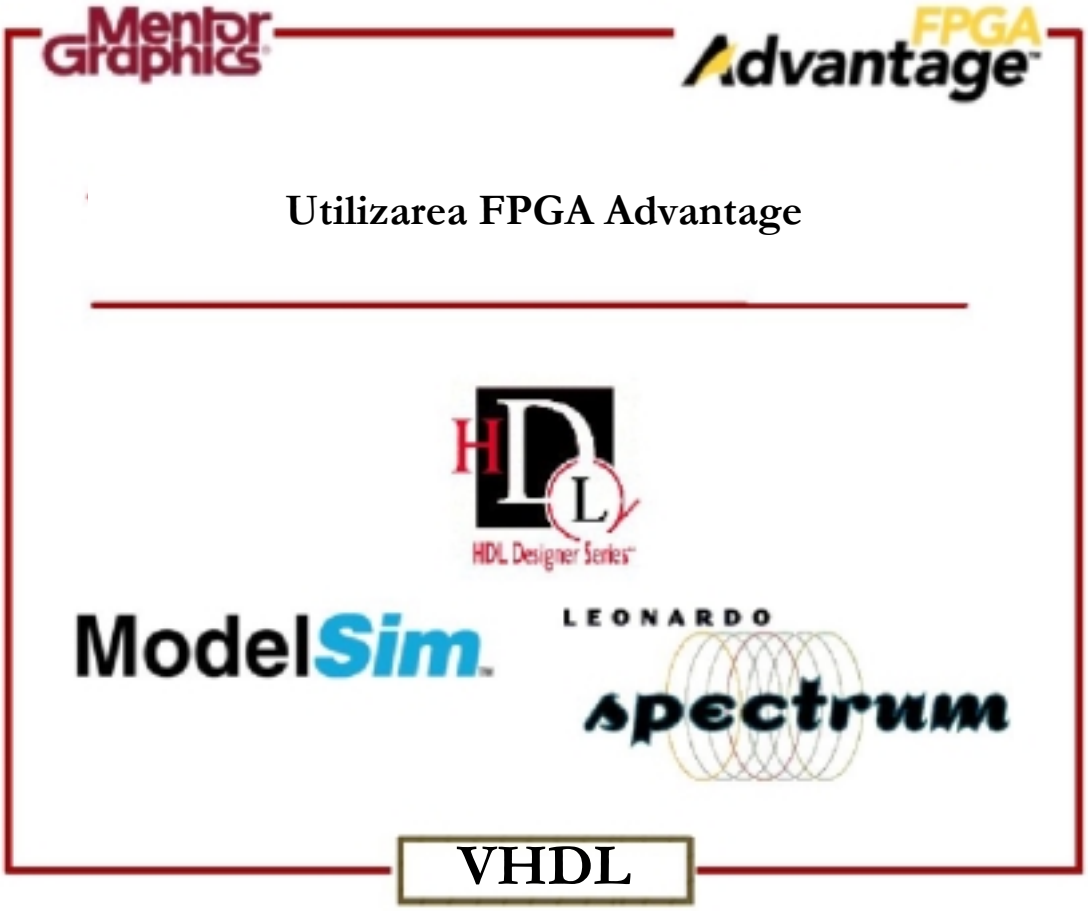
Universitatea Tehnică "Gheorghe Asachi" Iaşi
Facultatea de Electronică şi Telecomunicaţii

Proiectare Asistată de Calculator
Lucrări de Laborator



Cuprins:

1. Implementarea VHDL a unei FSM

În această parte se propune studiul cu ajutorul instrumentelor FPGA Advantage a unei maşini cu stări finite (FSM), a cărei diagramă se prezintă în figura 1. În figura 2 sunt ilustrate formele de undă care stimulează FSM. Apoi se prezintă implementarea VHDL a acesteia precum şi o structură de test (test bench), pe baza formelor de undă.

Studiaţi cu atenţie modul de implementare a FSM în arhitectura FSM_CAR_SPEED_CNTL_2, prin cele două procese: FSM2_COMB şi FSM2_SEQ!

Observaţi modul de realizare a test bench-ului, având în vedere formele de undă din figura 2 şi remarcaţi prezenţa procesului pentru atribuirea secvenţială de valori!

1. Precizaţi care front al clock-ului este activ în sistem şi argumentaţi! Nu uitaţi de timpul de simulare şi gândiţi-vă că în interiorul unui proces atribuirea de valori unui semnal este interpretată ca un latch dacă nu este clock sau ca un bistabil (cazul de faţă), dacă procesul include un clock!

2. Găsiţi o soluţie pentru a determina semnalul Speed să se modifice odată cu Speed_s!

3. Ce părere aveţi despre sincronicitatea (faţă de clock) a evenimentelor din procesul FSM2_COMB?

4. Precizaţi succesiunea de stări a FSM pentru formele de undă din figura 2!

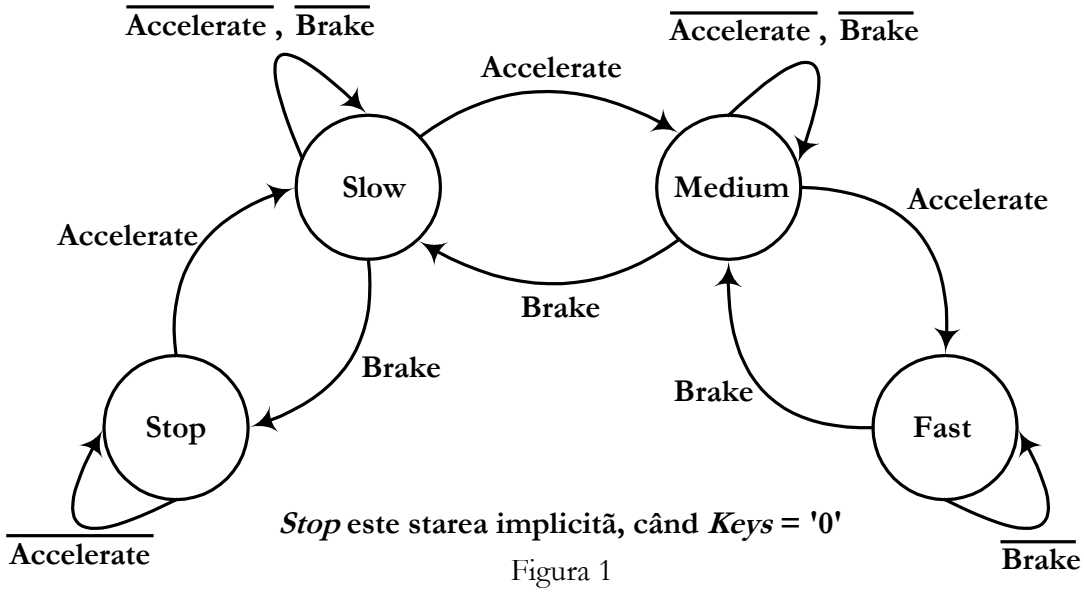


Figura 1

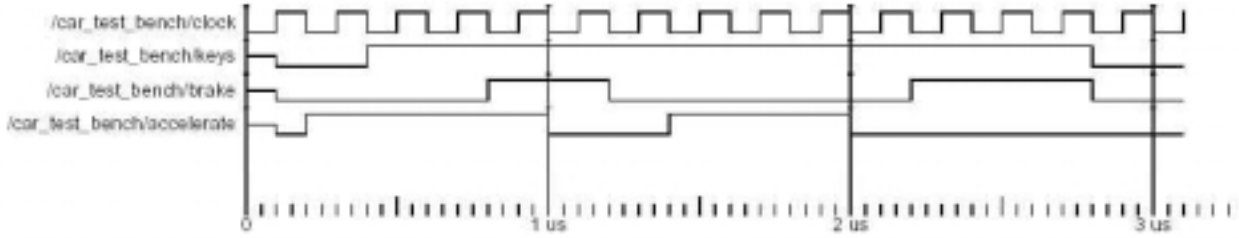


Figura 2

--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_2 is
	port (Clock, Keys, Brake, Accelerate: in std_logic;
				Speed:out STATE_TYPE);
end;-- entity  FSM_CAR_SPEED_CNTL_2;

architecture RTL of FSM_CAR_SPEED_CNTL_2 is
	signal   NextSpeed :STATE_TYPE;
	signal   Speed_s   :STATE_TYPE;
begin

	FSM2_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 FSM2_COMB;
```

```
	FSM2_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 FSM2_SEQ;

end; -- architecture RTL;

--=============================================================

-- 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_2
		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_2
		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;
```

2. Pornirea FPGA Advantage şi deschiderea unui proiect

Se accesează programul pe calea ***Start > Programs > FPGA Advantage > FPGAdv Pro*** şi pe ecran va apare fereastra principală "***HDL Design Browser***", care arată precum în figura 3.

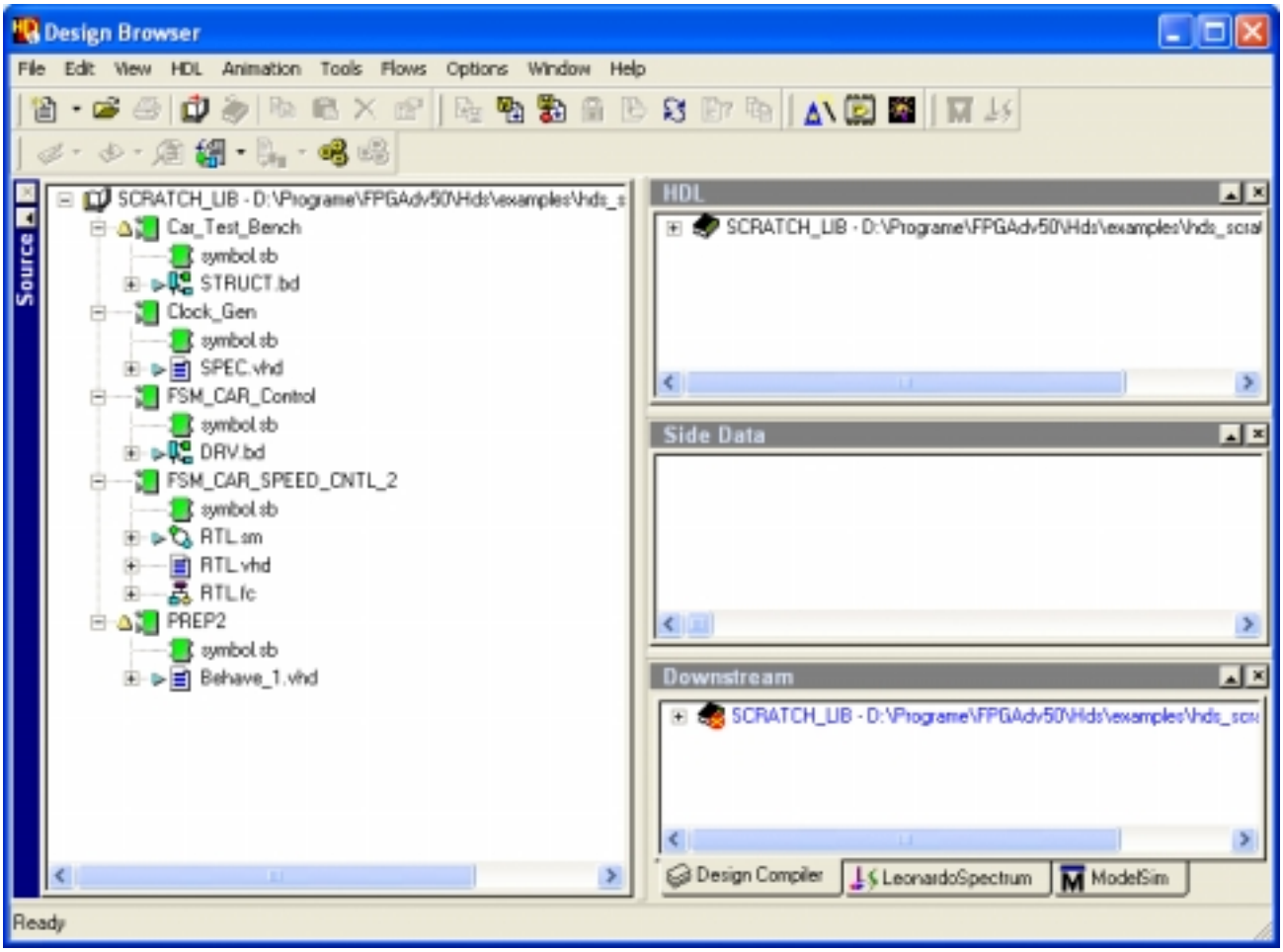


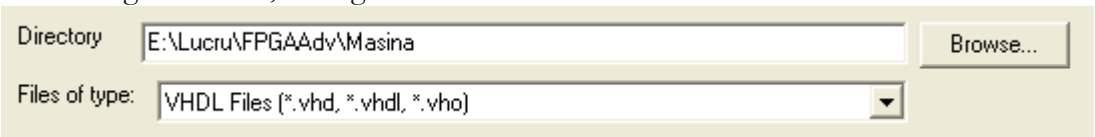
Figura 3

Se propune realizarea unui design pe baza surselor VHDL deja existente. ***FPGAdv*** poate recupera coduri VHDL sau Verilog si le poate converti coduri VHDL proprii şi diagrame grafice HDS.

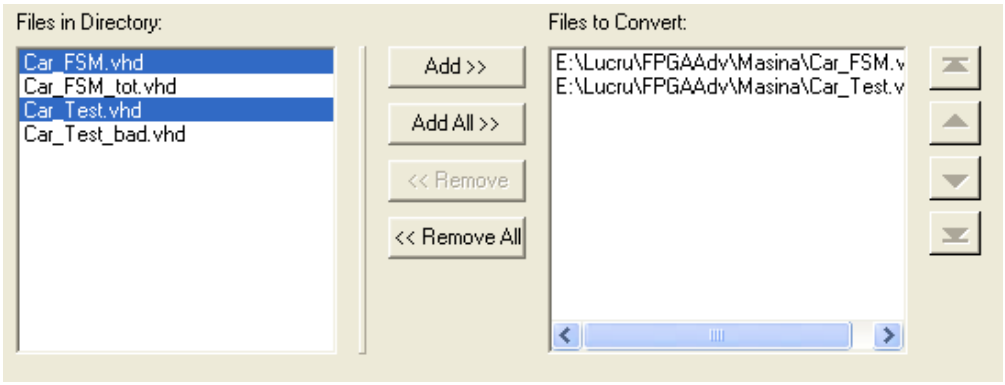
1. Pentru aceasta, mai întâi se deschide un nou browser pe calea: ***File > New Design Browser*** şi se închide cel anterior.

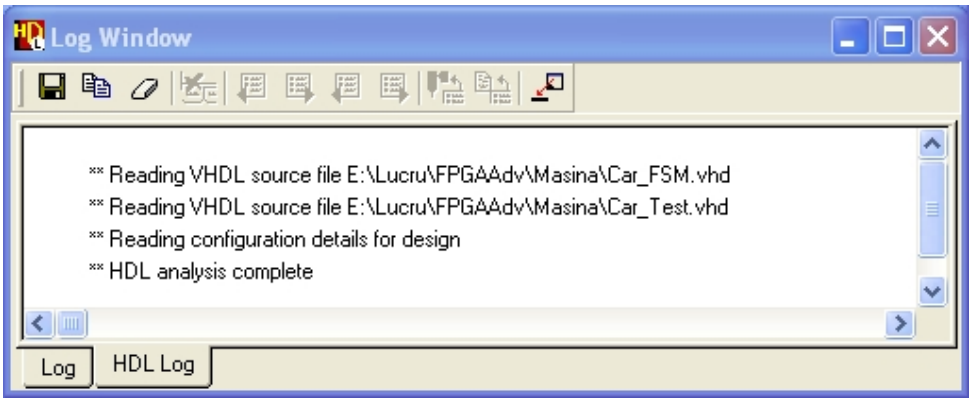
2. Se selectează ***HDL > Full HDL Import*** şi se bifează ***Specify HDL files*** şi ***VHDL*** şi se trece pasul următor apăsând butonul ***Next***.

3. În dialogul următor, se alege cu ***Browse*** calea către sursele VHDL.

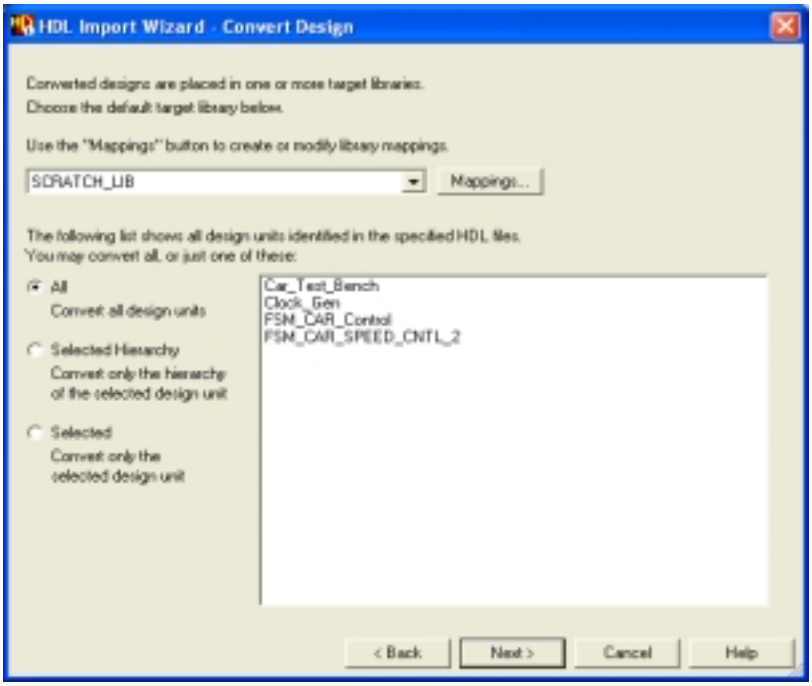


4. Se selectează ***Car_FSM.vhd*** şi ***Car_Test.vhd*** utilizând *butonul din stânga al mouse-ului* şi tasta *Ctrl*. Apoi se apasă butonul Add >> urmat de Next > pentru a converti sursele VHDL. Rezultatele se pot remarca în fereastra ***Log Window***.

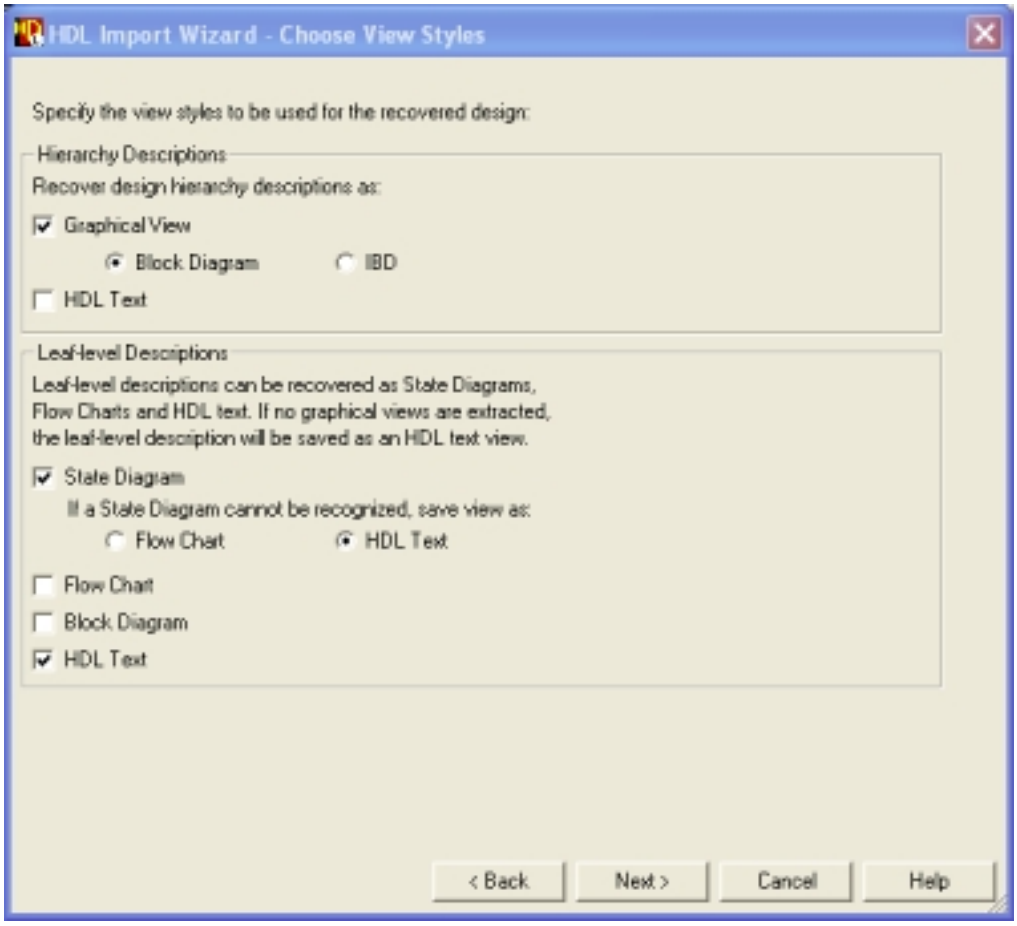




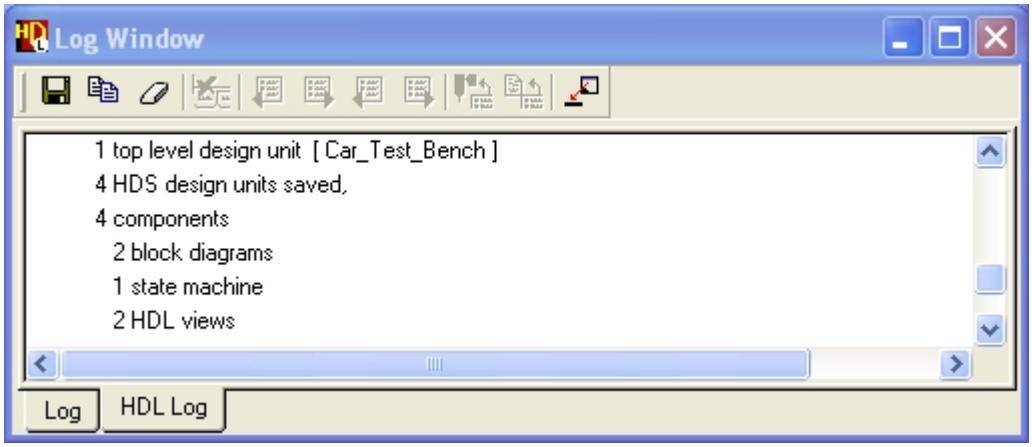
5. În dialogul următor se verifică faptul că sunt selectate opţiunile SCRATCH_LIB şi All şi se apasă butonul Next >.



6. În dialogul ***Choose View Styles*** se bifează opțiunile din figura de mai jos, apoi în dialogul următor – ***Confirm HDL Import*** - se bifează ***Overwrite*** din dialogul ***Options*** > ***General***.



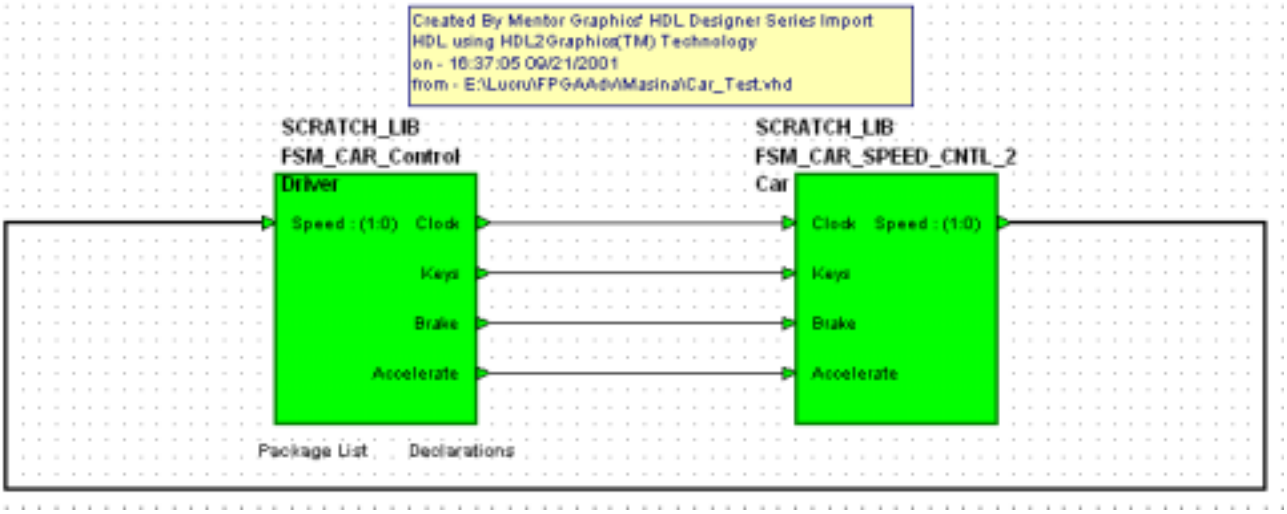
Se apasă apoi butonul  și începe conversia, rezultatele fiind afișate în fereastra ***Log Window***.



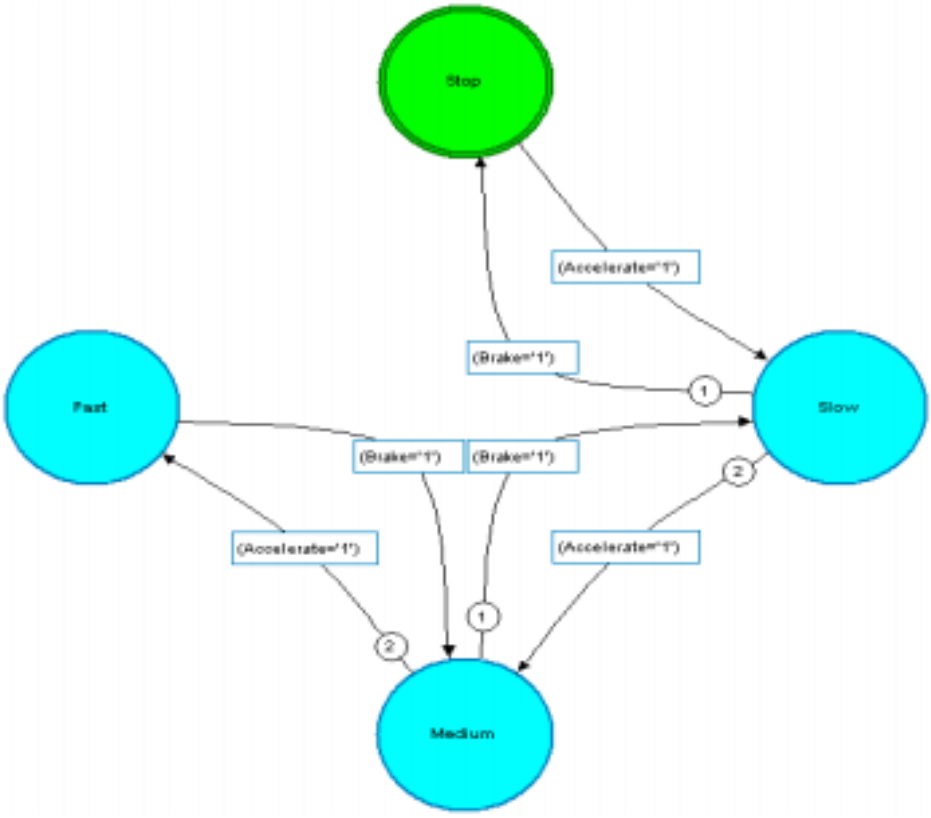
3. Examinarea proiectului

Se pot vizualiza toate componentele designului selectând SCRATCH_LIB şi alegând opţiunea Expand All cu ajutorul unui click pe butonul din dreapta al mouse-ului.

1. Se examinează modulul de test facând dublu click pe ***CAR_Test_Bench***. Se pot utiliza butoanele  pentru ***ZOOM***. De asemenea se poate vizualiza conţinutul blocurilor prin dublu click pe acestea. După examinare închideţi toate ferestrele cu excepţia ***Log Window*** şi ***Design Browser*** pentru a trece la pasul următor.



2. Cu un dublu click pe ***FSM_CAR_SPEED_CNTL_2*** se poate deschide diagrama de stări pe care a generat-o automat programul. Aceasta se poate analiza utilizând butoanele de ***ZOOM***.



4. Simularea

FPGAdv utilizează ca instrument pentru simulare ***ModelSim***-ul. În momentul lansării ***ModelSim***-ului din ***FPGAdv***, programul generează nişte surse ***VHDL*** pe care le simulează, pe baza celor originale introduse de utilizator (***Car_FSM.vhd*** şi ***Car_Test.vhd***). Nu întotdeauna însă programul reuşeşte să se descurce şi să interpreteze în mod corect ceea ce am vrut noi să spunem, aşa încât, uneori, după cum se va vedea în continuare, mai trebuie ajutat.

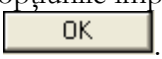
1. Se selectează ***CAR_Test_Bench*** din ***Design Browser*** şi apoi se apasă butonul  . În acest moment fereastra ***Log Window*** devine foarte importantă, deoarece putem urmări în aceasta modul în care unităţile design-ului sunt generate, încărcate şi compilate.

În cazul de faţă remarcăm că apar ***erori*** şi anume programul nu recunoaşte un anumit tip de date pe care el însuşi încercase anterior să-l creeze. Atunci se citeşte în ***Log Window*** în linia anterioară liniei ce indică eroarea (care spre exemplu ar putea fi: -- ***Reading file '...\FPGAdv50\Hds\examples\hds_scratch\scratch_lib\hdl\fsm_car_speed_cntl_2_rtl.vhd'***:), calea spre sursa generată în care apar erori şi se deschide acea sursă cu un editor de texte (spre exemplu ***NotePad***-ul) şi se comentează liniile următoare:

```
   -- Architecture Declarations
--TYPE STATE_TYPE IS (
--     Fast,
--     Medium,
--     Slow,
--     Stop
--);
```

deoarece acest tip este deja definit iar prin încercarea automată de redefinire apar erori.

După aceasta se salvează fişierul respectiv (***fsm_car_speed_cntl_2_rtl.vhd***) prin suprascriere.

2. În acest moment se reia pasul **1** şi se remarcă urmărind ***Log Window*** că nu mai apar erori. Se acceptă opţiunile implicite din fereastra de pornire a ***ModelSim***-ului şi se trece mai departe apăsându-se butonul 

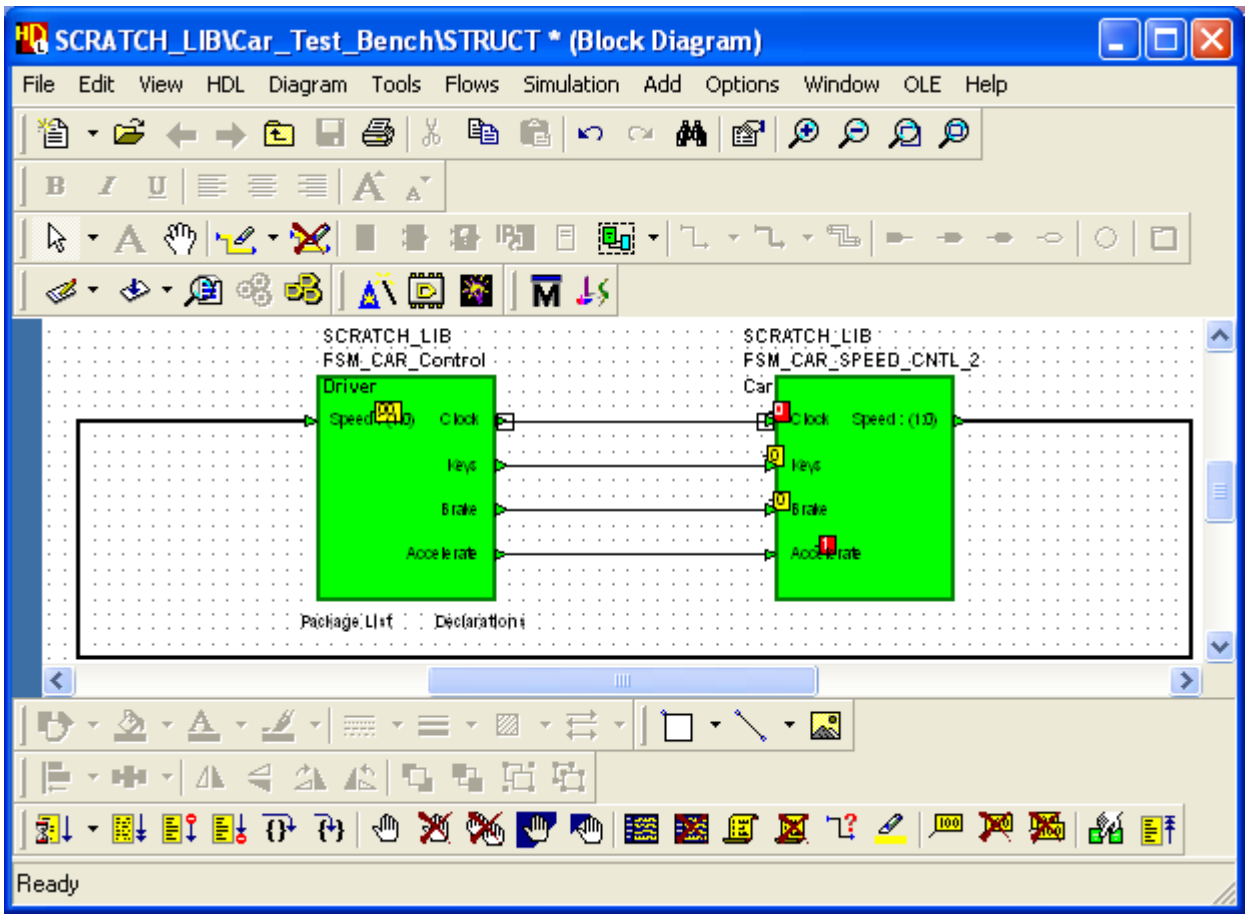
. Cum generarea şi compilarea s-au încheiat cu succes, simulatorul ***ModelSim*** este invocat şi designul compilat este încărcat.

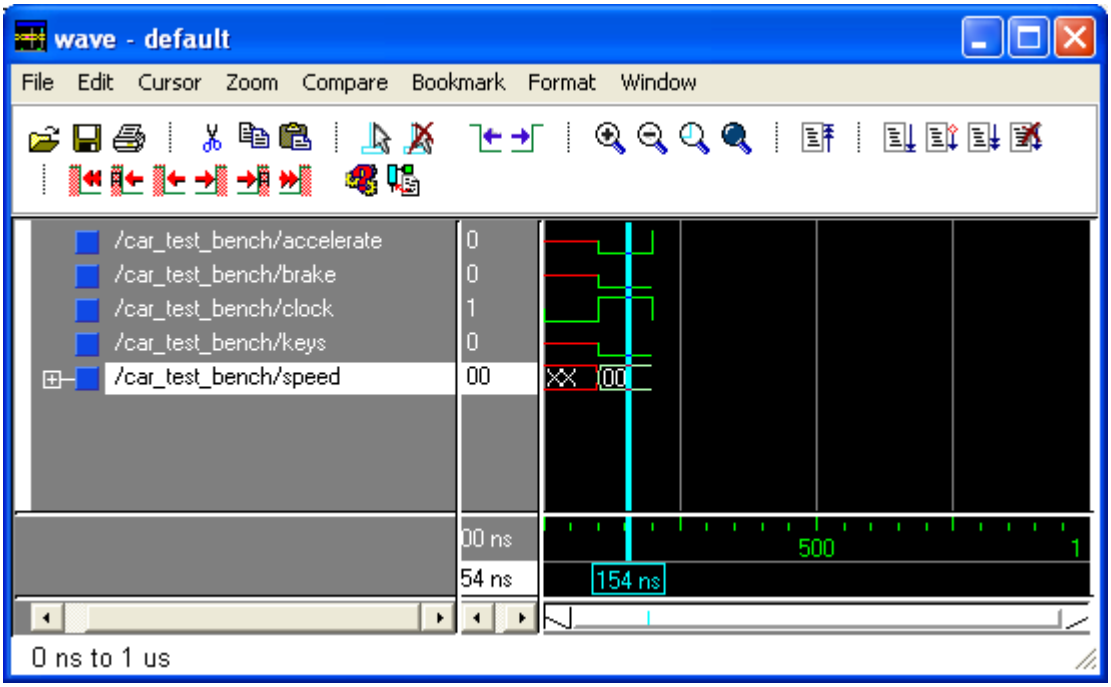
3. Se lasă deschis ***ModelSim***-ul şi se revine în ***Design Browser***, de unde se deschide diagrama bloc de test prin dublu click pe ***CAR_Test_Bench***. Se utilizează butoanele de ***ZOOM*** până când diagrama devine vizibilă, iar fereastra sa nu ocupă mai mult de un sfert de ecran. Se deplasează fereastra astfel încât să ocupe cu aproximaţie sfertul din dreapta sus.

4. În fereastra cu diagrama block, făcând uz de tasta ***Ctrl*** şi *butonul din stânga al mouse-ului* (click pe firele dintre blocuri), se selectează semnalele: ***Clock, Keys, Brake, Accelerate şi Speed***. Pentru a se monitoriza valorile acestor semnale, cu ele selectate, se apasă butonul  din bara meniuri din josul ferestrei. Apoi, pentru a se introduce automat aceste semnale în fereastra cu forme de undă (***wave***) a ***ModelSim***-ului, se apasă butonul  din aceeaşi bară de meniuri.

Se are grijă ca în fereastra ***wave*** pentru semnalul ***Speed*** să fie selectată opţiunea ***Binary*** din meniul ***Radix*** ce se deschide printr-un click dreapta pe semnal.

5. Redeschideţi diagrama de stări a FSM printr-un dublu click pe ***FSM_CAR_SPEED_CNTL_2***, în ***Design Browser***. Apoi activaţi animarea diagramei de stări, apăsând butonul  din fereastra cu diagrama de stări. Mutaţi această fereastră în colţul din stânga jos a ecranului astfel încât pe ecran să rămână vizibile următoarele ferestre: fereastra de control ***ModelSim***, fereastra cu formele de undă – ***Wave*** -, fereastra cu diagrama bloc a ***test bench***-ului şi fereastra cu ***diagrama de stări***.


SCRATCH_LIB\Car_Test_Bench\STRUCT * (Block Diagram)
File Edit View HDL Diagram Tools Flows Simulation Add Options Window OLE Help
SCRATCH_LIB
FSM_CAR_Control
Driver
Clock
Keys
Brake
Accelerate
Package List
Declarations
SCRATCH_LIB
FSM_CAR_SPEED_CNTL_2
Car
Clock
Speed : (10)
Keys
Brake
Accelerate
Ready


wave - default
File Edit Cursor Zoom Compare Bookmark Format Window
/car_test_bench/accelerate
/car_test_bench/brake
/car_test_bench/clock
/car_test_bench/keys
/car_test_bench/speed
0
0
1
0
00
00 ns
54 ns
500
1
154 ns
0 ns to 1 us

Utilizaţi butoanele de ***ZOOM*** în fereastra cu diagramele de stări pentru vizionarea optimă a tuturor celor patru stări. La pornirea simulării, în trei dintre ferestre se vor putea urmări modificările pe care le suferă semnalele monitorizate în timpul simulării. Pe diagrama de stări, starea curentă va fi colorată cu roşu, iar starea anterioară va fi galbenă.

6. În acest moment puteţi începe simularea: în fereastra cu diagrama de stări faceţi succesiv click pe butonul  din bara de meniuri din josul ferestrei, echivalentul cu rularea cu câte 100ns de fiecare dată. Observaţi modificările ce se petrec asupra semnalelor în cele trei ferestre (fereastra cu formele de undă – ***Wave*** -, fereastra cu diagrama bloc a ***test bench***-ului şi fereastra cu ***diagrama de stări***). Când ajungeţi la 3000ns, opriţi-vă şi trageţi nişte concluzii referitoare la răspunsurile date la întrebările 1-4 de la începutul studiului FPGA Advantage.

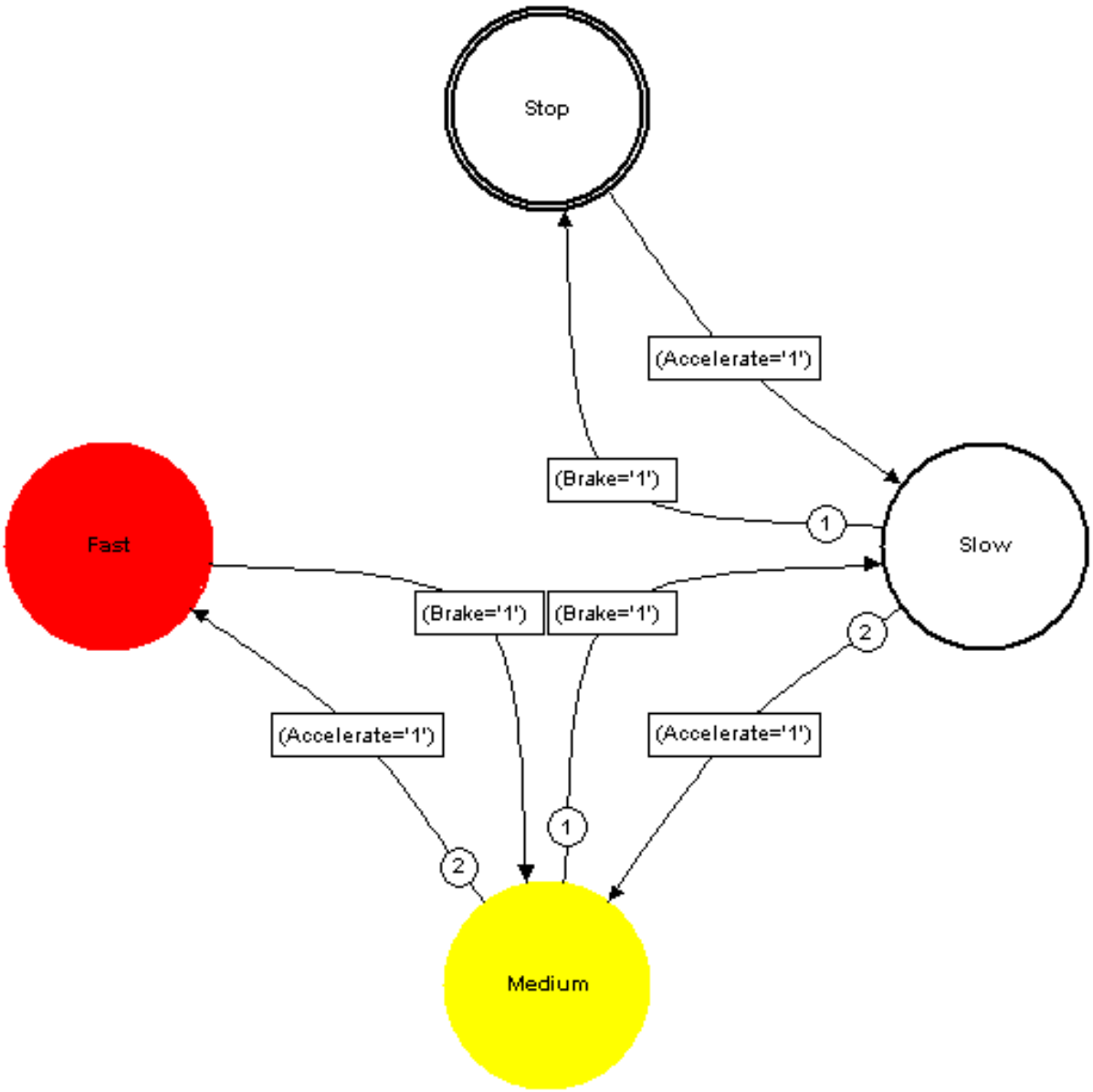
În primul rând, remarcaţi că deşi urmărind sursa VHDL, la prima vedere am fi tentaţi să credem că semnalul ***Speed*** se modifică pe frontul căzător al ***Clock***-ului, în realitate, numai ***Speed_s*** se modifică pe frontul căzător, însă ***Speed*** nu apucă să se actualizeze cu proaspăta valoare a lui ***Speed_s***, ci se actualizează cu valoarea veche a lui ***Speed_s***, adică cea pe care o avea înainte de intrarea în proces (mai exact cu cea de la evenimentul anterior ce a determinat intrarea în proces, adică de pe frontul crescător anterior al **Clock**-ului).

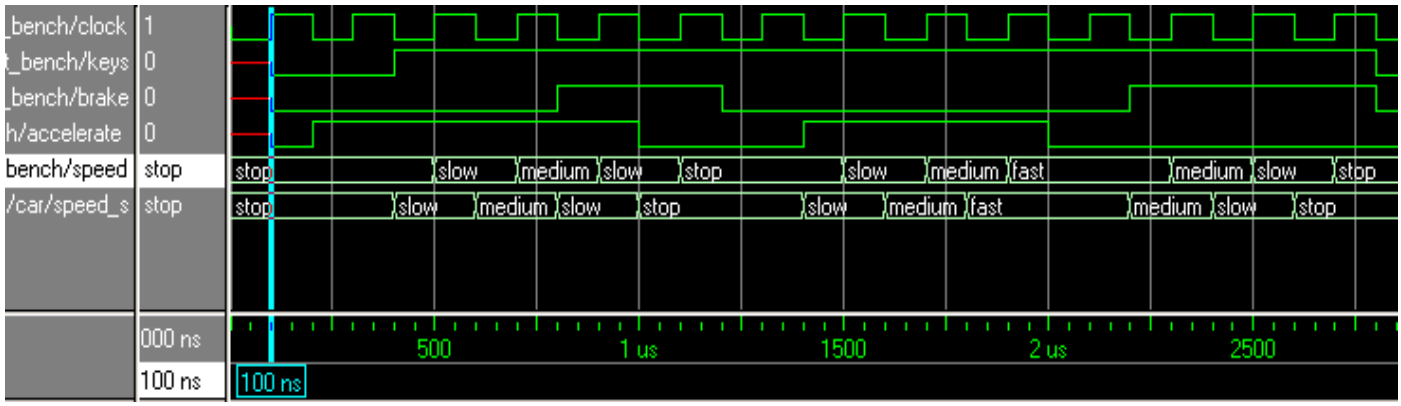
Pentru a răspunde la întrebarea a doua, există două căi: ***Speed*** să fie declarat ca variabilă locală a procesului, ceea ce nu convine, fiindcă ***Speed*** este necesar în afara procesului, sau cea de-a doua soluţie, care este de altfel şi viabilă, constă în scoaterea în afara procesului a atribuirii ***Speed <= Speed_s;***. De altfel, la sfârşitul laboratorului se poate încerca simularea (doar cu ***ModelSim***-ul) a acestei variante, care este descrisă în listingul de mai jos. De menţionat că varianta de mai jos conţine şi testarea maşinii în aceeaşi arhitectură

Referitor la sincronicitate, este clar că semnalul ***Keys***, ce joacă rolul de ***reset***, este prioritar faţă de ***Clock***, deci structura este asincronă.

Cât priveşte succesiunea stărilor FSM, aceasta este acum evidentă datorită simulării: Necunoscută, Stop, Stop, Slow, Medium, Slow, Stop, Stop, Slow, Medium, Fast, Fast, Medium, Slow, Stop, Stop, Stop, …

7. Se vor închide toate ferestrele legate de ***FPGA Advantage*** şi ***ModelSim***, cu excepţia ***Design Browser*** şi ***Log Window***, pentru a putea trece la etapa următoare: sinteza.


Stop
(Accelerate='1')
(Brake='1')
Fast
Slow
1
(Brake='1')
(Brake='1')
2
(Accelerate='1')
(Accelerate='1')
1
2
Medium


_bench/clock
_bench/keys
_bench/brake
h/accelerate
bench/speed
/car/speed_s
1
0
0
0
stop
stop
stop slow medium slow stop slow medium fast medium slow stop
stop slow medium slow stop slow medium fast medium slow stop
000 ns
100 ns
500
1 us
1500
2 us
2500
100 ns

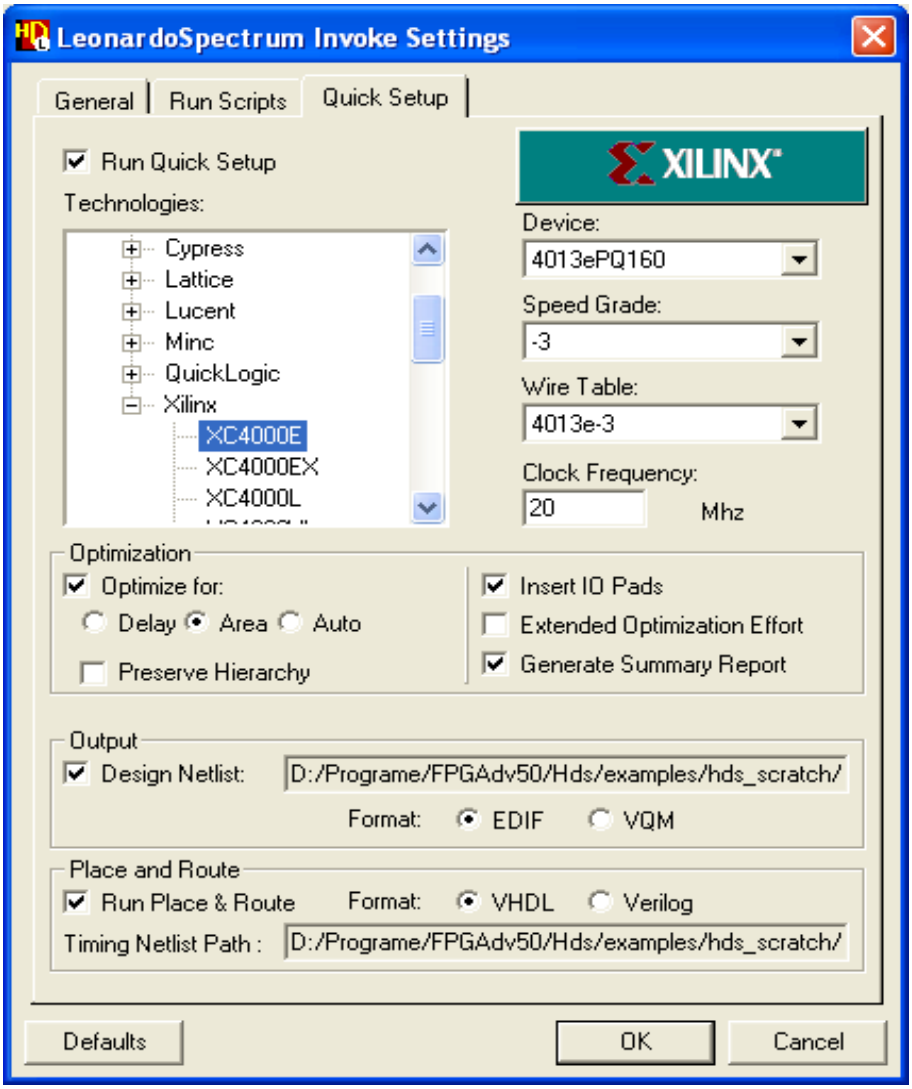
```
-- Car_FSM_Tot.vhd
-- Atentie la metoda de codare a starilor fara package cu tip de date
-- Stop   = 11
-- Slow   = 10
-- Medium = 01
-- Fast   = 00
-- Obs: Nu se mai poate face sinteza cu continutul acestui fisier
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

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;

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

entity FSM_CAR_SPEED_CNTL_2_Tot is
	port (Speed:out unsigned (1 downto 0));
end;-- entity  FSM_CAR_SPEED_CNTL_2_Tot;

architecture RTL of FSM_CAR_SPEED_CNTL_2_Tot is
	constant Stop:      unsigned(1 downto 0):="11";
	constant Slow:      unsigned(1 downto 0):="10";
	constant Medium:    unsigned(1 downto 0):="01";
	constant Fast:      unsigned(1 downto 0):="00";
	signal   NextSpeed: unsigned(1 downto 0);
	signal   Speed_s:   unsigned(1 downto 0);
	signal Clock, Keys, Brake, Accelerate: std_logic;
	constant clk_prd: time := 200 ns;

component Clock_Gen
	port (Clock: out std_logic);
end component;

begin

	Sursa: Clock_Gen
		port map (Clock => Clock);

	process
	begin

		Keys <= '0' after clk_prd/2,'1' after 2*clk_prd, '0' after 14*clk_prd;
		Brake <= '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 <= '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;
```

```
	FSM2_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 FSM2_COMB;

	--Atentie la scoaterea atribuirii in afara procesului:
	FSM2_SEQ: process (Clock,Keys)
	begin
		if (Keys='0') then
			Speed_s <= Stop;
		elsif falling_edge(Clock) then
			Speed_s <= NextSpeed;
		end if;
	end process FSM2_SEQ;
		Speed <= Speed_s;

end; -- architecture RTL;
```

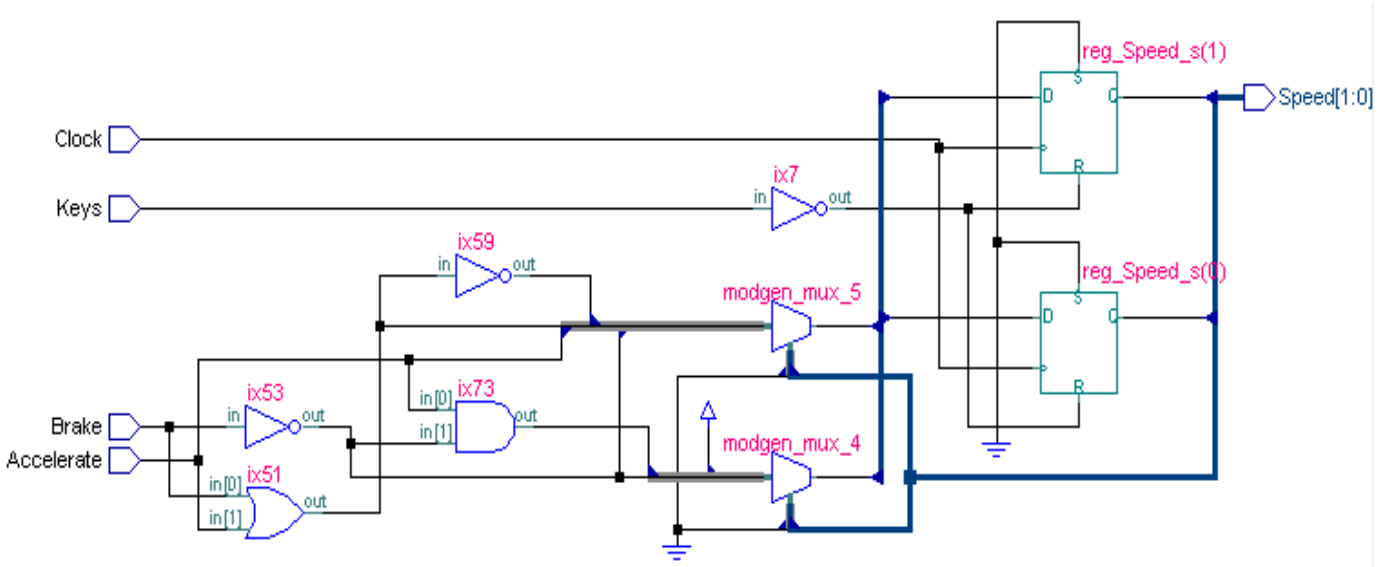
5. Sinteza

FPGA Advantage utilizează pentru sinteză ***LeonardoSpectrum***. Cu ajutorul acestuia se încearcă implementarea fizică a circuitului descris la nivel ***RTL*** (în ***VHDL*** în cazul nostru), utilizând componente (porţi, bistabile, latch-uri, etc.) predefinite, specifice unei tehnologii. În acest caz se utilizează tehnologia ***FPGA***, cu librăria de componente ***XC4000E*** de la firma ***Xilinx***. S-ar putea folosi spre exemplu şi tehnologia ***ASIC***, cu librăria de componente ***CX2001_nom*** a firmei ***Chip Express***.

1. Se selectează ***FSM_CAR_SPEED_CNTL_2*** şi se apasă butonul  lansându-se astfel ***LeonardoSpectrum*** pentru ***FSM_CAR_SPEED_CNTL_2***. În continuare se aleg parametrii din figura de mai jos şi se lasă programul să-şi urmeze cursul automat.



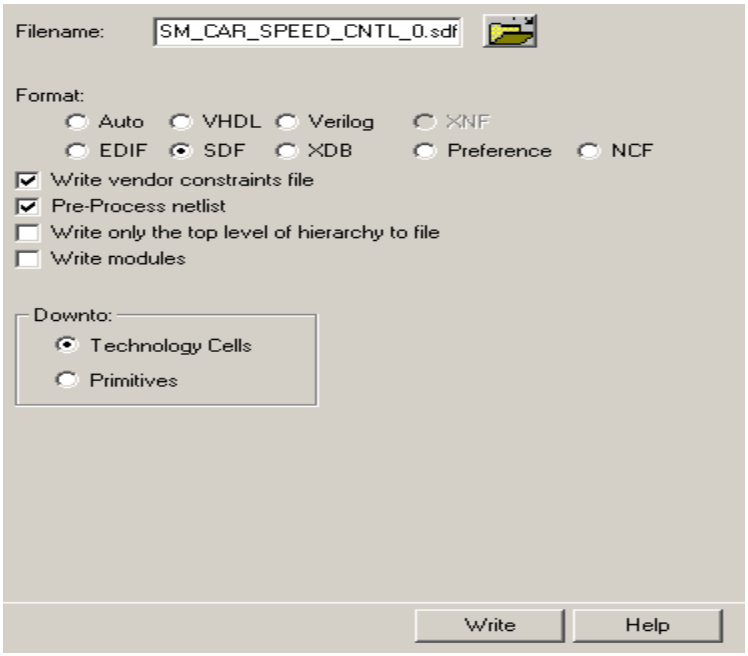
2. Următorul pas este vizualizarea schemei rezultate în urma sintezei din sursele ***VHDL***, prin apăsarea butonului  , rezultatul fiind prezentat în figura de mai jos.



3. Pentru vizualizarea optimă se pot folosi opţiunile de ***ZOOM*** din meniul ***Schematic Viewer***. De asemenea, pentru vizualizarea codului ***VHDL*** ce a dus la generarea unei componente, se poate utiliza comanda ***Trace to HDL Source*** din meniul ce se deschide printr-un click dreapta al mouse-ului pe una din componente.

Încercaţi să deduceţi care porţiune din codul ***VHDL*** sursă a determinat apariţia prin sinteză a fiecărei componente în parte.

Trebuie menţionat că schema obţinută prin sinteză are un corespondent în cod ***HDL***, precum şi în format ***SDF*** (***Standard Delay Format***) sau în alt format, care la simularea cu acelaşi ***test bench*** ca şi sursele originale ***VHDL*** trebuie să dea aceleaşi rezultate. Fişierele cu acest cod corespondent se poate obţine ca în figura de mai jos, folosind butonul ***Write***. Folosind fişierele SDF, după ce ne-am asigurat că ele conţin ceea ce am dorit noi să facem când am scris codul RTL original, se parcurg şi restul de etape până la realizarea şi verificarea layout-ului pentru chip – componenta fizică.



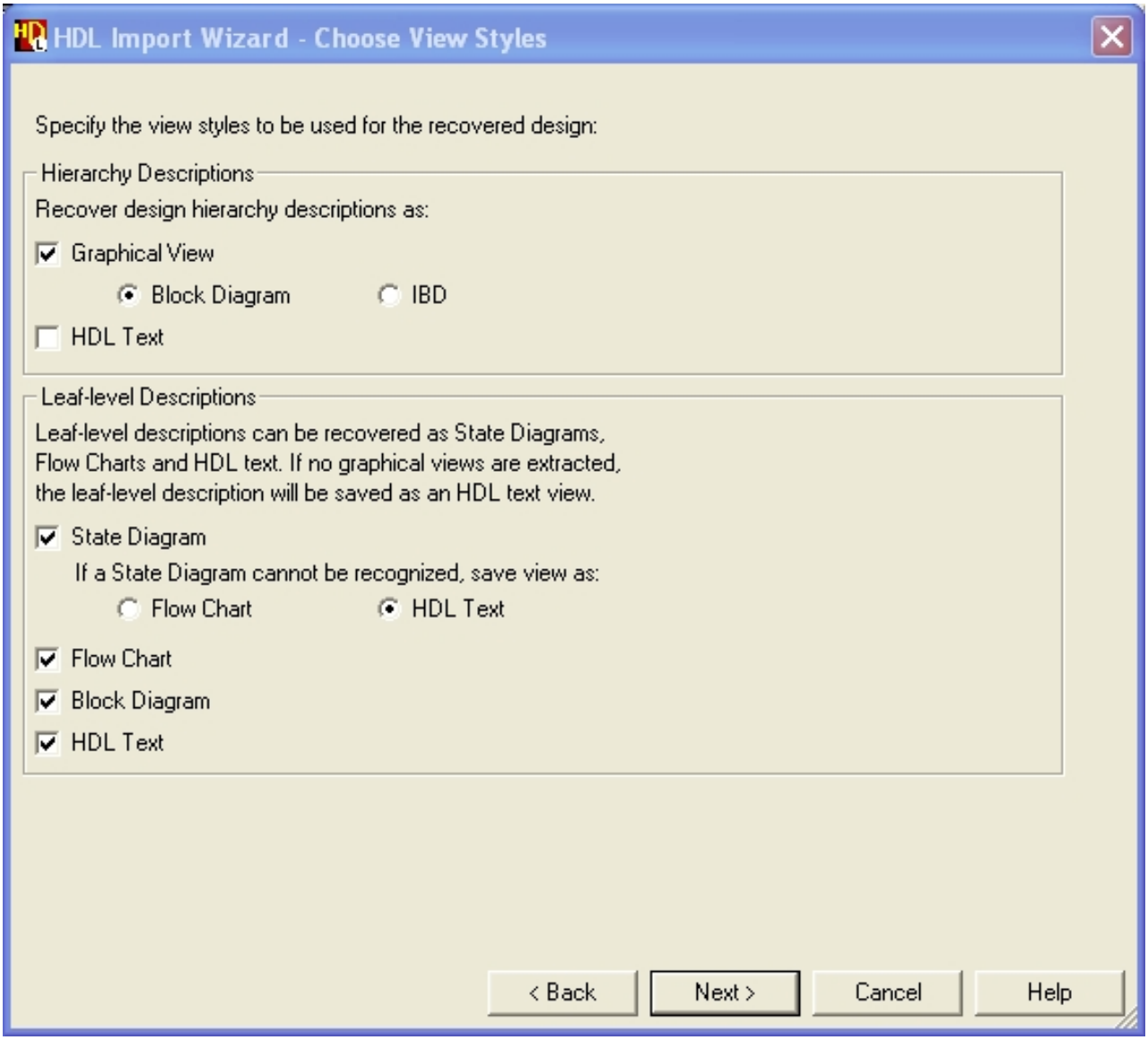
6. Lucru individual

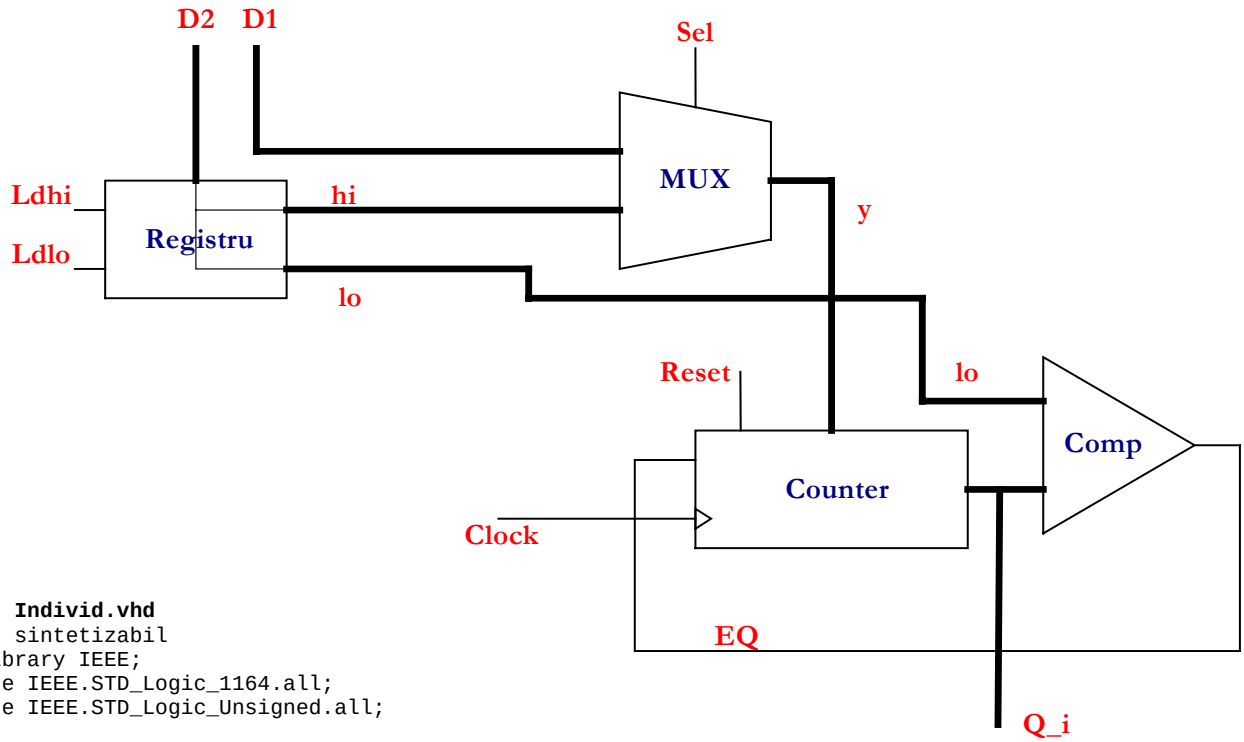
În cele ce urmează se prezintă o ***schemă bloc*** şi implementarea sa ***VHDL***. Se cere:

1. Implementarea în ***VHDL*** a unui ***test bench*** şi simularea cu ***ModelSim***.

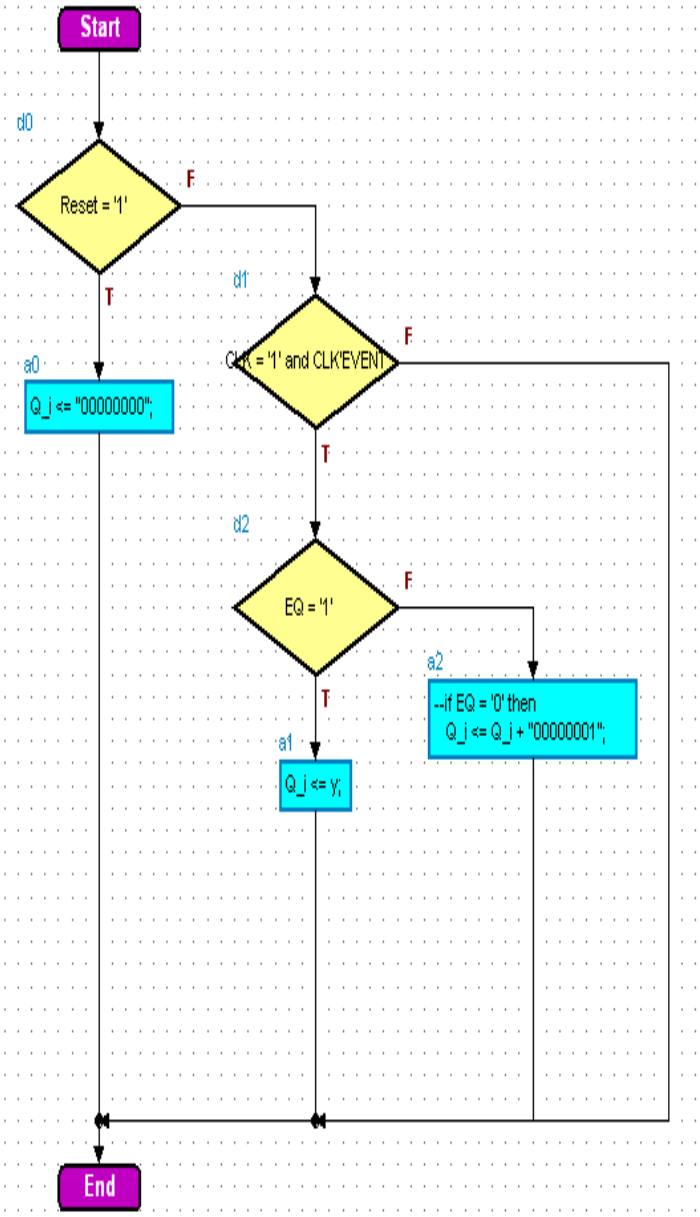
2. Deschiderea cu ***FPGA Advantage*** a arhitecturii ***PREP2*** şi vizualizarea acesteia prin dublu click pe ***PREP2***. **Observaţi**e: la deschiderea ***FPGAdv*** şi importarea unui design, se vor utiliza opţiunile din figura de mai jos. Menţionăm că în acest caz programul nu mai poate genera o ***diagramă de stări***, dar poate genera o ***schemă logică***.

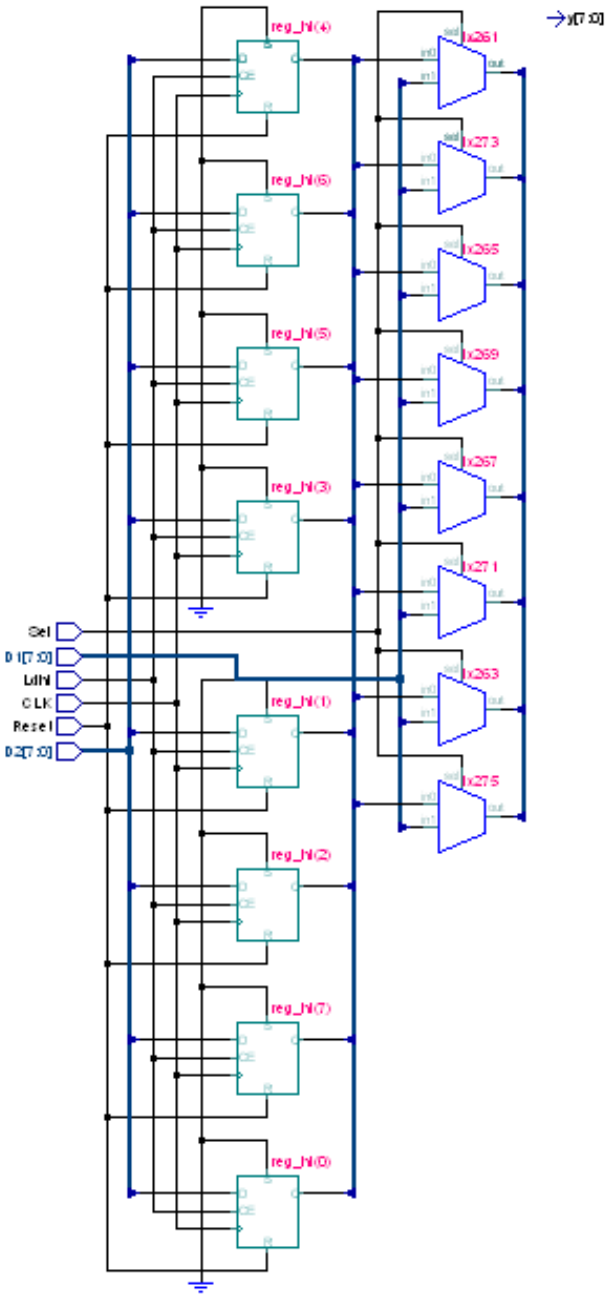
3. Sinteza cu ajutorul ***LeonardoSpectrum*** a arhitecturii ***PREP2*** şi studiul schemei obţinute după sinteză, prin compararea cu cea bloc iniţială, prezentată în cele ce urmează.



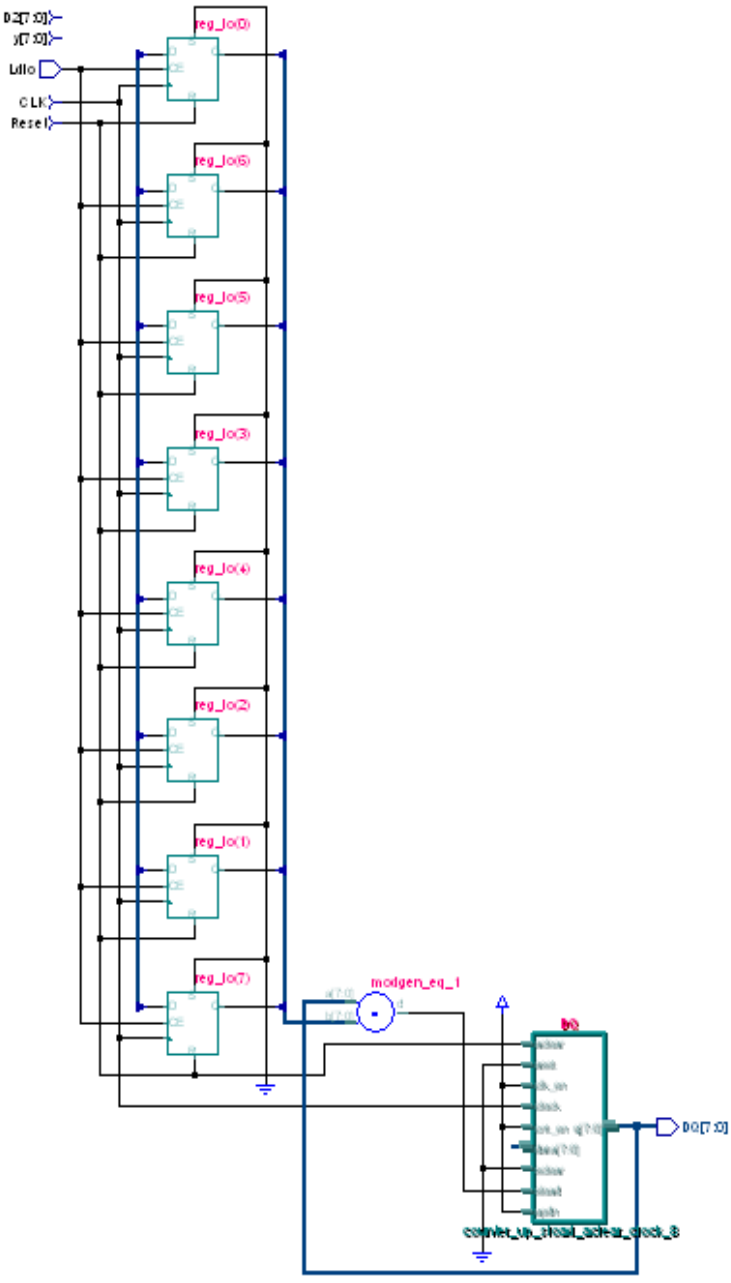



```
-- Individ.vhd
-- sintetizabil
library IEEE;
use IEEE.STD_Logic_1164.all;
use IEEE.STD_Logic_Unsigned.all;

entity PREP2 is
	port (CLK, Reset, Sel, Ldlo, Ldhi: BIT;
		D1,D2: STD_LOGIC_VECTOR(7 downto 0);
		DQ: out STD_LOGIC_VECTOR(7 downto 0) );
end;

architecture Behave_1 of PREP2 is
	signal EQ: BIT;
	signal y,lo,hi,Q_i: STD_LOGIC_VECTOR(7 downto 0);

begin
	outputDriver: DQ <= Q_i;
	mux: with Sel select y <= hi when '0', D1 when '1';
	comparator: EQ <= '1' when Q_i = lo else '0';

	registru: process(Reset,CLK)
	begin
			if Reset = '1' then
				hi <= "00000000";
				lo <= "00000000";
			elsif CLK = '1' and CLK'EVENT then
				if Ldhi = '1' then
					hi <= D2;
				end if;
				if Ldlo = '1' then
					lo <= d2;
				end if;
			end if;
	end process registru;

	counter: process(Reset, CLK)
	begin
		if Reset = '1' then
			Q_i <= "00000000";
		elsif CLK = '1' and CLK'EVENT then
			if EQ = '1' then
				Q_i <= y;
			else --if EQ = '0' then
				Q_i <= Q_i + "00000001";
			end if;
		end if;
	end process counter;

end;
```

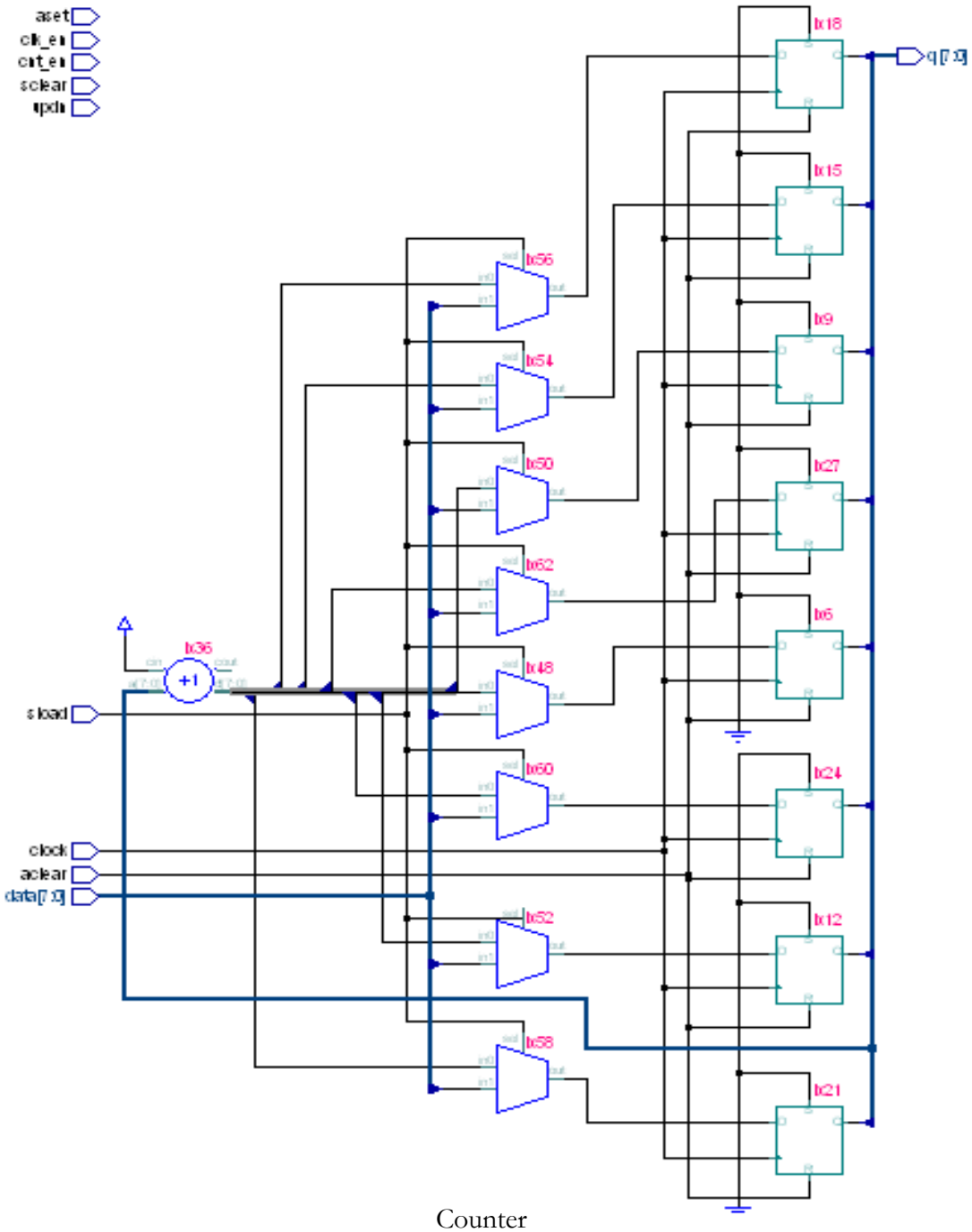

Start
d0
Reset = '1'
F
T
d1
CLK = '1' and CLK'EVENT
F
T
a0
Q_i <= "00000000";
d2
EQ = '1'
F
T
a2
--if EQ = '0' then
Q_i <= Q_i + "00000001";
a1
Q_i <= y;
End



Page 1 of schematic



Page 2 of schematic


aset
clk_en
cnt_en
sclear
sload
data[7..0]
clock
aclear
lx36
+1
lx18
lx15
lx9
lx27
lx6
lx24
lx12
lx21
lx56
lx54
lx50
lx52
lx48
lx60
lx62
lx58
q[7..0]

Counter