Lucrare de laborator nr. 13
Sinteza circuitelor cu programul Xilinx ISE

1. Scopul lucrării

Însuşirea cunoştinţelor privind sinteza circuitelor descrise în VHDL şi utilizarea în acest scop a programului Xilinx ISE.

2. Sinteza circuitelor digitale. Principii generale.

Sinteza este procesul prin care modelul comportamental descris într-un limbaj de descriere hardware (VHDL sau Verilog) este convertit la o structură de circuit cu componente dintr-o anumită tehnologie. Rezultatul sintezei este un fişier HDL tip netlist în care modelul comportamental sintetizat este descris structural la nivel de porţi sau blocuri logice (bistabile, mux-uri, etc) specifice tehnologiei adoptate. Prin sinteză se face practic trecerea de la reprezentarea la nivelul regiştrilor de transfer RTL (**R**egister **T**ransfer **L**ogic) la reprezentarea Gate Level.

Sinteza la nivelul regiştrilor de transfer, ca parte a procesului de proiectare a circuitelor ASIC şi FPGA (Field Programable Gate-Arrays), este realizată în prezent automat de către programe de sinteză. Astfel sinteza reprezintă cea mai rapidă şi eficientă modalitate de proiectare şi generare a circuitelor. Procesul de sinteză la nivel RTL în cadrul unui program se desfăşoară uzual conform diagramei din figura 1.

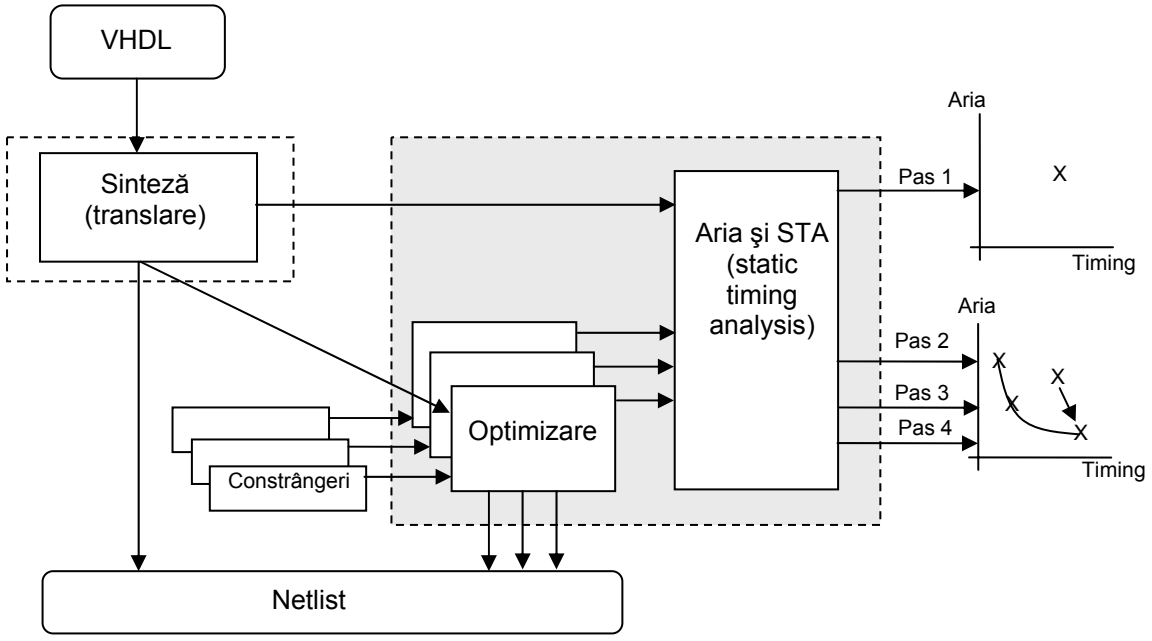


Figura 1 – *Diagrama procesului de translare şi optimizare în cadrul sintezei la nivel RTL*

În diagrama din figura 1 se observă că prin sinteză se poate efectua translarea (transformarea) direct la descrierea tip netlist, fără optimizare. Totuşi, în practică, instrumentele de sinteză utilizează algoritmi de optimizare privind raportul arie-timp. Astfel, iniţial programul de sinteză furnizează o soluţie de circuit cu un anumit raport arie-timp, corespunzător pasului 1 din figura 1. În continuare proiectul este optimizat de mai multe ori corespunzător diferitelor constrângeri. De exemplu, în diagrama din figura 1 pentru cele trei constrângeri proiectul este optimizat de 3 ori (paşii 2, 3 şi 4) ceea ce conduce la trei puncte diferite pe curba arie-timp. Uzual, metodologia de optimizare presupune mai intâi

optimizarea din punct de vedere a ariei ocupate şi apoi optimizare numai pentru aspectele de timing dacă există constrângeri de acest tip care nu sunt îndeplinite.

Procesul de sinteză constă din mai multe etape de transformări şi optimizări, trecerea unui proiect de la descrierea comportamentală VHDL la descrierea netlist realizându-se prin mai multe nivele intermediare, care corespund la diferite nivele de abstractizare a proiectului, aşa cum este reprezentat în figura 2.

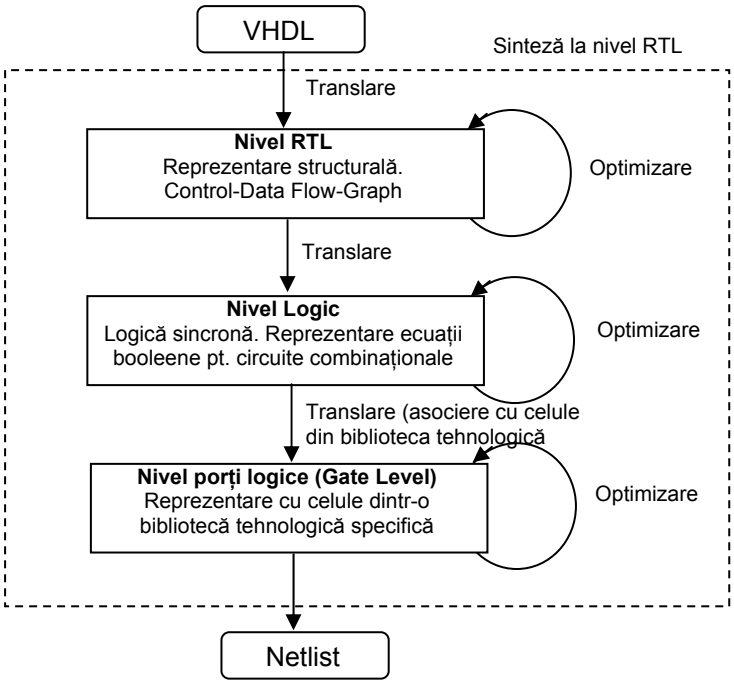


Figura 2 – *Diagrama proceselor de translare şi optimizare internă în cadrul sintezei la nivelul regiştrilor de transfer RTL*

Optimizarea automată are loc la fiecare din nivelele intermediare din cadrul sintezei RTL şi este ghidată de constângerile definite de utilizator. Constrângerile furnizează ţintele pe care procesele de translare şi optimizare trebuie să le atingă. Constrângerile tipice care pot fi impuse în cadrul instrumentelor de sinteză actuale sunt pentru obţinerea ariei minimale sau pentru timp de propagare minimal. De asemenea, constrângeri pot fi pentru puterea disipată şi, în viitor, pentru layout şi packaging.

Un exemplu de optimizare la nivel de poartă logică (gate-level) este prezentat în figura 3. Astfel, dacă circuitul generat iniţial este cel din figura 3a conţinând 24 de tranzistoare, după optimizare din punct de vedere al ariei rezultă circuitul din figura 3b având numai 14 tranzistoare. În figura 3b blocul OR2-NAND3 este considerat o singură celulă cu structura alcătuită din 8 tranzistoare care implementează funcţia f= not[(C+D)BC].

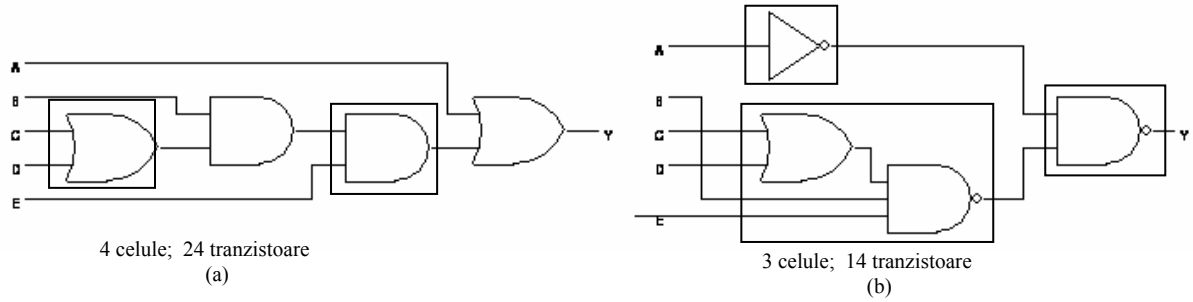


Figura 3 – *Exemplu optimizare la nivel Gate-Level. a) înainte de optimizare b) după optimizare*

3. Modelul VHDL care va fi sintetizat

În cadrul acestei lucrări se va exemplifica sinteza unui model descris ca o maşină cu stări finite (FSM) a cărei diagramă de stări este prezentată în figura 4. Modelul corespunde unui circuit de comandă a vitezei unui automobil şi are ca intrări următoarele semnale: Clock, Keys, Brake şi Accelerate iar ieşirea este semnalul Speed.

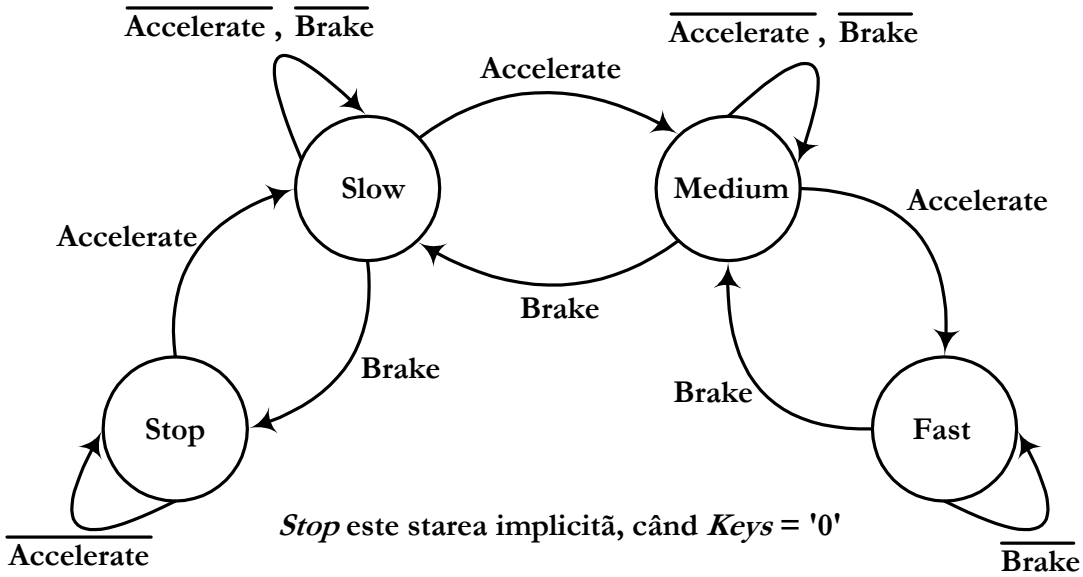


Figura 4 – *Diagrama de stări a modelului*

Fişierul care conţine codul VHDL corespunzător modelului FSM este prezentat mai jos. În cadrul fişierului este definit un package intitulat Enum_State_Encode_Types în care este definit tipul STATE_TYPE şi un atribut prin care celor 4 valori ale tipului STATE_TYPE li se asociază valori binare. Entitatea corespunzătoare modelului este denumită FSM_CAR_SPEED_CNTL iar arhitectura asociată este denumită RTL.

Fişier **Car_FSM.vhd**:

```
--CAR_pack
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

package Enum_State_Encode_Types is
       attribute Enum_State_Type_Encoding :string;
       type STATE_TYPE is
              (Stop, Slow, Medium, Fast);
       attribute Enum_State_Type_Encoding of STATE_TYPE: type is
              ("11 10 01 00");
end;

--Car_FSM
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;
use work.Enum_State_Encode_Types.all;

entity FSM_CAR_SPEED_CNTL is
       port (Clock, Keys, Brake, Accelerate: in std_logic;
                           Speed:out STATE_TYPE);
end;-- entity  FSM_CAR_SPEED_CNTL;
```

```
architecture RTL of FSM_CAR_SPEED_CNTL is
	signal   NextSpeed :STATE_TYPE;
	signal   Speed_s   :STATE_TYPE;
begin

	FSM_COMB:process(Keys, Brake, Accelerate, Speed_s, NextSpeed)
	begin
	--            "Speed_s" este un semnal intern care porteaza semnalul "Speed",
	--astfel incat "Speed" sa ramana de tip "out"
		case Speed_s is
			when Stop =>
				if (Accelerate='1') then
					NextSpeed <= Slow;
				else
					NextSpeed <= Stop;
				end if;
			when Slow =>
				if (Brake='1') then
					NextSpeed <= Stop;
				elsif (Accelerate='1') then
					NextSpeed <= Medium;
				else
					NextSpeed <= Slow;
				end if;
			when Medium =>
				if (Brake='1') then
					NextSpeed <= Slow;
				elsif (Accelerate='1') then
					NextSpeed <= Fast;
				else
					NextSpeed <= Medium;
				end if;
			when Fast =>
				if (Brake='1') then
					NextSpeed <= Medium;
				else
					NextSpeed <= Fast;
				end if;
			when others =>
					NextSpeed <= Stop;
		end case;
	end process FSM_COMB;
	FSM_SEQ: process (Clock,Keys)
	begin
		if (Keys='0') then
			Speed_s <= Stop;
		elsif falling_edge(Clock) then
			Speed_s <= NextSpeed;
		end if;
		Speed <= Speed_s;
	end process FSM_SEQ;

end; -- architecture RTL;
```

3.2 Testarea modelului

Fişierul care conţine codul VHDL pentru **simularea modelului** este următorul:
Fişier **Car_Test.vhd**:

```
-- Modulul generator de clock:
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

entity Clock_Gen is
	port (Clock: out std_logic);
end Clock_Gen;
```

```
architecture SPEC of Clock_Gen is
       constant clk_prd: time := 200 ns;
       signal   int_clk: std_logic := '0';

begin
       int_clk <= not int_clk after clk_prd/2;
       Clock <= int_clk;
end SPEC;

-----------------------------------------------------------------
-- Generarea semnalelor de test pentru CAR_SPEED_CONTROLER:

library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;
use work.Enum_State_Encode_Types.all;

entity FSM_CAR_Control is
             port (Speed: in STATE_TYPE;
                                Clock, Keys, Brake, Accelerate: out std_logic);
end FSM_CAR_Control;

architecture DRV of FSM_CAR_Control is
       constant clk_prd: time := 200 ns;
       signal Clock_in, Keys_in, Brake_in, Accelerate_in: std_logic;
       signal CountStop, CountSlow, CountMedium, CountFast: STATE_TYPE := Stop;
       component Clock_Gen
             port (Clock: out std_logic);
       end component;

begin
       Sursa: Clock_Gen  port map (Clock => Clock_in);
       process
       begin
         Keys_in <= '0' after clk_prd/2,'1' after 2*clk_prd, '0' after 14*clk_prd;
         Brake_in <= '0' after clk_prd/2, '1' after 4*clk_prd, '0' after 6*clk_prd,
'1' after 11*clk_prd, '0' after 14*clk_prd;
        Accelerate_in <= '0' after clk_prd/2, '1' after clk_prd, '0' after
5*clk_prd, '1' after 7*clk_prd, '0' after 10*clk_prd;
         wait for 15*clk_prd;

       end process;

       Accelerate <= Accelerate_in;
       Brake <= Brake_in;
       Keys <= Keys_in;
       Clock <= Clock_in;

end DRV;


-----------------------------------------------------------------
-- CAR TEST BENCH:
-- Este autoconsistent si contine modulul CAR_SPEED_CONTROLER
--   si modulul ce genereaza semnalele de test pentru acesta.

library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;
use work.Enum_State_Encode_Types.all;

entity Car_Test_Bench is
end Car_Test_Bench;

architecture STRUCT of Car_Test_Bench is
       signal Clock: STD_Logic;
       signal Keys, Brake, Accelerate: STD_Logic;
       signal Speed : STATE_TYPE;
```

```
        component FSM_CAR_SPEED_CNTL
                port(Clock, Keys, Brake, Accelerate: in std_logic;
                        Speed:out STATE_TYPE);
        end component;

        component FSM_CAR_Control
                port (Speed: in STATE_TYPE;
                        Clock, Keys, Brake, Accelerate: out std_logic);
        end component;

begin
        Car: FSM_CAR_SPEED_CNTL
                port map (    Clock => Clock,
                              Keys => Keys,
                              Brake => Brake,
                              Accelerate => Accelerate,
                              Speed => Speed);
        Driver: FSM_CAR_Control
                port map (    Speed => Speed,
                              Clock => Clock,
                              Keys => Keys,
                              Brake => Brake,
                              Accelerate => Accelerate);
end STRUCT;
```

4. Sinteza cu programul Xilinx ISE

4.1 Preliminarii

În acest capitol se vor prezenta etapele care trebuie parcurse pentru a realiza sinteza unui model VHDL cu ajutorul programului Xilinx ISE pe o placă FPGA Spartan-3. Modelul VHDL ce va fi utilizat pentru sinteză este cel al maşinii cu stări finite (FSM) descris de entitatea FSM_CAR_SPEED_CNTL şi arhitectura RTL în cadrul fişierului sursă Car_FSM.vhd prezentat în capitolul 3.

Programul Xilinx ISE WebPACK este un produs gratuit al firmei Xilinx (www.xilinx.com), destinat sintezei, simulării şi implementării proiectelor în circuite FPGA.

Placa de dezvoltare FPGA tip Xilinx Spartan-3 ale cărei caracteristici vor fi considerate pentru efectuarea sintezei este prezentată în figura 5.

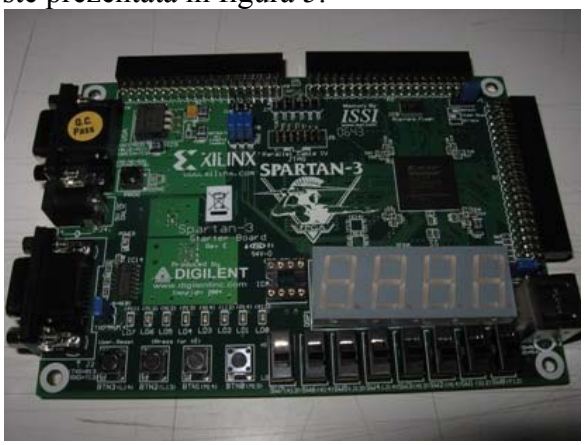


Figura 5 – *Placa de dezvoltare Xilinx Spartan-3*

Înainte de a porni programul Xilinx ISE, **creaţi** subdirectorul **XilinxGS** în directorul grupei din care faceţi parte, unde **G** – cifra grupei (1, 2, etc), iar **S**-semigrupa (A sau B).

De exemplu, în cazul grupei **G5401A**, se crează subdirectorul **Xilinx1A**:

….\G5401A\Xilinx1A

În directorul XilinxGS creați subdirectorul pentru proiect, **car_speed**. De exemplu:

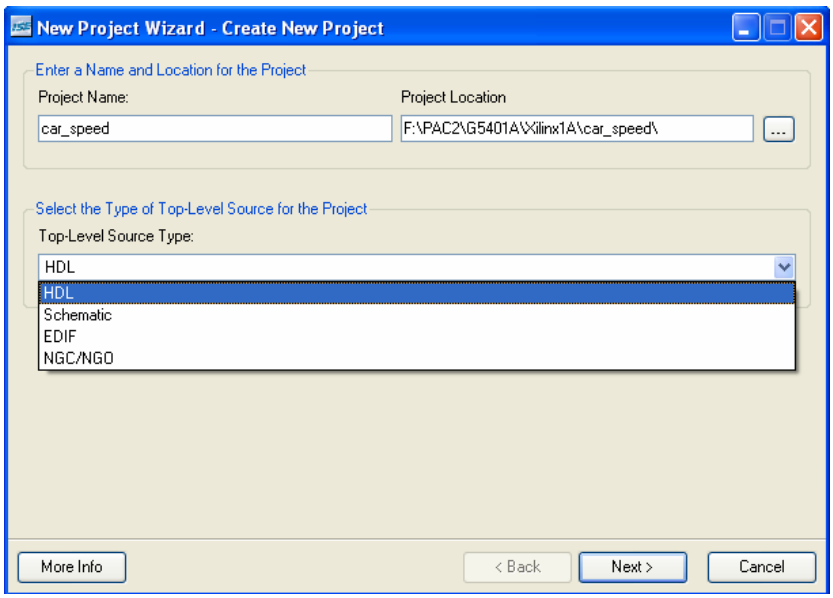
….\G5401A\Xilinx1A\car_speed

Copiați fişierul **Car_FSM.vhd** în directorul **car_speed**.

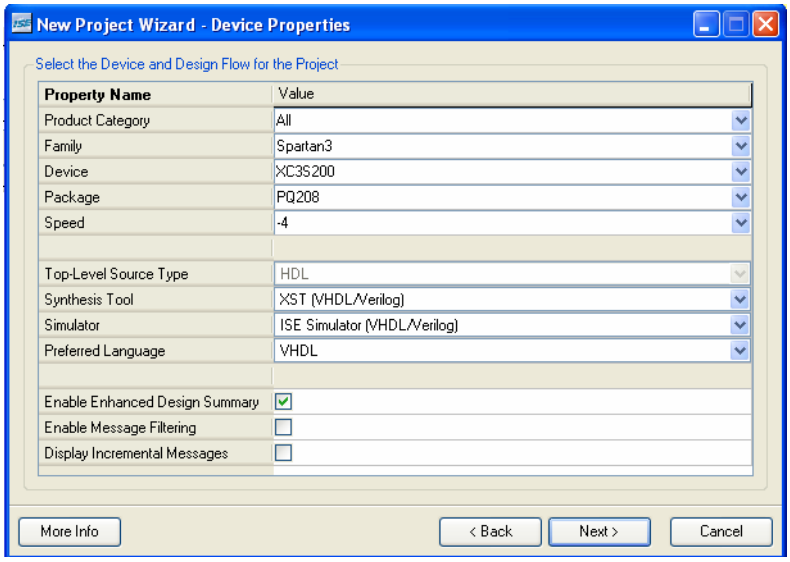
După aceşti paşi preliminari se poate trece la pornirea şi efectuarea etapelor pentru sinteză cu programl Xilinx ISE.

4.2 Etapele sintezei cu programul Xilinx ISE

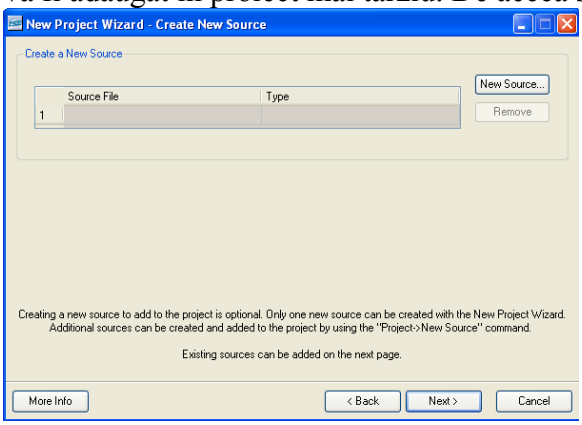
1. **Se porneşte programul Xilinx ISE** cu dublu-click pe icon-ul  de pe desktop.
2. **Se crează un proiect nou**:
   - Se dă click pe **File**, apoi se alege **New Project.** Apare fereastra de mai jos.
   - Se introduce numele proiectului (**car_speed)**
   - La rubrica **Project Location** se alege directorul de lucru, de exemplu **...\Xilinx1A\car_speed**.
   - La rubrica **Top-Level Source Type** se alege **HDL**.
   - Click pe butonul **Next**.



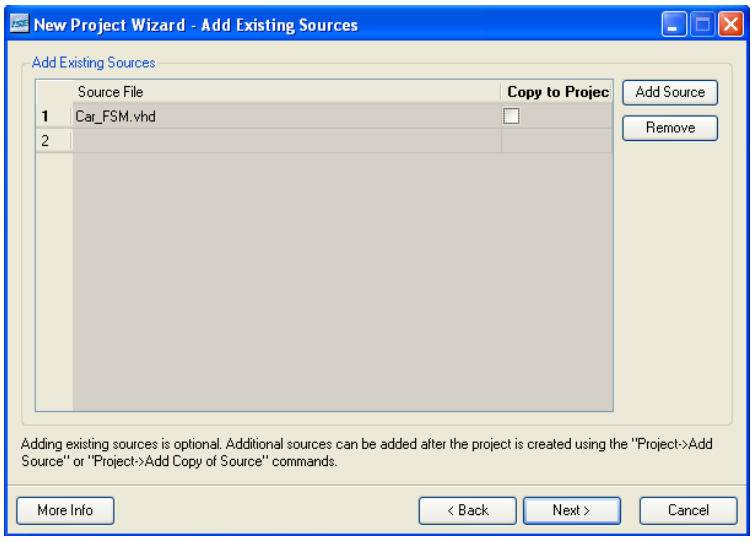
3. Apare o nouă fereastră (**Device Properties**) prezentată mai jos, în care trebuie selectate informații despre placa de dezvoltare (*hardware device*) considerată pentru sinteză şi despre instrumentele ce vor fi folosite în etapele (*flow*-ul) proiectului. Informațiile despre placă pot fi aflate de pe aceasta. În fereastra **Device Properties** se alege:
   - Pentru **Family**, se alege **Spartan3**
   - Pentru **Device**, se alege **XC3S200**
   - Pentru **Package**, se alege **PQ208**
   - Pentru **Speed** (viteză), se alege **-4**
   - Pentru **Synthesis Tool**, se alege **XST (VHDL/Verilog)**
   - Pentru **Simulator**, se alege **ISE Simulator (VHDL/Verilog)**
   - Pentru **Preferred Language**, se alege **VHDL**.
   - În final se apasă **click** pe buronul **Next**.



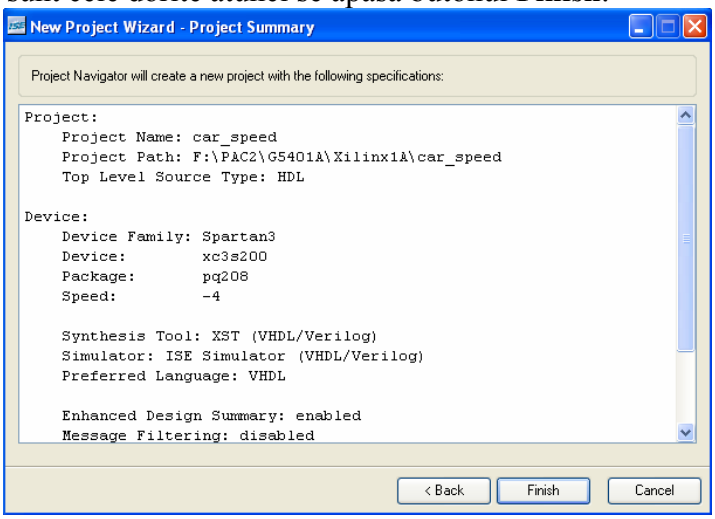
4. În următoarea fereastră (**Create New Source**) suntem întrebaţi dacă dorim să creăm un fişier sursă nou. Deoarece fişierul sursă pe care îl vom considera deja este creat (Car_speed.vhd), acesta va fi adăugat în proiect mai târziu. De aceea se apasă **Next**.



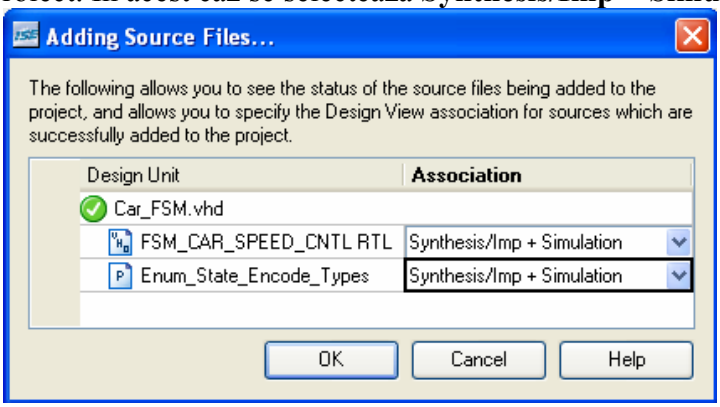
5. În următoarea fereastră (**Add Existing Sources**) putem să selectăm fişierul sursă al proiectului. Se apasă butonul **Add Source** şi se selectează fişierul **Car_FSM.vhd** după care se apasă **Next**. Fişierul sursă poate fi adăugat şi mai târziu, după crearea proiectului, selectând *Project –> Add Source*.



6. În fereastra următoare care apare (**Project Summary**) se afişează un sumar al proiectului care va fi creat. În cazul în care trebuie efectuate modificări se apasă **Back**. Dacă informaţiile afişate sunt cele dorite atunci se apasă butonul **Finish**.



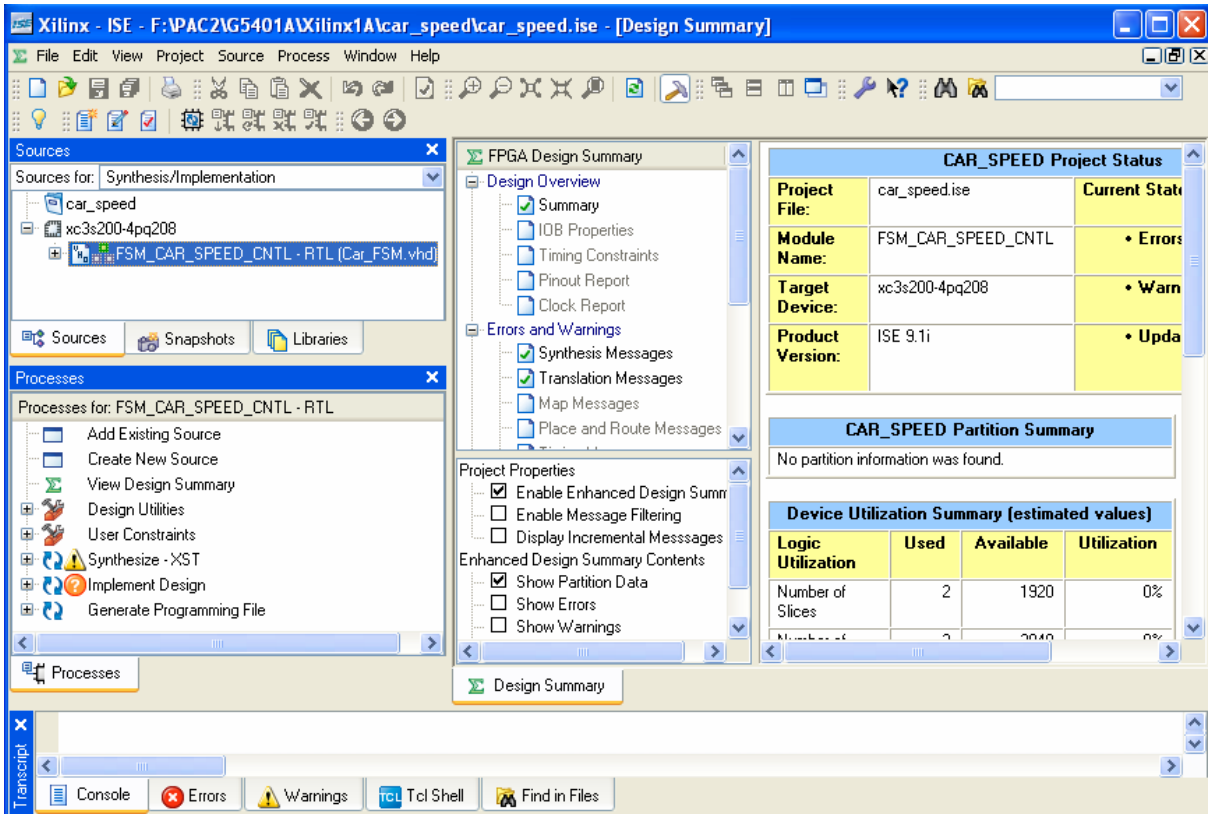
7. Înainte de crearea proiectului apare fereastra **Adding Source Files** pentru a vedea situaţia fişierelor sursă adăugate la proiect şi elementele conţinute de acestea. De asemenea, se poate selecta ce variante de vizualizare dorim să asociem pentru elementele din fişierele sursă adăugate în proiect. În acest caz se selectează **Synthesis/Imp + Simulation**, apoi **OK**.



8. Odată terminată etapa creării proiectului, fereastra principală a programuuli *Xilinx ISE* arată ca în figura de mai jos. În partea stângă a ferestrei se pot vedea două sub-ferestre (câmpuri) având butoanele **Sources** (selectat în figură), **Snapshots** şi **Libraries**, respectiv **Processes**.

9. În câmpul **Sources**, la rubrica **Sources for**: se selectează **Synthesis/Implementation**. De asemenea, se apasă **click-dreapta** pe unităţile de proiect din fişierul sursă FSM_CAR_SPEED_CNTL +RTL (Car_FSM.vhd), apoi **Properties**. Se verifică dacă asocierea pentru unităţile de proiect din fişier este **Synthesis/Imp+Simulation**.

10. Dacă în câmpul **Sources** este selectat cu click FSM_CAR_SPEED_CNTL +RTL (Car_FSM.vhd), atunci în câmpul **Processes** sunt vizibile comenzile (procesele) prin care se pot stabili constrângerile (*User constraints*) în vederea sintezei, procesele pentru sinteza propriu-zisă (*Synthesize-XST*) sau procese pentru implementarea după sinteză a proiectului pe placa FPGA (*Implement Design*). Pentru a vedea procesele ce pot fi executate în cadrul fiecărei categorii, se apasă butonul '+' pentru expandare.



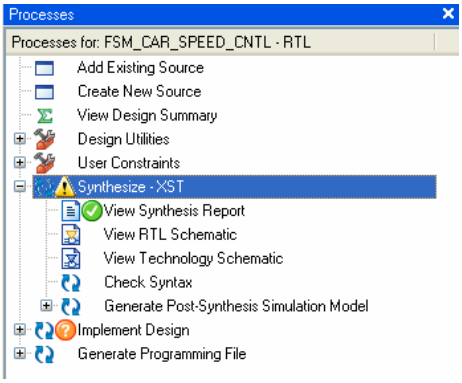
Astfel, prin expandarea ***User Constraints*** se pot executa procese pentru crearea constrângerilor legate de timing (*Create Timing Constraints*), pentru asocierea porturilor circuitului sintetizat la pinii circuitului FPGA (*Asign Package Pins*) sau pentru crearea constrângerilor legate de arie (*Create Area Constraints*).

În cadrul acestei lucrări **nu vom impune nici o constrângere**. Pentru a vedea în ce constă şi cum pot fi stabilite constrângerile, se poate acționa pe rând **dublu-click pe procesele menționate mai sus** pentru a fi executate. După ce se acționează prima dată **dublu-click** pe unul din procesele din cadrul ***User Constraints***, se cere confirmarea pentru crearea unui fişier pentru constrângeri (*Implementation Constraint File*–UCF). **În ferestrele care apar prin executarea proceselor nu se va edita nimic, acestea doar se vor vizualiza după care se vor închide**.

11. În continuare se poate trece la **efectuarea sintezei** propriu-zise a proiectului. Pentru aceasta, având selectat FSM_CAR_SPEED_CNTL +RTL (Car_FSM.vhd) în zona **Sources**, în câmpul **Proceses** se acționează **dublu-clik** pe **Synthesize – XST**.

Prin sinteză proiectul este transformat într-o structură cu componente logice (porți, bistabile, LUT-uri – lookup table, etc) din cadrul circutului FPGA al plăcii Spartan-3. Când procesul de sinteză s-a terminat, acesta este anunțat prin mesajul *Process „Synthesize" completed successfully*.

12. După terminarea sintezei, în câmpul **Processes**, prin expandarea procesului **Synthesize – XST** se pot vizualiza şi executa celelalte procese asociate acestuia (*View Synthesis Report*, *View RTL Schematic*, *View Technology Schematic*, etc)



13. În câmpul **Processes** acționați **dublu-click** pe procesul **View RTL Schematic**. Va apare o fereastră cu interfața (top-level) circuitului FSM_CAR_SPEED_CNTL. Pentru a vizualiza structura internă a blocurilor din ierarhia schemei la nivel RTL (nivel regiştri de transfer) se acționează dublu-click pe blocul dorit. De asemenea, pentru a intra sau ieşi dintr-un nive ierarhic se pot folosi butoanele  din partea superioară a ferestrei programului Xilinx ISE. În figura de mai jos se pot vedea cele 3 nivele ierarhice din schema la nivel RTL a circuitului.

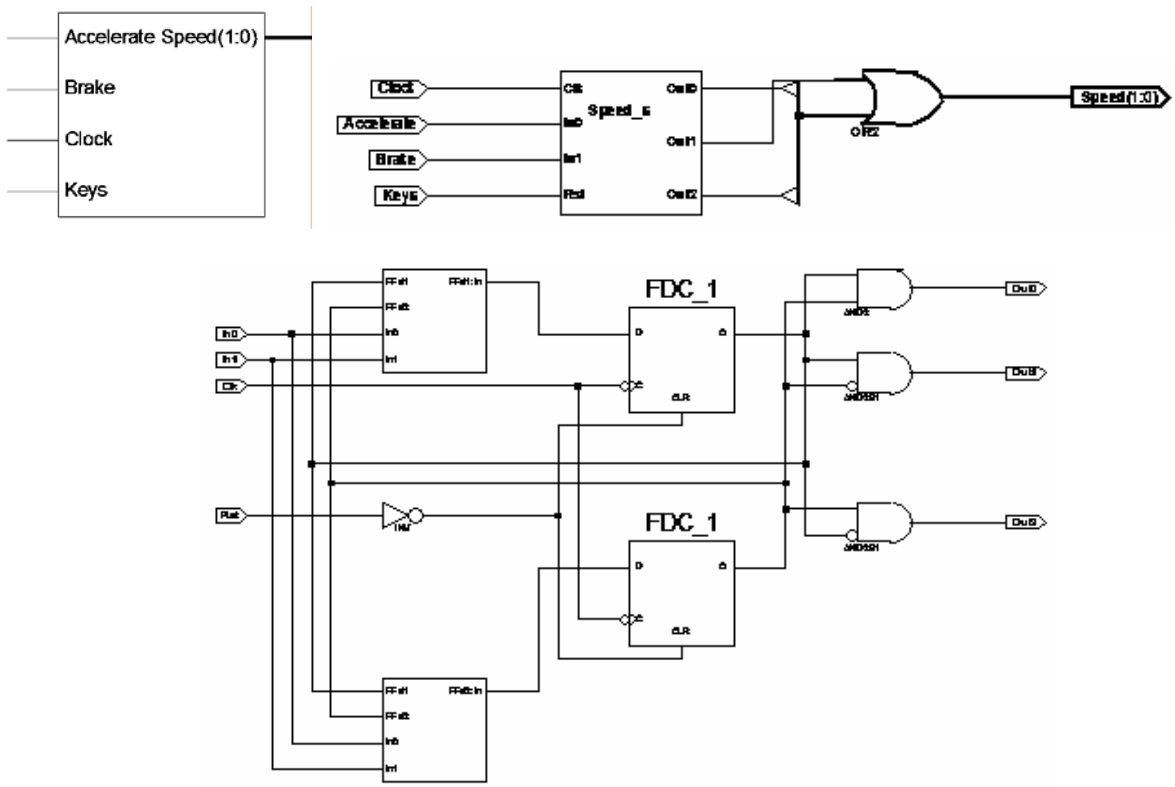


Figura 6 – *Schemele la nivel RTL a circuitului sintetizat*

După vizualizarea schemelor la nivel RTL închideți fereastra în care acestea au fost reprezentate.

14. În câmpul **Processes** acționați dublu-click pe procesul **View Technology Schematic**. Acest proces va determina afişarea schemei circuitului sintetizat cu componentele din biblioteca tehnologică (componentele din circuitul FPGA al plăcii), şi anume inversoare, buffere, bistabile sau blocuri LUT. Schema la nivel tehnologic este

reprezentată pe diverse nivele ierarhice care pot fi parcurse similar ca în cazul vizualizării schemei la nivel RTL. În figura 7 sunt reprezentate schemele la nivel tehnologic pentru circuitul sintetizat.

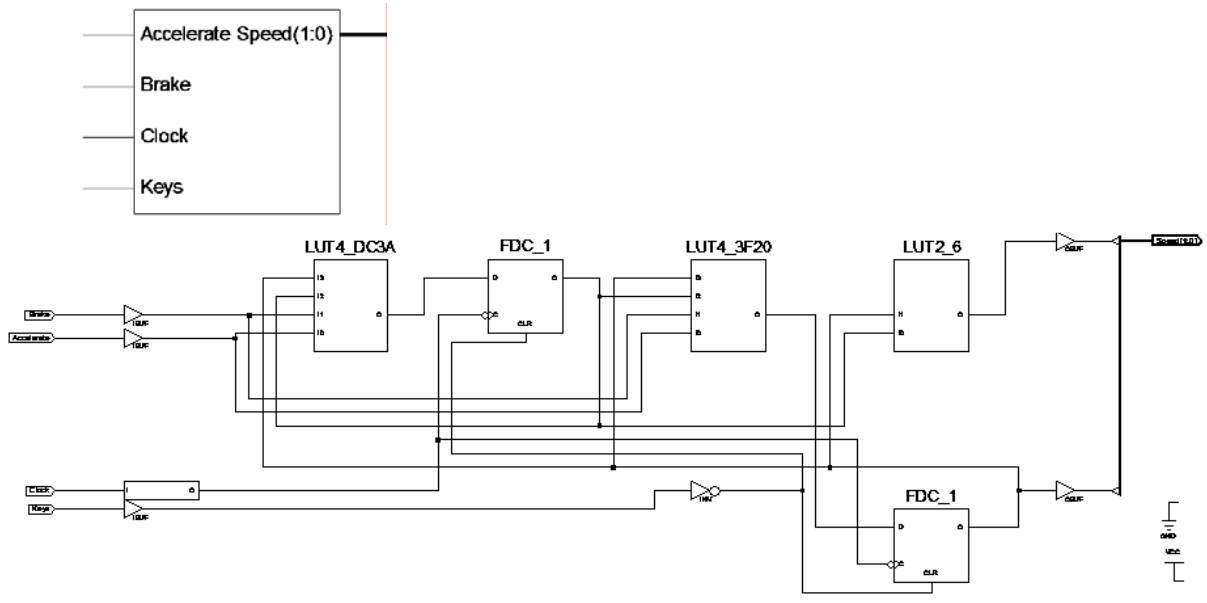
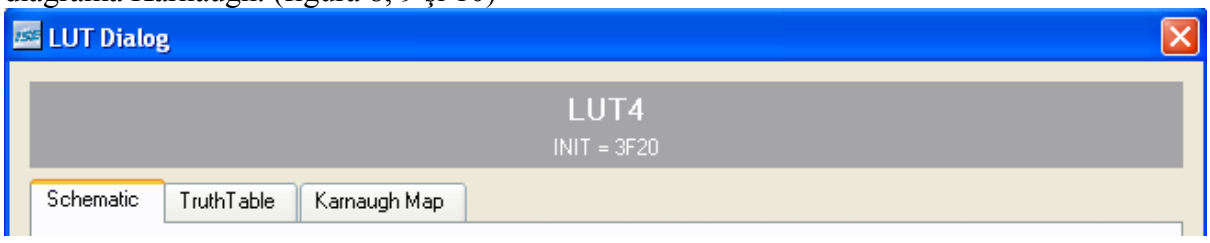


Figura 7 – *Schema la nivel tehnologic FPGA a circuitului sintetizat*

Dacă se dă **dublu-clik** pe blocul LUT4_3F20 din schema la nivel tehnologic, se deschide ferestra **LUT Dialog** (figura de mai jos) în care, cu butoanele *Schematic*, *Truth Table* şi *Karnaugh Map* pot fi vizualizate schema internă a acestuia, tabelul de adevăr şi diagrama Karnaugh. (figura 8, 9 şi 10)



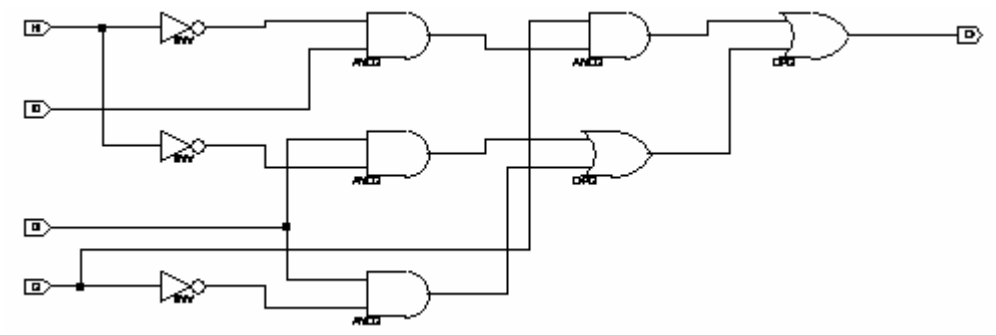


Figura 8 – *Schema tehnologică pentru blocul lookup table LUT4_3F20*

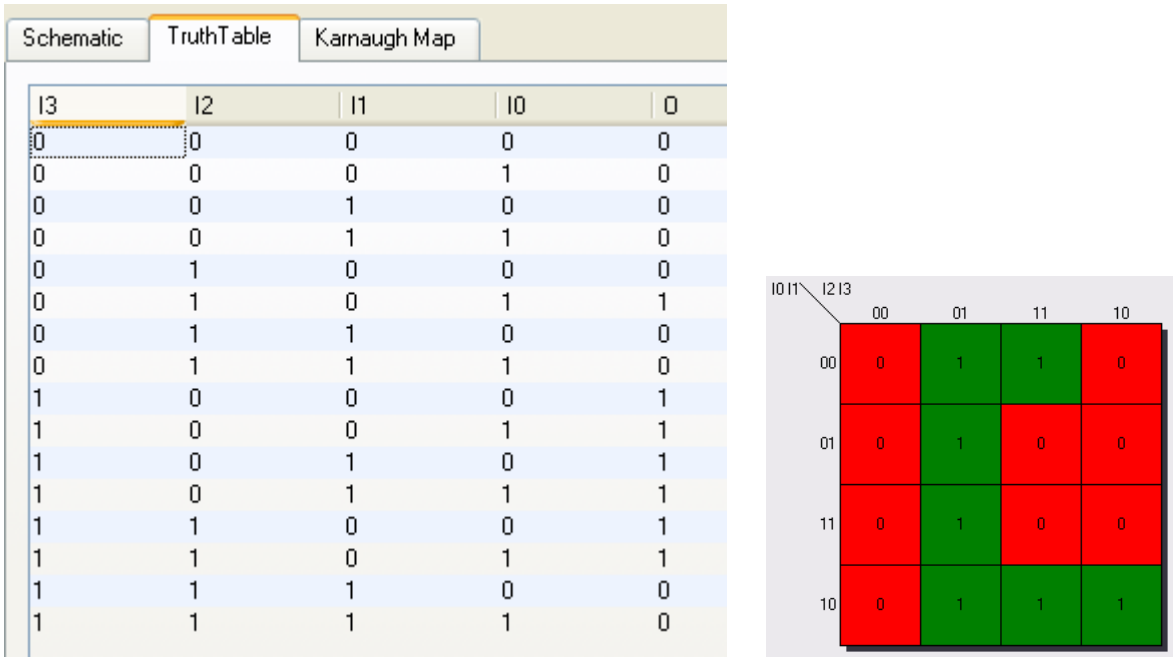


Figura 9 – *Tabelul de adevăr şi diagrama Karnaugh a blocului LUT4_3F20*

5. Lucru individual

În directorul .../Xilinx1A/ creați un subdirector numit FSM_10A, copiați aici fişierul sursă FSM_A din lucrarea nr. 10 şi realizați sinteza circuitului corespunzătoare descrierii respective, procedând similar ca în cazul sintezei proiectului car_speed.