Configurarea structurilor hardware reprogramabile de tip FPGA cu XILINX ISE

I. SCOPUL LUCRĂRII

În această lucrare este prezentată metodologia relativă la implementarea programelor scrise prin limbajul VHDL, pe structura reprogramabilă FPGA de tip Artix®-7, folosind pachetul software de dezvoltare Vivado. Pe scurt, sunt abordate următoarele etape: introducerea codului VHDL, simularea şi vizualizarea rezultatelor obţinute, sinteza şi implementarea surselor, iar în final programarea structurii reconfigurabile.

II. INTRODUCERE TEORETICĂ

Mediul software de proiectare Vivado este utilizat la realizarea şi implementarea completă a unui proiect pentru structurile programabile de tip XILINX. Acest mediu de programare facilitează parcurgerea pe următoarele etape in vederea realizării unui proiect:

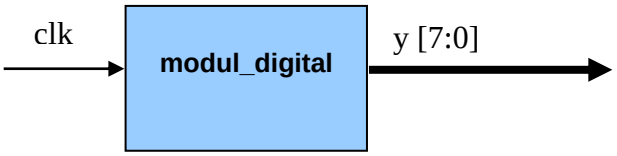
- *Descriere proiect.* Programatorul are posibilitatea să descrie proiectul prin introducerea de coduri sursă HDL (Hardware Description Language) pentru limbajele VHDL, Verilog, Abel sau utilizând schematice şi diagrame cu stări finite;
- *Sinteza.* În cadrul acestei etape fişierele de tip VHDL, Verilog sau schematice sunt transformate în fişiere de tip *netlist* care sunt acceptate ca fişiere de intrare la etapa de implementare;
- *Implementarea.* După sinteză, la implementare, proiectul este adaptat şi transformat din forma logică digitală în forma tehnologică implementabilă pe structura reconfigurabilă aleasă;
- *Verificarea.* Poate fi realizată în toate etapele de implementare ale proiectului. Utilizarea componentelor software de simulare conduce la verificarea completă a funcţionalităţii proiectului sau a unor porţiuni de proiect. De asemenea, pot fi realizate şi verificări directe pe circuit, după programarea acestuia;
- *Configurarea.* După generarea fişierelor de programare (bitstream file) proiectantul are posibilitatea programării circuitului reconfigurabil. În timpul procesului de configurare sunt programate interconexiunile structurii FPGA alese.

În vederea exemplificării modului de implementare a unui modul digital pe o structură de tip FPGA, se prezintă următorul model.

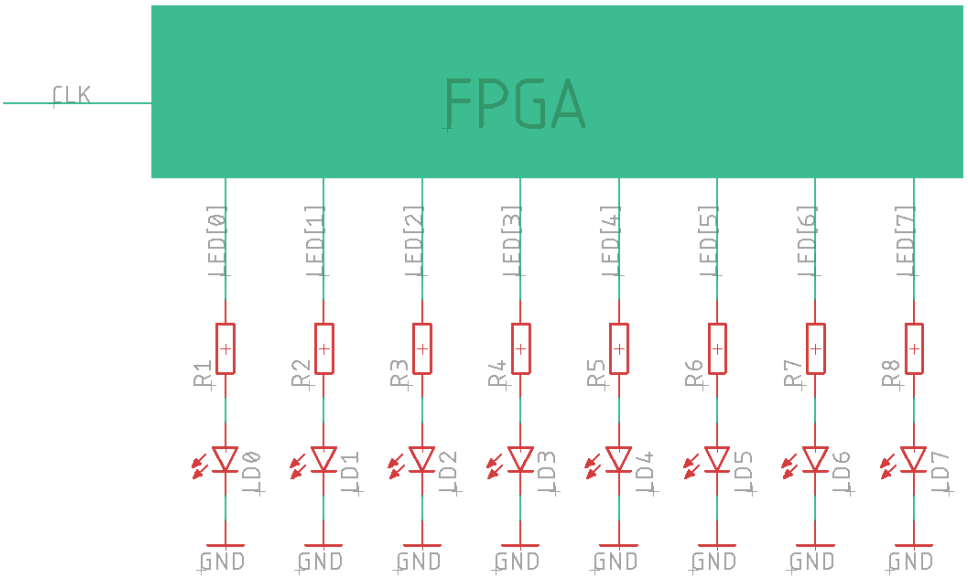
1. Enunţul problemei

Să se descrie etapele de implementare ale unui modul digital care realizează un joc de lumini pe 8 LED-uri, în limbajul VHDL, pe sistemul reconfigurabil BASYS 3 cu circuitul FPGA XC7A35T-1CPG236C.

Porturile de intrare/ieşire ale modulului digital sunt prezentate în figura următoare:



Verificarea modulului digital va fi realizată folosind schema electrică din figura de mai jos. Această schemă este doar o mică parte din schema electrică a sistemului reconfigurabil de dezvoltare XC7A35T-1CPG236C:



Notă: Se va consulta schema electrică completă a sistemului de dezvoltare şi se vor identifica pe aceasta elementele de circuit.

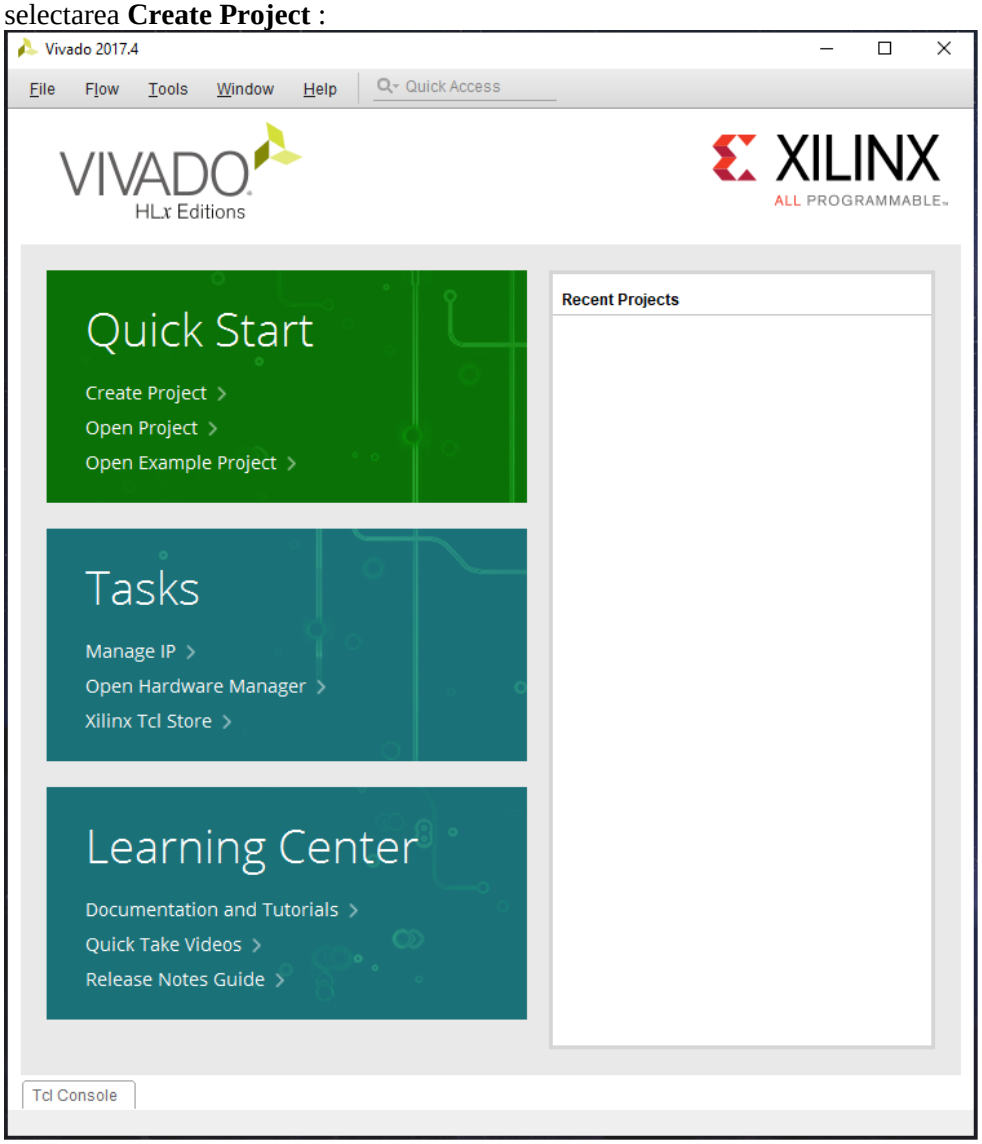
Pinii de interconectare ai structurii FPGA cu modulul digital descris în VHDL sunt atribuiți în fișierul **Basys-3-Master.xdc** .

1.1. Crearea proiectului

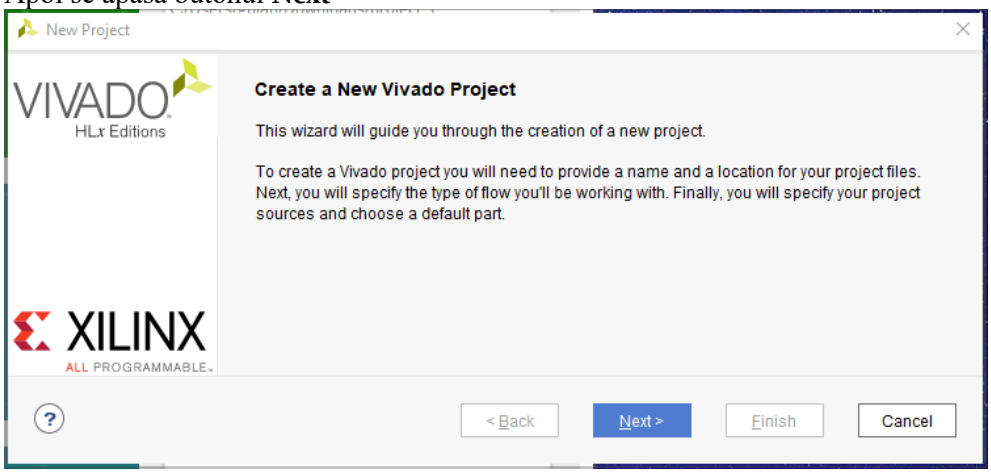
Se lansează programul Vivado din mediul Windows.

Primul pas în realizarea modulului digital constă în crearea proiectului.

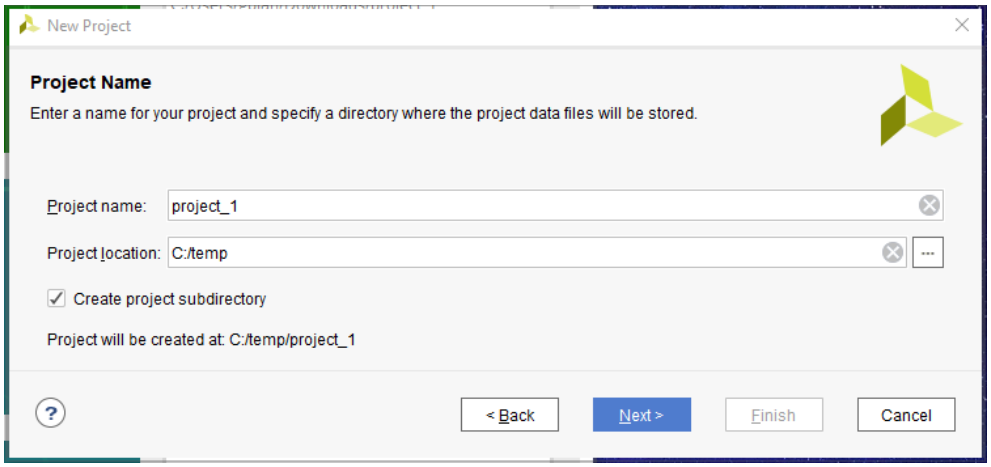
Crearea unui proiect se face plecând de la fereastra generală, prin selectarea **Create Project** :



Apoi se apasă butonul **Next**

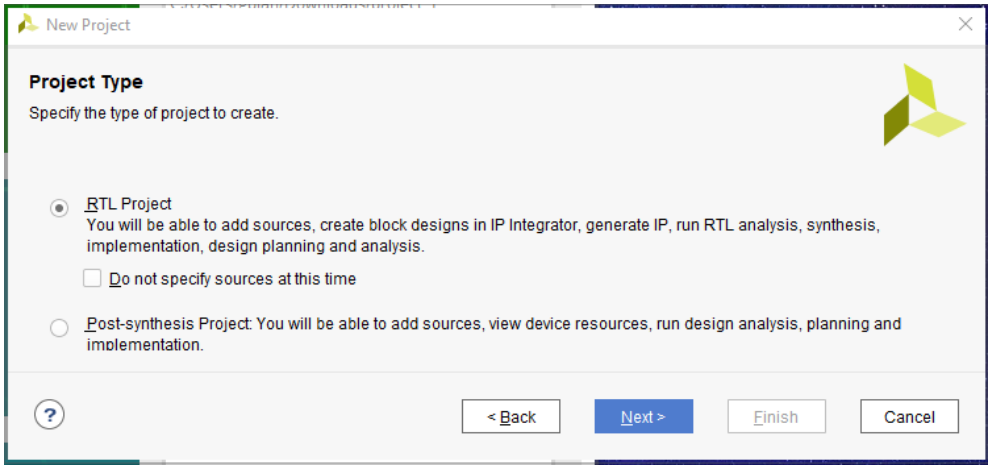


Se va creea un director **temp** pe unitatea C:\ sau D:\, se va copia fisierul **Basys-3-Master.xdc** se va salva proiectul intitulat **proiect_1** in acesta:

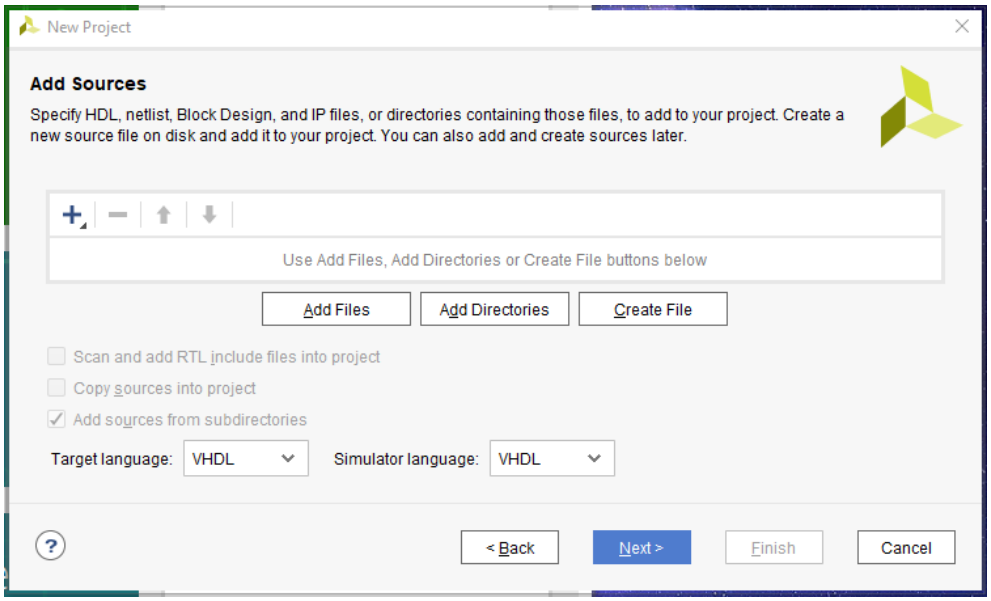


după care se apasă butonul **Next**.

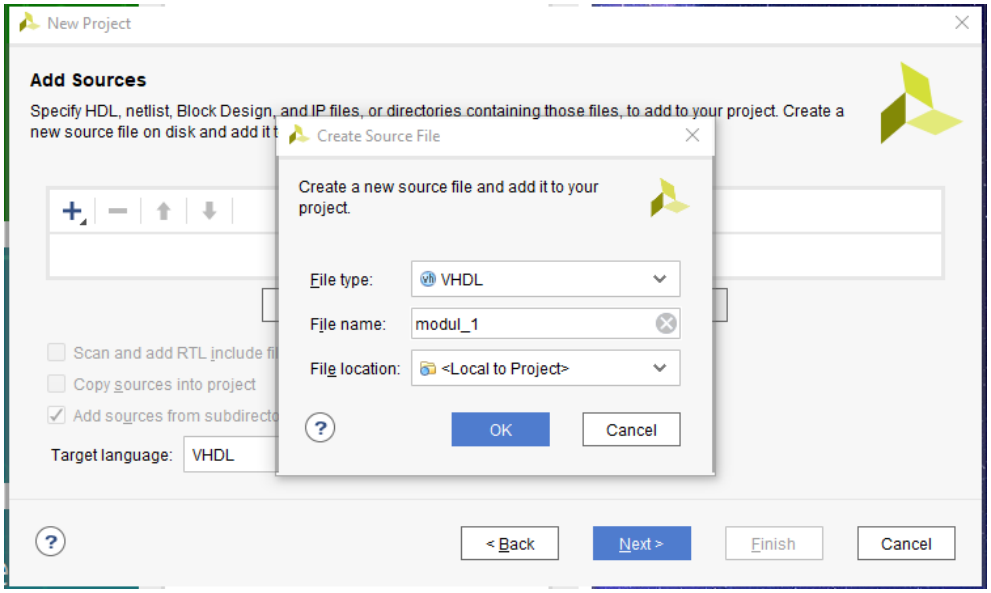
Se selectează **RTL Project** si se apasă butonul **Next**.



Se va alege **Target language:VHDL** si **Simulator language: VHDL**

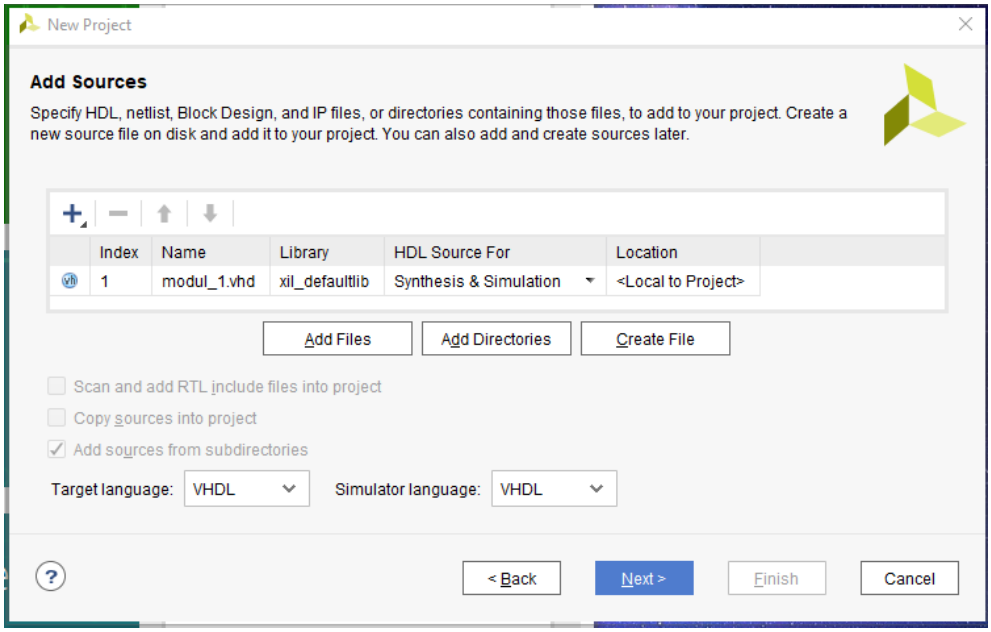


Se va creea un fisier VHDL numit **modul_1**

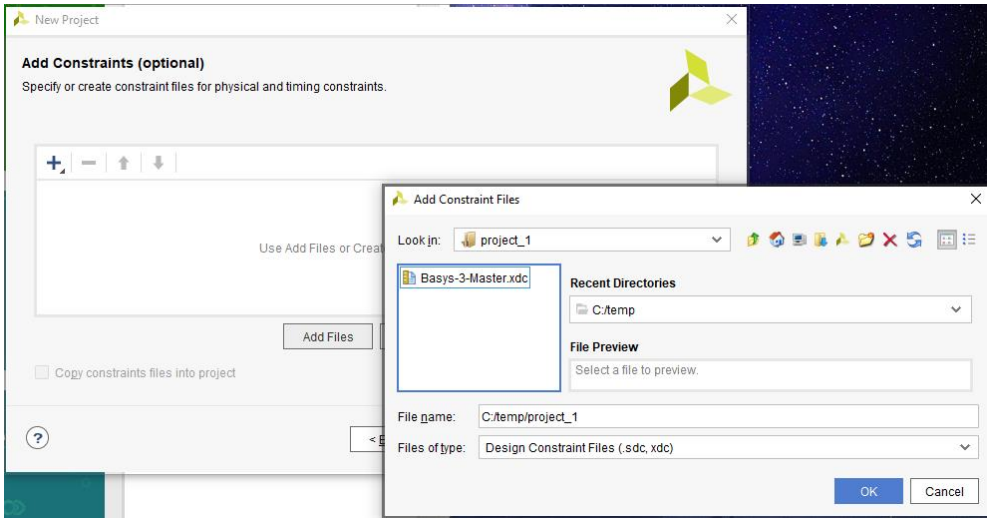


Dupa care se apasa butonul **OK** și apoi butonul **NEXT**.

Apoi se apasă butonul **Add file**



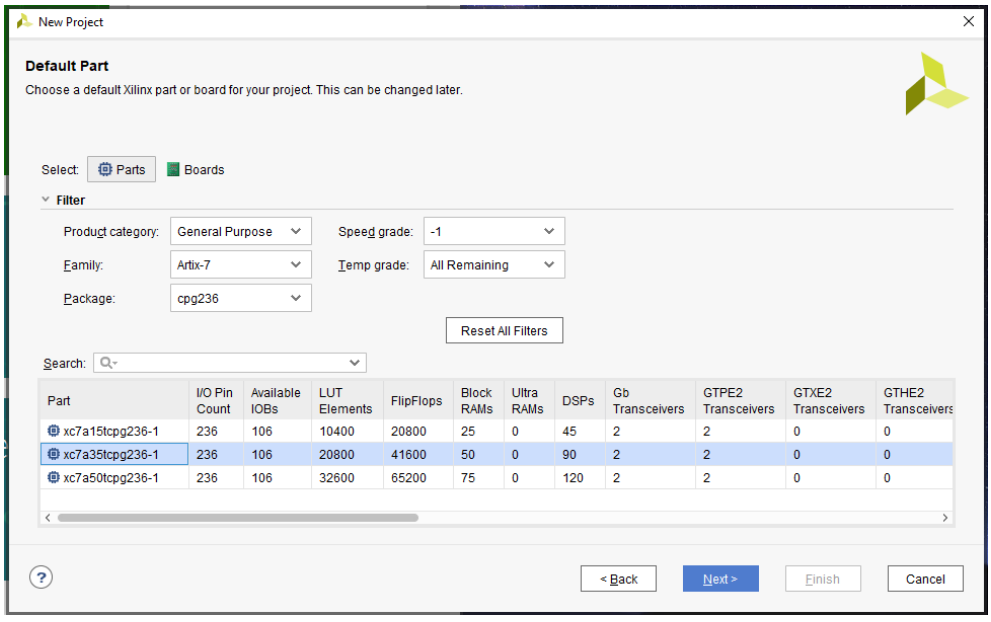
Se va copia fisierul de constrangeri a pinilor **Basys-3-Master.xdc** in directorul de lucru a proiectului (*C:\temp\project_1*) dupa care va fi selectat in fereastra **Add Constrains**.



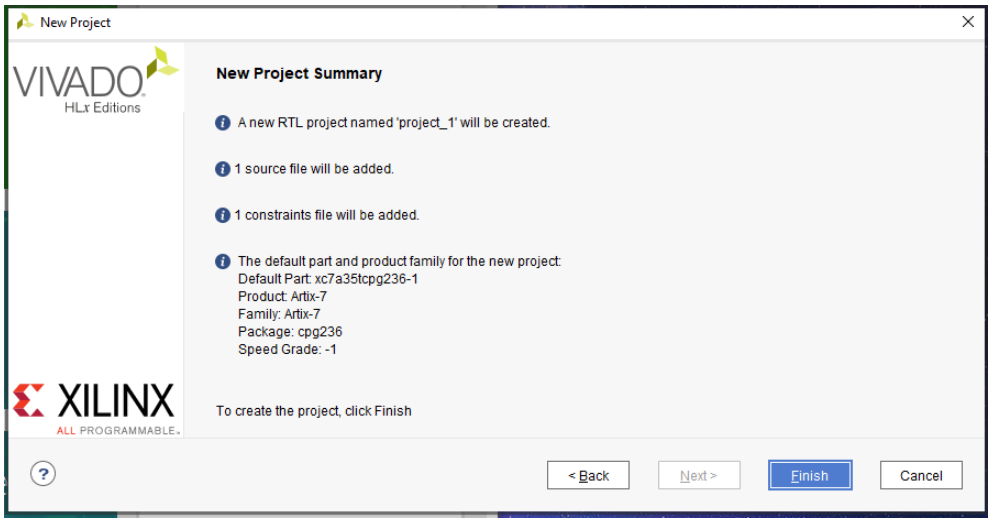
și se va selecta fisierul **Basys-3-Master.xdc** după care se apasă butonul **Next**.

Urmează fereastra în care se va selecta circuitul FPGA.

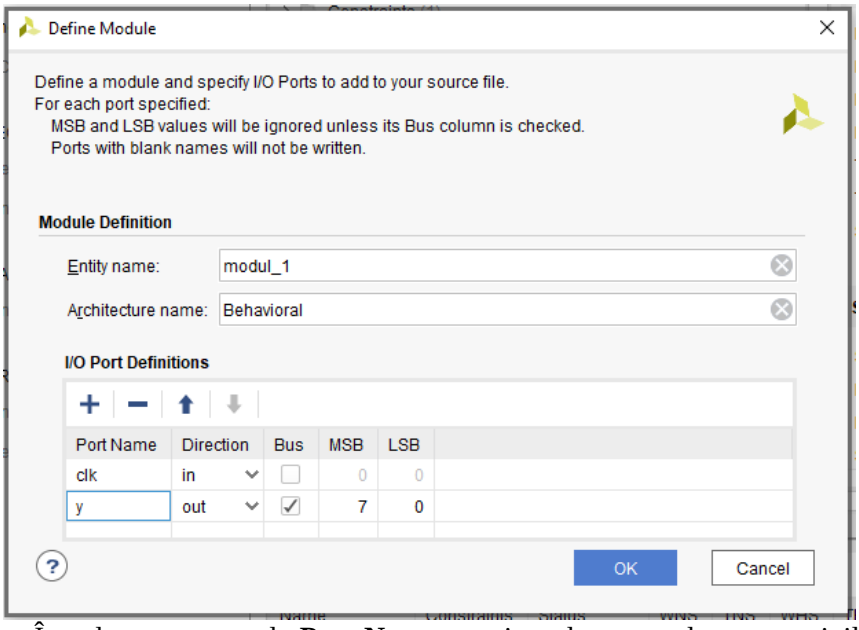
Circuitul FPGA utilizat este **xc7a35tcpg236-1**.



Și pentru validarea proiectului se apasă butonul **Finish**.



Se deschide automat fereastra in care pot fi introduse porturile de intrare/ieșire a molului digital dar se va închide prin apăsarea butonului **OK**.



În coloana cu numele **Port Name** sunt introduse numele tuturor pinilor cu direcţia specificată în coloana **Direction** (direcţia poate fi *input*, *output*, *inout*). În

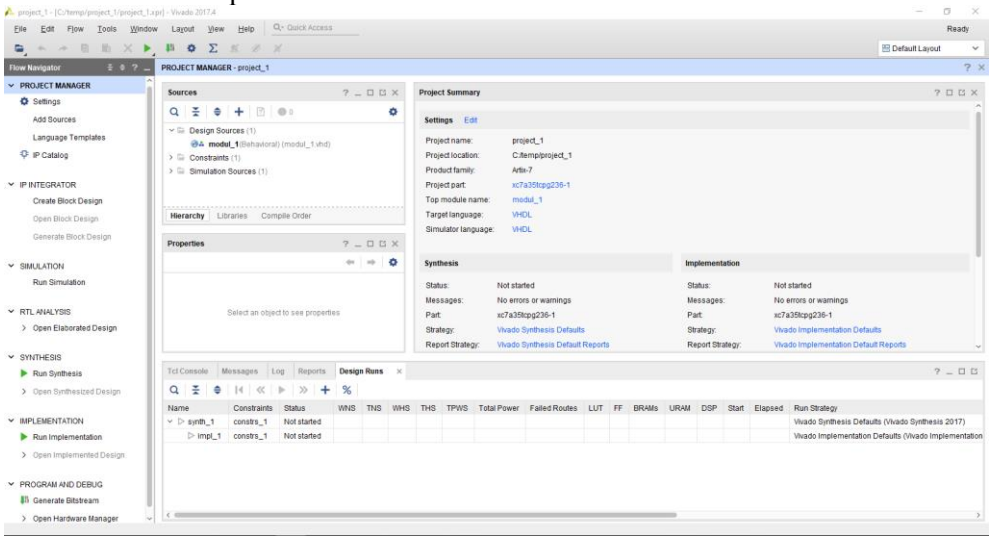
cazul în care porturile sunt vectori sau magistrale, pe coloanele MSB şi LSB se specifică dimensiunea acestora.

Porturile introduse sunt următoarele:

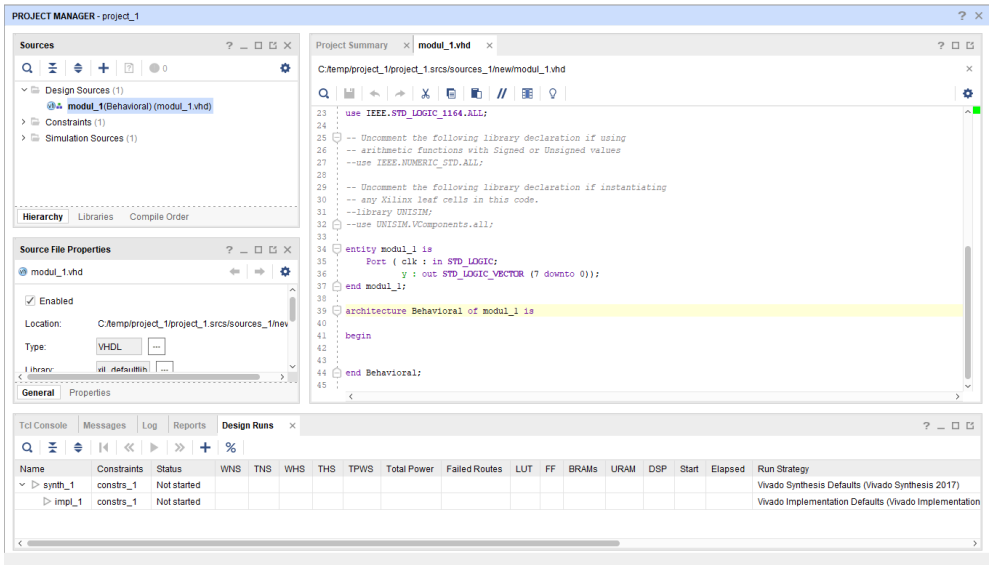
clk *input*

y *output* *7 .. 0*

După completarea tuturor porturilor modulului digital, se selectează butonul **OK** si va apărea următoarea fereastră:

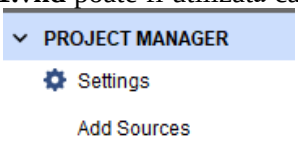


Se deschide fisierul **modul_1.vhdl** şi se observă ca este creată automat structura de bază a unui program VHDL.



1.2. Editarea fişierului VHDL

Fereastra în care este afişat fişierul sursă **modul_1.vhd** poate fi utilizată ca un editor în vederea completării sau modificării programului VHDL. Este recomandat ca programul să fie salvat periodic prin comanda **File → Save** din meniul principal. Programele VHDL pot fi editate în orice editor de text şi ataşate la proiect prin comanda **Add Source**.



Programul VHDL va fi completat cu liniile următoare de cod sursă:

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

entity modul_digital is
    Port ( clk : in std_logic;
           y : out std_logic_vector(7 downto 0));
end modul_digital;

architecture Behavioral of modul_digital is
            signal temporar: std_logic_vector(2 downto 0):= (others => '0');
begin
            process(clk)
            begin
             if (clk'event and clk = '1')           then
                        temporar <= temporar + '1';
             end if;
            end process;

            process (temporar)
            begin
                        case (temporar) is
                                    when "000" =>  y <= "10000001";
                                    when "001" =>  y <= "01000010";
                                    when "010" =>  y <= "00100100";
                                    when "011" =>  y <= "00011000";
                                    when "100" =>  y <= "00100100";
                                    when "101" =>  y <= "01000010";
                                    when "110" =>  y <= "10000001";
                                    when others => y <= (others => '1');
                        end case;
            end process;
end Behavioral;
```

Programul este format din doua blocuri. Primul bloc are funcţia de numărător pe 3 biţi, iar cel de-al doilea este un decodor binar într-o formă definită de proiectant.

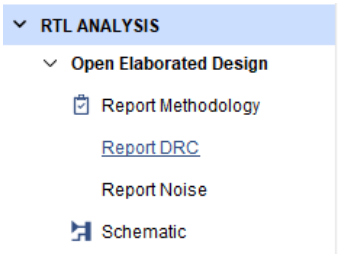
Legătura dintre blocuri este realizată prin doi regiştri declaraţi în prima parte a modulului digital (*y* şi *temporar*). Regiştrii au avantajul că pot păstra valorile salvate pe o perioadă nedeterminată de timp.

Determinarea corectitudinii codului sursă din fişierul VHDL se va face in timp real.

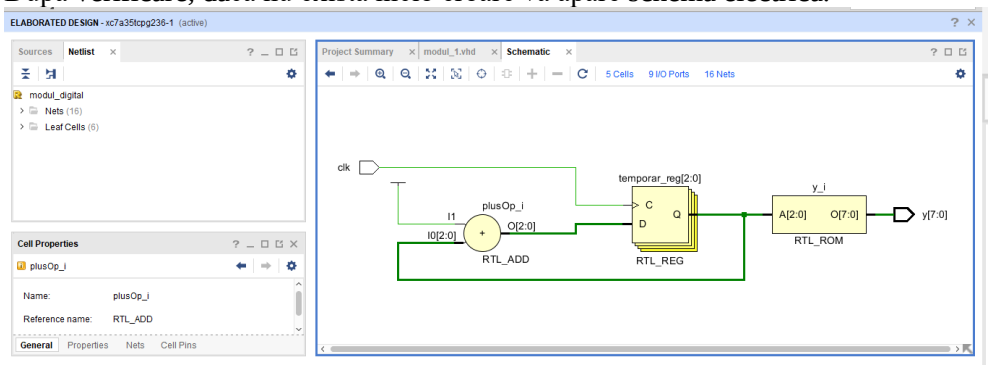
1.3. Analiza proiectului la nivel RTL (Register Transfer Logic)

.

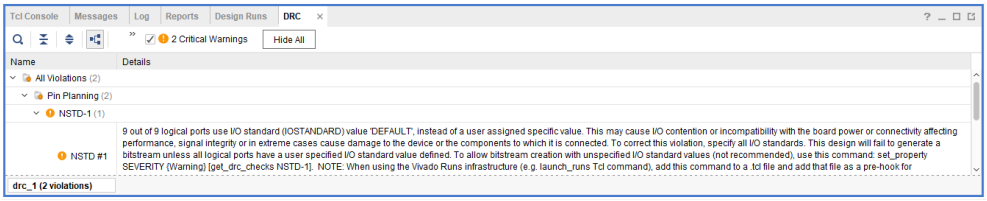
În cadrul ferestrei **Ferestrei Flow navigator** se selectează RTL ANALYSIS -> Open Elaborated Design -> report DRC.



După verificare, dacă nu există nicio eroare va apare schema electrica.



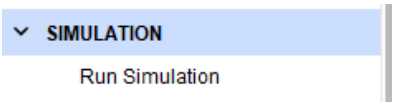
În caz contrar, erorile vor fi afişate în partea de jos a display-ului, în fereastra **DRC**.



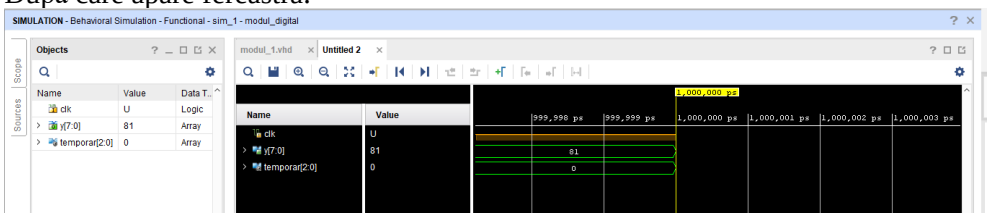
1.4. Simularea proiectului

Simularea proiectului este necesară pentru verificarea corectitudinii funcționale ale modulelor digitale create. În prima fază, simularea se face prin vizualizarea formelor de undă ale stimulilor de intrare şi ale semnalelor rezultate la ieşirile modulelor digitale.

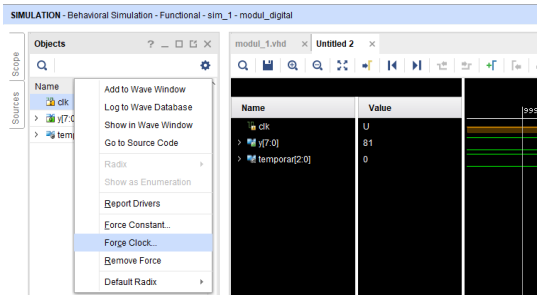
În acest scop, se selecteaza **SIMULATION -> Run Simulation -> Run Behavioral Simulation**



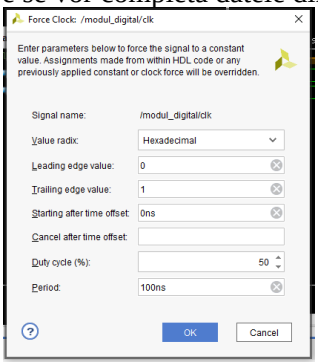
După care apare fereastra:

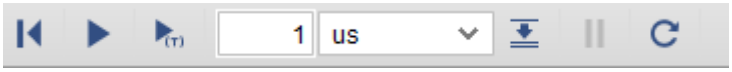


Semnalul clk va fi forta in semnal de ceas prin selectarea acestuia si **click drept -> Force Clock**



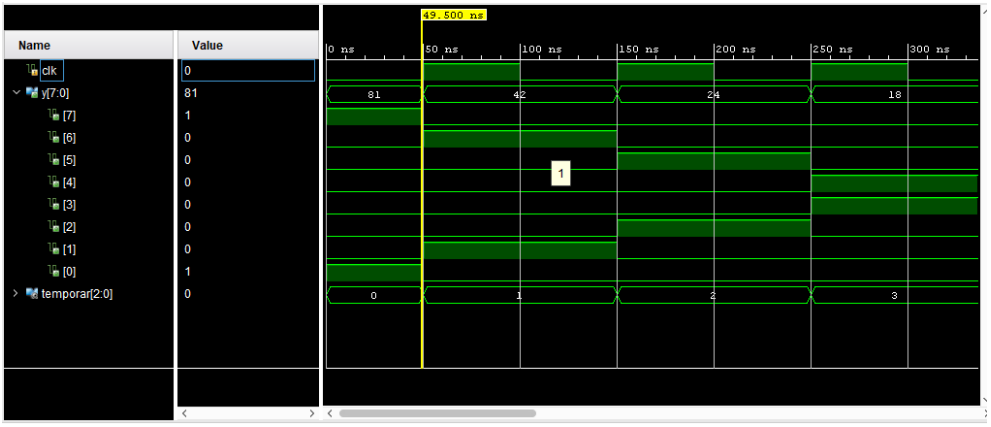
Si apoi apare fereastra in care se vor completa datele din figura:





Se va seta timpul de simulare de 1us după care se va realiza simularea prin apăsarea butonului 

Rezultatul va fi următorul:



În vederea realizării modelului final al modulului digital vor fi adăugate următoarele linii de cod VHDL:

- în zona declarativă a arhitecturii se adaugă linia signal semaf :std_logic;
- se adaugă următoarele procese:

```
process(clk)
        variable cont_temp:std_logic_vector(23 downto 0);
begin
        if (clk'event and clk = '1')            then
                semaf <= '0';
                cont_temp := cont_temp+1;
                if (cont_temp = x"FAF080") then
                        cont_temp := x"000000";
                        semaf <= '1';
                end if;
        end if;
end process;

process(clk)
begin
        if (clk'event and clk = '1')            then
                if semaf = '1'     then
                        temporar <= temporar + '1';
                end if;
        end if;
end process;
```

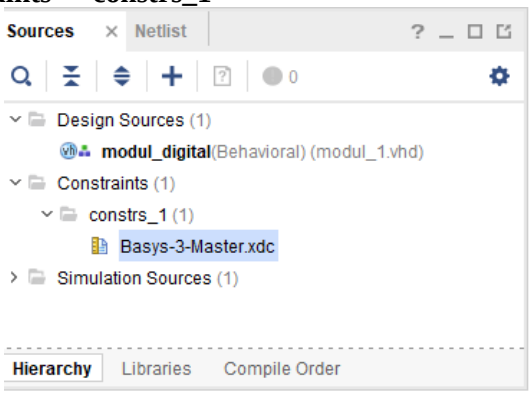
Programul final este următorul:

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

entity modul_digital is
    Port ( clk : in std_logic;
           led : out std_logic_vector(7 downto 0));
end modul_digital;

architecture Behavioral of modul_digital is
            signal temporar:        std_logic_vector(2 downto 0):= (others => '0');
            signal semaf :std_logic;
begin
        process(clk)
                variable cont_temp:std_logic_vector(23 downto 0);
        begin
                if (clk'event and clk = '1')            then
                        semaf <= '0';
                        cont_temp := cont_temp+1;
                        if (cont_temp = x"FAF080") then
                                cont_temp := x"000000";
                                semaf <= '1';
                        end if;
                end if;
        end process;
        process(clk)
        begin
                if (clk'event and clk = '1')            then
                        if semaf = '1'      then      temporar <= temporar + '1';
                        end if;
                end if;
        end process;
        process (temporar)
        begin
                case (temporar) is
                        when "000" => led <= "10000001";
                        when "001" => led <= "01000010";
                        when "010" => led <= "00100100";
                        when "011" => led <= "00011000";
                        when "100" => led <= "00100100";
                        when "101" => led <= "01000010";
                        when "110" => led <= "10000001";
                        when others => led <= (others => '1');
                end case;
        end process;
end Behavioral;
```

1.5. Constrângeri referitoare la atribuirea pinilor

Prin constrângerile de atribuire a pinilor sunt realizate asocierile dintre porturile modulului digital declarate în entitate şi pinii circuitului de tip FPGA utilizat.

Pentru realizarea acestora se deschide fisierul **Basys 3-Master.xdc** din **Source -> Constraints -> constrs_1**



Acest fisier contine toate asocierile pinilor kit-ului de dezvoltare Basys-3. Se vor decomenta doar pinii utilizati in proiect:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]


## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {led[7]}]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

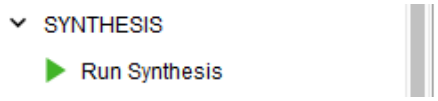
Restul liniilor vor rămâne comentate cu semnul # la începutul liniilor.

După ce sunt aduse aceste modificări, se salvează fiserul.

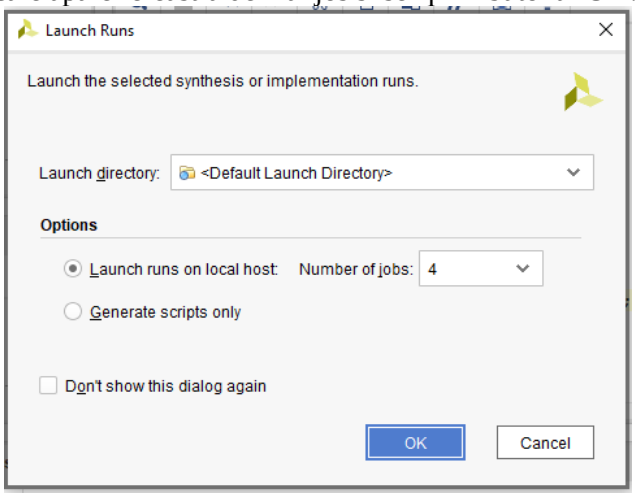
1.6. Sinteza proiectului

După ce proiectul a fost completat, verificat şi s-au adăugat fişierele de constrângeri, acesta poate fi sintetizat şi apoi implementat pentru a se face, în final, programarea structurii reconfigurabile.

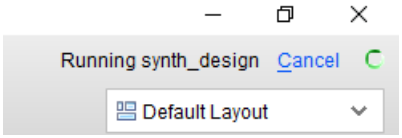
Sinteza se realizează prin selectarea (dublu clic) a grupului **SYNTHESIS –> Run synthesis**.



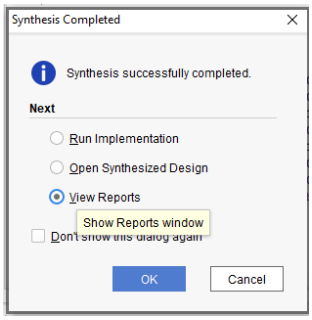
După care apare fereastra de mai jos si se apasă butonul **OK**.



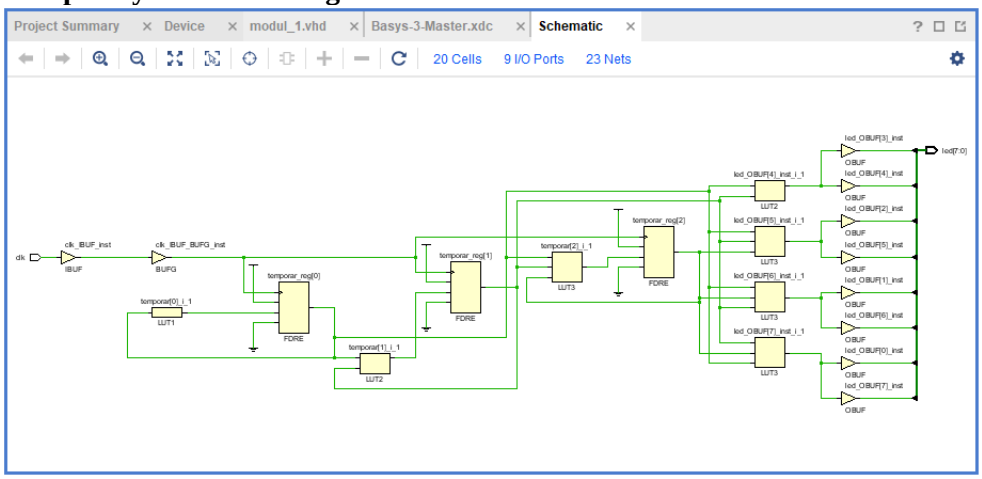
Aceste procese sunt consumatoare de resurse ale calculatorului si dureaza cel putin 1-2 minute. Derularea procesului este arătată în partea dreaptă, sus a ecranului



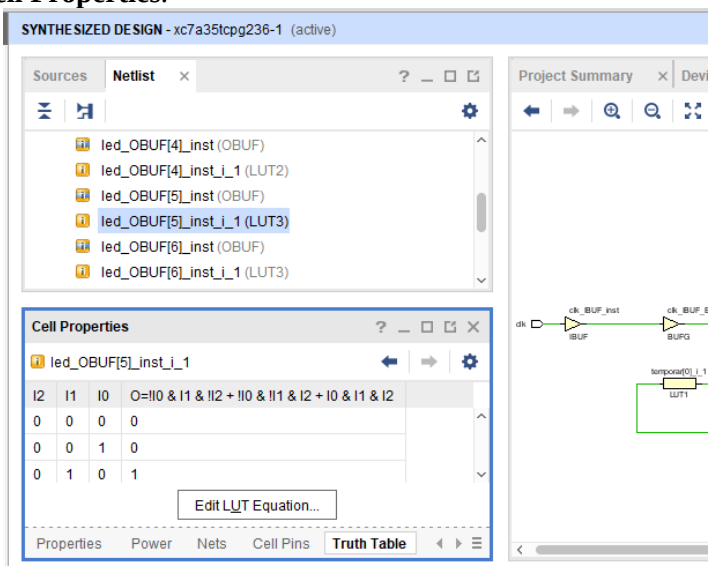
Dacă procesul nu a generat erori poate fi vizualizat primul raport referitor la modulul digital.



Schema electrică generată se poate vizualiza prin selectarea **SYNTHESIS –> Open Synthesized Design -> Schematic**



Proprietatile fiecarui bloc din schema electrica pot fi vizualizate in fereastra **Cell Properties**.



1.7. Implementarea proiectului

Implementarea constă în următoarele procese: Translate, Map şi Place & Route.

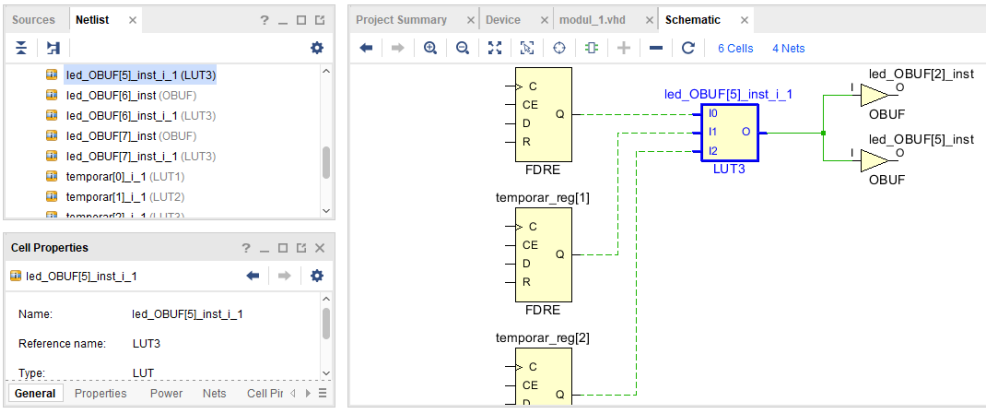
Translatarea serveşte la trecerea de la schema logică rezultată în urma sintezei la primitive Xilinx.

Map-area este procesul în care circuitul este descris la nivel de componente fizice în structura FPGA:

Procesul de Plasare şi rutare constă în formarea fişierelor ce specifică rutările dintre componentele interne ale structurii reconfigurabile, în vederea generării fişierelor de configurare.

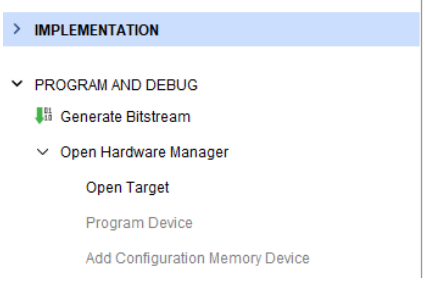
Procesul de implementare se realizează prin selectarea (dublu clic) a grupului **IMPLEMENTATION -> Run implementation** . Dacă procesul s-a terminat fără erori se poate trece la etapa de generare a fişierelor de configurare.

Schema electrica rezultată este cea reală care se realizează în circuitul FPGA. Vizualizarea acesteia se face prin selectarea **IMPLEMENTATION -> Open Implemented Design -> Schematic**.



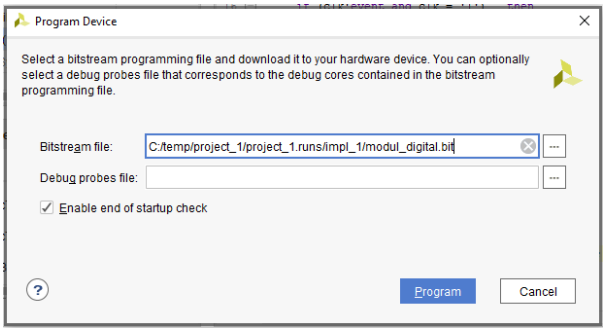
1.8. Configurarea circuitului FPGA

În final, pentru generarea şi implementarea fizică a fişierului binar în structura hardware reconfigurabilă, se apelează la **PROGRAM AND DEBUG**.



Pasii sunt urmatorii:

- Se genereaza fisierul binar de congurare prin selectarea **PROGRAM AND DEBUG -> Generate Bitstream**
- Se conecteaza la circuitul FPGA prin **PROGRAM AND DEBUG -> Open Hardware Manager -> Open Target**
- Se programeaza circuitul FPGA prin **PROGRAM AND DEBUG -> Open Hardware Manager -> Program Device**, se selectează fisierul cu extensia bin după care se apasă butonul **Program**

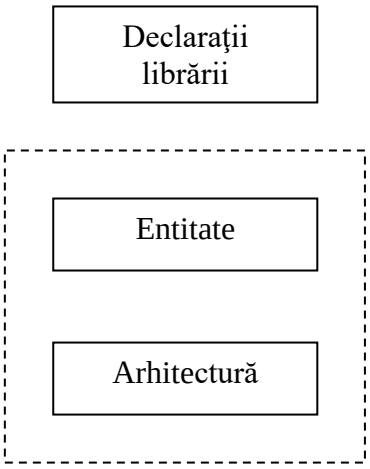


După programare se va vedea rezultatul pe kit-ul de dezvoltare Basys-3.

2. Structura unui program VHDL

Programul prin care se descrie un modul digital în limbajul VHDL conţine trei părţi:

- declararea librăriilor care vor fi utilizate în proiect;
- declararea entităţii modulului ce urmează a fi proiectat;
- descrierea arhitecturii acestuia.



Entitatea descrie interfaţa modulului digital cu semnalele din mediul exterior. O entitate deja declarată poate fi accesată de către alte entităţi.

Sintaxă:

```
entity nume_entitate is
        generic (listă_generică);
port (listă_de_porturi);]
        end entity nume_entitate;
```

Prin specificaţia **entity** se declară numele modulului digital. În plus, pot fi declaraţi parametrii generici şi porturi care fac parte din această entitate. Porturile declarate într-o entitate sunt vizibile în toate arhitecturile atribuite acesteia. Porturile pot fi de intrare, ieşire, buffer sau bidirecţionale (**IN, OUT, BUFFER, INOUT**)

Arhitectura descrie relaţia dintre intrările şi ieşirile porturilor entităţii căreia îi este asociată. O arhitectură poate avea asociată doar o singură entitate, dar o entitate poate avea asociate mai multe arhitecturi prin configuraţie.

Sintaxă:

```
architecture nume_arhitectură of nume_entitate is
-- declaraţii în arhitectură
                begin
                        --specificaţii_concurente
                end [ architecture ] [ nume_arhitectură ];
```

Zona declarativă a unei arhitecturi poate conţine declaraţii de tipuri, semnale, constante, subprograme, componente şi grupuri. Specificaţiile concurente din corpul arhitecturii definesc legăturile dintre intrările şi ieşirile modulului digital pe care-l reprezintă.

Sintaxa generală a unui program în cod VHDL este următoarea:

```
--Declararea librariilor prin clauza library si use
library nume_librarie;
use nume_librarie.nume_pachet.all;

--Declararea entitatii modulului digital
entity nume_entitate is
        generic(nume generice : type := valori_initiale);
        port(nume_porturi : directie tip port);
end entity nume_entitate; --entity[93]

--Corpul arhitecturii
architecture nume_arhitectura of nume_entitate is
        declaratii arhitectura
begin
        specificatii concurente
        end architecture nume_arhitectura;
```

3. Operatori utilizaţi în limbajul VHDL

Pentru implementarea circuitelor combinaţionale, în limbajul VHDL, s-au pus la dispoziţia programatorului operatori logici, aritmetici, de comparaţie, deplasare şi de concatenare.

În tabelul de mai jos sunt prezentaţi, pe scurt, operatorii logici:

Tipul operatorului	Operatori	Tipul datelor
Logic	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR STD_UNLOGIC, STD_UNLOGIC_VECTOR
Aritmetic	+, -, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
Comparaţie	=, /=, <, >, <=, >=	aproape toţi
Deplasare	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenare	&, (, , ,)	La fel ca la operatorii logici, plus

		SIGNED şi UNSIGNED

III. DESFĂŞURAREA LUCRĂRII

1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se va realiza şi implementa un numărător binar reversibil pe 4 biţi cu porturile de intrare/ieşire prezentate în figura de mai jos:

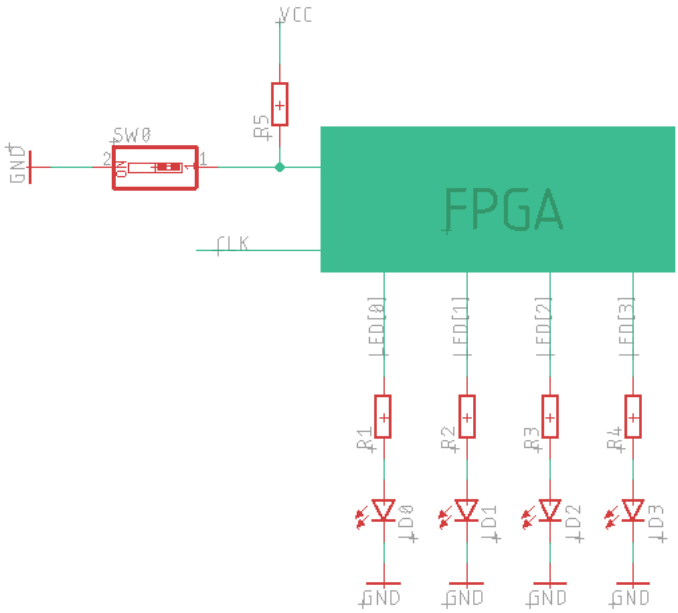
Se va utiliza următorul cod sursă VHDL al modulului digital:

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

entity counter_UpDown is
    Port (  btnC : in std_logic;
            sw : in std_logic_vector(0 downto 0);
            data_out : out std_logic_vector(3 downto 0));
end counter_UpDown;

architecture Behavioral of counter_UpDown is

begin

            process (btnC)
                        variable count_int: std_logic_vector(3 downto 0):= "0000";
            begin
                        if (btnC'event and btnC = '1') then
                                    if (sw(0) = '0') then
                                                count_int := count_int + '1';
                        else
                                                count_int := count_int - '1';
                                    end if;
                        end if;
                        led <= count_int;
            end process;

end Behavioral;
```

Se vor parcurge următorii paşi:

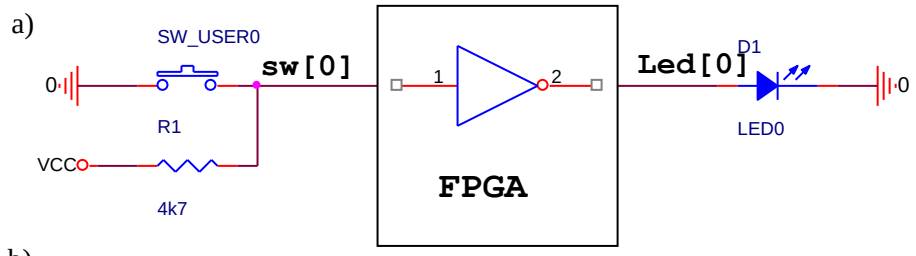
- crearea unui proiect nou;
- inserarea programului de mai sus ca sursă VHDL;
- simularea modulului digital;
- crearea fişierului de constrângeri al pinilor;
- realizarea sintezei acestuia;
- vizualizarea schemelor electrice;
- implementarea proiectului;
- vizualizarea schemei electrice tehnologice;
- configurarea circuitului fizic.

Acest modul digital va fi inserat în următoarea schemă electrică:

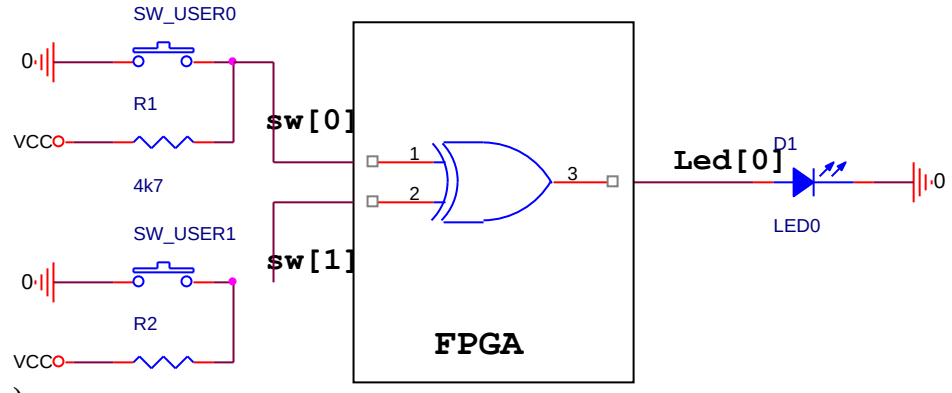


3. Respectând modelul de la punctul anterior, să se realizeze implementarea hardware a funcţiilor logice corespunzătoare pentru funcţionarea următoarelor scheme:

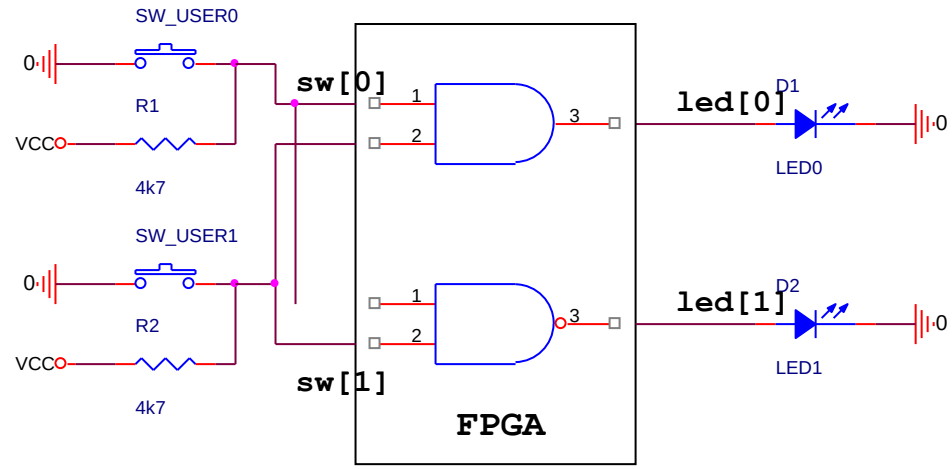
a)



b)



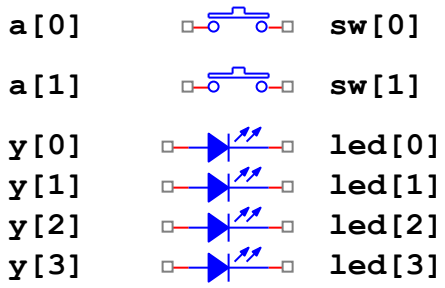
c)



4. După modelul de la punctul 2 să se implementeze hardware ecuaţiile booleene ale următoarelor tabele de adevăr, în forma directă şi forma canonică.

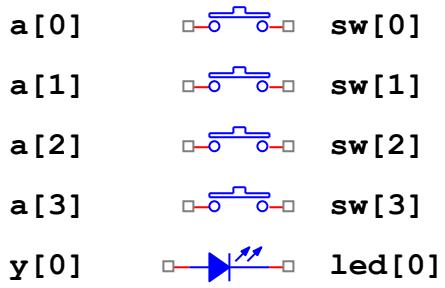
a) Tabela de adevăr pentru funcţia care realizează codarea unui numărător binar pe 2 biţi:

a[0]	a[1]	y[0]	y[1]	y[2]	y[3]
0	0	0	0	0	1
0	1	0	0	1	1
1	0	0	1	1	1
1	1	1	1	1	1



b). Tabela de adevăr pentru funcţia care verifică paritatea impară a unui număr binar pe 4 biţi:

a[0]	a[1]	a[2]	a[3]	y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



Se va realiza corespondența dintre porturile entităților şi pinilor circuitului programabil, după configurațiile din partea dreaptă a tabelelor de adevăr de mai sus.