GEORGE ZĂRNESCU

♦

CIRCUITE INTEGRATE DIGITALE
ÎNDRUMAR DE LABORATOR

Editura
NAUTICA

GEORGE ZĂRNESCU

CIRCUITE INTEGRATE DIGITALE
ÎNDRUMAR DE LABORATOR

Editura
NAUTICA

Editura NAUTICA**, 2013**

Editură recunoscută de CNCSIS

Str. Mircea cel Bătrân nr.104

900663 Constanţa, România

tel.: +40-241-66.47.40

fax: +40-241-61.72.60

e-mail: info@imc.ro

www.edituranautica.org.ro

Descrierea CIP a Bibliotecii Naţionale a României
ZĂRNESCU, GEORGE
Circuite integrate digitale : îndrumar de laborator /
George Zărnescu. - Constanţa : Nautica, 2013
Bibliogr.
ISBN 978-606-681-007-4

621.3.049.77(075.8)

Prefață

Acestă lucrare își propune să ajute, printr-o manieră sistematică, studenții electroniști din primii ani de facultate, să dobândească abilitățile necesare creării circuitelor logice digitale. În urma efectuării laboratorului de circuite integrate digitale studentul se va transforma într-un inginer proiectant de circuite logice digitale.

Scopul acestei lucrări este acela de a exemplifica modul în care pot fi create circuite logice digitale de complexitate medie și mare, utilizând cele mai recente metodologii de proiectare, software-uri industriale și platforme reconfigurabile de ultimă oră.

La finalul laboratorului studentul va cunoaște metodologia de proiectare, simulare și testare a circuitelor logice digitale, va putea utiliza cu ușurință metodologia RT cu scopul proiectării circuitelor logice și va putea implementa circuitele create în platforme reconfigurabile cu ajutorul software-urilor dedicate.

Constanța, 2013

Autorul

Cuprins

Laboratorul 1

Sintetizarea, simularea și implementarea circuitelor logice digitale

1.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea procedurilor de sintetizare, simulare și implementare a circuitelor logice digitale. Exercițiile prezentate în acest laborator au rolul de a exemplifica modul în care software-ul Quartus II este utilizat în crearea unui proiect, a unui fisișier VHDL, în sintetizarea, vizualizarea, simularea funcțională și implementarea circuitelor logice în sisteme reconfigurabile.

1.2 Instrumente necesare

Software-ul Altera Quartus II
Kit educațional

1.3 Noțiuni teoretice

Sintetizarea reprezintă procesul de prelucrare al liniilor de cod dintr-un fișier VHDL și are ca finalitate obținerea circuitului logic.

Simularea funcțională reprezintă procesul de verificare al funcționării circuitului logic într-un mediu virtual. Simularea funcțională este necesară pentru a observa dacă circuitul funcționează conform specificațiilor inițiale.

Implementarea reprezintă procesul de configurare al unui dispozitiv fizic cu circuitul logic sintetizat.

1.4 Desfășurarea lucrării

1.4.1 Exercițiul 1 - Crearea unui proiect și a unui fișier VHDL în Quartus II

Pasul 1. Se va deschide programul Quartus II: Start\All Programs\Altera\Quartus II.

Pasul 2. Pentru a crea un proiect în Quartus II se va proceda astfel: File\New Project Wizard. În fereastra nou apărută se va da Next.

Pasul 3. În fereastra nou apărută se va alege locația proiectului, numele proiectului și numele entității.

ATENȚIE!!!

Întotdeauna numele proiectului trebuie să fie identic cu numele entității. O locație adecvată poate fi D:\student\laborator_1.

La fiecare laborator, în locația D:\student\, se va crea un folder nou cu numele laborator_numărul-laboratorului în care se va lucra la laboratorul respectiv. Pentru acest exercițiu numele proiectului și numele entității vor fi proj1. Se va da Next de două ori.

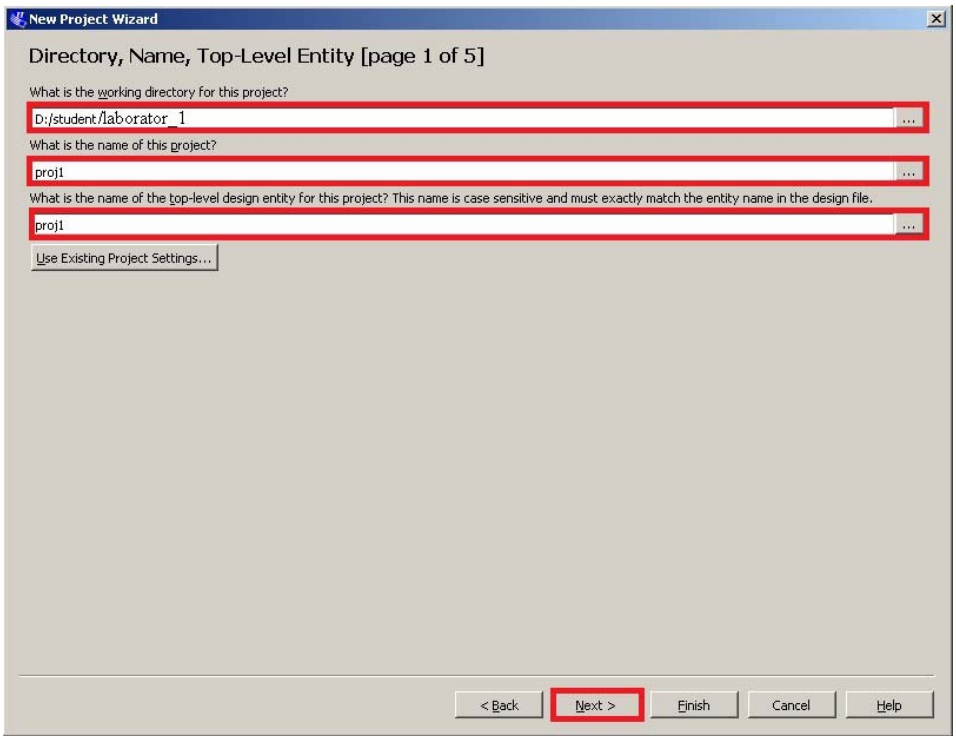


Figura 1.1. Fereastra New Project Wizard. Se vor introduce locația și numele proiectului.

Pasul 4. În fereastra nou apărută se va alege familia de FPGA-uri, Cyclone IV E, și tipul FPGA-ului, EP4CE22F17C8. Un FPGA este un circuit integrat configurabil.

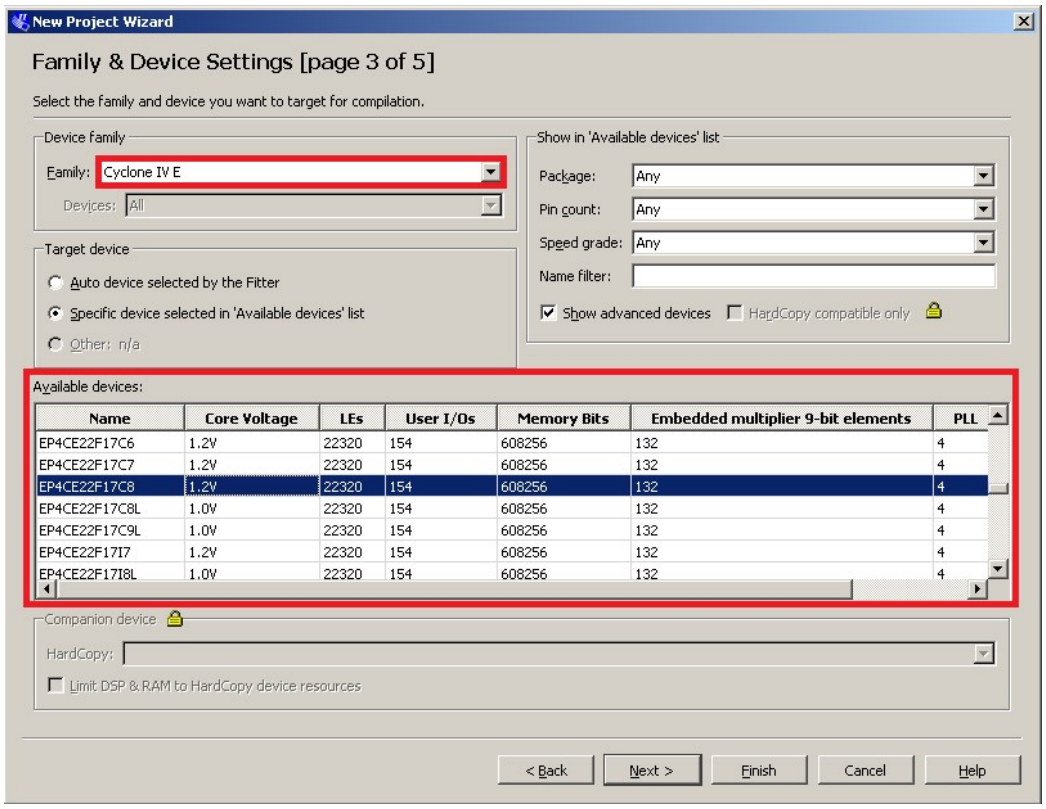


Figura 1.2. Fereastra New Project Wizard. Se va alege tipul FPGA-ului.

Pasul 5. Se va da click pe Finish.

În momentul de față am creat un proiect în Quartus II, iar acest lucru este evidențiat în figura 1.3. Proiectul nou creat se numește proj1 și poate fi accesat din Project Navigator, tab-ul Hierarchy. Fișierele unui proiect pot fi accesate din tab-ul Files.

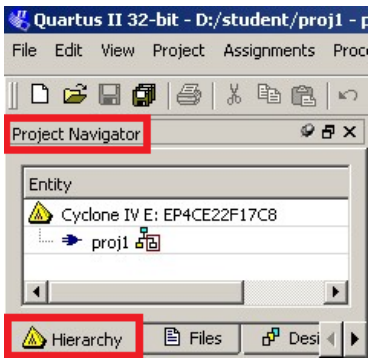


Figura 1.3. Fereastra Project Navigator.

În continuare vom crea un fișier VHDL pentru proiectul nou creat. Se va proceda în felul următor.

Pasul 1. Se va alege File\New...

Pasul 2. În fereastra nou apărută se va alege VHDL File și se va da Ok.

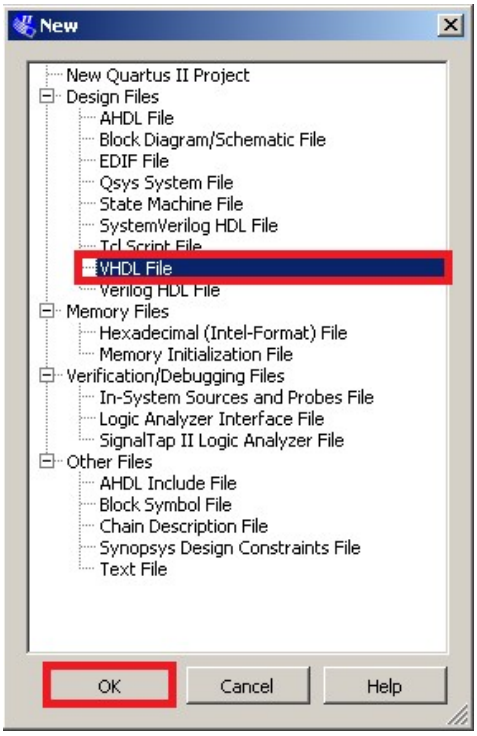


Figura 1.4. Fereastra New.

Pasul 3. În momentul de față am creat un fișier VHDL pe care îl vom salva prin File\Save As în forder-ul în care a fost salvat noul proiect. Acest folder se numește laborator_1. Un proiect și fișierele ce îl însoțesc trebuie să existe în același folder.

1.4.2 Exercițiul 2 – Sintetizarea, vizualizarea și implementarea unui circuit logic în Quartus II

Sintetizarea și implementarea se vor face pentru fișierul VHDL din figura 1.5. Liniile de cod se vor scrie în fișierul creat anterior.

```
1
2   -- librarii ----------------------
3   library IEEE;
4   use ieee.std_logic_1164.all;
5   ---------------------------------
6
7   -- intrari si iesiri -------------
8   entity proj1 is
9
10  port(
11        KEY   : in  std_logic_vector(1 downto 0);
12        LED   : out std_logic_vector(0 downto 0)
13       );
14  end;
15  ---------------------------------
16
17  -- arhitectura sistemului --------
18  architecture circ1 of proj1 is
19
20  begin
21
22  LED(0) <= KEY(0) AND KEY(1);
23
24  end;
25  ---------------------------------
```

Figura 1.5. Fișier VHDL.

Implementarea se va efectua în kit-ul educațional DE0-NANO evidențiat în figura 1.6. Intrările circuitului logic vor fi cele două întrerupătoare din dreapta sus. Ieșirea circuitului logic va fi primul led din stânga celor două întrerupătoare. Restul led-urilor nu se vor utiliza.

Figura 1.6. Kit-ul educațional.

Pasul 1. În exercițiul anterior am creat un proiect în Quartus II și un fișier VHDL. În fișierul nou creat se vor scrie liniile de cod de mai sus. Se va salva fișierul și se va compila, figura 1.7, sau sintetiza, figura 1.8: Processing\Start Compilation sau Processing\Start\Start Analysis & Synthesis.

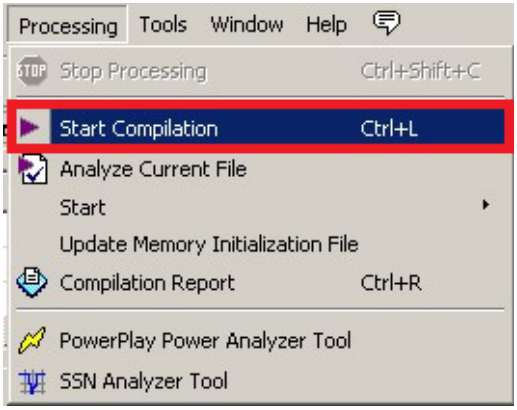


Figura 1.7. Comanda de compilare

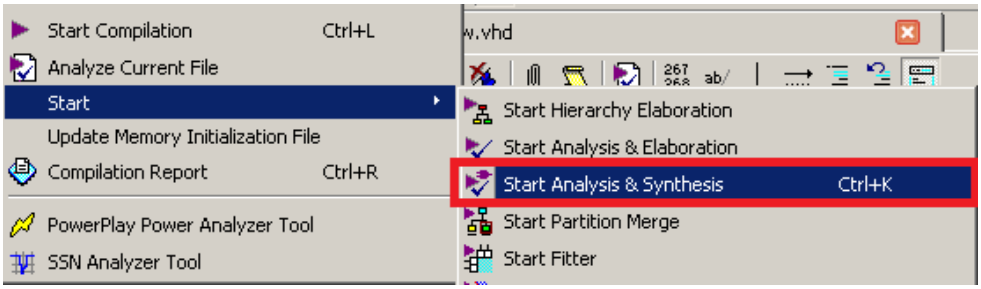


Figura 1.8. Comanda de sintetizare.

Pasul 2. Vizualizarea circuitului sintetizat se efectuează astfel:
Tools/Netlist Viewers/RTL Viewer.

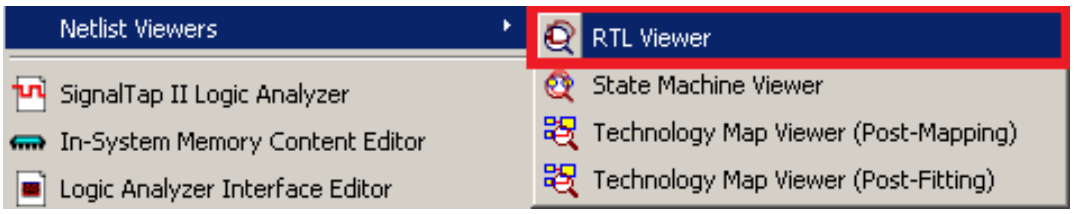


Figura 1.9. Vizualizarea circuitului sintetizat.

Pasul 3. Vom cupla intrările circuitului logic cu pinii FPGA-ului, astfel:
Assignments/Import Assignments…

Fereastra pentru cuplarea pinilor este evidențiată în figura 1.10. În această fereastra se va da click pe butonul reprezentat cu roșu și se va selecta fișierul DE0_Nano.qsf. Acest fișier ar trebui să existe pe partiția D:\.

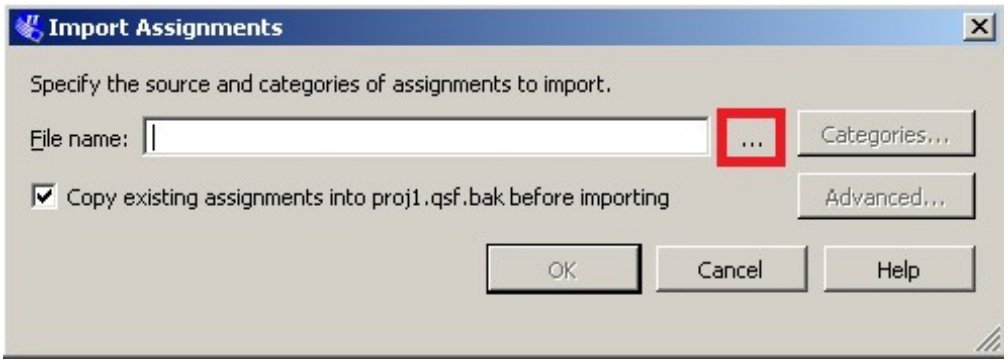


Figura 1.10. Fereastra Import Assignments.

Pasul 4. Se va recompila proiectul.

Pasul 5. Se va implementa circuitul logic în kit-ul educațional cu FPGA, astfel: Tools/Programmer

Figura 1.11. Comanda de programare.

Pasul 6. În fereastra Programmer se va selecta Hardware Setup, butonul marcat cu roșu.

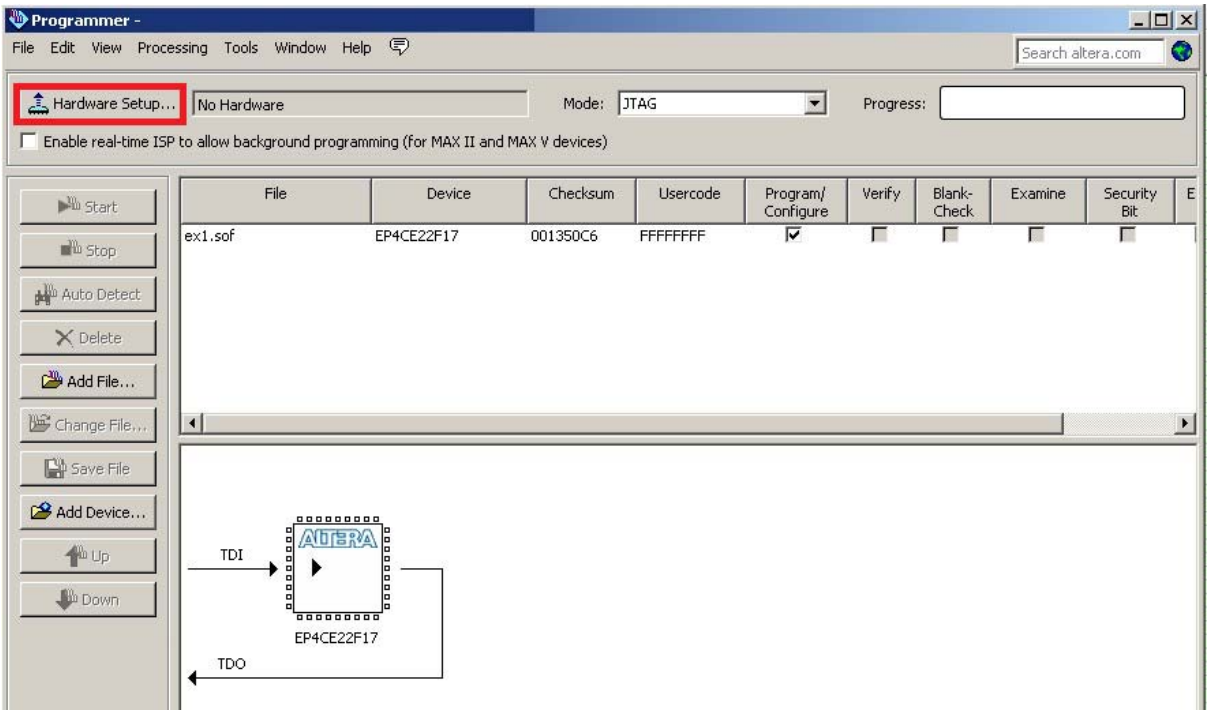


Figura 1.12. Fereastra Programer.

Pasul 7. În fereastra nou apărută se va selecta USB-Blaster din Currently selected hardware și se va închide fereastra apăsând pe Close.

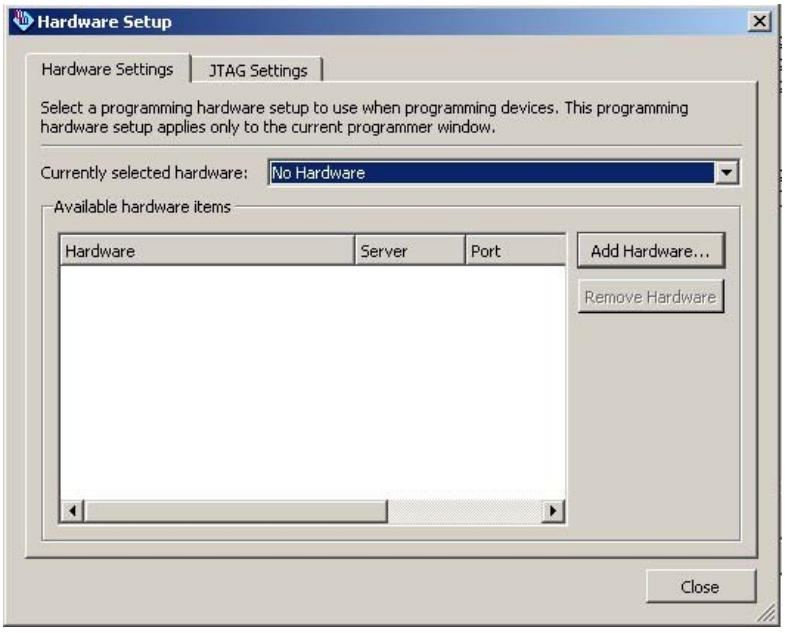


Figura 1.13. Fereastra Hardware Setup.

Pasul 8. În fereastra Programmer se va da Start. Implementarea circuitului logic va începe imediat. Atunci când progress bar-ul va afișa 100% știm că circuitul logic a fost implementat cu succes în kit-ul educațional cu FPGA.

1.4.3 Exercițiul 3 – Simularea funcțională a unui circuit logic sintetizat cu ajutorul Quartus II

Pasul 1. Se va deschide programul Altera U. P. Simulator, figura 1.14, astfel Start/All Programs/Altera/University Program/Simulation Tools/

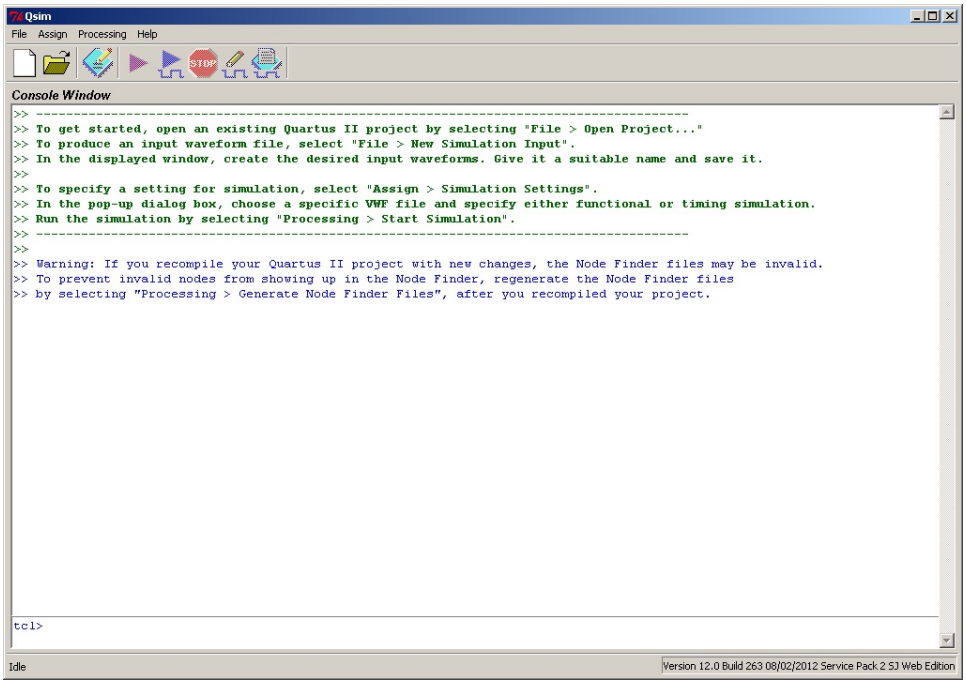


Figura 1.14. Fereastra Qsim.

Familia de FPGA-uri Cyclone IV E nu este recunoscută de simulator. În acest caz ne vom întoarce în Quartus II și vom alege un FPGA pe care simulatorul să îl recunoască.

Pasul 2. În Quartus II, pentru proiectul creat anterior, vom alege un FPGA, pe care simulatorul să îl recunoască, astfel: Assignments/Devices. În fereastra nou apărută se va alege familia de FPGA-uri, Cyclone II, și tipul FPGA-ului, EP2C20F484C7.

Pasul 3. Vom recompila proiectul.

Pasul 4. În programul Altera U. P. Simulator se va deschide proiectul creat anterior astfel: File/Open Project/
Proiectul este denumit proj1 şi are extensia qpf (quartus project file).

Pasul 5. Vom crea un netlist pentru circuitul logic din proiectul proj1. Vom da click pe butonul evidenţiat cu roşu în figura 1.15.

Figura 1.15. Butonul de sintetizare.

ATENŢIE!!!
Precizăm că un netlist este o listă de componente şi interconectări ce evidenţiază alcătuirea circuitului logic.

Pasul 6. Vom crea un fişier de simulare astfel:
File/New Simulation Input File/

Pasul 7. În fereastra nou apărută se va alege
Edit/Insert/Insert Node or Bus/
Astfel vom face vizibile intrările şi ieşirea circuitului logic.

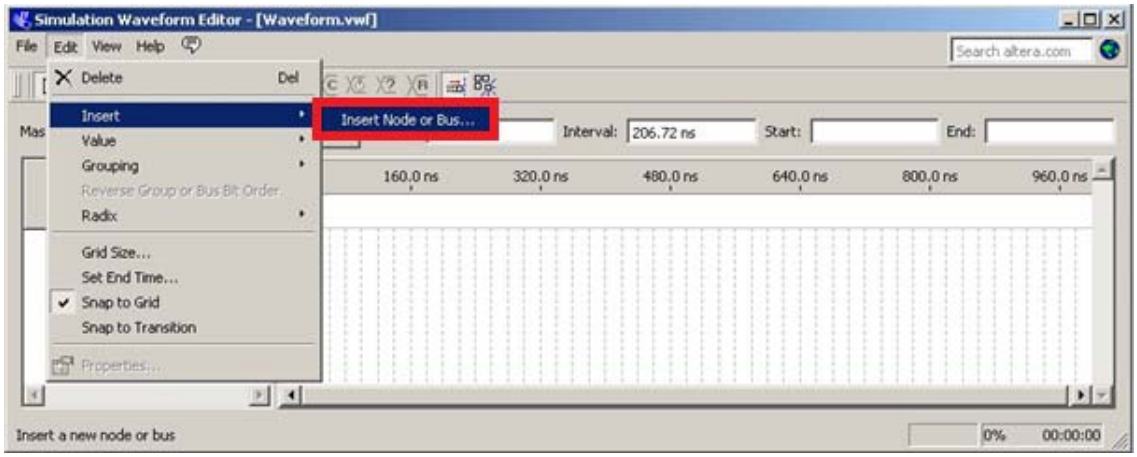


Figura 1.16. Comanda cu care se introduc în fişierul de simulare intrările şi ieşirile circuitului logic.

Pasul 8. În fereastra nou apărută se va da click pe Node Finder.

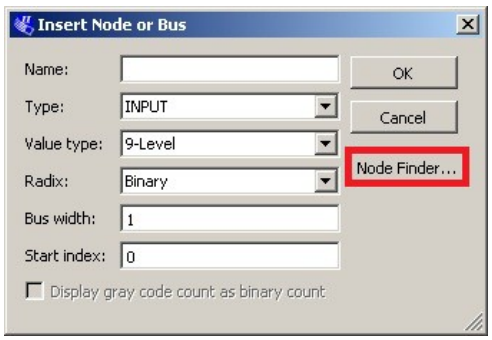


Figura 1.17. Fereastra Insert Node or Bus.

Pasul 9. În fereastra nou apărută, din tab-ul Filter, se va selecta Pins: all. Apoi se va da click pe List, după care cu ajutorul butonului >> selectăm toate nodurile găsite. Un nod poate fi o intrare, o ieșire sau un semnal intermediar. ATENȚIE!!! În cazul în care Node Finder-ul nu găsește nodurile circuitului logic, atunci ele trebuie introduse manual. Acest proces necesită timp și atenție.

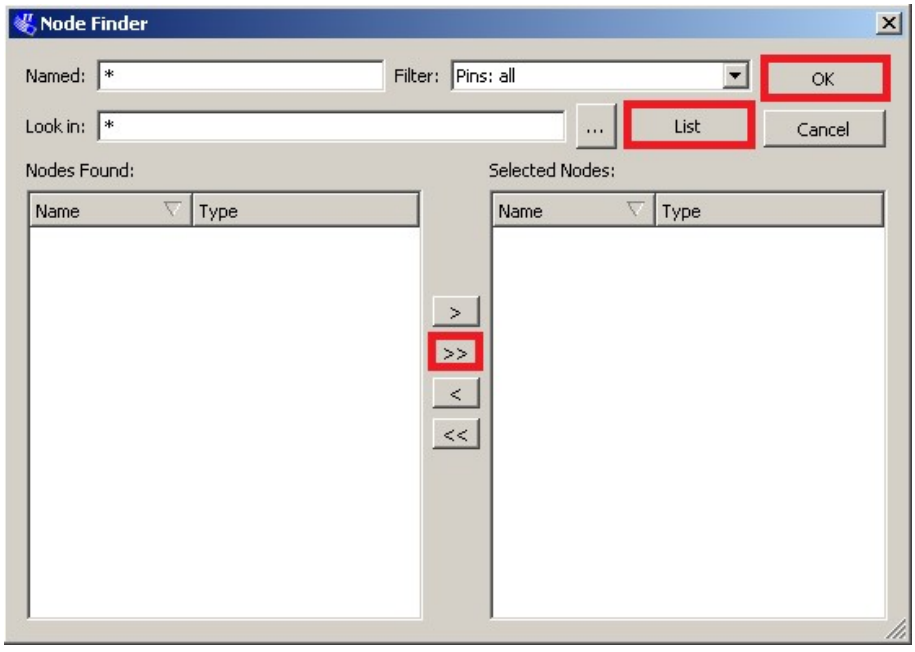


Figura 1.18. Fereastra Node Finder.

Pasul 10. În momentul de față vom salva fișierul de simulare cu numele proj1 în folderul unde au fost create proiectul și fișierul VHDL. În acest moment fereastra de simulare ar trebui să arate ca în figura 1.19.

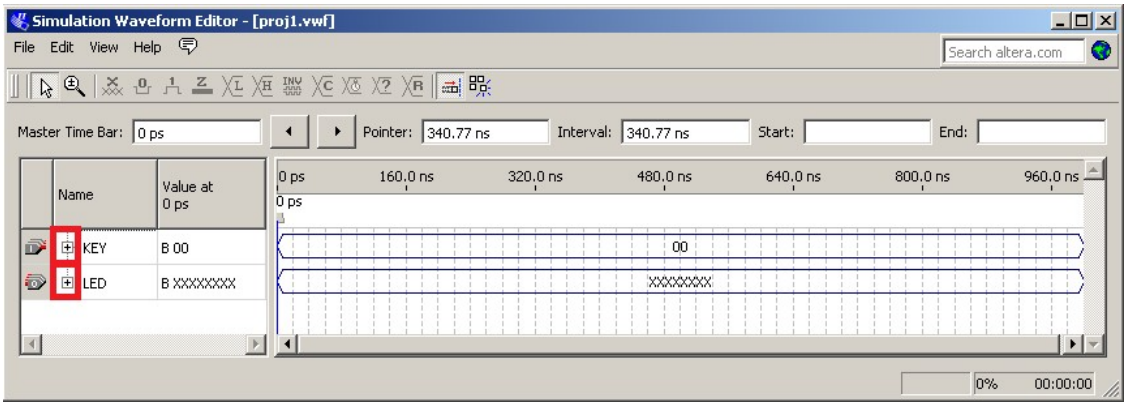


Figura 1.19. Fereastra Simulation Waveform Editor.

Pasul 11. Vom da click pe butoanele reprezentate cu roșu, în figura 1.19, și vom mări fereastra de simulare, pentru a vedea toate nodurile de intrare și ieșire, ca în figura 1.20.

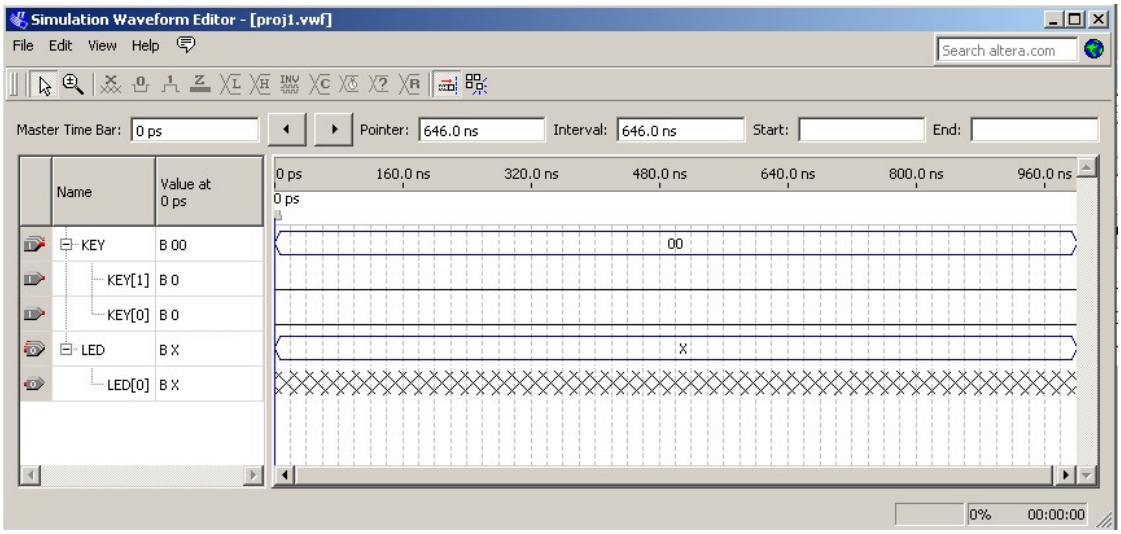


Figura 1.20. Vizualizarea componentelor semnalelor de intrare și ieșire.

Pasul 12. În acest moment vom crea un vector de simulare pentru cele două intrări KEY[0] și KEY[1]. Vom selecta vectorul KEY dând click pe numele acestuia în fereastra de simulare, după care vom da click pe butonul evidențiat cu roșu în figura 1.21.

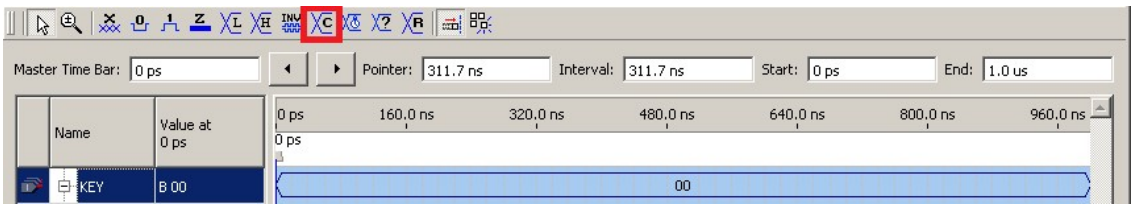


Figura 1.21. Configurarea semnalelor de intrare.

Pasul 13. În fereastra nou apărută vom modifica Count every 10.0 ns cu 100.0 ns. În acest moment fereastra de simulare arată în felul următor.

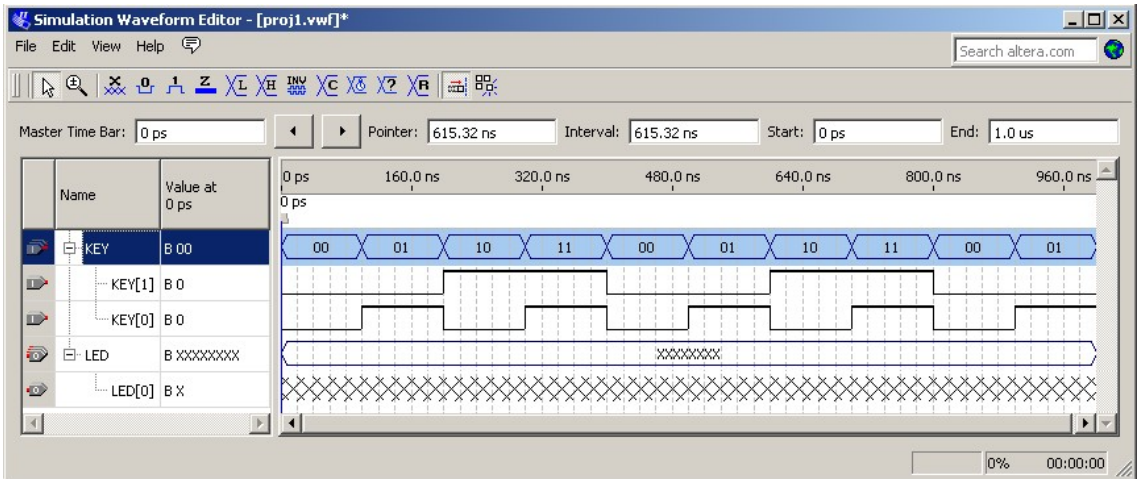


Figura 1.22. Semnalele de intrare au fost configurate.

Pasul 14. În acest moment vom salva fișierul de simulare și ne vom întoarce la fereastra Qsim, figura 1.14.

Pasul 15. Vom integra în simulare fișierul de simulare creat anterior, astfel:
Assign\Simulation Settings

Pasul 16. În fereastra nou apărută, evidențiată în figura 1.23, se va selecta fișierul de simulare dând click pe Browse... . Se va selecta la Simulation Type – Functional și se va da OK.

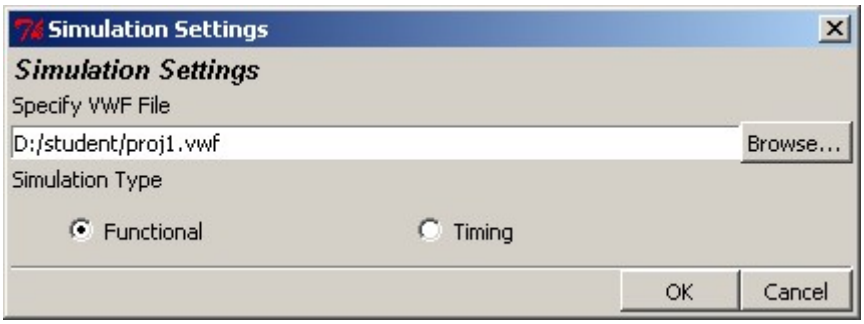


Figura 1.23. Fereastra Simulation Settings.

Pasul 17. Se va simula comportamentul circuitului logic. Se va da click pe butonul evidențiat cu roșu în figura 1.24.

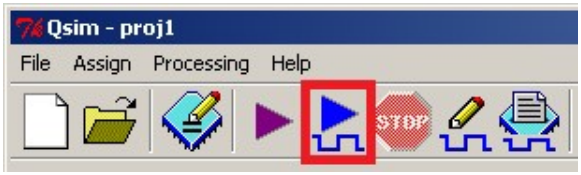


Figura 1.24. Butonul de simulare.

Pasul 18. Se va inspecta rezultatul simulării pentru a observa dacă circuitul logic funcționează corect sau nu. În cazul în care circuitul funcționează greșit, liniile de cod din fișierul VHDL trebuie revizuite, după care circuitul va fi recompilat și va fi resimulat. Procesul se repetă până în momentul în care specificația inițială privind comportamentul circuitului logic este identică cu rezultatul simulării.

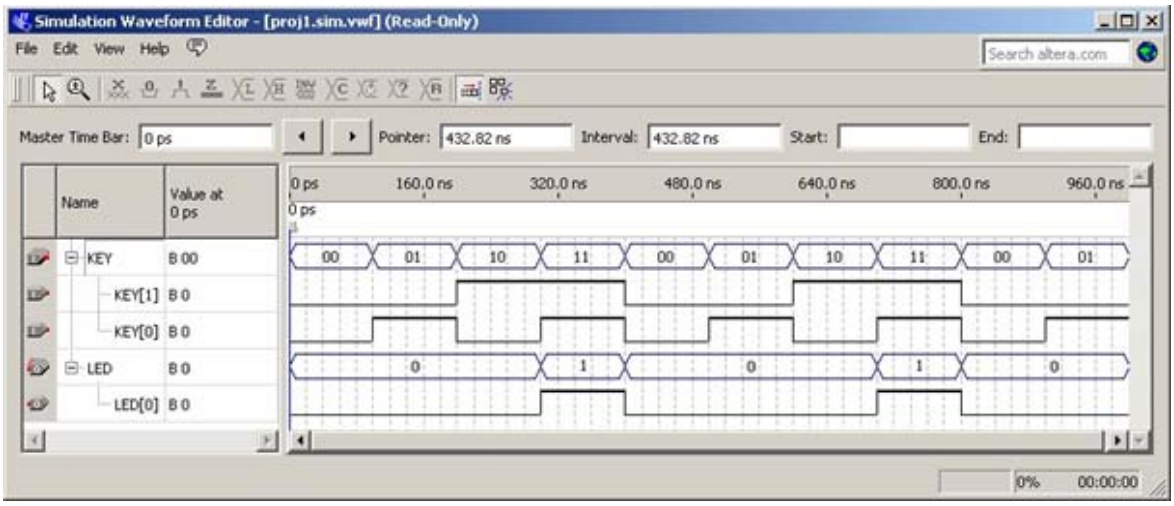


Figura 1.25. Rezultatul simulării funcționării circuitului considerat.

Laboratorul 2

Implementarea circuitelor logice combinaționale simple

2.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea metodologie de proiectare a circuitelor logice. Exercițiile prezentate în acest laborator au rolul de a exemplifica modul în care este utilizată metodologia de proiectare atunci când se dorește instanțierea unei porți logice sau crearea unui circuit logic combinațional simplu.

La finalul acestui laborator studentul va putea identifica porțile logice ce alcătuiesc un circuit simplu. Va putea determina funcția logică (descrisă ca sumă de produse) a unui circuit logic combinațional pe baza unui tabel de adevăr. Va putea deduce o funcție logică pe baza unui circuit logic implementat cu porți logice. Va putea descrie comportamentul unui multiplexor și a operatorilor relaționali (=, >, >=, <, <=, /=).

2.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

2.3 Noțiuni teoretice

Implementarea unui circuit logic într-un sistem reconfigurabil devine o sarcină foarte simplă în momentul în care se respectă cu strictețe etapele metodologiei de proiectare. Această metodologia cuprinde etapele de modelare, sintetizare, verificare și implementare a unui circuit logic. Procesul prin care se crează circuite logice devine unul automat cu ajutorul software-ului Quartus II și a metodologiei de proiectare. Acest proces automat poartă numele de proiectare electronică automată (EDA – electronic design automation).

În figura 2.1 sunt evidențiate etapele metodologiei de proiectare. Fiecare etapă este analizată în paginice ce urmează:

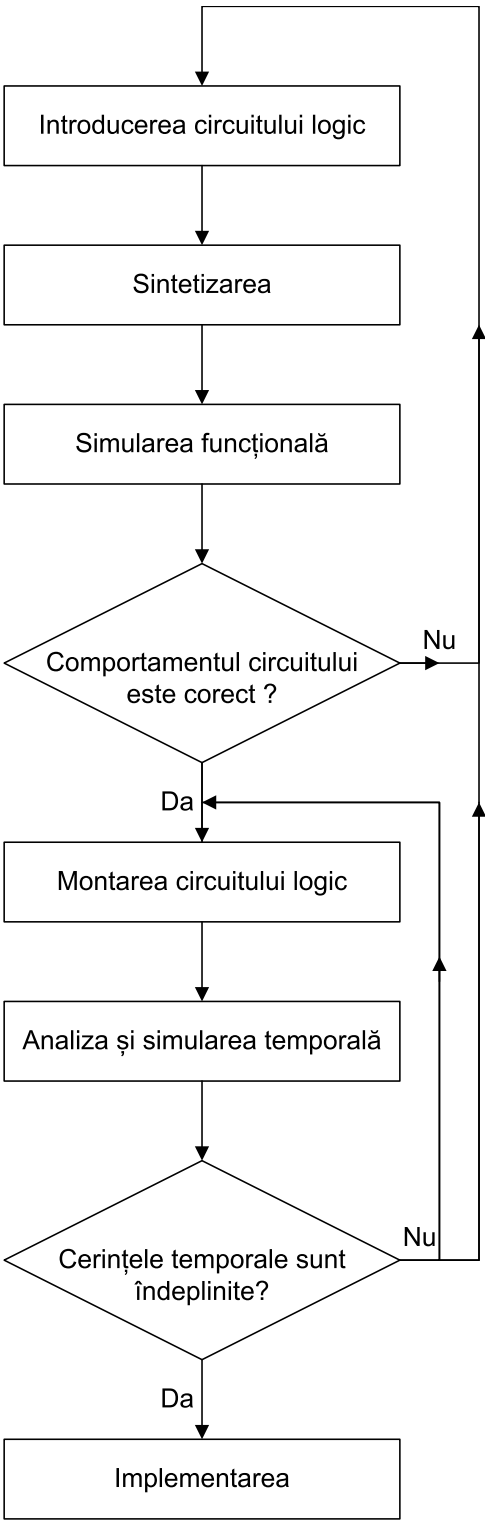


Figure 2.1. Etapele metodologiei de proiectare. Diagrama este preluată din tutorialul Quartus_II_Introduction.pdf de pe site-ul www.altera.com

Introducerea circuitului logic – reprezintă procesul prin care funcționarea unui circuit logic este evidențiată cu ajutorul limbajului de descriere hardware VHDL, figura 2.2.

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3   use ieee.std_logic_unsigned.all;
 4   ------------------------------------------------------
 5   entity proiect_top is
 6       port(
 7                CLOCK_50 : std_logic;
 8                KEY : in std_logic_vector(1 downto 0);
 9                SW : in std_logic_vector(9 downto 9);
10                LEDG : out std_logic_vector (0 downto 0);
11                LEDR : out std_logic_vector (9 downto 5);
12                HEX0, HEX1, HEX2, HEX3 : out std_logic_vector(0 to 6)
13           );
14   end proiect_top;
15   ------------------------------------------------------
16   architecture proiect_top_arch of proiect_top is
17
18   component debouncer
19       port(
20                clk, key_pressed    : in std_logic;
21                key_pressed_ena     : buffer std_logic
22           );
23   end component;
24
25   component pseudo_random_number
26       port (  clk1, rst1, clk2, rst2, start12 : in std_logic;
27                display_ena                    : buffer std_logic);
28   end component;
```

Figura 2.2. Circuit logic evidențiat cu ajutorul limbajului de descriere hardare VHDL.

Sintetizarea – reprezintă procesul de prelucrare al liniilor de cod dintr-un fișier VHDL și are ca finalitate obținerea circuitului logic, figura 2.3.

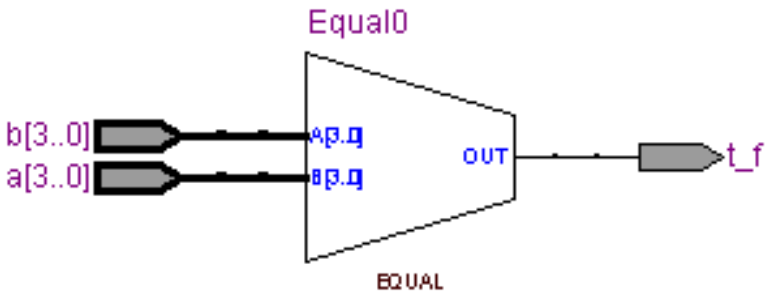


Figure 2.3. Circuit logic digital sintetizat.

Simularea funcțională – reprezintă procesul de verificare al comportamentului circuitului sintetizat. Se analizează dacă circuitul logic funcționează conform specificațiilor, figura 2.4. Analiza funcțională nu ia în considerare comportamentul temporal al circuitului.

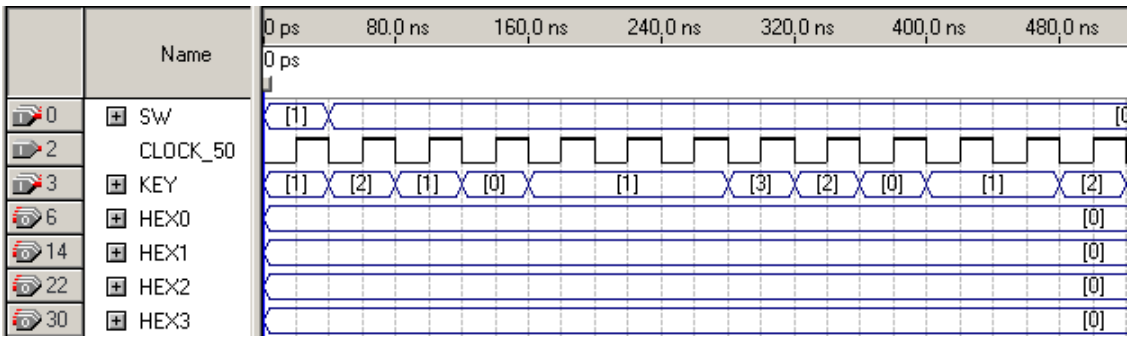


Figure 2.4. Simularea funcțională.

Montarea circuitului logic – se execută în două etape. Prima etapă poartă numele de plasare iar a doua etapă se numește rutare. Plasarea reprezintă procesul prin care circuitul logic digital este amplasat în blocurile logice ale FPGA-ului, figura 2.5, iar rutarea reprezintă procesul prin care se stabilesc conexiunile între blocurile logice, figura 2.6.

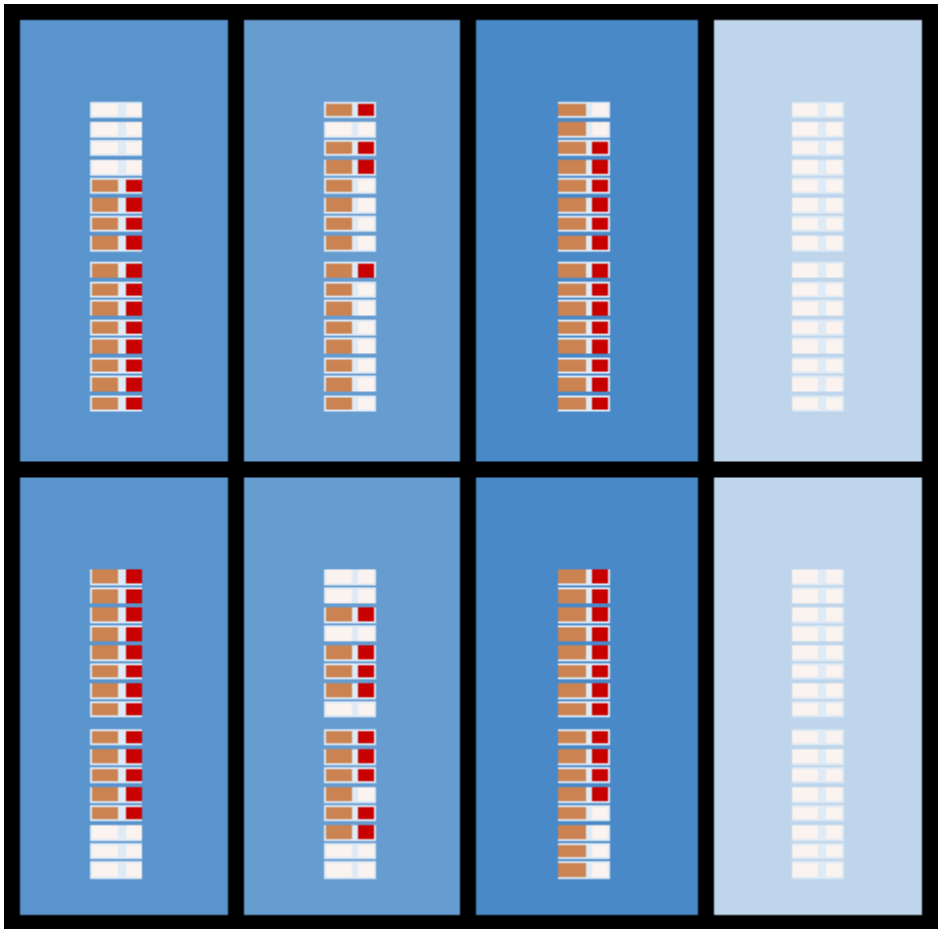


Figure 2.5. Plasarea unui circuit logic digital în blocurile logice configurabile ale unui FPGA.Blocurile logice sunt ocupate total, parțial sau deloc.

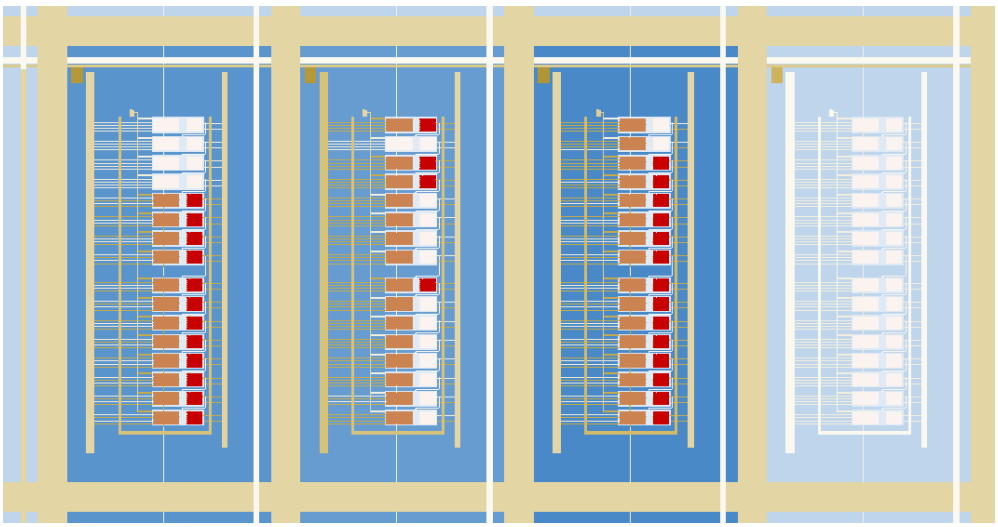


Figure 2.6. Rutarea blocurilor logice.

Analiza temporală – reprezintă procesul prin care se calculează întârzierile semnalelor ce se propagă pe diferite căi în circuitul logic și pe baza lor se determină performanțele circuitului, figura 2.7.

Timing Analyzer Summary			
Worst-case tsu	1.706 ns	5.000 ns	3.294 ns
Worst-case tco	2.008 ns	10.000 ns	7.992 ns
Worst-case th	N/A	None	0.252 ns
Clock Setup: 'CLOCK_50'	10.889 ns	50.00 MHz (period = 20.000 ns)	109.76 MHz (period = 9.111 ns)

Figure 2.7. Rezultatele analizei temporale ale circuituuil logic digital considerat.

Simularea temporală – reprezintă procesul de verificare al comportamentului temporal al circuitului logic, figura 2.8.

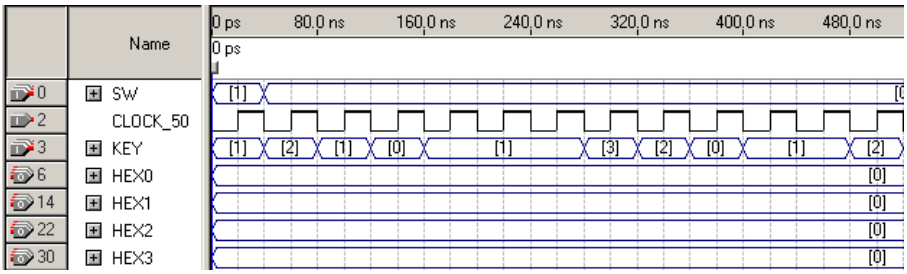


Figure 2.8. Simularea temporală a circuitlui logic digital considerat.

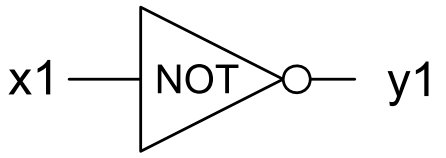
Implementarea – reprezintă procesul de configurare al unui dispozitiv fizic cu circuitul logic sintetizat.

2.4 Desfășurarea lucrării

2.4.1 Exercițiul 1 – Porțile logice fundamentale

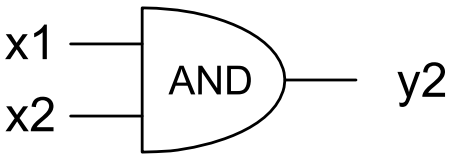
2.4.1.1 Noțiuni teoretice

În figurile 2.9-2.15 sunt evidențiate porțile logice fundamentale împreună cu tabelele lor de adevăr.



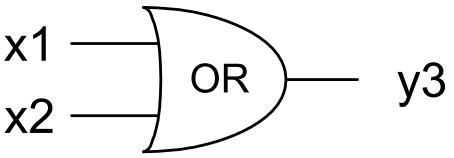
x1	y1
0	1
1	0

Figure 2.9. Poarta logică fundamentală NOT (NU).



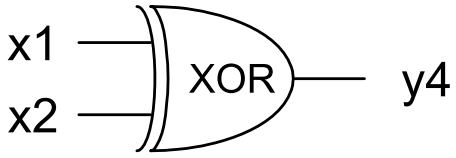
x1	x2	y2
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.10. Poarta logică fundamentală AND (ȘI).



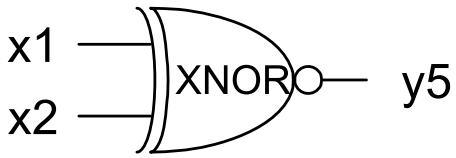
x1	x2	y3
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.11. Poarta logică fundamentală OR (SAU).



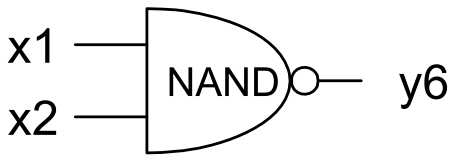
x1	x2	y4
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.12.Poarta logică fundamentală XOR (SAU EXCLUSIV).



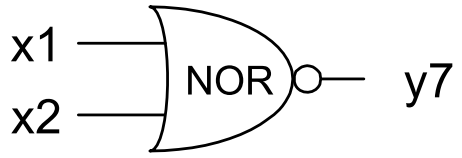
x1	x2	y5
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2.13. Poarta logică fundamentală XNOR (SAU NU EXCLUSIV).



x1	x2	y6
0	0	1
0	1	1
1	0	1
1	1	0

Figure 2.14. Poarta logică fundamentală NAND (ȘI NU).



x1	x2	y7
0	0	1
0	1	0
1	0	0
1	1	0

Figure 2.15. Poarta logică fundamentală NOR (SAU NU).

Poarta logică NOT are o singură intrare și o singură ieșire. Porțile AND, OR, NAND, NOR, XOR și XNOR pot avea două sau mai multe intrări și o singură ieșire. Cu ajutorul porților logice fundamentale se pot crea circuite logice de diferite complexități.

2.4.1.2 Efectuarea exercițiului 1

1. Funcționarea porților logice fundamentale va fi descrisă în programul VHDL numit porti_logice în locația indicată. Spre exemplu funcționarea porții AND poate fi descrisă astfel: y2 <= x1 and x2.
2. Se va compila proiectul.
3. Se vor vizualiza porțile logice sintetizate.
4. Se va simula comportamentul porților logice fundamentale.
5. Se va stabili dacă porțile logice simulate au același comporatament cu cel evidențiat în figurile 2.9-2.15.
6. Se va implementa proiectul în FPGA. Intrările se vor lega la SW-uri, iar ieșirile porților logice fundamentale se vor lega la LED-uri.

2.4.2 Exercițiul 2 - Circuit logic combinațional implementat ca sumă de produse

2.4.2.1 Cerințe de implementare

În tabelul 2.1 este evidențiat tabelul de adevăr al unui circuit logic combinațional. Funcționarea acestui circuit se poate defini astfel: ieșirea circuitului va avea valoare logică 1 atunci când la intrare numărul valorilor de 1 va fi mai mare ca numărul valorilor de 0, ieșirea

circuitului va avea valoare logică 0 atunci când la intrare numărul valorilor de 0 va fi mai mare ca numărul valorilor de 1. Acest circuit va fi implementat ca sumă de produse.

Tabelul 2.1. Tabelul de adevăr al circuitului logic combinational „Majoritatea contează”.

x1	x2	x3	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

2.4.2.2 Efectuarea exercițiului 2

1. Cu ajutorul tabelului de adevăr de mai sus se va scrie funcția logică, ca sumă de produse, pentru circuitul logic combinațional „Majoritatea contează”.
2. Funcția logică va fi scrisă în programul VHDL numit majoritatea_conteaza în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 2.1.
6. Se va implementa proiectul în FPGA. Intrările se vor lega la SW-uri, iar ieșirea se va lega la unul din LED-uri.

2.4.3 Exercițiul 3 – Circuit logic combinațional implementat ca produs de sume

2.4.3.1 Cerințe de implementare

Cu ajutorul tabelului de adevăral evidențiat în tabelul 2.1 se va scrie funcția logică ca produs de sume a circuitului logic „Majoritatea contează”.

2.4.3.2 Efectuarea exercițiului 3

1. Cu ajutorul tabelului de adevăr de mai sus se va scrie funcția logică, ca produs de sume, pentru circuitul logic combinațional „Majoritatea contează”.
2. Funcția logică va fi scrisă în programul VHDL numit majoritatea_conteaza în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 2.1.
6. Se va implementa proiectul în FPGA. Intrările se vor lega la SW-uri, iar ieșirea se va lega la unul din LED-uri.

2.4.4 Exercițiul 4 – Multiplexorul 2 la 1 pe 1 bit

2.4.4.1 Noțiuni teoretice

În figura 2.16 este reprezentat circuitul logic al multiplexorului 2 la 1 pe 1 bit. Acest circuit este unul combinațional și are două intrări de date pe 1 bit, o ieșire de date pe 1 bit și o intrare de selecție pe 1 bit. În funcție de valoarea logică a intrării de selecție multiplexorul va alege ce intrare de date va fi conectată la ieșirea de date. Pseudo-tabelul de adevăr al acestui circuit este evidențiat în tabelul 2.2.

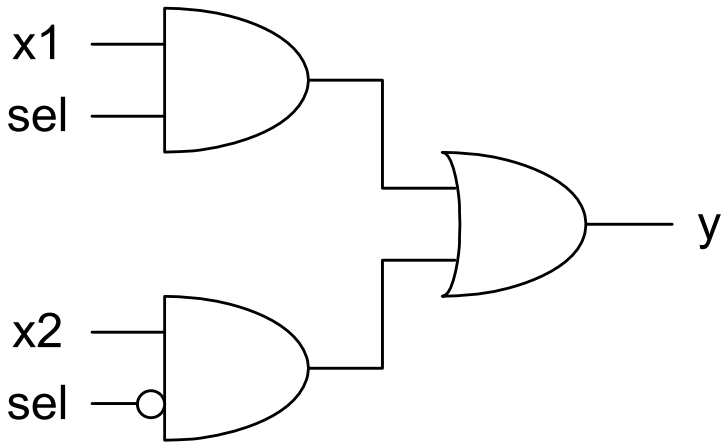


Figura 2.16. Multiplexorul 2 la 1 pe 1 bit.

Tabelul 2.2. Pseudo-tebelul de adevăr al multiplexorului 2 la 1 pe 1 bit.

Intrare de selecție	Ieșire de date
sel = 1	y = x1
sel = 0	y = x2

2.4.4.2 Efectuarea exercițiului 4

1. Cu ajutorul circuitului logic din figura 2.16 se va scrie funcția logică, ca sumă de produse, pentru multiplexorul 2 la 1 pe 1 bit.
2. Funcția logică va fi scrisă în programul VHDL numit multiplexor_2_la_1 în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat și se va compara cu cel din figura 2.16.
5. Se va simula comportamentul multiplexorului și se va compara cu cel din tabelul 2.2.
6. Se va implementa proiectul în FPGA. Intrarea de selecție se va lega la SW-uri, intrările de date se vor lega la KEY-uri, iar ieșirea se va lega la unul din LED-uri.

2.4.5 Exercițiul 5 – Multiplexorul 4 la 1 pe 1 bit

2.4.5.1 Noțiuni teoretice

În figura 2.17 este reprezentat circuitul logic al multiplexorului 4 la 1 pe 1 bit. Acest circuit este unul combinațional și are patru intrări de date pe 1 bit, o ieșire de date pe 1 bit și o intrare de selecție pe 2 bit, sau două intrări de selecție pe 1 bit. În funcție de valoarea logică a intrării de selecție multiplexorul va alege ce intrare de date va fi conectată la ieșirea de date. Multiplexorul 4 la 1 din figura 2.17 este creat cu ajutorul a trei multiplexoare 2 la 1 pe 1 bit. Pseudo-tabelul de adevăr al acestui circuit este evidențiat în tabelul 2.3.

2.4.5.2 Efectuarea exercițiului 5

1. Cu ajutorul circuitului logic din figura 2.17 se va scrie funcția logică, ca sumă de produse, pentru multiplexorul 4 la 1 pe 1 bit.
2. Funcția logică va fi scrisă în programul VHDL numit multiplexor_2_la_1 în locația indicată.

3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat și se va compara cu cel din figura 2.17.
5. Se va simula comportamentul multiplexorului și se va compara cu cel din tabelul 2.3.
6. Se va implementa proiectul în FPGA. Intrarea de selecție se va lega la SW-uri, intrările de date se vor lega la KEY-uri, iar ieșirea se va lega la unul din LED-uri.

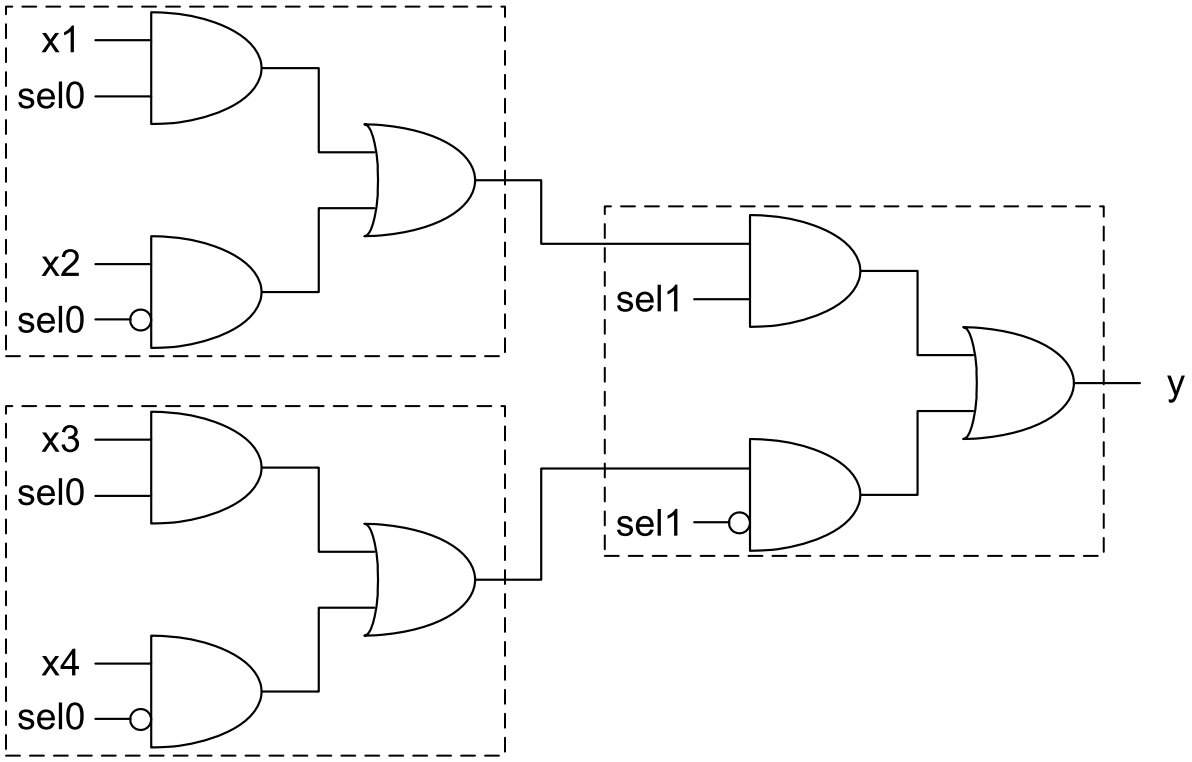


Figure 2.17. Multiplexorul 4 la 1 pe 1 bit.

Tabelul 2.3. Pseudo-tebelul de adevăr al multiplexorului 2 la 1 pe 1 bit.

Intrare de selecție	Ieșire de date
sel0 = 0, sel1 = 0	y = x4
sel0 = 0, sel1 = 1	y = x2
sel0 = 1, sel1 = 0	y = x3
sel0 = 1, sel1 = 1	y = x1

2.4.6 Exercițiul 6 – Operatorii relaționali

2.4.6.1 Noțiuni teoretice

Avem următorii operatori relaționali: egal cu (=), diferite de (/=), mai mare ca (>), mai mare sau egal ca (>=), mai mic ca (<), mai mic sau egal ca (<=). Funcționarea operatorilor egal cu, mai mare ca și mai mare sau egal ca este evidențiată în tabelul 2.4 pentru intrări de date pe 2 biți. Ieșirea fiecărui operator este pe 1 bit. Restul operatorilor se obțin negând cei trei operatori din tabelul de mai jos.

Tabelul 2.4 – Tabel de adevăr ce evidențiază funcționarea operatorilor =, >, >= pentru intrări de date pe 2 biți.

x11	x12	x21	x22	y1 (=)	y2 (>)	y3 (>=)
0	0	0	0	1	0	1
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	1	0	1
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	1	1
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	1
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	0	1

2.4.6.2 Efectuarea exercițiului 6

1. Cu ajutorul tabelului de adevăr de mai sus se vor scrie funcțiile logice, ca sumă de produse, pentru operatorii =, > și >=.

2. Funcțiile logice vor fi scrise în programul VHDL numit operatori_relationali în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul operatorilor relaționali şi se va compara cu cel din tabelul 2.4.
6. Se va implementa proiectul în FPGA. Intrările de date se vor lega la SW-uri, iar ieșirile se vor lega la LED-uri.

ATENȚIE!!!

Un bus sau o magistrală este un grup de fire de legătură ce cuprinde doi sau mai mulți biți. În figura 2.18 x1, x2 și y au în componență câte n fire de legătură. În concluzie x1, x2 și y sunt bus-uri pe n biți. În aceeași figură x3 este un fir ce cuprinde un singur bit.

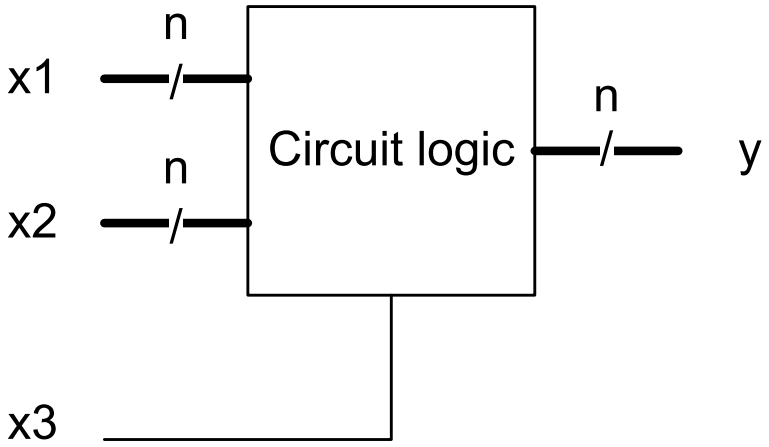


Figura 2.18. Evidențierea conceptului de bus. x1, x2 şi y sunt bus-uri pe n biți, iar x3 este un fir de legătură pe 1 bit.

2.4.7 Exercițiul 7 – Circuit logic combinațional de complexitate redusă

2.4.7.1 Cerințe de implementare

Avem circuitul logic din figura 2.19. Intrările de date x1 și x2 sunt pe 1 bit, iar ieșirea de date y este pe 1 bit. Circuitul este alcătuit dintr-un multiplexor și un circuit logic ce implementează operatorul relațional /= (diferit de). Intrările de date ale operatorului relațional sunt aceleași cu intrările de date ale multiplexorului.

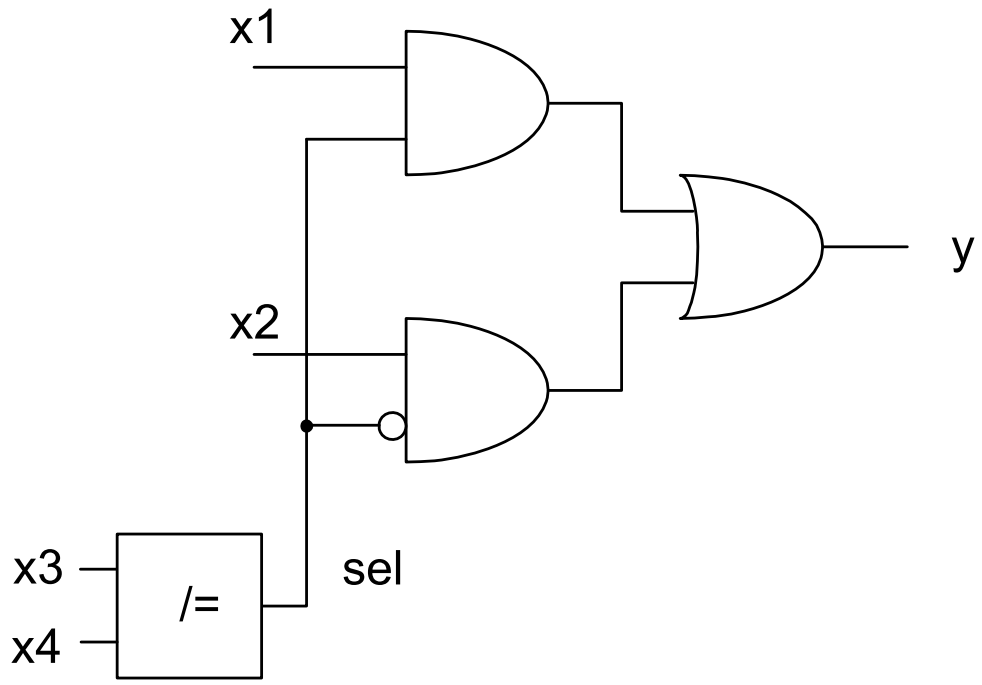


Figura 2.19. Circuit logic combinational de complexitate redusă.

2.4.7.2 Efectuarea exercițiului 7

1. Să se scrie tabelul de adevăr al operatorului /= pentru intrări de date pe 1 bit.
2. Ce poartă logică fundamentală implementează tabelul de adevăr de la punctul 1?
3. Cu ajutorul porții logice descoperite la punctul 2 și a funcției logice a multiplexorului 2 la 1 pe 1 bit să se scrie funcția logică a întregului circuit logic în programul VHDL numit circuit_logic_simplu în locația indicată..
4. Se va compila proiectul.
5. Se va vizualiza circuitul logic combinațional sintetizat.
6. Se va simula comportamentul circuitului logic combinațional.
7. Se va implementa proiectul în FPGA. Intrările de date se vor lega la SW-uri, iar ieșirea sa va lega la LED-uri.
8. Ce tabel de adevăr implementează circuitul din figura 2.19 ?

Laboratorul 3

Implementarea circuitelor logice combinaționale fundamentale

3.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea metodei de minimizare Karnaugh.

La finalul acestui laborator studentul va putea găsi funcțiile logice minime ale unui circuit logic combinațional definit printr-un tabel de adevăr cu un număr aleator de intrări și ieșiri. Va putea determina funcții logice minime utilizând eficient valorile don't care. Va putea descrie comportamentul unui multiplexor, codificator, decodificator sau a unui convertor.

3.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

3.3 Desfășurarea lucrării

3.3.1 Exercițiul 1 – Multiplexorul 2 la 1 pe 1 bit

3.3.1.1 Noțiuni teoretice

Multiplexorul 2 la 1 are două intrări de date și o ieșire de date. Acest circuit logic combinațional mai are o intrare de selecție de $\log_2(\text{numărul\_intrărilor})$ biți. Simbolul multiplexorului 2 la 1 este evidențiat în figura 3.1. Tabelul de adevăr al multiplexorului 2 la 1 pe 1 bit este reprezentat în tabelul 3.1.

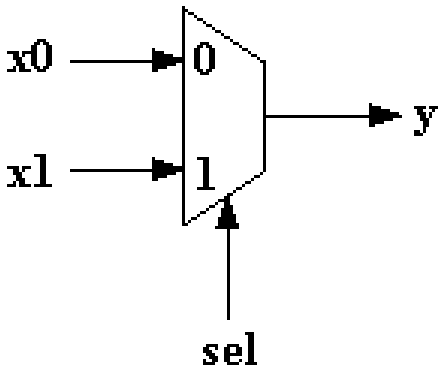


Figura 3.1. Simbolul multiplexorului 2 la 1.

Tabelul 3.1. Tabelul de adevăr al multiplexorului 2 la 1.

sel	x0	x1	y
0	**0**	0	**0**
0	**0**	1	**0**
0	**1**	0	**1**
0	**1**	1	**1**
1	0	**0**	**0**
1	0	**1**	**1**
1	1	**0**	**0**
1	1	**1**	**1**

3.3.1.2 Efectuarea exercițiului 1

1. Se va minimiza ecuația booleană reprezentată tabelul 3.1 şi se va scrie funcția logică ca sumă de produse.
2. Ecuația logică booleană minimizată se va scrie în programul VHDL numit multiplexor_2_la_1_1, în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia şi se va compara cu cel din tabelul 3.1.

ATENȚIE!!!

În fişierul multiplexor_2_la_1_1.vhd pe lângă multiplexorul 2 la 1 pe 1 bit, care este un circuitul logic combinațional, există un alt circuit numit numărător sau contor, acesta fiind

un circuit logic secvențial sincron (un circuit cu memorie). Rapiditatea funcționării numărătorului este determinată de viteza impulsurilor de tact (frecvența ceasului intern). Despre aceste circuite secvențiale sincrone vom studia în săptămânile următoare.

3.3.2 Exercițiul 2 – Multiplexorul 2 la 1 pe 4 biți

3.3.2.1 Noțiuni teoretice

Multiplexorul 2 la 1 pe 4 biți este alcătuit din patru multiplexoare 2 la 1 pe 1 bit. Organizarea multiplexorului 2 la 1 pe 4 biți este evidențiată în figura 3.2.

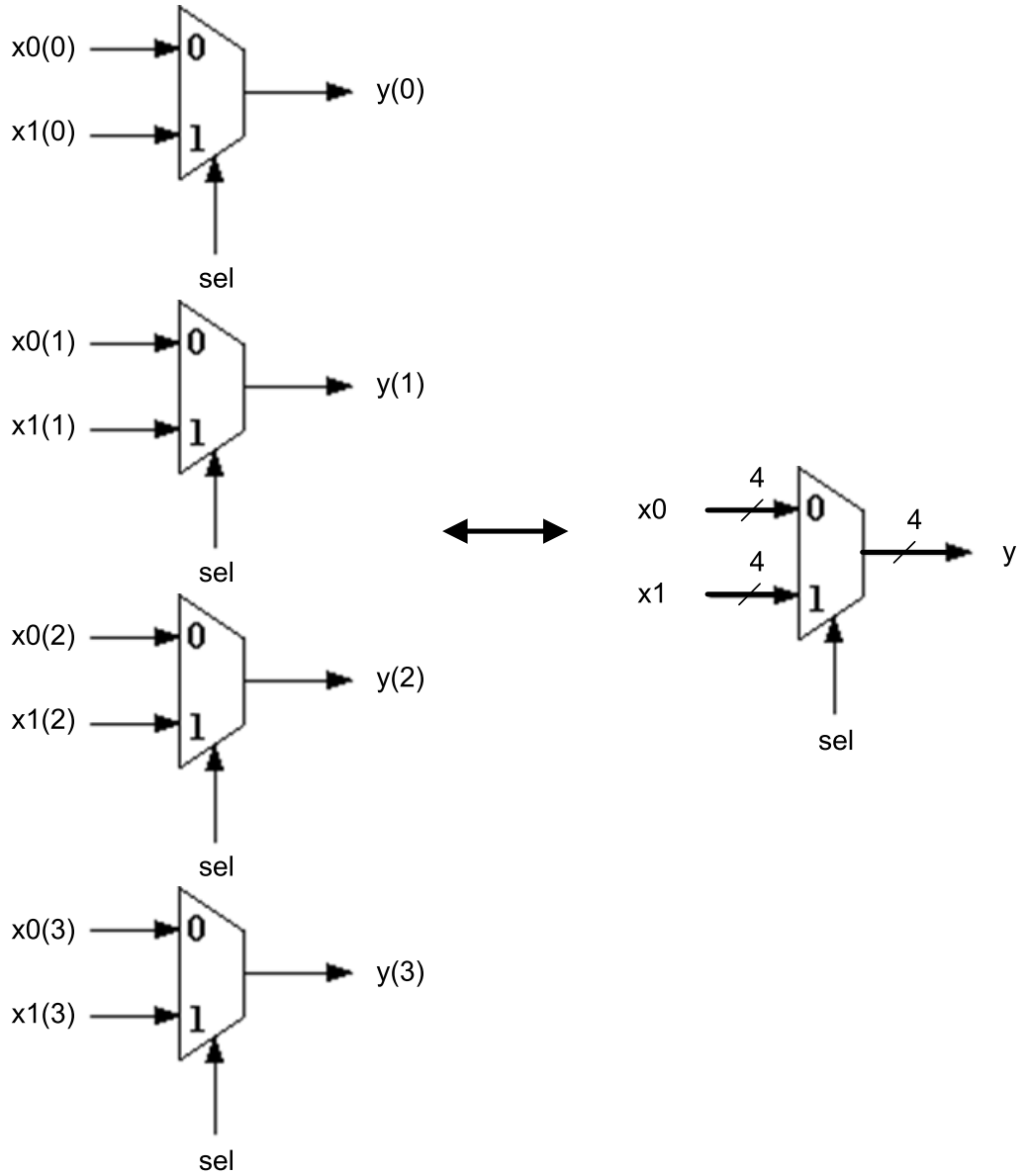


Figura 3.2. Multiplexorul 2 la 1 pe 4 biți.

3.3.2.2 Efectuarea exercițiului 2

1. Se vor scrie ecuațiile logice ale multiplexorului 2 la 1 pe 4 biți în programul VHDL numit multiplexor_2_la_1_4, în locația indicată.
2. Se va compila proiectul.
3. Se va vizualiza circuitul logic combinațional sintetizat.
4. Se va simula comportamentul acestui circuit logic combinațional.

3.3.3 Exercițiul 3 - Decodificatorul 2 la 4

3.3.3.1 Noțiuni teoretice

Decodificatorul 2 la 4 are două intrări de date și 4 ieșiri de date. Acest circuit logic combinațional modifică intrarea de date în felul următor: în funcție de valoarea intrării de date, la un moment dat, un singur bit din semnalul de ieșire va avea valoarea logică 1, restul biților vor avea valoarea logică 0. Locația bitului de valoarea logică 1 va corespunde cu valoarea zecimală a intrării de date. Tabelul de adevăr al decodificatorului 2 la 4 este reprezentat în tabelul 3.3. Simbolul acestui circuit este evidențiat în figura 3.3.

Tabelul 3.3. Tabelul de adevăr al decodificatorului 2 la 4.

Indexul bitului de valoare logică 1 în semnalul de ieșire	x1	x0	y0	y1	y2	y3
0	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
2	1	0	0	0	**1**	0
3	1	1	0	0	0	**1**

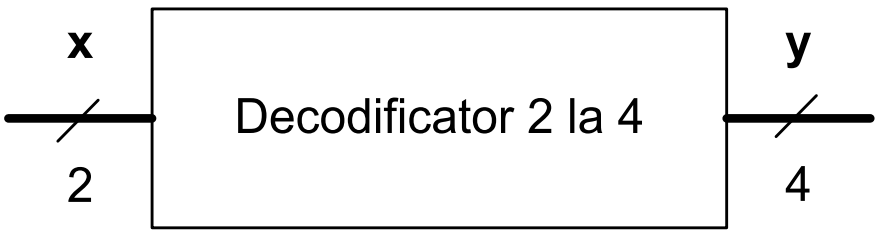


Figura 3.3. Decodificatorului 2 la 4.

3.3.3.2 Efectuarea exercițiului 3

1. Se vor minimiza ecuațiile booleene reprezentate de tabelul 3.3 şi se vor scrie funcțiile logice ca sumă de produse.

ATENȚIE!!!

Vom avea 4 ecuații booleene, una pentru fiecare ieşire. Toate ieşirile depind de cele 2 intrări.

2. Ecuațiile logice booleene minimizate se vor scrie în programul VHDL numit decodificator_2_la_4, în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia şi se va compara cu cel din tabelul 3.3.

ATENȚIE!!!

În fişierul decodificator_2_la_4.vhd pe lângă decodificatorul 2 la 4, care este un circuitul logic combinațional, există un alt circuit numit numărător sau contor, acesta fiind un circuit logic secvențial sincron (un circuit cu memorie). Funcționarea numărătorului este determinată de viteza impulsurilor de tact (frecvența ceasului intern). Despre aceste circuite secvențiale sincrone vom studia în săptămânile următoare.

3.3.4 Exercițiul 4 - Decodificatorul 3 la 8

3.3.4.1 Noțiuni teoretice

Decodificatorul 3 la 8 are trei intrări de date şi opt ieşiri de date. Acest circuit logic combinațional modifică intrarea de date în felul următor: în funcție de valoarea intrării de date, la un moment dat, un singur bit din semnalul de ieşire va avea valoarea logică 1, restul biților vor avea valoarea logică 0. Locația bitului de valoarea logică 1 va corespunde cu valoarea zecimală a intrării de date. Tabelul de adevăr al decodificatorului 3 la 8 este reprezentat în tabelul 3.4. Simbolul acestui circuit este evidențiat în figura 3.4.

Tabelul 3.4. Tabelul de adevăr al decodificatorului 3 la 8.

Indexul bitului de valoare logică 1 în semnalul de ieșire	x2	x1	x0	y0	y1	y2	y3	y4	y5	y6	y7
0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	0	1	0	**1**	0	0	0	0	0	0
2	0	1	0	0	0	**1**	0	0	0	0	0
3	0	1	1	0	0	0	**1**	0	0	0	0
4	1	0	0	0	0	0	0	**1**	0	0	0
5	1	0	1	0	0	0	0	0	**1**	0	0
6	1	1	0	0	0	0	0	0	0	**1**	0
7	1	1	1	0	0	0	0	0	0	0	**1**

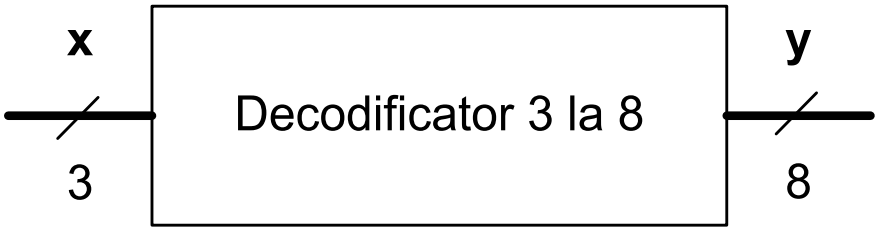


Figura 3.4. Decodificatorului 3 la 8.

3.3.4.2 Efectuarea exercițiului 4

1. Se vor minimiza ecuațiile booleene reprezentate de tabelul 3.4 și se vor scrie funcțiile logice ca sumă de produse.

ATENȚIE!!!

Vom avea 8 ecuații booleene, una pentru fiecare ieșire. Toate ieșirile depind de cele 3 intrări.

2. Ecuațiile logice booleene minimizate se vor scrie în programul VHDL numit decodificator_3_la_8, în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 3.4.

ATENȚIE!!!

În fișierul decodificator_3_la_8.vhd pe lângă decodificatorul 3 la 8, care este un circuitul logic combinațional, există un alt circuit numit numărător sau contor, acesta fiind un circuit logic secvențial sincron (un circuit cu memorie). Funcționarea numărătorului este determinată de viteza impulsurilor de tact (frecvența ceasului intern). Despre aceste circuite secvențiale sincrone vom studia în săptămânile următoare.

3.3.5 Exercițiul 5 - Codificatorul 4 la 2

3.3.5.1 Noțiuni teoretice

Codificatorul 4 la 2 are patru intrări de date și două ieșiri de date. Acest circuit logic combinațional modifică intrarea de date în felul următor: indexul ultimului bit de la intrare de valoare logică 1 va fi evidențiat la ieșire. Tabelul de adevăr al codificatorului 4 la 2 este reprezentat în tabelul 3.5. Simbolul acestui circuit este evidențiat în figura 3.5.

ATENȚIE!!!

– reprezintă valoarea nu mă interesează.

Tabelul 3.5. Tabelul de adevăr al codificatorului 4 la 2.

x3	x2	x1	x0	y1	y0	Indexul bitului de valoarea logică 1 în semnalul de ieșire
0	0	0	0	0	0	0
0	0	0	**1**	0	0	0
0	0	**1**	-	0	1	1
0	**1**	-	-	1	0	2
1	-	-	-	1	1	3

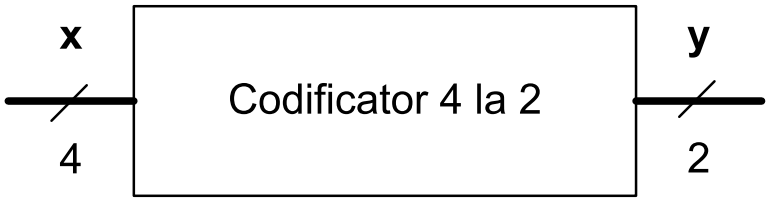


Figura 3.5. Codificatorul 4 la 2.

3.3.5.2 Efectuarea exercițiului 5

1. Se vor minimiza ecuațiile booleene reprezentate de tabelul 3.5 ca sumă de produse.

ATENȚIE!!!

Vom avea 2 ecuații booleene, una pentru fiecare ieșire, toate ieșirile depind de cele 4 intrări.

2. Ecuațiile logice booleene minimizate se vor scrie în programul VHDL numit codificator_2_la_4, în locația indicată.
3. Se va compila proiectul.
4. Se va vizualiza circuitul logic combinațional sintetizat.
5. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 3.5.

ATENȚIE!!!

În fișierul codificator_4_la_2.vhd pe lângă codificatorul 4 la 2, care este un circuitul logic combinațional, există un alt circuit numit numărător sau contor, acesta fiind un circuit logic secvențial sincron (un circuit cu memorie). Funcționarea numărătorului este determinată de viteza impulsurilor de tact (frecvența ceasului intern). Despre aceste circuite secvențiale sincrone vom studia în săptămânile următoare.

3.3.6 Exercițiul 6 – Convertor 4 la 7 pentru numere hexazecimale

3.3.6.1 Cerințe de proiectare

Se va implementa convertorul 4 la 7 evidențiat în figura 3.6 a cărui funționare este reprezentată în tabelul de adevăr din tabelul 3.6. Acest convertor va modifica intrarea de date în așa fel încât pe afișor să apară numerele 0, 1, 2, ... , 9, A, B, C, D, E, F (numere hexazecimale). Cu alte cuvinte dacă la intrarea în convertor avem 0000 pe afișor va apărea numărul 0, dacă la intrarea în convertor avem 1111 pe afișor va apărea litera E adică numărul 15.

3.3.6.2 Efectuarea exercițiului 6

1. Se vor minimiza toate ieșirile convertorului (y0, y1, ... , y6).
2. Se va deschide proiectul bin_to_hex.
3. Se vor scrie funcțiile logice minime în zona delimitată cu verde.

4. Se va implementa circuitul logic creat.
5. Se va compila proiectul.
6. Se va vizualiza circuitul logic combinațional sintetizat.
7. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 3.6.
8. Se va implementa și testa circuitul logic. Se va observa dacă pe afișor sunt reprezentate corect numerele hexazecimale, ca în figura 3.7.

Tabelul 3.6. Convertor 4 la 7 pentru numere hexazecimale.

x1	x2	x3	x4	y0	y1	y2	y3	y4	y5	y6
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	0	0	0	0	0	0	1
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

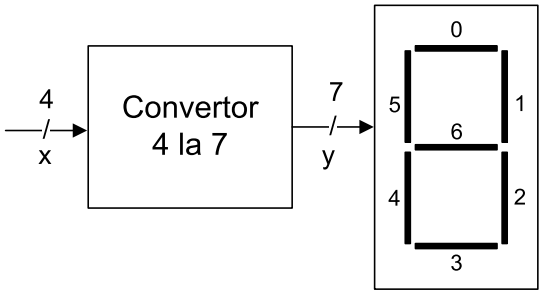


Figure 3.6. Convertor 4 la 7 pentru un afișor cu 7 segmente pentru numere hexazecimale.

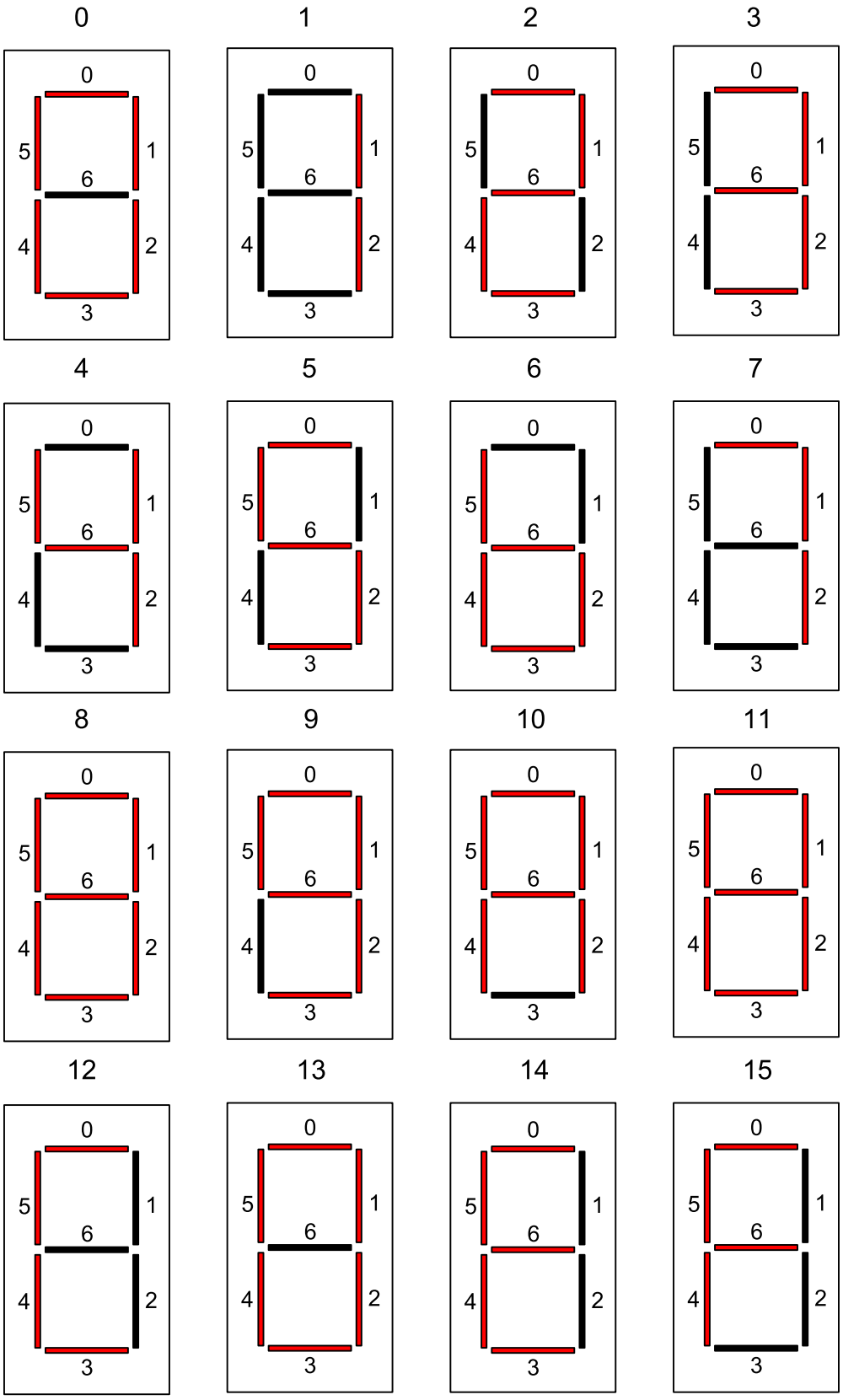


Figure 3.7. Aprinderea segmentelor afișorului pentru numere hexazecimale.

3.3.7 Exercițiul 7 – Convertor 4 la 7 pentru numere zecimale

3.3.7.1 Cerințe de proiectare

Se va implementa convertorul 4 la 7 evidențiat în figura 3.8 a cărui funționare este reprezentată în tabelul de adevăr din tabelul 3.7. Acest convertor va modifica intrarea de date în așa fel încât pe afișor să apară numerele 0, 1, 2, ... , 9 (numere zecimale). Cu alte cuvinte dacă la intrarea în convertor avem 0000 pe afișor va apărea numărul 0, dacă la intrarea în convertor avem 1001 pe afișor va apărea numărul 9. Pe intrarea de date nu vom avea valorile 1010, 1011, 1100, 1101, 1110, 1111, în concluzie la ieșire vom avea valori don't care.

3.3.7.2 Efectuarea exercițiului 7

1. Se vor minimiza toate ieșirile convertorului (y0, y1, ... , y6).
2. Se va deschide proiectul bin_to_dec.
3. Se vor scrie funcțiile logice minime în zona delimitată cu verde.
4. Se va implementa circuitul logic creat.
5. Se va compila proiectul.
6. Se va vizualiza circuitul logic combinațional sintetizat.
7. Se va simula comportamentul acestuia și se va compara cu cel din tabelul 3.7.
8. Se va implementa şi testa circuitul logic. Se va observa dacă pe afișor sunt reprezentate corect numerele hexazecimale, ca în figura 3.9.

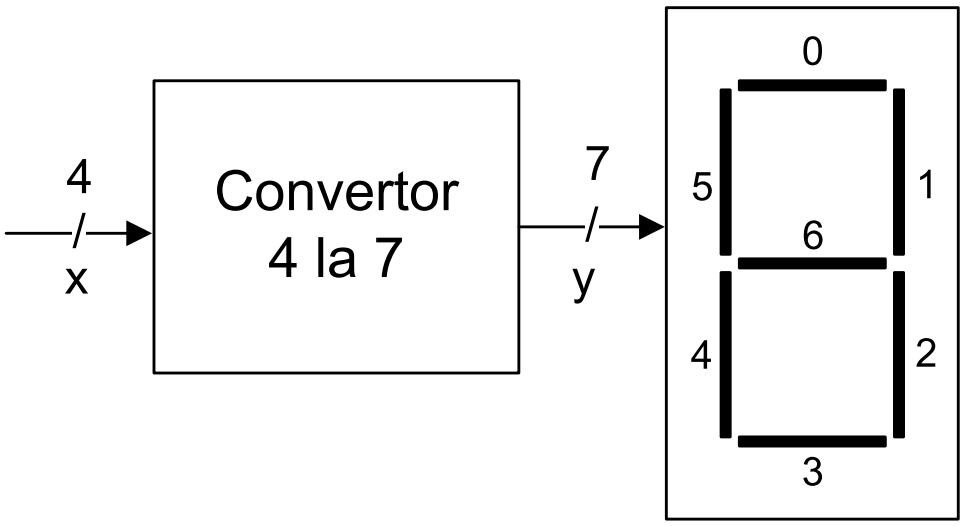


Figure 3.8. Convertor 4 la 7 pentru un afişor cu 7 segmente pentru numere zecimale.

Tabelul 3.7. Convertor 4 la 7 pentru numere zecimale.

x1	x2	x3	x4	y0	y1	y2	y3	y4	y5	y6
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	-	-	-	-	-	-	-
1	0	1	0	-	-	-	-	-	-	-
1	0	1	1	-	-	-	-	-	-	-
1	1	0	0	-	-	-	-	-	-	-
1	1	0	1	-	-	-	-	-	-	-
1	1	1	0	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-

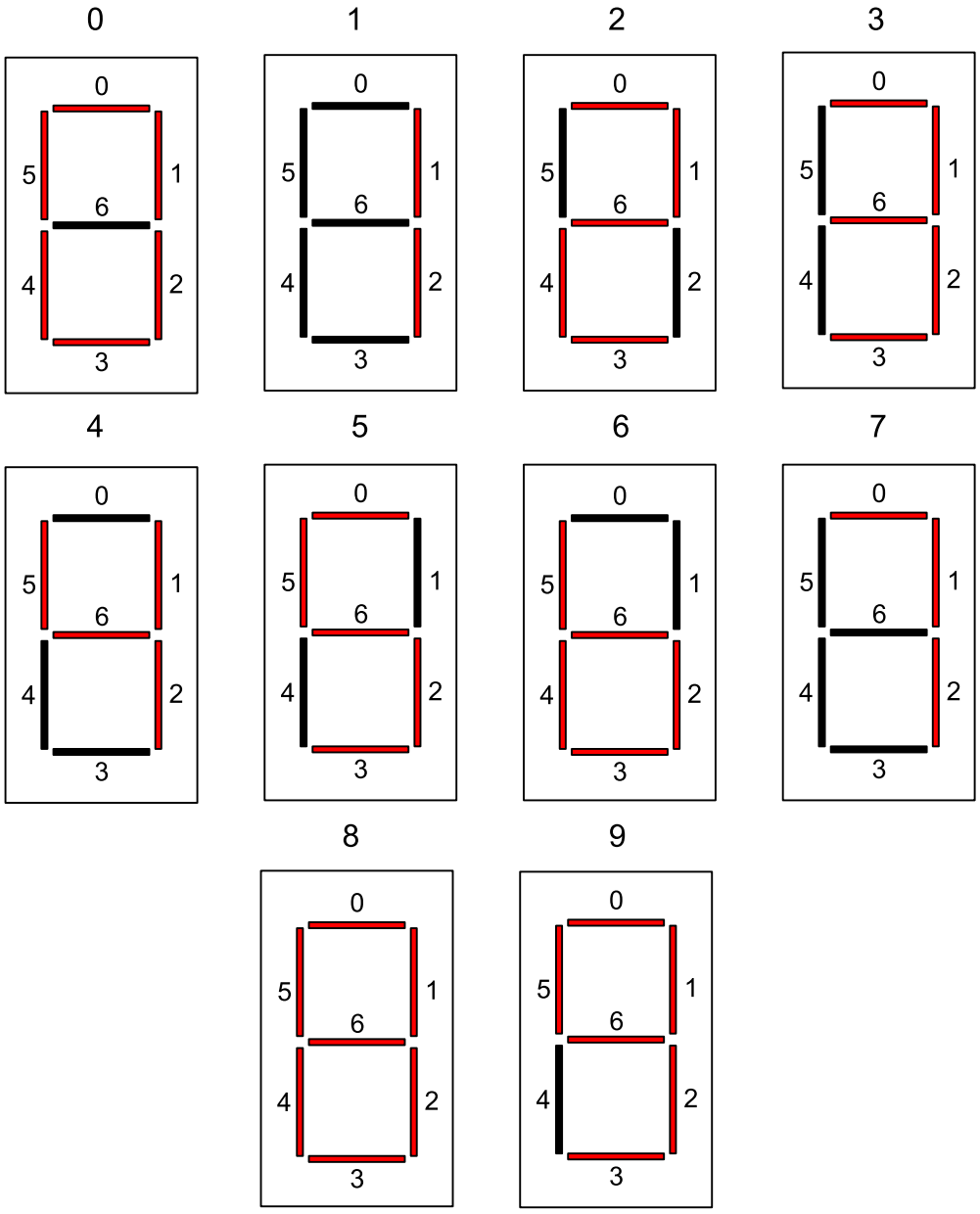


Figure 3.9. Aprinderea segmentelor afişorului pentru numere zecimale.

Laboratorul 4

Implementarea circuitelor logice secvențiale sincrone simple

4.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea comportamentului circuitelor logice secvențial sincron simple.

La finalul acestui laborator studentul va putea identifica intrările specifice ale unui circuit logic secvențial sincron. Va putea identifica un flip-flop de tip D. Va putea crea circuite logice secvențiale sincrone, cum ar fi registrul paralel, serial, rotativ sau universal. Va putea descrie comportamentul flip-flop-ului de tip D, a registrului serial, a registrului paralel, a registrului rotativ și a registrului universal.

4.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

4.3 Desfășurarea lucrării

4.3.1 Exercițiul 1 – Perioada/Frecvența impulsului de tact

4.3.1.1 Noțiuni teoretice

Circuitele logice pot fi de două tipuri: circuite logice combinaționale și circuite logice secvențiale. Circuitele logice combinaționale sunt circuite fără memorie. Circuitele logice secvențiale sunt circuite logice cu memorie. Acestea sunt de două tipuri sincrone și asincrone. Cele sincrone înregistrează datele sub coordonarea unui impuls de tact. Cele asincrone înregistrează datele fără coordonarea unui impuls de tact.

ATENȚIE!!!

Circuitele secvențiale sincrone au trei intrări specifice, pe care un circuit logic combinațional nu le posedă, și anume: intrarea de tact, intrarea de reset și intrarea de enable. Funcționarea acestor intrări va fi prezentată în primele trei exerciții.

Dispozitive digitale complexe (telefon mobil, televizor LED, termostat, etc.) au în componență circuite logice combinaționale și circuite logice secvențiale sincrone. Acestea funcționează mai repede sau mai lent în funcție de perioada/frecvența impulsului de tact.

În acest exercițiu se va modifica perioada/frecvența unui circuit digital și se va observa comportamentul acestuia. Dacă perioada impulsului de tact este mare (frecvență mică) atunci circuitul funcționează lent. Dacă perioada impulsului de tact este mică (frecvență mare) atunci circuitul funcționează repede.

În figurile 4.1 și 4.2 sunt evidențiate formele de undă ale impulsului de tact pentru un circuit secvențial sincron. În figura 4.1 semnalul de tact are perioada de 20 de ns, iar în figura 4.2 semnalul de tact are perioada de 10 ns.

Figura 4.1. Semnal de tact cu perioada de 20 ns.

Figura 4.2. Semnale de tact cu perioada de 10 ns.

4.3.1.2 Efectuarea exercițiului 1

1. Se va deschide proiectul numit decodificator.
2. În fișierul VHDL, în locul delimitat cu verde, se va modifica perioada semnalului de tact.
3. Se va mări perioada și se va constata că sistemul funcționează mai lent.

4. Se va micșora perioada și se va constata că sistemul funcționează mai rapid.
5. Se va implementa circuitul logic în FPGA.

ATENȚIE!!!

Toate dispozitive digitale au în componență flip-flop-uri de tip D (elemente fundamentale de memorie). Dacă modificăm valoarea impulsului de tact, modificăm rapiditatea cu care flip-flop-urile de tip D memorează (înregistrează) datele. Flip-flop-urile de tip D memorează datele pe frontul crescător al impulsului de tact, figura 4.3.

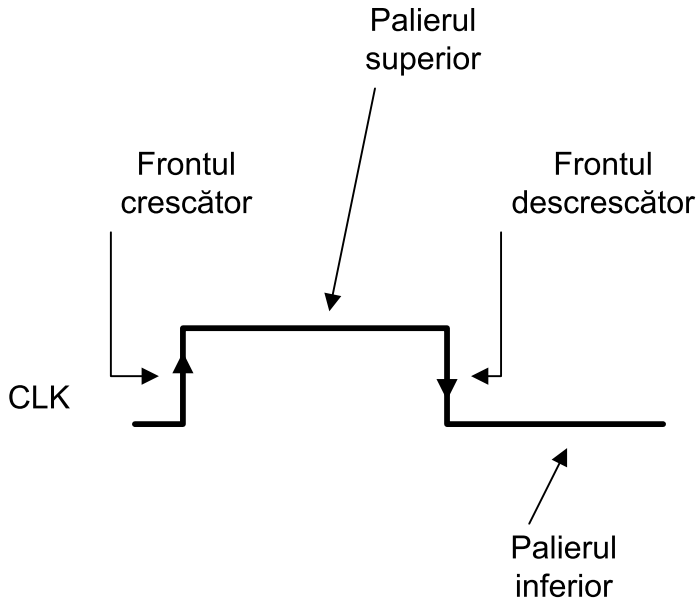


Figura 4.3. Impulsul de tact – frontul crescător, frontul descrescător, palierul superior, palierul inferior.

4.3.2 Exercițiul 2 – Resetarea unui circuit logic digital

4.3.2.1 Noțiuni teoretice

Resetarea unui circuit digital reprezintă ștergerea datelor înregistrate până în acel moment în elementele de memorie, cu alte cuvinte circuitul digital va fi adus în starea inițială.

ATENȚIE!!!

Despre circuitele logice secvențiale sincrone se afirmă că sunt circuite logice cu stări. Starea unui astfel de circuit este memorată în flip-flop-urile de tip D și se numește starea curentă sau starea prezentă. Pe baza acestei stări și în funcție de valorile intrărilor de date se

crează o stare următoare care va fi memorată în flip-flop-urile de tip D pe frontul crescător al impulsului de tact. Modul în care se modifică starea prezentă și felul în care se crează starea următoare este evidențiat în laboratoarele ce urmează.

4.3.2.2 Efectuarea exercițiului 2

1. Se va deschide proiectul numit decodificator.
2. Se va implementa circuitul logic în FPGA.
3. Se va apăsa butonul KEY(1) (KEY(1) = 1) și se va constata că sistemul digital va fi resetat, adică va fi adus în starea inițială.
4. Dacă butonul KEY(1) (KEY(1) = 0) nu va fi apăsat, circuitul va funcționa normal.

4.3.3 Exercițiul 3 – Enable-ul unui circuit logic digital.

4.3.3.1 Noțiuni teoretice

Dacă se dorește, la un moment dat, ca un sistem digital "să fie pus pe pauză", cu alte cuvinte să își memoreze starea anterioară, atunci pe intrarea de enable vom avea valoarea logică 1. Dacă pe intrarea de enable avem valoarea logică 0, circuitul va funcționa normal.

ATENȚIE!!!

Intrarea de reset are cea mai mare prioritate. Intrarea de reset este succedată ca prioritate de intrarea de enable. Intrarea de enable este succedată ca prioritate de intrarea de date.

4.3.3.2 Efectuarea exercițiului 3

1. Se va deschide proiectul numit decodificator.
2. Se va implementa circuitul logic în FPGA.
3. Se va apăsa butonul KEY(0) (KEY(0) = 1) și se va constata că sistemul digital va fi pus pe pauză.
4. Dacă butonul KEY(0) (KEY(0) = 0) va fi lăsat în poziția inițială atunci circuitul pornește din starea memorată anterior apăsării butonului KEY(0) (butonului de enable) și circuitul va funcționa normal.

4.3.4 Exercițiul 4 – Registrul paralel

4.3.4.1 Noțiuni teoretice

În figura 4.4 este reprezentat un registru paralel pe 4 biți. Acest circuit logic este alcătuit din 4 flip-flop-uri de tip D care lucrează în paralel. Dacă pe intrarea de date există valoarea logică 0011, iar impulsul de tact trece din valoarea logică 0 în valoarea logică 1 (frontul crescător), atunci datele de la intrare vor fi memorate și după un timp foarte scurt se vor găsi la ieșirea registrului paralel.

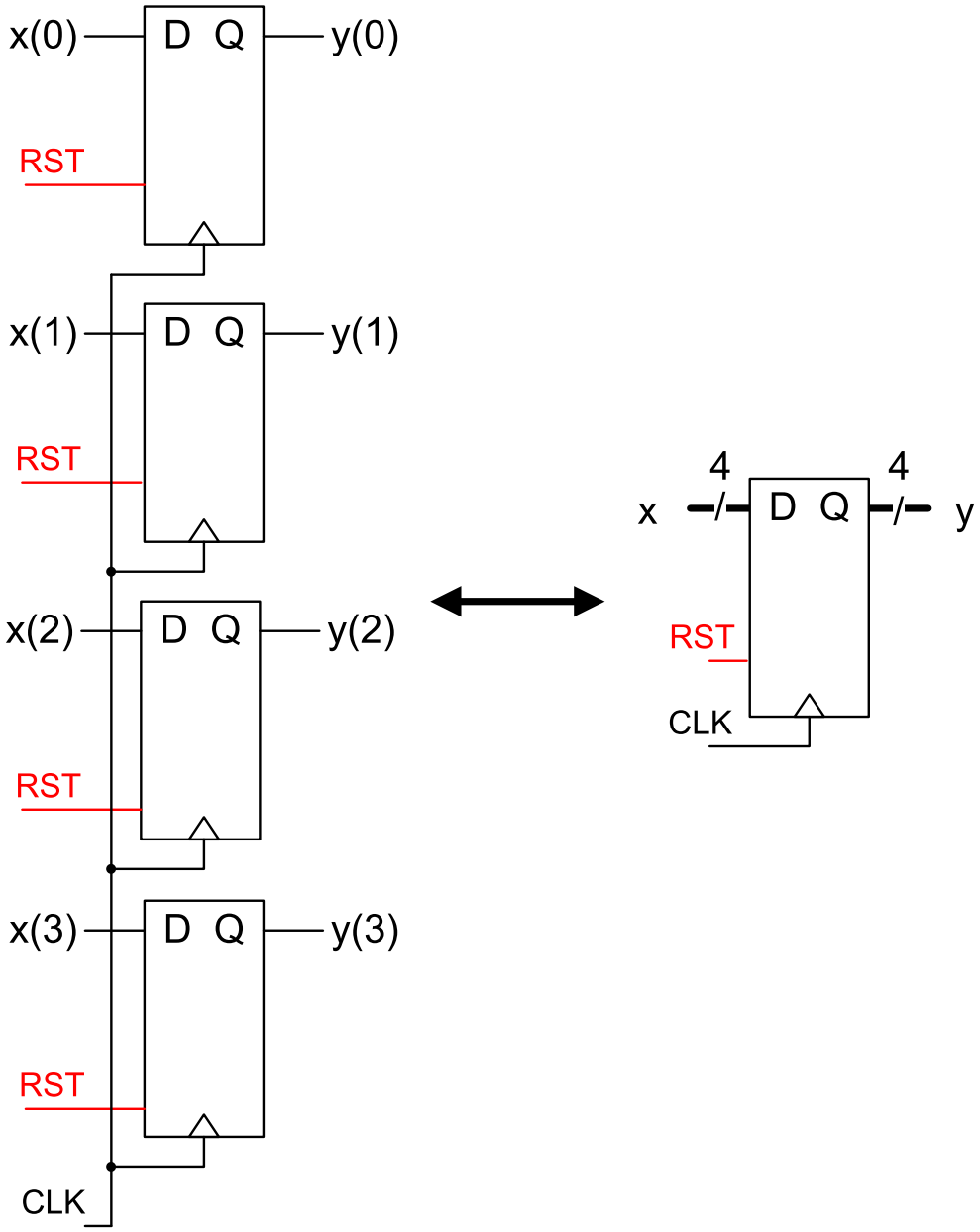


Figura 4.4. Registru paralel pe 4 biți.

ATENȚIE!!!

Intrările x(0), x(1), x(2) și x(3) corespund SW(0), SW(1), SW(2) și SW(3). Ieșirile y(0), y(1), y(2) și y(3) corespund LED(0), LED(1), LED(2) și LED(3). Impulsul de tact poate fi generat cu ajutorul intrării KEY(0). Resetul poate fi generat cu ajutorul intrării KEY(1).

4.3.4.2 Efectuarea exercițiului 4

1. Se va deschide proiectul registrul_paralel.
2. Se va compila proiectul
3. Se va vizualiza circuitul logic sintetizat.
4. Se va simula funcționarea circuitului logic
5. Se va implementa circuitul logic în FPGA.
6. Se vor acționa switch-urile, după care se va apăsa KEY(0), pentru a genera frontul crescător al impulsului de tact. Datele vor fi înregistrate în flip-flop-urile de tip D pe frontul crescător al tactului. Aceste date sunt imediat afișate la ieșire. Se vor încerca mai multe combinații ale switch-urilor.
7. La un moment dat va fi acționat butonul de reset KEY(1) și datele memorate anterior vor fi șterse din flip-flop-uri.

4.3.5 Exercițiul 5 – Registrul serial

4.3.5.1 Noțiuni teoretice

În figura 4.5 este reprezentat un registru serial pe 8 biți. Acest circuit logic este alcătuit din 8 flip-flop-uri de tip D. Ieșirea primului flip-flop reprezintă intrarea următorului flip-flop. Flip-flop-urile sunt legate în serie. Ieșirea fiecărui flip-flop este legată la un LED. Registrul serial are o intrare de date și este reprezentată de SW(0).

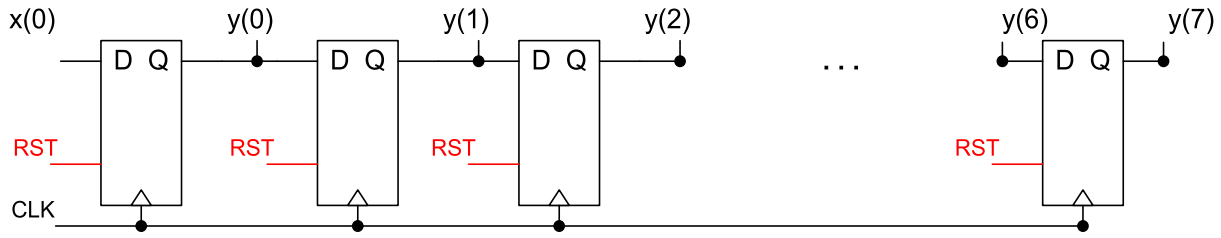


Figura 4.5. Registrul serial pe 8 biți.

ATENȚIE!!!

Intrarea de date x(0) este reprezentată de SW(0). Ieșirea fiecărui flip-flop este legată la un LED. Impulsul de tact poate fi generat cu ajutorul intrării KEY(0). Resetul poate fi generat cu ajutorul intrării KEY(1).

4.3.5.2 Efectuarea exercițiului 5

1. Se va deschide proiectul registrul_serial.
2. Se va compila proiectul
3. Se va vizualiza circuitul sintetizat.
4. Se va simula funcționarea circuitului logic.
5. Se va implementa proiectul în FPGA.
6. Vom introduce în registrul serial biți de 0 sau 1, prin acționarea switch-ului SW(0), după care se va apăsa KEY(0), pentru a genera frontul crescător al impulsului de tact. Pentru fiecare bit introdus în registrul serial conținutul registrului se va deplasa cu un bit, pentru a face loc bit-ului de la intrare. Deplasarea biților înregistrați în flip-flop-uri va putea fi observată pe LED-uri.
7. La un moment dat va fi acționat butonul de reset KEY(1) și datele memorate anterior vor fi șterse din flip-flop-uri.

4.3.6 Exercițiul 6 – Registrul rotativ pe 8 biți

4.3.6.1 Cerințe de proiectare

În figura 4.6 este reprezentat circuitul logic al unui registru rotativ pe 8 biți. Acesta este alcătuit dintr-un multiplexor 2 la 1 și un registru serial pe 8 biți.

Atunci când intrarea de selecție r (rotație) are valoarea logică 0, datele de la intrare vor putea fi memorate în registrul serial. Atunci când intrarea de selecție r are valoarea logică 1, datele se vor deplasa în cerc prin registrul serial.

4.3.6.2 Efectuarea exercițiului 6

1. Se va deschide proiectul registru_rotativ.
2. În fișierul VHDL, în locul delimitat cu verde, se va scrie funcția logică a multiplexorului. Intrările de date sunt x sau SW(0), fiind o intrare de date externă și

y7, reprezentând ieșirea registrului serial. Intrarea de selecție este r sau SW(4). Ieșirea este z și se conectează la intrarea registrului serial pe 8 biți. Intrarea de tact sau clk este determinată de KEY(0), iar intrarea de reset sau rst este determinată de KEY(1).

3. Se va compila proiectul
4. Se va vizualiza circuitul logic sintetizat.
5. Se va simula funcționarea circuitului logic
6. Se va implementa circuitul logic în FPGA.
7. Se va testa circuitul implementat și se va observa comportamentul registrului rotativ.

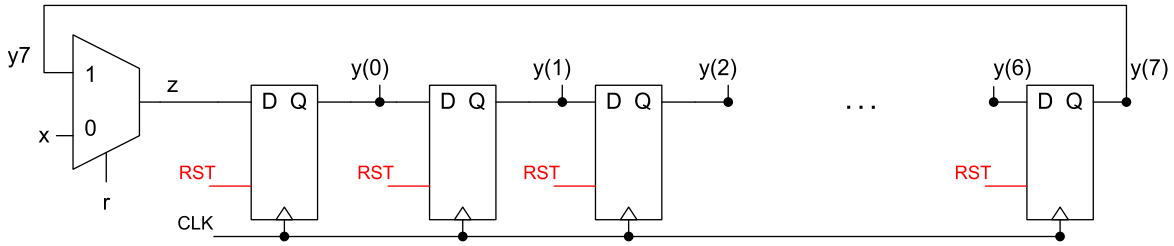


Figura 4.6. Registrul rotativ pe 8 biți.

4.3.7 Exercițiul 7 – Registrul universal pe 8 biți

4.3.7.1 Noțiuni teoretice

În figura 4.7 este reprezentat circuitul logic al unui registru universal pe 8 biți. Acesta este alcătuit din 8 multiplexoare 4 la 1 și un 8 flip-flop-uri de tip D.

În tabelul 4.1 este evidențiată funcționarea registrului universal. Atunci când intrarea de selecție, sel, are valoarea logică 00 atunci registrul universal se comportă ca un registru paralel, atunci când intrarea de selecție are valoarea logică 01 registrul universal se comporată ca un registru serial, atunci când intrarea de selecție are valoarea logică 10 registrul universal se comportă ca un registru rotativ, iar atunci când intrarea de selecție are valoarea logică 11 registrul universal își reține starea anterior.

4.3.7.2 Efectuarea exercițiului 7

1. Se va deschide proiectul registru_universal.
2. Se va compila proiectul
3. Se va vizualiza circuitul logic sintetizat.
4. Se va simula funcționarea circuitului logic.

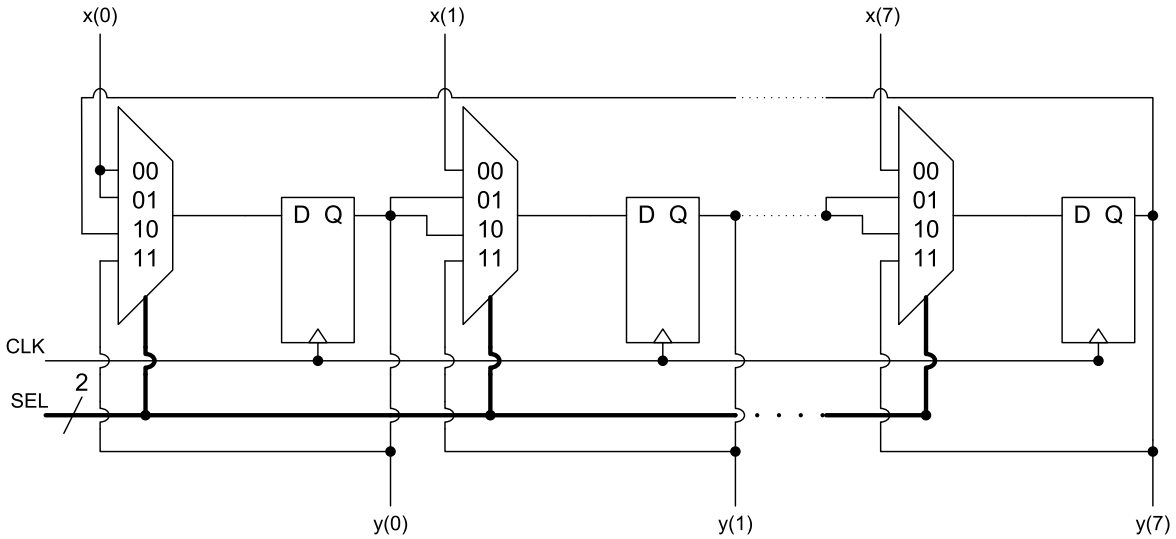


Figura 4.7. Registrul universal pe 8 biți.

Tabelul 4.1. Pseudo-tabelul de adevăr al registrului universal pe 8 biți.

Intrarea de selecție	Registrul universal
sel = 00	devine registru paralel
sel = 01	devine registru serial
sel = 10	devine registru rotativ
sel = 11	își reține starea anterioară

Figurile 4.8-4.11 evidențiază deplasarea datelor prin registrul universal.

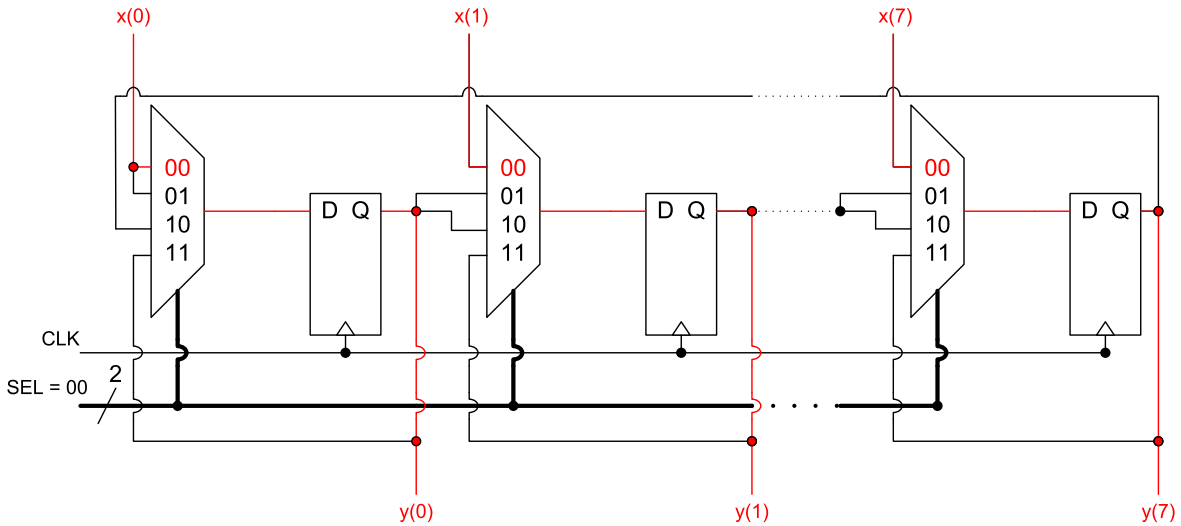


Figure 4.8. Registrul universal se comportă ca un registru paralel atunci când intrarea de selecție are valoarea logică 00.

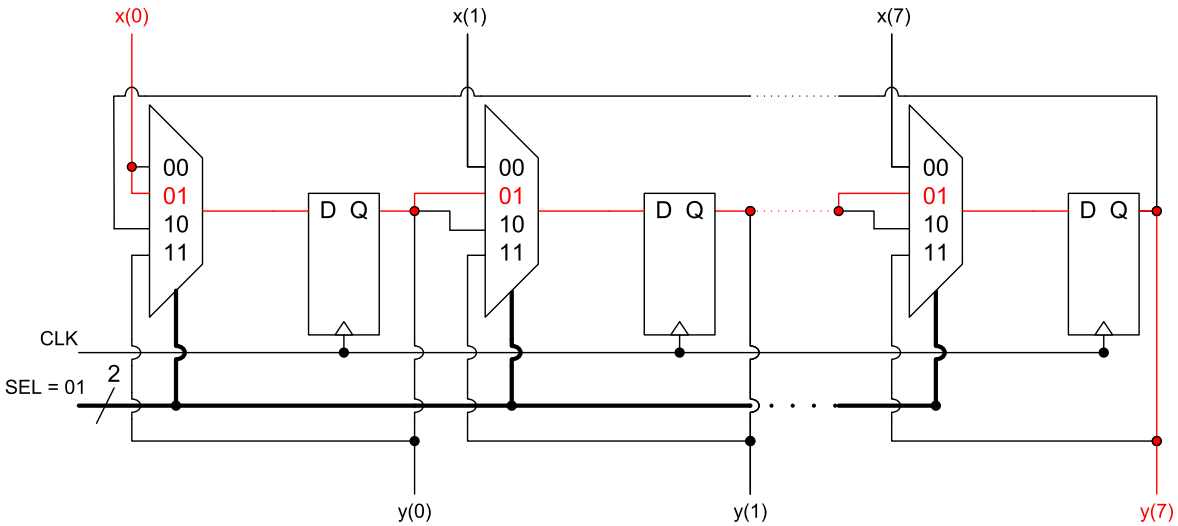


Figure 4.9. Registrul universal se comportă ca un registru serial atunci când intrarea de selecție are valoarea logică 01.

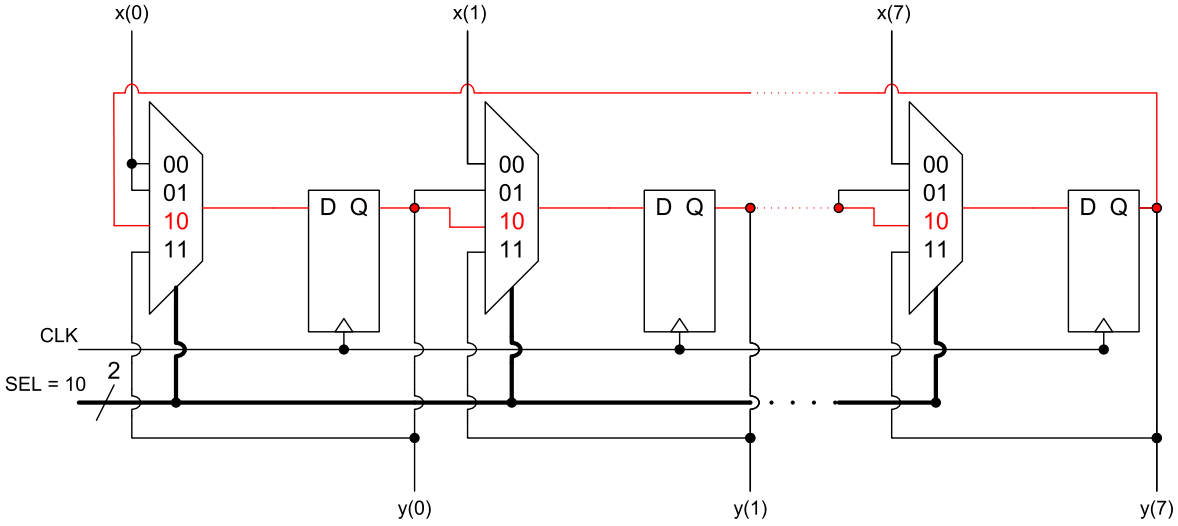


Figure 4.10. Registrul universal se comportă ca un registru rotativ atunci când intrarea de selecție are valoarea logică 10.

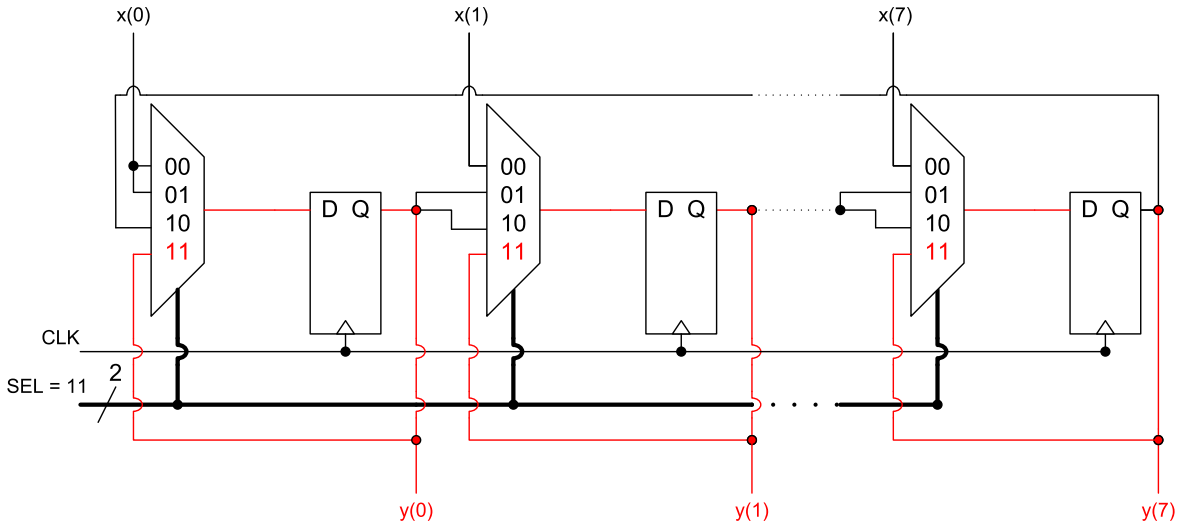


Figura 4.11. Registrul universal se comportă ca un registru rotativ atunci când intrarea de selecție are valoarea logică 11.

Laboratorul 5

Implementarea numărătoarelor

5.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea conceptelor de starea prezentă (actuală) și starea următoare. Exercițiile prezentate în acest laborator au rolul de a exemplifica modul în care se obține starea următoare din starea prezentă a unui circuit logic secvențial sincron simplu.

La finalul acestui laborator studentul va putea identifica părțile componente ale unui circuit logic secvențial sincron simplu. Va putea determina funcția logică a stării următoare cu ajutorul stării prezente pe baza tabelului de adevăr. Va putea descrie comportamentul unui numărător (contor).

5.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

5.3 Desfășurarea lucrării

5.3.1 Exercițiul 1 – Numărător pe 2 biți

5.3.1.1 Noțiuni teoretice

Un numărător este alcătuit dintr-un registru paralel (circuit logic secvențial sincron) și un incrementator (circuit logic combinațional). În figura 5.1 este reprezentat circuitul logic al unui numărător pe n biți. La ieșirea registrului paralel găsim valoarea actuală a socotelii. Observăm că această valoare actuală o găsim și la intrarea în incrementator. Incrementatorul

are rolul de a aduna o unitate la valoarea actuală a socotelii. Astfel incrementatorul calculează valoarea următoarea a socotelii, valoare ce va fi memorată în registrul paralel pe frontul crescător al impulsului de tact şi va deveni valoarea actuală a socotelii.

valoarea următoare a socotelii = valoarea actuală a socotelii + 1 (1)

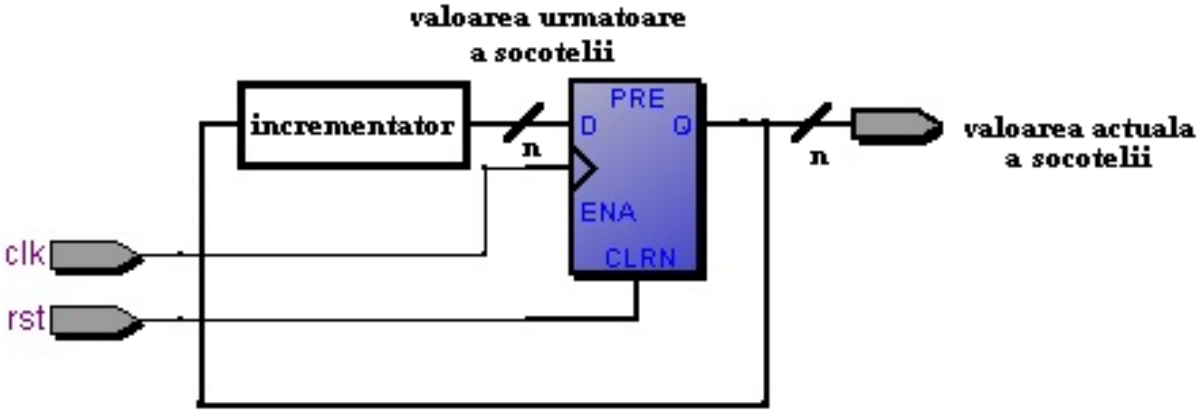


Figura 5.1. Numărător pe n biţi.

Un numărător pe 2 biţi va număra astfel 00, 01, 10, 11, 00, 01, ... Numărătorul va număra fără oprire. Cu alte cuvinte dacă numărătorul a ajuns la finalul numărătorii, acesta va continua să numere de la început. Variabilele din relaţia 1 le vom nota în felul următor: valoarea actuală a socotelii cu a, valoarea următoare a socotelii cu c, iar variabila 1 cu b.

Tabelul 5.1. Tabelul de adevăr al incrementatorului numărătorului pe 2 biţi.

a0	a1	b	c0	c1
0	0	0	-	-
0	0	1	0	1
0	1	0	-	-
0	1	1	1	0
1	0	0	-	-
1	0	1	1	1
1	1	0	-	-
1	1	1	0	0

Un numărător pe 2 biți va fi alcătuit dintr-un registru paralel pe 2 biți și un incrementator cu 3 intări (2 intrări provin de la valoarea actuală a socotelii (a0 și a1) și cea de a treia intrare provine de la variabila binară b = 1) și două ieșiri (valoarea următoare a socotelii (c0 și c1)). Incrementatorul va putea fi implementat cu ajutorul tabelului de adevăr din tabelul 5.1. În acest tabelul observăm că pentru toate combinațiile la intrare unde b = 0, la ieșire avem valori don't care. Acest lucru se explică prin faptul că variabila b nu va lua niciodată valoarea 0, de aceea nu ne interesează ce valori vom obține la ieșire.

5.3.1.2 Efectuarea exercițiului 1

1. Se va deschide proiectul numit counter_2_biti.
2. Cu ajutorul tabelului 5.1 se vor găsi ecuațiile logice minime ale incrementatorului c0 <= ?; c1 <= ?;
3. În fișierul VHDL, în locul delimitat cu verde, se vor scrie aceste ecuații minime.
4. Se va compila proiectul
5. Se va simula proiectul
6. Se va implementa circuitul logic în FPGA.
7. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

5.3.2 Exercițiul 2 – Numărător pe 3 biți

5.3.2.1 Cerințe de proiectare

Un numărător pe 3 biți va număra astfel 000, 001, 010, 011, 100, 101, 110, 111, 000, 001, 010, 011, 100 ... . Numărătorul va număra fără oprire. Cu alte cuvinte dacă numărătorul a ajuns la finalul numărătorii, acesta va continua să numere de la început. Variabilele din relația 1 le vom nota în felul următor: valoarea actuală a socotelii cu a, valoarea următoare a socotelii cu c, iar variabila 1 cu b.

Un numărător pe 3 biți va fi alcătuit dintr-un registru paralel pe 3 biți și un incrementator cu 4 intări (3 intrări provin de la valoarea actuală a socotelii (a0, a1 și a2) și cea de a patra intrare provine de la variabila binară b = 1) și 3 ieșiri (valoarea următoare a socotelii (c0, c1 și c2)). Incrementatorul va putea fi implementat cu ajutorul tabelului de adevăr din tabelul 5.2.

Tabelul 5.2. Tabelul de adevăr al incrementatorului numărătorului pe 3 biți.

c0	c1	c2	b	a0	a1	a2
0	0	0	0	-	-	-
0	0	0	1	0	0	1
0	0	1	0	-	-	-
0	0	1	1	0	1	0
0	1	0	0	-	-	-
0	1	0	1	0	1	1
0	1	1	0	-	-	-
0	1	1	1	1	0	0
1	0	0	0	-	-	-
1	0	0	1	1	0	1
1	0	1	0	-	-	-
1	0	1	1	1	1	0
1	1	0	0	-	-	-
1	1	0	1	1	1	1
1	1	1	0	-	-	-
1	1	1	1	0	0	0

În tabelul 5.2 observăm că pentru toate combinațiile la intrare unde b = 0, la ieșire avem valori don’t care. Acest lucru se explică prin faptul că variabila b nu va lua niciodată valoarea 0, de aceea nu ne interesează ce valori vom obține la ieșire.

5.3.2.2 Efectuarea exercițiului 2

1. Se va deschide proiectul numit counter_3_biti.
2. Cu ajutorul tabelului 5.2 se vor găsi ecuațiile logice minime ale incrementatorului c0 <= ?; c1 <= ?; c2 <= ?;
3. În fișierul VHDL, în locul delimitat cu verde, se vor scrie aceste ecuații minime.
4. Se va compila proiectul.
5. Se va simula proiectul.
6. Se va implementa circuitul logic în FPGA.

7. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

5.3.3 Exercițiul 3 – Numărător pe 4 biți

5.3.3.1 Cerințe de proiectare

Un numărător pe 4 biți va număra astfel 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 0000, 0001 ... . Numărătorul va număra fără oprire. În acest exercițiu se va efectua implementarea incrementatorului numărătorului cu ajutorul tabelului de adevăr din tabelul 5.3.

Tabelul 5.3. Tabelul de adevăr al incrementatorului numărătorului pe 4 biți.

Valoarea actuală a socotelii Starea prezentă				Valoarea următoare a socotelii Starea următoare			
c0	c1	c2	c3	a0	a1	a2	a3
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Acest tabel conține doar valorile posibile ale socotelii actuale, reprezentate de c0, c1, c2 și c3, și valorile posibile ale socotelii următoare, reprezentate de a0, a1, a2 și a3.

Tabelul 5.3 evidențiază funcționarea numărătorului din figura 5.1. Valoarea actuală a socotelii (starea prezentă) este memorată în registrul paralel și se găsește atât la ieșirea acestuia cât și la intrarea incrementatorului. Pe baza stării prezente incrementatorul calculează valoarea următoare a socotelii (starea următoare). Starea următoare va fi memorată în registrul paralel pe frontul crescător al impulsului de tact și va deveni starea actuală a numărătorului.

5.3.3.2 Efectuarea exercițiului 3

1. Se va deschide proiectul numit counter_4_biti.
2. Cu ajutorul tabelului 5.3 se vor găsi ecuațiile logice minime ale incrementatorului
   c0 <= ?; c1 <= ?; c2 <= ?; c3 <= ?;
3. În fișierul VHDL, în locul delimitat cu verde, se vor scrie aceste ecuații minime.
4. Se va compila proiectul.
5. Se va simula proiectul.
6. Se va implementa circuitul logic în FPGA.
7. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

5.3.4 Exercițiul 4 – Numărător pe 8 biți

5.3.4.1 Cerințe de proiectare

Un numărător pe 8 biți va fi alcătuit dintr-un registru paralel pe 8 biți și un incrementator cu 9 intări și 8 ieșiri. Incrementatorul va putea fi implementat cu ajutorul unui sumator pe 8 biți. Circuitul logic al numărătorului pe 8 biți este evidențiat în figura 5.2.

ATENȚIE!!!

Funcționarea și implementarea sumatorului va fi exemplificată la seminar. Sumatorul este un circuit logic aritmetic cu 2 intrări pe n biți și o ieșire pe n+1 biți. În cazul nostrul o intrare este reprezentată de valoarea actuală a socotelii, iar a doua intrare este reprezentată de variabila binară 1. Ieșirea va reprezenta valoarea următoare a socotelii.

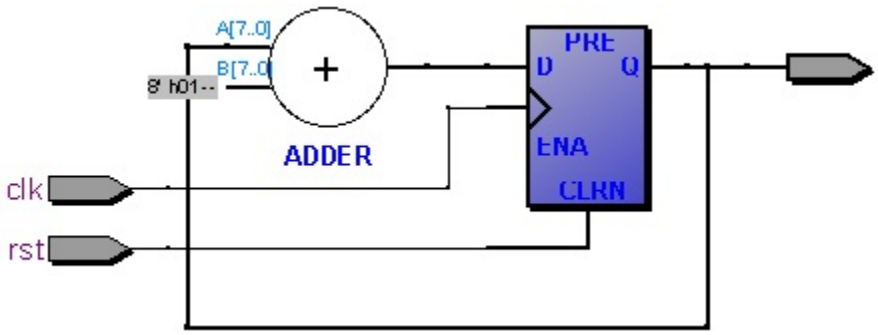


Figura 5.2. Numărător pe 8 biți.

5.3.4.2 Efectuarea exercițiului 4

1. Se va deschide proiectul numit counter_8_biti.
2. În fișierul VHDL se va scrie următoarea funcție logică în locul delimitat cu verde

   c <= a + ’1’;
3. Se va compila proiectul.
4. Se va simula proiectul.
5. Se va implementa circuitul logic în FPGA.
6. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

5.3.5 Exercițiul 5 – Numărătorul cu 2 pe 4 biți

5.3.5.1 Cerințe de proiectare

Un numărător cu 2 pe 4 biți va număra astfel 0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110, 0000, 0010 ... . Numărătorul va număra fără oprire din doi în doi începând de la 0. În acest exercițiu se va efectua implementarea incrementatorului numărătorului cu ajutorul tabelului de adevăr din tabelul 5.4.

Acest tabel conține valorile socotelii actuale, reprezentate de c0, c1, c2 și c3, și valorile socotelii următoare, reprezentate de a0, a1, a2 și a3. Datorită faptului că incrementarea se face cu 2 și numărătoarea începe de la 0, valorile impare nu vor apărea la ieșirea numărătorului, deci ele vor fi considerate valori don’t care.

Tabelul 5.4. Tabelul de adevăr al incrementatorului numărătorului cu 2 pe 4 biți.

Valoarea actuală a socotelii Starea prezentă				Valoarea următoare a socotelii Starea următoare			
c0	c1	c2	c3	a0	a1	a2	a3
0	0	0	0	0	0	1	0
0	0	0	1	-	-	-	-
0	0	1	0	0	1	0	0
0	0	1	1	-	-	-	-
0	1	0	0	0	1	1	0
0	1	0	1	-	-	-	-
0	1	1	0	1	0	0	0
0	1	1	1	-	-	-	-
1	0	0	0	1	0	1	0
1	0	0	1	-	-	-	-
1	0	1	0	1	1	0	0
1	0	1	1	-	-	-	-
1	1	0	0	1	1	1	0
1	1	0	1	-	-	-	-
1	1	1	0	0	0	0	0
1	1	1	1	-	-	-	-

5.3.5.2 Efectuarea exercițiului 5

1. Se va deschide proiectul numit counter_2_4_biti.
2. Cu ajutorul tabelului 5.4 se vor găsi ecuațiile logice minime ale incrementatorului
   c0 <= ?; c1 <= ?; c2 <= ?; c3 <= ?;
3. În fișierul VHDL, în locul delimitat cu verde, se vor scrie aceste ecuații minime.
4. Se va compila proiectul.
5. Se va simula proiectul.
6. Se va implementa circuitul logic în FPGA.
7. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

5.3.6 Exercițiul 6 – Numărător aleator pe 8 biți

5.3.6.1 Cerințe de proiectare

În figura 5.3 este reprezentat circuitul logic al unui numărător aleator pe 8 biți. Acest circuit logic este alcătuit dintr-un registru serial și o poartă xor cu 4 intrări.

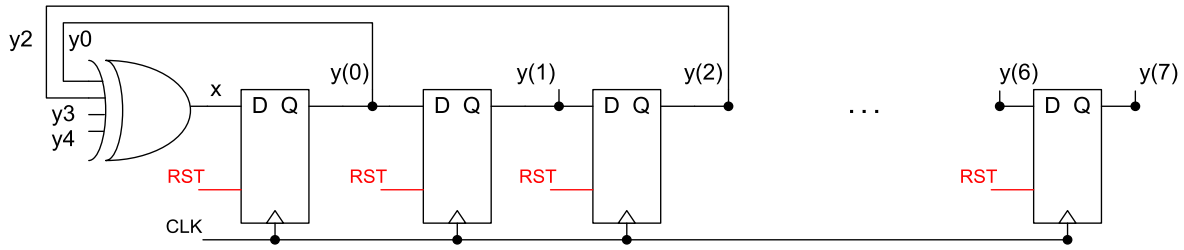


Figura 5.3. Numărător aleator pe 8 biți.

5.3.6.2 Efectuarea exercițiului 6

1. Se va deschide proiectul counter_aleator.
2. În fișierul VHDL se va scrie ecuația logică ce corespunde porții xor în locul delimitat cu verde. Biții 0, 2, 3 și 4 ai registrului serial reprezintă intrările y0, y2, y3 și y4 în poarta xor. Ieșirea porții xor reprezintă intrarea de date x a registrului serial.
3. Se va compila proiectul.
4. Se va simula proiectul.
5. Se va implementa circuitul logic în FPGA.
6. Se va testa circuitul implementat, apăsând pe butonul KEY(0) pentru a genera frontul crescător al tactului sau pe butonul KEY(1) pentru a reseta numărătorul.

Laboratorul 6

Implementarea automatelor finite

6.1 Obiective operaționale

Obiectivul principal al acestui laborator reprezintă însușirea metodologie de proiectare a mașinilor cu stări finite. Exercițiile prezentate în acest laborator au rolul de a exemplifica modul în care este utilizată o diagramă algoritmică cu stări în descrierea comportamentului unei mașini cu stări finite.

La finalul acestui laborator studentul va putea crea diagrama algortimică a unei mașini cu stări finite. Va putea identifica părțile componente ale diagramei. Va putea face diferența dintre o mașină cu stări finite de tip Moore și una de tip Mealy. Va putea identifica părțile componente ale unei mașini cu stări finite. Va putea determina modul în care Quartus II montează circuitele logice.

6.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

6.3 Desfășurarea lucrării

6.3.1 Exercițiul 1 – Diagrama algoritmică cu stări

6.3.1.1 Noțiuni teoretice

O diagramă algoritmică cu stări descrie ordinea în care se succed stările unei mașini cu stări finite (automat finit). Trecerea din starea prezentă în starea următoare în funcție de

valorile intrărilor şi modul în care se comportă ieşirile automatului finit sunt evidenţiate cu uşurinţă printr-o diagramă algoritmică cu stări.

O diagramă algoritmică este alcătuită din blocuri. Un astfel de bloc este evidenţiat în figura 6.1. Un bloc cuprinde caseta stării actuale, al cărui nume este evidenţiat în stânga sus, în interiorul căreia este descris comportamentul ieşirilor de tip Moore. Aceste ieşiri depind de starea actuală. Opţional un bloc mai poate avea o casetă de decizie şi o casetă de ieşire condiţionată. O casetă de decizie cuprinde o expresie booleană. În funcţie de valoarea logică a expresiei putem afla care va fi starea următoare a automatului finit. În interiorul casetei de ieşire condiţionată este evidenţiat comportamentul ieşirilor de tip Mealy. Aceste ieşiri depind de starea actuală şi de valorile intrărilor. O casetă de ieşire condiţionată nu poate succeda o casetă a stării actuale aşadar o casetă de ieşire condiţionată întotdeauna va succeda o casetă de decizie.

ATENŢIE!!!

Ambele tipuri de ieşiri (Moore şi Mealy) îşi păstrează valoarea în starea actuală şi îşi modifică valoarea curentă imediat ce automatul finit ajunge în starea următoare.

O diagramă algoritmică cu stări este evidenţiată în figura 6.2. Lista 6.1 evidenţiază modul în care diagrama din figura 6.2 este descrisă prin cod VHDL.

6.3.1.2 Efectuarea exerciţiului 1

1. Se va deschide proiectul asm_exemplu.
2. Se va vizualiza codul VHDL ce implementează diagrama din figura 6.2.
3. Se va observa modul în care au fost create stările automatului finit (liniile de cod delimitate de comentariile “stările maşinii cu stări finite”).
4. Se va compila proiectul.
5. Se vor vizualiza stările automatului finit astfel: Tools\Netlist Viewers\State Machine Viewer.
6. Se vor verifica expresiile booleene de trecere dintr-o stare în altă stare.

ATENŢIE!!!

În figura 6.2 expresiile din caseta stării actuale şi caseta de ieşire condiţionată reprezintă atribuiri de valori logice. Spre exemplu y0 = 0 se va scrie în cod VHDL ca y0 <= ’0’ şi înseamnă y0 va dobândi valoarea logică 0. Expresiile din caseta de decizie

reprezintă expresii booleene, așadar expresiile pot fi adevărate sau false. Spre exemplu start = 1 se va scrie în cod VHDL ca start <= ’1’.

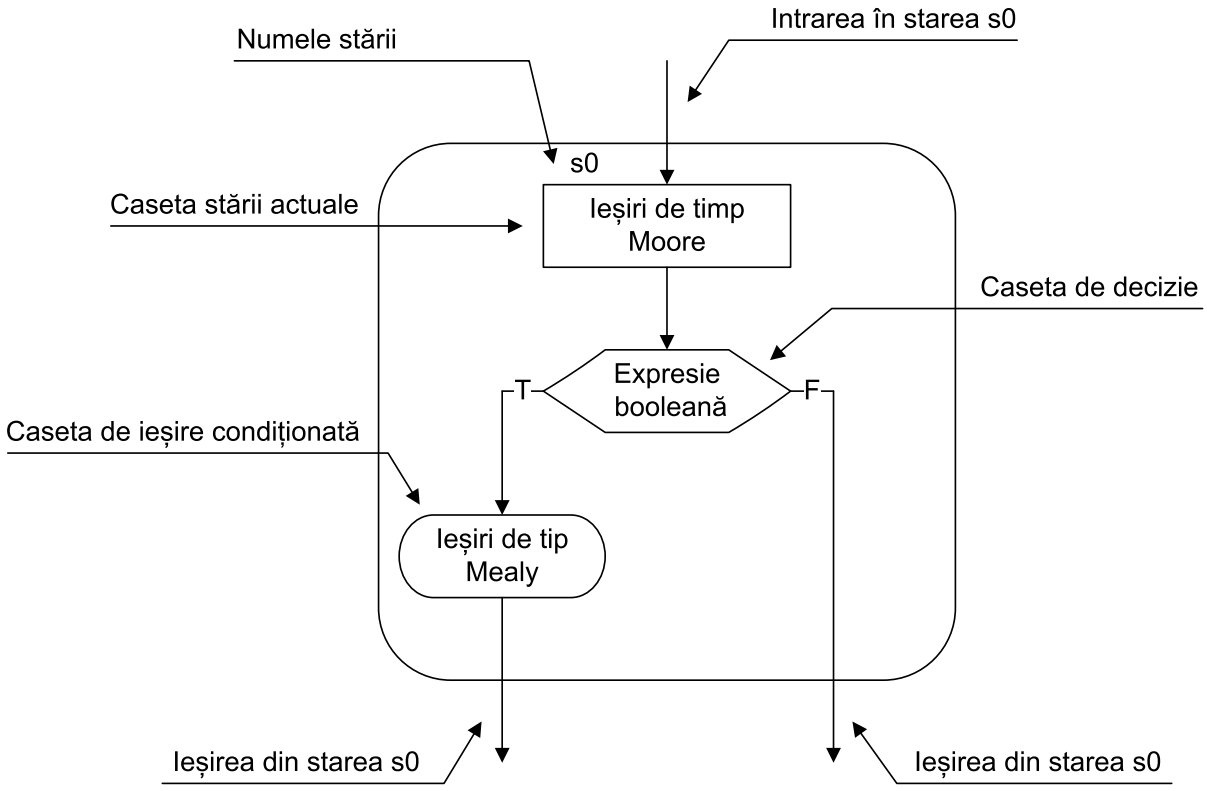


Figura 6.1. Alcătuirea unui bloc al unei diagrame algoritmice cu stări. Figura este preluată și adaptată din cartea RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability de Pong P. Chu.

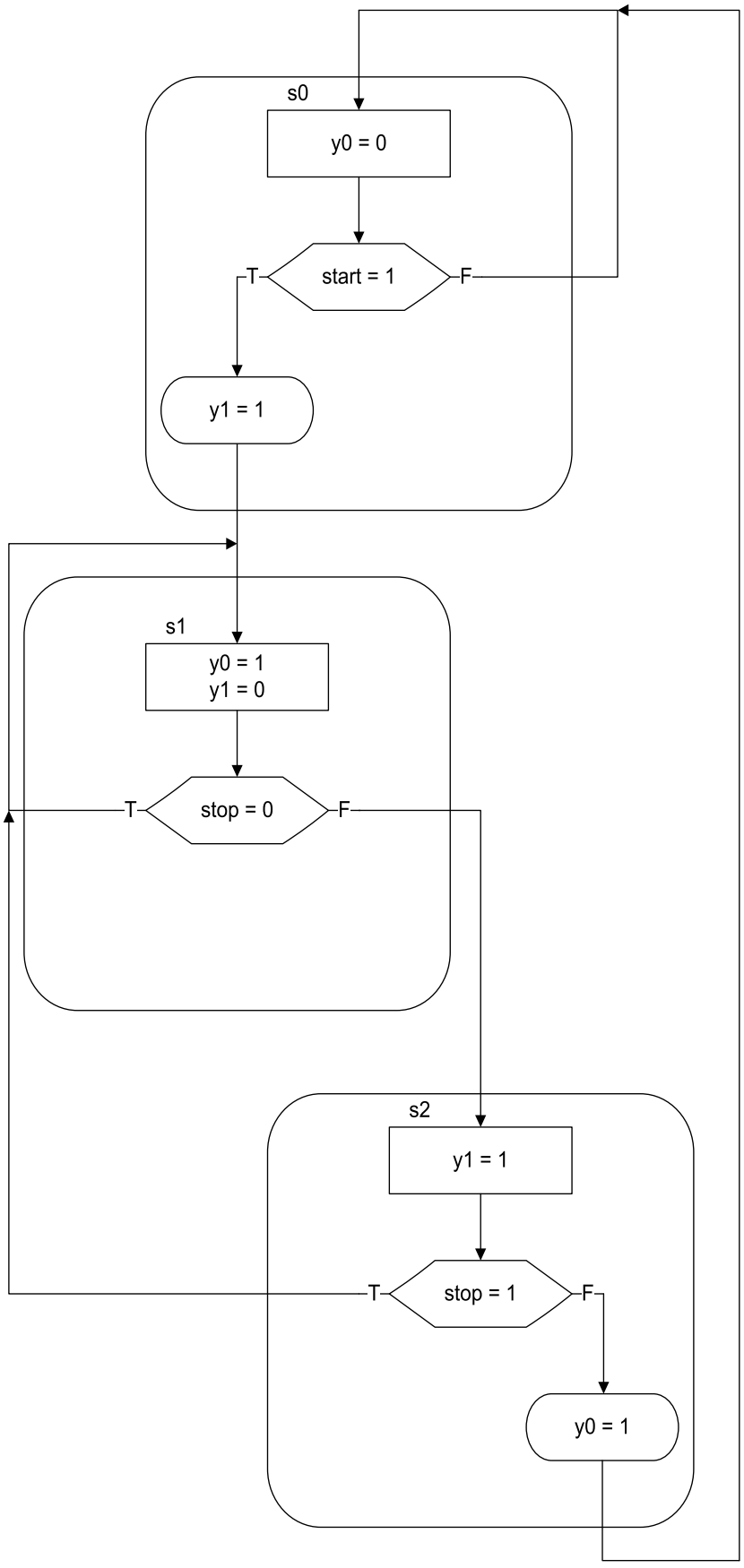


Figura 6.2. Exemplu de diagramă algoritmică cu stări.

Lista 6.1. Codul VHDL al diagramei algoritmice cu stări evidențiată în figura 6.2.

```
-- librarii----------------------------------

library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------

-- intrari-iesiri-----------------------------

entity asm_exemplu is
port (
        clk, reset      : in std_logic;
        start           : in std_logic;
        stop            : in std_logic;
        y0              : out std_logic;
        y1              : out std_logic);
end;

-----------------------------------------------

-- arhitectura circuitului digital-----------

architecture arch_asm_exemplu of asm_exemplu is

-- stările mașinii cu stări finite------------

type state_type is (s0, s1, s2);
signal starea_prezenta, starea_urmatoare : state_type;

-----------------------------------------------

begin

---------- Flip-flop-uri ----------------

process(clk, reset)
begin
        if reset = '1' then
                starea_prezenta <= s0;
        elsif (clk'event and clk = '1') then
                starea_prezenta <= starea_urmatoare;
        end if;
end process;

-------------------------------------------

------------ Starea urmatoare ----------------
```

```
process(starea_prezenta, start, stop)
begin
     case starea_prezenta is
       when s0 =>
            y0 <= '0';
            if start = '1' then
              y1 <= '1';
              starea_urmatoare <= s1;
          else
              y1 <= '0';
             starea_urmatoare <= s0;
          end if;
      when s1 =>
            y0 <= '1';
            y1 <= '0';
            if stop = '0' then
              starea_urmatoare <= s1;
            else
              starea_urmatoare <= s2;
            end if;
       when s2 =>
            y1 <= '1';
            if stop = '1' then
               y0 <= '0';
               starea_urmatoare <= s1;
            else
               y0 <= '1';
               starea_urmatoare <= s0;
            end if;
     end case;
end process;

-----------------------------------------------

end arch_asm_exemplu;
```

6.3.2 Exercițiul 2 – Mașina cu stări finite de tip Moore/de tip Mealy

6.3.2.1 Noțiuni teoretice

În figura 6.3 este reprezentată structura mașinii cu stări finite de tip Moore. În figura 6.4 este reprezentată structura mașinii cu stări finite de tip Mealy. Flip-flop-urile de tip D reprezintă elementele de memorie ale celor două circuite, modulele starea următoare și ieșire sunt circuite logice combinaționale.

Starea actuală a circuitului este reținută de flip-flop-urile de tip D. În cazul automatului Moore modulul ieşire utilizează starea actuală pentru a crea semnalele de ieşire. În cazul automatului Mealy modulul de ieşire utilizează starea actuală şi intrările pentru a crea semnalele de ieşire. În cazul ambelor autoamate starea actuală împreună cu intrarea de date sunt utilizate de către modulul starea următoare pentru a crea starea următoare. Starea următoare este memorată de flip-flop-urile de tip D şi devine starea actuală a circuitului.

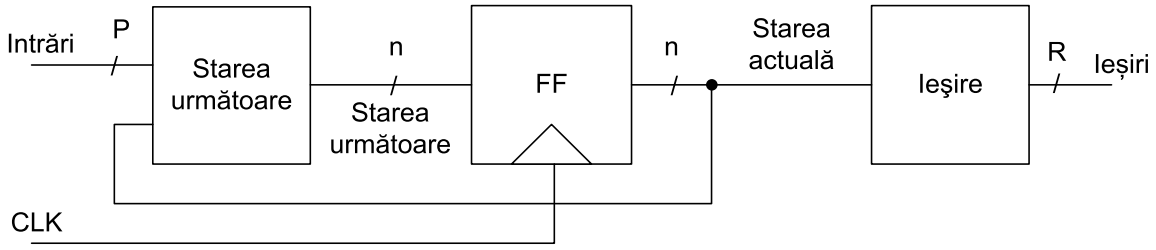


Figura 6.3. Structura maşinii cu stări finite de tip Moore.

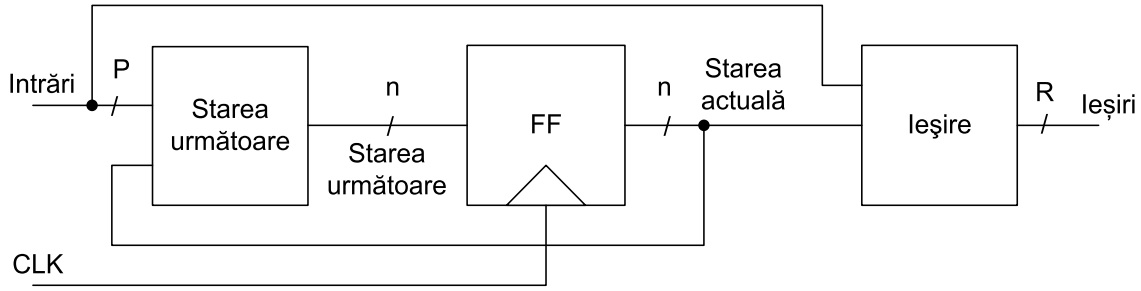


Figura 6.4. Structura maşinii cu stări finite de tip Mealy.

În lista 6.2 este evidențiat codul VHDL ce descrie funcționarea unor automate Moore și Mealy. Organizarea codului este conformă cu organizarea standard a celor două automate evidențiată în figurile 6.3 și 6.4. Observăm zona de flip-flop-uri, zona stării următoare și zona ieșirii. Modulul starea următoare va implementa un incrementator, iar ieșirea va decodifica starea actuală a circuitului. Circuitul va fi alcătuit din 3 flip-flop-uri, așadar vom avea un numărător pe 3 biți și un decodificator 3 la 8.

Lista 6.2. Implementarea maşinilor cu stări finite de tip Moore şi Mealy

```
-- librarii---------------------------------

library ieee;
use ieee.std_logic_1164.all;
```

```
-------------------------------------------

-- intrari-iesiri----------------------------

entity decodificator is
port (
        clk, reset      : in std_logic;
        decod_ena       : in std_logic;
        output          : out std_logic_vector(7 downto 0));
end;

-------------------------------------------

-- arhitectura circuitului digital-----------

architecture circ of decodificator is

signal count_starea_urmatoare : std_logic_vector(2 downto 0);
signal count_starea_prezenta  : std_logic_vector(2 downto 0);

begin

--------------- Flip-flop-uri ----------------

process(clk, reset)
begin
        if rst = '1' then
                count_starea_prezenta <= (OTHERS => '0');
        elsif (clk'event and clk = '1') then
                count_starea_prezenta <= count_starea_urmatoare;
        end if;
end process;

-------------------------------------------

------------ Starea urmatoare ----------------

count_starea_urmatoare <= count_starea_prezenta + '1';

-------------------------------------------

---------------- Iesire de tip Moore---------

process(count_starea_prezenta)
begin
        case count_starea_prezenta is
                when "000" =>
                        output <= "00000001";
                when "001" =>
                        output <= "00000010";
```

```
                when "010" =>
                        output <= "00000100";
                when "011" =>
                        output <= "00001000";
                when "100" =>
                        output <= "00010000";
                when "101" =>
                        output <= "00100000";
                when "110" =>
                        output <= "01000000";
                when others =>
                        output <= "10000000";
                end case;
end process;

-------------------------------------------------

----------------- Iesire de tip Mealy---------

process(count_starea_prezenta, decod_ena)
begin
      if decod_ena = '1' then
         case count_starea_prezenta is
                when "000" =>
                        output <= "00000001";
                when "001" =>
                        output <= "00000010";
                when "010" =>
                        output <= "00000100";
                when "011" =>
                        output <= "00001000";
                when "100" =>
                        output <= "00010000";
                when "101" =>
                        output <= "00100000";
                when "110" =>
                        output <= "01000000";
                when others =>
                        output <= "10000000";
                end case;
      else
           output <= "00000000";
end process;

-----------------------------------------------

end circ;
```

6.3.2.2 Efectuarea exercițiului 2

1. Se va deschide proiectul numit decodificator.
2. Se va compila proiectul.
3. Se va simular proiectul
4. Se va vizualiza circuitul sintetizat cu ajutorul RTL Viewer-ul.
5. Se vor identifica componentele automatelor finite.

6.3.3 Exercițiul 3 – Utilizarea Chip Planner-ului

6.3.3.1 Cerințe de proiectare

Codul VHDL din lista 6.2 va fi implementat în FPGA, iar cu ajutorul Chip Planner-ului vom putea observa felul în care software-ul Quartus II a ales să monteze acest circuit.

6.3.3.2 Efectuarea exercițiului 3

1. Se va deschide proiectul numit decodificator.
2. Se va compila proiectul.
3. Cu ajutorul Chip Planner-ului vom observa modul în care software-ul Quartus II a ales să monteze automatele în FPGA, astfel: Tools\Chip Planner (Floorplan and Chip Editor).

6.3.4 Exercițiul 4 – Identificatorul de secvențe

6.3.4.1 Cerințe de proiectare

În figura 6.5 este evidențiată funcționarea unui identificator de secvențe cu ajutorul unei diagrame algortimice cu stări. Identificatorul de secvențe este evidențiat în figura 6.6 și va avea ca sarcină găsirea secvenței 111 de pe intrare de date. Ieșirea circuitului va fi 1 în mometul când pe intrarea de date au fost găsite trei valori logice 1 consecutive, altfel ieșire circuitului va fi 0.

Identificatorul de sevcențe se implementează ca o mașină cu stări finite de tip Moore a cărei organizare este evidențiată în figura 6.3.

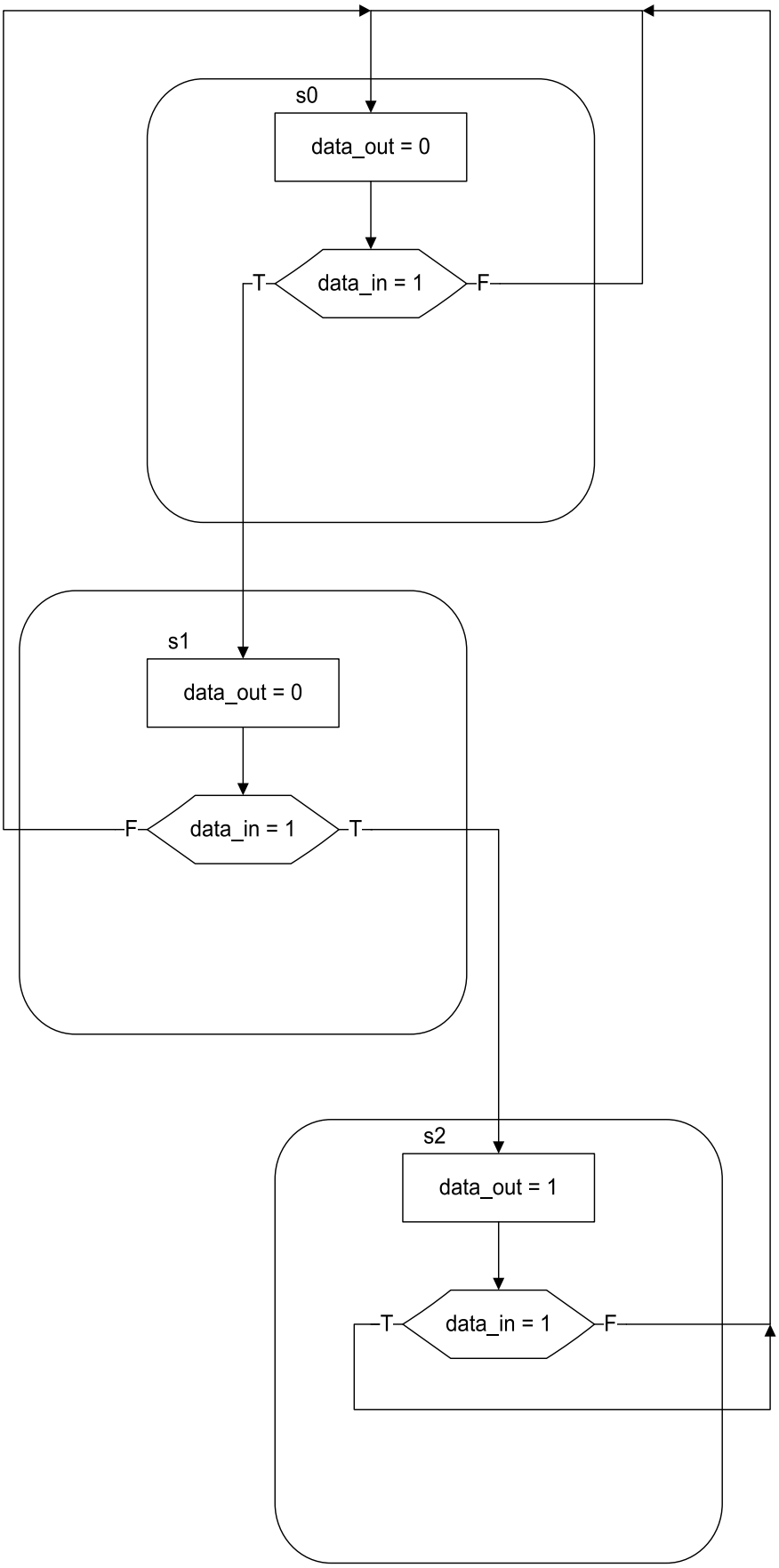


Figure 6.5. Identificatorul de secvențe. Identificarea secvenței 111.

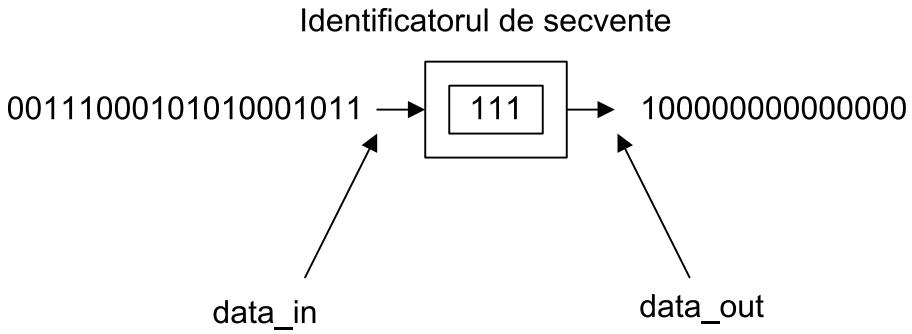


Figura 6.6. Schema de principiu a identificatorului de secvențe. Identificarea secvenței 111.

6.3.4.2 Efectuarea exercițiului 4

1. Se va deschide proiectul numit idetificator_secventa.
2. Se vor scrie liniile de cod VHDL ce evidențiază funcționarea identificatorului de secvențe, în locul delimit cu verde, conform figurii 6.6. Se va folosi ca șablon lista 6.1.
3. Se va compila proiectul.
4. Se va sintetiza proiectul.
5. Se va simula circuitul sintetizat.
6. Se va vizualiza structura circuitului sintetizat.
7. Se va vizuliza structura mașinii cu stări finite.
8. Se va utiliza Chip Planner-ul pentru a vizualiza modul în care software-ul Quartus II a decis să implementeze în FPGA identificatorul de secvențe.

Laboratorul 7

Metodologia de transfer de date între registre

7.1 Obiective operaționale

Obiectivele principale ale acestui laborator reprezintă însușirea metodologie RT și a metodologie de testare a circuitelor logice. Exercițiile prezentate în acest laborator au rolul de a exemplifica modul în care este utilizată metodologia RT în crearea circuitelor logice și felul în care sunt testate circuitele digitale.

La finalul acestui laborator studentul va putea crea circuite logice de complexitate medie și mare utilizând metologia RT. Va putea testa circuitele create cu ajutorul analizorului logic.

7.2 Instrumente necesare

Software-ul Altera Quartus II

Kit-ul educațional

7.3 Desfășurarea lucrării

7.3.1 Exercițiul 1 – Metodologia RT

7.3.1.1 Noțiuni teoretice

Un circuit logic complex, cum ar fi un microprocesor, are două componente principale: calea de date (unitatea funcțională) și unitatea de control (mașina cu stări finite). În laboratorul 6 mașinile cu stări finite au fost implementate cu ajutorul diagramei algoritmice cu stări. Pentru a crea un circuit complex trebuie să utilizăm o tehnică care permite descrierea concomitentă a mașinii cu stări finite și a unității sau unităților funcționale ce alcătuiesc circuitul logic complex. Această tehnică poartă numele de metodologia RT și integrează

principiul de funcționare al diagramei algortimice cu stări și principiul de transfer de date între registre.

O operație de transfer de date între registre se definește astfel

$$r_{destinație} \leftarrow f(r_{sursă1}, r_{sursă2}, r_{sursă3,\ \ldots})$$

f poate fi orice funcție logică ce poate fi implementată printr-un circuit logic. Operația ← se definește astfel: valoarea următoare a registrului $r_{destinație}$ va fi stocată pe frontrul crescător al tactului și va fi egală cu valoarea $f(r_{sursă1}, r_{sursă2}, r_{sursă3,\ \ldots})$, ce va fi obținută aplicând funcția f valorilor actuale ale registrelor $r_{sursă1}, r_{sursă2}, r_{sursă3,\ \ldots}$. Operația de transfer de date poate exista numai în caseta stării actuale și în caseta de ieșire condiționată.

În figura 7.1 este evidențiat un exemplu de diagramă algoritmică cu stări și transfer de date. Diagrama descrie funcționarea unui multiplicator. Un multiplicator este un circui logic ce implementează operația de înmulțire. Circuitul are două intrări și o singură ieșire. Spre exemplu dacă datele de intrare sunt pe 8 biți atunci ieșirea de date este pe 16 biți.

7.3.1.2 Efectuarea exercițiului 1

1. Se va deschide proiectul asmd_exemplu.
2. Se va vizualiza codul VHDL ce implementează diagrama din figura 7.1.
3. Se va observa modul în care au fost create stările automatului finit (liniile de cod delimitate de comentariile “stările mașinii cu stări finite”).
4. Se va observa felul în care operațiile de transfer de date între registrii au fost
5. Se va compila proiectul.
6. Se va vizualiza întregul sistem în RTL Viewer.
7. Se va identifica mașina cu stări finite și unitatea funcțională împreună cu restul circuitelor logice de legătută (multiplexoare).
8. Se va simula circuitul logic sintetizat.
9. Se va observa modul în care Quartus II a decis să plaseze în FPGA circuitul logic sintetizat.

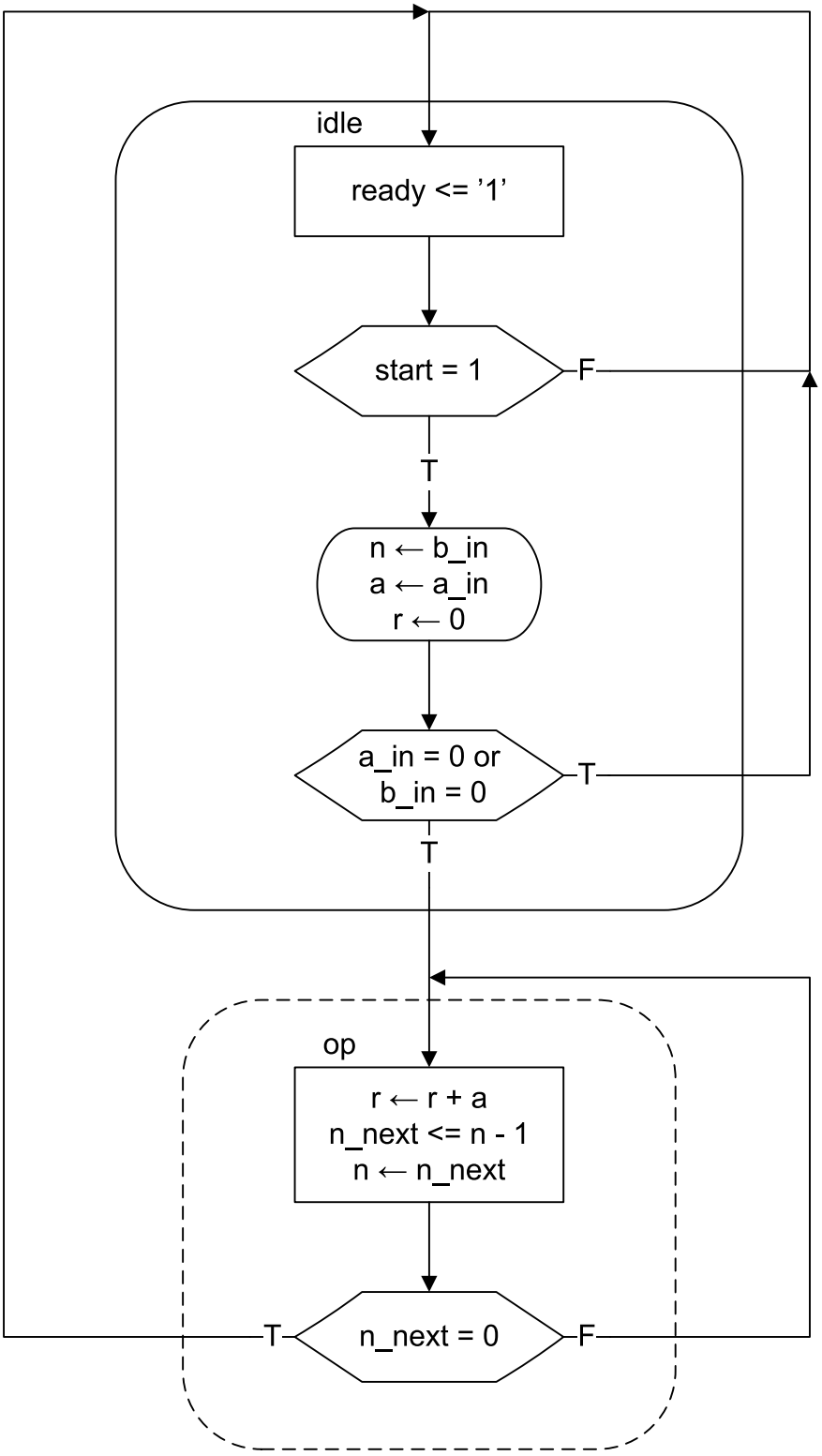


Figura 7.1. Exemplu de diagramă algoritmică cu stări și transfer de date. Figura este preluată și adaptată din cartea RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability de Pong P. Chu.

7.3.2 Exercițiul 2 – Utilizarea analizorului logic SignalTap II

7.3.2.1 Noțiuni teoretice

Un inginer electronist, ce proiectează circuite logice digitale, a creat circuitul din figura 7.2. Incrementatorul are rolul de a aduna o unitate la starea actuală a sistemului, deci va fi alcătuit dintr-un sumator. Prima intrare a sumatorului este starea actuală a sistemului, iar a doua intrare (care nu este reprezentată) este constanta 1. Circuitul are trei elemente de memorie (trei flip-flop-uri de tip D sau un registru paralel pe 3 biți). Decodificatorul 3 la 8 preia starea actuală a sistemului și o decodifică. Ieșirea decodificatorului este conectată la 8 led-uri. Decodificatorul va aprinde un singur led in funcție de starea actuală (dacă starea actuală este $001_2 = 1_{10}$ atunci ieșirea decodificatorului va fi 00000001_2). Tabelul de adevăr al decodificatorului 3 la 8 este reprezentat în tabelul 7.1.

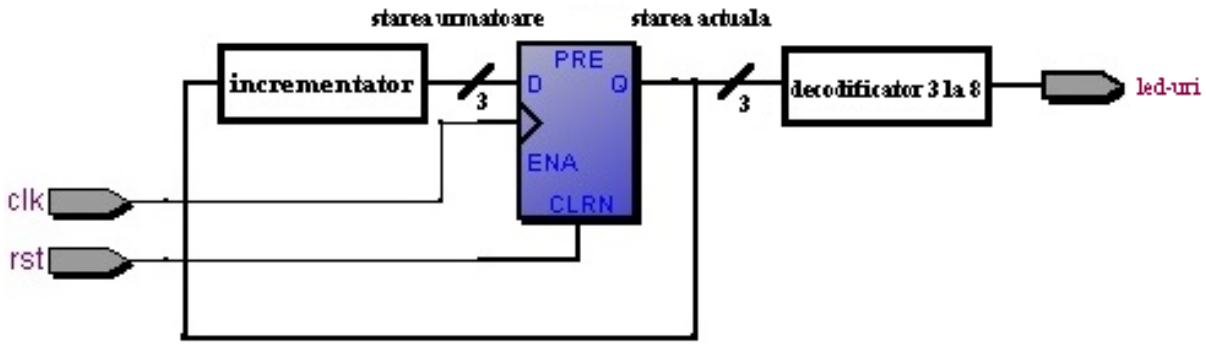


Figura 7.2. Circuit logic secvențial sincron pentru decodificare.

Tabelul 7.1 – Tabel de adevăr pentru decodificaorul 3 la 8.

Reprezentare în zecimal	Starea actuală	Ieșire	Reprezentare în zecimal
0	000	00000001	1
1	001	00000010	2
2	010	00000100	4
3	011	00001000	8
4	100	00010000	16
5	101	00100000	32
6	110	01000000	64
7	111	10000000	128

Inginerul ce a proiectat circuitul din figura 7.2 nu a respectat metodologia de proiectare a circuitelor logice. Acesta a creat un fisier vhdl ce evidențiază funcționarea circuitului însă a omis să îl simuleze și a presupus că circuitul funcționează corect și că fișierul vhdl este scris fără nicio greșeală.

Fisieul vhdl conține liniile de cod din lista 7.1.

Lista 7.1. Codul VHDL al unui circuit logic secvențial sincron.

```
-- librarii---------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-----------------------------------------

-- intrari iesiri----------------------------

entity decodificator_incorect is
port (          KEY          : in std_logic_vector(0 downto 0);
                CLOCK_50     : in std_logic;
                LED          : out std_logic_vector(7 downto 0)
      );
end;

-----------------------------------------

-- arhitectura circuitului digital-----------

architecture circ of decodificator_incorect is

signal starea_urmatoare_numarator   : std_logic_vector(2 downto 0);
signal starea_prezenta_numarator    : std_logic_vector(2 downto 0);
signal iesire                       : std_logic_vector(7 downto 0);
signal rst, clk                     : std_logic;

attribute keep : boolean;
attribute keep of rst : signal is true;
attribute keep of starea_urmatoare_numarator : signal is true;

begin

clk <= CLOCK_50;
rst <= KEY(0);
```

```
---------- Flip-flop-uri ----------

process(clk, rst)
begin
	if rst = '1' then
		starea_prezenta_numarator <= (OTHERS => '0');
	elsif (clk'event and clk = '1') then
		starea_prezenta_numarator <= starea_urmatoare_numarator;
	end if;
end process;

-----------------------------------------

---------- Starea urmatoare ----------

starea_urmatoare_numarator <= starea_prezenta_numarator + 2;

-----------------------------------------

---------- Iesire ----------

process(starea_prezenta_numarator)
begin
	case starea_prezenta_numarator is
		when "000" =>
			iesire <= "00000001";
		when "001" =>
			iesire <= "00000000";
		when "010" =>
			iesire <= "00000100";
		when "011" =>
			iesire <= "00101000";
		when "100" =>
			iesire <= "00010000";
		when "101" =>
			iesire <= "00100010";
		when "110" =>
			iesire <= "01000000";
		when others =>
			iesire <= "00000010";
		end case;
end process;

LED <= iesire;

-----------------------------------------

end circ;
```

Cu ajutorul analizorului logic SignalTap II vom vedea dacă circuitul logic din figura 7.2 funcționează corect.

7.3.2.2 Efectuarea exercițiului 2

1. Se va deschide proiectul numit decodificator_incorect.
2. Se va deschide analizorul logic SignalTap II (Tools\SignalTap II Logic Analyzer).
3. Din fereastra SignalTap II se va implementa în FGPA circuitul deja compilat.
4. Din fereastra SignalTap II se va analiza circuitul implementat în FPGA (Processing\Run Analysis).
5. Se vor analiza valorile stării prezente, stării următoare și ieșirii decodificatorului. În funcție de valoarea acestora se va decide dacă circuitul funcționează corect sau nu. Dacă circuitul nu funcționează corect atunci se va corecta fișierul vhdl pentru ca circuitul să funcționeze corect.

Bibliografie

1. Brown S., Vranesic Z., *Fundamentals of Digital Logic with VHDL*, 2nd ed., McGraw Hill, 2005.
2. Chu P., *RTL Hardware Design using VHDL*, John Wiley & Sons, 2000.
3. Ciletti M., *Advanced Digital Design with the Verilog HDL*, Prentice Hall, 2003.
4. Harris D., Harris S., *Digital Design and Computer Architecture*, Morgan Kaufmann, 2007.
5. Katz, R., *Modern Logic Design*, 2nd ed., Addison-Wesley, 2004.
6. Mano M., Ciletti M., *Digital Design*, Prentice Hall, 4th ed., 2006.
7. Patterson D., Hennessy J., *Computer Organization and Design*, Morgan Kaufmann, 4th ed., 2009.
8. Pedroni V., *Circuit Design with VHDL*, MIT Press, 2004.
9. Roth C., *Digital System Design with VHDL*, PWS Publishing Company, 1998.
10. Wakerly J., *Digital Design: Principles and Practices*, 3rd ed., Prentice Hall, 2000.
11. www.altera.com
12. www.terasic.com