APLICAŢIA 7
CIRCUITE SECVENŢIALE
REGISTRUL CU ÎNCĂRCARE PARALELĂ

1. Rezumat

Acest laborator îşi propune implementarea unui cicuit secvenţial simplu: registrul pe 4 biţi cu încărcare paralelă. Pentru aceasta sunt necesare: 4 intrări de date şi 1 intrare care comandă încărcarea registrului. Ieşirea registrului va fi afişată folosind afişajul cu segmente. Circuitul descris este un circuit secvenţial care are 4 biţi de intrare, 1 intrare de comandă şi 4 ieşiri corespunzătoare valorii încărcate în registru.

Obiectivele lucrării

Obiectivul acestui laborator este acela de implementare şi înţelegere a principiilor de funcţionare pentru un circuit secvenţial simplu folosind Verilog HDL. De asemenea se cere descrierea Verilog a modulului şi verificare funcţionării corecte folosind placa Nexys-2.

Obiective tehnice

1. Modelarea circuitelor secvenţiale în Verilog HDL.
2. Implementarea unui circuit secvenţial simplu.
3. Sinteza şi implementare design pe placa FPGA Nexys-2.

Timp necesar

2-3 ore

Pregătirea pentru laborator

- Citiţi documentul înainte de a începe realizarea practică.
- Salvați ieşirile pentru fiecare cerință sau anunțați cadrul didactic în vederea prezentării rezultatelor.

Echipamente și Materiale

Acces la software-ul Xilinx

Necesar	Cantitate
Software ISE® WebPACK™ 14.4 de pe pagina de WEB Xilinx, www.xilinx.com	1
Plugin Digilent (www.digilent.com)	1
Placă Digilent Nexys 2	1
Cablu PMOD	1
Placă de expansiune – PMODSw	1
Placă de expansiune – PMODSSD	1

2. Registrul cu încărcare paralelă și semnal de validare a intrărilor

2.1 Elementele secvenţiale

Principalul element secvenţial folosit în circuitele digitale este bistabilul sincron activ pe frontal semnalului de tact (edge triggered flip-flop). În cazul acestor bistabile, comutarea este posibilă doar pe unul dintre fronturile semnalului de tact; spre deosebire de bistabilul activ pe palierul semnalului de tact, edge triggered flip-flop prezintă avantajul de avea ieşirea stabilă pe întreaga durată a semnalului de tact, nefiind afectată de eventualele glitch-uri ce pot apărea la intrare.

În ceea ce priveşte funcţionarea acestor dispozitive, există o serie de timpi legați de funcționarea FF-urilor, care trebuiesc luați în calcul și respectați în cazul proiectarii circuitelor reale:

- *Timpul de setup*: se referă la timpul necesar pentru ca semnalul de intrare (D) să rămână stabil înainte de apariţia frontului semnalului de tact;

- *Timpul de hold*: reprezintă timpul în care datele de intrare nu pot fi modificate după apariţia frontului semnalului de tact în vederea încărcării corecte a acestora;
- *Timpul aferent întârzirii datorate propagării* (tp): constituie timpul necesar basculării FF-ului (clock to Q delay)

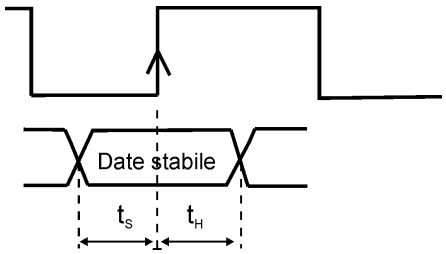


Fig. 7.1 – Timpii de setup şi hold (sursa 1.[11])

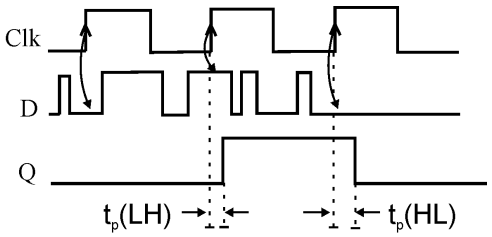
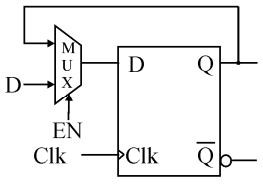


Fig. 7.2 – Timpii de întârziere aferenţi bistabilelor D active pe front (sursa 1.[11])

În continuare este prezentat bistabilul de tip D cu semnal de validare (EN) a intrărilor:



EN	D	Clk	Q_next	$\overline{Q}$_next
0	0	↑	Q	$\overline{Q}$
0	1	↑	Q	$\overline{Q}$
1	0	↑	0	1
1	1	↑	1	0
*	*	0	Q	$\overline{Q}$
*	*	1	Q	$\overline{Q}$

Fig.7.3 - FF de tip de cu semnal de validare (EN) (sursa 1.[11])

Codul Verilog HDL aferent lui este prezentat mai jos.

```
module D_FF_EN
        (input clk,
        input rst,
        input d,
        input en,
        output reg q);

        always @( posedge clk or posedge rst)
        begin

                if (rst)
                        q <= 0;
                else if (en )
                        q <= d;
        end
endmodule
```

În sistemele de calcul este necesară memorarea datelor pe *n* biți. Acest lucru poate fi realizat în două moduri: paralel – când toți biții sunt transmiși concomitant pe *n* linii, sau serial când biții sunt transmiși pe rând – bit după bit. Pentru prima situație este nevoie de un ciclu de tact pentru a încărca informația, iar pentru cea de-a doua sunt necesari *n* cicli de tact pentru acest lucru. De asemenea din punct de vedere al modului prin care sunt accesați cei *n* biți memorați, sunt 2 variante: serie – biții sunt citiți pe rând în *n* cicluri de tact, sau paralel - toți biții sunt citiți concomitant în același ciclu de tact.

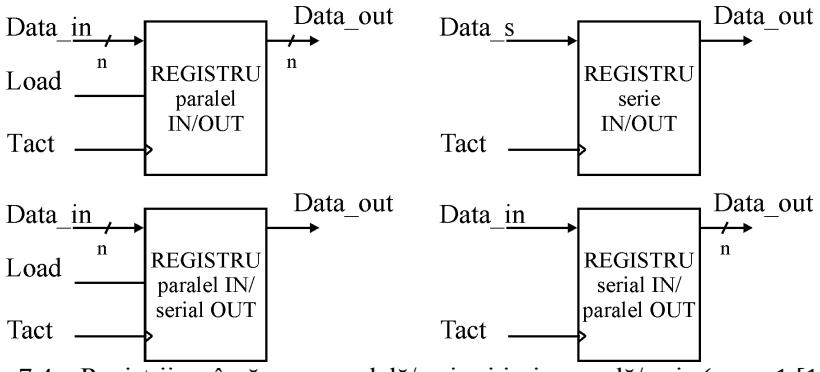


Fig. 7.4 – Regiştrii cu încărcare paralelă/serie şi ieşire parelă/serie (sursa 1.[11])

Cea mai simplă variantă de registru este registrul cu încarcare paralelă și citire a datelor paralelă. Pentru a stabili momentul când se încarcă

informație nouă, este prevăzut un semnal de validare a intrărilor (*LD*). Schema pentru acest registru este prezentată în continuare:

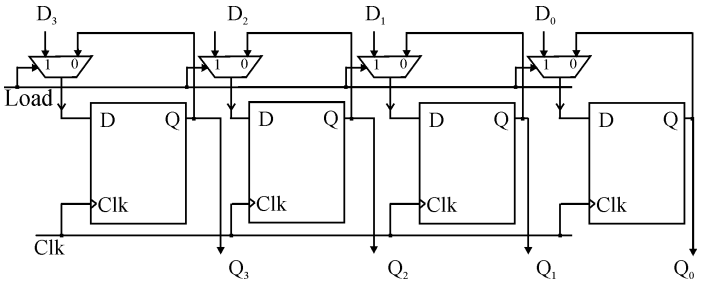


Fig. 7.5 – Registru cu încărcare paralelă şi ieşire paralelă (sursa 1.[11])

Codul Verilog aferent acestui registru este următorul:

```
module paralel_register
        (input clk, rst,
        input load,
        input [3:0]d,
        output reg [3:0] q);

        always @( posedge clk or posedge rst)
        begin

                if (! rst)
                        q <= 0;
                else
                        if (load == 1'b1)
                                q <= d;
        end
endmodule
```

Aşa cum se poate observa din figură, semnalul de încărcare *Load* este activ pe 1 logic. Acesta determină activarea corespunzătoare a căii din multiplexor care selectează la ieşirea acestuia valoarea semnalului de intrare date (*Di*).

2.2 Modelarea circuitelor secvenţiale folosind 2 segmente

Pentru circuite mai complexe, care conțin o parte combinațională și o parte secvențială, este util să separăm partea de memorare a datelor (secvențială), de cea care calculează noile date care vor fi memorate (combinațională). Pentru aceasta folosim o construcție cu două segmente (două blocuri *always*): unul pentru partea combinațională și unul aferent părții secvențiale. În acest sens, încercăm construcția unui numărător pe 4 biți, care numără crescător până la atingerea valorii 15 (*F* în baza hexazecimală). Cele 2 părți pot fi cu ușurință ditinse în figura ce urmează: element de memorare de tip registru cu încărcare paralelă, și element combinațional de procesare - sumatorul.

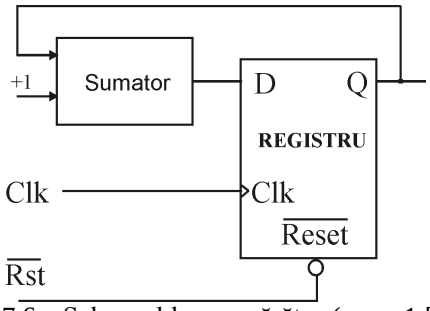


Fig. 7.6 – Schema bloc numărător (sursa 1.[11])

Codul Verilog HDL este prezentat în cele ce urmează. Se impun următoarele precizări:

- Primul *always* block este folosit pentru a descrie partea secvențială – registrul în care este memorată starea numărătorului. În lista sa de sensitivități sunt prezente:
  - Tactul – frontul crescător;
  - Resetul – fron descrescător;

Ca și constrângere, din punctul de vedere al sintaxei, nu pot coexista în lista de sensitivități atât frontul cât și palierul unui semnal. Așadar, dacă este specificat reset-ul acesta trebuie să fie specificat tot ca și front.

- Al doilea *always* block este folosit pentru a descrie partea combinațională– pregătește următoarea stare a numărătorului. În lista sa de sensitivități sunt prezente toate semnalele a căror modificare atrage

după sine modificarea valorii noii stări, deci re-evaluarea codului dintre *begin* și *end* al blocului *always*.

```
module numarator
        (input clk, rst
        output reg [3:0] q);

        reg [3:0] q_nxt, q_reg;

        //always block for sequential part
        always @( posedge clk or posedge rst)
        begin
                if ( rst)
                        q_reg <= 0;
                else
                        q _reg<=  q_nxt;

        end

//always block for combinational part
        always @( q_reg)
        begin
                q_nxt = q_reg + 1'b1;
        end

endmodule
```

3. Implementarea unui registru cu încărcare parelelă şi ieşire paralelă

Se cere implementarea unui registru cu încărcare paralelă pe 4 biţi şi ieşire paralelă. Se va merge pe implementarea codului descris în secţiunea 2.1. De asemenea, se cere implementarea unui numărător descrescător pe 4 biţi, folosind metodologia descrisă în secţiunea 2.2

Pas 1 – Crearea unui proiect Xilinx ISE şi descrierea unei circuit de tip registru

Succint vor fi punctate etapele realizării unui proiect nou:

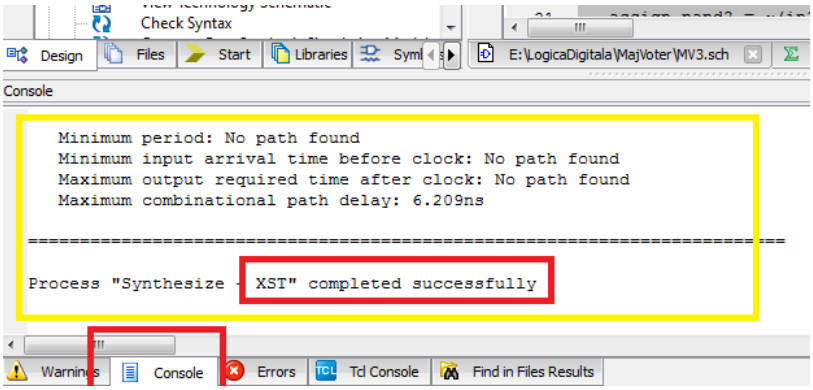
- Pentru pornire ISE: deschideți un terminal și tastați *ise*
- Creați un proiect nou în directorul workspace: *parallel_reg_hex*
- În continuare realizați utilizând limbajul de descriere hardware Verilog componenta din figura de mai jos. La *Hierarchy* în tab-ul de *Design* selectați *Project→New source* deschide fereastra *New Source Wizard*. Pentru implementarea folosind descrierea Verilog HDL alegeți la *Select Source Type – Verilog Module.*

  Proiectul va avea cinci surse*:*
  - *parallel_reg* – este modulul ce modelează registrul cu încărcare paralelă ieşire paralelă
  - *hex_7display* – este modulul care va implementa decodificatorul pentru afişajul cu 7 segmente;
  - *parallel_reg_hex* – instanţiază o unitate *parallel_reg* şi un *hex_7display*
  - *counter_dec* – este modulul aferent numărătorului
  - *counte_dec_hex* – reprezintă modulul ce instanţiază modulele aferente numărătorului şi afişajului

- Adăugați la proiect un fişier de tip testbench . *Project→New source* deschide fereastra *New Source Wizard*, alegeți la *Select Source Type – Verilog Test Fixture.* Simulaţi circuitul folosind simulatorul ISIM.

Pas 3 – Sinteza circuitului

La *Hierarchy* în tab-ul de *View* selectați *Implementation*. Se poate observa că fişierul testbench a dispărut.
În continuare selectați modulul care doriți să-l setați ca și top-level (cel al cărui design va fi programat pe FPGA) – *parallel_reg_hex* şi *counter_hex*.
În tabul de *Design* dați click pe *Synthesize->Run*. Alternativa este să dați dublu click pe *Synthesize*.
Remarcați la output-ul din tab-ul *Console*, finalizarea cu succes a operației de sinteză.



Pas 3 – Implementarea circuitului

Înainte de a trece la configurarea design-ului pe placă mai aven nevoie de crearea fişierului .UCF. Placa folosită este Nexys-2 cu FPGA-ul Spartan3-E 500 FG320. Toate aceste informații se găsesc specificate în manualul plăcii (Nexys-2 Board Reference Manual).
Circuitul pe care dorim sa-l verificăm foloseşte 6 comutatoare pentru intrări şi 1 afişaj cu 7 segmente.
Va fi folosită componenta PmodSWT care este conectata la interfața PMOD2, atunci trebuie consultat manualul aferent acestuia şi trebuie identificați pinii pentru conectorul PMOD2 ai plăcii Digilent Nexys2.

Pentru placa Nexys2, din manual studiați specificația pentru PMOD2 şi extrageți informațiile referitoare la pini. Vor fi folosiţi pinii indicaţi mai jos:

Table 3: Nexys2 Pmod Connector Pin Assignments

Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:; P18	JD10: J14[4]

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3

În ceea ce priveşte afişajul cu 7 segmente, se va folosi modulul de expansiune PMODSSD. Acesta va fi conectat la portul de expansiune JC.

Table 3: Nexys2 Pmod Connector Pin Assignments							
Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:; P18	JD10: J14[4]

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3

Semnalul de incărcare a registrului paralel va fi la unul dintre butoanele plăcii. De asemenea, semnalul de reset va fi legat şi el la butoanele plăcii.

Vor fi create fişierele *parallel_reg_hex.ucf*, respectiv *counter_hex.ucf*.
Se va continua prin implementarea şi crearea fişierului de configurare .bit.

Pas 4 – Configurare placă FPGA

Ultimul pas constă în descărcarea design-ului pe placă.

Din Terminal tastați:

djtgcfg prog -d *Nexys2* -i 0 -f parallel_reg_hex.bit
djtgcfg prog -d *Nexys2* -i 0 -f counter_hex.bit

Completați liniile de cod lipsă, simulati şi verificați funcționarea corectă a design-urilor!

Bibliografie:

[1] Xilinx - Xilinx UG695 ISE In Depth Tutorial - http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf , 2012

[2] C. Kief, A. Vera, A. Haddad, Q. Cao. COSMIAC FPGA Tutorials http://cosmiac.org/thrust-areas/education-and-workforce-development/fpga/ate-developed-material/.

[3] J. F. Wakerly – Digital Design: Principles and Practices, 3rd Edition, Prentice Hall, 2000

[4] J. Bhasker - A Verilog HDL Primer, Third Edition - Star Galaxy Publishing, 2005

[5] P. Chu - RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability, Wiley – IEEE Press, 2006

[6] S. Brown, Z. Vrsaniec - Fundamentals of Digital Logic with Verilog Design - McGraw-Hill, 2007

[7] R. Haskell, D. Hanna - Introduction to Digital Design Using Digilent FPGA Boards – Block Diagram/Verilog Examples – LBE Books, 2009

[8] Digilent Nexys 2 Reference Manual - https://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

[9] Digilent PMODSSD Reference Manual - http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-SSD

[10] Digilent PMODSWT Reference Manual - https://www.digilentinc.com/Data/Products/PMOD-SWITCH/Pmod%20SWT_rm.pdf

[11] O. Boncalo, A. Amăricăi. "Proiectarea circuitelor digitale folosind Verilog HDL – Analiza si Sinteza". Editura Politehnica, 2011.