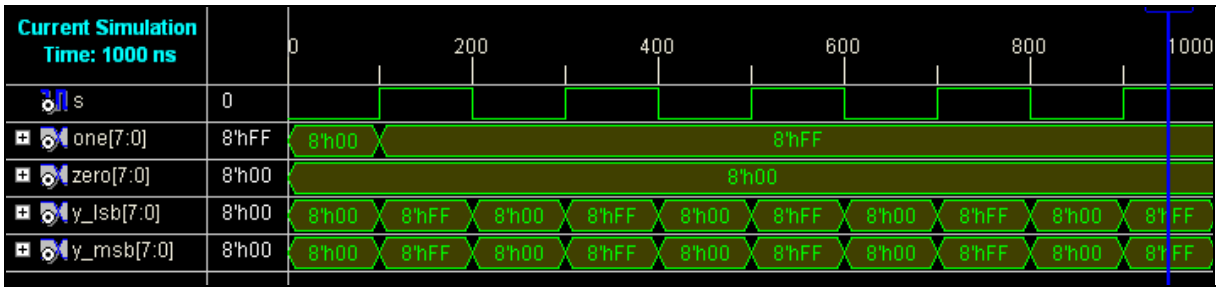


Figura 12.

Se poate observa pentru stările definite în codul VHDL, 0 şi 1 ale semnalului de selecție S, ieşirea este corespunzătoare h'0 respectiv h'1. Se închide fereastra de testbench.

Pasul 4: Implementarea şi testarea proiectului

În *Sources, Sources for* se revine din nou la *Synthesis/Implementation.*
Implementarea se realizează conform procedurii din laboratorul 1.

Pasul 5: Analizarea fişierelor raport

În urma implementării în fiecare etapă se generează un fişier raport, acestea pot fi vizualizate fie expandând (click pe +) fiecare etapă din fereastra *Processes*, fie rezumate în *Design Sumary*, vezi fugura 13. Informațiile din figura 13 permit proiectantului să determine rapid resursele consumate din circuitul FPGA în urma implementării proiectului. Astfel, de interes sunt numărul de *Slice*-uri consumate (acestea reprezentând blocurile de bază ale circuitelor FPGA), numărul de *LUT*-uri (tabele de adevăr, câte 2 pentru fiecare *slice*), şi numărul de blocuri de intrare/ieşire *IOB,* utilizate.
Pentru a avea acces la informații mai detailate se pot accesa rapid fişierele raport generate în fiecare etapă, tot din *Design Sumary*, vezi figura 14.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of 4 input LUTs	8	4,704	1%
Logic Distribution			
Number of occupied Slices	4	2,352	1%
Number of Slices containing only related logic	4	4	100%
Number of Slices containing unrelated logic	0	4	0%
Total Number of 4 input LUTs	8	4,704	1%
Number of bonded IOBs	33	142	23%
Total equivalent gate count for design	48		
Additional JTAG gate count for IOBs	1,584		

Figura 13

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	V 14. mar. 00:31:12 2008	0	1 Warning	0
Translation Report	Current	V 14. mar. 00:31:23 2008	0	0	0
Map Report	Current	V 14. mar. 00:31:31 2008	0	2 Warnings	2 Infos
Place and Route Report	Current	V 14. mar. 00:31:41 2008	0	0	1 Info
Static Timing Report	Current	V 14. mar. 00:31:45 2008	0	0	3 Infos
Bitgen Report	Current	V 14. mar. 00:31:51 2008	0	0	0

Figura 14

1. Translate Report - prezintă orice eroare legată de proiectare sau de constrângeri;
2. Map Report – confirmă resursele utilizate şi oferă detalii despre logica redundantă înlăturată şi despe optimizarea celei rămase. Prezintă şi zona din dispozitiv alocată modulelor proiectului;
3. Post-Map Static Timing Report – prezintă întârzierile din circuit, rezultate în urma aplicării constrângerilor definite de utilizator. În acest raport întârzirile se referă numai la cele prin porți nu şi la cele care apar în urma rutării.
4. Place and Route Report – prezintă pas cu pas procesul de plasare şi rutare. Se ține cont şi de constrângerile aplicate de utilizator, dacă acestea nu pot fi respectate procesul se termină cu eroare.
5. Asynchronous Delay Report – prezintă un raport al întârzierilor, ținând cont atât de cele prin porți cât şi de cele de rutare;
6. Pad Report – prezintă plasarea porturilor şi standardele semnalelor ce pot fi prezente la porturi.

Activități suplimentare

- Se va crea un director cu numele *Lab.3*, şi pentru fiecare din codurile VHDL din figurile 3,5,6,7 se va crea un proiect în schematic, se va scrie codul cu editorul de cod HDL, se va face o analiză sintactică a fiecărui cod scris, se salvează codul după ce a fost verificat şi se generează un simbol schematic. Se va simula funcțional fiecare simbol schematic descris.
- Se implementează toate cele patru exemple, simulate la punctul anterior şi se studiază fişierele raport rezultate în urma implementării, se încarcă în placă şi se testează . La implementare se scriu şi fişierele de contrângeri corespunzătoare ținându-se cont de pini asociați comutatoarelor şi LED-urilor de pe placa DIO4, vezi laborator 1.

Circuite Logice Programabile

LABORATOR 4

DESCRIEREA ÎN VHDL A CIRCUITELOR SECVENȚIALE. DEFINIREA CONSTRÂNGERILOR DE TIMP

SCOPUL LUCRĂRII

Logica secvențială este termenul generic folosit pentru proiectele care conțin elemente de stocare, în special bistabile. Toate circuitele secvențiale beneficiază de un semnal de sincronizare numit semnal de tact sau clock.

În această lucrare se vor extinde cunoştinţele de VHDL modelând circuitele secvențiale de bază: bistabile , numărătoare , registre. Pe lângă modelele VHDL corespunzătoare circuitelor secvențiale în această lucrare vor fi tratate şi aspectele legate de generarea nedorită a logici secvențiale, datorată manierei greşite de descriere a proiectelor în VHDL. De asemenea va fi prezentată şi modalitatea de definire a constrângerilor de timp: frecvență de lucru, timp de set up şi modalitatea de vizualizarea a fişierelor raport aferente acestora. Modelele descrise vor fi implementate cu ajutorul programului Xilinx ISE.

IINTRODUCERE TEORETICĂ

Circuitele secvențiale conțin atât logică combinațională cât şi elemente de stocare, ca urmare a acestui fapt toate circuitele secvențiale pot fi descompuse în două blocuri, unul combinațional şi unul de stocare. În figura 1 se prezintă diagrama unui sistem secvențial.
În acest sistem secvențial elementul de stocare este circuitul bistabil. Fiecare model secvențial se înscrie într-o schemă de cod ca şi cea prezentată în figura 2.
În continuare se vor dezvolta câteva modele VHDL ale circuitelor secvențiale de bază:

- Latch-uri
- Circuite bistabile
- Numărătoare
- Registre.

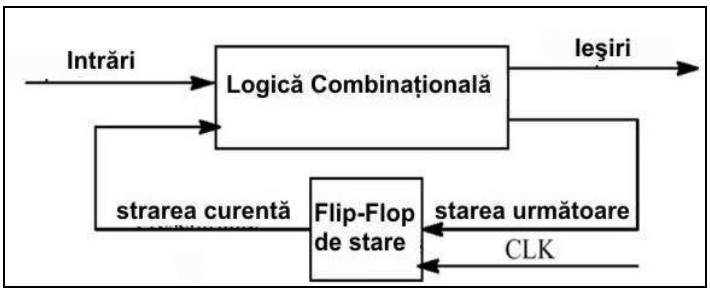


Figura 1. Sistem secvențial

```
library – declararea bibliotecilor folosite
entity nume_model is
port
(
lista intrărilor şi a ieşirilor
);
end nume_mode;
architecture behavior of nume_model is
        declarare de semnale interne
begin
        –– procesul state defineşte elementele de stocare
  state: process ( lista senzitivităților –– clock, reset, next_state inputs)
  begin
        instrucțiuni VHDL
  end process state;
        –– procesul combinațional va definii logica combinațională
  comb: process ( lista sensitivităților––de obicei include toate semnalele de intrare)
  begin
        instrucțiuni VHDL
  end process comb;
end behavior;
```

Figura 2. Model de descriere a circuitelor secvențiale în VHDL

Latch şi circuite bistabile

Rolul unui element secvențial de tip latch sau bistabil este de a păstra o valoare (a unui semnal) o anumită perioadă de timp. În această secțiune vor fi prezentate câteva exemple VHDL care modelează un astfel de comportament. Latch-urile şi bistabilele sunt de fapt celule de memorie care pot stoca la un moment dat 1–bit. Diferența dintre ele este ca latch-ul comută pe nivel logic (0 sau 1), în timp ce bistabilul comută pe frontul semnalului (crescător sau descrescător).

Model VHDL pentru Latch de tip D

În figura 3 este prezentat modelul VHDL al latch-ului de tip D. Am ales acest tip pentru exemplificare datorită frecventei lui utilizării mai ales ca element de stocare a biților de control (ex. stocare a bitului de flag). Gândind latch-ul în aceşti termeni, valoarea intrării D se va regăsii la ieşire ori de câte ori intrarea de control C are valoarea 1, altfel ieşirile latch-ului rămân neschimbate.
Modelul de latch din figura 3 este un model comportamental (behavioral model) care după cum se poate observa a necesitat practic două linii de cod (vezi codul din interiorul blocului proces).Compilatorul VHDL va asocia această descriere cu un latch deoarece în cod nu s-a specificat ce se întâmplă dacă semnalul C nu are valoarea 1. Astfel compilatorul va genera un latch pentru a reține valoarea lui Q între două invocări ale procesului.

În general compilatorul VHDL generează latch-uri pentru semnalele din cadrul instrucțiunilor *if* sau *case* , în cazul în care nu s-a ținut cont de toate combinațiile semnalelor de intrare.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dlatch is
  port (D, C: in STD_LOGIC;
       Q, QN: buffer STD_LOGIC );
end dlatch;

architecture dlatchc_b of dlatch is
begin
process( C, D, Q)
  begin
    if (C = '1') then Q <= D;
QN <= not Q;
end if;
  end process;
end dlatchc_b;
```

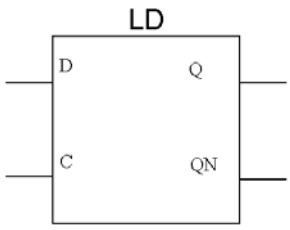


Figura 3. Cod VHDL şi diagrama bloc pentru latch-ul de tip D

În secvența de cod din figura 3 se mai poate observa folosirea unui nou tip de port, *buffer.* Acest tip este asemănător cu tipul *out*, cu deosebirea că semnalele de la acest port pot fi şi citite, adică în cadru unei secvențe de cod semnalul poate apărea în membrul stâng al unei declarații de atribuire.

Modele VHDL pentru bistabili de tip D

Bistabilele mai sunt de asemenea cunoscute şi sub numele de registre (pe un bit), ele sunt modelate în VHDL în cadrul blocurilor proces folosind instrucțiunile *wait* şi *if* , deoarece instrucțiunea *wait* nu este sintetizabilă nu ne vom ocupa de ea.
De asemenea în cadrul procesului mai apar şi expresii care permit detectarea tranziției unui semnal (în acest caz este vorba de semnalul de tact).

În figura 4.a. se prezintă codul VHDL care modelează comportamental un bistabil de tip D.

După cum se poate observa din ambele coduri (figura 4a. şi 4b.) pentru a descrie un bistabil folosim atributul *event,* care este un atribut specific semnalelor. Dacă SIG este un nume de semnal , atunci construcția “SIG ‘event” va returna valoarea booleană “adevărat” (adică procesul din care face parte semnalul SIG va fi evaluat, se vor executa instrucțiunile din acest bloc) ori de câte ori semnalul SIG tranzitează dintr-o stare logică în alta, altfel valoarea returnată este “fals”. În cadrul instrucțiunii *if* expresia “CLK ‘event” declanşează o evaluare a procesului la fiecare tranziție a semnalului CLK, pentru a ne asigura că valoarea semnalului D este atribuită ieşirii Q numai la tranzițiile semnalului CLK din 0 în 1 (front crescător) se mai impune şi condiția “CLK=’1’”. A se observa că în lista senzitivităților este prezent numai semnalul CLK, tranzițiile pe care le suferă semnalul D nu pot să declanşeze evaluări ale procesului. Acest tip de procese în care un front al semnalului CLK sincronizează toate atribuirile de semnale se numesc procese sincrone cu tactul (clocked process).

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dff is
 port (D, CLK: in STD_LOGIC;
      Q: out STD_LOGIC );
end dff;

architecture dff_b of dff is
begin
process(CLK)
 begin
   if (CLK'event and CLK='1')
then
   Q <= D;
   end if;
 end process;
                end dff_b;
```

Figura 4.a. Codul VHDL pentru bistabilul de tip

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity dff74 is
 port (
      D, CLK, PR_L, CLR_L: in STD_LOGIC;
      Q, QN: out STD_LOGIC );
end dff74;

architecture dff74_b of dff74 is
signal PR, CLR: STD_LOGIC;
begin
process(CLR_L, CLR, PR_L, PR, CLK)
 begin
   PR <= not PR_L; CLR <= not CLR_L;
   if (CLR and PR) = '1' then
   Q <= '0'; QN <= '0';
   elsif CLR = '1' then Q <= '0'; QN <= '1';
   elsif PR = '1' then Q <= '1'; QN <= '0';
   elsif (CLK'event and CLK = '1') then
   Q <= D;    QN <= not D;
   end if;
 end process;
end dff74_b;
```

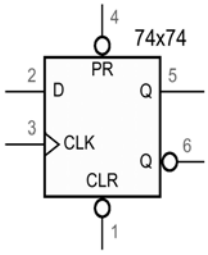


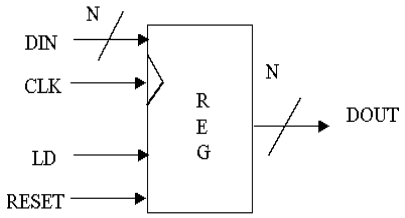
Figura 4.b. Codul VHDL şi diagrama bloc pentru bistabilul de tip D cu intrări asincrone

Conform secvenței de cod din figura 4.b. modelul de bistabil D poate fi extins descriindu-i-se şi intrări asincrone de preset, clear, precum şi o ieşire QN. Ieşirea QN poate avea un comportament ne-complementar față de Q dacă se face o setare simultană a intrărilor de preset şi clear. În modelul din figura 4.b. semnalele de preset şi clear sunt asincrone cu tactul, dacă însă expresia "*(CLK'event and CLK = '1') then*" va fi trecută ca primă condiție în cadru instrucțiunii *if* , atunci ținând cont de faptul ca toate instrucțiunile din cadrul unui proces sunt executate secvențial, atribuirea semnalelor de preset şi clear se va desfăşura sincron cu tactul astfel obținem bistabilul de tip D cu preset şi clear sincron.

Model VHDL pentru registre

Un grup de *n* bistabile care au semnal de tact comun formează un registru pe *n* biți. Cel mai adesea regiştri sunt folosiți pentru a stoca o colecție (grup) de biți înrudiți, spre exemplu un *byte* de date. În figura 5 este prezentat codul VHDL şi diagrama bloc cu intrările şi ieşirile unui registru pe 8-biți cu încărcare sincronă şi reset asincron.

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8bit is
  port ( clk, reset, ld: in std_logic;
   din: in std_logic_vector(7 downto 0);
   dout: out std_(7 downto 0));
end reg8bit;

architecture behavior of reg8bit is
signal n_state: std_logic_vector(7 downto 0);
signal p_state : std_logic_vector(7 downto 0);
begin

process(clk, reset)
begin
if (reset = '0') then p_state <= (others => '0');
elsif (clk'event and clk = '1') then
if (ld='1') then n_state <= din;
else null;
end if;
p_state <= n_state;
end if;
end process;
dout <= p_state;
end behavior;
```



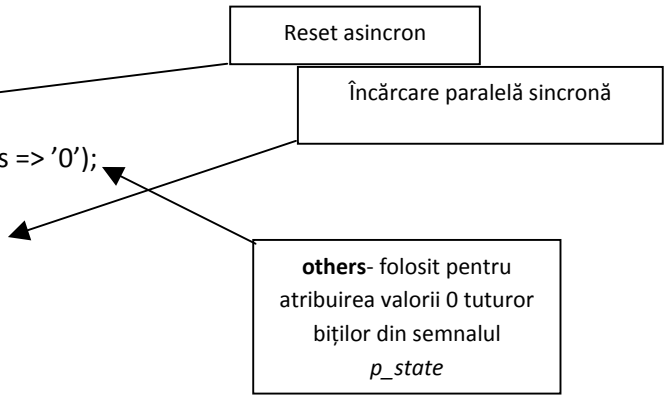


Figura 5. Cod VHDL pentru registru pe 8-biți cu încărcare sincronă şi resetare asincronă

Spre deosebire de bistabile unde informația stocată se schimbă la fiecare semnal de tact, în cazul registrului conținutul acestuia se va schimba numai pentru LD='1' (vezi procesul combinațional) şi în mod sincron cu tactul (vezi proces secvențial, expresia "p_state <= n_state" este evaluată după "clk 'event…").

Din figura 5 se poate observa că modelul VHDL al registrului respectă întru totul blocurile proces prezentate în figura 1 ca fiind specifice oricărui sistem sincron.

Procesul *state* defineşte elementul de stocare pe 8 biți, sincron cu tactul şi asincron cu un semnal de reset activ pe zero. Semnalul de ieşire a acestui proces este *p_state*. Acțiunea de stocare a informației în cadrul acestui proces este dată de faptul că lui *p_state* i se atribuie o valoare numai dacă semnalul *reset = '1'* sau semnalul de tact nu are o tranziție din 0 în 1.

Alte forme de atribuire de valori semnalelor (pe lângă cea din exemplu, care foloseşte cuvântul cheie **others**) sunt: atribuirea poziționlă (ex. p_state <= (s,s,s,s,s,s,s,s) şi atribuire după nume, ex. p_state <= (4=>s, 7=>s, 2=>s, 5=>s, 3=>s, 1=>s, 6=>s, 0=>s)), unde s este un semnal de tip std_logic care poate lua valorile 0 sau 1.

Model VHDL pentru numărător sincron

Numărătoarele sunt circuite secvențiale care parcurg un anumit număr de stări. Numărătoarele sunt folosite în sistemele digitale pentru a contoriza evenimente sau pentru

a genera adrese de memorie. Numărul stărilor prin care trece un numărător până ajunge din nou la starea din care a plecat (astfel având loc un ciclu de numărare) defineşte modulul numărătorului.

Un numărător cu m stări se numeşte numărător modulo m sau numărător divizor cu m. Un numărător binar pe n-biți este alcătuit din n bistabile şi are 2n = m stări.
Un numărător trece de la valoarea curentă la următoarea valoare ca urmare a răspunsului pe care îl dă la un impuls de numărare (tactul sistemului).

În figura 6 se prezintă modelul VHDL şi simbolul numărătorului 74x163, acesta este un numărător sincron pe 4-biți. Caracteristic numărătorului sincron este faptul că, toate bistabilele din care este alcătuit au semnal de tact comun, astfel încât toate ieşirile îşi schimbă starea în acelaşi timp. Numărătorul descris în lucrarea de față este alcătuit din bistabile de tip D pentru a facilita funcțiile de încărcare (LD) şi reset (CLR).

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity num_sin is
   port (
      CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
      D: in UNSIGNED (3 downto 0);
      Q: out UNSIGNED (3 downto 0);
      RCO: out STD_LOGIC );
end num_sin;

architecture num_sin_arch of num_sin is
signal IQ: UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
 begin
   if CLK'event and CLK='1' then
     if CLR_L='0' then IQ <= (others => '0');
     elsif LD_L='0' then IQ <= D;
     elsif (ENT and ENP)='1' then IQ <= IQ + 1;
     end if;
   end if;
   if (IQ=15) and (ENT='1') then RCO <= '1';
   else RCO <= '0';
   end if;
   Q <= IQ;
 end process;
end num_sin_arch;
```

Bibliotecă care conține tipul *unsingned* şi operatorii specifici acestui tip

IQ-semnal intern, păstrează starea curentă a numărătorului

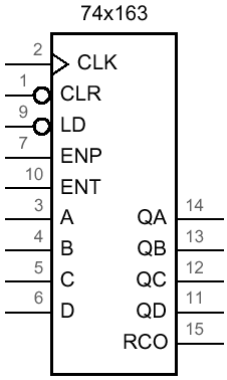


Figura 6. Simbol standard şi cod VHDL pentru numărător sincron pe 4-biți cu încărcare şi resetare sincronă

Observați că în codul din figura 6 s-a introdus o nouă bibliotecă "*use IEEE.std_logic_arith.all;*", această bibliotecă include tipurile de vectori *unsigned* şi *signed* şi operatori specifici pentru aceste tipuri. Cele două tipuri se bazează pe tipul *std_logic,* dar un vector definit *unsigned* nu poate fi atribuit unui alt vector definit *std_logic*, pentru acest gen

de atribuire se folosesc funcții de conversie. Avantajul folosirii tipului *unsigned* este ca biblioteca căreia îi aparține include operatorii "+" şi "-" aceasta permițând operații de adunare şi scădere directe între vectori. În programul nostru am declarat intrările şi ieşirile de tipul *unsigned* şi folosim operatorul "+" pentru a incrementa valoarea curentă a numărătorului.
În program se defineşte un semnal intern IQ pentru a păstra valoarea curentă a numărătorului. Am fi putu utiliza direct semnalul Q dar în acest caz trebuia sa-l declarăm ca port de tip *buffer* .

Desfăşurarea lucrării

Această parte a lucrării constă în implementarea cu programul Xilinx a codurilor VHDL prezentate anterior. De asemenea se va prezenta şi modalitatea de aplicare a constrângerilor de timp. Se va pleca de la codul VHDL al numărătorului cu încărcare sincronă şi resetare asincronă , vezi figura 6.

Pasul 1: Crearea proiectului.

În directorul personal se va crea un subdirector cu numele *lab4,* aici vor fi salvate toate proiectele ce vor fi create în cadrul acestei lucrări.
Se va crea un proiect cu numele *num,* atenție la directorul de lucru.
Se va selecta ca *Top-Level Source...*, *HDL* , vezi laborator 2.
După ce proiectul a fost creat, conform laboratorului 2, se adaugă un fişier sursă nou, *Project →New Source*, se alege un nume pentru fişierul sursă *num_sin* şi se alege să fie de tipul *VHDL*, vezi laborator 1. Porturile pot fi adăugate, conform codului din figura 6, utilizând *wizardul* sau dacă se trece peste această etapă, pot fi adăugate direct în codul VHDL generat la crearea fişierului. În această etapă poate fi făcută o verificare a sintaxei, din fereastra P*rocesses*, *Synthesize, Check Syntax.*

Pasul 2: Vizualizarea conversiei RTL a numărătorului descris în VHDL

Alegând opțiunea *View RTL Schematic* (din P*rocesses*, *Synthesize...*),vezi figura 7, poate fi vizualizată sub formă schematică „traducerea" proiectului din descriere abstractă (cod VHDL) într-o descriere cu simboluri implementabile în hardware, aşa numita descriere RTL (Register Transfer Level), vezi figurile 8, 9, 10 . Cu click dreapta pe simbolul din figura 8 şi prin alegerea opțiunii *Push Into Selected Instance*, se pot vizualiza blocurile ierarhice inferioare. Se observă că s-a generat un numărător, o poartă ŞI cu 5 intrări acestea sunt componente primare şi un bloc de cotrol. Prin acelaşi procedeu *Push...,* se poate observa că în urma sintezei blocul de control este alcătuit din două porți logice (ŞI, SAU cu intrare negată), vezi figura 10. Se închide fereastra de schematic.

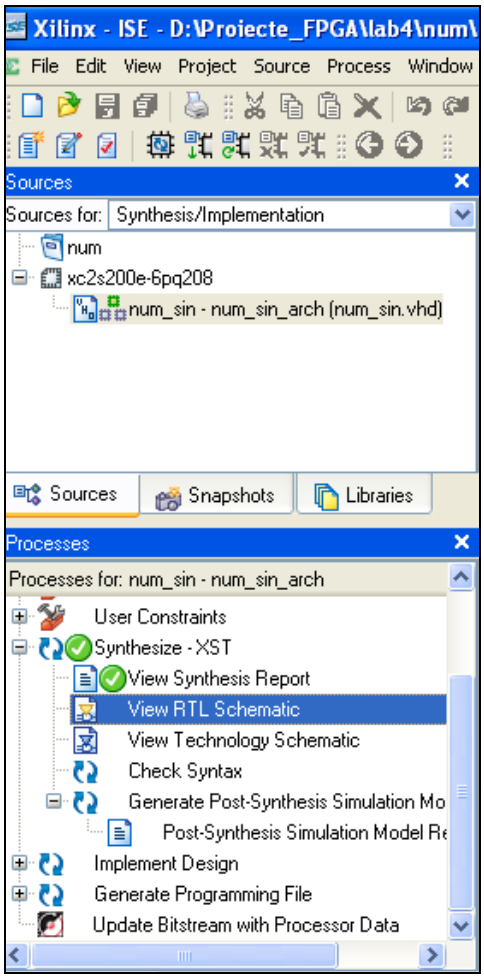


Figura 7.

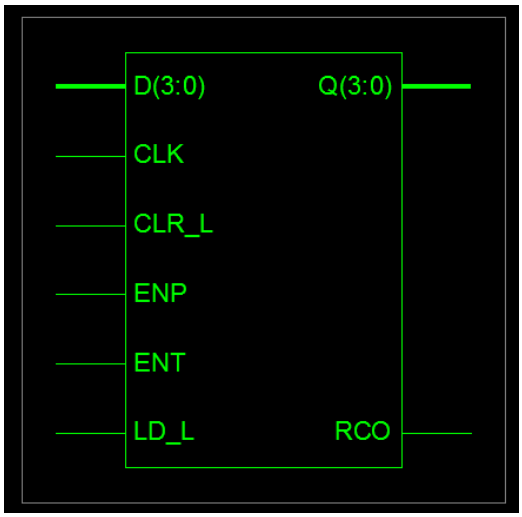


Figura 8.

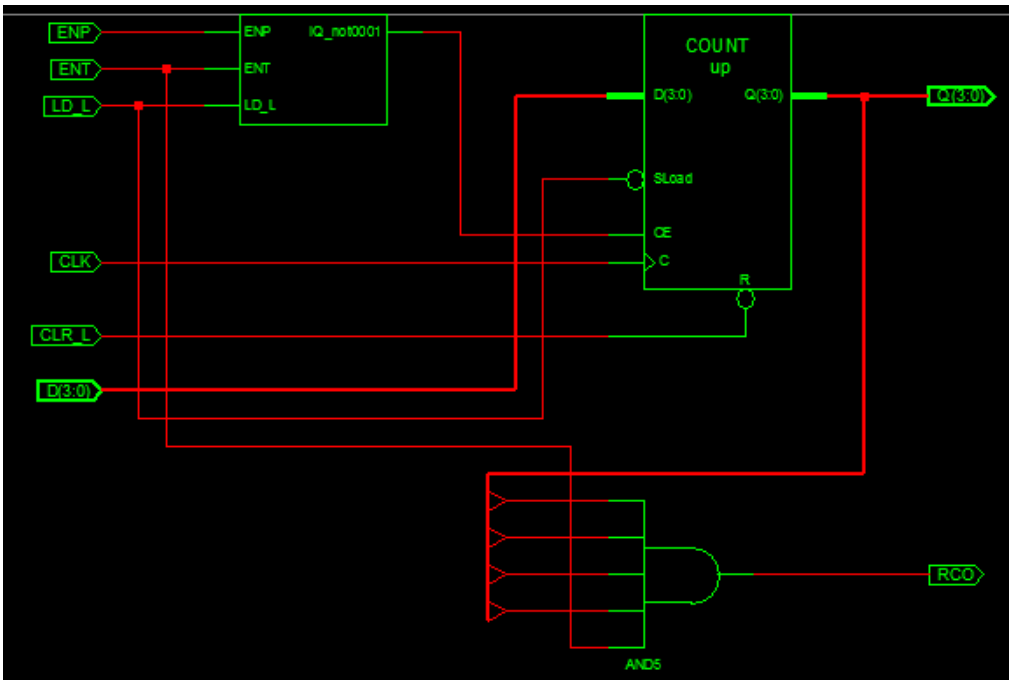


Figura 9.

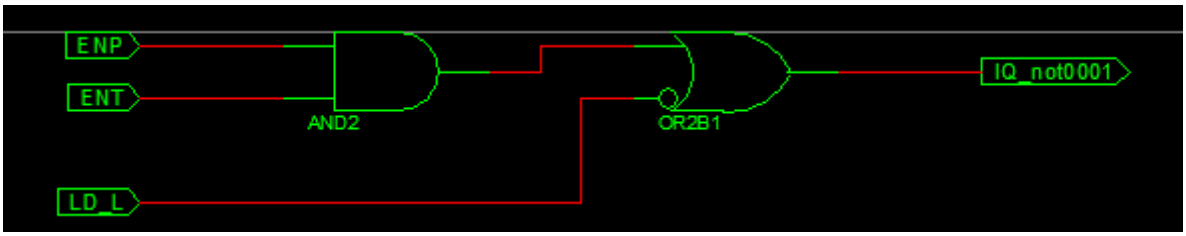


Figura 10.

Pasul 3: Crearea testbench-ului şi simularea funcțională a proiectului

Pentru crearea testbenchului la fel ca şi în laboratorul 3, se selectează entitate *Num_sin*, click dreapta *New Source,* se selectează tipul de sursă *Test Bench...*şi se alege numele *Num_sin_tb*, *Next,* se selectează entitatea top la care se asociază testbenchul *Num_sin*, *Next, Finish*.
De data acesta fiind vorba de simularea unei componente secvențiale va trebui să selectăm: frecvența semnalului de tact, timpul de setup şi întârzierea pentru validarea ieşirii.
Se va selecta o frecvență de 25 MHz pentru semnalul de tact şi următoarele valori pentru parametrii enunțați anterior:
ϕ Clock High Time: **20** ns.
ϕ Clock Low Time: **20** ns.
ϕ Input Setup Time: **10** ns.
ϕ Output Valid Delay: **10** ns.
ϕ Offset: **0** ns.
ϕ Global Signals: **GSR (FPGA)**
ϕ Initial Length of Test Bench: **2000** ns.

Astfel semnalele de la intrare vor fi valide înainte cu 10 ns de frontul crescător al semnalului de clock iar la ieşire vor rămâne valide 10 ns după frontul crescător. Durata simulării va fi de 2000 ns. Setările trebuie să fie ca şi cele din figura 11.

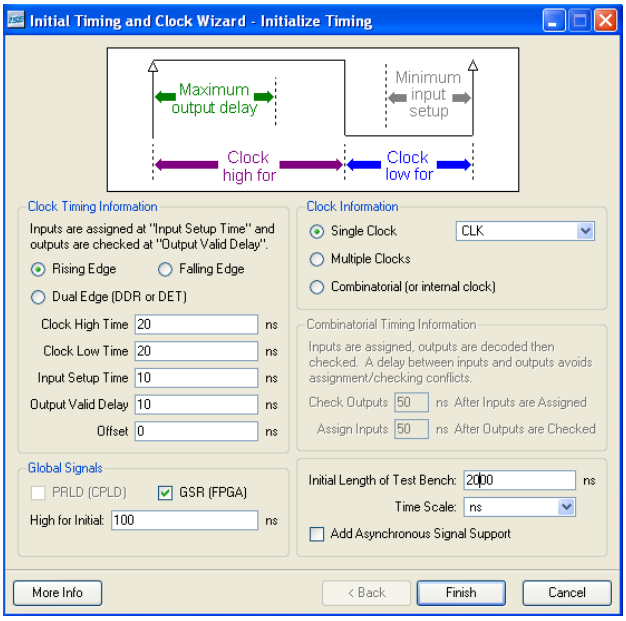


Figura 11.

În figura 12, semnalul de ştergere *CLR_L*, pleacă din 0, se face un reset, semnalele de validare *ENP, ENT,* vor fi trecute în 1 pentru a permite numărarea, la intrarea de date *D[3:0]* se stabilesc valori aleatorii (click pe + pentru a expanda magistrala), acestea vor fi citite doar în momentul în care intrarea *LD_L* trece în 0, în toate situațiile *„_L”* semnifică faptul că semnalul este activ pe 0. Dacă se dă click dreapta pe oricare dintre semnale se poate alege baza numerică de reprezentare a semnalului , în situația de față se recomandă *Decimal (unsigned)*. Se salvează şi se închide editorul.

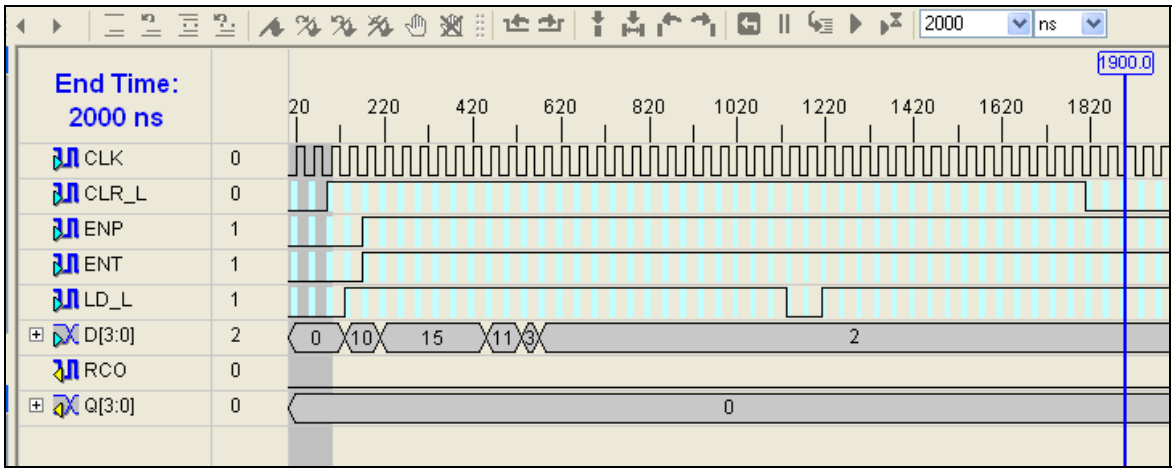


Figura 12.

În fereastra *Sources, Sources for* se selectează *Behavioral Simulation, Xilinx ISE Simulator, Simulate Behavioral Model.*
În figura 13, sunt prezentate rezultatele simulării.

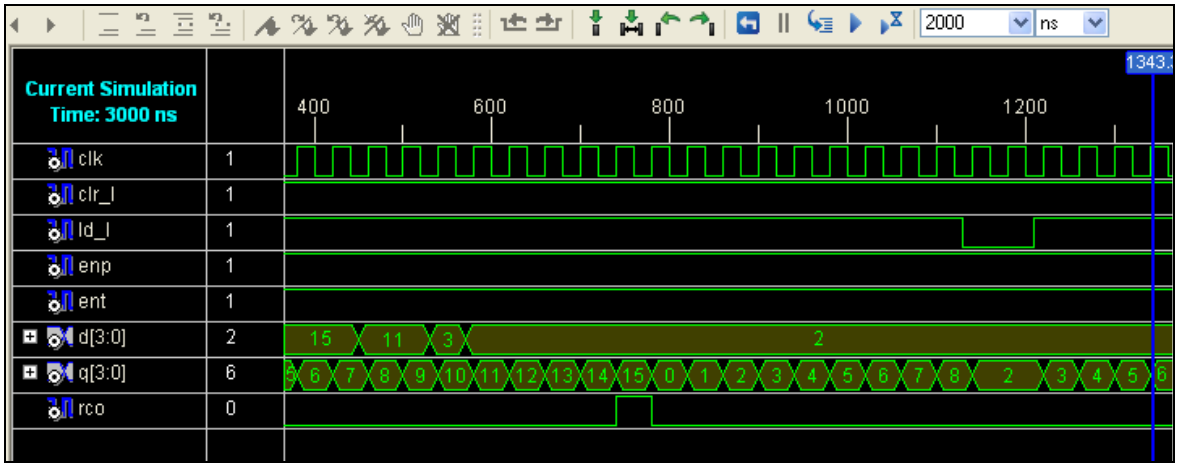


Figura 13.

Se poate observa că numărătorul se incrementează, iar în momentul în care semnalul de încărcare *ld_l* este 0 se încarcă valoarea de la intrarea de date (valoarea 2) şi incrementarea continuă de la aceasta. Urmărind codul VHDL verificați toate modurile de funcționare ale numărătorului, modificând starea stimulilor de la intrare.
Închideți simularea.

Pasul 4: Crearea constrângerilor de timp

În continuare se vor specifica constrângerile de timp cu privire la frecvența de lucru la care ne aşteptăm să funcționeze numărătorul implementat în FPGA şi de asemenea cu privire la întârzierile de la pinii circuitului FPGA. Cu alte cuvinte se vor specifica momentele în care circuitul FPGA este pregătit să primească date la pini şi cât timp să păstreze date valide la pini.
Se revine *Sources for, Synthesis...,* se selectează codul VHDL, iar în fereastra *Processes* se alege *User Constraints, Create Timing Constraints.* Va fi rulată etapa de translatare din faza de implementare şi se va crea automat un fişier UCF, se dă click *Yes* pe fereastra care apare. Fişierul UCF se adaugă proiectului şi devine vizibil în fereastra cu fişiere sursă.
Se va deschide editorul de constrângeri şi se dă click pe tabul *Global*, se selectează câmpul *Period* şi se dă click pe simbolul  din bara de meniuri sau dublu click în câmpul *Period*, se specifică perioada 40 ns (frecvența tactului va fi 25 MHz) şi se lasă factorul de umplere 50 %, vezi figura 14, click OK.

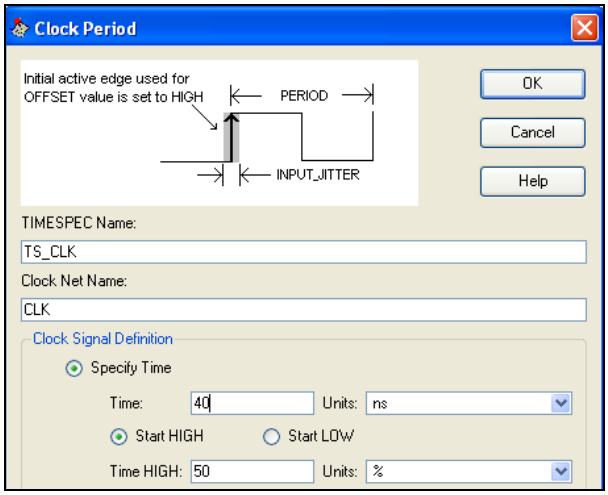


Figura 14.

Dublu click în câmpul *Pad to setup*, se introduce valoarea 10 ns în câmpul OFFSET, se setează cu cât timp înainte de a avea un front crescător datele să fie stabile, vezi figura 15, click OK.
Dublu click în câmpul *Clock to pad,* se introduce valoarea 10 ns în câmpul OFFSET, se setează cât timp datele rămân valide la ieşire după un front crescător de tact, vezi figura 16, click OK.

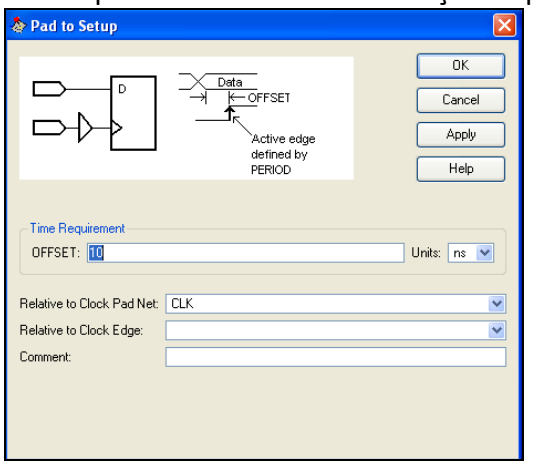


Figura 15.

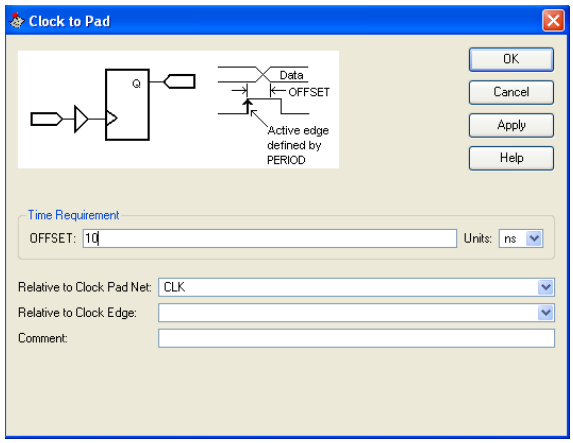


Figura 16.

Toate constrângerile vor apărea în fereastra din stânga jos a editorului schematic, click *Save* şi închideți editorul de constrângeri.

Pasul 5: Implementarea proiectului şi verificarea constrângerilor

În *Sources, Sources for* se revine din nou la *Synthesis/Implementation.*
Implementarea se realizează conform procedurii din laboratorul 1.
Se identifică *Static Timing Report* în fereastra *Design Sumarry* şi se alege *Timing Sumarry,* vezi figura 17, se poate observa că frecvența maximă de lucru este 176 MHz, constrângerea de la intrare (4,59 ns) este respectată fiind mai mică de 10 ns, dar la ieşire (10, 87 ns) aceasta este depăşită, astfel că pentru a finaliza implementarea cu succes aceasta va trebui modificată la o valoare mai mare.
Modificați şi reimplementați. De asemenea pentru a vedea ce se întâmplă se poate creşte şi constrângerea de clock la 200 MHz.

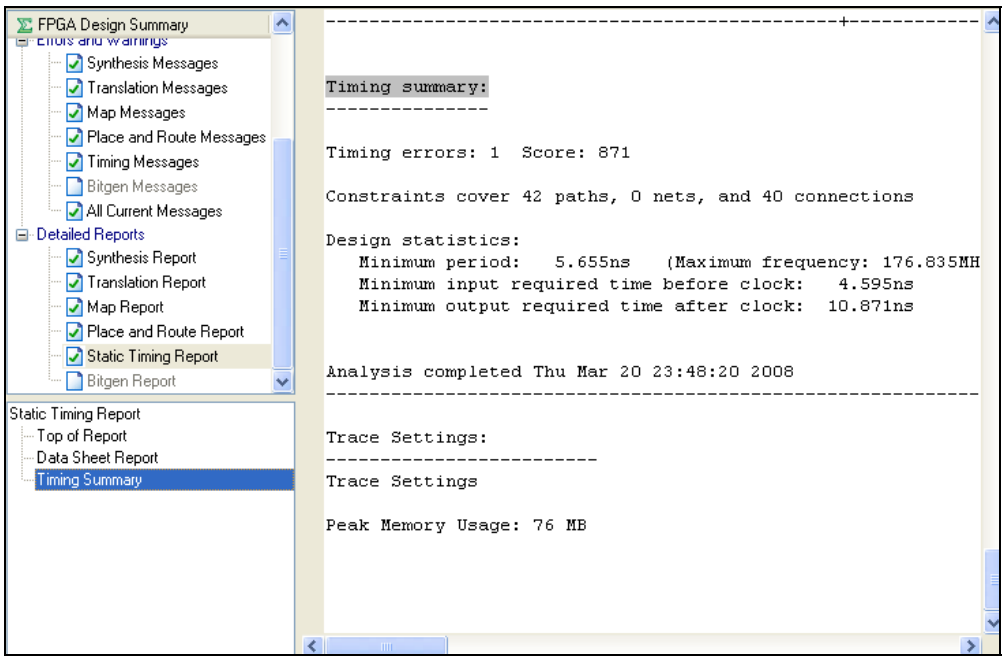


Figura 17.

Pasul 6: Definirea constrângerilor la pini

Conform procedurilor din laboratoarele anterioare definiți constrângerile la pini circuitului FPGA, vezi tabelul cu pini din lab 1.
Se va alege pinul corespunzător pentru semnalul de clock, se va alege un comutator pentru clear, unul pentru load, două pentru validări şi patru pentru biții de date, vor fi folosite astfel toate cele 8 comutatoare (SW7....SW0)).
Pentru ieşiri stabiliți constrângeri la pinii conectați la LED-urile de pe placă, 4 pentru date 1 pentru transport.
Nu uitați că pentru a afişa corect starea, LED-urile au nevoie şi de semnalul de validare LEDG, vezi documentația plăcii DIO4!!!

Activități suplimentare

- Se vor repeta operațiunile descrise anterior pentru fiecare din codurile VHDL din figurile 3,4, 5. Pentru fiecare cod în parte se va crea un proiect cu numele entității, în subdirectorul LAB4.
- Se vor implementa constrângeri de timp şi de loc pentru fiecare proiect şi se vor analiza fişierele raport.

Indicaţie!!!

Datorită faptului că semnalul de clock din hardware este de 50 MHz, dacă nu se face o divizare a acestuia, la implementarea în hardware secvența de numărare, pentru numărător sau de deplasare a biților în cazul registrelor, nu va putea fi urmărită.
Se recomandă divizarea semnalului de clock, secvența de cod corespunzătoare, pentru numărărtor este prezentată în figura 18, similar se procedează şi pentru celelelate circuite secvențiale.

```
architecture num_sin_arch of num_sin is
signal IQ: UNSIGNED (3 downto 0);
signal div: std_logic_vector (24 downto 0);
signal C: std_logic;
begin
--se face divizarea semnalului de clock cu 2**25
p1:process(clk)
        begin
        if CLK'event and CLK='1' then
                div <= div+1;
        end if;
        C <= div(24);
        end process;
P2:process (C, ENT, IQ)
        begin
  if C'event and C='1' then
    if CLR_L='0' then IQ <= (others => '0');
    elsif LD_L='0' then IQ <= D;
    elsif (ENT and ENP)='1' then IQ <= IQ + 1;
    end if;
  end if;
  if (IQ=15) and (ENT='1') then RCO <= '1';
  else RCO <= '0';
  end if;
  Q <= IQ;
 end process;

 LEDG <= '1';
end num_sin_arch;
```

Figura 18.

Circuite Logice Programabile

LABORATOR 5

PROIECTAREA IERARHICĂ, UTILIZAREA TEMPLATE-URILOR DE COD VHDL

SCOPUL LUCRĂRII

Obiectivul acestei lucrări constă în extinderea cunoştinţelor în ceea ce priveşte proiectarea ierarhică, crearea proiectelor mixte care să conţină atât logică combinaţională cât şi secvenţială. Se va studia şi modalitatea de integrare într-o descriere VHDL a secvenţelor de cod predefinite.

IINTRODUCERE TEORETICĂ

Vezi lucrările anterioare.

Desfăşurarea lucrării

Se va crea un proiect de tip schematic care va include trei componente, divizor de clock, numărător cu 16 stări şi decodificator binar-7 segmente. Module vor fi scrise în VHDL utilizând *Language Templates* , acolo unde este cazul . Pentru fiecare componentă în parte se va crea un test bench separat pentru a se verifica funcţionarea acestora. Din codurile VHDL se vor genera macro-simboluri în schematic, vezi laborator 3. Simbolurile schematic vor fi conectate, iar pentru componenta rezultată se va crea un testbench şi sa va verifica funcţionarea acesteia. Se vor aplica constrângerile corespunzătoare la pinii circuitului FPGA şi se va face implementarea.

Circuitul rezultat trebuie să poată fi resetat la valoarea 0, trebuie să permită selectarea secvenței de numărare up/down şi să poată afişa pe modulul de afişare 7 segmente fiecare stare.

Pasul 1: Crearea proiectului.

În directorul personal se va crea un subdirector cu numele *lab5,* aici vor fi salvate toate proiectele ce vor fi create în cadrul acestei lucrări.
Se va crea un proiect de tip schematic cu numele *num_dcd,* atenție la directorul de lucru.

Pasul 2: Proiectarea ierarhică, crearea simbolurilor în schematic

Numărătorul
Se descrie modulul de numărare. Se adaugă o nouă sursă VHDL la proiect *num* şi se definesc porturile: *clock, reset, dir* (intrări) şi Q ieşire de tip magistrală pe 4 biți (se selectează opțiune *bus 3 ->0*).
Ne poziționăm cu cursorul în interiorul codului VHDL din fişierul *num,* în cadrul arhitecturii după *begin*.
Se va accesa biblioteca cu tipare de cod VHDL, meniul *Edit, Language Templates* sau din bara de meniuri simbolul  şi se selectează *VHDL -> Synthesis Construct -> Coding Examples ->Counters -> Binary ->Up/Down Counters->w/CE and Async Active High Reset*, vezi figura 1.

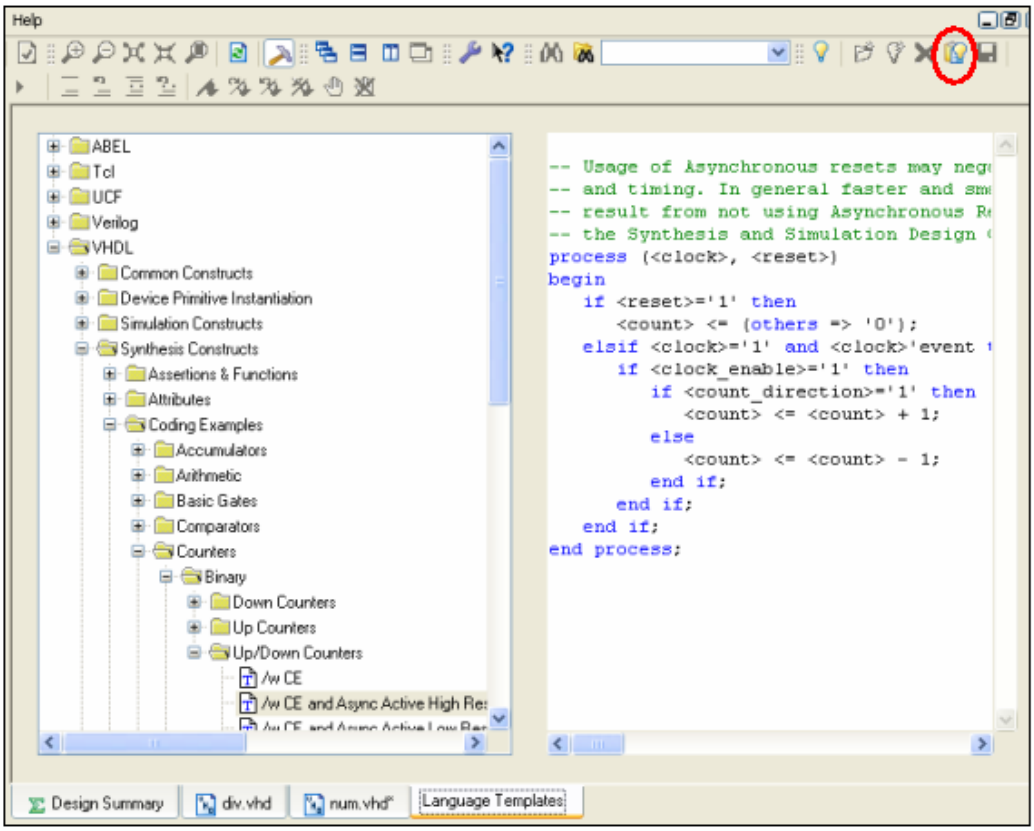


Figura 1.

Din bara de meniuri se alege opțiunea *Use Template in File*, pictograma  , vezi figura 2.
În acest moment secvența de cod VHDL ar trebui să fie inserată în codul din fişierul *num*.
În continuare se şterge comentariile inutile şi se fac câteva modificări de nume şi se adaugă semnale suplimentare în codul VHDL, forma finală a codului VHDL pentru numărător este prezentată în figura 3. Se creează simbolul schematic.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity num is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           dir : in  STD_LOGIC;
           num : out  STD_LOGIC_VECTOR (3 downto 0));
end num;
architecture Behavioral of num is
signal count: std_logic_vector (3 downto 0);
begin
process (clock, reset)
begin
   if reset='1' then
      count <= (others => '0');
   elsif clock ='1' and clock'event then
         if dir ='1' then
            count <= count + 1;
         else
            count <= count - 1;
         end if;
   end if;
end process;
num <= count;
end Behavioral;
```

Figura 2

Se creează un testbench conform celui din figura 3.

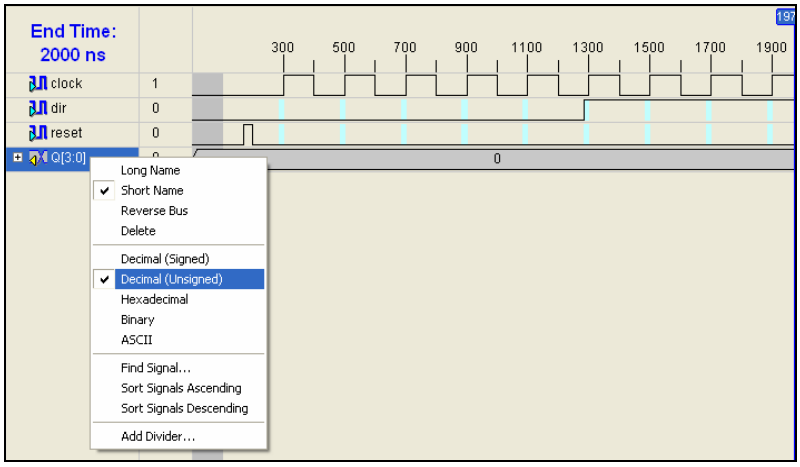


Figura 3

Se face testarea componentei *num,* rezultatele simulării trebuie să fie similare cu cele din figura 4.

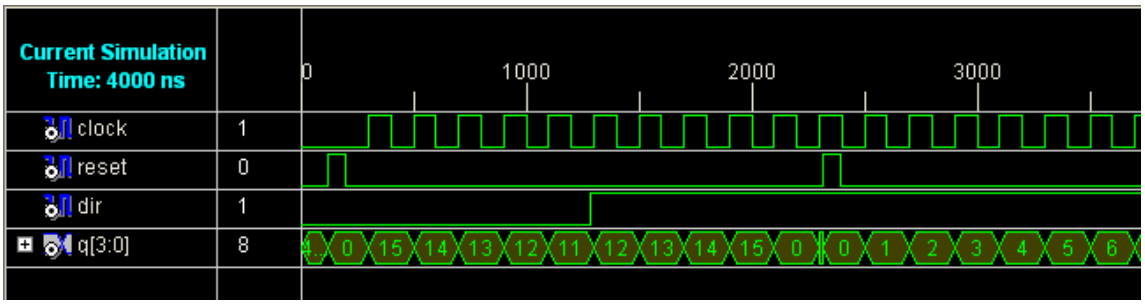


Figura 4

Decodificatorul

Se descrie modulul de decodificare binar 7 segmente. Se adaugă o nouă sursă VHDL la proiect *dcd* şi se definesc porturile: *hex* intrare de tip magistrală pe 4 biți şi LED ieşire de tip magistrală pe 7 biți .
Ne poziționăm cu cursorul în interiorul codului VHDL din fişierul *dcd,* în cadrul arhitecturii după *begin*.
Se va accesa biblioteca cu tipare de cod VHDL, meniul *Edit, Language Templates* sau din bara de meniuri simbolul  şi se selectează *VHDL -> Synthesis Construct -> Coding Examples ->Misc -> 7-Segment Dispaly ..*, vezi figura 5.

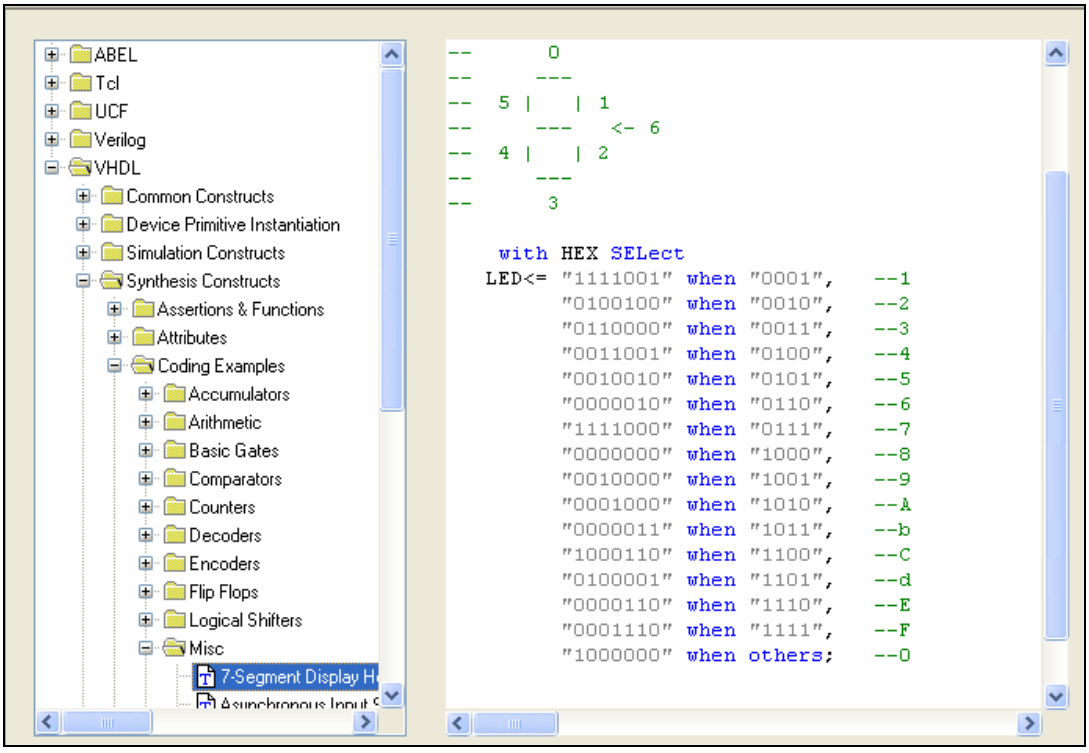


Figura 5.

Forma finală a codului VHDL pentru decodificatorul 7 segmente este prezentată în figura 6.
Se creează simbolul schematic.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dcd is
    Port ( hex : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (6 downto 0));
end dcd;
architecture Behavioral of dcd is
begin
     with HEX SELect
   LED<= "1111001" when "0001",    --1
         "0100100" when "0010",    --2
         "0110000" when "0011",    --3
         "0011001" when "0100",    --4
         "0010010" when "0101",    --5
         "0000010" when "0110",    --6
         "1111000" when "0111",    --7
         "0000000" when "1000",    --8
         "0010000" when "1001",    --9
         "0001000" when "1010",    --A
         "0000011" when "1011",    --b
         "1000110" when "1100",    --C
         "0100001" when "1101",    --d
         "0000110" when "1110",    --E
         "0001110" when "1111",    --F
         "1000000" when others;    --0
 end Behavioral;
```

Figura 6.

Pasul 3: Crearea testbench-ului şi simularea funcțională a modulelor de numărare şi decodificare

Pentru a verifica şi funcționarea decodificatorului, o abordare ar fi conectarea împreună cu numărătorul, într-un proiect de tip schematic, aceasta reduce numărul de intrări la care se aplică stimuli.
Astfel se va crea şi adăuga la proiect fişierul de tip schematic pe care îl vom numii *afişaj*.
Deschidem acest fişier şi în biblioteca specifică proiectului şi trebuie să regăsim simbolurile *num* şi *dcd* create în etapele anterioare. Conectăm cele două componente conform figurii 7. În acest moment structura proiectului nostru trebuie să fie ca şi cea din figura 8.

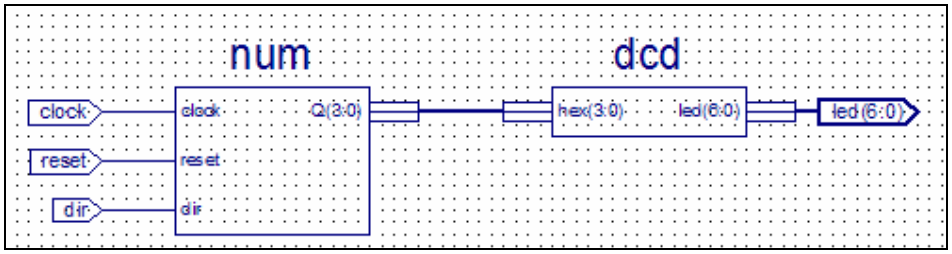


Figura 7.

Atenție numele unei instanțe trebuie să fie diferit de numele simbolului pe care îl instanțiază, de exemplu, numele instanței este *num_4*, iar numele simbolului este *num*, la fel şi la decodificator, vezi figura 8.

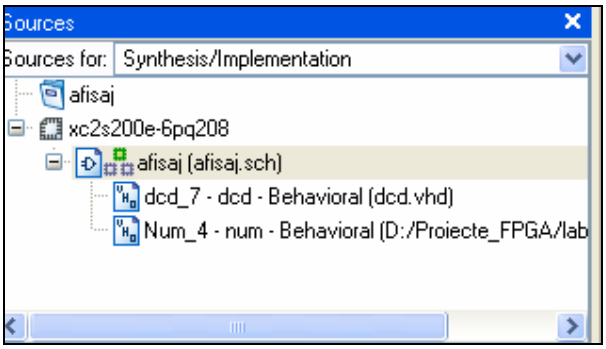


Figura 8.

Pentru proiectul afişaj se poate crea în acest moment un testbench asociat entității *top*, *afişaj*. Numele testbenchului va fi *afisaj_tb* si este prezentat în figura 9.

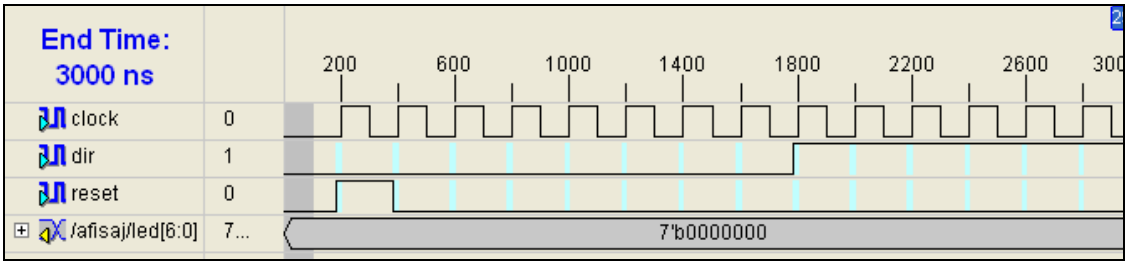


Figura 9.

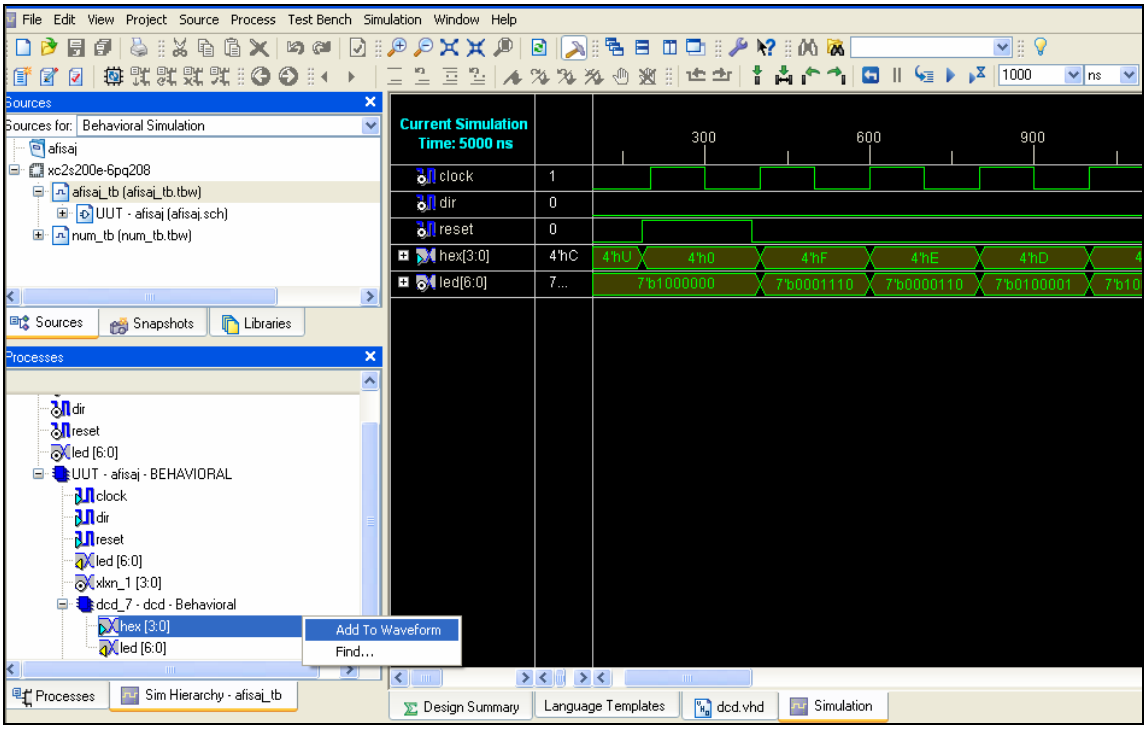


Figura 10

O parte din rezultatul simulării poate fi vizualizat în figura 10. Pentru a putea verifica corectitudinea simulării trebuie adăugate semnalele care provin de la numărător astfel în fereastra *Processes* se va expanda instanța *dcd_7*, click dreapta pe intrarea *hex, Add to Waveform* şi automat acest semnal va fi inserat în fereastra de simulare. Se compară stările semnalelor *hex, led* cu cele din figura 6.

Pasul 4: Implementarea şi testarea proiectului

Înainte de a crea fişierul de constrângeri şi de a implementa proiectul mai trebuie proiectat un bloc, cel de divizare a semnalului de clock.
La fel ca şi în laboratorul anterior pentru a putea urmării secvența de numărare semnalul de clock trebuie divizat, în acest caz cu 2^{24} .

Blocul de divizare

Blocul va avea o singură intrare şi o singură ieşire.
În continuare se va adăuga o nouă sursă proiectului, *Project, New Source,* de tipul *VHDL,* cu numele *div,* intrare *clock,* ieşirea *clock_div*. Utilizând *wizardul* se vor definii cele două porturi, iar în arhitectură se va insera secvența de cod similară cu cea din laboratorul 4, figura 18, procesul p1. Codul rezultat va fi de forma celui din figura 11. Se creează simbolul schematic.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity div is
    Port ( clock : in  STD_LOGIC;
           clock_div : out  STD_LOGIC);
end div;

architecture Behavioral of div is

   signal div: std_logic_vector (24 downto 0);

   begin
--se face divizarea semnalului de clock cu 2**24
   p1:process(clock)
      begin
      if clock'event and clock='1' then
         div <= div+1;
      end if;
      clock_div <= div(24);
end process;

end Behavioral;
```

Figura 11

Ne întoarcem în fişierul schematic *afişaj* şi adăugăm şi simbolul nou creat *div*, conform figurii 12.

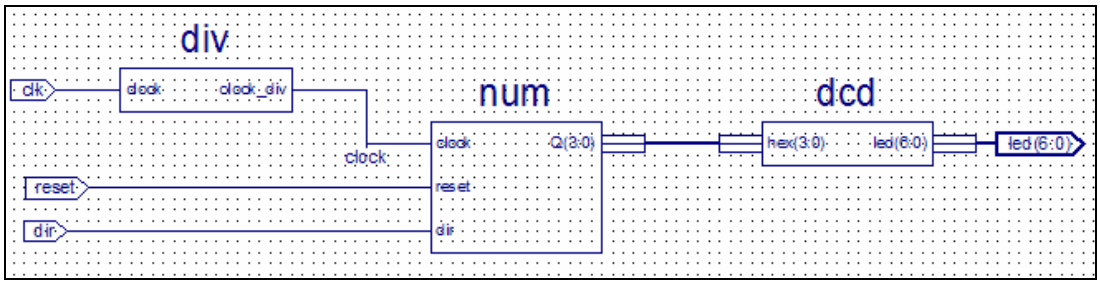


Figura 12.

În acest moment suntem gata să creăm fişierul de constrângeri, vezi figura 13.

```
NET "clk"   LOC = "P182"  ;#
NET "dir"   LOC = "P23"   ; #SW1
NET "led<0>"  LOC = "P22"  ;
NET "led<1>"  LOC = "P20"  ;
NET "led<2>"  LOC = "P17"  ;
NET "led<3>"  LOC = "P15"  ;
NET "led<4>"  LOC = "P10"  ;
NET "led<5>"  LOC = "P8"  ;
NET "led<6>"  LOC = "P6"  ;
NET "reset"  LOC = "P3"  ; #BTN1
```
Figura 13.

Se salvează fişierul de constrângeri şi se poate face sinteza, implementarea, generarea şi programarea circuitului FPGA.
Testați proiectul implementat, apăsând butonul BTN1 pentru reset şi schimbând starea comutatorul SW1.

Activități suplimentare

- Modificați frecvența de afişare, afişați punctul zecimal, activați şi dezactivați fiecare caracter în parte.
- Să se proiecteze un circuit care să permită introducerea de la comutatoarele plăcii de test, a patru combinații pe patru biți şi vizualizarea pe cele 8 LED-uri a deplasării în buclă a acestora atât spre stânga cât şi spre dreapta.

Indicație

Proiectul va include un bloc de multiplexare şi registre de deplasare. Pentru descrierea VHDL, vezi laboratoarele anterioare şi *Language Templates.*

Circuite Logice Programabile

LABORATOR 6

PROIECTAREA AUTOMATELOR DE STĂRI

SCOPUL LUCRĂRII

Scopul acestei lucrări constă în este introducerea conceptului de automat de stări sau FSM (*Finite State Machine*) şi prezentarea modelelor de bază. Exemplificarea discuțiilor care vor avea loc în introducerea teoretică se va face pe un model de numărător, construit cu bistabile de tip D şi având şi o parte de logică combinațională. De asemenea va fi făcută şi o introducere în utilizarea editorului FSM din pachetul de programe Xilinx.

O detailare pe un proiect mai complex a acestui mod de proiectare se va face în lucrarea următoare.

IINTRODUCERE TEORETICĂ

Prezentarea conceptului şi a configurațiilor de bază FSM

Se face următoarea convenție, pe parcursul întregii lucrări, în locul expresiei "automat de stări" se va folosii abrevierea FSM.

Un FSM simplu foloseşte unul sau mai multe bistabile pentru a-şi păstra stările interne (vezi figura 1). Şirul de biții de 1 şi 0 de la ieşirea bistabilului reprezintă starea curentă a FSM-ului. Pe frontul crescător a unui impuls de tact starea curentă (*current sate*) este înlocuită de starea următoare (*next state*). Pentru calcularea stării următoare din starea curentă se foloseşte un circuit combinațional. Un exemplu de FSM de tipul celui descris mai sus este un numărător. Acesta îşi stochează starea curentă în bistabilii de tip D din care este alcătuit, iar această valoare va fi trecută mai departe la un circuit combinațional (un simplu circuit de incrementare) care va calcula starea următoare.

O variantă mai utilă de FSM este cea prezentată în figura 2, aceasta interacționează cu mediul exterior prin intermediul unei intrări suplimentare. Această intrare permite mărimilor introduse din exterior să interacționeze cu starea curentă în blocul combinațional, afectând astfel starea următoare a FSM-ului. Un exemplu pentru acest caz este un numărător cu intrare de încărcare. Un astfel de numărător permite încărcarea unei valori în timpul procesului de numărare, astfel că numărarea va continua de la valoarea încărcată. În acest caz rolul intrări externe este de a permite încărcarea noi valori de la care să înceapă numărarea precum şi „citirea" semnalului de comandă care va indica momentul în care să se facă încărcarea valori prestabilite.

Starea curentă poate fi utilizată direct ca şi mărime de ieşire a FSM-ului (de exemplu valoarea binară de la ieşirea unui numărător),de asemenea această stare poate fi mai întâi procesată prin intermediul unui circuit combinațional. O maşină care are o astfel de

configurație se numeşte *maşină Moore* (vezi figura 3). Un exemplu de maşină Moore este un numărător a cărui ieşire este conectă la un decodoficator 7 segmente cu afişaj. Astfel că valoarea binară de la ieşirea numărătorului va fi convertită şi afişată mult mai sugestiv.

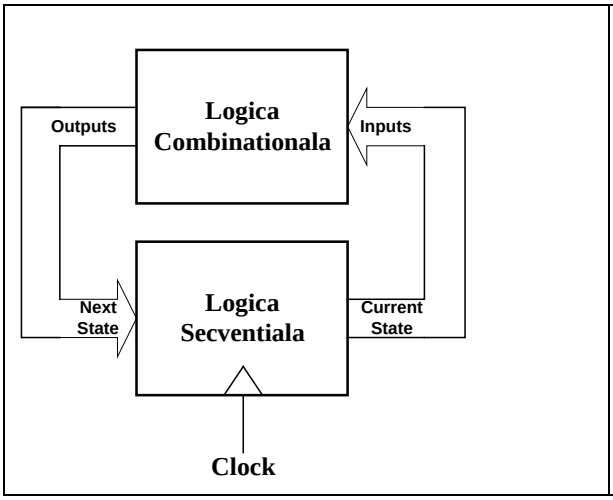


Figura 1. Un model simplu de FSM

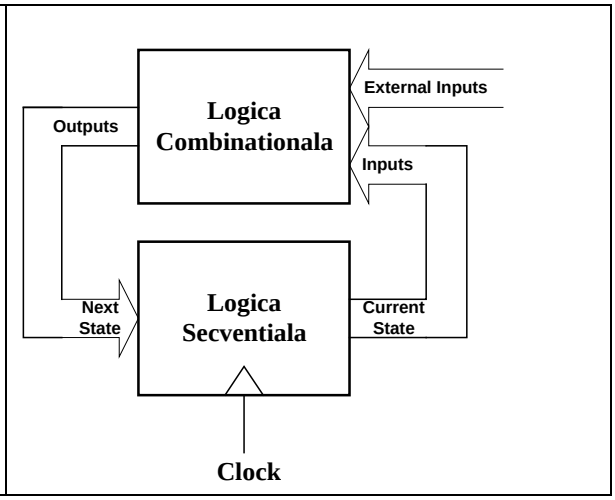


Figura 2. FSM cu intrări externe

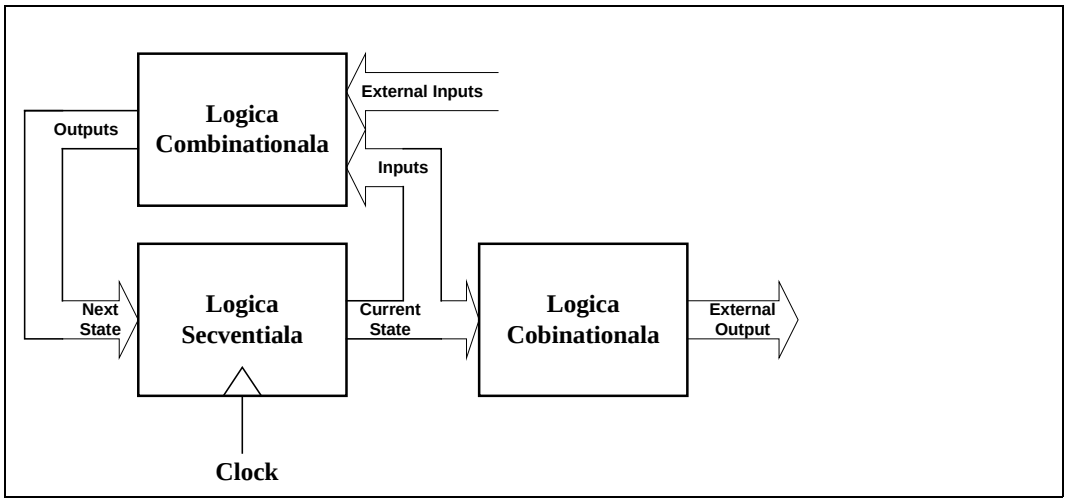


Figura 3. FSM de tip Moore

O formă şi mai flexibilă de FSM este cea a cărei intrări externe sunt conectate şi la modulul de ieşire combinațional. În acest caz mărimile de ieşire din FSM vor fi calculate în funcție atât de starea curentă cât şi de valorile de la intrarea externă. Acest tip de configurație se numeşte *maşină Moore,* vezi figura 4*.* Exemplul în acest caz este tot numărătorul a cărui ieşiri sunt conectate la un decodor 7-segmente cu afişaj, dar al cărui afişaj poate fi activat sau dezactivat, aceasta neafectând procesul de numărare şi nici pe cel de decodificare, atât doar că valoarea decodificată nu va putea fi vizualizată.

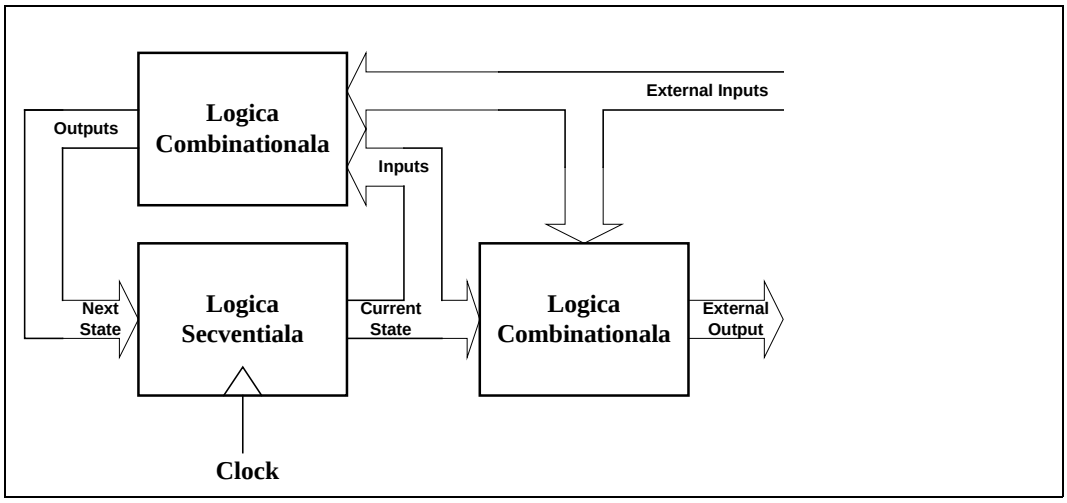


Figura 4. FSM de tip Mealy

Etapele proiectării unui automat de stări (FSM)

Pentru a face mai sugestivă descrierea etapelor de proiectare a unui FSM vom folosi ca şi model proiectat un simplu numărător modulo 3.

Etapele proiectării unui FSM sunt următoarele:

1. Stabilirea specificațiilor pentru modelul proiectat;
2. Definirea intrărilor, ieşirilor şi a stărilor intermediare;
3. Atribuirea unei valori binare fiecărei stări, valoare ce va fi stocată într-un bistabil;
4. Definirea tuturor tranzițiilor posibile între stări;
5. Definirea unui tabel de adevăr care să conțină valorile stării curente, a intrărilor şi a ieşirilor;
6. Definirea ecuațiilor booleene corespunzătoarelor tabelului de adevăr;
7. Simularea modelului proiectat la nivel de porți logice;
8. Construirea circuitului digital;
9. Verificarea circuitului digital.

În cadrul primei etape, vom stabili modulul numărătorului (modulo 3), funcțiile pe care le are (numărare *up/down*). În etapa a doua stabilim intrările şi ieşirile numărătorului (intrarea de tact (*Clk*), intrarea de reset (*Reset*), intrarea care stabileşte sensul de numărare (*up*), ieşire pe 2-biți (*Output*) şi numărul de stări. Fiind vorba de numărător modulo 3 (ciclu de numărare este 0,1,2,0..) este evident ca numărul de stări distincte va fi egal cu trei. După determinarea numărului de stări în etapa a treia se va acorda câte o valoare binară fiecărei stări. Un set de bistabili vor stoca aceste valori. În cazul numărătorului se folosesc chiar valorile intermediare din procesul de numărare. Odată definite stările trebuie să determinăm modul în care se face tranziția între acestea în funcție de valorile de la intrările numărătorului, de asemenea trebuie specificate şi valorile la ieşirea din fiecare stare. Sistematizarea tuturor stărilor şi tranzițiilor între acestea se poate face conform diagramei din figura 5.

Mediul Xilinx ISE are un editor FSM care operează exact cu acest tip de diagrame, astfel că vom putea face descrierea numărătorului conform figurii 5.

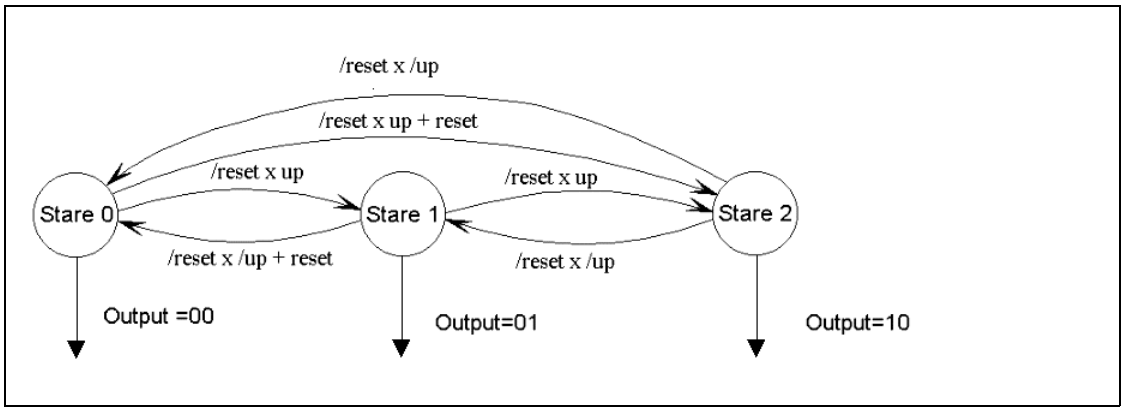


Figura 5 Diagrama de funcționare a numărătorului modulo 3

Realizând proiectarea numărătorului în mediul Xilinx etapele 5,6 vor fi parcurse în urma sintezei circuitului descris cu editorul FSM. Astfel se va trece la următoarele etape de proiectare, cea de simulare (etapa 7), implementare a circuitului în FPGA (etapa 8) şi verificarea circuitului implementat (etapa 9) prin atribuire de semnale intrărilor şi citirea semnalului de ieşire.

Desfăşurarea lucrării

StateCAD este un modul integrat în sistemul de proiectare Xilinx WebPACK care permite crearea şi editarea sub formă grafică a automatelor cu stări finite. Acest modul permite definirea stărilor, a semnalelor de ieşire care trebuie activate în fiecare stare (acțiunile stărilor) şi a tranzițiilor dintre stări. Acțiunile stărilor şi condițiile tranzițiilor pot fi specificate prin ecuații cu o sintaxă simplă, fără a fi necesară cunoaşterea limbajelor de descriere hardware. Modulul StateCAD permite şi adăugarea la automatele de stare create a unor circuite logice uzuale, ca porți logice, multiplexoare, sumatoare, comparatoare, bistabile, registre de deplasare şi numărătoare.

StateCAD permite validarea diagramelor de stare înainte de simularea funcționării acestora. Proiectantul poate descoperi condiții nedeterminate, erori de sintaxă şi porțiuni specificate incomplet din cadrul diagramelor de stare. După validarea unei diagrame, aceasta poate fi compilată în codul echivalent într-un limbaj de descriere hardware (VHDL, Verilog sau ABEL). Codul obținut poate fi utilizat ulterior pentru simularea automatului de stare, pentru sinteza şi implementarea automatului, sau pentru crearea unui simbol (macro) din automatul de stare, care poate fi plasat apoi într-o schemă. După crearea şi validarea unui automat de stare, se poate verifica funcționarea acestuia utilizând programul StateBench. Acest program permite crearea unui banc de test pentru automatul care trebuie testat, prin specificarea semnalelor de intrare ale automatului şi a semnalelor de ieşire care trebuie generate în fiecare stare. StateBench utilizează bancul de test creat pentru simularea funcționării automatului, indicând diferențele față de rezultatele aşteptate. După simulare, se poate genera un banc de test care conține constrângeri de timp. Acest banc de test se poate utiliza pentru verificarea constrângerilor de timp după sinteza automatului.

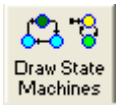
Pasul 1: Crearea proiectului şi lansarea în execuție a programului StateCAD

În directorul personal se va crea un subdirector cu numele *la6,* aici vor fi salvate toate proiectele ce vor fi create în cadrul acestei lucrări. Se va crea un proiect cu numele *fsm,* de tip HDL, atenție la directorul de lucru.

Pentru lansarea în execuție a programului StateCAD şi crearea unui fişier pentru diagrama de stare, se procedează astfel:
1. Se selectează comanda *Project → New Source*.
2. În fereastra de dialog *New*, se selectează opțiunea *State Diagram*.
3. În câmpul *File Name* se introduce numele fişierului în care se va păstra diagrama de stare, de exemplu, *num3*.
4. Se selectează butonul *Next*.
5. În fereastra de dialog *New Source Information*, se selectează butonul *Finish*.
Se va lansa în execuție programul StateCAD într-o nouă fereastră.

Pasul 2: Crearea automatului de stare

Pentru crearea automatului de stare se va folosi utilitarul *State Machine Wizard*.

1. Se selectează butonul *Draw State Machines* 

. Se va deschide fereastra *State Machine Wizard*.
2. În câmpul *Shape of state machine* se selectează opțiunea *Row* pentru plasarea stărilor automatului pe linie.
3. În câmpul *Number of States* se selectează 3 ca număr de stări ale automatului. Ulterior, se pot introduce stări suplimentare.
4. Se selectează butonul *Next*. Se va deschide fereastra de dialog *Reset The State Machine*.
5. În câmpul *Reset Mode* se selectează opțiunea *Asynchronous* pentru a se genera o logică asincronă de resetare a automatului de stare.
6. Se selectează butonul *Next*. Se va deschide fereastra de dialog *Setup Transitions*.
7. În câmpul *Add Transitions* se selectează tipul tranzițiilor care vor fi prezente în diagrama de stare. În mod implicit, va fi selectată doar opțiunea *Next*, opțiunile *Loop back* şi *Previous* fiind neselectate. Astfel, diagrama va conține doar tranziții de la fiecare stare la starea următoare a acesteia. Se păstrează aceste setări.
8. Se selectează butonul *Finish*. În fereastra editorului StateCAD se va afişa un dreptunghi cu conturul de culoare verde, care indică marginile diagramei.
9. Se deplasează dreptunghiul în zona centrală a ferestrei şi se execută un clic cu butonul din stânga. Se va afişa diagrama de stare cu forma ilustrată în figura 6.

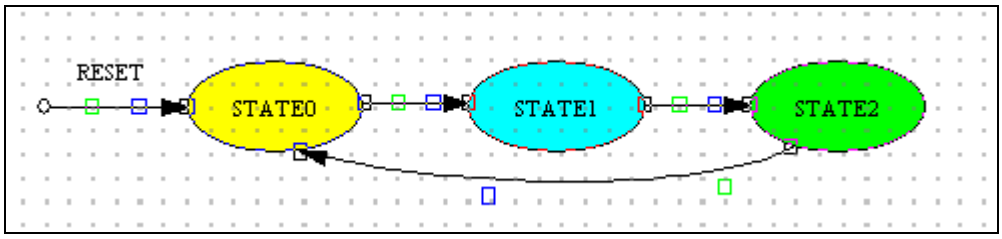


Figura 6.

Pasul 3: Editarea stărilor

În continuare se modifică numele stărilor astfel încât acestea să reflecte funcția lor şi se specifică semnalele de ieşire care trebuie activate în stările respective. Pentru aceasta se execută următoarele operații:

1. Se execută un clic dublu pe simbolul stării STATE0. Se va afişa fereastra de dialog *Edit State*. În câmpul *State Name* se introduce numele S0. În câmpul *Outputs* se introduc următoarele ecuații indicând valorile cu care se inițializează registrele, *Q="00"* şi se selectează butonul *OK;*

2 Se execută un clic dublu pe simbolul stării STATE1. Se va afişa fereastra de dialog *Edit State*. În câmpul *State Name* se introduce numele S1. În câmpul *Outputs* se introduc următoarele ecuații indicând valorile cu care se inițializează registrele, *Q="01"* şi se selectează butonul *OK;*

3. Se execută un clic dublu pe simbolul stării STATE2. Se va afişa fereastra de dialog *Edit State*. În câmpul *State Name* se introduce numele S2. În câmpul *Outputs* se introduc următoarele ecuații indicând valorile cu care se inițializează registrele, *Q="10"* şi se selectează butonul *OK;*

Pasul 4: Adăugarea tranzițiilor suplimentare

Pentru completarea diagramei de stare cu tranzițiile care nu au fost generate în mod automat, se procedează astfel:

1. Se selectează butonul *Draw Transitions* , aflat în partea stângă a ecranului,  .

2. Se selectează starea **S0** prin execuția unui clic în interiorul stării, iar apoi se execută un nou clic în interiorul stării, în apropierea marginii de sus. Va apare un pătrat de culoare roşie pe marginea simbolului stării, iar un alt pătrat va urmări deplasarea cursorului.

3. Se deplasează cursorul deasupra stării **S2**. Va apare un pătrat de culoare roşie pe marginea simbolului stării şi o linie între stările **S0** şi **S2**. Se fixează poziția punctului de capăt al liniei şi se execută un clic în poziția respectivă. Dacă o linie de tranziție nu a fost adăugată în mod corespunzător, aceasta va fi ştearsă prin apăsarea tastei **Delete**, iar apoi va fi adăugată din nou.

4. În acelaşi mod se adaugă o tranziție de la starea **S1** la starea **S0**.

5. Se procedează similar pentru adăugarea unei tranziții de la starea **S2** la starea **S1**. Pentru transformarea liniei de tranziție într-o linie curbă, se execută un clic pe unul din cele două puncte de control din interiorul liniei (puncte indicate prin pătrate cu contururi de culoare roşie), iar apoi se repoziționează punctul prin deplasarea cursorului în jos, menținând butonul apăsat. Se procedează similar cu al doilea punct de control.

6. Se poate adaugă o tranziție de buclare pentru o starea. Pentru aceasta, se execută un clic dublu în interiorul simbolului acestei stări, în apropierea marginii de sus. Editorul va adăuga linia pentru tranziția de buclare (care începe şi se termină în starea respectivă) şi va adăuga condiția @ELSE pentru această tranziție, nu este cazul în acest punct al lucrării.

7. Se selectează butonul *Pointer*  , pentru a termina adăugarea tranzițiilor.

În acest moment diagrama de stări trebuie să arate ca şi cea din figura 7.

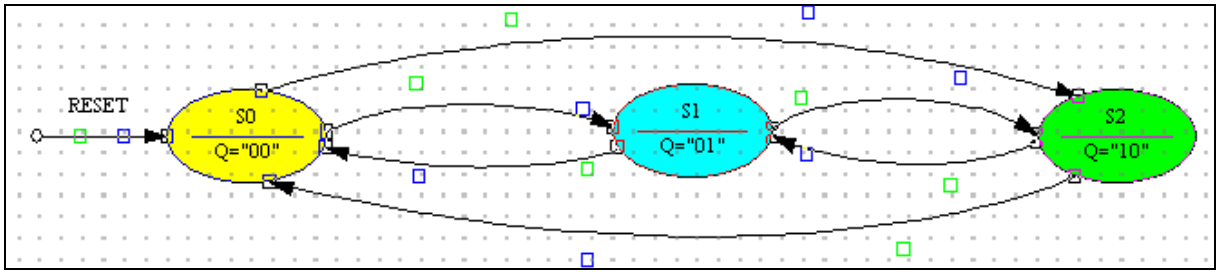


Figura 7.

Pasul 5: Adăugarea condițiilor pentru tranziții

Pentru adăugarea condițiilor care determină execuția tranzițiilor, se execută următoarele operații:
1. Se execută un clic dublu pe linia tranziției de la starea **S0** la starea **S1**. Se va deschide fereastra de dialog *Edit Condition*. În câmpul *Condition* se introduce condiția **reset='0' and up ='1'**, iar apoi se selectează butonul *OK*. Se deplasează dreptunghiul de culoare roşie deasupra liniei de tranziție.
2. În mod similar se execută operațiile menționate anterior pentru toate semnalele de tranziție stfel că diagrama finală trebuie să arate ca şi cea din figura 8.
Se salvează fişierul cu diagrama de stare prin selecția butonului *Save File*.

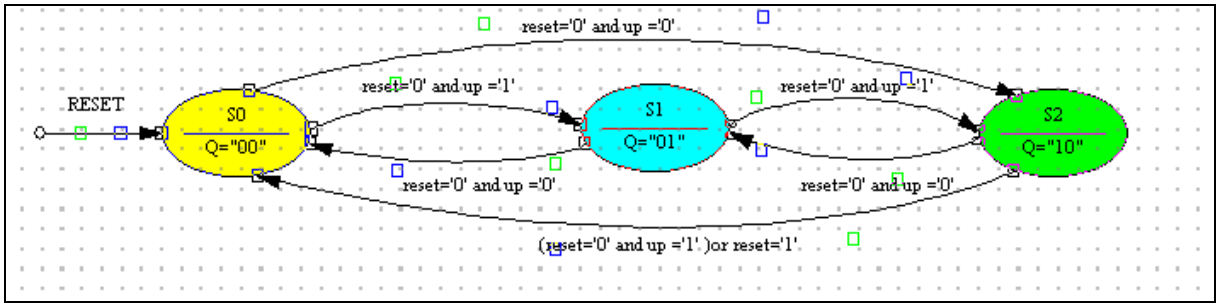


Figura 8.

Pasul 5: Adăugarea unor noi vectori

Programul StateCAD defineşte în mod automat vectorii care au fost utilizați în ecuațiile circuitelor logice adăugate pentru generarea semnalelor de ieşire. Variabilele de tip vector care nu apar în ecuațiile circuitelor logice adăugate trebuie definite în mod explicit de proiectant, specificând dimensiunea acestora. În caz contrar, aceste variabile vor fi considerate de tip bit.
În cazul acestui circuit de numărare, variabila de tip vector a căror dimensiune nu a fost specificată este Q. Pentru definirea acestei variabile ca vector, se procedează astfel:

1. Se selectează butonul *Draw Vectors*  , aflat în partea stângă a ecranului.
2. Se deplasează cursorul într-o zonă liberă de sub diagrama de stare şi se execută un clic cu butonul din stânga. Va apare simbolul unui vector **VAR0[7:0]**, cu dimensiunea 8.
3. Pentru modificarea numelui şi dimensiunii vectorului **VAR0**, se execută un clic dublu deasupra simbolului acestuia. Se va deschide fereastra de dialog *Edit Vector*.
5. În câmpul *Name* se modifică numele vectorului în **Q**, iar în câmpul *Range* se modifică domeniul biților în **1:0**. Se selectează apoi butonul *OK*. Se poate afişa o fereastră indicând faptul că variabila *Q* este utilizată. În acest caz, se selectează butonul *Yes* pentru ştergerea variabilei existente *Q* şi redenumirea vectorului în *Q*.

Pasul 5: Vizualizarea opțiunilor de configurație

În continuare se vizualizează opțiunile implicite care vor fi utilizate la compilarea automatului de stare şi se modifică una din opțiuni.
1. Se selectează comanda *Options* → *Configuration*, prin care se va deschide fereastra de dialog *Configuration*.
2. În secțiunea *Options* a acestei ferestre se validează opțiunea *Retain Output Values*, prin care semnalele de ieşire care nu sunt specificate în mod explicit într-o stare îşi vor păstra valorile de la ultima asignare.

3. În secțiunea *State Assignment* se indică modul în care se va realiza asignarea stărilor. Opțiunea selectată în mod implicit este *One Hot*, care indică metoda de asignare cu un bistabil pe stare. Se poate păstra această opțiune, sau se poate selecta o altă opțiune.
4. Se închide fereastra prin selectarea butonului *OK*.

Pasul 6: Compilarea diagramei de stare şi corectarea erorilor

Pentru compilarea diagramei de stare, se selectează butonul *Generate HDL* 

. Se va deschide fereastra *View Error* în cazul în care există erori. După corectarea acestora dacă se afişează fereastra de dialog *Delete Unused Variables*, se selectează butonul *Delete* pentru ştergerea variabilelor neutilizate. Dacă totul este corect se va deschide fereastra cu codul VHDL generat, care trebuie să fie identic cu cel din figura 9.

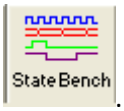
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SHELL_NUM3 IS
        PORT (CLK,RESET,up: IN std_logic;
                Q0,Q1 : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF SHELL_NUM3 IS
        TYPE type_sreg IS (S0,S1,S2);
        SIGNAL sreg, next_sreg : type_sreg;
        SIGNAL next_Q0,next_Q1 : std_logic;
        SIGNAL Q : std_logic_vector (1 DOWNTO 0);
BEGIN
        PROCESS (CLK, RESET, next_sreg, next_Q1, next_Q0)
        BEGIN
                IF ( RESET='1' ) THEN
                        sreg <= S0;
                        Q1 <= '0';
                        Q0 <= '0';
                ELSIF CLK='1' AND CLK'event THEN
                        sreg <= next_sreg;
                        Q1 <= next_Q1;
                        Q0 <= next_Q0;
                END IF;
        END PROCESS;

        PROCESS (sreg,RESET,up,Q)
        BEGIN
                next_Q0 <= '0'; next_Q1 <= '0';
                Q<=std_logic_vector'("00");
                next_sreg<=S0;
                CASE sreg IS
                        WHEN S0 =>
                                IF ( RESET='1' ) THEN
                                        next_sreg<=S0;
                                        Q <= (std_logic_vector'("00"));
                                END IF;
                                IF ( RESET='0' AND up='1' ) THEN
                                        next_sreg<=S1;
                                        Q <= (std_logic_vector'("01"));
                                END IF;
                                IF ( RESET='0' AND up='0' ) THEN
                                        next_sreg<=S2;
```

```
                                        Q <= (std_logic_vector'("10"));
                                END IF;
                        WHEN S1 =>
                                IF ( RESET='1' ) THEN
                                        next_sreg<=S1;
                                        Q <= (std_logic_vector'("01"));
                                END IF;
                                IF ( RESET='0' AND up='1' ) THEN
                                        next_sreg<=S2;
                                        Q <= (std_logic_vector'("10"));
                                END IF;
                                IF ( RESET='0' AND up='0' ) THEN
                                        next_sreg<=S0;
                                        Q <= (std_logic_vector'("00"));
                                END IF;
                        WHEN S2 =>
                                IF ( up='1' ) OR ( RESET='1' ) THEN
                                        next_sreg<=S0;
                                        Q <= (std_logic_vector'("00"));
                                END IF;
                                IF ( RESET='0' AND up='0' ) THEN
                                        next_sreg<=S1;
                                        Q <= (std_logic_vector'("01"));
                                END IF;
                        WHEN OTHERS =>
                END CASE;

                next_Q1 <= Q(1);
                next_Q0 <= Q(0);
        END PROCESS;
END BEHAVIOR;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY NUM3 IS
        PORT (Q : OUT std_logic_vector (1 DOWNTO 0);
                CLK,RESET,up: IN std_logic);
END;
ARCHITECTURE BEHAVIOR OF NUM3 IS
        COMPONENT SHELL_NUM3
                PORT (CLK,RESET,up: IN std_logic;
                        Q0,Q1 : OUT std_logic);
        END COMPONENT;
BEGIN
        SHELL1_NUM3 : SHELL_NUM3 PORT MAP (CLK=>CLK,RESET=>RESET,up=>up,Q0=>Q(0),Q1
                =>Q(1));
END BEHAVIOR;
```

Figura 9.

Pasul 7: Verificarea funcționării cu programul StateBench

Pentru verificarea funcționării circuitului de înmulțire, se va lansa în execuție programul StateBench şi se va utiliza mai întâi comanda *Verify Behavior*, iar apoi comanda *Cycle*. Pentru aceasta, se execută următoarele operații:

1. Se lansează in execuție programul StateBench prin selectarea butonului *StateBench* 

. În jumătatea de jos a ecranului se va afişa fereastra programului StateBench, diagrama de stare fiind afişată în jumătatea de sus a ecranului.

2. În fereastra programului StateBench se selectează butonul *Verify Behavior* 

. Se va deschide fereastra de dialog *Verify Behavior*.
3. În secțiunea *Set initial conditions* se pot seta valorile inițiale ale semnalelor.
4. În secțiunea *Choose the target state* se selectează una din stările S1, S2. Se selectează butonul *Go*. Programul StateBench va executa simularea cu valorile inițiale specificate ale semnalelor până când se ajunge în starea specificată. Stările care au fost parcurse vor fi indicate pe diagrama de stare prin culoarea verde, iar starea în care s-a ajuns va fi indicată prin culoarea galbenă. În fereastra cu diagramele de timp ale semnalelor se trasează forma acestora în ciclurile de ceas 0, 1 şi 2. Se va afişa apoi fereastra de dialog *Reached Target State*. Aceasta indică faptul că s-a ajuns în starea specificată, fiind posibilă selectarea unei alte stări destinație sau oprirea simulării. Se selectează butonul *No* pentru închiderea ferestrei şi oprirea simulării prin comanda *Verify Behavior*.
5. În cazul proiectelor mai complexe, cu un număr mare de stări se poate utiliza opțiunea *Automatic Test bench* 

. În urma rulării se deschide o fereastră *Coverage Statistics*, care prezintă numărul de cicluri executate şi procentul de stări acoperite în funcție de combinațiile dfe la intrare.
După parcurgerea etapelor menționate, rezultatul trebuie să fie ca şi cel din figura 10.

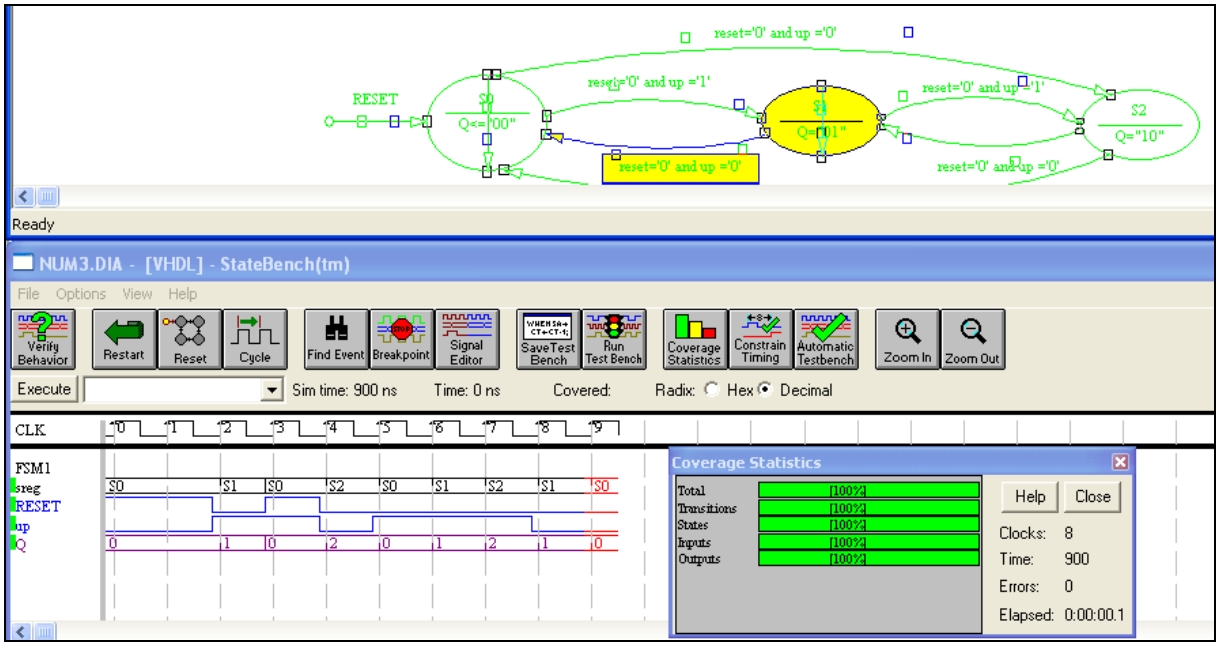


Figura 10.

Se salvează bancul de test prin selectarea butonului *Save Test Bench* 

. Se va deschide fereastra de dialog *Save Test Bench/Regression*. În această fereastră se păstrează numele implicite ale fişierelor şi se selectează butonul *Save*. Bancul de test generat în limbajul VHDL va fi afişat în fereastra *StateCAD HDL Browser*. Se vizualizează codul generat, după care se închide fereastra.

Pasul 8: Implementarea proiectului

Pentru a implementa proiectul, trebuie înlăturată din proiect sursa *num3.dia*, corespunzătoare diagramei de stare şi adăugat codul VHDL generat, *num3.vhd,* în acest caz.

Se va scrie fişierul de constrângeri corespunzător, intrarea de reset la un buton, ieşirile la LED-uri.

În continuare implementarea va decurge conform procedurilor prezentate în laboratoarele anterioare.

Atenţie!!!

Pentru a putea vizualiza tranziţiile între stări, semnalul de tact trebuie divizat, adăugaţi componenta de divizare definită în laboratorul anterior!

Indicaţii:

- adăugați la proiect fişierul VHDL corespunzător divizorului;
- creați simboluri schematice din fişierele VHDL ale divizorului şi ale numărătorului;
- adăugați la proiect un fişier schematic, conectați între ele simbolurile numărătorului şi ale divizorului, de asemenea adăugați pad-uri, salvați;
- vizualizați codul VHDL corespunzător fişierului schematic, copiati acest cod;
- înlăturați din proiect fişierul schematic şi adaugați un fişier VHDL nou, gol, în care „lipiți" codul VHDL copiat anterior.
- Gata, aveti acum la dispoziție entitatea top corespunzătoare conectării celor două module, vizualizați codul VHDL de conectare a acestora, astfel încât data viitoare să puteți scrie unul asemănător dacă doriți să adugați componente noi.

NU uitați să activați latch-urile asociate LED-urilor, vezi semnal de validare LEDG!

Activități suplimentare

- Se va urmării pas cu pas descrierea numărătorului folosind editorul FSM din mediul Xilinx;
- Se va identifica cărui tip FSM (Mealy sau Moore) corespunde numărătorul proiectat, se va argumenta;
- Se va implementa numărătorul proiectat şi se testează;
- Bazându-ne pe numărătorul proiectat anterior se va extinde numărul de stări şi se vor adăuga caracteristicile necesare astfel încât să îndeplinească funcțiile specifice unui automat de stări de tip Mealy.
- Se va proiecta un numărător decadic pe patru biți cu următoarele specificații: reset sincron (reset), intrare de comandă pentru direcția de numărare (up/down ->1/0), intrare de date pe patru biți (data) , intrare de comandă încărcare (load), ieşire pe patru biți (output).

Circuite Logice Programabile

LABORATOR 7-8

PROIECTAREA UNUI CONTROLER DE TRAFIC. CREAREA PROIECTELOR MIXTE

SCOPUL LUCRĂRII

În această lucrare se va proiecta un controler pentru un semafor care va coordona circulația într-o intersecție. Controlerul va fi proiectat ca şi automat de stări (FSM). Pornind de la definiția problemei se va determina diagrama de stării după care se va face implementarea proiectului în FPGA. În această lucrare se va exersa crearea proiectelor mixte: schematic, cod VHDL, diagrame de stare (FSM). Intervalele de timp dintre tranziții vor fi contorizate şi afişate pe afişajul 7-segmente. Va fi prezentată şi modalitatea de comandă independentă a caracterelor afişajului 7-segmente.

IINTRODUCERE TEORETICĂ

Definirea problemei

Specificațiile pentru proiectarea controlerului de trafic sunt:

- Controlerul va comanda un semafor amplasat la intersecția dintre o autostradă (A) şi un drum secundar (DS),vezi figura 1;
- Senzorii S sesizează prezența unei maşini care staționează pe drumul secundar. Atâta timp cât nici o maşină nu staționează pe drumul secundar autostrada are prioritate timp de 20s după care se verifică starea senzorului S. Dacă nu este prezentă nici o maşină pe drumul secundar autostrada îşi va menține prioritatea în continuare, pentru alte 20s.
- Semaforul va acorda prioritate (un anumit interval, 10s) drumului secundar după o anumită perioadă , din momentul în care senzorul S a detectat prezența unei maşini;
- Trecerea de la culoarea verde la cea roşie nu se va face direct, ci trecând prin culoarea galbenă. Semaforul va păstra culoarea galbenă pentru un interval de timp, 5s.

Notații folosite în figura 1:
S – senzor; AS – semafor autostradă; DSS – semafor drum secundar.
Algoritmul de funcționare a controlerului este:

1. Verde pentru autostradă / Roşu pentru drumul secundar, timp de 20s;
2. Se verifică dacă există maşini staționate pe drumul secundar.
3. Dacă DA se trece la faza următoare;
4. Dacă NU se reia primul pas;
5. Galben pentru autostradă / Roşu pentru drumul secundar, timp de 5s;
6. Roşu pentru autostradă / Roşu pentru drumul secundar, timp de 5s;
7. Roşu pentru autostradă / Verde pentru drumul secundar, timp de 10s;
8. Roşu pentru autostradă / Galben pentru drumul secundar, timp de 5s;
9. Roşu pentru autostradă / Roşu pentru drumul secundar, timp de 5s;
10. Mergi la pasul 1.

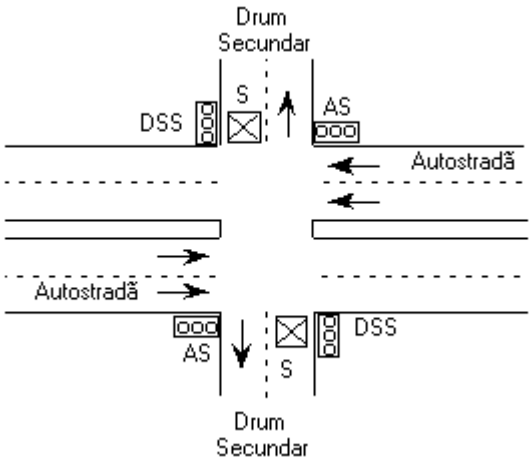


Figura 1 Configurația intersecției semaforizate

Definirea intrărilor, ieşirilor şi a stărilor intermediare şi a tranzițiilor între stări

În figura 2 este prezentată diagrama bloc a controlerului de trafic, cu intrările şi ieşirile specifice.

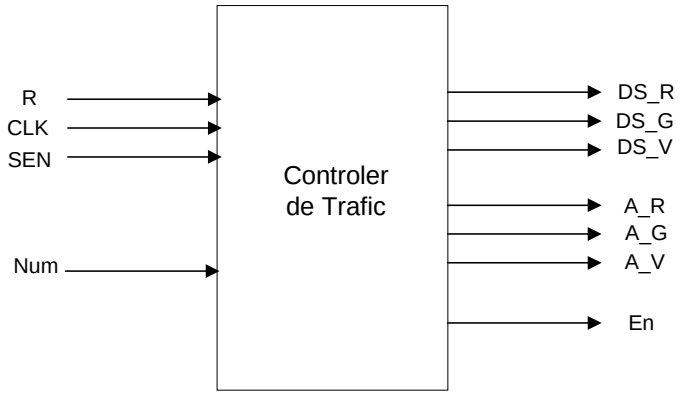


Figura 2 Intrările şi ieşirile Controlorului de Trafic proiectat ca şi FSM

Intrarea RST aduce FSM-ul în stare inițială (autostrada A are prioritate timp de 20s). Intrarea SEN detectează prezența unei maşini care aşteaptă pe drumul secundar DS. Cele şase ieşiri (DS_R, DS_G, DS_V, A_R, A_G, A_V) stabilesc culorile (roşu, galben, verde) pe care le vor afişa cele două semafoare (AS, DSS). Se va folosii un numărător extern, pentru a stabilii intervalele de timp (TL=20s, TM=10s şi TS=5s). Semnalul Start activează numărătorul, după fiecare schimbare de stare a FSM-ului. Numărătorul va implementa şi funcția de divizare a semnalului de tact pentru controlerul de trafic, astfel încât frecvența de lucru a acestuia să fie 1Hz. În aceste condiții pentru o perioadă de 1s a tactului, numărătorul va lua valorile: Num=20 pentru Ts=20s, Num=10pentru TM=10s şi Num=5 pentru TS=5s. În tabelul T.1 sunt prezentate cele 14 stării ale controlerului precum şi tranzițiile între ele. În coloanele tabelului se pot urmării valorile pe care le au ieşirile controlerului pentru anumite valori ale intrărilor corespunzătore unei stări. Cele 14 stări sunt după cum urmează:

- starea de *R*, stabileşte condițiile inițiale din care va pornii controlerul;
- starea *Start1,* lansează procesul de numărare în acest caz până la 20 şi totodată activează starea *Normală* de funcționare a controlerului;
- pe parcursul stării *Normale* de funcționare autostrada are prioritate timp de 20s, după care urmează tranziția în starea următoare;
- starea *Verifica* , decide după scurgerea celor 20s dacă tranziția va avea loc înapoi la starea *Normala* în cazul în care *Sen='0'*, adică nu este prezentă nici o maşină pe drumul secundar sau înspre starea *S1 ;*
- stările *Start1->Start6* declanşează tranzițiile dintr-o stare în alta a celor două semafoare (AS şi DSS) şi de asemenea declanşează (pentru *En='1'*) procesul de contorizare a timpului cât se staționează într-o stare *S1->S6*;
- stările *S1->S6* definesc efectiv schimbările de culori pentru cele două semafoare, starea *S3* care durează 10s este starea pe parcursul căreia drumul secundar are prioritate față de autostradă.

Tabelul T.1 Stările şi tranzițiile Controlerului de Trafic

Stări	**Intrări**				**Ieşiri**						**Cond.Tranz.**
	R	Sum	Sen	A_R	A_G	A_V	DS_ R	DS_ G	DS_ V	En	**Starea Următoare**
Reset	1	-	-	0	0	0	0	0	0	0	-
Start1	0	-	-	0	0	1	1	0	0	1	-
Normal	0	<19	-	0	0	1	1	0	0	0	N=19
Verific	0	-	0	0	0	1	1	0	0	1	Sen='0'
			1								Sen='1'
S1	0	<4	1	0	1	0	1	0	0	0	N=19
Start2	0	-	-	0	1	0	1	0	0	1	-
S2	0	<4		1	0	0	1	0	0	0	N=4
Start3	0	-	-	1	0	0	0	0	1	1	-
S3	0	<9	-	1	0	0	0	0	1	0	N=9
Start4	0	-	-	1	0	0	0	1	0	1	-
S4	0	<4	-	1	0	0	0	1	0	0	N=4
Start5	0	-	-	1	0	0	1	0	0	1	-
S5	0	<4	-	1	0	0	1	0	0	0	N=4
Start6	0	-	-	0	0	1	1	0	0	1	-

Desfăşurarea lucrării

Partea I

Pasul 1: Crearea proiectului

Odată stabilite stările controlerului, tranziţiile şi valorile de ieşire din fiecare stare putem să trecem la descrierea proiectului în schematic, pentru aceasta vor fi urmărite câteva etape.

În directorul personal se va crea un subdirector cu numele *lab7,* aici vor fi salvate toate proiectele ce vor fi create în cadrul acestei lucrări. Se va crea un proiect cu numele *semafor,* atenţie la directorul de lucru.

Proiectul va avea modulul *top* de tip schematic, iar celelalte vor fi simboluri schematice create din cod VHDL.

În prima fază se va proiectat controlerul, ca şi diagrama FSM, se va genera cod VHDL, se va crea un testbench şi se va testa prin simulare.

În faza a doua se va proiecta blocul de numărare, se va crea un testbench pentru acesta şi se va verifica funcţionarea.

În faza a treia se va proiecta modulului de divizare, se va crea un testbench pentrua acesta şi se va verifica prin simulare.

În ce ade a patra fază, ultima, se crează o sursă nouă în schematic şi toate simbolurile schematice corespunzătoare ficărui modul vor fi aduse în această sursă şi interconectate, după care se poate face implementarea proiectului.

Pasul 2: Proiectarea ierarhică, crearea simbolurilor în schematic

Proiectul poate fi descompus ierarhic în trei blocuri: partea efectivă de comandă (*Controler*), partea de contorizare (*Num*), care contorizează timpii de tranziţie (vezi tabelul T1) dintr-o stare în alta şi partea de divizare a semnalului de clock care furnizează tact cu perioada de 1 s. În figura 3 este prezentată o posibilă conectare a celor trei module şi porturile de intrare/ieşire necesare.

Proiectarea controlerului

Partea de control este descrisă ca şi FSM respectând stările şi tranziţiile din tabelul T1. Se deschide utilitarul *StateCad, Start->Programs->Xilinx ISE…->Accessories->StateCad* şi se va proiecta blocul conform instrucţiunilor din introducerea teoretică şi utilizând procedurile de lucru cu diagrame de stare, deja deprinse în laboratorul anterior.

După construirea diagramei de stări a controlerului aceasta va fi sintetizată, iar din codul VHDL rezultat se va crea un simbol în schematic. Simularea funcţională a controlerului se va face conform lucrări anterioare.

Atenţie, înainte de generarea codului VHDL verificaţi ca acesta să fie compatibil cu unealta de sinteză (XST) a mediului ISE Xilinx, , astfel verificaţi următoarea setare: click pe icoana *Optmize, Next*]n toate ferestrele până se ajunge la fereastra ca şi cea din figura 4, aici se selectează *Xilinx XST*, *next.*

În figura 5 este prezentată diagrama de stări specifică controlerului de trafic. În figura 6 sunt prezentate formele de undă rezultate în urma simulării.

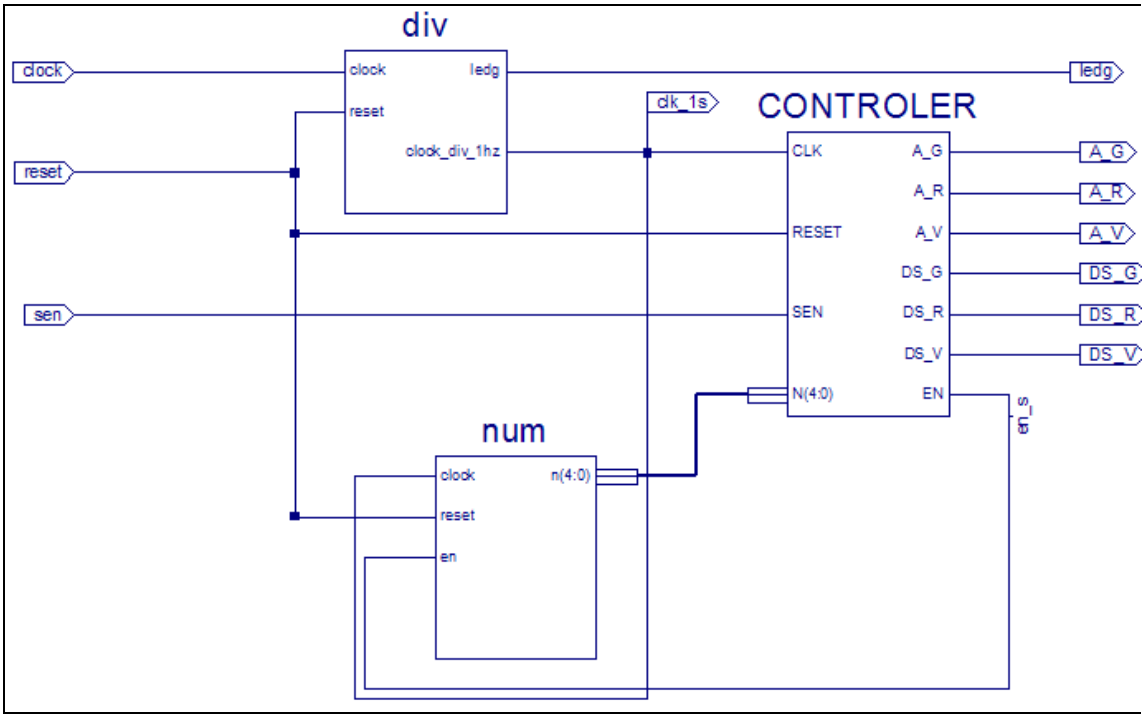


Figura 3 Descrierea în schematic a controlerului de trafic şi a blocurilor auxiliare

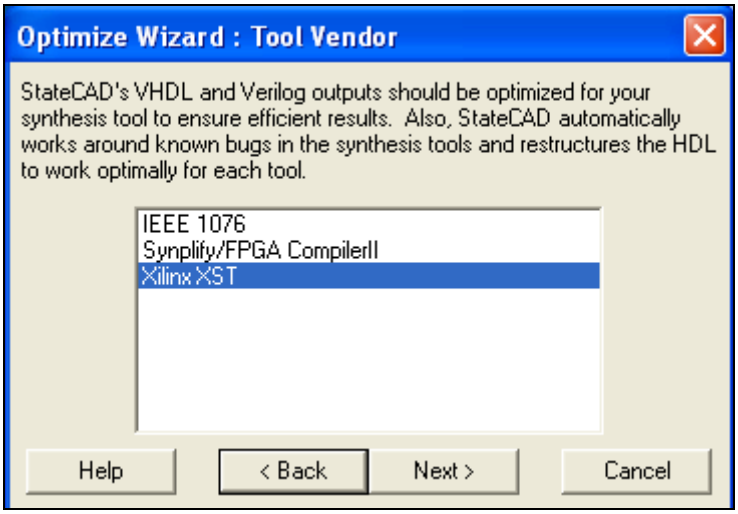


Figura 4. Opțiunea de sinteză a codului VHDL

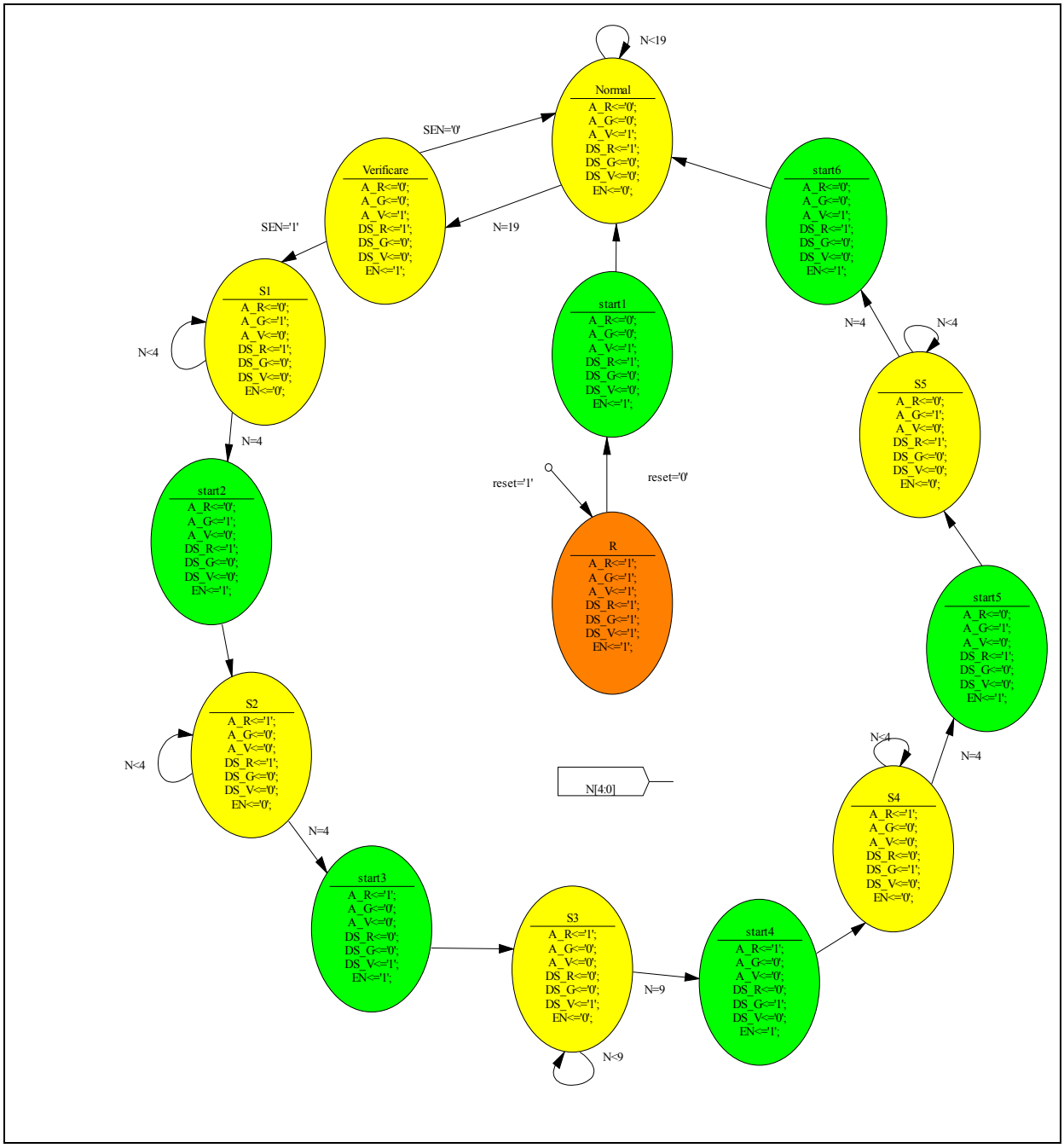


Figura 5. Diagrama de stării corespunzătoare părții de comandă a Controlerului de Trafic

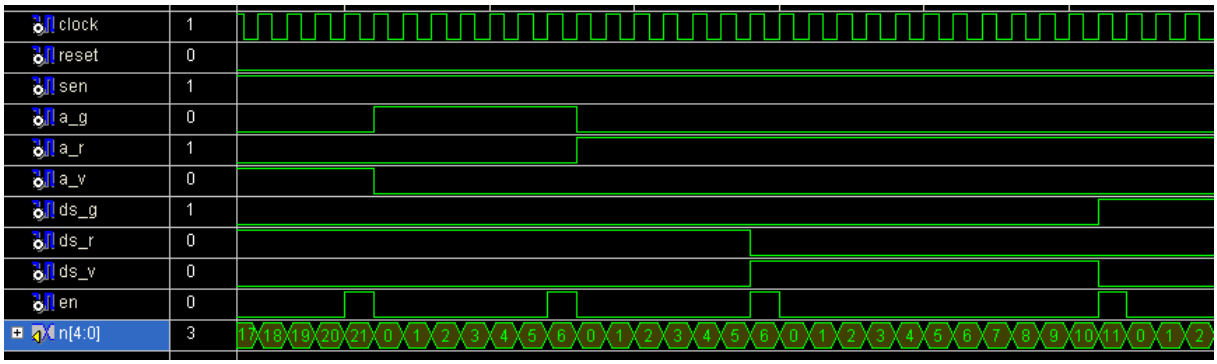


Figura 6. Formele de undă de la ieşirea controlerului de trafic

Proiectarea contorului

Blocul de numărare este descris în VHDL. O posibilă descriere poate arăta ca şi cea din figura 7. La fiecare impuls de activare primit de la controler acest bloc se va reseta şi va începe o nouă secvență de numărare.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity num is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           en : in  STD_LOGIC;
           n : out  STD_LOGIC_VECTOR (4 downto 0));
end num;
architecture Behavioral of num is
signal count: std_logic_vector (4 downto 0);


begin

p2:process (clock, reset)
begin
   if reset='1' then
      count <= (others => '0');
   elsif clock ='1' and clock'event then
         if en = '1' then
            count <= (others => '0');
         else
            count <= count + 1;
         end if;
   end if;
end process;

n <= count;
end Behavioral;
```

Figura 7. Cod VHDL corespunzător blocului de contorizare

Proiectarea divizorului

Blocul de divizare este descris în VHDL. O posibilă descriere poate arăta ca şi cea din figura 8. Semnalul de clock de 50 MHz este divizat cu un factor de 50.000.000, astfel obținându-se un semnal cu perioada de 1s.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity div is
   Port ( clock : in  STD_LOGIC;
          ledg: out std_logic;
          reset:in std_logic;
          clock_div_1hz: buffer  STD_LOGIC;
          clock_div_1khz: buffer  STD_LOGIC);
end div;

architecture Behavioral of div is
```

```
signal div2: integer;
            begin
p2_1s:process(reset, clock, div2)
                        begin
                        if reset ='1' then
                                    div2<= 0;
                                    clock_div_1Hz<='0';
                        elsif clock'event and clock='1' then
                                    div2 <= div2+1;
                        end if;
                        if div2 = 25000000 then
                                                clock_div_1hz <='1';
                        elsif div2 = 50000000 then
                                    clock_div_1hz <='0';
                        end if;
                        if div2 = 50000000 then
                                    div2<= 0;
                        end if;
end process;
ledg<='1';

end Behavioral;
```

Figura 8. Cod VHDL corespunzător blocului de divizare

Pasul 4: Implementarea şi testarea proiectului

- Se vă urmării pas cu pas desfăşurarea lucrării, conform indicațiilor din paragrafele anterioare şi se va verifica ca toate stările şi condițiile descrise în tabelul T1 să fie atinse de automatul de stări proiectat.
- Simularea modulelor va fi făcută separat pentru fiecare modul în parte, la frecvența de clock implicită a simulatorului. Numai în fază de implementare se va stabilii divizarea reală cu 50x10*6, pentru a obține clockul cu perioada de 1s . Obligatoriu se va efectua simularea funcțională a fiecărui modul pentru a verifica corectitudinea codurilor VHDL.
- La implementare pinii FPGA –ului vor fi aleşi astfel încât, pentru fiecare culoare a celor două semafoare să avem asociat un LED, pentru reset să avem asociat un buton, iar pentru senzor un comutator, o variantă posibilă ar putea arăta ca şi cea din figura 9.

```
NET "clock"  LOC = "P182" ;
NET "reset"  LOC = "P3"    ;#BTN1
NET "sen"  LOC = "P23"  ;#SW1
NET "A_R"  LOC = "P111" ;#LED1
NET "A_G"  LOC = "P109" ;#LED2
NET "A_V"  LOC = "P102" ;#LED3
NET "DS_R"  LOC = "P100";#LED4
NET "DS_G"  LOC = "P98" ;#LED5
NET "DS_V"  LOC = "P96" ;#LED6
NET "clk_1s"  LOC = "P89"  ;#LED7 vizualire tact 1 s,
NET "ledg"  LOC = "P45"  ;#ACTIVARE LED-uri
```

Figura 9. Constrângeri aplicate pinilor circuitului FPGA

Partea a II -a

În această parte, funcțiile proiectului vor fi extinse. Astfel, se doreşte vizualizare pe afişajul 7-segmente a intervalelor de timp contorizate, de 5, 10 respectiv 20s. Pentru aceasta proiectul va suferii câteva modificări. În această parte este necesar ca intervalul de timp de 20 de s să fie afişat pe două caractere separate ale afişajului, aceasta presupune controlul separat al caracterelor., detalii vor fi oferite în următorul paragraf.

Prezentare mod de lucru afişaj 7-segmente

Placa DIO4 conține un afişaj 7-segmente pe patru caractere, cu anod comun. Cei şapte anozi ai celor şapte segmente care alcătuiesc un caracter sunt conectați la un punct comun notat AN. Conectând la 0 sau 1 logic acest punct comun, fiecare caracter va putea fi activat în mod independent. Catozii segmentelor similare de la toți cei patru digiți ai afişajului sunt conectați împreună , având astfel şapte circuite independente. Astfel fiecare catod al celor patru digiți poate fi activat sau dezactivat independent. Prin această schemă de conexiuni s-a obținut un afişaj multiplexat, în care comandând succesiv semnalele comune anozilor şi trimițând în mod repetat secvența corespunzătoare catozilor fiecărui digit, se obține afişarea pe patru caractere. Schema de conectare a anozilor şi catozilor, precum şi secvența de activare a a acestora este prezentată în figura 10. Tiparul care se aplică catozilor pentru afişarea 7 segmente este cel cunoscut.

Pentru ca fiecare din cele patru caractere să fie iluminat în mod continuu şi intensitatea iluminării să fie corespunzătoare, secvența de date trebuie reîmprospătată la fiecare 1 până la 16 ms, vezi figura 10.

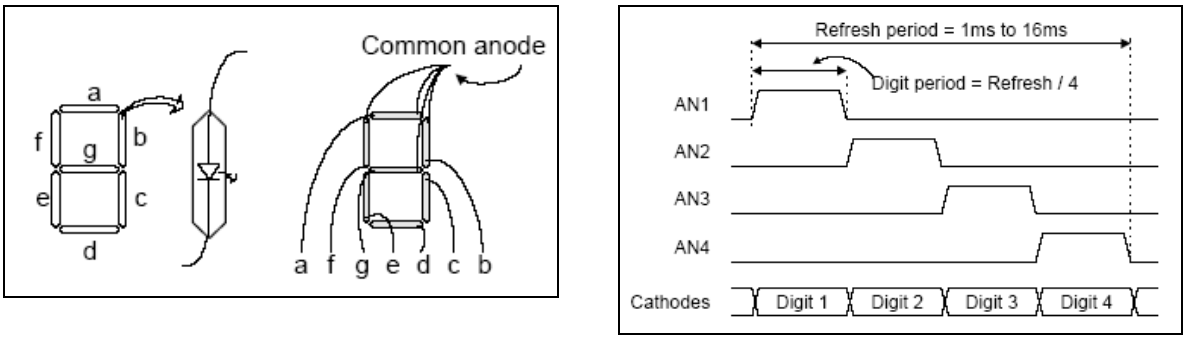


Figura 10 Schema de conectare anozi, catozi şi secvența de activare a acestora

Pasul 5: Modificarea proiectului

Proiectarea blocului de decodificare/multiplexare

Diagrama în schematic din figura 3 va fi modificată prin adăugarea unui nou bloc, vezi figura 11, care are rol de decodificare 7-segmente şi de multiplexare a semnalului pentru afişajul de pe placa de test DIO4.

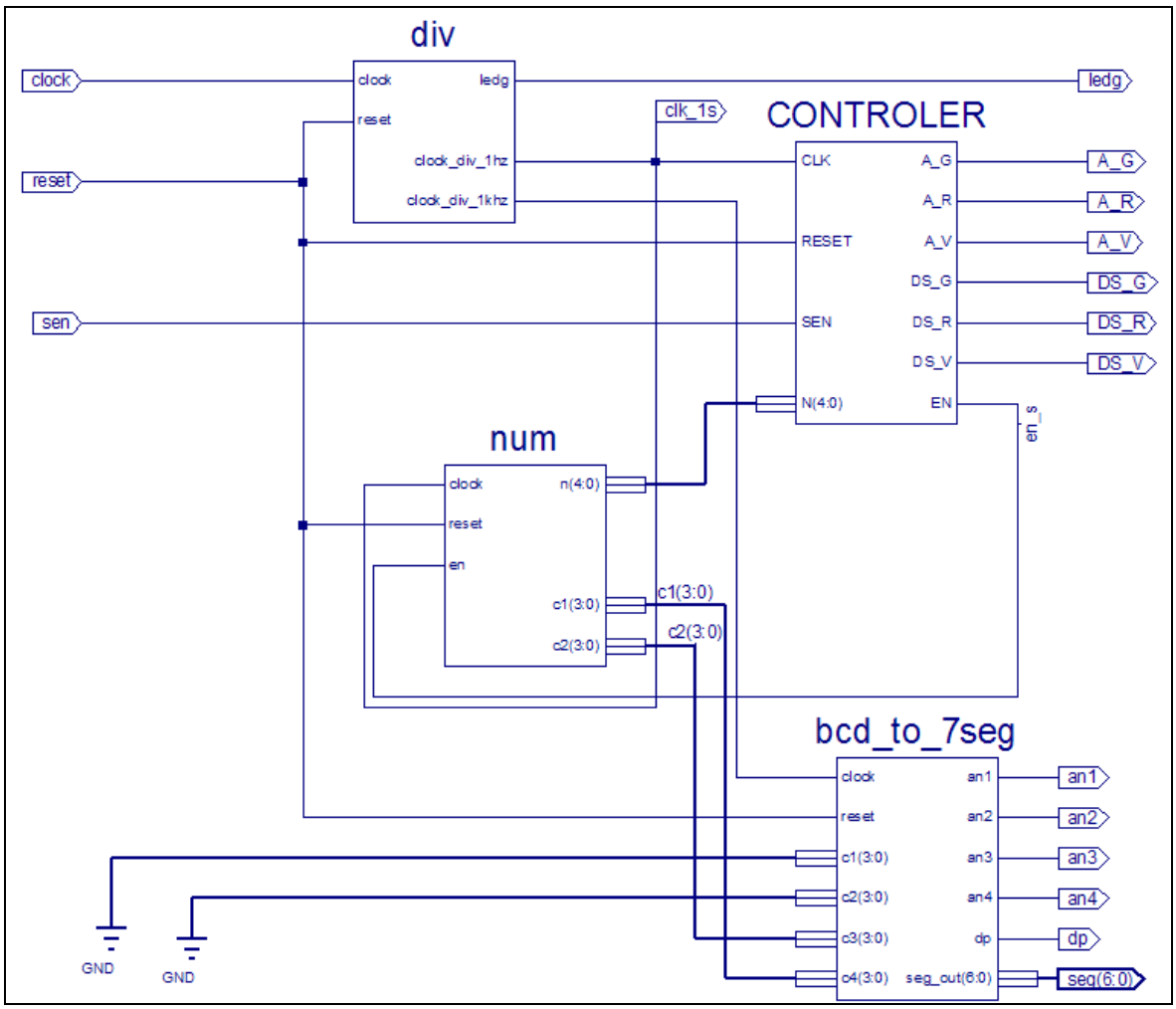


Figura 11. Descrierea în schematic a controlerului de trafic şi a blocurilor auxiliare

Codul VHDL corespunzător blocului BCD (binar codificat zecimal) 7-segmente, este prezentat în figura 12.

```
-- modul VHDL de conversie BCD--intreg-7_segmnente si afisare pe patru caractere
-- placa Digilent-DIO4
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity bcd_to_7seg is
                port(
                        clock : in std_logic;
                        reset : in std_logic;
                        c1: in std_logic_vector (3 downto 0);
                        c2: in std_logic_vector (3 downto 0);
                        c3: in std_logic_vector (3 downto 0);
                        c4: in std_logic_vector (3 downto 0);
                        an1 : out std_logic;
                        an2 : out std_logic;
                        an3 : out std_logic;
                        an4 : out std_logic;
```

```
                        dp: out std_logic;
                        seg_out : out std_logic_vector (6 downto 0)
        );
end bcd_to_7seg;
architecture Behavioral of bcd_to_7seg is
-----------------------------------------------------------
        -- functie de conversie din binar in intreg

        function vec2int (x:std_logic_vector) return integer is
                variable result : integer;
        begin
                result :=0;
                for i in x'range  loop
                        result :=result*2;
                        case x(i) is
                                when '0' =>null;
                                when '1' => result := result +1;
                                when others => null;
                        end case;
                end loop;
                return result;
        end vec2int;
 ------------------------------------------------------------
         -- functie de conversie din intreg in 7-segmente

         function int7seg (x:integer) return std_logic_vector      is
         variable dis7seg: std_logic_vector (6 downto 0);
         begin
                        if x = 0 then dis7seg := "0111111";
                        elsif x = 1 then dis7seg := "0000110";
                        elsif x = 2 then dis7seg := "1011011";
                        elsif x = 3 then dis7seg := "1001111";
                        elsif x = 4 then dis7seg := "1100110";
                        elsif x = 5 then dis7seg := "1101101";
                        elsif x = 6 then dis7seg := "1111101";
                        elsif x = 7 then dis7seg := "0000111";
                        elsif x = 8 then dis7seg := "1111111";
                        elsif x = 9 then dis7seg := "1101111";
                        else dis7seg := "0001000";
                        end if;
                  return dis7seg;
           end;
 ------------------------------------------------------------
signal seg_out1: std_logic_vector (6 downto 0):="0000000";
signal seg_out2: std_logic_vector (6 downto 0):="0000000";
signal seg_out3: std_logic_vector (6 downto 0):="0000000";
signal seg_out4: std_logic_vector (6 downto 0):="0000000";
signal count2 : std_logic_vector (1 downto 0):="00";
begin
        display: process (clock, reset)
                begin
                if reset ='1' then

                        an1<='0';an2<='0';an3<='0';an4<='0'; dp<='0';
```

```
                    seg_out<=not int7seg(8);
                    count2<="00";
                elsif  clock ='1' and clock'event then
                                count2 <= count2+1;


        seg_out4<=int7seg(vec2int(c4));          -- se converteste 7 segmente fiecare digit in parte
        seg_out3<=int7seg(vec2int(c3));
        seg_out2<=int7seg(vec2int(c2));
        seg_out1<=int7seg(vec2int(c1));

                if count2 = "00" then             --se multiplexează semnalul pentru cele 4 caractere
                        seg_out <= not seg_out1;
                        an1<='0';an2<='1';an3<='1';an4<='1';dp<='1';
                elsif count2 = "01" then
                        seg_out <= not seg_out2;
                        an1<='1';an2<='0';an3<='1';an4<='1';dp<='0';
                elsif count2 = "10" then
                        seg_out <= not seg_out3;
                        an1<='1';an2<='1'; an3<='0'; an4<='1';dp<='1';
                elsif count2 = "11" then
                        seg_out <= not seg_out4;
                        an1<='1'; an2<='1';an3<='1';an4<='0';dp<='1';
                end if;
        end if;
        end process;
end Behavioral;
```

Figura 12. Modul VHDL de conversie BCD--intreg-7_segmnente si afisare pe patru caractere

În acest modul se face o descriere a două funcții. Prima este de conversie din binar în valoare întreagă, iar cea de a doua descrie decodificatorul 7 segmente sub forma de funcție. Aceste funcții vor fi aplelate simultan în arhitectură, vezi zona cu roşu. Odată se face o conversie în valoare întreagă a semnalului de la cele patru intrări binare *C1-4, funcția vec2int,* după care se face conversia în cod 7 segmente, *funcția int7seg*. Semnalele rezultate *seg_out1-4,* vor fi apoi multiplexate, conform figurii şi trimise la afişajul 7 segmente.

Potrivit celor menționate într-un paragraf anterior acest bloc va avea nevoie să facă multiplexare într-un interval de minim 1ms şi maxim 16 ms. Din figura 11 se poate observa că intrarea de clock a blocului de decodificare/multiplexare provine de la o a doua intrare a blocului de divizare care asigură un semnal cu perioada de 1ms, respectiv frecvența de 1kHz.

Codul VHDL din figura 12 va fi testa prin simulare funcțională, se descrie un testbench, după care se va genera un simbol în schematic, simbol ce va fi adăugat la diagrama din figura 3.

Se poate observa că în această lucrare, doar două dintre intrările binare *C3, C4* vor fi conectate la blocul de numărare, celelalte două vor fi conectate la masă, astfel că pe caracterele 1 şi 2 va fi afişată valoarea 0.

Atenție!!! Când se adaugă simbolurile *GND* (masă) se dă dublu click pe acestea şi în câmpul *value* se definesc ca fiind pe 4 biți, se adaugă *(3:0).*

Din diagrama 3 se poate observa că au apărut porturi noi şi la celelalte blocuri.

Modificare divizorului

În figura 13 este prezentat codul VHDL modificat corespunzător blocului de divizare. Modificările sunt evidențiate cu, culoare roşie. Se poate observa ca fost adăugat încă un bloc proces

care realizează divizarea semnalului de clock cu valoarea întreagî 50.000, astfel încât se va obține un semnal de ieşire cu frecvența de 1kHz. Acest semnal divizat va constitui semnalul de tact al blocului de divizare/multiplexare.

Odată codul modificat, acesta va fi resintetizat şi se va crea din nou simbolul în schematic .

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity div is
   Port (  clock : in  STD_LOGIC;
                     ledg: out std_logic;
                     reset:in std_logic;
                     clock_div_1hz: buffer  STD_LOGIC;
                     clock_div_1khz: buffer  STD_LOGIC);
end div;
architecture Behavioral of div is
          signal div1: integer;
          signal div2: integer;
          begin
p1_1ms:process(reset, clock, div1)
                     begin
                     if reset ='1' then
                               div1<= 0;
                               clock_div_1kHz<='0';
                     elsif clock'event and clock='1' then
                               div1 <= div1+1;
                     end if;
                     if div1 = 25000 then
                                        clock_div_1khz <='1';
                     elsif div1 = 50000 then
                                        clock_div_1khz <='0';
                     end if;
                     if div1 = 50000 then
                               div1<= 0;
                     end if;
end process;
p2_1s:process(reset, clock, div2)
                     begin
                     if reset ='1' then
                               div2<= 0;
                               clock_div_1Hz<='0';
                     elsif clock'event and clock='1' then
                               div2 <= div2+1;
                     end if;
                     if div2 = 25000000 then
                               clock_div_1hz <='1';
                     elsif div2 = 50000000 then
                               clock_div_1hz <='0';
                     end if;
                     if div2 = 50000000 then
                               div2<= 0;
                     end if;

end process;
ledg<='1';
end Behavioral;
```

Figura 13. Cod VHDL modificat, corespunzător blocului de divizare, *Div*

Modificarea contorului

Din figura 11 se poate observa că blocul de contorizare *num* are două ieşiri suplimentare care sunt conectate ca şi intrări la blocul de multiplexare/afişare. Codul VHDL modificat al blocului de contorizare este prezentat în figura 14.
Cu culoare roşie s-a evidenţiat codul introdus suplimentar. Pentru a nu fi necesară o conversie din binar în BCD a valorii date de numărătorul ce contorizează intervalele 5,10, 20s, s-au făcut un contor BCD în paralel, vezi procesele P1 şi P2. Au fost necesare două procese, pentru valori mai mari decât 9, astfel primul proces este pentru cifra unităților, iar cel de al doilea este pentru cifra zecilor. La fel s-ar putea extinde pentru cifra sutelor şi a miilor, etc.

Odată făcute modificările în codul VHDL, acesta poate fi simulat, se creează un testbench, se resintetizează şi se creează din nou simbolul în schematic.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity num is
   Port ( clock : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          en : in  STD_LOGIC;
                                  c1 : out  STD_LOGIC_VECTOR (3 downto 0);
                                  c2 : out  STD_LOGIC_VECTOR (3 downto 0);
          n : out  STD_LOGIC_VECTOR (4 downto 0));
end num;
architecture Behavioral of num is
signal count: std_logic_vector (4 downto 0);
signal count1_bcd: std_logic_vector (3 downto 0);
signal count2_bcd: std_logic_vector (3 downto 0);

begin

p1_c1_bcd:process (clock, reset)
begin
  if reset='1' then
            count1_bcd <= (others => '0');
  elsif clock ='1' and clock'event then
       if en = '1' then
                     count1_bcd <= (others => '0');
       else
                     count1_bcd <= count1_bcd + 1;
       end if;
       if count1_bcd = "1001" then
                     count1_bcd <= "0000";
        else null;
  end if;
end if;
end process;

p2_c2_bcd:process (clock, reset)
begin
  if reset='1' then
                     count2_bcd<= (others => '0');
  elsif clock ='1' and clock'event then
                                  if en = '1' then
                                           count2_bcd<= (others => '0');
                                  elsif count1_bcd = "1001" then
```

```
                                    count2_bcd <= count2_bcd+1;
                                else null;
                                end if;
                                if count2_bcd = "1001" then
                                    count2_bcd <= "0000";
                                else null;
                                end if;
            end if;
end process;

p3_num_bin:process (clock, reset)
begin
  if reset='1' then
    count <= (others => '0');
  elsif clock ='1' and clock'event then
                                if en = '1' then
        count <= (others => '0');
      else
        count <= count + 1;
      end if;
          end if;
end process;

n <= count;
c1<= count1_bcd;
c2 <= count2_bcd;

end Behavioral;
```

Figura 14. Cod VHDL modificat, corespunzător blocului de contorizare, *Num*

Pasul 6: Re-implementarea şi testarea proiectului

- Se vă urmării pas cu pas desfăşurarea etapelor indicate în pasul 5.
- Simularea modulelor va fi făcută separat pentru fiecare modul în parte
- La implementare se va modifica fişierul UCF, adăugânduse constrângerile corespunzătoare afişajului 7-segmente, figura 15.

```
NET "an1"  LOC = "P41" ; #activare primul caracter
NET "an2"  LOC = "P40" ; #activare al doilea caracter
NET "an3"  LOC = "P36" ; #activare al treilea caracter
NET "an4"  LOC = "P35" ; #activare patrulea caracter
NET "seg(0)"  LOC = "P22" ;#segmentul "a" al afisajului 7-seg
NET "seg(1)"  LOC = "P20" ;#segmentul "b" al afisajului 7-seg
NET "seg(2)"  LOC = "P17" ;#segmentul "c" al afisajului 7-seg
NET "seg(3)"  LOC = "P15" ;#segmentul "d" al afisajului 7-seg
NET "seg(4)"  LOC = "P10" ;#segmentul "e" al afisajului 7-seg
NET "seg(5)"  LOC = "P8" ; #segmentul "f" al afisajului 7-seg
NET "seg(6)"  LOC = "P6" ; #segmentul "g" al afisajului 7-seg
NET "dp"  LOC = "P4" ;                  #caracterul punct
```

Figura 15. Constrângeri aplicate pinilor circuitului FPGA

Activități suplimentare

- Se vor modifica intervalele de timp dintre tranziții;
- Se vor afişa valori aleatoare pe cele două caractere nefolosite ale afişajului, pentru a se verifica controlul independent al acestora;
- Se va modifica modulul de contorizare astfel încât la tranziția dintre stări să fie afişată valoarea finală (ex. 5, 10, 20) a intervalului de timp, iar aceasta să se decrementeze. Astfel încât să se ştie anticipat cât durează intervalul de aşteptare dintre două tranziții.

Circuite Logice Programabile

LABORATOR 9-10

PROIECTAREA CEAS DIGITAL

SCOPUL LUCRĂRII

În această lucrare se va proiecta ceas digital. Ceasul proiectat se va baza pe module proiectate anterior (divizor, decodificato/afişor 7 segmente) şi pe module nou create.

Desfăşurarea lucrării

```
ibrary IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity count is
   Port ( clock : in  STD_LOGIC;
          reset : in  STD_LOGIC;
                                  sec_m : out std_logic_vector (3 downto 0); --secunde digitul zecilor
                                  sec_l : out std_logic_vector (3 downto 0) --secunde digitul unitatilor
          );
end count;
architecture Behavioral of count is
signal sec_m_s:std_logic_vector (3 downto 0);
signal sec_l_s:std_logic_vector (3 downto 0);

begin
p1_secunde_bcd:process (clock, reset)
begin
  if reset='1' then
                        sec_m_s <= (others => '0');
                        sec_l_s <= (others => '0');
  elsif clock ='1' and clock'event then
                                   sec_l_s <= sec_l_s + 1;
                                  if sec_l_s = "1001" then
                                            sec_l_s <= "0000";
                                            sec_m_s <= sec_m_s + 1;
                                  elsif sec_m_s = "0110" then
                                            sec_m_s <= "0000";
                                  else null;
                                  end if;
            end if;
end process;
sec_l <= sec_l_s;
sec_m <= sec_m_s;

end Behavioral;
```

Figura 1. Cod VHDL pentru contorizarea secundelor

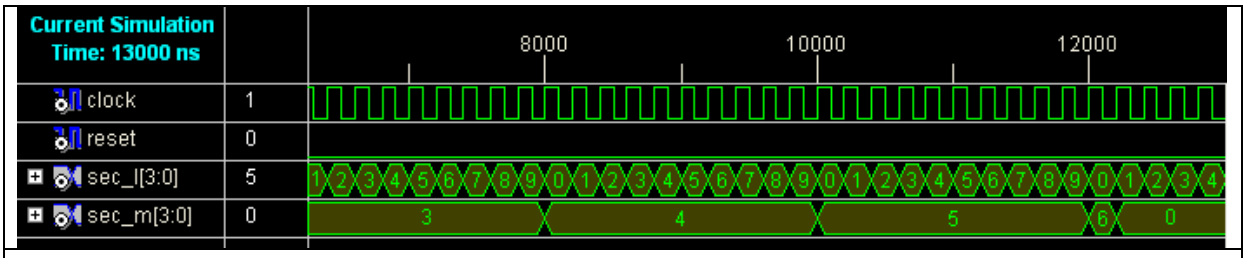


Figura 2. Simularea codului VHDL pentru contorizarea secundelor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity count2 is
   Port ( clock : in  STD_LOGIC;
        reset : in  STD_LOGIC;
                                sec_m : out std_logic_vector (3 downto 0); --secunde digital zecilor
                                sec_l : out std_logic_vector (3 downto 0); --secunde digital unitatilor
                                min_m : out std_logic_vector (3 downto 0); --minute digital zecilor
                                min_l : out std_logic_vector (3 downto 0); --minute digital unitatilor
                                ora_m : out std_logic_vector (3 downto 0); --ora digital zecilor
                                ora_l : out std_logic_vector (3 downto 0) –ora digital unitatilor
        );
end count2;
architecture Behavioral of count2 is
signal sec_m_s:std_logic_vector (3 downto 0);
signal sec_l_s:std_logic_vector (3 downto 0);
signal min_m_s:std_logic_vector (3 downto 0);
signal min_l_s:std_logic_vector (3 downto 0);
signal ora_m_s:std_logic_vector (3 downto 0);
signal ora_l_s:std_logic_vector (3 downto 0);

begin

p1_secunde_bcd:process (clock, reset)
begin
  if reset='1' then
                        sec_m_s <= (others => '0');
                        sec_l_s <= (others => '0');
  elsif clock ='1' and clock'event then
                                  sec_l_s <= sec_l_s + 1;
                                 if sec_l_s = "1001" then
                                            sec_l_s <= "0000";
                                            sec_m_s <= sec_m_s + 1;
                                 elsif sec_m_s = "0110" then
                                            sec_m_s <= "0000";
                                 else null;
                                 end if;
            end if;
end process;

p2_minute_bcd:process (sec_m_s, reset)
begin
  if reset='1' then
```

```
                    min_m_s <= (others => '0');
                    min_l_s <= (others => '0');
  elsif clock ='1' and clock'event then
                                if sec_m_s = "0110" then
                                 min_l_s <= min_l_s + 1;
                                elsif min_l_s = "1010" then
                                        min_l_s <= "0000";
                                        min_m_s <= min_m_s + 1;
                                elsif min_m_s = "0110" then
                                        min_m_s <= "0000";
                                else null;
                                end if;
          end if;
end process;

p3_ora_bcd:process (clock, reset)
begin
  if reset='1' then
                    ora_m_s <= (others => '0');
                    ora_l_s <= (others => '0');
  elsif clock ='1' and clock'event then
                                if min_m_s = "0110" then
                                 ora_l_s <= ora_l_s + 1;
                                elsif ora_l_s = "1010" then
                                        ora_l_s <= "0000";
                                        ora_m_s <= ora_m_s + 1;
                                elsif ora_m_s = "0010"  and ora_l_s="0011" and min_m_s = "0101" and min_l_s =
"1001" and sec_m_s = "0101" and sec_l_s = "1001"then
                                        ora_m_s <= "0000";
                                        ora_l_s <= "0000";
                                else null;
                                end if;
          end if;
end process;

sec_l <= sec_l_s;
sec_m <= sec_m_s;
min_l <= min_l_s;
min_m <= min_m_s;
ora_l <= ora_l_s;
ora_m <= ora_m_s;

end Behavioral;
```

Figura 3. Figura 1. Cod VHDL corespunzător modulului de contorizare a secundelor, minutelor şi orelor

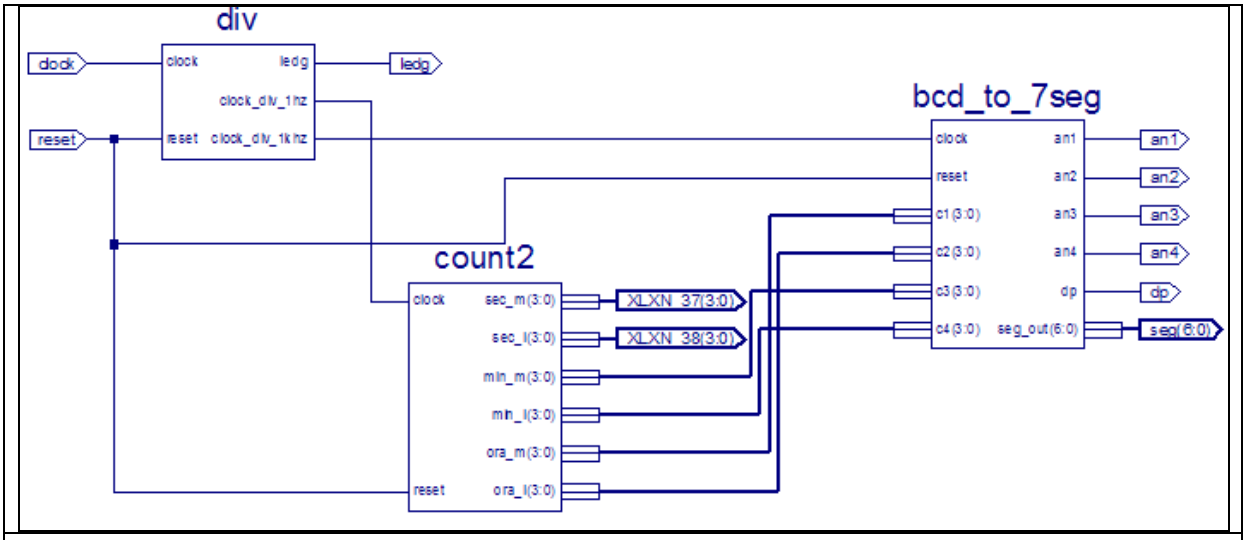


Figura 4. Ceaas digital cu afişare ora şi minute

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

entity Mux4X4 is
   Port ( S : in  STD_LOGIC;
          A1 : in  STD_LOGIC_VECTOR (3 downto 0);
          B1 : in  STD_LOGIC_VECTOR (3 downto 0);
          O1 : out  STD_LOGIC_VECTOR (3 downto 0);
                                  A2 : in  STD_LOGIC_VECTOR (3 downto 0);
          B2 : in  STD_LOGIC_VECTOR (3 downto 0);
          O2 : out  STD_LOGIC_VECTOR (3 downto 0);
                                  A3 : in  STD_LOGIC_VECTOR (3 downto 0);
          B3 : in  STD_LOGIC_VECTOR (3 downto 0);
          O3 : out  STD_LOGIC_VECTOR (3 downto 0);
                                  A4 : in  STD_LOGIC_VECTOR (3 downto 0);
          B4 : in  STD_LOGIC_VECTOR (3 downto 0);
          O4 : out  STD_LOGIC_VECTOR (3 downto 0));
end Mux4X4;

architecture Behavioral of Mux4X4 is
begin
process(S,A1,B1,A2,B2,A3,B3,A4,B4)
begin
            if S = '0' then
                        O1 <= A1;
                        O2 <= A2;
                        O3 <= A3;
                        O4 <= A4;
            else
                        O1 <= B1;
                        O2 <= B2;
                        O3 <= B3;
                        O4 <= B4;
            end if;
end process;
end Behavioral;
```

Figura 5. Cod VHDL corespunzător unui multiplexor cu 4 intrări de date pe 4 biți

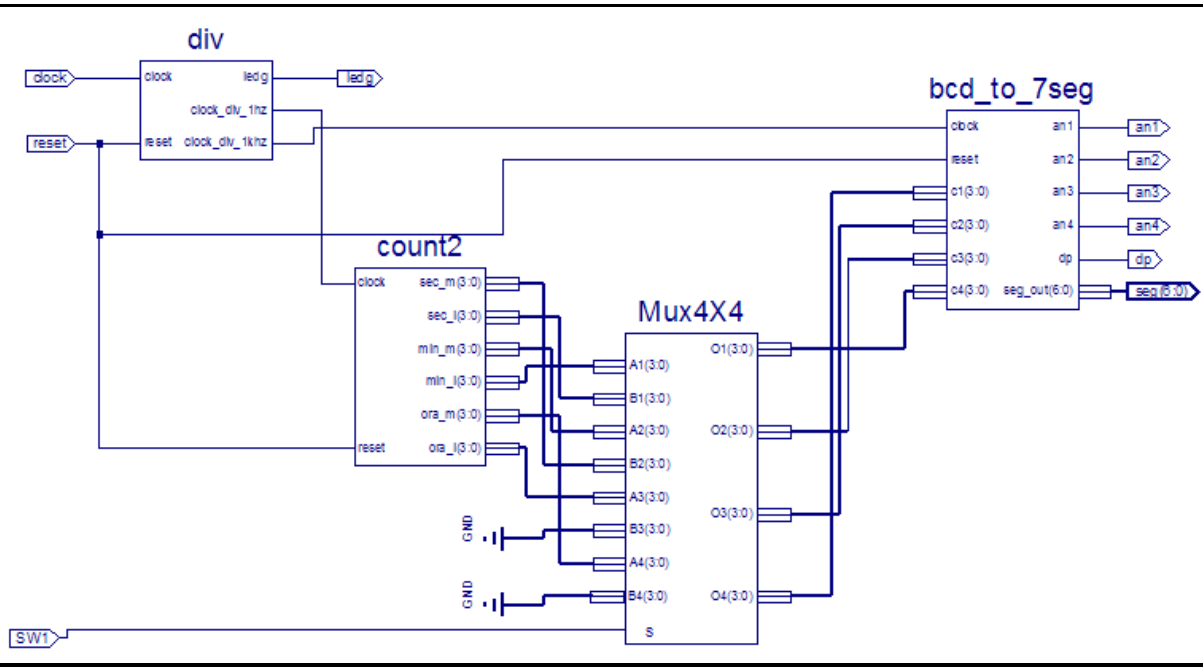


Figura 6. Ceas digital ce permite şi afişarea secundelor

```
NET "clock"  LOC = "P182" ;
NET "reset"  LOC = "P42"  ;#BTN5
NET "SW1"  LOC = "P23"   ;# selectare secunde
NET "an1"  LOC = "P41" ; #activare primul caracter
NET "an2"  LOC = "P40" ; #activare al doilea caracter
NET "an3"  LOC = "P36" ; #activare al treilea caracter
NET "an4"  LOC = "P35" ; #activare patrulea caracter
NET "seg(0)"  LOC = "P22" ;#segmentul "a" al afisajului 7-seg
NET "seg(1)"  LOC = "P20" ;#segmentul "b" al afisajului 7-seg
NET "seg(2)"  LOC = "P17" ;#segmentul "c" al afisajului 7-seg
NET "seg(3)"  LOC = "P15" ;#segmentul "d" al afisajului 7-seg
NET "seg(4)"  LOC = "P10" ;#segmentul "e" al afisajului 7-seg
NET "seg(5)"  LOC = "P8" ; #segmentul "f" al afisajului 7-seg
NET "seg(6)"  LOC = "P6" ; #segmentul "g" al afisajului 7-seg
NET "dp"  LOC = "P4" ;      #caracterul punct
```

Figura 7. Fisier de constrângeri corespunzător diagramei din figura 6.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity count2 is
    Port ( clock : in  STD_LOGIC;
```

```
        reset : in  STD_LOGIC;
                                SW2 :   in  STD_LOGIC;
                                digit1 : in std_logic_vector (3 downto 0);
                                digit2 : in std_logic_vector (3 downto 0);
                                digit3 : in std_logic_vector (3 downto 0);
                                digit4 : in std_logic_vector (3 downto 0);
                                sec_m : out std_logic_vector (3 downto 0); --secunde digital zecilor
                                sec_l : out std_logic_vector (3 downto 0); --secunde digital unitatilor
                                min_m : out std_logic_vector (3 downto 0); --minute digital zecilor
                                min_l : out std_logic_vector (3 downto 0); --minute digital unitatilor
                                ora_m : out std_logic_vector (3 downto 0); --ora digital zecilor
                                ora_l : out std_logic_vector (3 downto 0) –ora digital unitatilor
        );
end count2;
architecture Behavioral of count2 is
signal sec_m_s:std_logic_vector (3 downto 0);
signal sec_l_s:std_logic_vector (3 downto 0);
signal min_m_s:std_logic_vector (3 downto 0);
signal min_l_s:std_logic_vector (3 downto 0);
signal ora_m_s:std_logic_vector (3 downto 0);
signal ora_l_s:std_logic_vector (3 downto 0);

begin
p1_secunde_bcd:process (clock, reset)
begin
  if reset='1' then
                        sec_m_s <= (others => '0');
                        sec_l_s <= (others => '0');
  elsif clock ='1' and clock'event then
                                 sec_l_s <= sec_l_s + 1;
                                if sec_l_s = "1001" then
                                        sec_l_s <= "0000";
                                        sec_m_s <= sec_m_s + 1;
                                elsif sec_m_s = "0110" then
                                        sec_m_s <= "0000";
                                else null;
                                end if;
           end if;
end process;

p2_minute_bcd:process (sec_m_s, reset)
begin
  if reset='1' then
                        min_m_s <= (others => '0');
                        min_l_s <= (others => '0');
   elsif SW2 ='1' then
                        min_l_s <= digit1;
                        min_m_s <= digit2;
  elsif clock ='1' and clock'event then
                                if sec_m_s = "0110" then
                                 min_l_s <= min_l_s + 1;
                                elsif min_l_s = "1010" then
                                        min_l_s <= "0000";
                                        min_m_s <= min_m_s + 1;
                                elsif min_m_s = "0110" then
                                        min_m_s <= "0000";
                                else null;
```

```
                                end if;
        end if;
end process;

p3_ora_bcd:process (clock, reset)
begin
  if reset='1' then
                ora_m_s <= (others => '0');
                ora_l_s <= (others => '0');
  elsif SW2 ='1' then
        ora_l_s <= digit3;
        ora_m_s <= digit4;
  elsif clock ='1' and clock'event then
                                if min_m_s = "0110" then
                                 ora_l_s <= ora_l_s + 1;
                                elsif ora_l_s = "1010" then
                                        ora_l_s <= "0000";
                                        ora_m_s <= ora_m_s + 1;
                                elsif ora_m_s = "0010"  and ora_l_s="0011" and min_m_s = "0101" and min_l_s =
"1001" and sec_m_s = "0101" and sec_l_s = "1001"then
                                        ora_m_s <= "0000";
                                        ora_l_s <= "0000";
                                else null;
                                end if;
        end if;
end process;

sec_l <= sec_l_s;
sec_m <= sec_m_s;
min_l <= min_l_s;
min_m <= min_m_s;
ora_l <= ora_l_s;
ora_m <= ora_m_s;

end Behavioral;
```

Figura 8. Cod VHDL corespunzător modulului de contoriazare, ce permite încărcarea unei valori prestabilite pentru minute şi oră

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Input is
   Port ( reset : in std_logic;
                                clock : in std_logic;
                                sw5 : in std_logic;
                                sw6 : in std_logic;
                                sw7 : in std_logic;
                                sw8 : in std_logic;
        inc : in std_logic;
        digit1 : out std_logic_vector(3 downto 0);
                                digit2 : out std_logic_vector(3 downto 0);
                                digit3 : out std_logic_vector(3 downto 0);
                                digit4 : out std_logic_vector(3 downto 0));
```

```
end Input;

architecture Behavioral of Input is
 signal inc_sig : std_logic:= '0';
 signal inc_sig1 : std_logic:= '0';
 signal inc_sig2 : std_logic:= '0';
 signal sel : std_logic_vector (3 downto 0);

begin
sel <= SW8&SW7&SW6&SW5;
decuplare:process(clock,inc)
	begin
		if clock'event and clock = '1' then
			inc_sig1 <= inc;
			inc_sig2 <= inc_sig1;
			inc_sig <= inc_sig2 and (not(inc_sig1));

		end if;
	end process decuplare ;

	process (clock,reset)
	variable digit1_v : std_logic_vector (3 downto 0);
	variable digit2_v : std_logic_vector (3 downto 0);
	variable digit3_v : std_logic_vector (3 downto 0);
	variable digit4_v : std_logic_vector (3 downto 0);

	begin
		if reset='1' then

				digit1_v := (others => '0');
				digit2_v := (others => '0');
				digit3_v := (others => '0');
				digit4_v := (others => '0');

		 elsif clock'event and clock = '1' then

			 if inc_sig ='1' then
					case (sel) is
						when "0001" => digit1_v := digit1_v+1;

						when "0010" => digit2_v := digit2_v+1;
						when "0100" => digit3_v := digit3_v+1;
						when "1000" => digit4_v := digit4_v+1;
						when others => null;
					end case;
				end if;

			if digit1_v = "1010" then – maxim 9
				digit1_v := "0000";
			end if;
			if digit2_v = "0110" then – maxim 5
				digit2_v := "0000";
			end if;
			if digit3_v = "0100" then – maxim 4
				digit3_v := "0000";
			end if;
			if digit4_v = "0011" then – maxim 2
```

```
                                        digit4_v := “0000”;
                                end if;
        end if;
        digit1 <= digit1_v;
        digit2 <= digit2_v;
        digit3 <= digit3_v;
        digit4 <= digit4_v;

        end process;

end Behavioral;
```

Figura 9. Cod VHDL corespunzător blocului de intrare pentru introducerea vaorii prestabilite

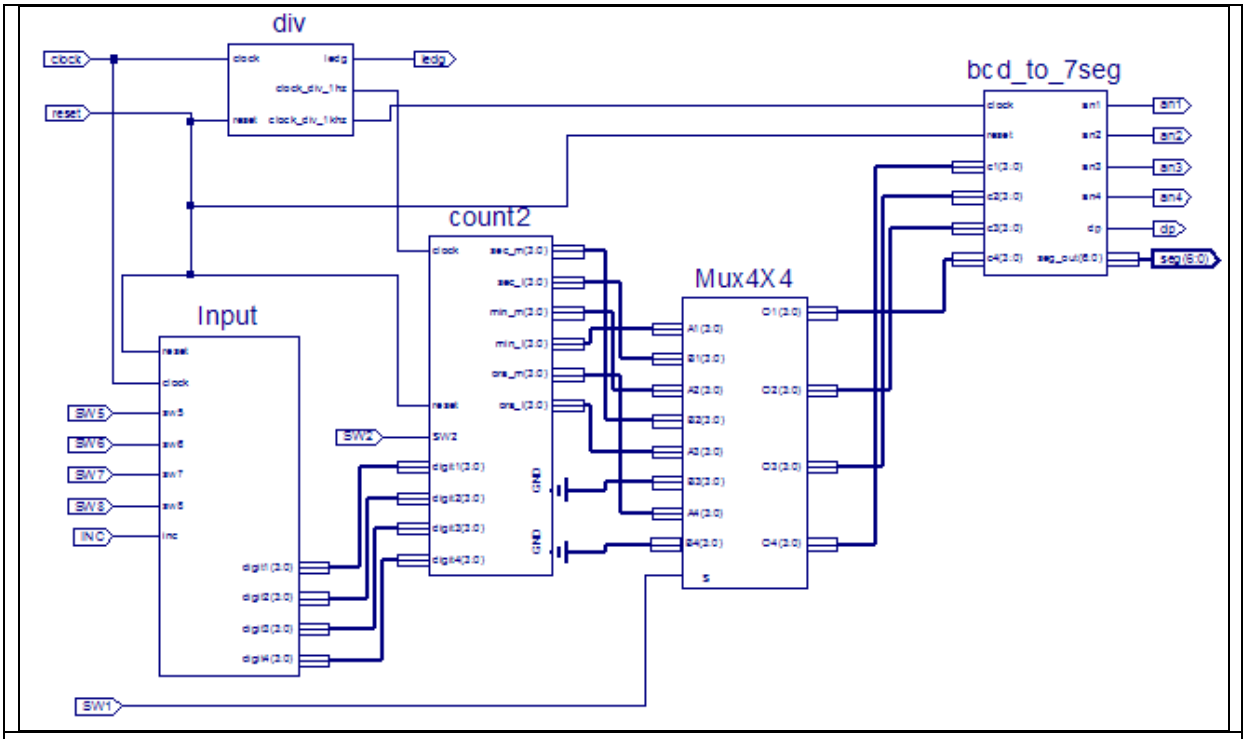


Figura 10. Ceas digital cu afişare minute, ore şi introducere valori prestabilite

```
NET “SW1”  LOC = “P23”   ;# selectare secunde
NET “SW2”  LOC = “P21”   ;# selectare ora si minute
NET “SW5”  LOC = “P11”   ;# selectare digit 1 minute
NET “SW6”  LOC = “P9”    ;# selectare digit 2 minute
NET “SW7”  LOC = “P7”    ;# selectare digit 1 ora
NET “SW8”  LOC = “P5”    ;# selectare digit 2 ora
NET “INC”  LOC = “P3”    ;# BTN1, buton de incrementare
```

Figura 11. Completare la fişierul UCF din figura 7.