Oniga Ștefan

Circuite digitale I

Lucrări de laborator

UTPRESS
Cluj-Napoca, 2020
ISBN 978-606-737-483-4

Editura U.T.PRESS
Str.Observatorului nr. 34
C.P.42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Șl. dr. ing. Ioan Orha
Șl. dr. ing. Claudiu Lung

ISBN 978-606-737-483-4

Cuprins

Lucrarea de laborator nr. 1

- Prezentarea și testarea plăcilor de dezvoltare Nexys2
- Prezentarea mediului de dezvoltare integrat Xilinx ISE

Nexys 2- Basys 3 - Nexys 4

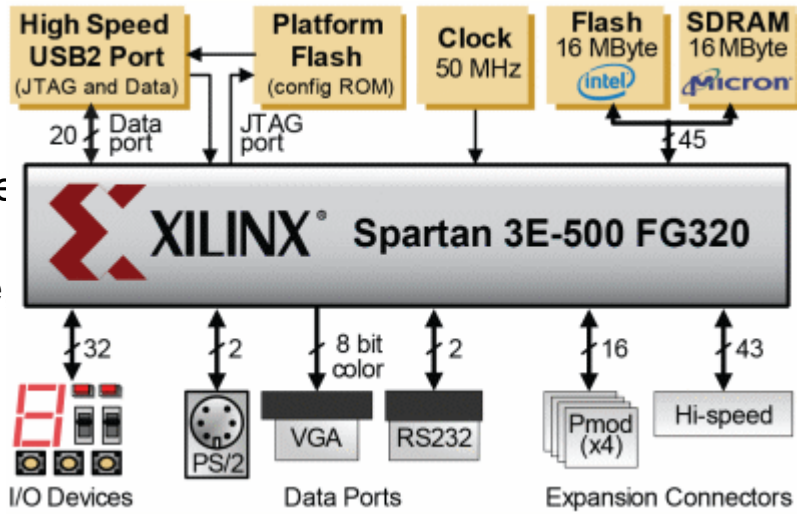
- Plăcile de dezvoltare folosite în cadrul lucrărilor de laborator sunt Nexys 2, Basys 3 și Nexys 4
- Principalele lor caracteristici sunt prezentate în tabelul următor

Nexys 2	Basys 3	Nexys 4
Spartan-3E FPGA	Artix-7 FPGA	Artix-7 FPGA
4656 slices	5200 slices	15580 slices
9312 Flip-Flops	41600 flip-flops	126800 flip-flops
20 Dedicated Multipliers	90 DSP slices	240 DSP slices
360 Kbits Block RAM	1800 Kbits Block RAM	4860 Kbits Block RAM
16MB fast Micron PSDRAM		256 MB DDR2
ISE Design Suite	VivadoDesign Suite	VivadoDesign Suite

Digilent Nexys 2

FPGA Xilinx Spartan-3E, 500K porți logice echivalente

- Port USB2 – permite alimentarea, configurarea și transferul datelor
- Se poate folosi cu ISE/Webpack și EDK
- Memorie DRAM:16MB fast Micron PSDRAM
- Memorie Flash: 16MB Intel StrataFlash
- Memorie ROM: Xilinx Platform Flash ROM
- Surse de alimentare în comutație eficiente
- Oscilator de 50MHz oscillator, și soclu pentru un oscilator suplimentar
- 75 de pini de I/O conectate la conectoarele de expandare (Hirose FX2 cu 43 signals și 4 conectoare 2x6 Pmod)
- Toate semnalele de I/O sunt protejate la descărcare electrostatică și la scurtcircuit.
- Periferice on-boar: 8 leduri, 4 afișaje cu 7 segmente butoane și 8 switch-uri

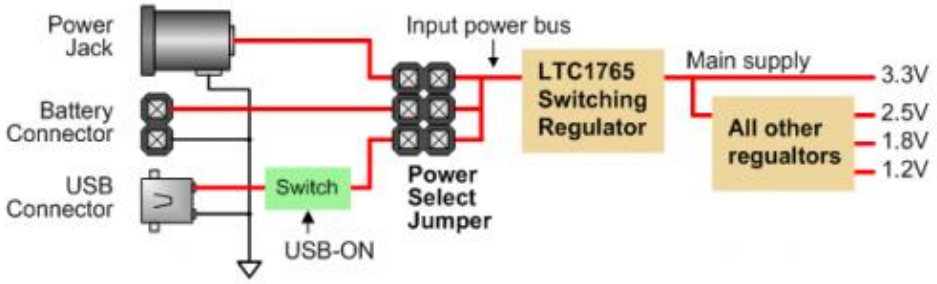


- Nexys2 manual
  http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf
- Digilent Nexys2 :
  http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,790&Prod=NEXYS2

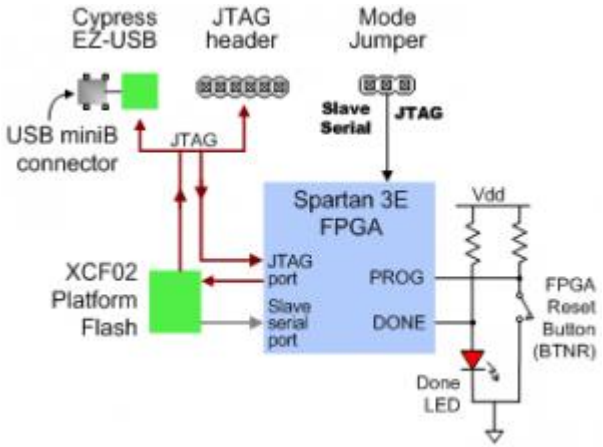
Digilent Nexys 2

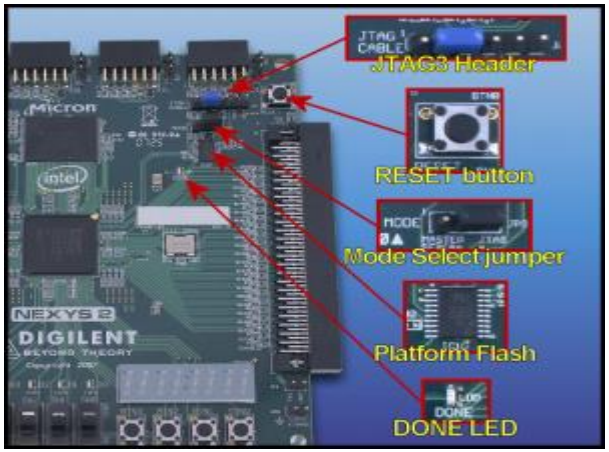
- **Posibilități de alimentare plăcii**
  - USB
  - Baterie
  - Sursă externă

Supply	Device	Amps (max/typ)
3.3V main	IC6: LTC1765	3A/100mA
2.5V FPGA	IC7: LTC3417	1.4A/50mA
1.2V FPGA	IC7: LTC3417	1.4A/200mA
1.8V SRAM	IC8: LTC1844	150mA/90mA
3.3V USB	IC5: LTC1844	150mA/60mA



- **Configurarea circuitului FPGA și a memoriei Flash**

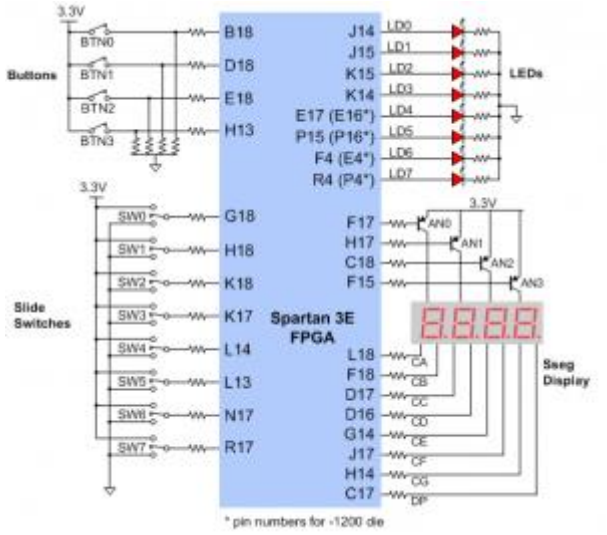


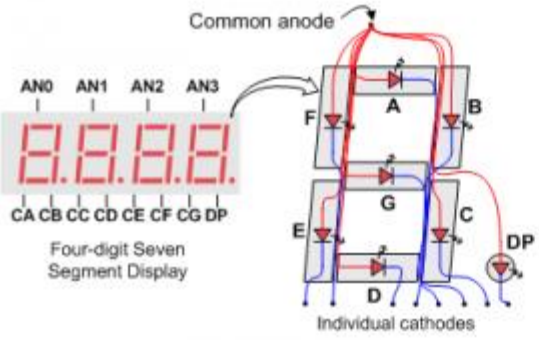


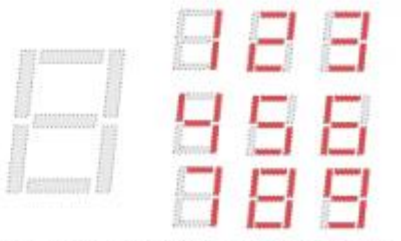
[Nexys 2 Reference Manual]

Intrări și ieșiri utilizator

- Butoanele în stare de repaus sunt conectate la masă. La apăsare generează un nivel logic ridicat
- Comutatoarele generează nivel ridicat sau coborât în funcție de poziția lor
- Butoanele și comutatoarele utilizează rezistoare de protecție la scurtcircuit
- Ledurile sunt conectate la pinii FPGA prin intermediul unor rezistoare de 390 Ω care limitează curentul prin leduri la 3-4 mA
- Afișajele cu 7 segmente sunt cu anod comun







[Nexys 2 Reference Manual]

Digilent Adept suite

- Digilent Adept este un program dezvoltat de firma Digilent care permite comunicarea cu plăcile Digilent

- **ADEPT pentru Windows**

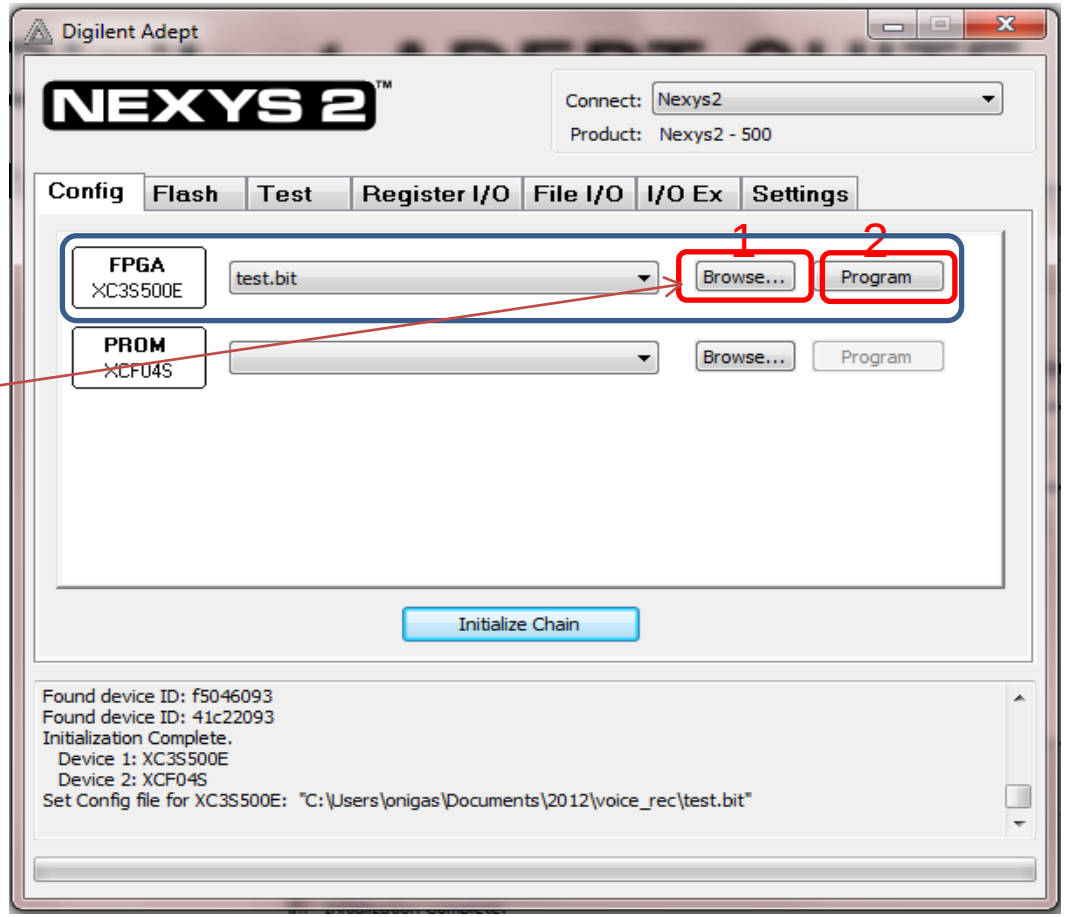
- Adept 2 permite configurarea și transferul datelor folosind standardul JTAG
- De asemenea oferă posibilitatea testării plăcilor și transferul datelor I/O
- Configurează circuitele logice configurabile Xilinx . Initializează un scan chain, programează circuiteleFPGA, CPLD, și PROM
- Transferă datele spre și dinspre circuitul FPGA de pe placă.
- Programează dispozitivul Xilinx XCFS Platform Flash folosind fișierele .bit sau .mcs.
- Programează dispozitivele CPLDs Xilinx CoolRunner2 folosind fișierele .jed.
- Programează marea majoritate a FPGA-urilor de tip Spartan și Virtex cu fișierele .bit

Digilent Adept Suite

- https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2

Încărcarea fișierului de testare

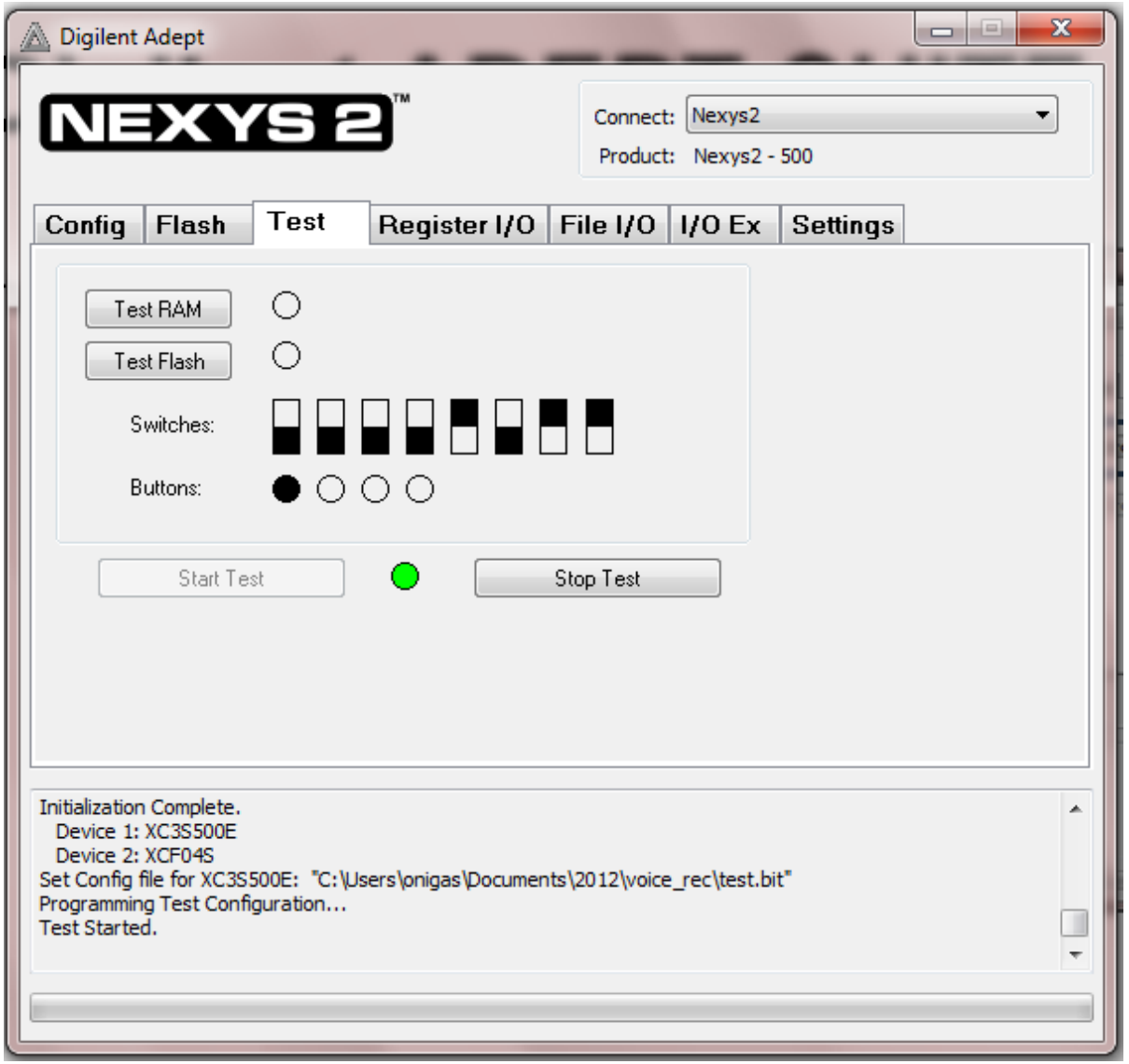
Configurarea FPGA:
.bit (fisierul de configurare generat)



Testarea plăcii

Start Test:

- RAM
- Flash
- Comutatoare
- Butoane
- LED-uri
- Afișaj cu 7 segmente



Mediul de dezvoltare

- Mediul de dezvoltare folosit în laborator pentru circuitele FPGA Xilinx se numește Xilinx ISE (Integrated Software Enviroment).
- Vom folosi varianta gratuită a programului Xilinx ISE 14.7 numit WebPack care este o variantă complet funcțională, singurele limitări fiind legate de tipul circuitului FPGA cu care se pot face implementările.
- Programul permite implementarea doar cu circuitele produse de firma Xilinx până la generația a 6-a (Spartan®-6, Virtex®-6, și respectiv CoolRunner™), singurele dispozitive din generația a 7-a ce pot fi totuși folosite fiind două circuite Artix-7: XC7A100T și XC7A200T.
- Implementarea cu circuitele FPGA din generația a 7-a se poate realiza folosind programul Xilinx Vivado.

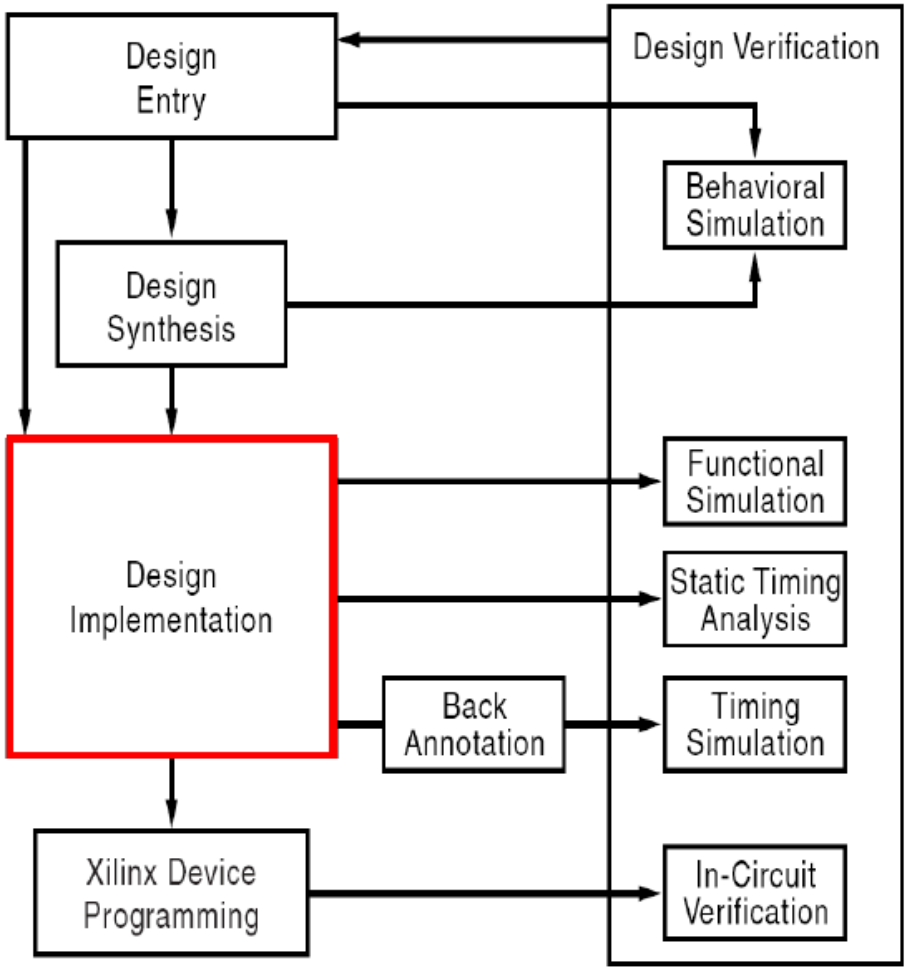
Instalarea programului Xilinx ISE

1. Programul poate fi descărcat de pe site-ul Xilinx website după o înregistrare prealabilă. (Xilinx ISE - Full Installer for Windows (TAR/GZIP - 6.18 GB)), sau folosind o copie locală a cestuia direct de aici.
2. Licența este gratuită și se poate obține de pe site-ul Xilinx product licensing după crearea unui cont sau poate fi descărcată de aici.
3. ATENTIE

- Softul nu funcționează automat pe variantele de 64 biți Windows 8, 8.1 si 10.
- Problema poate fi rezolvata așa cum este descris aici (În fișier există și două linkuri Youtube care prezintă aceasta soluție), sau preferabil se poate rezolva folosind un program care face automat aceste modificări și poate fi descărcat de aici.
- Pe site există și o variantă numită ISE 14.7 Windows 10 dar această variantă ISE 14.7 VM pentru Windows 10 se execută pe un mediu virtualizat, și anume pe o Mașină virtuală Oracle Linux.

Medii de dezvoltare

- Design Entry (Descrierea proiectului)
  - Xilinx Foundation ISE
  - Alte programe
    - Mentor Graphics: FPGA Advantage
    - Celoxica: DK Design Suite
- Sintetizarea: (Design Synthesis)
  - XST: Xilinx Synthesis Technology
  - Mentor: Leonardo Spectrum
  - Synplicity: Synplify Pro
  - Celoxica: DK Design Suite
- Simulare:
  - Mentor: Modelsim
  - Aldec: Active-HDL
  - Celoxica: DK Design Suite
- Verificare în Circuit:
  - Xilinx: ChipScope

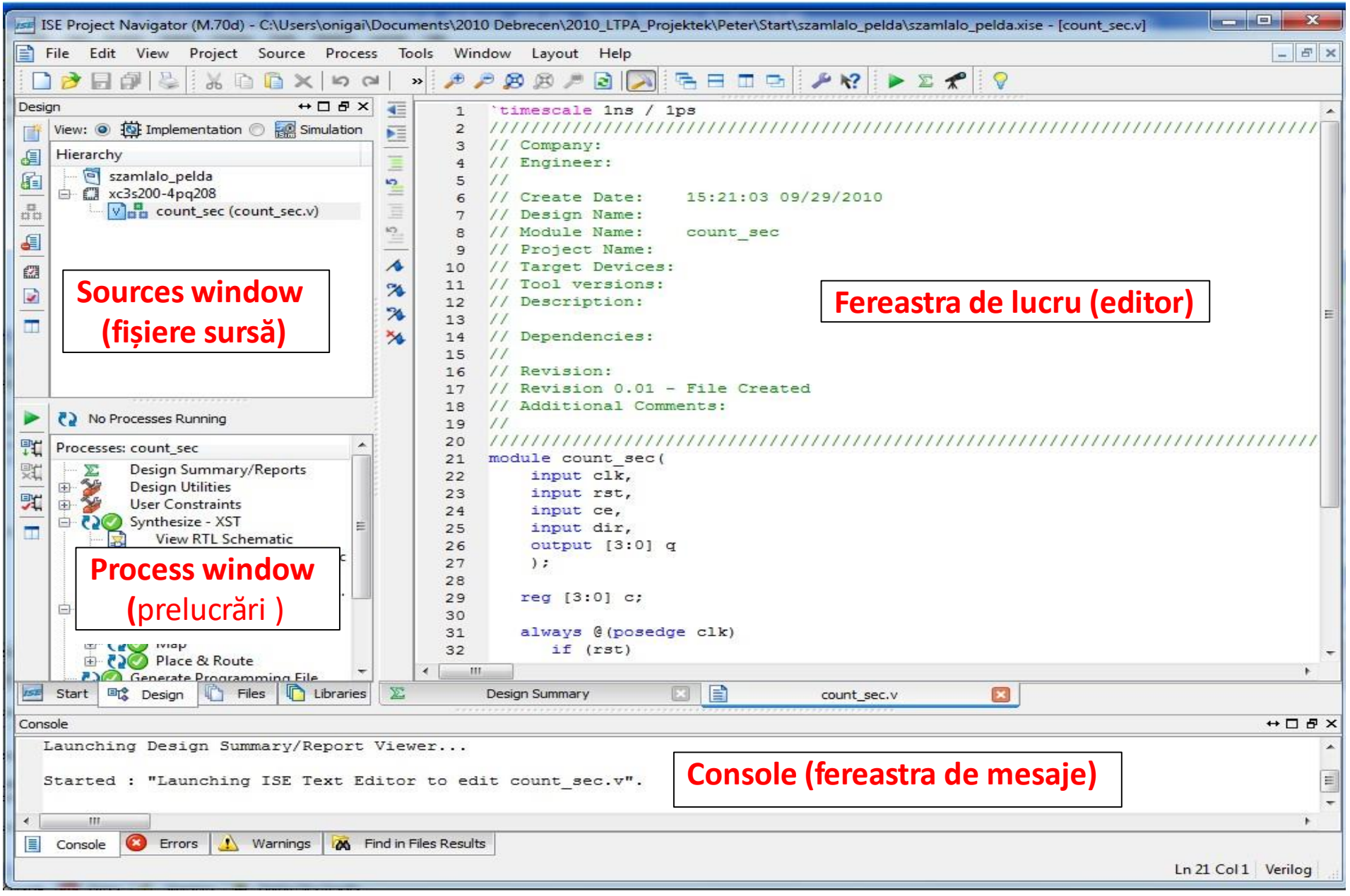


Implementare

TRANSLATE → MAP →PAR (place & route)

- **TRANSLATE**: merge, netlist (EDIF)
- **MAP** = mapare tehnologică
- **PAR** = plasare și rutare
- **Configurare**
  - **Bitstream** – fisier .bit
    - generare
    - JTAG
  - IMPACT

Prezentarea programului Xilinx ISE

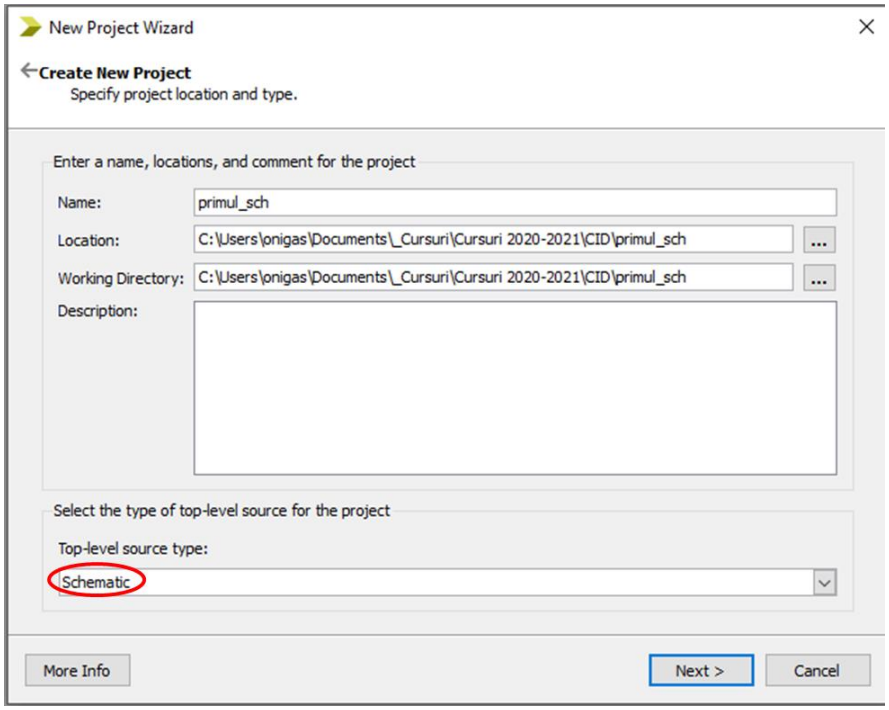


Lucrarea de laborator nr. 2

- Introducere în utilizarea programului XILINX ISE
- Implementarea porților logice – schematic
- Implementarea porților logice în limbaj Verilog

Crearea unui proiect

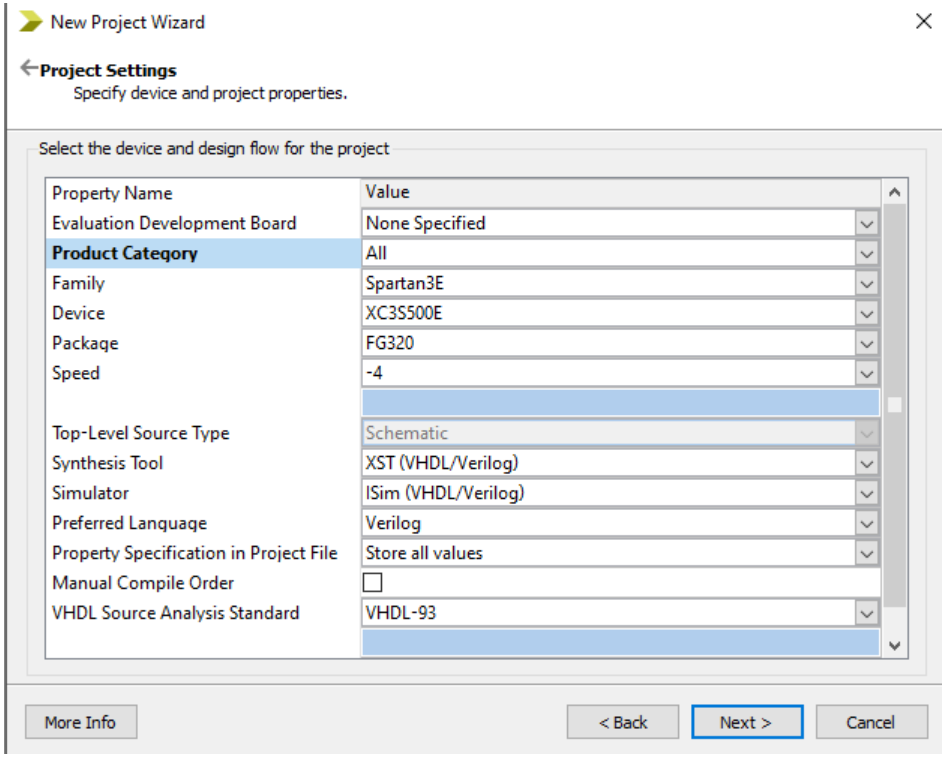
- **Pornirea programului:** C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Xilinx Design Tools\ISE Design Suite 14.7\ISE Design Tools\64-bit Project Navigator.
- **Proiect nou** (*File→New Project*) – programul creează câte un director nou pentru fiecare proiect.
- Numele proiectului „*primul_sch*”
- **Reguli privind directoarele și numele proiectului:**
  - Calea spre director si numele proiectului să nu conțină spații, caractere speciale și diacritice
  - Numele proiectului să nu înceapă cu un număr
  - Pentru înțelegerea mai ușoară a mesajelor de eroare este recomandat ca numele proiectului să fie diferit de numele fișierelor sursă
    - Fișier sursa nivelul ierarhic superior de tip **schematic**!



Setarea proprietăților FPGA

- La apăsarea butonului **Next** se deschide fereastra **Device Properties.** In coloana **Value** se aleg:

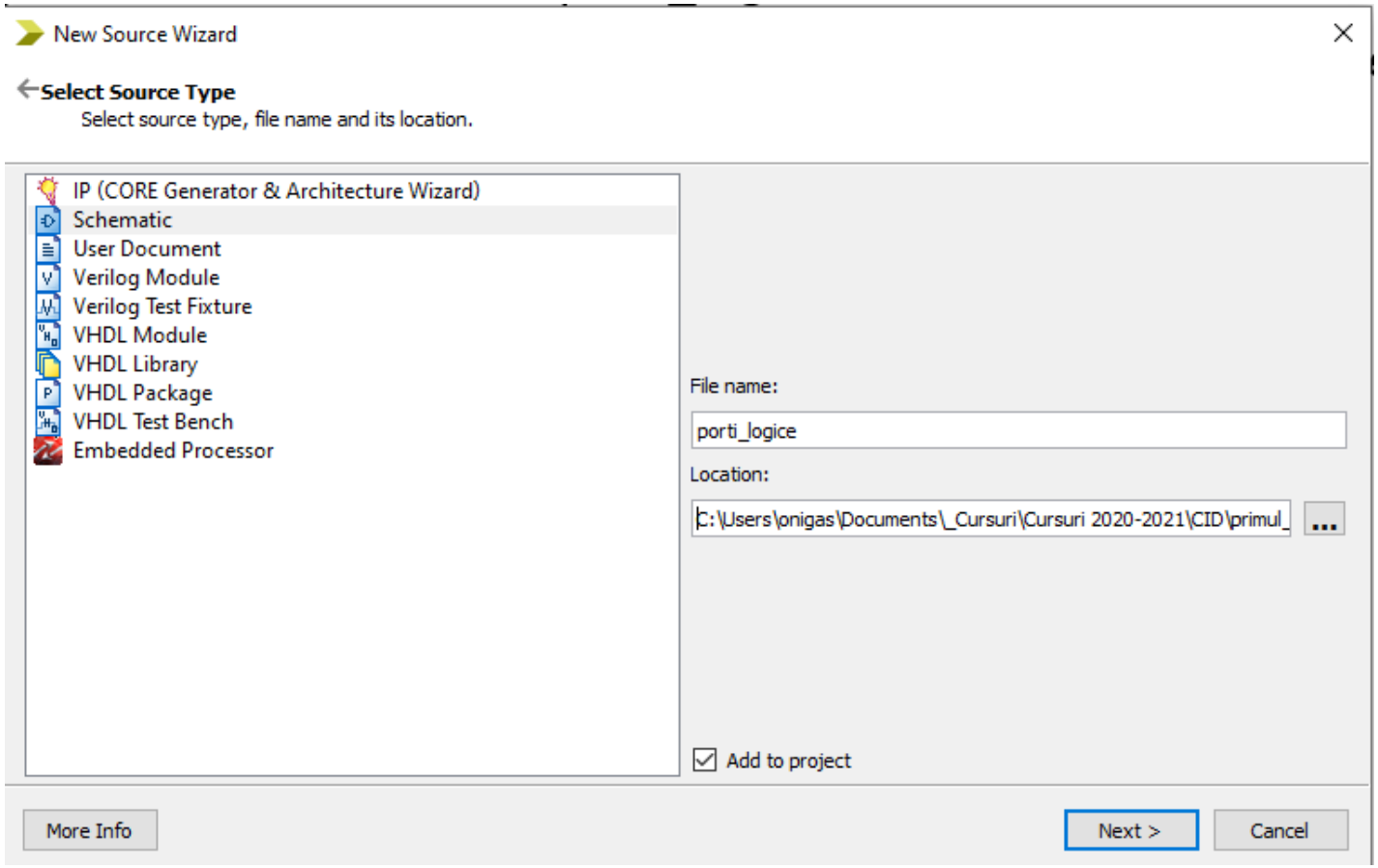
  - **Device Family:** Spartan3E
  - **Device:** xc3s500E
  - **Package:** FG320
  - **Speed Grade:** -4
  - **Synthesis Tool:** XST (VHDL/Verilog)
  - **Simulator:** ISim (VHDL/Verilog)



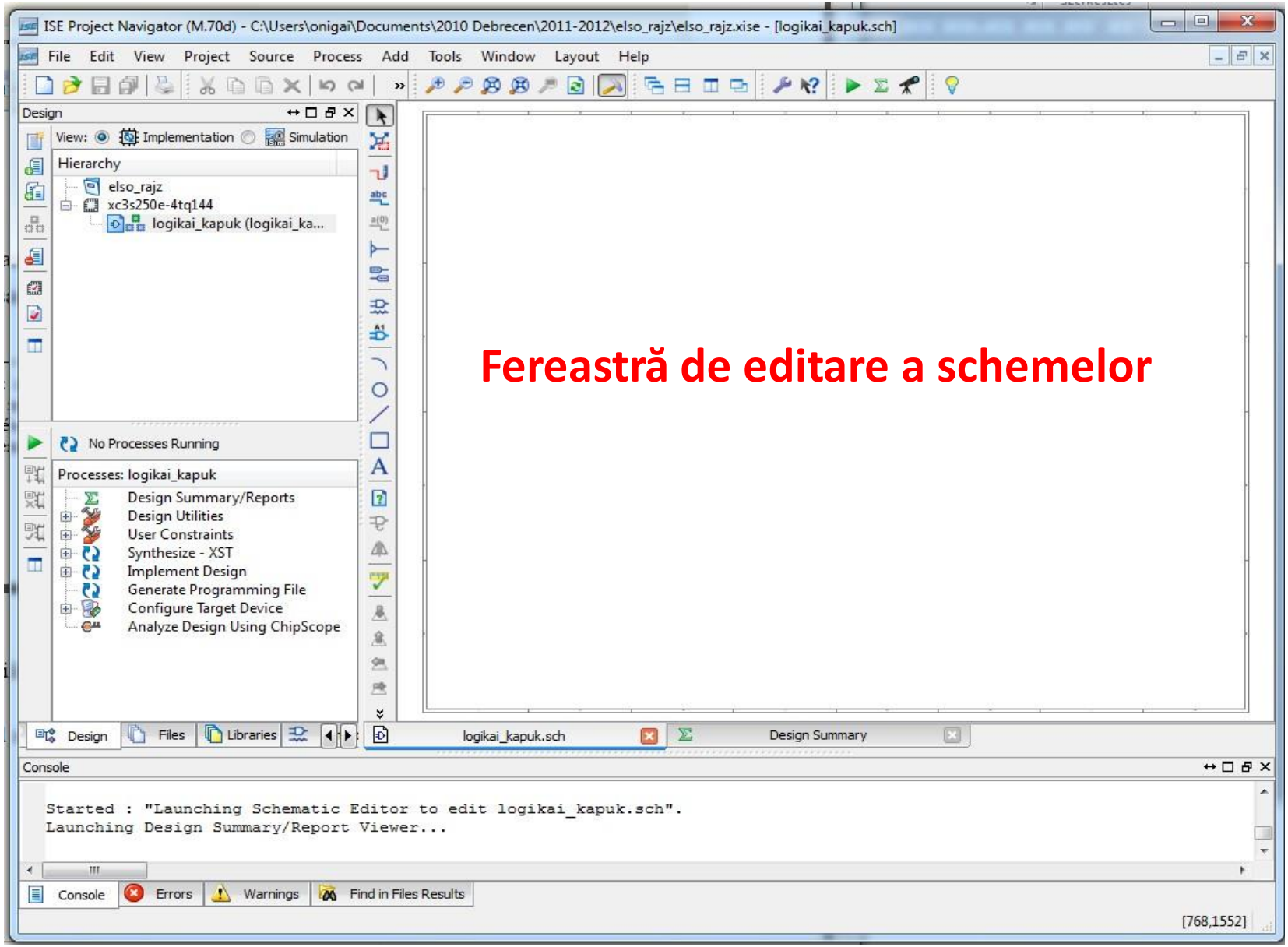
- Se apasă **Next,** apoi **Finish** și se creează un proiect gol.

Adăugarea unui fișier sursă

- Creearea unei surse noi: (*Project→New Source...*)!
- Tipul: **schematic**, numele: porti_logice!
- Daca vrem să adăugăm o sursă existentă: (*Project→Add Source...*)

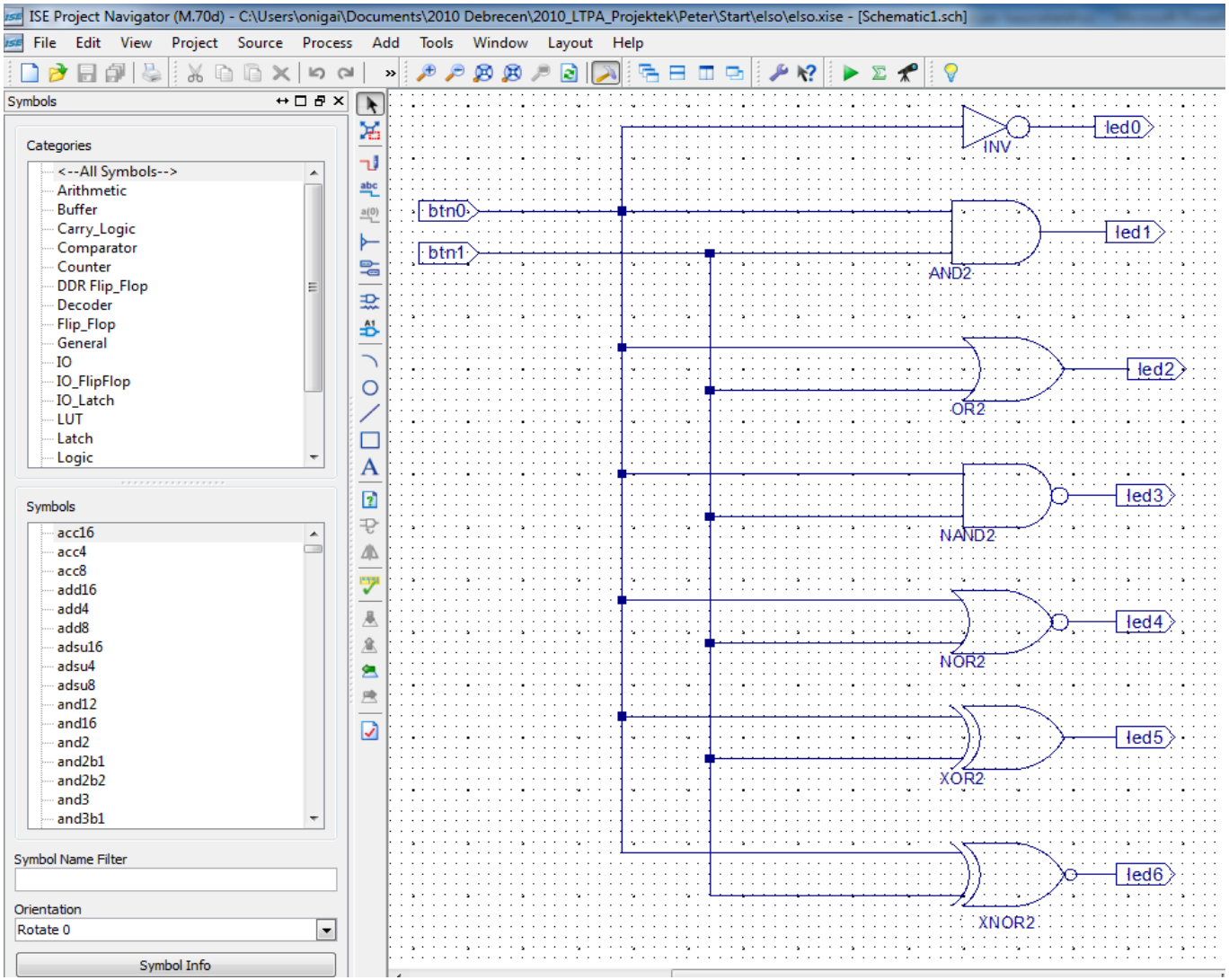


Proiecte pe bază de scheme



Tema propusă

- **Implementarea porților logice**



Bara de unelte pentru editarea schemei

- Add wire: desenarea unei legături
- Add Net Name: alocarea unui nume unei legături
- Rename Selected Bus: redenumirea magistralei selectate
- Add Bus Tap: adugarea unei conexiuni la o magistrală
- Adăugarea porturilor de intrare-ieșire (markere)
- Adugarea unei componente

După plasarea componentelor, a porturilor și efectuarea conexiunilor, se trece la redenumirea magistralele și markerele de I/O !

• **Adăugarea numelor de fire simple și magistrale**: *Add Net Name* (în fereastra stânga-jos *Options în rândul Name se scrie numele după care se face click pe firul care se dorește a fi denumit.*

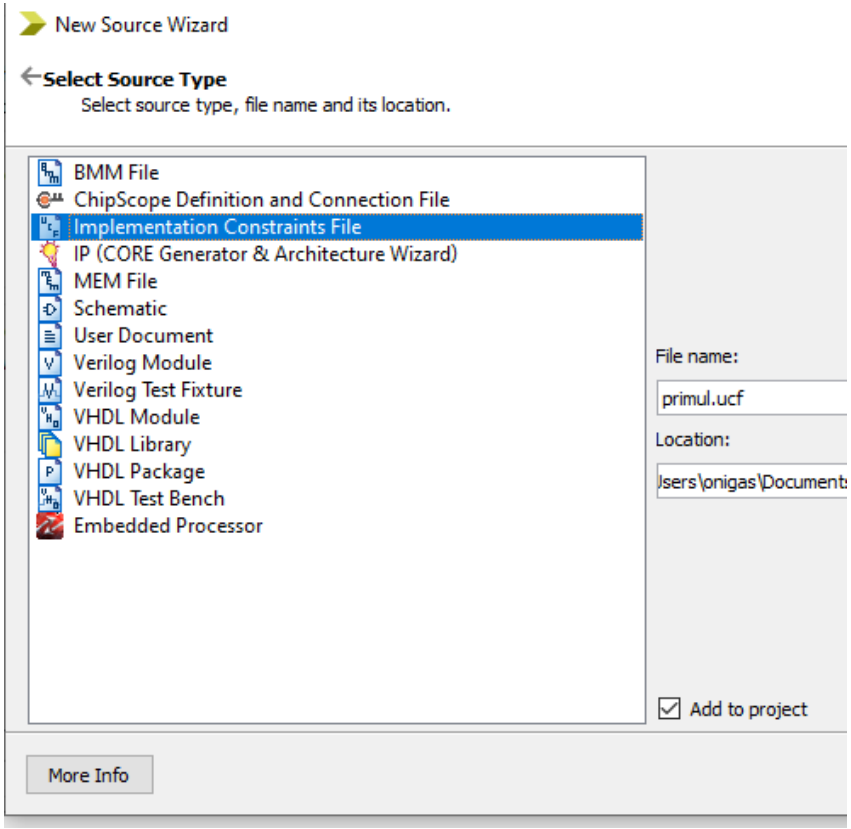
• **I/O marker**: dblu click pe I/O marker, în fereastra pop-up se modifică câmpul *Name*

.

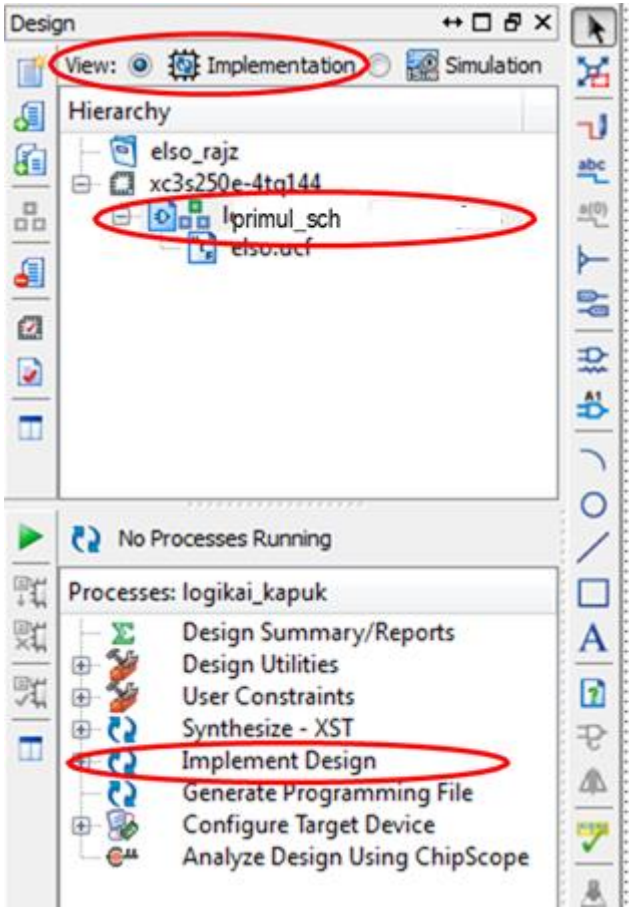
Constraints file

- Adăugarea fișierului de constrângeri (constraints file).
- Alegem **Project / New Source** și apoi **Implementation Constraint File**, alegem numele primul(.ucf).
- După apăsarea **Next/Finish** în fereastra **Sources** apare fișierul *primul.ucf*.
- Fișierul trebuie editat și completat cu datele înscrise pe placa.

```
NET "btn0" LOC = "B18";
NET "btn1" LOC = "D18";
NET "led0" LOC = "J14";
NET "led1" LOC = "J15";
NET "led2" LOC = "K15";
NET "led3" LOC = "K14";
NET "led4" LOC = "E17";
NET "led5" LOC = "P15";
NET "led6" LOC = "F4";
```



Implementarea proiectului



1. **Implementarea în FPGA:**

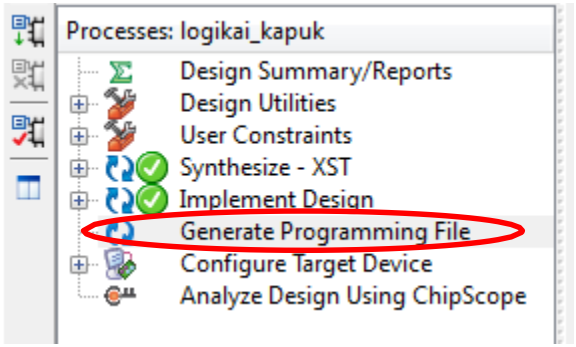
   (Implement Design),

   – **View** →*implementation*

   – **Hierarchy** → fișierul tot level

   – **Processes** → Implement Design

2. **Generarea fisierului de configurare**



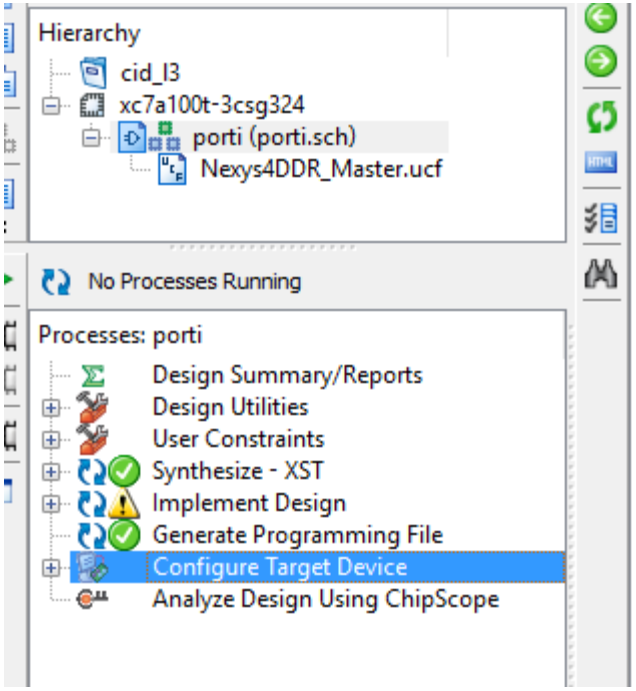
Configurarea FPGA

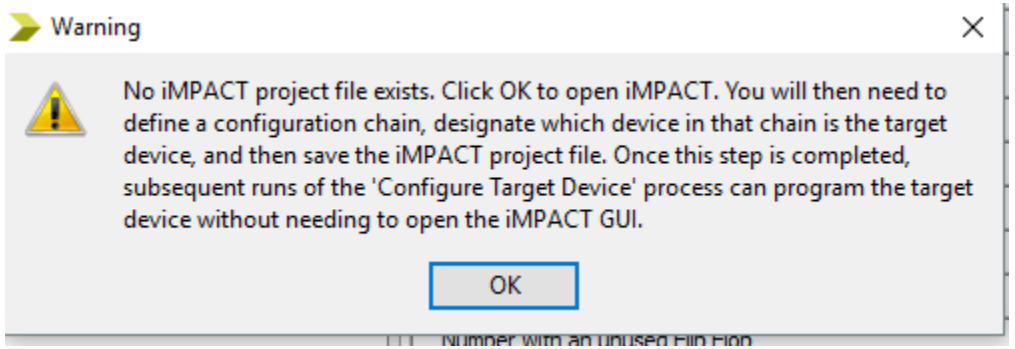
- Se face prin încărcarea în FPGA a fișierului primul_sch.*bit* creat anterior folosind:
  1. Folosind programul Impact care face parte din ISE (de preferat)

     sau folosind
  2. Programul Digilent Adept Suite (alternativă pentru cazul în care sunt probleme la încărcarea cu programul Impact)

Configurarea FPGA folosind IMPACT

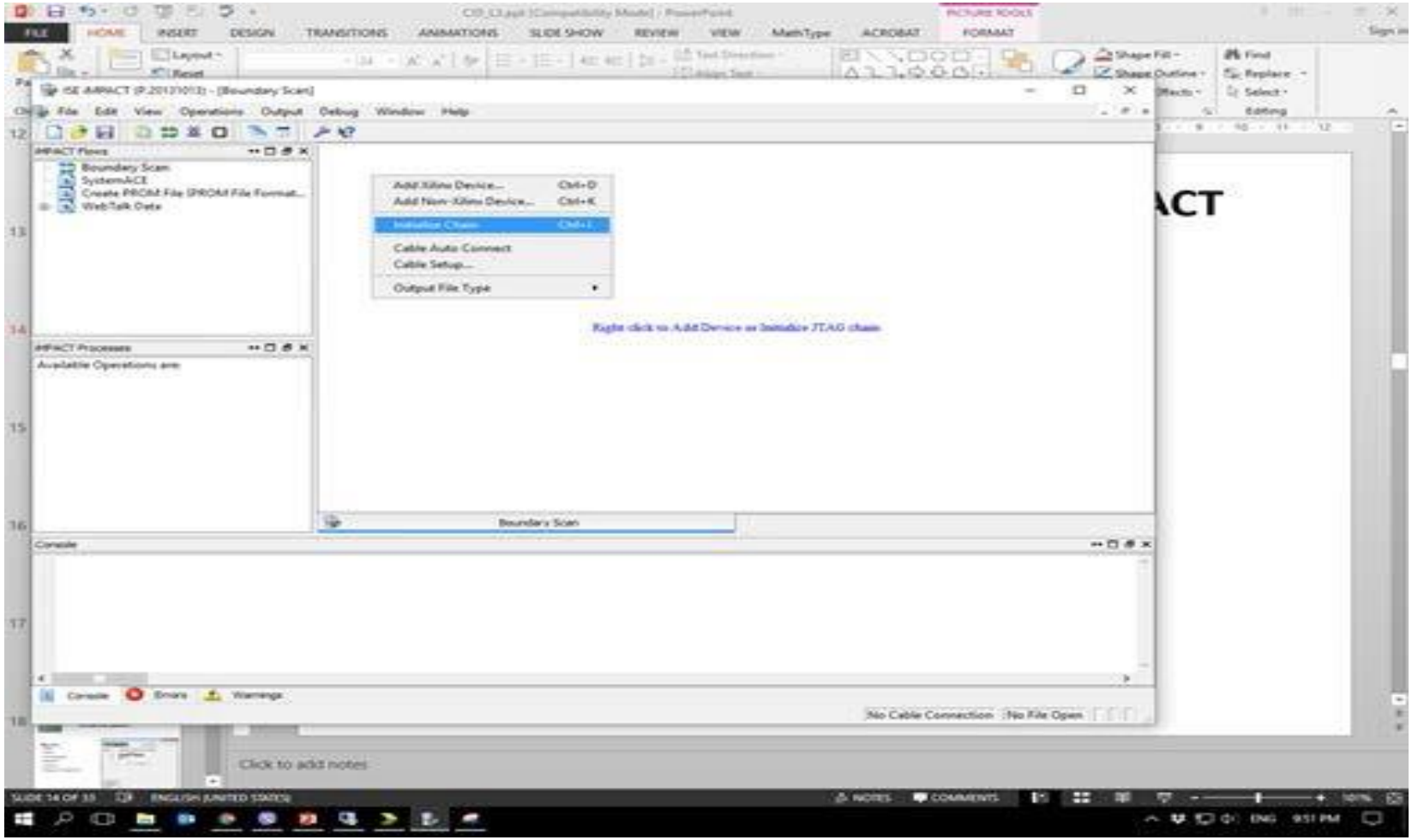
1. Configure Target Devices
2. OK





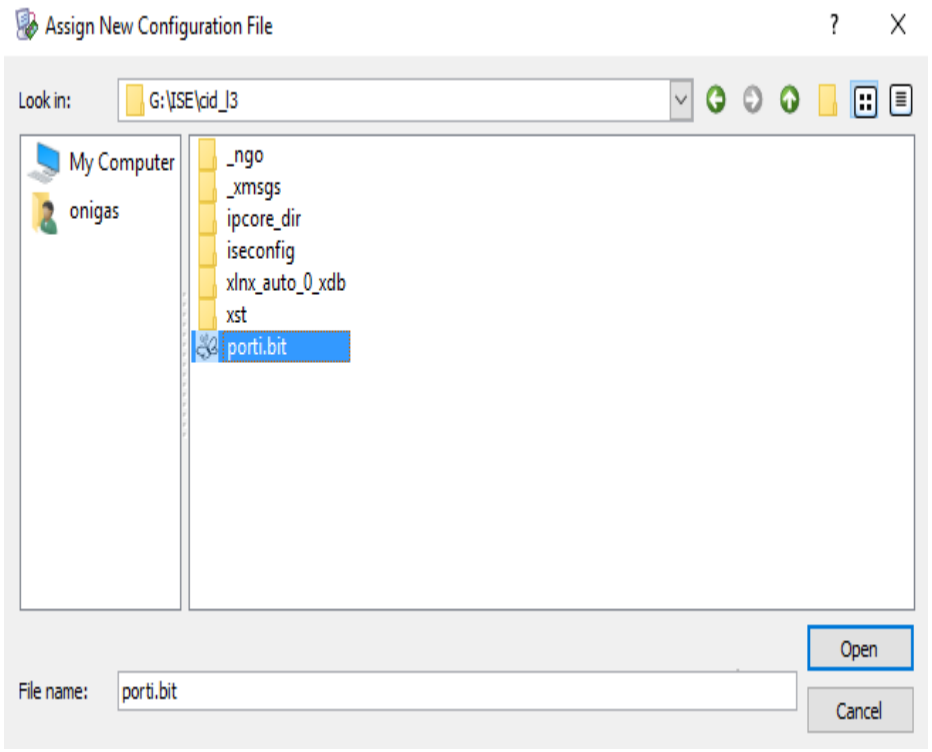
Configurarea FPGA folosind IMPACT (cont.)

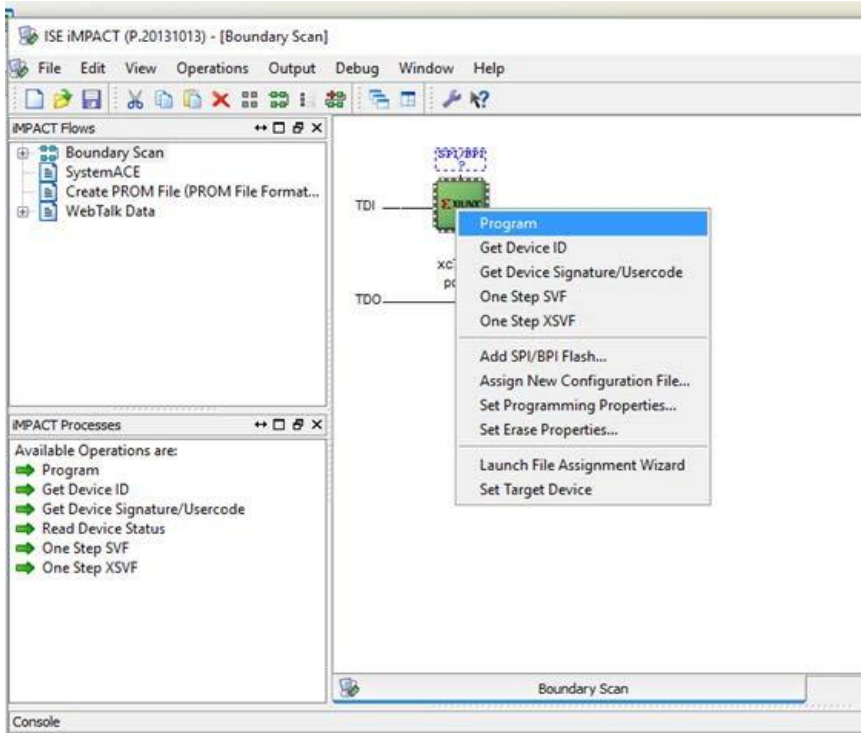
3. Boundary Scan (dublu click)
4. Click dreapta în fereastra Boundary scan
5. Initialize chain



Configurarea FPGA folosind IMPACT (cont.)

3. Assign new configuration file
4. Open, No, Ok
5. Click dreapta pe pătratul verde, Program





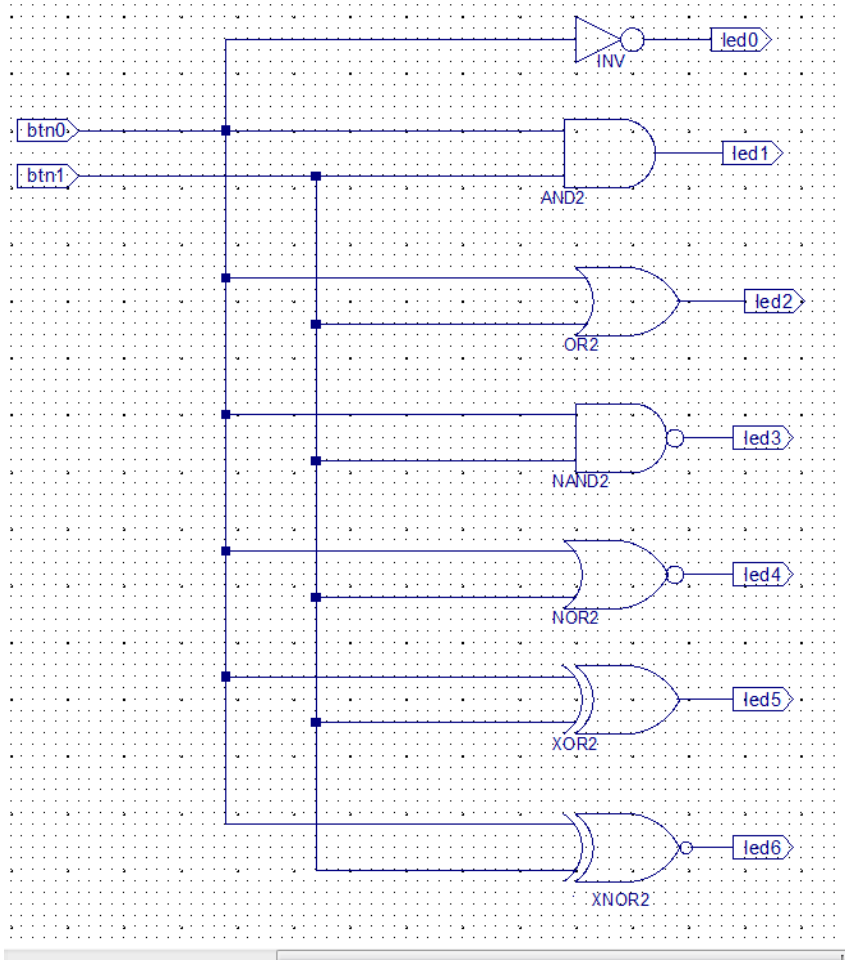
Rezultate

- Se apasă btn0 și btn1 pentru a creea toate cele 4 combinații posibile și se notează starea ledurilor
- Se verifică corectitudinea funcționării conform tabelelor de adevăr ale funcțiilor logice

btn0	btn1	led0 NOT	led1 AND	led2 OR	led3 NAND	led4 NOR	led5 XOR	led6 XNOR
0	0							
0	1							
1	0							
1	1							

Implementarea porților logice folosind limbajul Verilog

Descrierea într-un proiect de tip Schematic



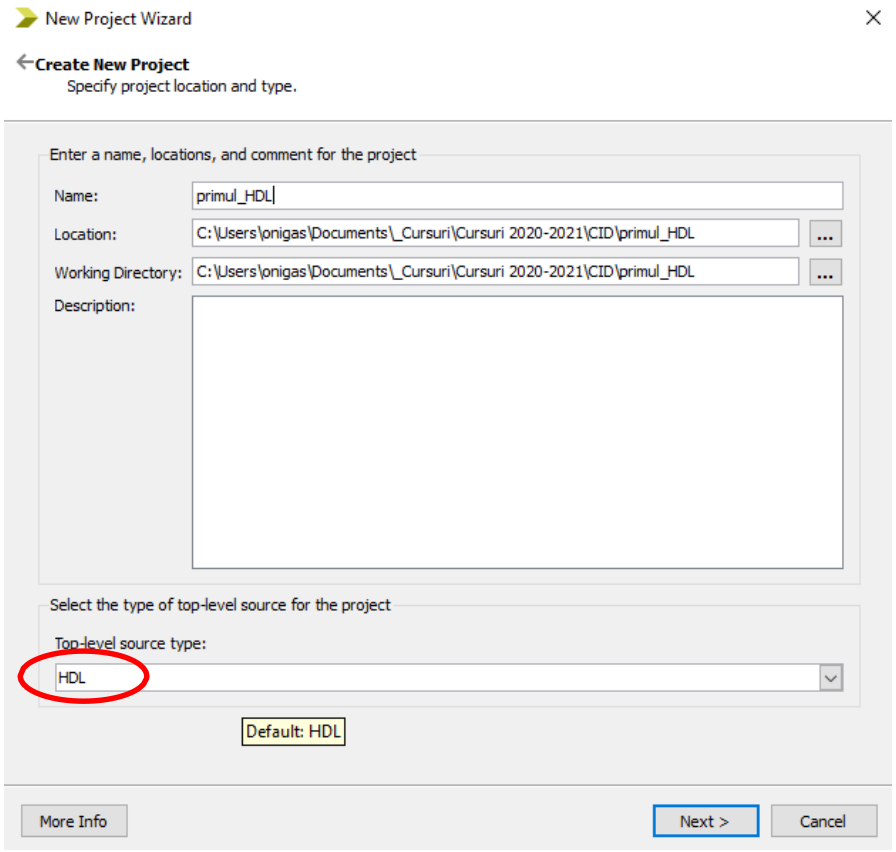
Descrierea într-un proiect de tip HDL (Verilog)

```
module ElsoHDL(
    input btn0,
    input btn1,
    output led0, led1, led2, led3, led4, led5, led6
    );

assign led0 = ~ btn0;
assign led1 = btn0 & btn1;
assign led2 = btn0 | btn1;
assign led3 = ~ (btn0 & btn1);
assign led4 = ~ (btn0 | btn1);
assign led5 = btn0 ^ btn1;
assign led6 = btn0 ~^ btn1;

endmodule
```

Crearea proiectului

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Xilinx Design Tools\ISE Design Suite 14.7\ISE Design Tools\64-bit Project Navigator.

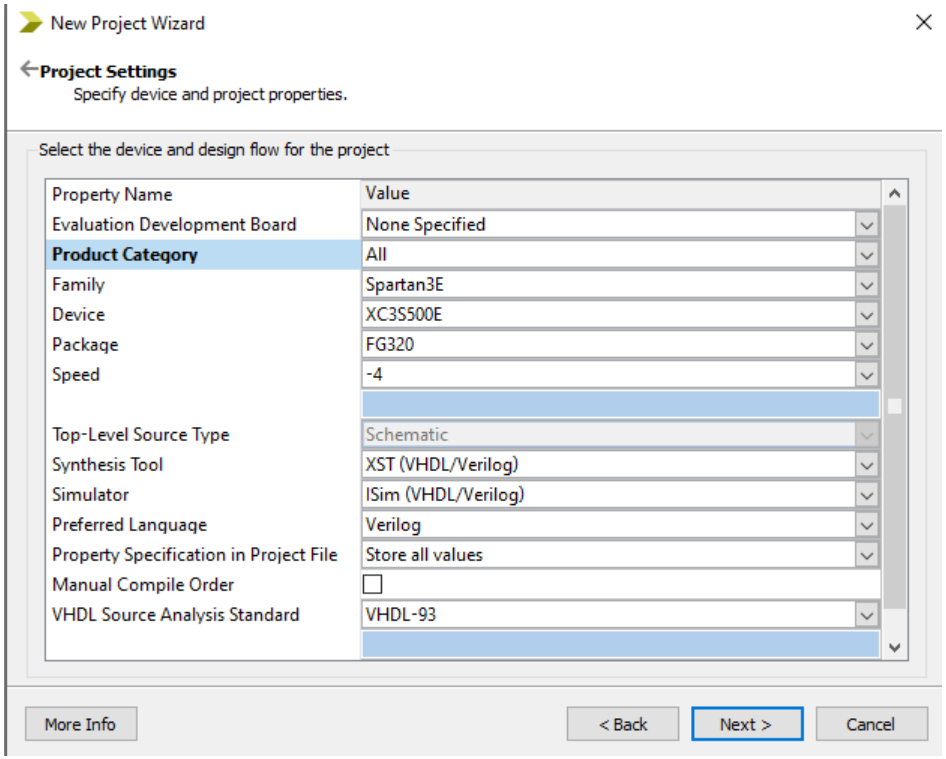
- *File → New Project* - *„primulHDL”*,
- **Fișierul top level: HDL**!



Setarea proprietăților FPGA

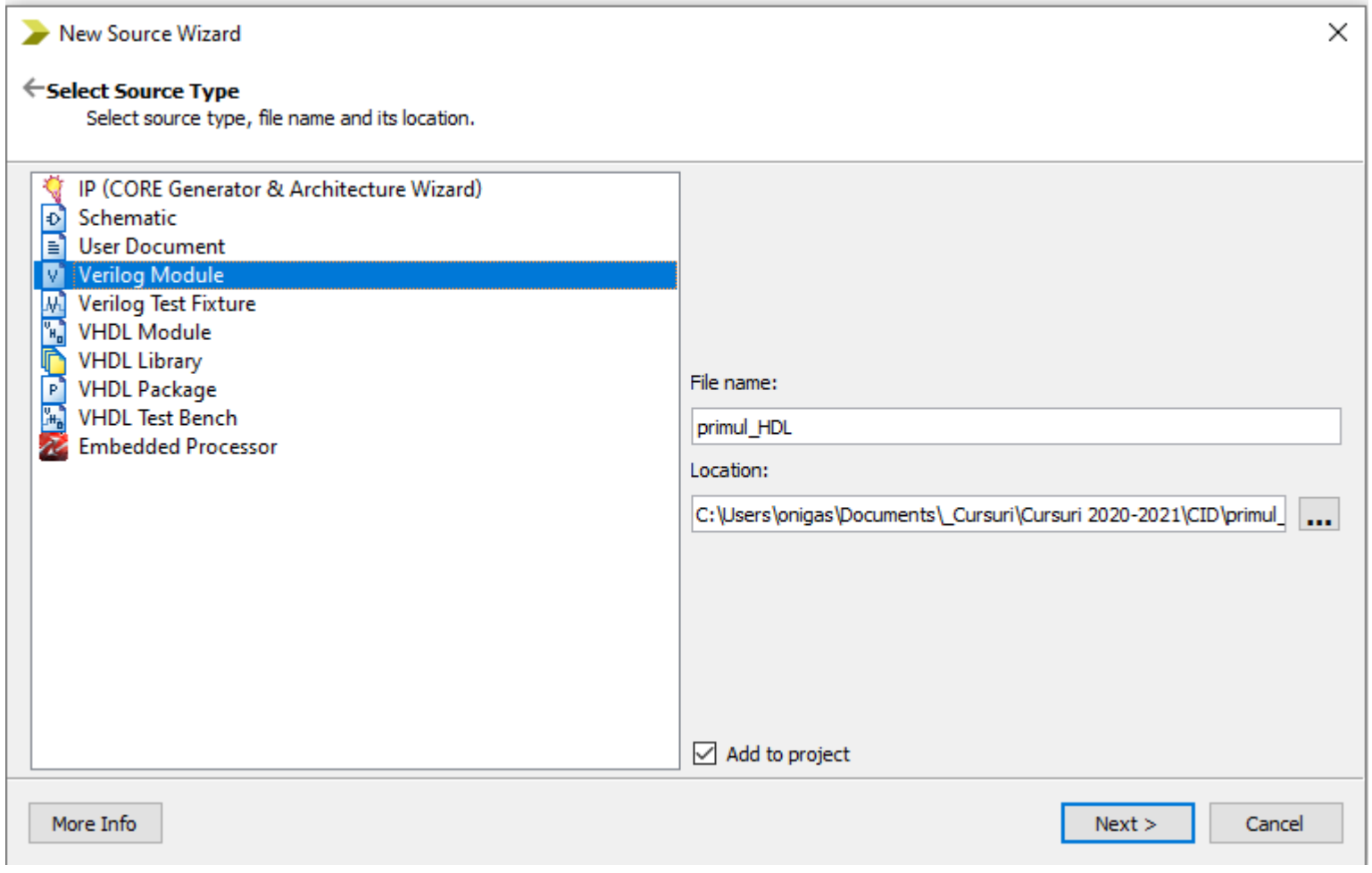
- **Device Family:** Spartan3E
- **Device:** xc3s500E
- **Package:** FG320
- **Speed Grade:** -4
- **Synthesis Tool:** XST (VHDL/Verilog)
- **Simulator:** ISim (VHDL/Verilog)

- **Next, Finish**.



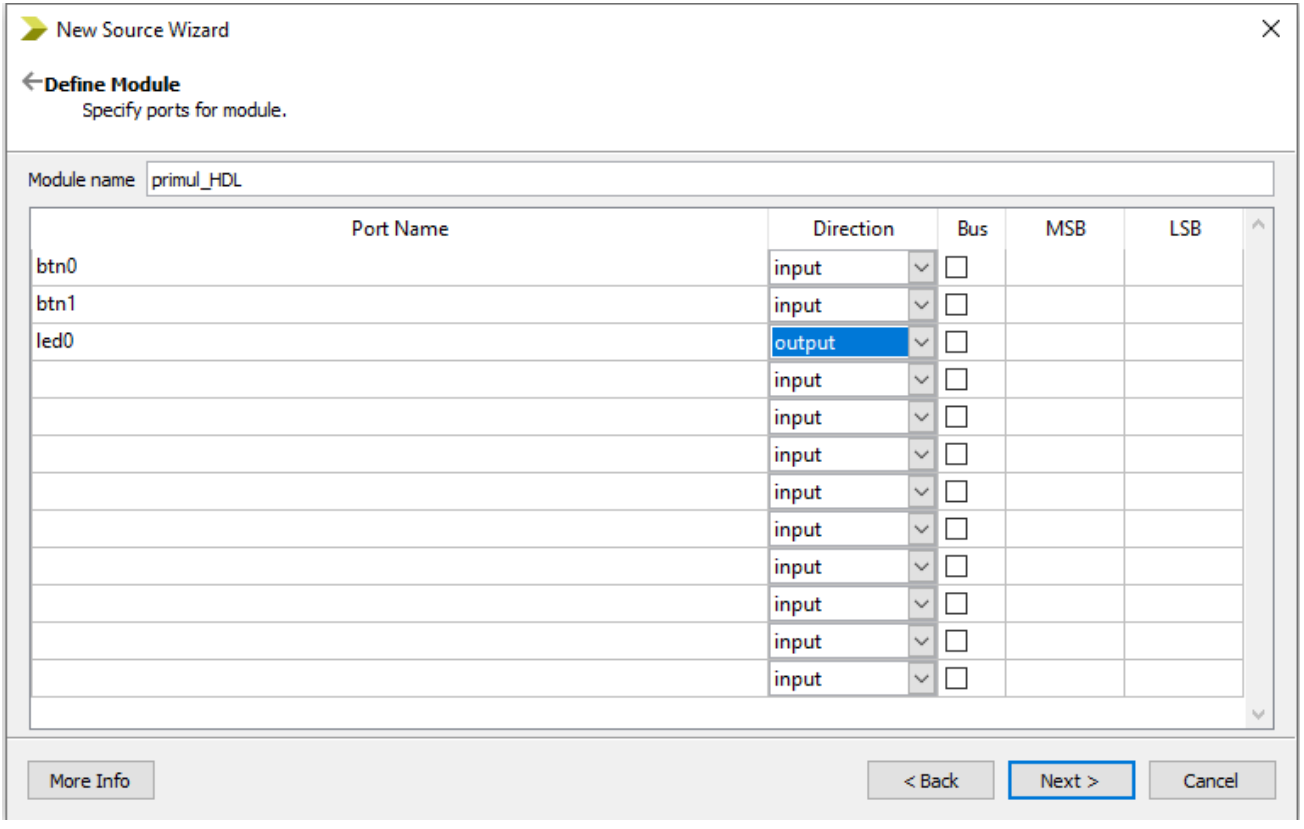
Adăugarea fișierului HDL

- *Project→New Source…*!
- Tipul: **Verilog Module**, nume primul_HDL!



Adăugarea porturilor

- Se poate face în acest meniu, precizând tipul intrare sau ieșire și în cazul în care este vorba de o magistrală (bus) și numărul de biți prin precizarea bitului cel mai semnificativ (MSB = Most significant bit) și a bitului cel mai puțin semnificativ (Least significant bit)
- Se poate și ulterior în codul ce va fi generat.
- În exemplul de mai jos vom adăuga doar o parte a porturilor urmând să completăm în codul Verilog generat



Fișierul HDL generat

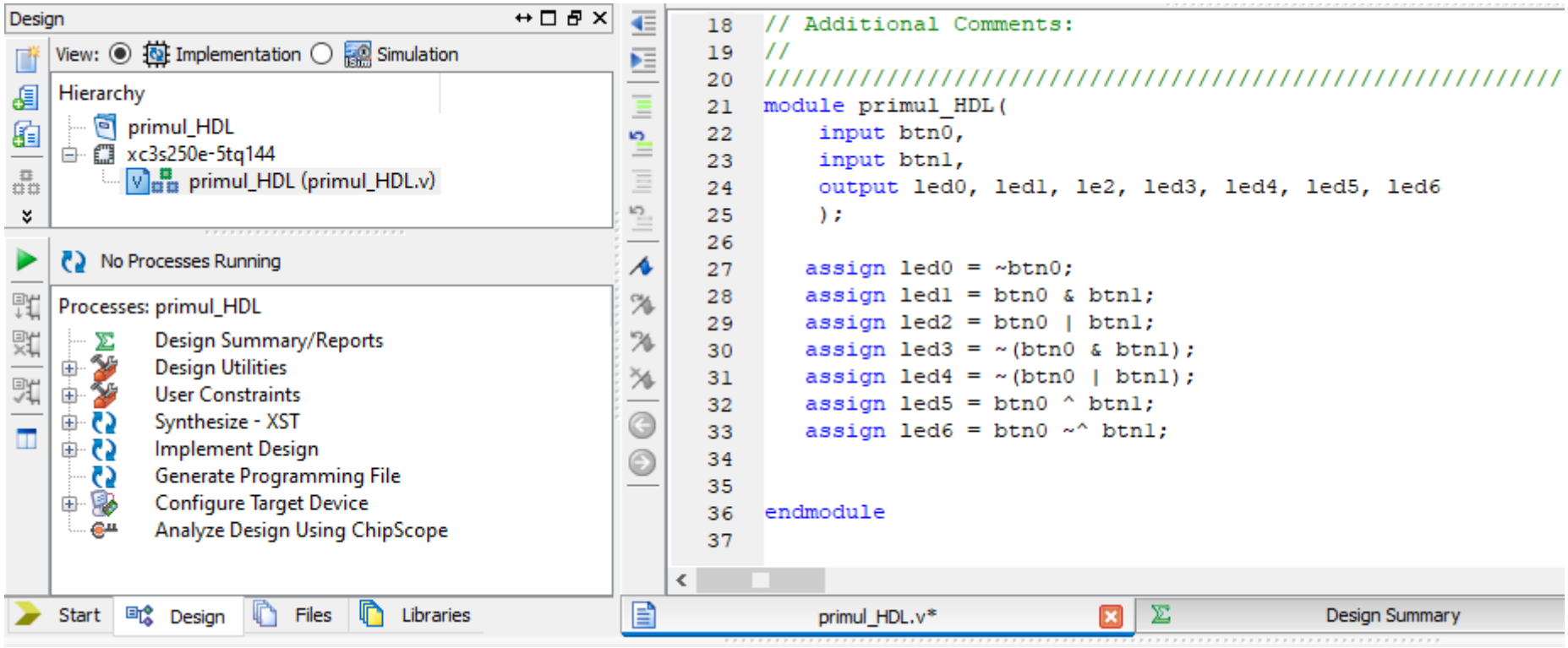
```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    11:34:01 10/24/2020
7  // Design Name:
8  // Module Name:    primul_HDL
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////
21 module primul_HDL(
22     input btn0,
23     input btn1,
24     output led0
25     );
26
27
28 endmodule
29
```

Editarea fișierului HDL generat

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    11:34:01 10/24/2020
7  // Design Name:
8  // Module Name:    primul_HDL
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////
21 module primul_HDL(
22     input btn0,
23     input btn1,
24     output led0
25     );
26
27
28 endmodule
29
```

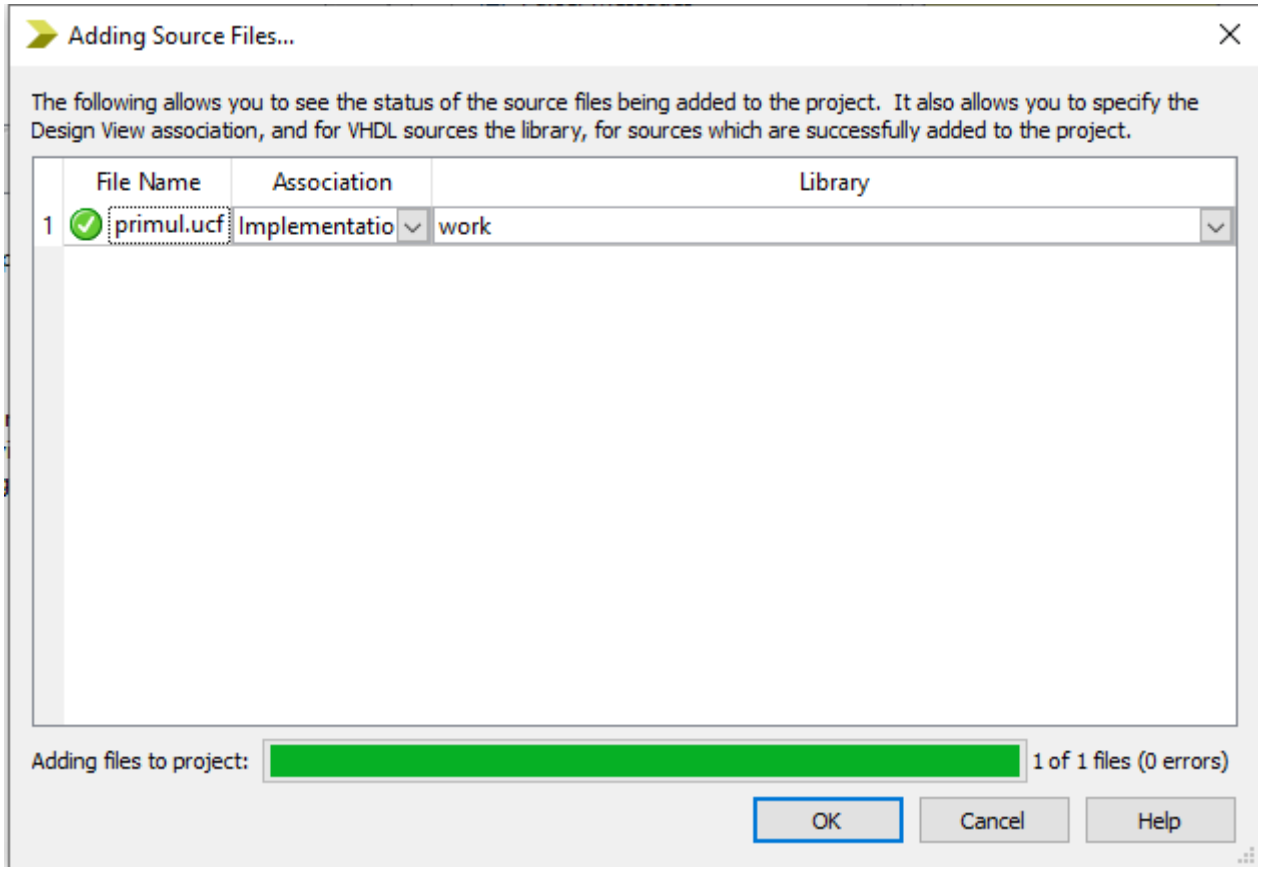
Aici se introduce codul care descrie circuitul

Descrierea completă în cod Verilog



Adăugarea constrângerilor

- **Project / Add Copy of Source** se caută fișierul primul.ucf din proiectul anterior.



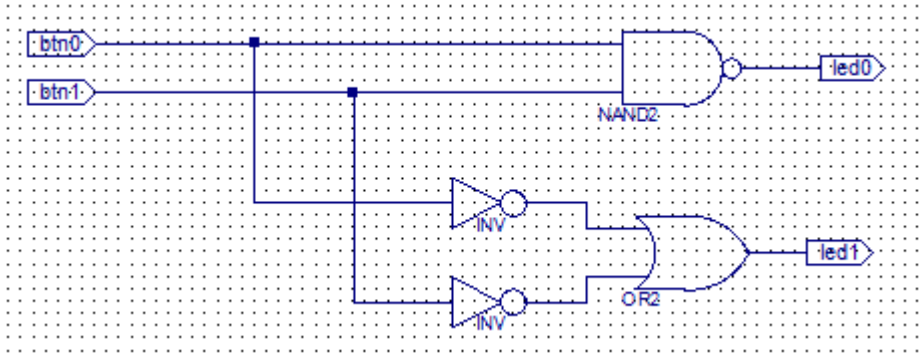
Rezultate

- Se apasă btn0 și btn1 pentru a creea toate cele 4 combinații posibile și se notează starea ledurilor
- Se verifică corectitudinea funcționării conform tabelelor de adevăr ale funcțiilor logice

btn0	btn1	led0 NOT	led1 AND	led2 OR	led3 NAND	led4 NOR	led5 XOR	led6 XNOR
0	0							
0	1							
1	0							
1	1							

Problemă propusă de tip „Schematic”

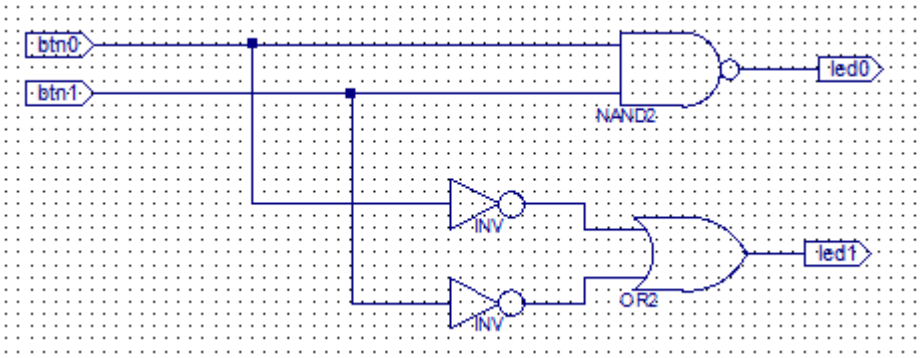
- Implementați circuitul de mai jos într-un proiect de tip schematic
- Încărcați fișierul de configurare și testați funcționarea circuitului
- Completați tabelul de adevăr pentru cele două funcții de ieșire



btn0	btn1	led0=(AB)’	led1=A’+B’
0	0		
0	1		
1	0		
1	1		

Problemă propusă de tip „HDL” (1)

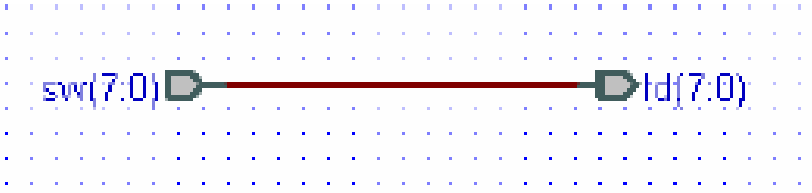
- Implementați circuitul de mai jos într-un proiect de tip schematic
- Încărcați fișierul de configurare și testați funcționarea circuitului
- Completați tabelul de adevăr pentru cele două funcții de ieșire



btn0	btn1	led0=(AB)’	led1=A’+B’
0	0		
0	1		
1	0		
1	1		

Problemă propusă HDL (2)

- Să se implementeze folosind limbajul Verilog circuitul prezentat în figura următoare



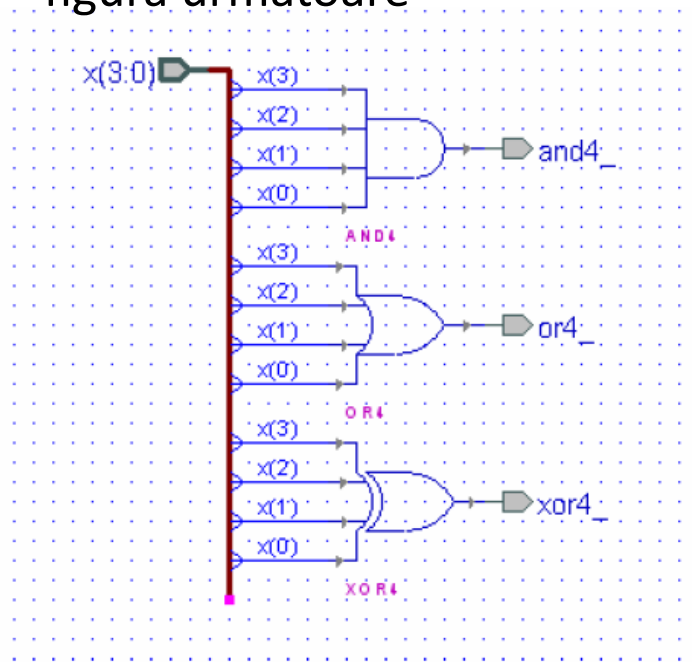

```
// Title         : sw2led2
module sw2led2 (
input wire [7:0] sw ,
output wire [7:0] ld
) ;

assign ld = sw;

endmodule
```

```
NET "sw<0>"  LOC= "G18";
NET "sw <1>" LOC="H18";
NET "sw <2>" LOC="K18";
NET "sw <3>" LOC="K17";
……
```

- Fișierul de constrângeri general pentru placă de dezvoltare cu toate perifericele conectate la FPGA se poate descărca de pe pagina Digilent, de aici: **Master UCF** For Nexys 2-500
- Rândurile care conțin pinii care se doresc a fi folosiți se decomentează și numele acestora se adaptează dacă este necesar pentru a corespunde exact cu numele porturilor din proiect. Programul face diferență între litere mici și litere mari
- **Observație**: In exemplul de mai sus porturile au fost definite ca magistrale de câte 8 biți
- Astfel sw <0> nu este același lucru cu sw0.
  - sw <0> este bitul cel mai puțin semnificativ al magistrale de 8 biți sw [7:0]
  - sw0 este un semnal de 1 bit și nu face parte dintr-o magistrala (bus)

Problemă propusă HDL (3)

- Să se implementeze folosind limbajul Verilog circuitul prezentat în figura următoare




```
module gates4b (
input [3:0] x ,
output and4_ ,
output or4_ ,
output xor4_
);
assign and4_ = &x;
assign or4_ = |x;
assign xor4_ = ^x;
endmodule
```

```
NET "x<0>" LOC=" P11 ";
NET "x<1>" LOC=" L3 ";
NET "x<2>" LOC=" K3 ";
NET "x<3>" LOC=" B4 ";
……
```

Lucrarea de laborator nr. 3

- Demonstrarea proprietăților algebrei Booleene cu ajutorul porților
  - Asociativitatea
  - Distributivitatea
  - Absorbția
  - Teorema lui De Morgan
- Implementarea funcțiilor AND, OR, XOR și NOR de 4 variabile
- Simularea folosind vectorii de test

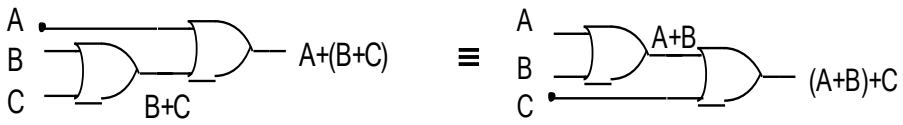
Tema 3_1a :

- Regulile de asociativitate-

- Creați un proiect nou
- Adăugați o sursă nouă de tip "schematic”
- Desenați circuitele prezentate

$$A \bullet (B \bullet C) = (A \bullet B) \bullet C$$

$$A + (B + C) = (A + B) + C$$

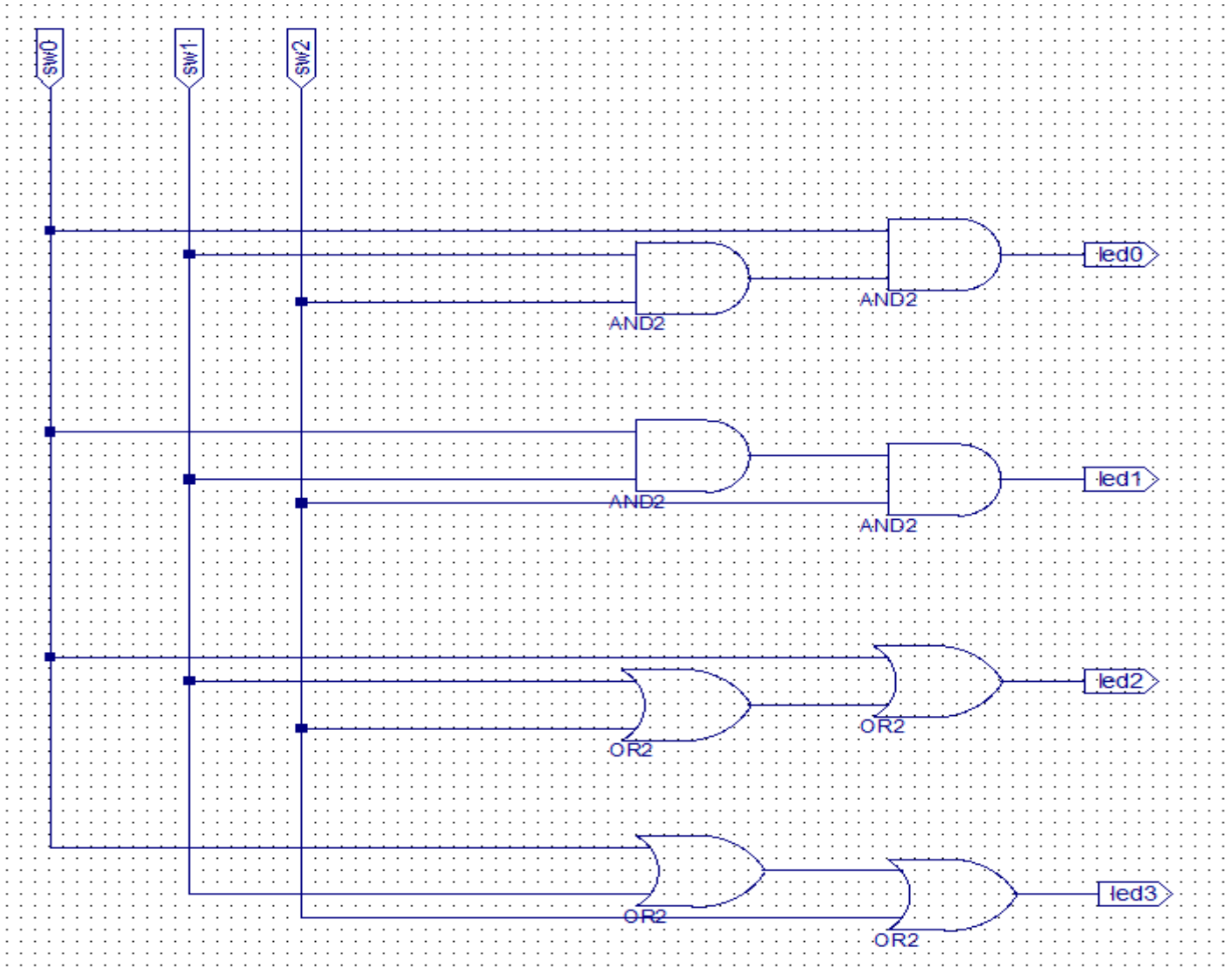


- Adăugați fișierul de constrângeri Nexys.UCF
  - Legați intrările celor 4 circuite (A, B, C) la switch-urile de pe placă
    - sw0 -> A; sw1 -> B; sw2->C
  - Legați ieșirile celor 4 circuite la câte un led

- Generați fișierul de configurare și încărcați-l în placă
- Testați funcționarea circuitelor
- Notați concluziile în raportul lucrării de laborator

Implementarea funcțiilor de 3 variabile

- Regulile de asociativitate-



Rezultate tema 3_1a

- Folosind switch-urile sw0, sw1 și sw2 generați toate cele 8 combinații posibile și notați stările corespunzătoare ale ledurilor în tabelul următor

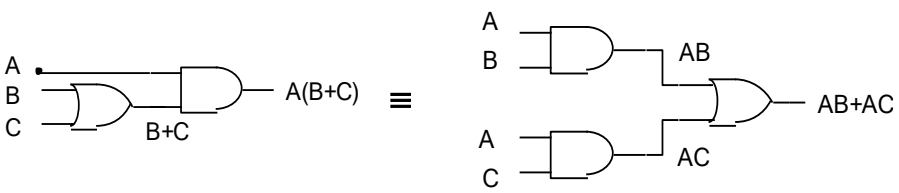
sw0	sw1	sw2	led0 A(BC)	led1 (AB)C	led2 A+(B+C)	led3 (A+B)+C
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Tema 3_1b:

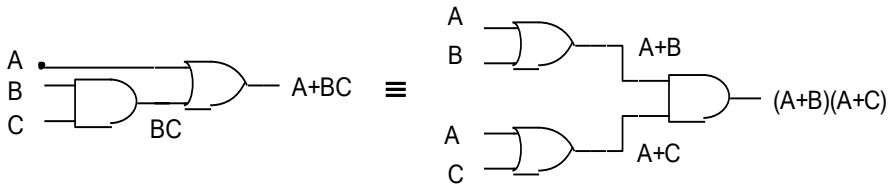
Regulile de distributivitate

- Creați un proiect nou
- Adăugați o sursă nouă de tip "schematic”
- Desenați circuitele prezentate

$$A \bullet (B + C) = A \bullet B + A \bullet C$$



$$A + B \bullet C = (A + B) \bullet (A + C)$$



- Adăugați fișierul de constrângeri Nexys.UCF
  - Legați intrările celor 4 circuite (A, B, C) la switch-urile *de pe placă*
    - sw0 -> A; sw1 -> B; sw2->C
  - Ieșirile celor 4 circuite legați-le la câte un led
- Generați fișierul de configurare și încărcați-l în placă
- Testați funcționarea circuitelor
- Notați concluziile în raportul lucrării de laborator

Rezultate tema 3_1b

- Folosind switch-urile sw0, sw1 și sw2 generați toate cele 8 combinații posibile și notați stările corespunzătoare ale ledurilor în tabelul următor

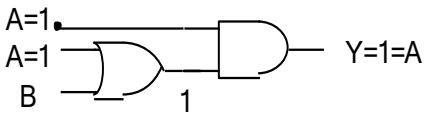
sw0	sw1	sw2	led0 A(B+C)	led1 AB+AC	led2 A+BC	led3 (A+B)+(A+C)
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

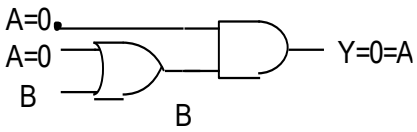
Tema 3_1c:

- Legile de absorbție -

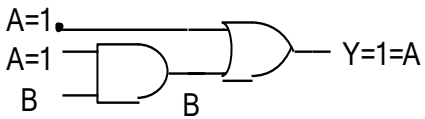
- Creați un proiect nou
- Adăugați o sursă nouă de tip "schematic”
- Desenați circuitele prezentate

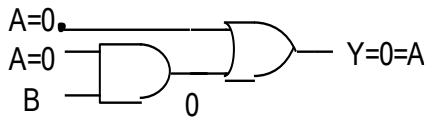
$$A \bullet (A + B) = A$$





$$A + A \bullet B = A$$





- Adăugați fișierul de constrângeri Nexys.UCF
  - Legați intrările celor 4 circuite (A, B) la switch-urile *de pe placă*
    - sw0 -> A; sw1 -> B;
  - Ieșirile celor 4 circuite legați-le la câte un led

- Generați fișierul de configurare și încărcați-l în placă
- Testați funcționarea circuitelor
- Notați concluziile în raportul lucrării de laborator

Rezultate tema 3_1c

- Folosind switch-urile sw0 și sw1 generați toate cele 4 combinații posibile și notați stările corespunzătoare ale ledurilor în tabelul următor.

sw0	sw1	led0 A(A+B)	led1 A+AB
0	0		
0	1		
1	0		
1	1		

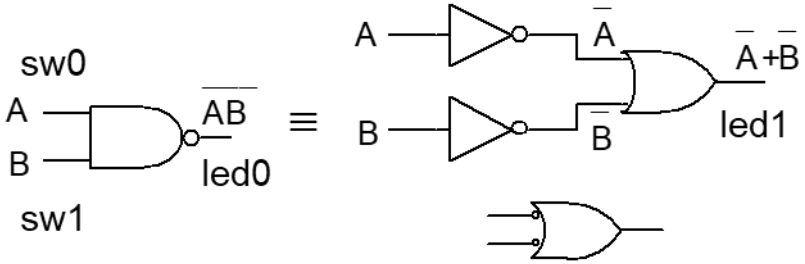
Tema 3_2a

Teoremele lui De Morgan pentru 2 variabile

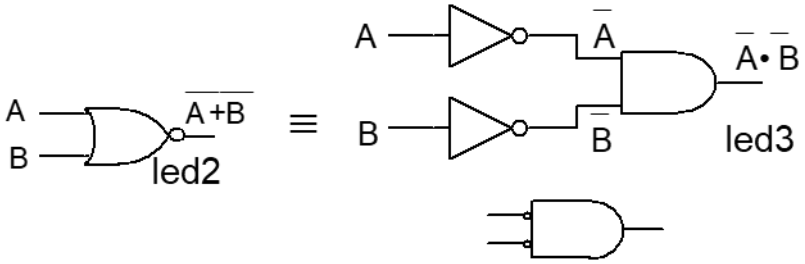
$$\overline{A \bullet B} = \overline{A} + \overline{B}$$

$$\overline{A + B} = \overline{A} \bullet \overline{B}$$

- Creați un proiect nou
- Adăugați o sursă nouă de tip "schematic"
- Desenați circuitele prezentate



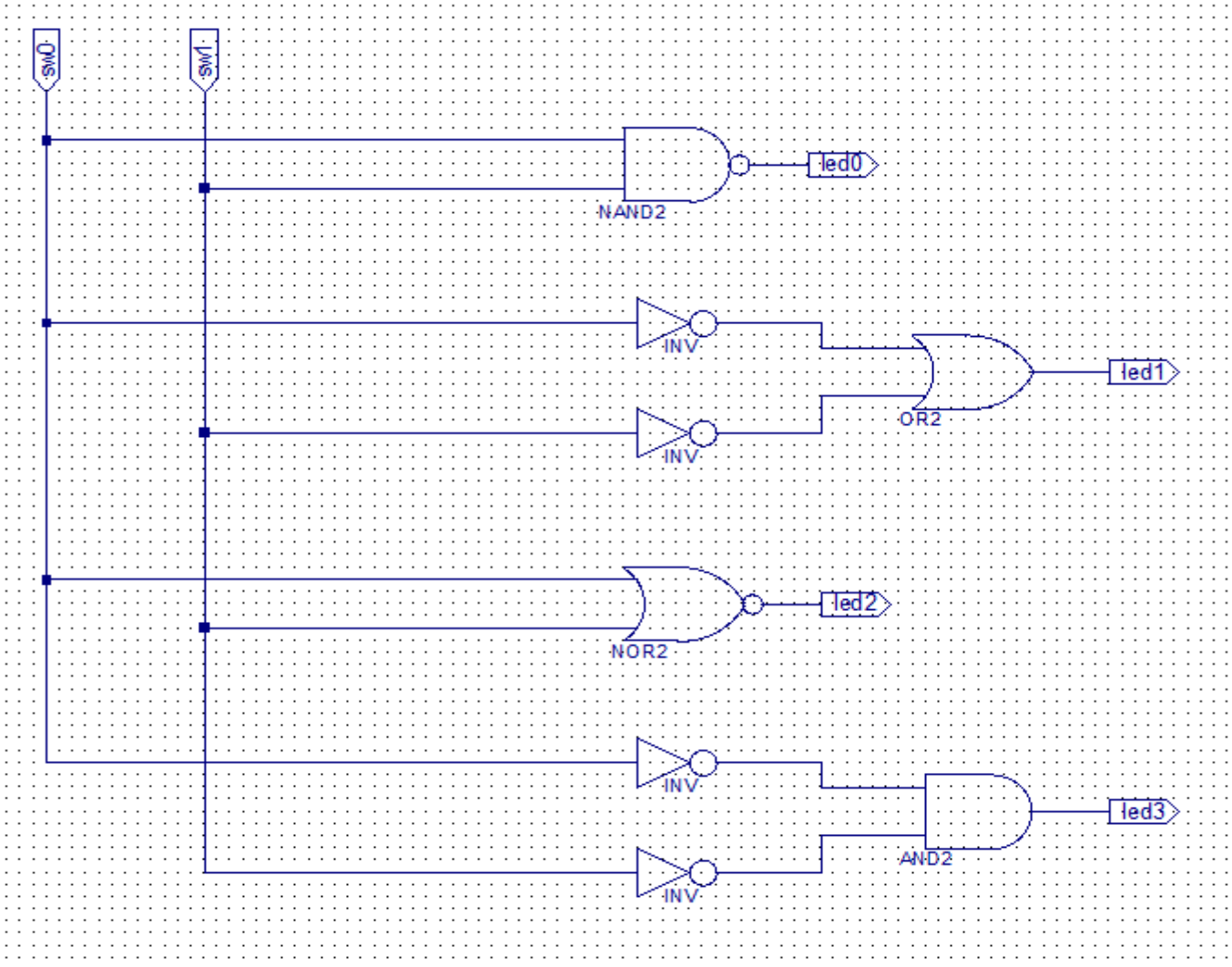
- Adăugați fișierul de constrângeri Nexys.UCF
  - Legați intrările celor 4 circuite (A, B) la switch-urile *de pe placă*
    - sw0 -> A; sw1 -> B;
  - Legați ieșirile celor 4 circuite la câte un led



- Generați fișierul de configurare și încărcați-l în placă
- Testați funcționarea circuitelor
- Notați concluziile în raportul lucrării de laborator

Tema 3_2a

Teoremele lui De Morgan pentru 2 variabile



Rezultate tema 3_2a

- Folosind switch-urile sw0 și sw1 generați toate cele 4 combinații posibile și notați stările corespunzătoare ale ledurilor în tabelul următor.

sw0	sw1	led0 /(AB)	led1 /A+/B	led2 /(A+B)	led3 /A*/B
0	0				
0	1				
1	0				
1	1				

Tema 3_2b

Teoremele lui De Morgan pentru 3 variabile

$$\overline{A \bullet B \bullet C} = \overline{A} + \overline{B} + \overline{C}$$

$$\overline{A + B + C} = \overline{A} \bullet \overline{B} \bullet \overline{C}$$

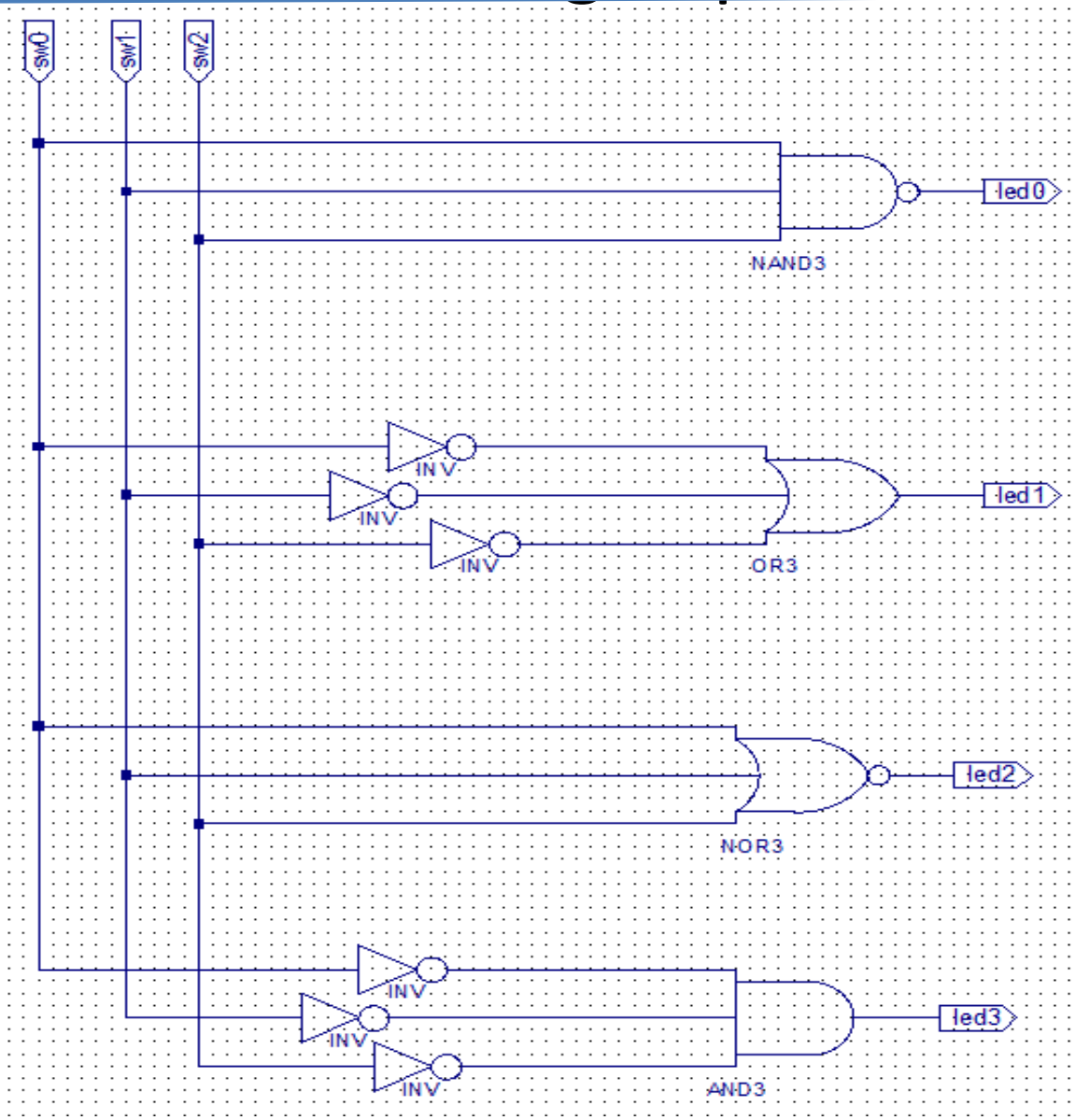
- Creați un proiect nou
- Adăugați o sursă de tip "schematic"
- Desenați circuitele prezentate

- Adăugați fișierul de constrângeri
- Legați intrările celor 4 circuite (A, B, C) la switch-urile *de pe placă*
  - sw0 -> A; sw1 -> B; sw2->C
  - Ieșirile celor 4 circuite legați-le la câte un led

sw0	sw1	sw2	led0 /(ABC)	led1 /A+/B+/C	Led2 /(A+B+C)	led3 /A*/B*/C
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

- Generați fișierul de configurare și încărcați-l în placă
- Folosind switch-urile sw0 și sw1 generați toate cele 4 combinații posibile și notați stările corespunzătoare ale ledurilor în tabel.
- Notați concluziile în raportul lucrării de laborator

Teoremele lui De Morgan pentru 3 variabile



Tema 3_2c (opțional)

Generalizarea teoremelor lui De Morgan

$$X = \overline{A \bullet B + A \bullet \overline{C} + ABC} = \overline{A \bullet B} \bullet \overline{A \bullet \overline{C}} \bullet \overline{ABC}$$

$$Y = \overline{(A \bullet B + A \bullet \overline{C}) \bullet (ABC + \overline{B}C)} = \overline{A \bullet B + A \bullet \overline{C}} + \overline{ABC + \overline{B}C}$$

- Creați un proiect nou
- Adăugați o sursă de tip "schematic”
- Desenați circuitele prezentate
- Adăugați fișierul de constrângeri
- Legați intrările celor 4 circuite (A, B, C) la switch-urile *de pe placă*
  - sw0 -> A; sw1 -> B; sw2->C
- Ieșirile celor 4 circuite legați-le la câte un led

sw0	sw1	sw2	led0 X1	led1 X2	led2 Y1	led3 Y2
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

- Generați fișierul de configurare și încărcați-l în placă
- Folosind switch-urile sw0 și sw1 generați toate cele 4 combinații posibile și notați stările corespunzătoare ale ledurilor în tabel.
- Notați concluziile în raportul lucrării de laborator

Tema 3_3a: Implementarea funcțiilor de 4 variabile

- **Implementarea funcțiilor AND, OR, XOR și NOR de 4 variabile**
  - Intrări: comutatoarele sw0-sw3
  - Ieșiri: ledurile LED0-LED3
- **Simulare folosind vectori de test generați cu funcția “for”**

Urmați pașii de proiectare prezentați în lucrarea de laborator precedentă

- Start ISE, creați un proiect nou
- Adăugați o sursă nouă de tip Verilog Lab3_3a.v. Editați fișierul adăugând funcționalitatea dorită
- Adăugați o copie a fișierului: Nexys4.UCF, adaptați în mod corespunzător intrările și ieșirile
- Efectuați simulare funcțională a circuitului
- Generați fișierul de configurare, încărcați-l în placă de test și testați-l.

Tema 3_3a

- **Specificarea intrărilor ca biți individuali**

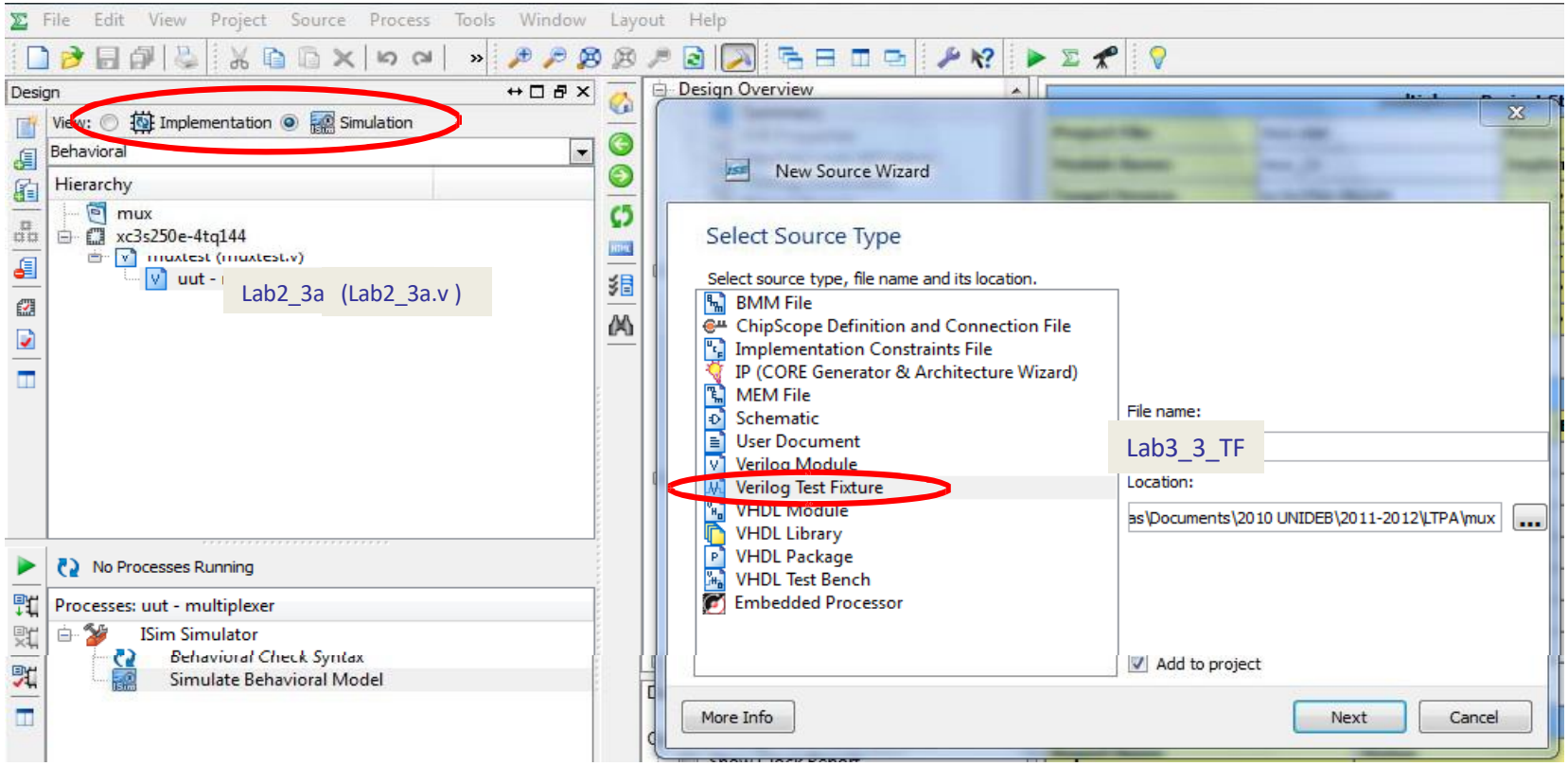
```
21  module Lab2_3a(
22      input [3:0] sw,
23      output [3:0] ld
24      );
25
26  assign ld[0] = sw[3] & sw[2] & sw[1] & sw[0]  ;
27  assign ld[1] = sw[3] | sw[2] | sw[1] | sw[0]  ;
28  assign ld[2] = sw[3] ^ sw[2] ^ sw[1] ^ sw[0]  ;
29  assign ld[3] = ~sw[3] & ~sw[2] & ~sw[1] & ~sw[0]  ;
30
31  endmodule
```

- **Utilizarea operatorilor de tip bit reduction pe semnale exprimate ca vectori**

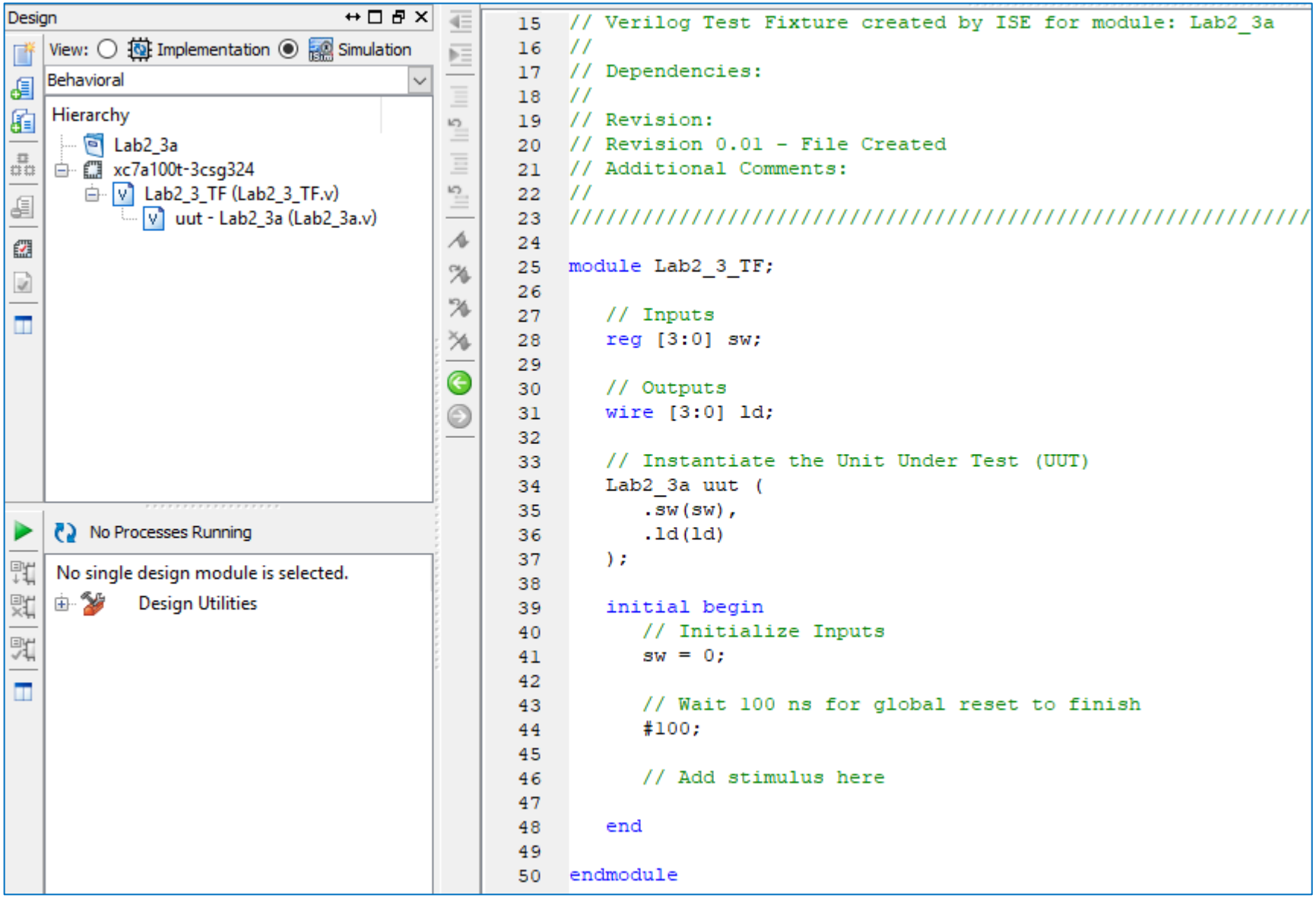
```
34  module Lab2_3a(
35      input [3:0] sw,
36      output [3:0] ld
37      );
38
39  assign ld[0] = &sw[3:0];
40  assign ld[1] = |sw[3:0];
41  assign ld[2] = ^sw[3:0];
42  assign ld[3] = ~|sw[3:0];
43
44  endmodule
```

Tema 3_3a: simulare

- Alegeți modul de simulare
- Creați un fișier de testare (text fixture) și specificați vectorii de test
  - Project / New Source - Verilog Test Fixture. Numele fișierului: Lab3_3_TF
- Selectați modulul care trebuie testat.



Tema 3_3a: simulare



Generarea vectorului de test

- Modificați conținutul fișierului Verilog Test Fixture creat automat
- Funcție de 4 variabile
  - Max. 16 combinații

Test vector generat folosind funcția ***for***

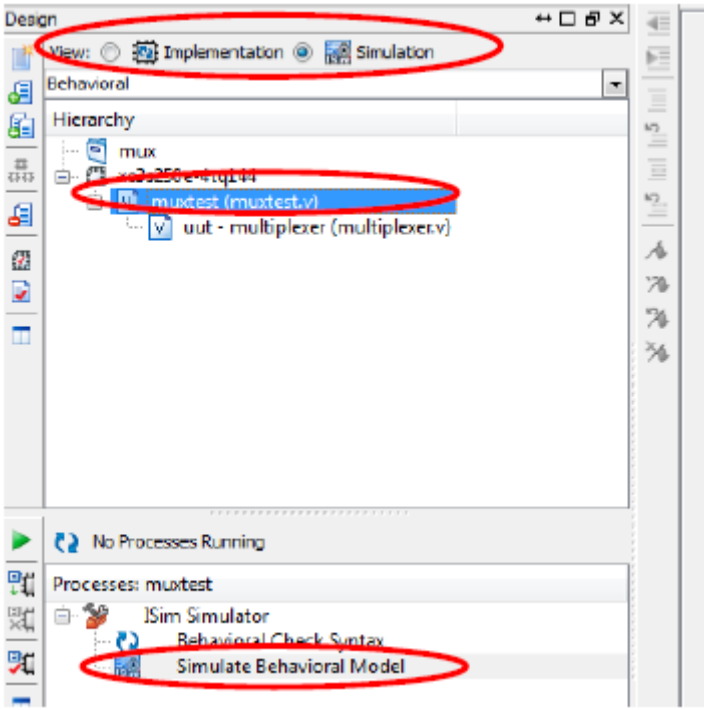
```
25  module Lab2_3_TF;
26     // Inputs
27     reg [3:0] sw;
28     // Outputs
29     wire [3:0] ld;
30     // Instantiate the Unit Under Test (UUT)
31     Lab2_3a uut (
32        .sw(sw),
33        .ld(ld)
34     );
35
36  integer i ;
37     initial begin
38        // Initialize Inputs
39        sw = 0;
40        // Wait 100 ns for global reset to finish
41        #100;
42        // Add stimulus here
43
44  // Teljes tesztvektorkészlet ciklussal generálva
45         for (i = 0 ; i<=15; i = i+1)
46         begin
47           #100 sw = i;
48         end
49
50     end
51  endmodule
```

Test vector generat folosind cod linear

```
25  module Lab2_3_TF;
26     // Inputs
27     reg [3:0] sw;
28     // Outputs
29     wire [3:0] ld;
30     // Instantiate the Unit Under Test (UUT)
31     Lab2_3a uut (
32        .sw(sw),
33        .ld(ld)
34     );
35
36  integer i ;
37     initial begin
38        // Initialize Inputs
39        sw = 0;
40        // Wait 100 ns for global reset to finish
41        #100;
42        // Add stimulus here
43        // Teljes tesztvektorkészlet lineáris felsorolással
44        #100    sw = 4'h0;
45        #100    sw = 4'h1;
46        #100    sw = 4'h2;
47        #100    sw = 4'h3;
48        #100    sw = 4'h4;
49        #100    sw = 4'h5;
50        #100    sw = 4'h6;
51        #100    sw = 4'h7;
52        #100    sw = 4'h8;
53        #100    sw = 4'h9;
54        #100    sw = 4'ha;
55        #100    sw = 4'hb;
56        #100    sw = 4'hc;
57        #100    sw = 4'hd;
58        #100    sw = 4'he;
59        #100    sw = 4'hf;
60
61     end
62  endmodule
```

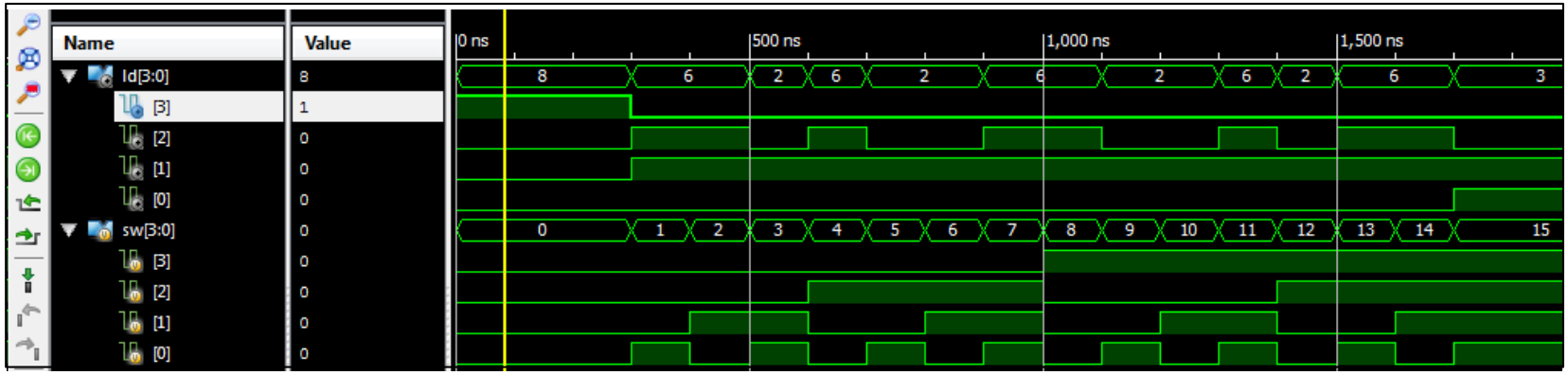
Simularea

- În Project Navigator se alege: Simulation, în Hierarchy selectați fișierul test fixture (Lab3_3_TF).
- In fereastra Processes alegeți ISim Simulator /Simulate Behavioral Model.



Lab3_3a rezultate

- **Rezultatele simulării trebuie să fie ca cele din figura de mai jos**
  - LD[0] → AND, LD[1] → OR, LD[2] → XOR, LD[3] → NOR



- Pentru implementare:
  - Alegeți Generate .bit file
  - Încărcați în placă și testați
- Notați rezultatele în raportul de laborator

Lucrarea de laborator nr. 4

- Implementarea funcțiilor logice pe două nivele – funcția XOR
- Decodificator BCD – 7 segmente, implementarea funcției pt. segmentul „a”
- Afișajul cu 7 segmente

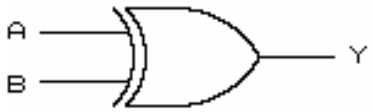
Lab4_1

Implementarea funcțiilor logice pe două nivele - Funcția XOR -

- Creați un nou proiect
- Adăugați o sursă nouă de tip "schematic”
- Desenați schemele prezentate pe slide-ul următor.

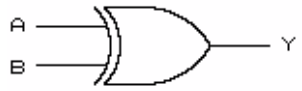
- Adăugați și adaptați fișierul de constrângeri Nexysx.ucf
  - Intrări: ***sw[2:0]*** *(A și B din figură)*
  - Ieșiri: ***led[4:0]*** (cele 5 implementări propuse)

A	B	Y=A⊕B
L	L	L
L	H	H
H	L	H
H	H	L

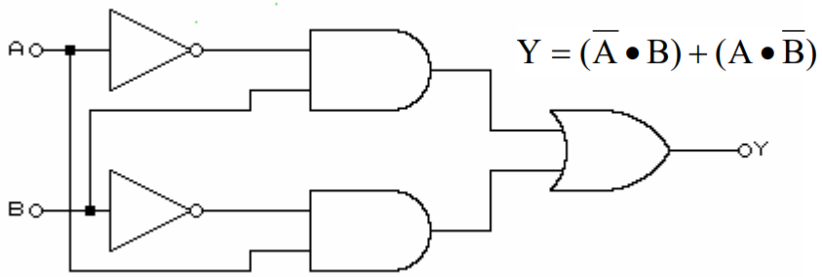


Implementări posibile ale funcției XOR

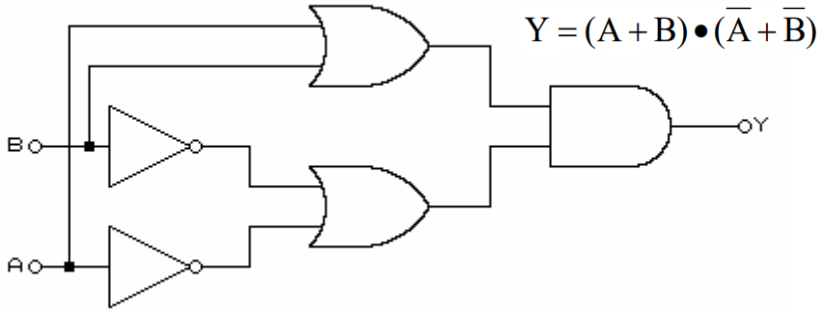
A	B	Y=A⊕B
L	L	L
L	H	H
H	L	H
H	H	L



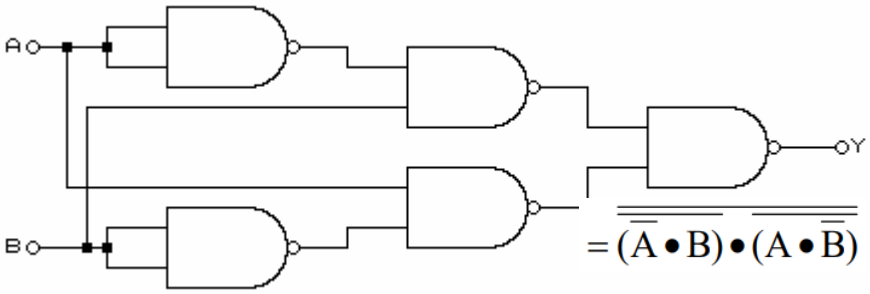
Implementare ca SOP



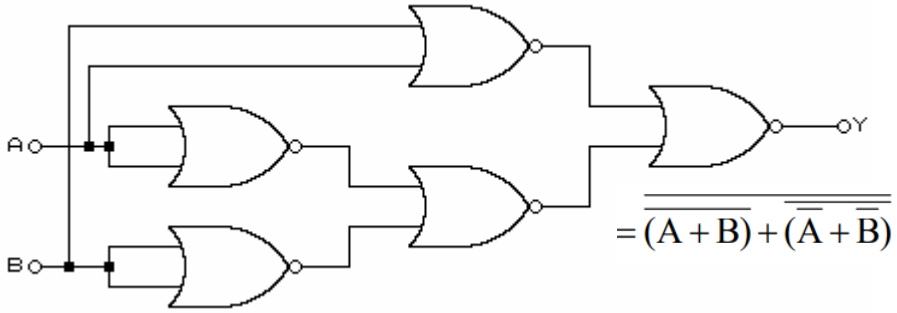
Implementare ca POS



Implementare cu porți NAND

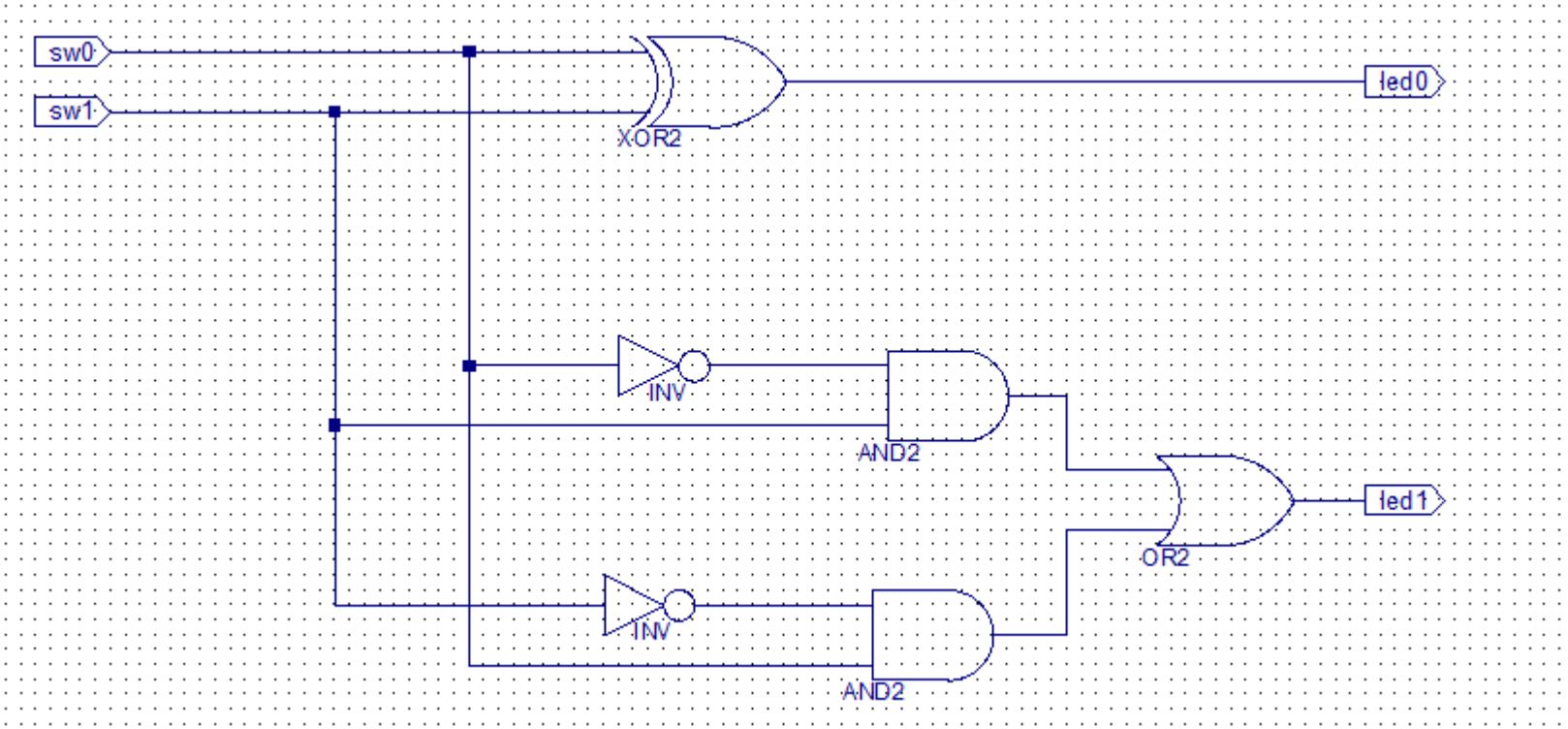


Implementare cu porți NOR



Implementarea funcției XOR

- Completați desenul cu următoarele 3 implementări conectând toate intrările la sw0 și respectiv sw1



Lab4_1 Rezultate

- Generați fișierul de configurare, încărcați fișierul în placă și testați funcționarea
- Folosind comutatoarele sw0 și sw1 generați cele 4 combinații posibile și notați în tabel starea corespunzătoare a ledurilor
- Completați raportul de laborator.
- Răspundeți la următoarele întrebări
  - Toate ieșirile au fost identice?
  - Care dintre implementările funcției XOR este mai avantajoasă și de ce?

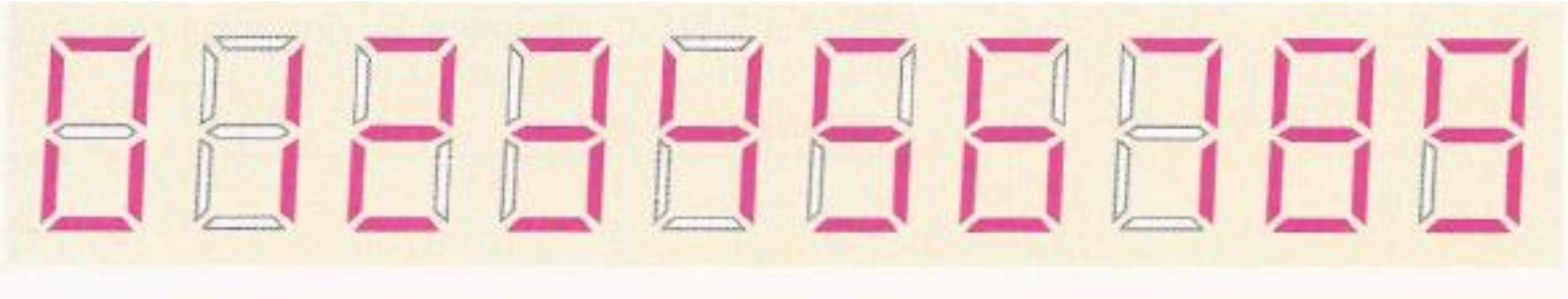
sw0	sw1	led0 AxorB	led1 (SOP)	led2 (POS)	led3 ("NAND")	led4 ("NOR")
0	0					
0	1					
1	0					
1	1					

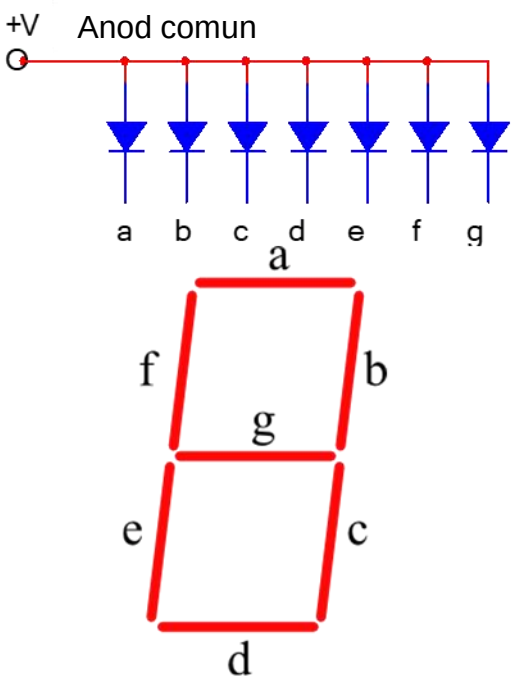
Lab4_2

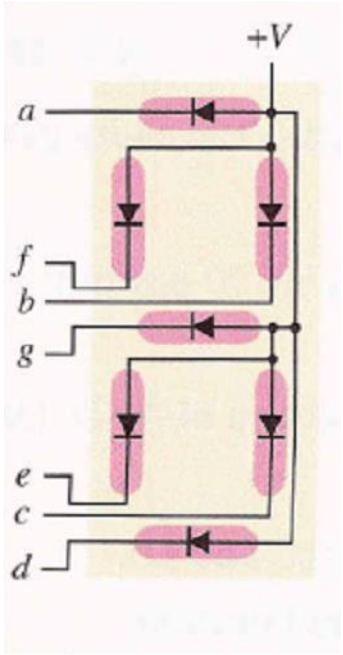
Decodificatorul BCD – 7 segmente, implementarea segmentului „a”

- Creați un nou proiect
- Adăugați o sursă nouă de tip "schematic”
- Proiectați segmentul “a” al unui decodificator BCD – 7 segmente
- Desenați toate cele 3 implementări prezentate în continuare pe aceeași pagină de tip schematic.
  - Intrările în cod BCD se vor genera cu comutatoarele: ***sw3, sw2, sw1, sw0***
  - Ieșiri:
    - ***„ca”*** *catodul ledului din segmentul a =* ***ieșirea negată*** *a primului circuit*
      (catozii CA…CG sunt activi pe nivel logic zero)
    - ***„an0”*** *anodul comun al ledurilor – trebuie conectate la masă (GND)*
      (anozii sunt activi pe nivel logic zero din construcția plăcii)
    - ***led1*** *și* ***led0*** *ieșirile pentru al doilea și al treilea circuit*
- Adăugați și adaptați fișierul de constrângeri Nexysx.ucf

Segmentul „a” al decodificatorului BCD – 7 segmente







Decimális	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

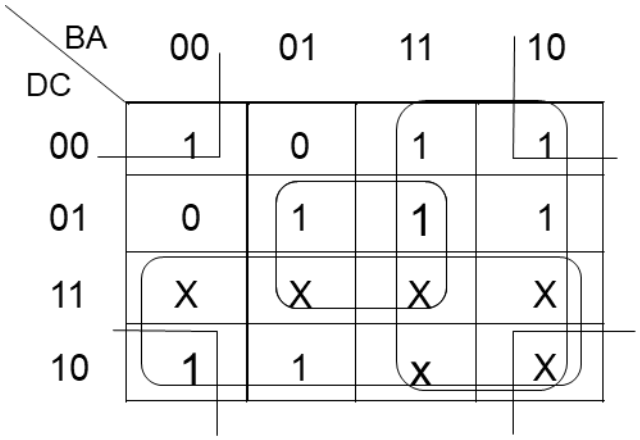
Implementarea directă sub forma SOP

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\,\overline{A} + \overline{D}\,\overline{C}\,B\,A + \overline{D}\,C\,\overline{B}\,A + \overline{D}\,C\,B\,\overline{A} + \overline{D}\,C\,B\,A + D\,\overline{C}\,\overline{B}\,\overline{A} + D\,\overline{C}\,\overline{B}\,A$$

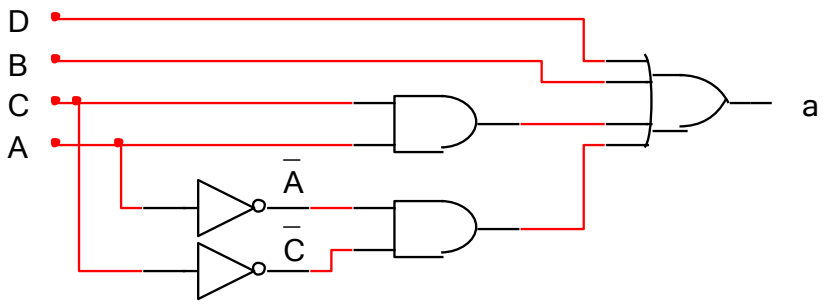
Segmentul „a” al decodificatorului BCD – 7 segmente

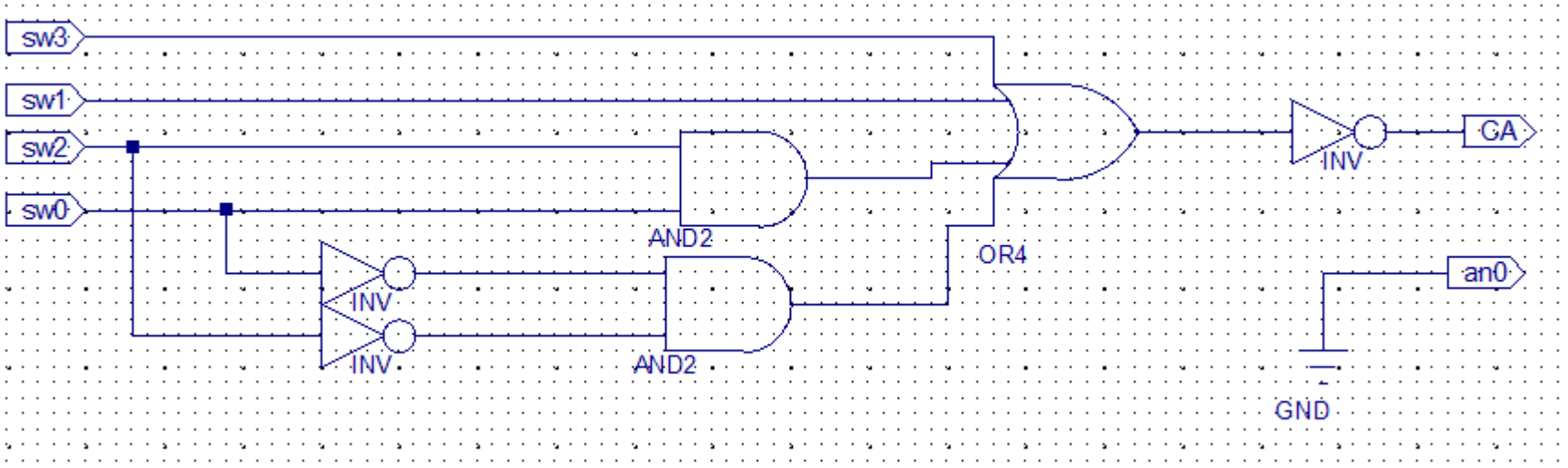
Implementarea după simplificare grafică

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}B\,\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\,\overline{B}A + \overline{D}CB\overline{A} + \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$$



$$a = D + B + CA + \overline{C}\,\overline{A}$$





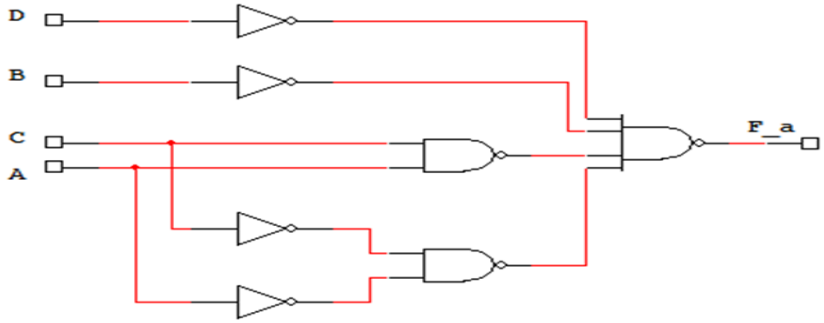
Segmentul „a” al decodificatorului BCD – 7 segmente

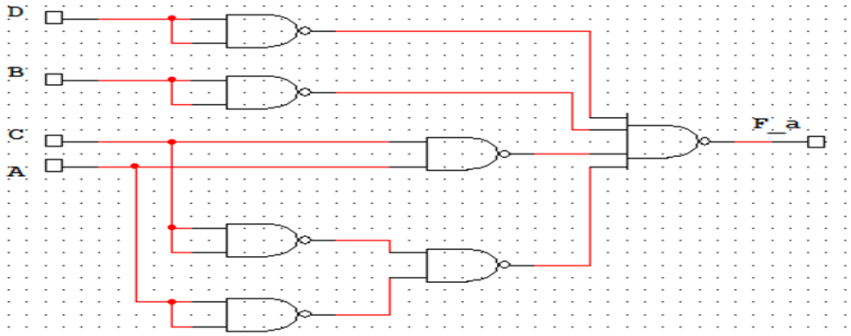
Implementarea cu porți NAND

- Funcția precedentă se poate transforma folosind teoremele lui De Morgan:

$$a = D + B + CA + \overline{C}\,\overline{A}$$

$$a = D + B + CA + \overline{C}\,\overline{A} = \overline{\overline{D + B + CA + \overline{C}\,\overline{A}}} = \overline{\overline{D} \bullet \overline{B} \bullet \overline{C \bullet A} \bullet \overline{\overline{C} \bullet \overline{A}}}$$





Lab4_2 Rezultate

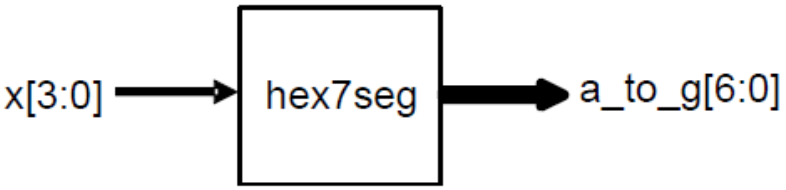
- Generați fișierul de configurare, încărcați fișierul în placă și testați funcționarea
- Folosind comutatoarele sw0, sw1, sw2 și sw3 generași toate combinațiile posibile și notați în tabel starea corespunzătoare a segmentului “a” și a ledurilor led0 și led1

sw3	sw2	sw1	sw0	„a”	led0	Led1	„a”=led0=led1?
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	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				

- Completați raportul de laborator cu concluziile voastre

Lab4_3

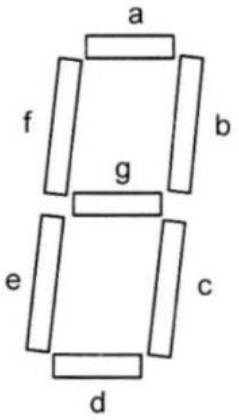
Afișajul cu 7 segmente – proiectare Verilog



x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	0	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

1 = off

0 = on



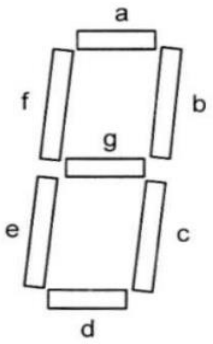
- Segmentele și punctele sunt comandate individual
- Segmentele sunt active pe zero

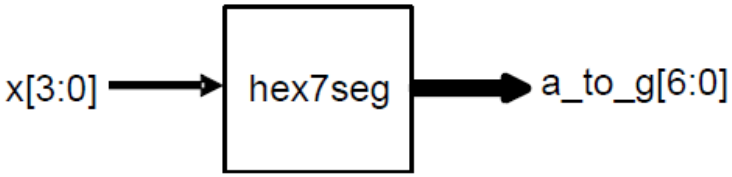
Comanda afișajului cu 7 segmente

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	0	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

1 = off

0 = on



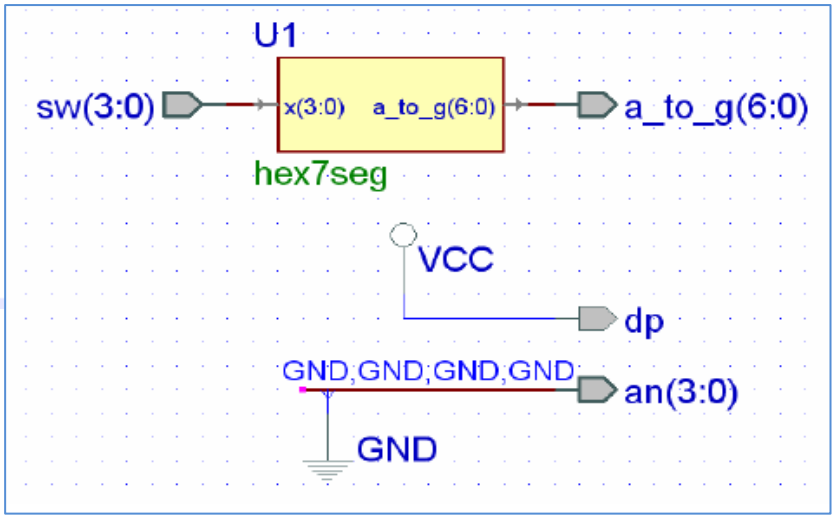



```
module hex7seg ( input [3:0] x,
                 output reg [6:0] a_to_g );
always @(*)
case(x)
0: a_to_g = 7'b0000001;
1: a_to_g = 7'b1001111;
2: a_to_g = 7'b0010010;
3: a_to_g = 7'b0000110;
4: a_to_g = 7'b1001100;
5: a_to_g = 7'b0100100;
6: a_to_g = 7'b0100000;
7: a_to_g = 7'b0001111;
8: a_to_g = 7'b0000000;
9: a_to_g = 7'b0000100;
'hA: a_to_g = 7'b0001000;
'hb: a_to_g = 7'b1100000;
'hC: a_to_g = 7'b0110001;
'hd: a_to_g = 7'b1000010;
'hE: a_to_g = 7'b0110000;
'hF: a_to_g = 7'b0111000;
default: a_to_g = 7'b0000001; // 0
endcase
endmodule
```

- Creați un proiect nou
- Adăugați o sursă nouă de tip Verilog cu conținutul din partea dreapta
  - Intrări: x[3:0]
  - Ieșiri: a_to_g[6:0]
- Adăugați o sursă nouă de tip Verilog, care va reprezenta fișierul „top module” (vezi pagina următoare)
  - Intrări: sw[3:0]
  - Ieșiri: a_to_g[6:0]

Comanda afișajului cu 7 segmente

```
module hex7seg_top (input [3:0] sw, output [6:0] a_to_g ,
output [3:0] an, output dp);

assign an = 4'b0000; // all digits on
assign dp = 1; // dp off

hex7seg D4 (.x(sw),.a_to_g(a_to_g));
endmodule
```



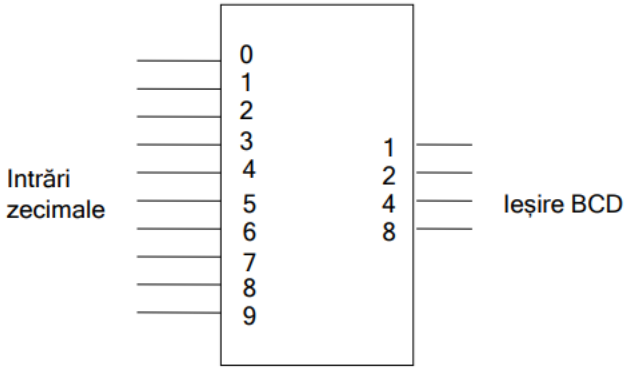
- Adăugați și adaptați fișierul de constrângeri
  - Intrările sunt: ***sw[3:0]***
  - Ieșirile sunt: ***a_to_g [6:0], an(3:0), dp.***

- Generați fișierul de configurare, încărcați fișierul în placă și testați funcționarea
- Folosind comutatoarele sw0, sw1, sw2 și sw3 generași toate combinațiile posibile și verificați corectitudinea stărilor afișajului cu 7 segmente
- Modificați codul în așa fel încât doar un singur display să afișeze numărul hexazecimal
- Completați raportul cu concluziile voastre și răspundeți la următoarele întrebări
  - Cum ați reușit să afișați numărul pe un singur digit?
  - Ce nivel de semnal este folosit pentru a „stinge” un digit de pe placa Nexys4?
  - Care este starea activă pentru semnalele comandă a segmentelor afișajului?
  - La ce este folosit semnalul de comandă "dp"?

Laboratorul 5

- Circuite codificatoare
- Circuite decodificatoare
- Circuite de multiplexare

Lab5_1a: Codificator zecimal BCD

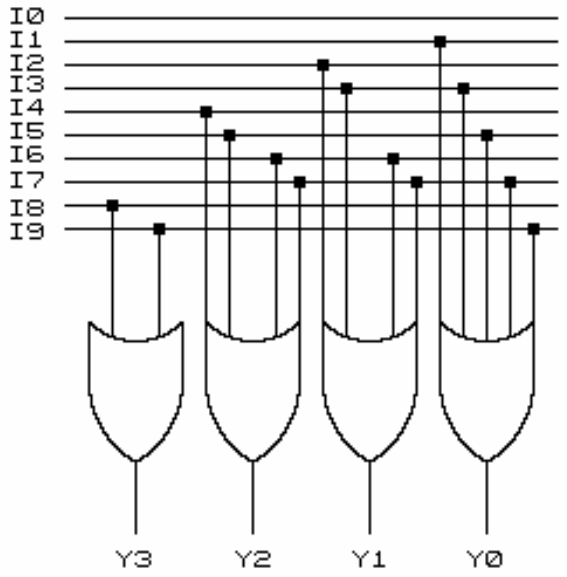


- Creați un proiect nou de tip HDL (Lab5_1a)
- Adăugați o sursă nouă de tip Verilog (Lab5_1a.v)
- Editați fișierul Lab5_1a.v conform descrierii din slide-ul următor
- Adăugați o sursă nouă de tip Verilog text fixture (Lab5_1a_tf.v) și adăugați semnale de stimulare
- Simulați circuitul
- Adăugați fișierul de constrângeri Nexysx.ucf și adaptați-l după cum urmează
  - Intrări: sw[9:0] (I0-I9 din figură)
  - Ieșiri: ld[3:0] (Y3-Y0 din figură)

Codificator zecimal BCD descriere structurală

I	Y_3	Y_2	Y_1	Y_0
I_0	0	0	0	0
I_1	0	0	0	1
I_2	0	0	1	0
I_3	0	0	1	1
I_4	0	1	0	0
I_5	0	1	0	1
I_6	0	1	1	0
I_7	0	1	1	1
I_8	1	0	0	0
I_9	1	0	0	1

- $Y_0=I_1+I_3+I_5+I_7+I_9$
- $Y_1=I_2+I_3+I_6+I_7$
- $Y_2=I_4+I_5+I_6+I_7$
- $Y_3=I_8+I_9$



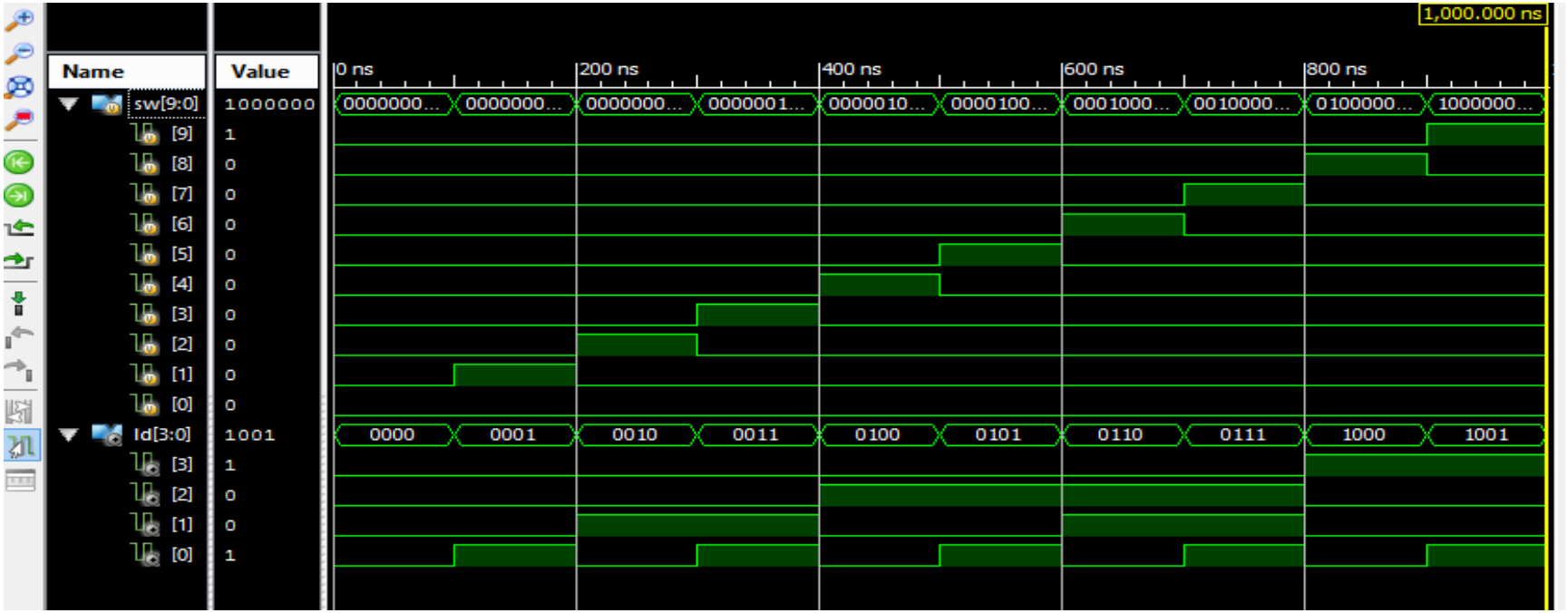

```
module BCDmod(
   input [9:0] sw,
output [3:0] ld );
            assign ld[0]=sw[1]|sw[3]|sw[5]|sw[7]|sw[9];
            assign ld[1]=sw[2]|sw[3]|sw[6]|sw[7];
            assign ld[2]=sw[4]|sw[5]|sw[6]|sw[7];
            assign ld[3]=sw[8]|sw[9];

endmodule
```

Lab5_1a simulare

I	Y_3	Y_2	Y_1	Y_0
I_0	0	0	0	0
I_1	0	0	0	1
I_2	0	0	1	0
I_3	0	0	1	1
I_4	0	1	0	0
I_5	0	1	0	1
I_6	0	1	1	0
I_7	0	1	1	1
I_8	1	0	0	0
I_9	1	0	0	1

// Add stimulus here

```
sw[1]=1;
#100; sw[1]=0; sw[2]=1;
#100; sw[2]=0; sw[3]=1;
#100; sw[3]=0; sw[4]=1;
#100; sw[4]=0; sw[5]=1;
#100; sw[5]=0; sw[6]=1;
#100; sw[6]=0; sw[7]=1;
#100; sw[7]=0; sw[8]=1;
#100; sw[8]=0; sw[9]=1;
```



Lab5_1b. Codificator de la 8 la 3 descriere structurală

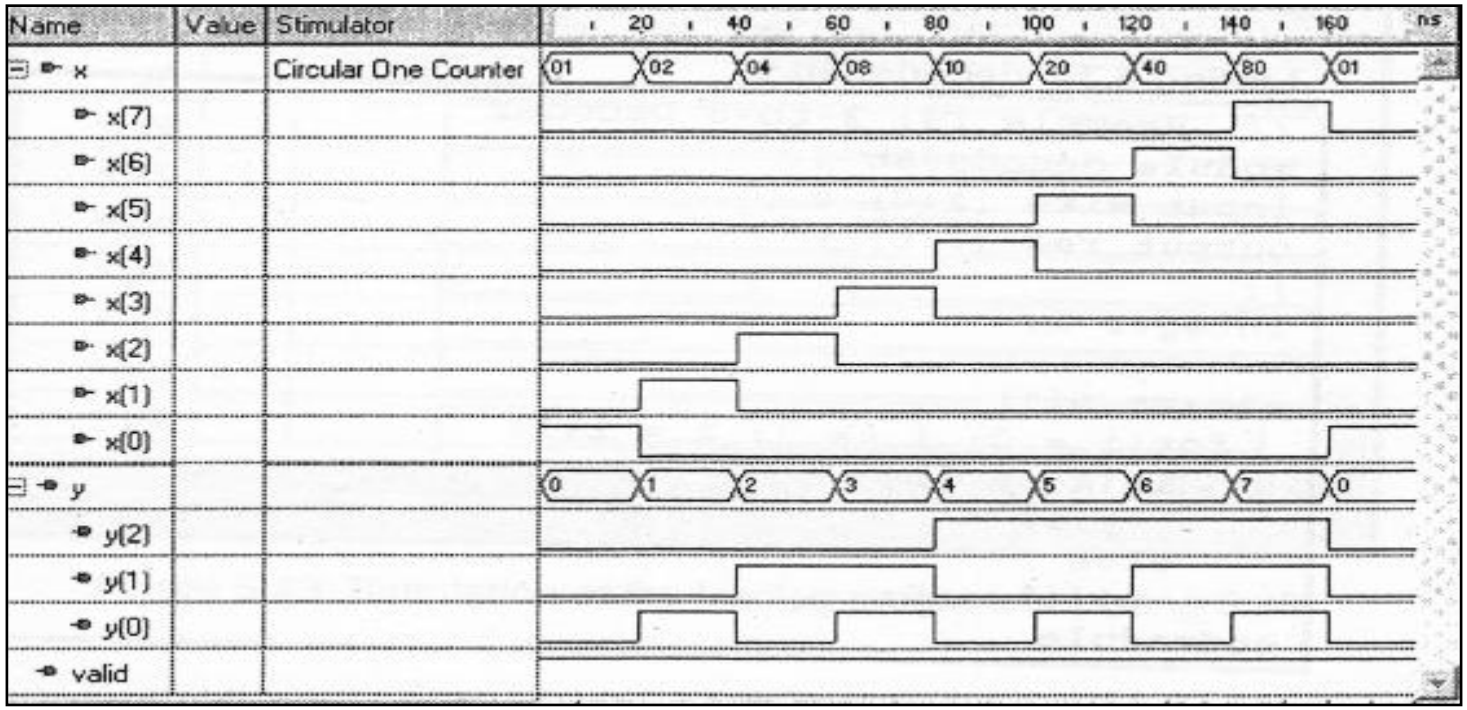
Codificator de la 8 la 3 (descriere structurală)

```
module encode83a (
input wire [7:0] x ,
output wire [2:0] y ,
output wire valid
);

assign y[2] = x[7] | x[6] | x[5] | x[4];
assign y[1] = x[7] | x[6] | x[3] | x[2];
assign y[0] = x[7] | x[5] | x[3] | x[1];
assign valid = |x;

endmodule
```

- Adăugați un fișier Verilog test fixture și simulați circuitul.
- Scrieți tabelul de adevăr al circuitului.

Codificator 8:3 simulare

Scrieți fișierul text fixture pentru a genera semnalul de intrare X (pe baza figurii de mai jos) și verificați starea ieșirilor.



Codificator 8:3 implementare

Selectați modul implementare și adăugați următorul fișier top modul la proiect

```
module encode83a_top (
input wire [7:0] sw ,
output wire [2:0] ld ,
output wire dp
);
wire valid;
assign dp = ~valid;

encode83a E1 (.x(sw),
      .y(ld),
      .valid(valid)
);

endmodule
```

- Adăugați fișierul ucf.
- Generați fișierul de configurare, încărcați fișierul în plăcă și testați funcționarea

Lab5_1c: Codificator prioritar

- Descriere comportamentală cu **IF**

```
// Codificator prioritar pe 3 biți (1-din-9)
//Verilog 1995

module v_priority_encoder_1 (sw, ld);
   input  [7:0] sw;
   output [2:0] ld;
   reg    [2:0] ld;

   always @(sw)
   begin
      if      (sw[0]) ld = 3'b000;
      else if (sw[1]) ld = 3'b001;
      else if (sw[2]) ld = 3'b010;
      else if (sw[3]) ld = 3'b011;
      else if (sw[4]) ld = 3'b100;
      else if (sw[5]) ld = 3'b101;
      else if (sw[6]) ld = 3'b110;
      else if (sw[7]) ld = 3'b111;
      else            ld = 3'bxxx;
   end

endmodule
```

```
// Codificator prioritar pe 3 biți (1-din-9)
//Verilog 2001

module v_priority_encoder_1 (input  [7:0] sw,
                             output reg [2:0] ld);


   always @(sw)
   begin
      if      (sw[0]) ld = 3'b000;
      else if (sw[1]) ld = 3'b001;
      else if (sw[2]) ld = 3'b010;
      else if (sw[3]) ld = 3'b011;
      else if (sw[4]) ld = 3'b100;
      else if (sw[5]) ld = 3'b101;
      else if (sw[6]) ld = 3'b110;
      else if (sw[7]) ld = 3'b111;
      else            ld = 3'bxxx;
   end

endmodule
```

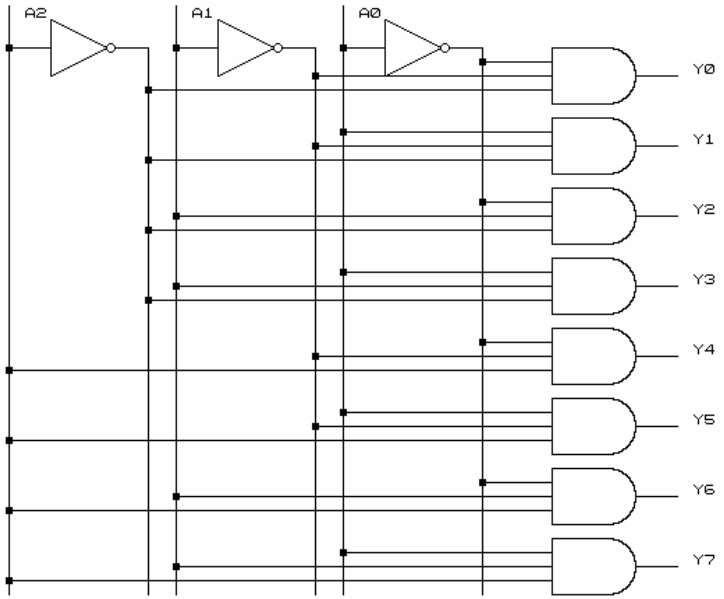
- Folosiți una din variantele de mai sus
- Adăugați fișierul ucf.
- Generați fișierul de configurare, încărcați fișierul în plăcă și testați funcționarea

Lab5_2: Decodificatoare

Lab5_2a: Decodificator binar de la 3 la 8 în limbaj structural

- Implementați un Decodificator binar de la 3 la 8 în limbaj Verilog structural

A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1




```
module decod(
   input A0,
   input A1,
   input A2,
   output [7:0] led
   );
            assign led[0]=~A0&~A1&~A2;
            assign led[1]=A0&~A1&~A2;
            assign led[2]=~A0&A1&~A2;
            assign led[3]=A0&A1&~A2;
            assign led[4]=~A0&~A1&A2;
            assign led[5]=A0&~A1&A2;
            assign led[6]=~A0&A1&A2;
            assign led[7]=A0&A1&A2;

endmodule
```

Lab5_2b: Decodificator binar de la 3 la 8

Descriere comportamentală

- Implementați un Decodificator binar de la 3 la 8 în limbaj Verilog comportamental

```
// 1-of-8 decoder (One-Hot)
module v_decoders_1 (input [2:0] sel, output reg [7:0] res);

  always @(sel or res)
  begin
    case (sel)
      3'b000 : res = 8'b00000001;
      3'b001 : res = 8'b00000010;
      3'b010 : res = 8'b00000100;
      3'b011 : res = 8'b00001000;
      3'b100 : res = 8'b00010000;
      3'b101 : res = 8'b00100000;
      3'b110 : res = 8'b01000000;
      default : res = 8'b10000000;
    endcase
  end
endmodule
```

```
// 1-of-8 decoder (One-Cold)
module v_decoders_1 (input [2:0] sel, output reg [7:0] res);

  always @(sel)
  begin
    case (sel)
      3'b000 : res = 8'b11111110;
      3'b001 : res = 8'b11111101;
      3'b010 : res = 8'b11111011;
      3'b011 : res = 8'b11110111;
      3'b100 : res = 8'b11101111;
      3'b101 : res = 8'b11011111;
      3'b110 : res = 8'b10111111;
      default : res = 8'b01111111;
    endcase
  end
endmodule
```

Lab5_3a: Multiplexoare

- Implementați un multiplexor 2:1 folosind una din cele trei tipuri de descriere prezentate

```
module mux_21 (input in0, in1, sel, output r);
assign r = (sel==1'b1) ? in1 : in0;
endmodule
```

Assign

```
module mux_21 (input in0, in1, sel, output reg r);
always @ (*)
if (sel==1'b1) r <= in1;
else           r <= in0;
endmodule
```

If

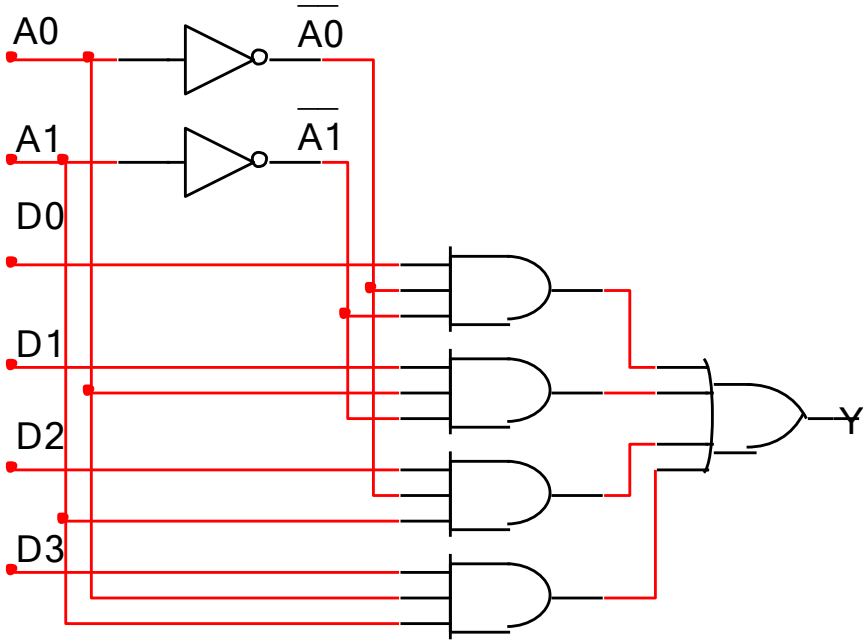
```
module mux_21 (input in0, in1, sel, output reg r);
always @ (*)
case(sel)
    1'b0:    r <= in0;
    1'b1:    r <= in1;
endmodule
```

Case

Lab5_3b: Multiplexor 4:1

Descriere structurală

- Implementați un multiplexor 4:1 în limbaj Verilog structural



Y = ~A1 & ~A0 & D0 | ~A1 & A0 & D1 | A1 & ~A0 & D2 | A1 & A0 & D3

Lab5_3c: Multiplexor 4:1

Descriere comportamentală

- Implementați un multiplexor 4:1 în limbaj Verilog folosind descrierea comportamentală
  - 4:1 multiplexer

```
module mux_41 (input in0, in1, in2, in3, input [1:0] sel, output reg
r);
always @ (*)
case(sel)
    2'b00: r <= in0;
    2'b01: r <= in1;
    2'b10: r <= in2;
    2'b11: r <= in3;
endcase

endmodule
```

Lab5_3d Multiplexor generic

- Implementați un multiplexor generic (parametrizabil) în limbaj Verilog

```
module mux2g
#(parameter N = 4)
(input wire [N-1:0] a,
 input wire [N-1:0] b,
 input wire s,
 output reg [N-1:0] y
);

always @(*)
      if(s == 0)
     y = a;
  else
     y = b;

endmodule
```

```
module mux28(
input wire [7:0] a,
input wire [7:0] b,
input wire s,
output wire [7:0] y
);

mux2g #(
  .N(8))
M8 (.a(a),
  .b(b),
  .s(s),
  .y(y)
);

 endmodule
```

Lucrarea de laborator nr. 6

- Comparatoare
- Circuite de generare/verificare a parității
- Sumatoare

Lab6_1 Comparatoare pentru numere pe 2-biți

- Creați un proiect nou de tip HDL (Lab6_1)
- Adăugați o sursă nouă de tip Verilog (Lab6_1.v)
- Editați fișierul Lab6_1.v ca în descrierea de mai jos
- Adăugați și adaptați fișierul de constrângeri Nexys.ucf
- Generați și încărcați fișierul de configurare
- Testați funcționarea circuitului și completați tabelul alăturat

```
module compare_2 (output reg A_lt_B, A_gt_B, A_eq_B,
                  input [1:0] A,B);

always @ (A or B)
        begin
                A_lt_B = 0;
                A_gt_B = 0;
                A_eq_B = 0;

                if (A==B) A_eq_B = 1;
                else if (A>B) A_gt_B = 1;
                else A_lt_B = 1;

        end

endmodule
```

A1	A0	B1	B0	A_lt_B	A_eq_B	A_gt_B
0	0	0	0			
0	0	0	1			
0	0	1	0			
1	1	1	1			

Lab6_2: Generator de paritate

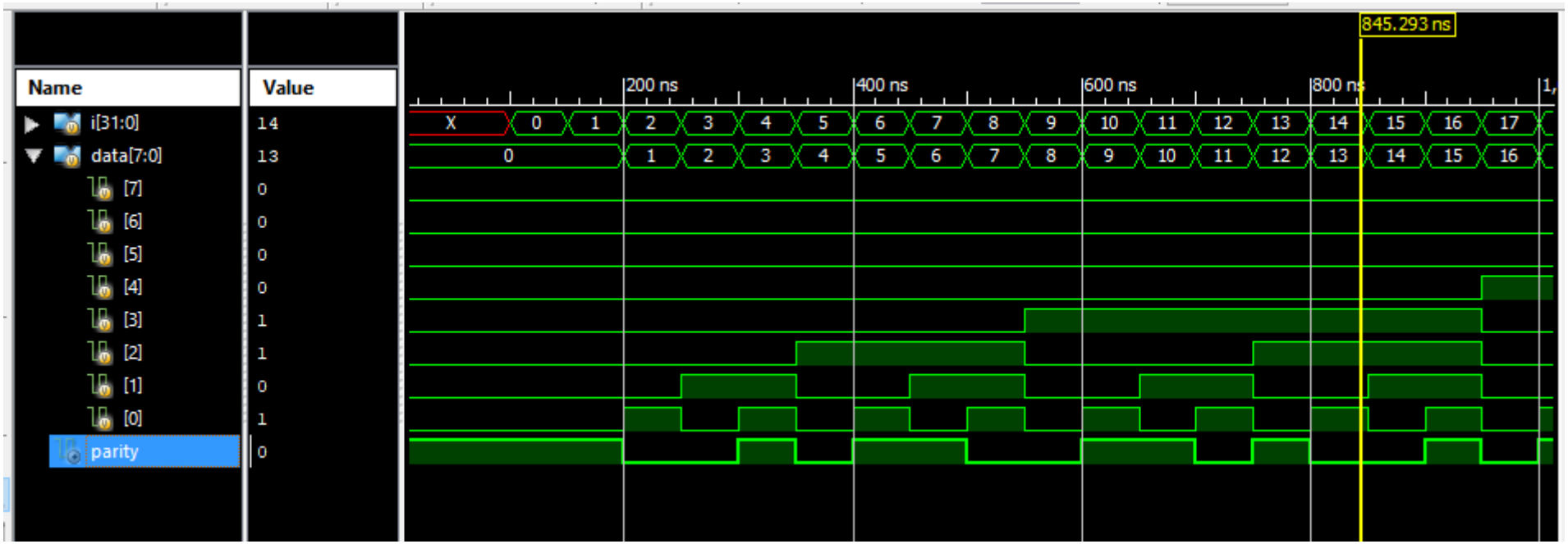
- Creați un proiect nou de tip HDL (Lab6_2)
- Adăugați o sursă nouă de tip Verilog (Lab6_2.v) și completați-l ca în descrierea de mai jos
- Adăugați o sursă nouă de tip Verilog text fixture (Lab6_2_tf.v) și adăugați semnalele de stimulare ca în codul din partea dreaptă
- Simulați funcționarea circuitului

```
module oddparity_for (output reg parity, input [7:0] data);
integer k;
always@(data)
begin
        parity = 1;
        for (k = 0; k <= 7; k = k+1)
        begin
                if (data[k] == 1)
                   parity = ~parity;
                end
        end
endmodule
```

```
25  module oddparity_test;
26
27      // Inputs
28      reg [7:0] data;
29
30      // Outputs
31      wire parity;
32
33      // Instantiate the Unit Under Test (UUT)
34      oddparity_for uut (
35          .parity(parity),
36          .data(data)
37      );
38      integer i;
39      initial begin
40          // Initialize Inputs
41          data = 0;
42
43          // Wait 100 ns for global reset to finish
44          #100;
45
46          for (i = 0; i <= 255; i = i+1)
47          begin
48              #50 data =i;
49
50          end
51
52      end
53
54  endmodule
```

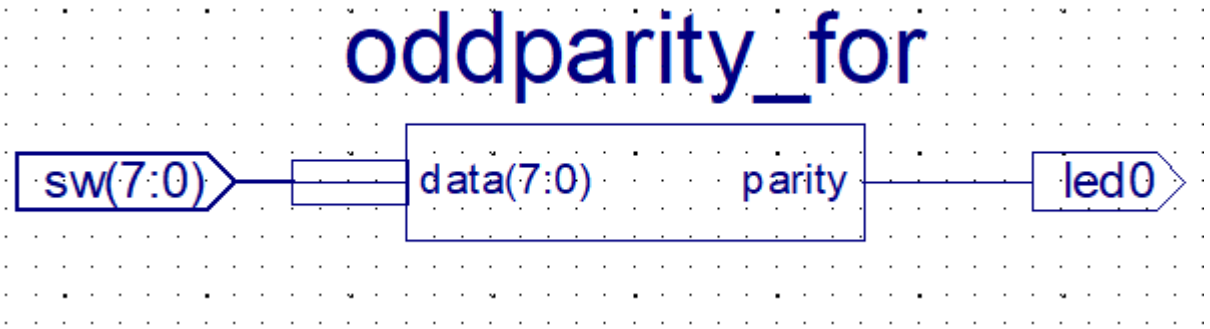
Lab6_2: Generator de paritate simulare

- Setați fereastra de simulare
  - Afișați întreaga porțiune simulată folosind butonul Zoom to Full View 
  - Modificați reprezentarea bazei de numerație a semnalelor i[3:0] și data [7:0] în zecimal fără semn (click dreapta pe semnal: Radix -> Unsigned Decimal)
  - Apăsați săgeata din stânga magistralei data [7:0] pentru a afișa diagrama fiecărui semnal component
- Comparați simularea obținută cu cea din figura de mai jos.

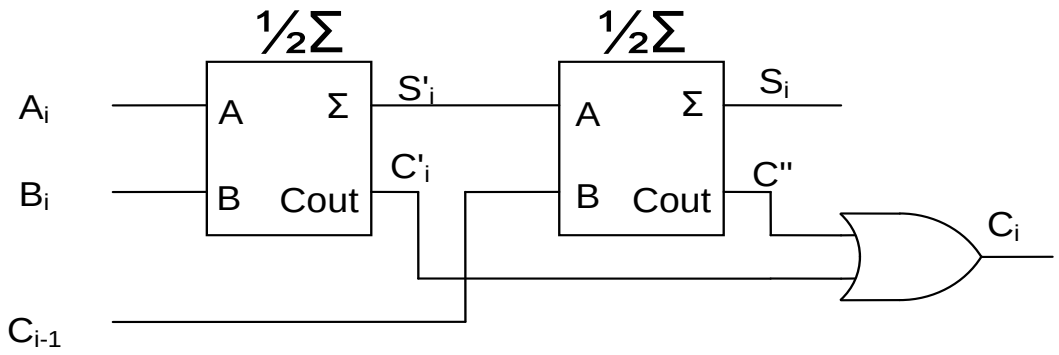


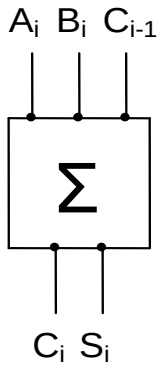
Lab6_2: Generator de paritate - implementare

- Adăugați și adaptași fișierul de constrângeri Nexys.ucf
- Generați fișierul de configurare, încărcați și testați circuitul.

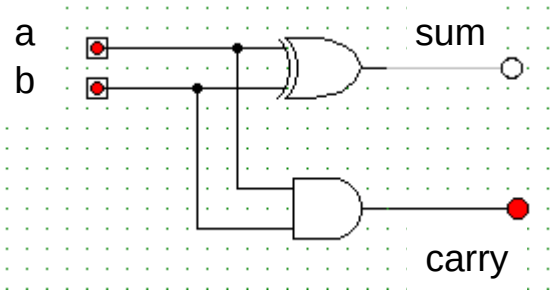


Lab6_3: Sumator complet pe 1 bit





- Creați un proiect nou de tip HDL (Lab6_3)
- Adăugați o sursă nouă de tip Verilog (half_add.v)

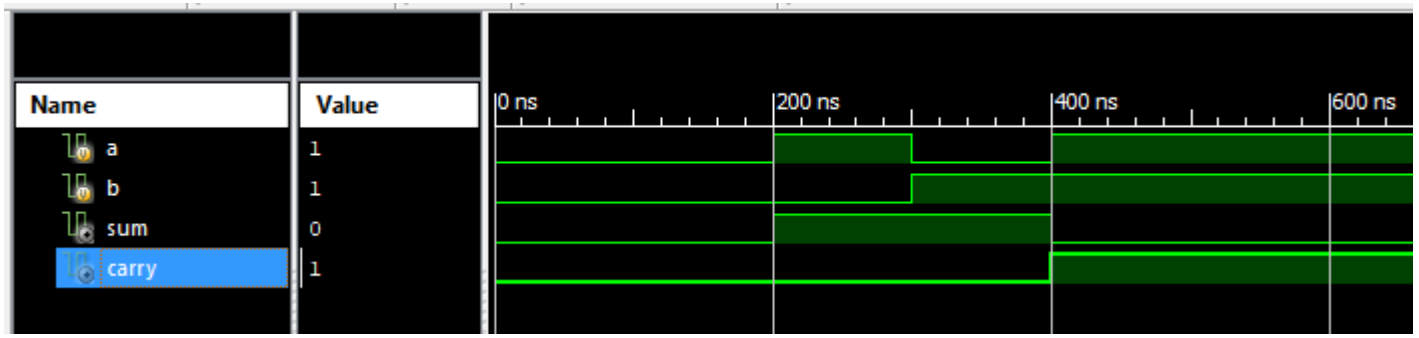



```
module half_add (output sum, carry, input a, b);
        xor (sum, a, b);     // exclusive OR
        and (carry, a ,b);   // AND
endmodule
```

Lab6_3a: Semi-sumator simulare

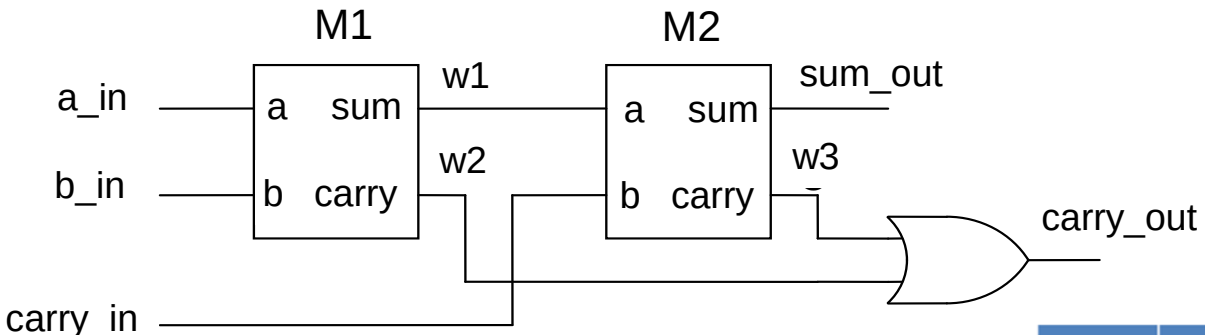
- Adăugați o sursă nouă de tip Verilog text fixture (half_add_test.v)
- Specificați semnalele de stimulare (ca mai jos)
- Simulați circuitul și verificați rezultatele

```
// Add stimulus here
#100
        a=1;
#100
        a=0;
        b=1;
#100
        a=1;
        b=1;
```



Lab6_3b: Sumator complet pe 1 bit

- Adăugați o sursă nouă Verilog (full_add.v)
- Descrieți un sumator complet ca cel prezentat în figura următoare




```
module full_add (output sum_out, carry_out, input a_in, b_in, carry_in );

wire w1, w2, w3;

        half_add M1 (.a(a_in), .sum(w1), .b(b_in), .carry(w2));
        half_add M2 (.sum(sum_out), .b(w1), .carry(w3), .a(carry_in));

        or (carry_out, w2, w3);
endmodule
```

Cin	A	B	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

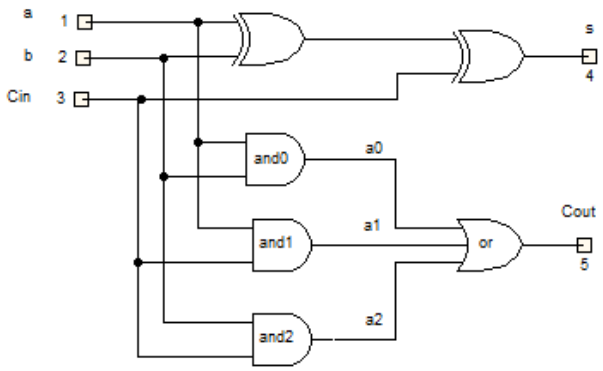
- Adăugați și adaptași fișierul de constrângeri Nexys.ucf
- Generați fișierul de configurare, încărcați și testați circuitul.

Lab6_4: Sumator complet pe 1 bit

```
// Versiunea A (descriere structurală explicită)
module add1_full (input a, b, cin, output cout, s);
xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
wire a0, a1, a2;
and2_m and0(.i0(a), .i1(b), .o(a0));
and2_m and1(.i0(a), .i1(cin), .o(a1));
and2_m and2(.i0(b), .i1(cin), .o(a2));
or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```

```
// Versiunea B (descriere structurală implicită)
module add1_full (input a, b, cin, output cout, s);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

```
// Versiunea C (descriere comportamentală)
module add1_full (input a, b, cin, output cout, s);
assign {cout, s} = a + b + cin;
endmodule
```



Cin	A	B	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- Creați un proiect nou HDL (Lab6_4)
- Adăugați o sursă Verilog (add1_full.v), utilizați una din variantele de mai sus
- Adăugați și adaptați fișierul de constrângeri Nexys.ucf
- Generați fișierul de configurare, încărcați și testați circuitul.

Lab6_5a: Sumator pe 4-biți

Sumator pe 4-biți descriere structurală

- Creați un proiect nou HDL (Lab6_5a)
- Creați în fișier nou Verilog add4.v cu conținutul de mai jos
- Adăugați o copie a sursei create în proiectul precedent (add1_full.v) (Add copy of source)
- Adăugați și adaptați fișierul de constrângeri Nexys.ucf
- Generați fișierul de configurare, încărcați și testați circuitul

```
// 6_5a 4-bits adder (descriere structurală)
module add4 (input [3:0] a, b, output [4:0] s);
wire [3:0] c;
add1_full add0(.a(a[0]), .b(b[0]), .cin(1'b0),    .cout(c[0]), .s(s[0]));
add1_full add1(.a(a[1]), .b(b[1]), .cin(c[0]), .cout(c[1]), .s(s[1]));
add1_full add2(.a(a[2]), .b(b[2]), .cin(c[1]), .cout(c[2]), .s(s[2]));
add1_full add3(.a(a[3]), .b(b[3]), .cin(c[2]), .cout(s[4]),    .s(s[3]));
endmodule
```

Lab6_5b: Sumator pe 4-biți

Sumator pe 4-biți descriere comportamentală

- Creați un proiect nou HDL (Lab6_5b)
- Creați în fișier nou Verilog add4.v cu conținutul de mai jos

```
// 6_5b
module add4 (input [3:0] a, b, output [4:0] s);
assign s = a + b;
endmodule
```

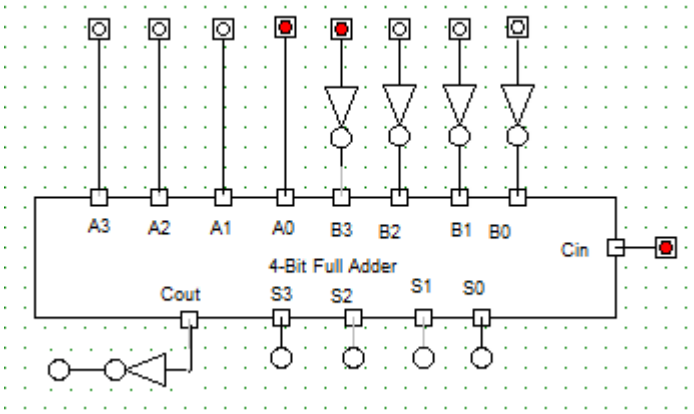
- Adăugați și adaptați fișierul de constrângeri Nexys.ucf
- Generați fișierul de configurare, încărcați și testați circuitul

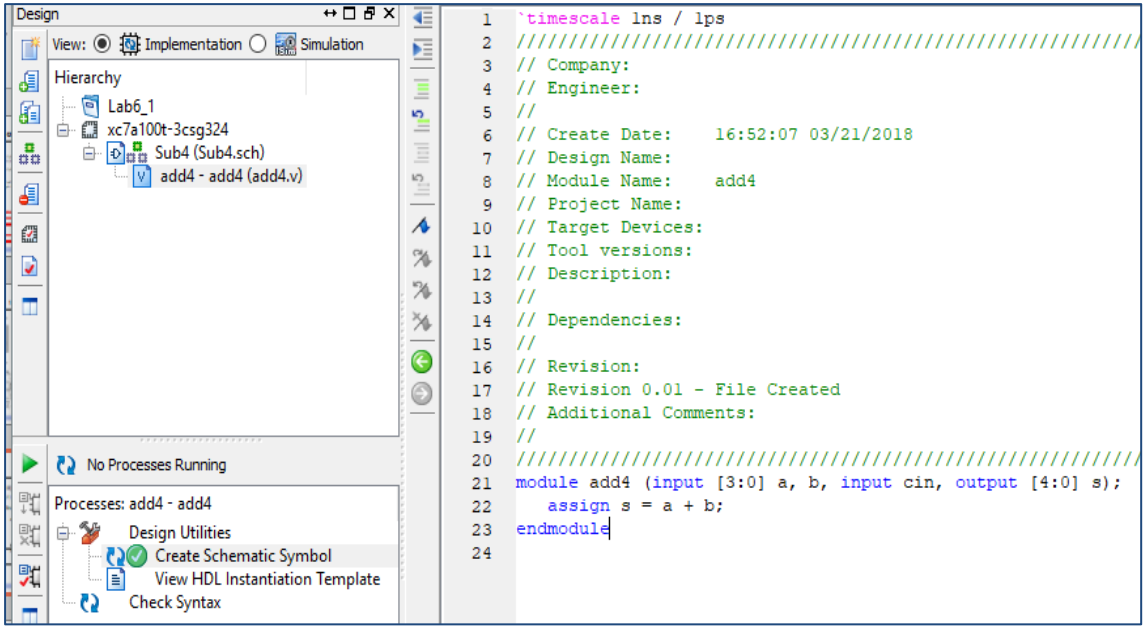
Lucrarea de laborator nr. 7

- Circuite aritmetice și logice
  - Circuit de adunare/scădere pe 4 biți
  - ALU pe 1 bit
  - ALU pe 4 biți
  - ALU pe 4 biți cu afișaj pe 7 segmente

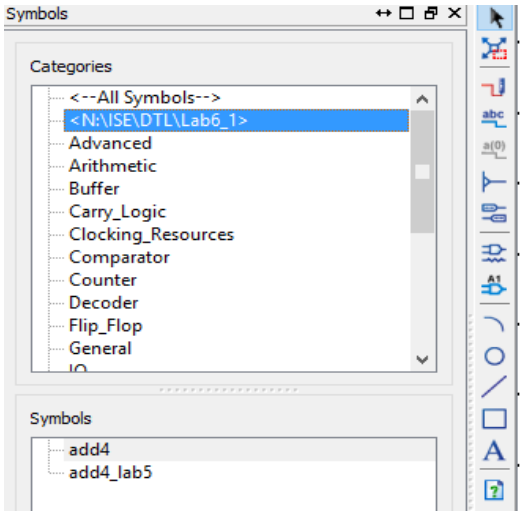
Lab7_1: Circuit de scădere pe 4 biți

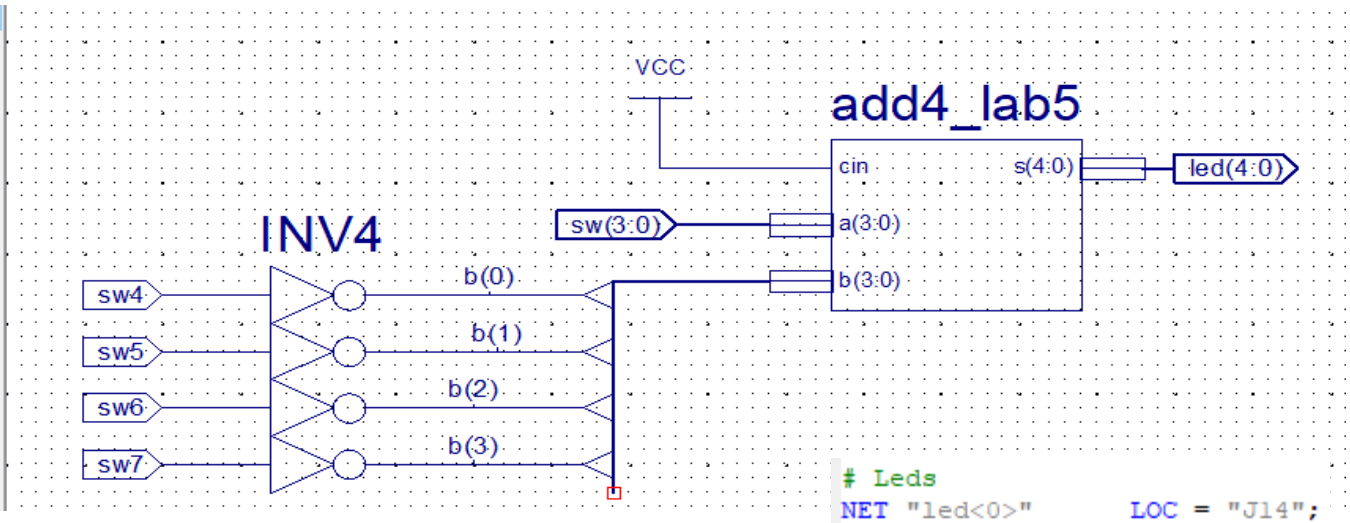
- Creați un proiect nou de tip HDL (Lab7_1)
- Adăugați(***Add Copy of Source***) fișierul sursa Verilog ***add4.v*** e creat în laboratorul anterior la tema ***Lab6_5b***. Adăugați un port de intrare nou cu numele ***cin*** la această sursă.
- În fereastra ***Processes*** faceți dublu clic pe ***Create Schematic Symbol***
- Adăugați o sursă nouă de tip schematic cu numele ***subb4.v***.
- În editorul schematic în fereastra ***Categories*** selectați directorul de lucru curent
- În fereastra ***Symbols*** selectați simbolul ***add4*** creat anterior. Adăugați acest simbol la pagină și completați schema așa cum este ea prezentată în pagina următoare.





Lab7_1: Circuit de scădere pe 4 biți



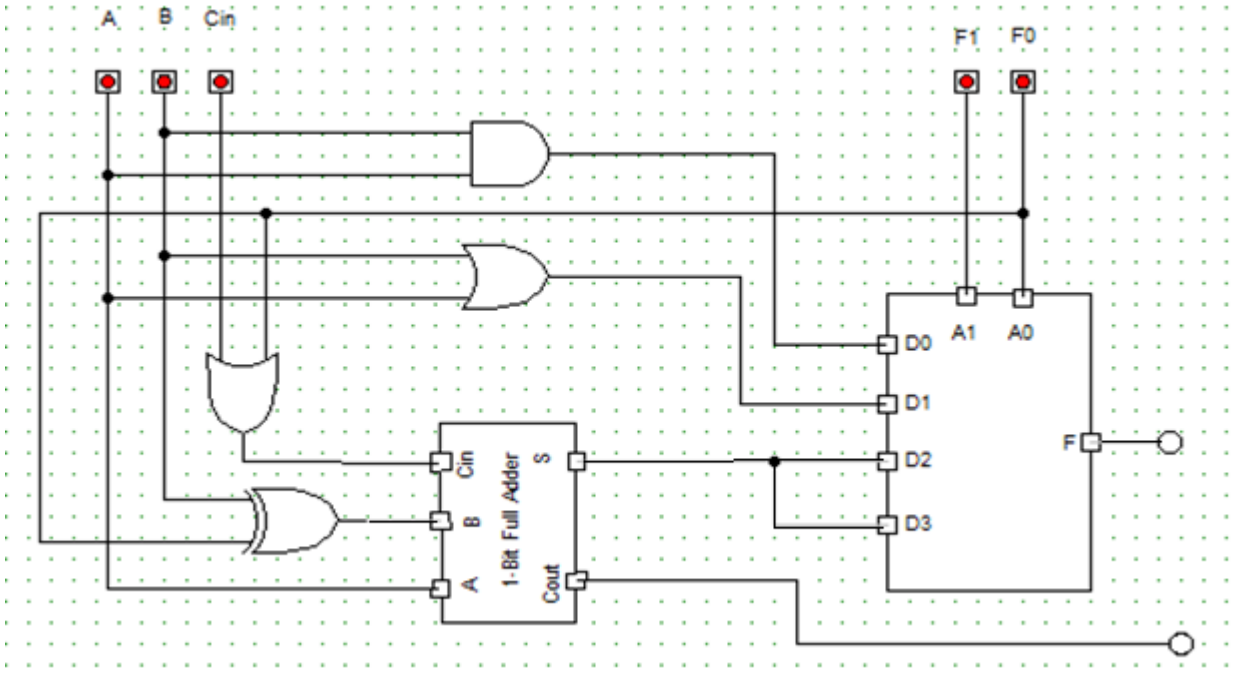



```
# Leds
NET "led<0>"        LOC = "J14";
NET "led<1>"        LOC = "J15";
NET "led<2>"        LOC = "K15";
NET "led<3>"        LOC = "K14";
NET "led<4>"        LOC = "E17";
#NET "Led<5>"        LOC = "P15";
#NET "Led<6>"        LOC = "F4";
#NET "Led<7>"        LOC = "R4";
#NET "Led<4>"        LOC = "E16";
#NET "Led<5>"        LOC = "P16";
#NET "Led<6>"        LOC = "E4";
#NET "Led<7>"        LOC = "P4";

## Switches
NET "sw<0>"         LOC = "G18";
NET "sw<1>"         LOC = "H18";
NET "sw<2>"         LOC = "K18";
NET "sw<3>"         LOC = "K17";
NET "sw4"         LOC = "L14";
NET "sw5"         LOC = "L13";
NET "sw6"         LOC = "N17";
NET "sw7"         LOC = "R17";
```

- Adăugați și adaptați fișierul Nexys2_500General.ucf descărcat de pe pagina Digilent. Ștergeți comentariul din rândurile referitoare la switch-uri și leduri [4:0]. Modificați denumirile porturilor astfel încât să corespundă cu cele ale porturilor din schemă
- Remarcați diferența dintre modul de specificare a constrângerilor pentru o magistrală: sw(3:0) respectiv fire separate sw4, sw5, sw6, sw7 din fișierul ucf.
- Generați fișierul de configurare, încărcați-l în placă și testați funcționarea circuitului.

Lab7_2: ALU pe 1 bit

- Creați un proiect nou (Lab7_2)
- Adăugați (Add Copy of Source) fișierul add1_full .v creat la tema Lab6_4.
- În fereastra ***Processes*** faceți dublu clic pe ***Create Schematic Symbol***
- Adăugați un fișier nou de tip schematic (Sub4)
- In editorul schematic în fereastra Categories selectați directorul de lucru curent
- In fereastra Symbols selectați simbolul add1_full și adăugați-l la schemă și completați desenul ca în schema de următoare.
- Adăugați la proiect descrierea Verilog a unui multiplexor din cele realizate în laboratorul Lab5_3c. Pentru acesta creați de asemenea un simbol schematic care trebuie adăugat la schemă și conectat ca în figura următoare (schema prezentată este doar cu titlu informativ, cea din Xilinx ISE va arăta diferit).



Lab7_2: ALU pe 1 bit

- Adăugați și adaptați fișierul .ucf.
- Generați fișierul de configurare, încărcați-l în placă și testați funcționarea circuitului.
- Completați tabelul următor

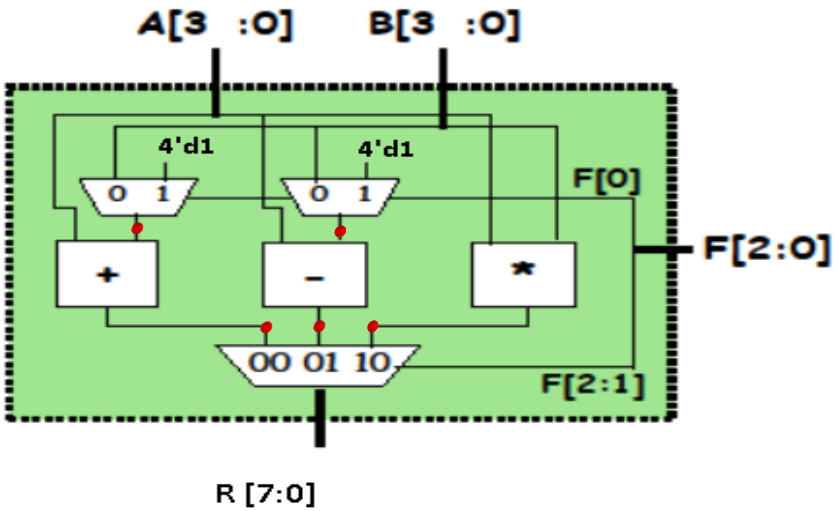
Operații logice

F1	F0	A	B	Cin	F
0	0	0	0	x	
0	0	0	1	x	
0	0	1	0	x	
0	0	1	1	x	
0	1	0	0	x	
0	1	0	1	x	
0	1	1	0	x	
0	1	1	1	x	
1	0				

Operații aritmetice

F1	F0	A	B	Cin	F	Cout
1	0	0	0	0		
1	0	0	1	0		
1	0	1	0	0		
1	0	1	1	0		
1	0	0	0	1		
1	0	0	1	1		
1	0	1	0	1		
1	0	1	1	1		
1	1	0	0	0		
1	1	0	1	0		
1	1	1	0	0		
1	1	1	1	0		
1	1	0	0	1		
1	1	0	1	1		
1	1	1	0	1		
1	1	1	1	1		

Lab7_3: ALU pe 4 biți



F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

- Magistrale pe 4 și 8 biți

- Creați un proiect nou Lab7_3
- Adăugați o sursă nouă de tip „Verilog” (alu_top). Acest modul va conecta toate sub-modulele:

```
module alu_top(input [3:0] a, b, input [2:0] f, output [7:0] r ); wire
[3:0] addmux_out, submux_out;
wire [7:0] add_out, sub_out, mul_out;
          mux2_4 adder_mux(b, 4'd1, f[0], addmux_out);
          mux2_4 sub_mux(b, 4'd1, f[0], submux_out);
          add4 our_adder(a, addmux_out, add_out);
          sub4 our_subtracer(a, submux_out, sub_out);
          mul4 our_multiplier(a,b,mul_out);
          mux3_8 output_mux(add_out, sub_out, mul_out, f[2:1], r);
endmodule
```

- Adăugați o sursă nouă de tip „Verilog” (alu4_modules) care va conține descrierea submodulelor (prezentată în pagina următoare)

Descrierea modulelor

```
module mux2_4(input [3:0] i0, i1, input sel, output [7:0] out);
          assign out = sel ? i1 : i0;
endmodule

module mux3_8(input [7:0] i0, i1, i2, input [1:0] sel, output reg [7:0] out);
     always @(i0 or i1 or i2 or sel)
          begin
               case (sel)
                    2'b00: out = i0;
                    2'b01: out = i1;
                    2'b10: out = i2;
                    default: out = 8'bx;
               endcase
          end
endmodule

module add4(input [3:0] i0, i1, output [7:0] sum);
          assign sum=i0+i1;
endmodule

module sub4(input [3:0] i0, i1, output [7:0] diff);
          assign diff=i0-i1;
endmodule

module mul4(input [3:0] i0, i1, output [7:0] prod);
          assign prod=i0*i1;
endmodule
```

Implementare și testare ALU

- Adăugați și adaptați fișierul .ucf astfel încât F[2:0] <=> btn [2:0]; a [3:0] <=> sw [3:0], b [3:0] <=> sw [7:4], r [7:0] <=> led [7:0]
- Generați fișierul de configurare, încărcați-l în placă și testați funcționarea circuitului.
- Completați tabelul următor
- Folosind sw [7:0] setați următorii operanzi: a = 3, b = 2
- Folosind btn [2:0] setați cele 5 operații posibile. În fiecare caz completați rezultatul în tabel.

F2 btn[2]	F1 btn[1]	F0 btn[0]	r[7]	r[6]	r[5]	r[4]	r[3]	r[2]	r[1]	r[0]
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	X								

- Testați din nou folosind alți operanzi: a = 10, b = 12. Rezultatul scăderii este corect?
- Cât va fi cea mai mare valoare a rezultatului înmulțirii?

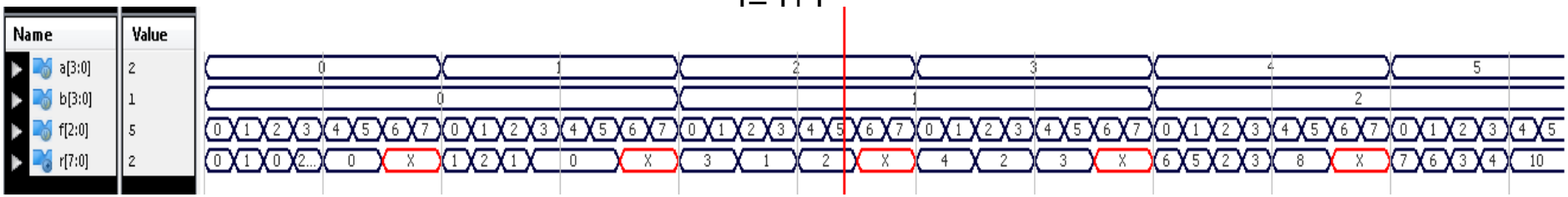
```
NET "a<0>" LOC=G18;
NET "a<1>" LOC=H18;
NET "a<2>" LOC=K18;
NET "a<3>" LOC=K17;

NET "b<0>" LOC=L14;
NET "b<1>" LOC=L13;
NET "b<2>" LOC=N17;
NET "b<3>" LOC=R17;

NET "f<0>" LOC=B18;
NET "f<1>" LOC=D18;
NET "f<2>" LOC=E18;
## LEDs
NET "r<0>" LOC=J14;
NET "r<1>" LOC=J15;
NET "r<2>" LOC=K15;
NET "r<3>" LOC=K14;
NET "r<4>" LOC=E17;
NET "r<5>" LOC=P15;
NET "r<6>" LOC=F4;
NET "r<7>" LOC=R4;
```

Lab7_3b: 4-bits ALU – simulare (facultativ)

- Adăugați o sursă nouă de tip Verilog text fixture
- Specificați semnalele de stimulare (ca mai jos)
- Simulați circuitul și verificați rezultatele

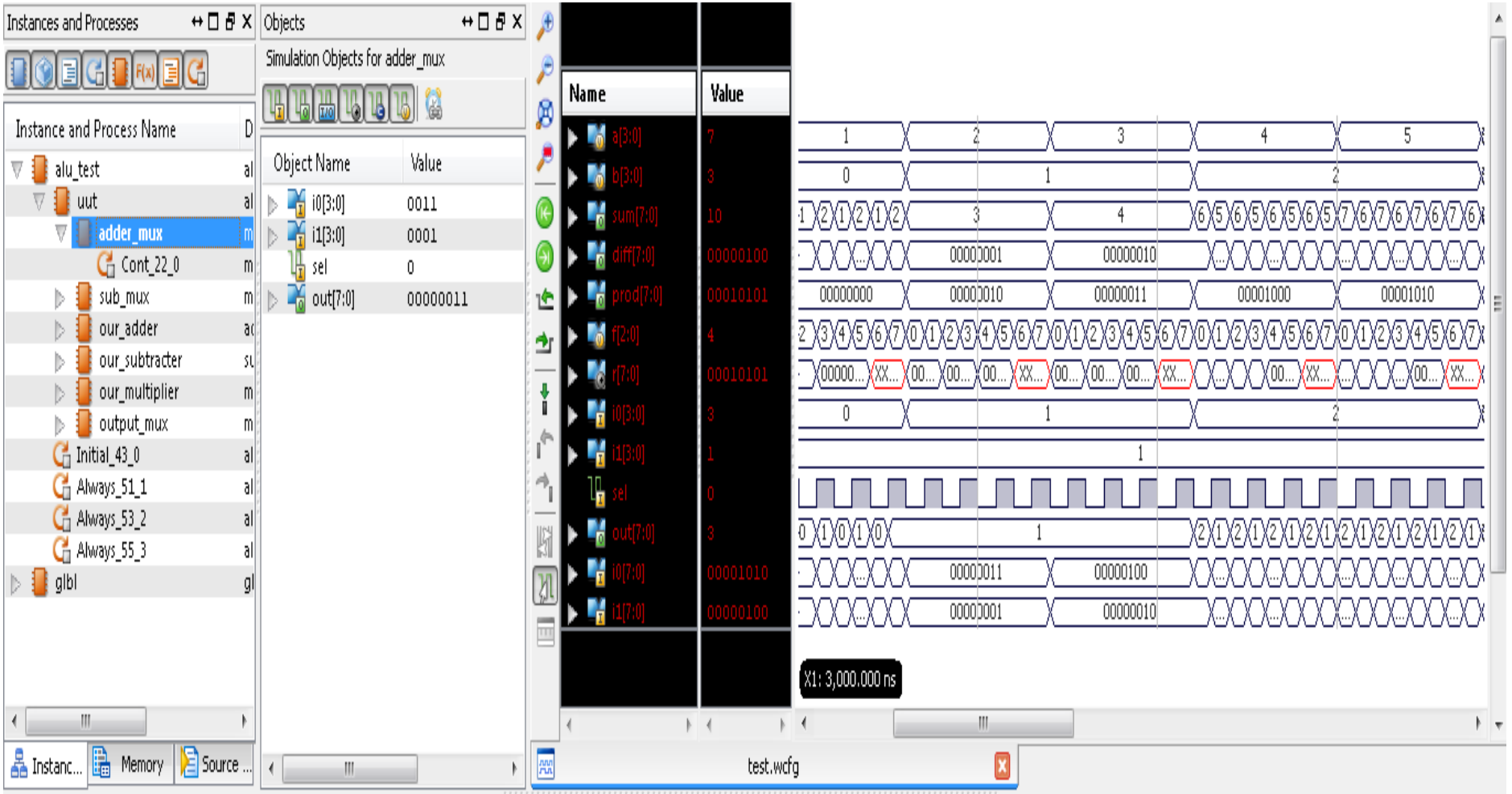
Descriere modul de test ALU

```
initial begin
        a = 0;
        b = 0;
        f = 0;
end

always # 4000
        a= a+1;
always # 800
        b= b+1;
always # 50
        f= f+1;
```



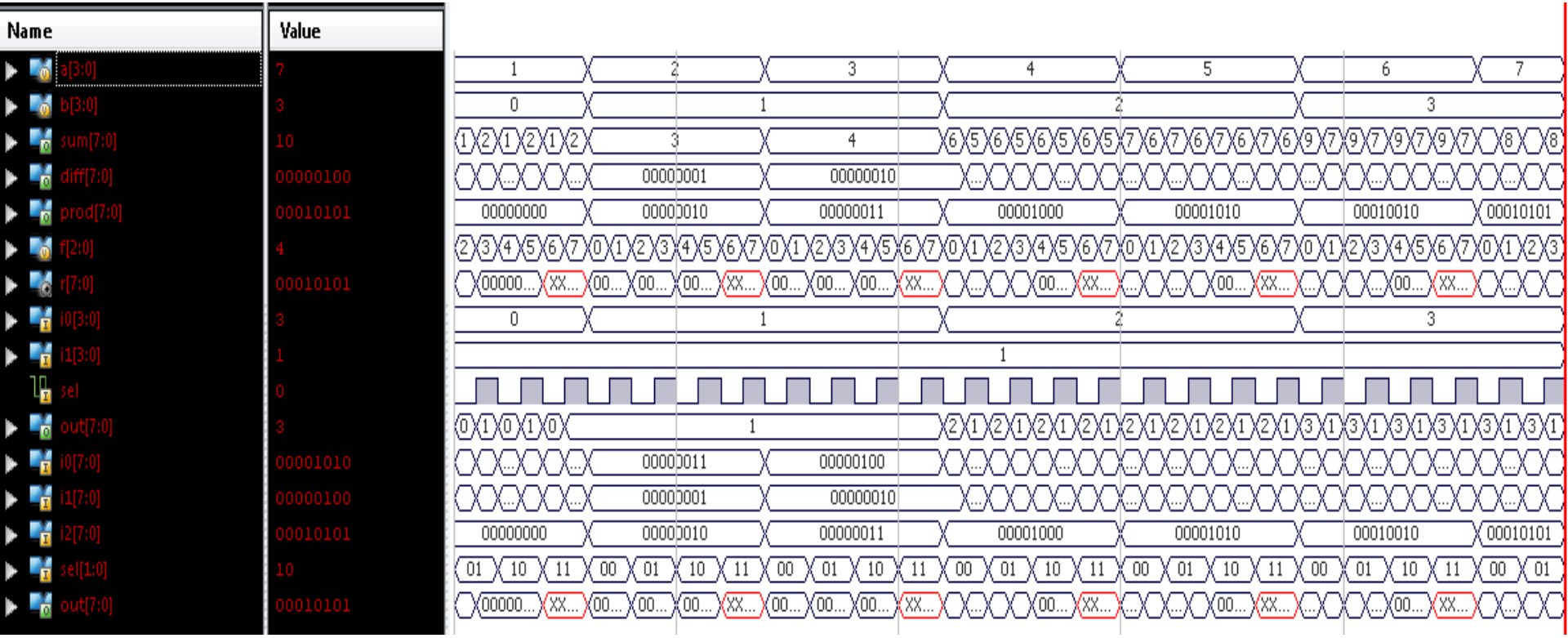
Verificarea stării conexiunilor interne

- Verificarea stării conexiunilor interne se poate face alegând modulul al cărui semnale dorim să le vizualizăm (în meniul din stânga).
- Semnalele modulului apar in fereastra din mijloc (Simulation Objects for…)
- Se trage semnalul dorit în fereastra din dreapta



Verificarea stării conexiunilor interne

- Detaliu asupra semnalelor ce se doresc a fi analizate.



Lab7_4: ALU pe 4 biți cu afișaj pe 7 segmente

- Creați un proiect nou Lab7_4
- Adăugați (Add copy of source) toate fișierele din Lab7_3 (alu_top, alu4_modules, Nexys.ucf).
- Adăugați modulul hex7seg din laboratorul 4.
- Adăugați o sursă nouă de tip „Verilog” (alu4_top). Aceasta va conecta modulele alu_top și hex7seg.

```
module alu4_top( input [3:0] a, input [3:0] b, input [2:0] f,
                 output [6:0] a_to_g, output [3:0] an, output dp);
wire [7:0] r1;

assign an = 4'b1110; // 1 digit on, 3 digits off
assign dp = 1; // dp off

hex7seg D4 (.x(r1[3:0]),.a_to_g(a_to_g));
alu_top D1 (.a(a), .b(b), .f(f), .r(r1));

endmodule
```

ALU implementare și testare

- Adăugați și adaptați fișierul Nexys.ucf astfel încât
  - F[2:0] <=> btn[2:0];
  - a [3:0] <=> sw [3:0],
  - b [3:0] <=> sw [7:4],
  - a_to_g [6:0] <=> a_to_g [6:0],
  - an[3:0] <=> an[3:0],
  - dp <=> dp
- Generați fișierul de configurare, încărcați-l în placă și testați funcționarea circuitului.
- Folosind sw [7:0] setați următorii operanzi:
  - a = 3, b = 2
- Folosind btn[2:0] setați pe rând cele 5 operații posibile. In fiecare caz observați corectitudinea rezultatului.

F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

F2 btn[2]	F1 btn[1]	F0 btn[0]	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	X	

Lucrarea de laborator nr. 8

- Circuite logice secvențiale simple:
  - Latch-uri de tip D
  - Bistabile de tip D
  - Bistabile de tip T

Lab8_1: Latch de tip D

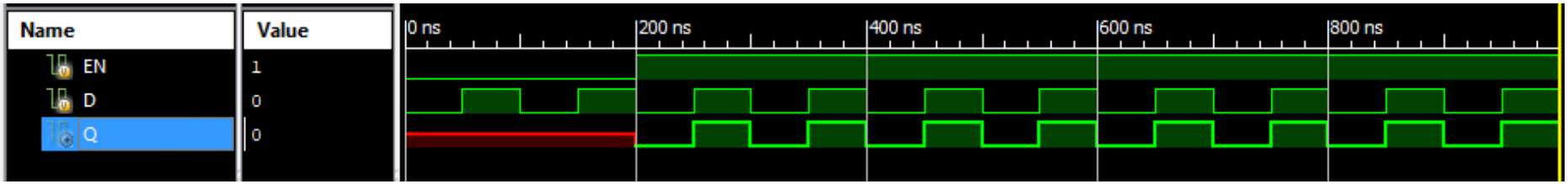
- Creați un proiect nou de tip HDL (Lab8_1)
- Adăugați o sursă nouă de tip Verilog Lab8_1.v
- Descrieți în limbaj Verilog comportamental un latch de tip D
- Simulați circuitul folosind semnalele de test descrise mai jos

D-latch – în cod Verilog

```
module v_Dlatch_G (input EN, D, output reg Q);
    always @(EN or D)
        begin
                if (EN)
                        Q = D;
end endmodule
```

Descriere semnale

```
// Add stimulus here
#100;
EN = 1;

end
always #50
   D <= ~ D;
```



Temă suplimentară:

- Descrieți și simulați un latch de tip D cu semnal de autorizare activ pe zero.

Lab8_2a: Bistabil de tip D

- Creați un proiect nou de tip HDL (Lab8_2a)
- Adăugați o sursă nouă de tip Verilog Lab8_2a.v
- Folosiți descrierea Verilog de mai jos (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//
//   Bistabil D activ pe front crescător de clock
//

module v_registers_1 (input C, D, output reg Q);

    always @(posedge C)
    begin
        Q <= D;
    end

endmodule
```

```
    // Add stimulus here
    #100;
end
always #100
    D <= ~ D;
always #70
    C <= ~ C;
```

Temă suplimentară:

- Descrieți și simulați un bistabil de tip D cu semnal clock cu front descrescător.

Lab8_2b: Bistabil de tip D

- Creați un proiect nou de tip HDL (Lab8_2b)
- Adăugați o sursă nouă de tip Verilog Lab8_2b.v
- Folosiți descrierea Verilog de mai jos (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//
// Bistabil D activ pe front descrescător de clock și semnal de ștergere asincron
//

module v_registers_2 (input C, D, CLR, output reg Q);

    always @(negedge C or posedge CLR)
    begin
        if (CLR)
            Q <= 1'b0;
        else
            Q <= D;
    end

endmodule
```

```
nitial begin
  // Initialize Inputs
  C = 0;
  D = 0;
  CLR = 0;

  // Wait 100 ns for global
  #100;
  CLR =1;
  # 100
  CLR =0;
   end
  always #100
  D<=~D;
  always #70
  C<= ~C;
```

Temă suplimentară:

- Descrieți și simulați un bistabil de tip D cu semnal clock cu front crescător și semnal de ștergere asincron pe front descrescător.

Lab8_2c: Bistabil de tip D

- Creați un proiect nou de tip HDL (Lab8_2c)
- Adăugați o sursă nouă de tip Verilog Lab8_2c.v
- Folosiți descrierea Verilog de mai jos (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//
//   Bistabil D activ pe front crescător de clock și semnal de autorizare clock
//

module v_registers_4 (input C, D, CE, output reg Q);

    always @(posedge C)
    begin
        if (CE)
            Q <= D;
    end

endmodule
```

Temă suplimentară:

- Descrieți și simulați un bistabil de tip D cu semnal clock cu front crescător și semnal de autorizare (CE) activ pe zero.

Lab8_2d: Bistabil de tip D

- Creați un proiect nou de tip HDL (Lab8_2d)
- Adăugați o sursă nouă de tip Verilog Lab8_2d.v
- Folosiți descrierea Verilog de mai jos (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//
// Bistabil D activ pe front crescător de clock și semnat  de setare sincron        t
//

module v_registers_3 (input C, D, S, output reg Q);

    always @(posedge C)
    begin
        if (S)
            Q <= 1'b1;
        else
            Q <= D;
        end

endmodule
```

Temă suplimentară:

- Descrieți și simulați un bistabil de tip D cu semnal de clock cu front descrescător și semnal de setare activ pe zero.

Lab8_3a: Bistabil de tip T cu reset asincron

- Creați un proiect nou de tip HDL (Lab8_3a)
- Adăugați o sursă nouă de tip Verilog Lab8_3a.v
- Folosiți descrierea Verilog de mai jos pentru un bistabil de tip T cu reset asincron (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//-----------------------------------------------------
//   Bistabil T cu reset asincron
//-----------------------------------------------------
module tff_async_reset ( input data, clk, reset, output reg q);

always @ ( posedge clk or negedge reset)
if (~reset) begin
  q <= 1'b0;
end else if (data) begin
  q <= !q;
end

endmodule
```

Lab8_3b: Bistabil de tip T cu reset sincron

- Creați un proiect nou de tip HDL (Lab8_3b)
- Adăugați o sursă nouă de tip Verilog Lab8_3b.v
- Folosiți descrierea Verilog de mai jos pentru un bistabil de tip T cu reset asincron (codurile se pot descărca de pe site-ul laboratorului)
- Simulați circuitul folosind un mediu de testare descris în Verilog (Verilog test fixture)

```
//-------------------------------------------------------
//   Bistabil T cu reset sincron
//-------------------------------------------------------
module tff_sync_reset ( input data, clk, reset, output reg q);

always @ ( posedge clk)
if (~reset) begin
  q <= 1'b0;
end else if (data) begin
  q <= !q;
end

endmodule
```

Întrebări

1. Care este diferența între un latch de tip D și un bistabil de tip D și în ce porțiune de cod apare această diferență ?
2. Ce înseamnă instrucțiunea a #50 din fișierul test fixture ?
3. Ce tip de funcționare descrie codul următor:
    1. Always @ #50
    2. D<= ~ D 4.
4. Instrucțiunea a #50 poate fu implementată?
5. Prin ce diferă codul unui bistabil de tip T cu reset sincron de cel cu reset asincron?

Lucrarea de laborator nr. 9

- Simularea numărătoarelor
  - Numărător direct sincron pe 4 biți cu semnal de ștergere sincron/asincron
  - Numărător invers sincron pe 4 biți cu semnal de ștergere sincron
  - Numărător reversibil sincron pe 4 biți
  - Numărător zecimal direct cu intrare de încărcare
  - Numărător direct sincron pe N biți cu semnal de ștergere asincron

Simularea numărătoarelor sincrone

Temele 9_1 - 9_5

Următoarea descriere este comună temelor 9_1 – 9_5.

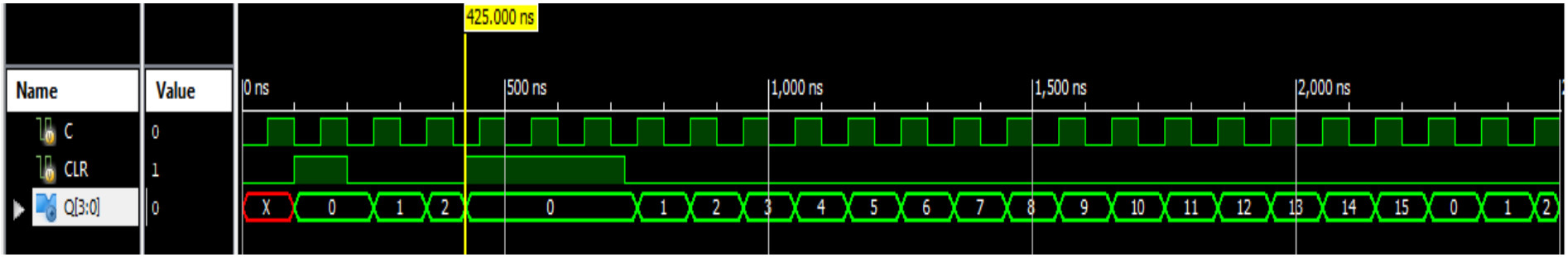
- Creați un proiect HDL (Lab9_x).
- Adăugați o sursă nouă de tip Verilog Lab9_x.v.
- Descrieți numărătorul în limbaj Verilog comportamental.
- Adăugați un fișier nou de tip Verilog test fixture.
- Specificați semnalele de testare.
- Simulați circuitul.

9_1a: Numărător direct sincron pe 4 biți cu semnal de ștergere asincron

```
//
//  Numărător direct pe 4 biți cu ștergere asincronă
//
module counter_1 (input C, CLR, output reg
[3:0] Q);

      always @(posedge C or posedge CLR)
      begin
            if (CLR)
                        Q <= 4'b0000;
            else
                        Q <= Q + 1'b1;
      end

endmodule
```

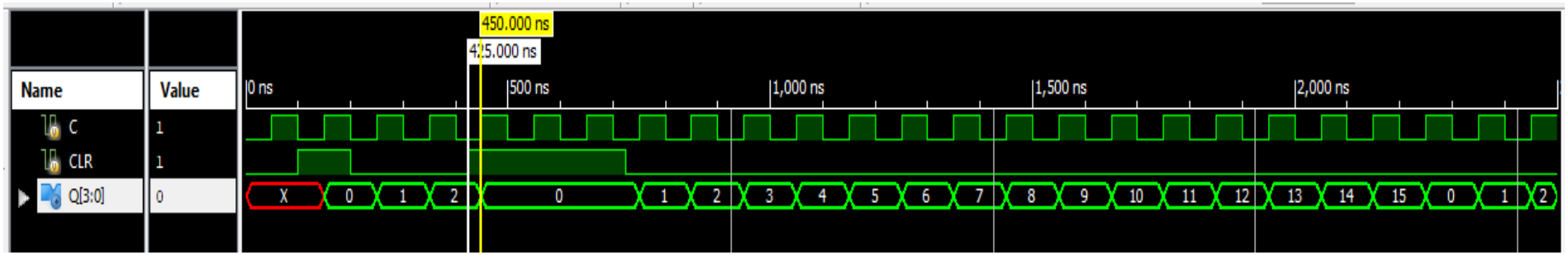
Specificarea semnalelor de testare

```
initial begin // Initialize Inputs
C = 0;
CLR = 0;
// Wait 100 ns for global reset to finish
#100;
// Add stimulus here
CLR = 1;
#100;
CLR = 0;
# 225
CLR = 1;
# 300
CLR = 0 ;
end
always #50
C <= ~ C;
```



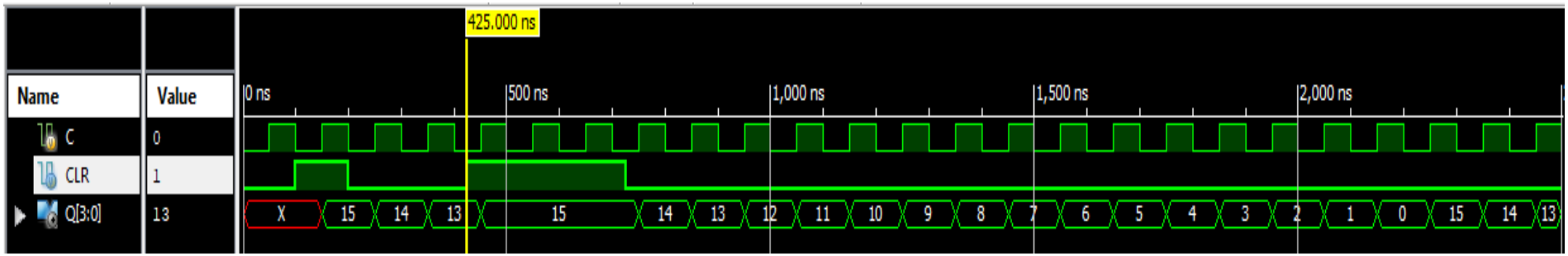
9_1b: Numărător direct sincron pe 4 biți cu semnal de ștergere sincron

```
//
//  Numărător direct pe 4 biți cu ștergere sincronă.
//
module counter_2 (input C, CLR, output reg [3:0] Q);

     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b0000;
          else
                    Q <= Q + 1'b1;
     end

endmodule
```



9_2: Numărător invers sincron pe 4 biți cu semnal de ștergere sincron

```
module counter_3 (input C, CLR, output reg [3:0] Q);
     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b1111;
          else
                    Q <= Q - 1'b1;
     end
endmodule
```

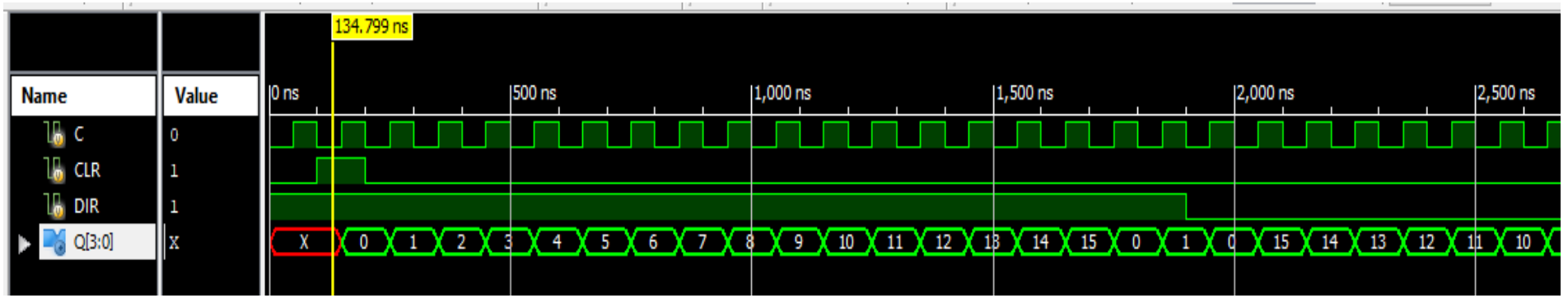


9 3: Numărător reversibil sincron pe 4 biți

```
module counter_4 (input C, CLR, DIR, output
reg [3:0] Q);
      always @(posedge C)
      begin
            if (CLR)
                        Q <= 4'b0000;
            else if (DIR)
                        Q <= Q + 1'b1;
            else
                        Q <= Q - 1'b1;
      end
endmodule
```

Specificarea semnalelor de testare

```
initial begin
    // Initialize Inputs
    C = 0;
    CLR = 0;
    DIR = 1;

    // Wait 100 ns for global reset to finish
    #100;
    // Add stimulus here
    CLR = 1;
    #100;
    CLR = 0;
    #1700;
    DIR=0;

end

always #50
    C <= ~ C;
```

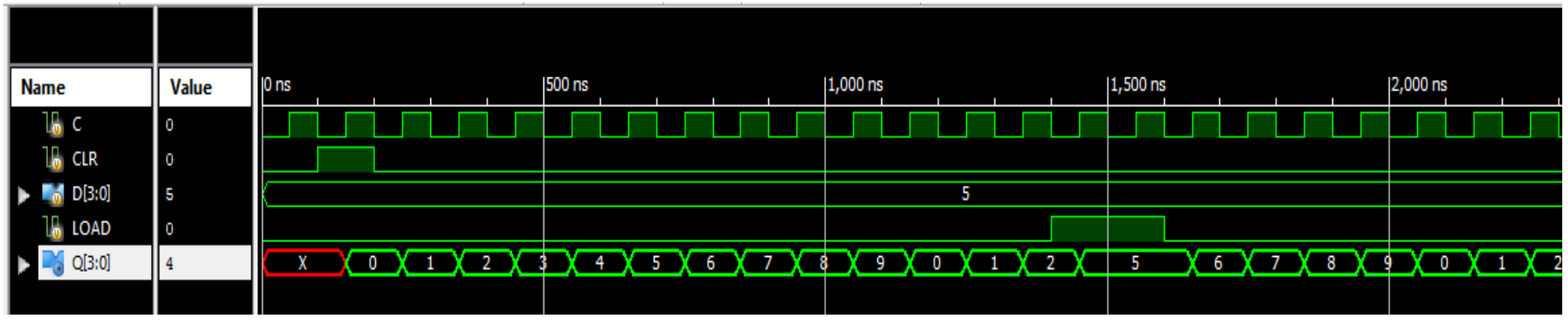


9_4: Numărător zecimal direct cu intrare de încărcare

```
module counter_5 (input C, CLR, LOAD, input [3:0]
D, output reg [3:0] Q);

assign q9 = (Q== 4'd9); //assign q12 = (Q== 4'd12);

always @(posedge C)
          begin
          if (CLR | q9)
                    Q <= 4'b0000;
          else if (LOAD) // (LOAD==1)
                    Q <= D; //sau o constanta;
          else
                    Q <= Q + 1'b1;
          end
endmodule
```

Specificarea semnalelor de testare

```
    initial begin
        // Initialize Inputs
        C = 0;
        CLR = 0;
        LOAD = 0;
        D = 4'b0101;;

        // Wait 100 ns for global reset
        #100;
        // Add stimulus here
        CLR = 1;
        #100;
        CLR = 0;
        #1200;
        LOAD = 1;
        # 200;
        LOAD = 0;

    end

    always #50
    C <= ~ C;
```

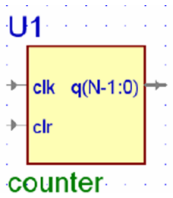


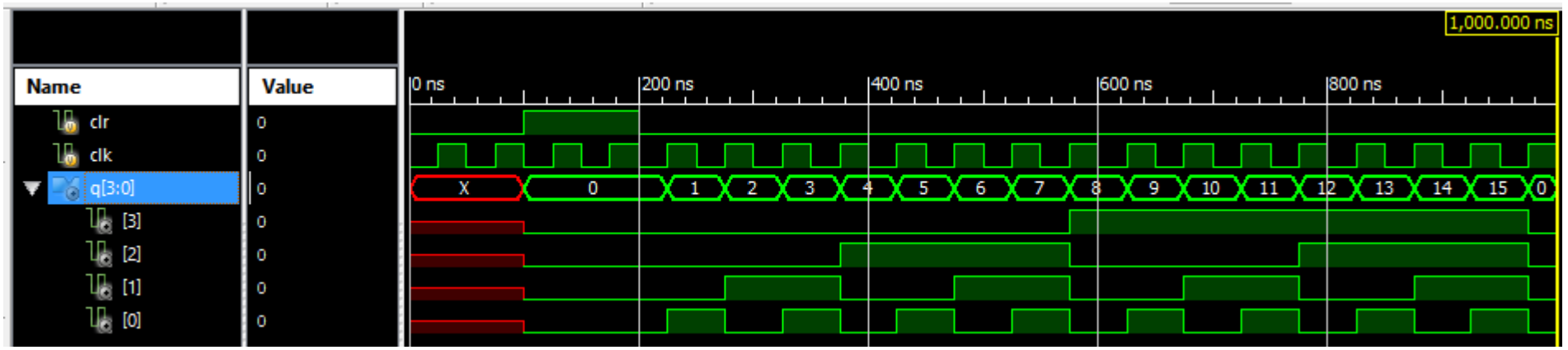
9_5: Numărător direct sincron pe N biți cu semnal de ștergere asincron

```
module counter
#(parameter N = 4)
   (input wire clr , clk ,
   output reg [N-1:0] q );
// N-bit counter

always @(posedge clk or posedge clr)
   begin
      if(clr == 1)
         q <= 0;
      else
         q <= q + 1;
   end
endmodule
```

Specificarea semnalelor de testare

```
initial begin
   // Initialize Inputs
   clr = 0;
   clk = 0;

   // Wait 100 ns for global reset to finish
   #100 clr = 1;
   #100 clr = 0;

   // Add stimulus here

end
   always #25
   clk <= ~clk;
```





Simularea numărătorului pe N biți arată că ieșirile *q*[*i*] sunt semnale dreptunghiulare cu ieșirea *q*[0] având frecvența egală cu jumătate din frecvența semnalului de clock, ieșirea *q*[1] are frecvența egală cu jumătate din frecvența semnalului *q*[0], etc.

Lucrarea de laborator nr. 10

- Implementarea numărătoarelor
  - Divizor de clock
  - Numărător pe 8 biți cu ieșire pe leduri
  - Numărător pe 8 biți cu afișare pe 7 segmente

10_1: Divizor de clock

- Placa de dezvoltare Nexys-2 are un semnal de clock de 50 MHz
- În exemplu care urmează se prezintă modalitatea de proiectare în Verilog a unui numărător pe N biți care poate fi utilizat pentru a genera un semnal de clock de frecvență mai mică
- Fiecare bistabil din componența numărătorului va diviza cu 2 frecvența semnalului de intrare:
  - q[0]= clk/2; q[1]= q[0]/2 …
  - Vom folosi un numărător pe 26 biți, astfel la ieșirea q[25] vom obține un semnal cu frecvența de 0,75 HZ la q[24] aprox 1,5 Hz…

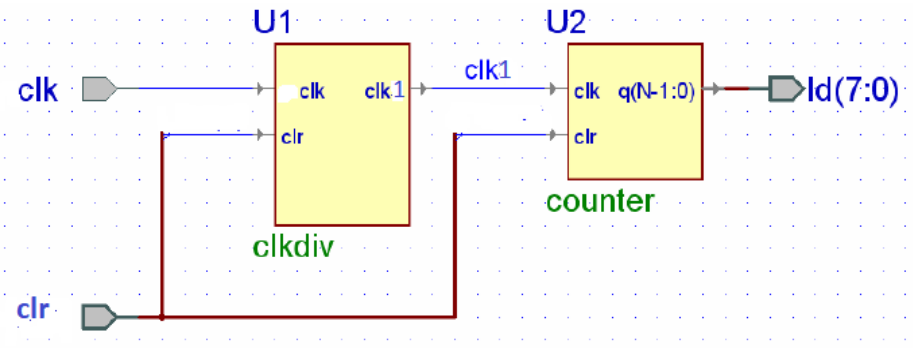
Q(i)	Frecvența (Hz)	Perioda (ms)
	50000000.00	0.00002
0	25000000.00	0.00004
1	12500000.00	0.00008
2	6250000.00	0.00016
3	3125000.00	0.00032
4	1562500.00	0.00064
5	781250.00	0.00128
6	390625.00	0.00256
7	195312.50	0.00512
8	97656.25	0.01024
9	48828.13	0.02048
10	24414.06	0.04096
11	12207.03	0.08192
12	6103.52	0.16384
13	3051.76	0.32768
14	1525.88	0.65536
15	762.94	1.31072
16	381.47	2.62144
17	190.73	5.24288
18	95.37	10.48576
19	47.68	20.97152
20	23.84	41.94304
21	11.92	83.88608
22	5.96	167.77216
23	2.98	335.54432
24	1.49	671.08864
25	0.745	1342.17728

10_1: Implementare divizor de clock

```
module clkdiv ( input clk, clr, output clk1, clk2, clk3 );
reg [25:0] q;
// 26-bit counter
always @(posedge clk or posedge clr)
        begin
                if(clr == 1)
                        q <= 0;
                else
                        q <= q + 1;
                end
assign clk1 = q[25]; // ~0.75 Hz
assign clk2 = q[24]; // ~1.5
assign clk3 = q[23]; // ~3 Hz
endmodule
```

- Adăugați fișierul de constrângeri (ucf)
  - Intrări
    - clk (la pinul B8): NET "clk" LOC = "B8";
    - clr
  - Ieșiri
    - clk1 -> led<0>
    - clk2 -> led<1>
    - clk3 -> led<2>
- Generați fișierul de configurare, încărcați în placă și testați funcționarea circuitului.

10_2: Numărător cu ieșire pe leduri




```
module count8_top (input clk, input  clr,
output [7:0] led) ;

wire clk1;

clkdiv
U1(.clk1(clk1),.clr(clr),.clk(clk));

counter #( .N(8)) U2
(.clk(clk1), .clr(clr), .q(led[7:0]));
endmodule
```

- Creați un proiect nou(Lab10_2)
- Adăugați modulele clkdiv și numărătorul pe N biți create la temele anterioare.
- Adăugați un fișier nou de tip VERILOG (Lab10_2.v), acesta va fi modulul care va conecta cele două module anterioare, ca în figură.
- Adăugați fișierul de constrângeri specificând
  - Intrările: clk și clr,
  - Ieșirile: 8 leduri.
- Generați fișierul de configurare, încărcați în placă și testați funcționarea circuitului.
  - Observați secvența de numărare binară pe leduri.
  - Ștergeți numărătorul folosind butonul declarat

10_3: Numărător cu afișare pe 7 segmente

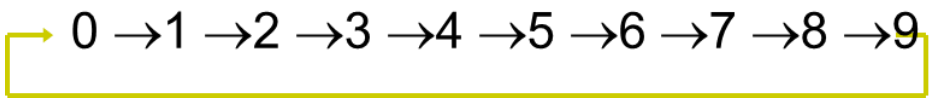
- Creați un proiect nou (Lab10_3)
- Adăugați modulele clkdiv și numărătorul pe N biți create la temele anterioare.
- Adăugați o copie a modulului hex7seg creat la o temă anterioară.
- Adăugați un fișier nou de tip VERILOG (Lab10_3.v), acesta va fi modulul care va conecta cele trei module adăugate la proiect.

```
module count4_top (input clk, input  clr, output [7:0] led,
          output [6:0] a_to_g, output [3:0] an, output dp )
;

wire clk1;

clkdiv U1(.clr(clr),.clk(clk), .clk1(clk1));
counter #( .N(8)) U2 (.clk(clk1), .clr(clr), .q(led[7:0]));
hex7seg U3 (.x(led[3:0]), .a_to_g(a_to_g));

assign an = 4'b1110;
assign dp = 1; // dp off

endmodule
```

- Adăugați fișierul de constrângeri specificând
  - Intrările: clk, clr,
  - Ieșirile: an, a_to_g, dp, led.
- Generați fișierul de configurare, încărcați în placă și testați funcționarea circuitului.
  - Observați secvența de numărare binară pe afișajul cu 7 segmente.
  - Ștergeți numărătorul folosind butonul declarat

10_4: Numărător decadic (modulo 10)

- Creați un proiect nou (Lab10_4)
- Adăugați toate modulele de la tema 10_3.
- Modificați numărătorul astfel încât secvența de numărare să se fie:



- Adăugați fișierul de constrângeri :
  - Intrările: clk, clr,
  - Ieșirile: an, a_to_g, dp, led.
- Generați fișierul de configurare, încărcați în placă și testați funcționarea circuitului.
- Observați secvența de numărare BCD pe leduri și afișajul cu 7 segmente.
- Ștergeți numărătorul folosind butonul declarat

Sugestie pentru modificarea numărătorului: adăugați la condiția de ștergere și un semnal care indică atingerea stării finale dorite (ca și în exemplul 9_4):

```
if (clr | q9)
```

Unde q9 se obține astfel:

```
assign q9 = (q== 4'd9);
```

10_5: Numărător modulo 13

- Creați un proiect nou (Lab10_5)
- Adăugați toate modulele de la tema 10_2.
- Modificați numărătorul astfel încât să descrie un numărător modulo 13
- Adăugați fișierul de constrângeri :
  - Intrările: clk, clr,
  - Ieșirile: an, a_to_g, dp, led.
- Generați fișierul de configurare, încărcați în placă și testați funcționarea circuitului.
- Observați secvența de numărare BCD pe leduri și afișajul cu 7 segmente.
- Ștergeți numărătorul folosind butonul declarat

Lucrarea de laborator nr. 11

- Registre
  - Registru pe 4 biți, cu PRESET asincron și autorizare a semnalului de clock
  - Registru de deplasare stânga pe 8 biți
  - Registru paralel-serie pe 8 biți cu încărcare asincronă și deplasare stânga
  - Registru serie-paralel pe 8 biți cu deplasare stânga/dreapta și extragere paralelă
  - Numărător în inel
  - Numărător Johnson

Lab11_1: Registru paralel-paralel pe 4 biți

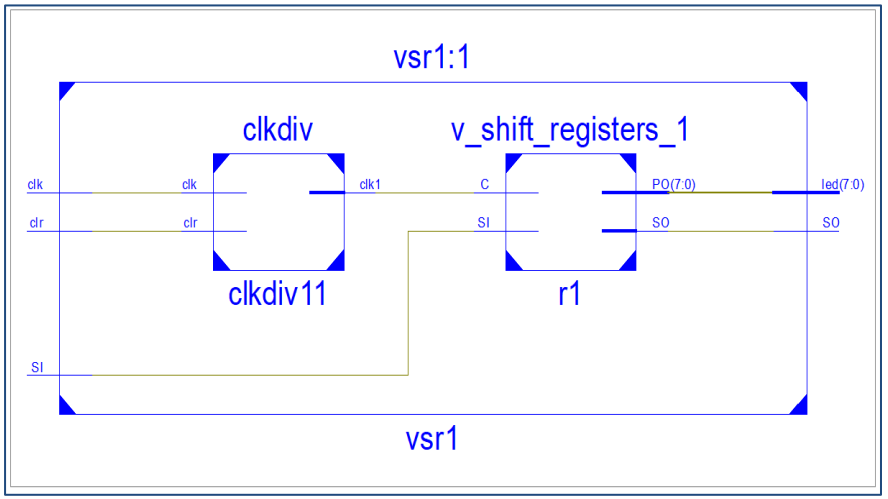
- Creați un proiect nou (Lab11_1)
- Adăugați o sursă nouă de tip Verilog (Lab11_1.v).
- Descrieți funcționarea unui registru paralel-paralel pe 4 biți cu clock activ pe front crescător și semnale de PRESET asincron și autorizare a semnalului de ceas.
- Adăugați un fișier de testare Verilog test fixture.
- Descrieți semnalele de test. Testați funcționarea circuitului cu ajutorul simulării.

```
//
// Registru de 4 biți clock activ pe front crescător, Set asincron și Clock Enable
//
module v_registers_5 (input C, CE, PRE, input [3:0 ] D,
       output reg [3:0] Q);

       always @(posedge C or posedge PRE)
       begin
              if (PRE)
              Q <= 4'b1111;
              else if (CE)
              Q <= D;
       end
endmodule
```

Lab11_2: Registru de deplasare pe 8 biți

- Creați un proiect nou (Lab11_2)
- Adăugați o sursă nouă de tip Verilog (Lab11_2.v).
- Descrieți funcționarea unui registru de deplasare pe 8 biți.
- Testați funcționarea circuitului cu ajutorul simulării. (Facultativ)
- Adăugați un divizor de clock pentru a genera un semnal de aproximativ 1 Hz și folosiți acest semnal ca semnal de clock pentru registru
- Conectați cele două module ca în figura alăturată într-un fișier top-module
- Adăugați fișierul de constrângeri, generați fișierul de configurare și testați funcționarea circuitului în placa de dezvoltare

```
// Registru de deplasare pe 8 biți cu Clock activ
// pe front crescător, Serial In și Serial Out

module v_shift_registers_1 (input C, SI,
output reg [7:0] PO, output SO);

    always @(posedge C)
    begin
          PO <= PO << 1;
          PO[0] <= SI;
    end

    assign SO = PO[7];

endmodule
```



Lab11_3: Registru paralel-serie pe 8 biți

- Creați un proiect nou (Lab11_3) și implementați un registru paralel-serie pe 8 biți cu încărcare asincronă și deplasare spre stânga
- Testați funcționarea registrului în mod similar cu cel anterior.

```
// Registru paralel-serie pe 8 biți
// Încărcare paralelă, Serial In și Serial Out

module v_shift_registers_6 (C, ALOAD, SI, D, SO);
    input C,SI,ALOAD;
    input [7:0] D;
    output SO;
    reg [7:0] tmp;

    always @(posedge C or posedge ALOAD)
    begin
        if (ALOAD)
            tmp <= D;
        else
            tmp <= {tmp[6:0], SI};
    end

    assign SO = tmp[7];

endmodule
```

Lab11_4: Registru serie-paralel pe 8 biți

- Creați un proiect nou (Lab11_4) și implementați un registru serie-paralel pe 8 biți cu deplasare stânga/dreapta și extragere paralelă
- Testați funcționarea registrului în mod similar cu cel anterior.

```
// Registru serie-paralel pe 8 biți cu deplasare stânga/dreapta,
// Serial In și Paralel Out

module v_shift_registers_8 (C, SI, LEFT_RIGHT, PO);
    input C,SI,LEFT_RIGHT;
    output PO;
    reg [7:0] Q;

    always @(posedge C)
    begin
        if (LEFT_RIGHT==1'b0)
          Q <= {Q[6:0], SI};
        else
          Q <= {SI, Q[7:1]};
    end

    assign PO = Q;

endmodule
```

Lab11_5: Numărător în inel

- Creați un proiect nou (Lab11_5) și implementați un numărător în inel pe 8biți
- Testați funcționarea registrului în mod similar cu cel anterior.

```
module ring_count(input clk, load, output reg [7:0]q);

  always @(posedge clk)
       if(load==1)
            q<=8'b10000000;
        else
            begin
                q <= {q[0], q[7:1]};
            end
endmodule
```

Lab1_6: Numărător Johnson

- Creați un proiect nou (Lab11_5) și implementați un numărător Johnson pe 8biți
- Testați funcționarea registrului în mod similar cu cel anterior.

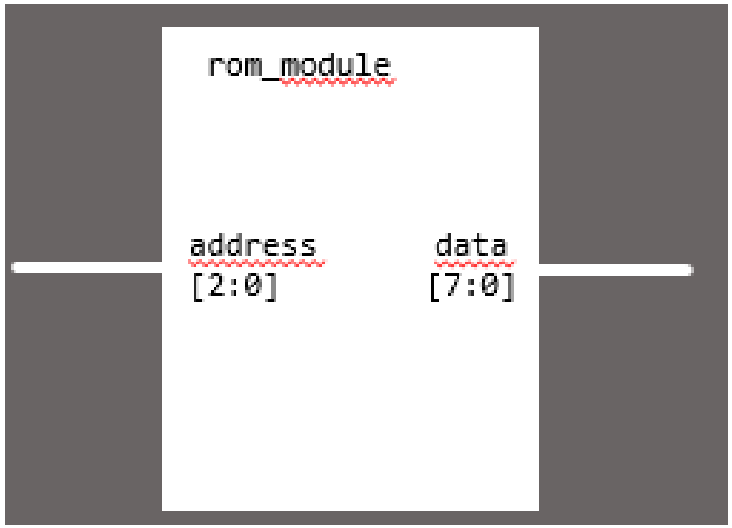
```
module johnson_count(input clk, clr, output reg [7:0]q);

  always @(posedge clk)
       if(clr==1)
            q<=8'b00000000;
        else
            begin
                q <= {~q[0], q[7:1]};
            end
endmodule
```

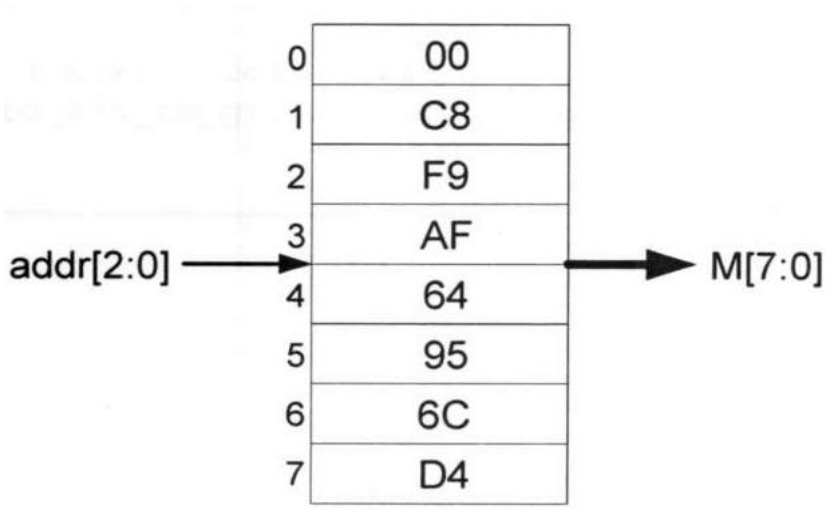
Lucrarea de laborator nr. 12

- Memorii
  - Verilog ROM
  - Distributed RAM/ROM
    - Simulare
    - Implementare
  - Block RAM

Lab12_1 Memorie ROM simplă

- Scopul: crearea unei memorii ROM simple cu conținutul specificat în cod Verilog din care datele pot fi citite de la adresa specificată
- Parametrii: 3-biți de adresă: 2 ^ 3 = 8 adrese
- La fiecare adresă este stocat un byte
- Rezultă o memorie ROM de 8 bytes





De exemplu la specificarea adresei 3'b110 ieșirea va fi 'h6C

ROM Simplu

```
`timescale 1ns / 1ps

module rom_module(input wire[2:0] address, output wire[7:0] data);

    reg [7:0] rom [0:7];

    parameter init_data = 64'h00C8F9AF64956CD4;

    integer i;

    initial
    begin
        for(i=0; i<8; i=i+1)

            rom[i] = init_data[63-i*8 -: 8];
    end

    assign data = rom[address];

endmodule
```

reg – registru
reg [7:0] – registru de 8 biți
reg [7:0] rom [0:7] –
Arie de 8 registre de 8 biți
rom este numele ariei

Exemplu de ROM de 1 kilobyte:
reg [7:0] rom [0:1024]

Constantă de 64-biți hex

Top modul

- Intrări de la comutatoare (SW)
- Ieșiri pe leduri

```
module topmodule(input [2:0] SW, output [7:0] LED) ;

    rom_module memory(.address(SW), .data(LED));

endmodule
```

- Instanțierea modulului ROM folosind numele de instanță „memory” (poate fi orice nume)

- Adăugați fișierul de constrângeri
- Implementați și testați
- Generați adresele folosind comutatoarele.
- Verificați pe leduri conținutul de la adresa specificată

Lab12_2. Memorie RAM/ROM distribuită

- Creați un proiect nou (Lab12_2) și implementați o memorie ROM distribuită de 16 octeți
- Adăugați un fișier Verilog top modul

```
module Lab12_2top(
        input [3 : 0] addr,
        output [7 : 0] spo
    );

endmodule
```

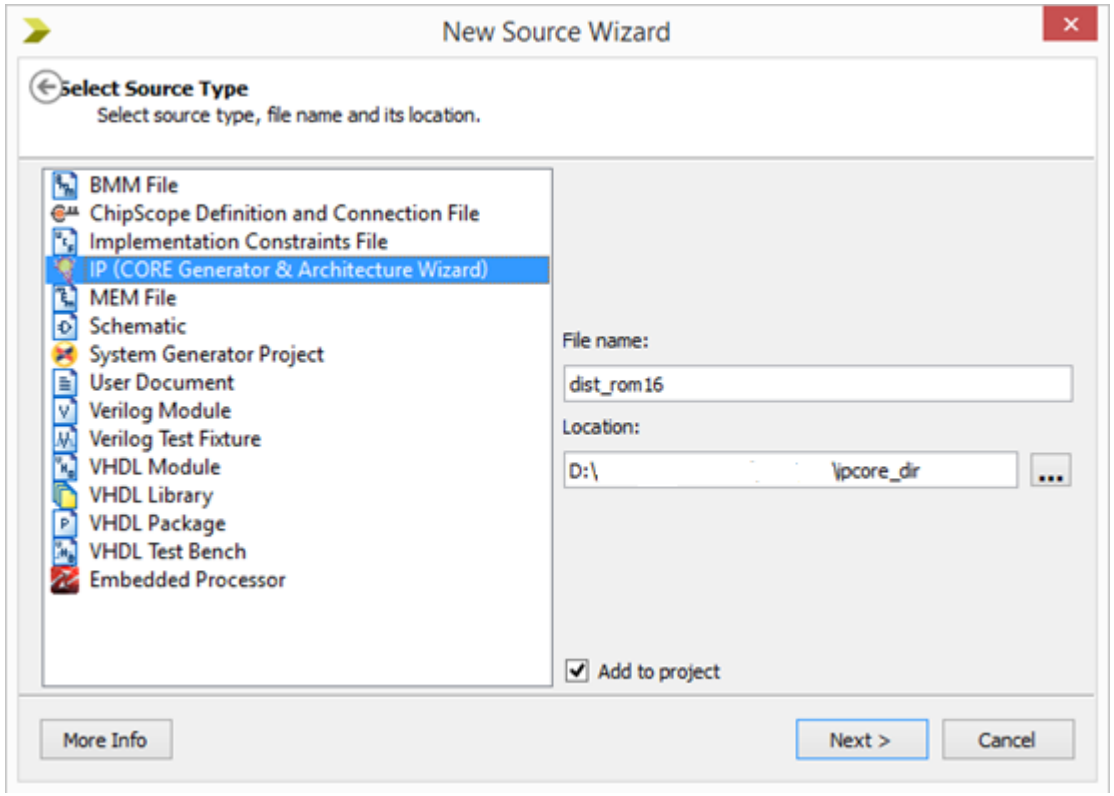
Fișierul de inițializare a memoriei

- Avem nevoie de un fișier de inițializare a memoriei (MIF = Memory Initialization file)
- Folosind un editor de texte (de exemplu Notepad++) creați fișierul cu conținutul prezentat:

```
;Initialization file for a 16x8 distributed ROM
memory_initialization_radix = 16;
memory_initialization_vector =
0 C8 F9 AF
64 95 6C D4
39 E7 5A 96
84 37 28 4C;
```

- Salvați fișierul cu numele, Lab12_2_init.coe. IMPORTANT fișierul trebuie să aibă extensia.coe (nu txt)!

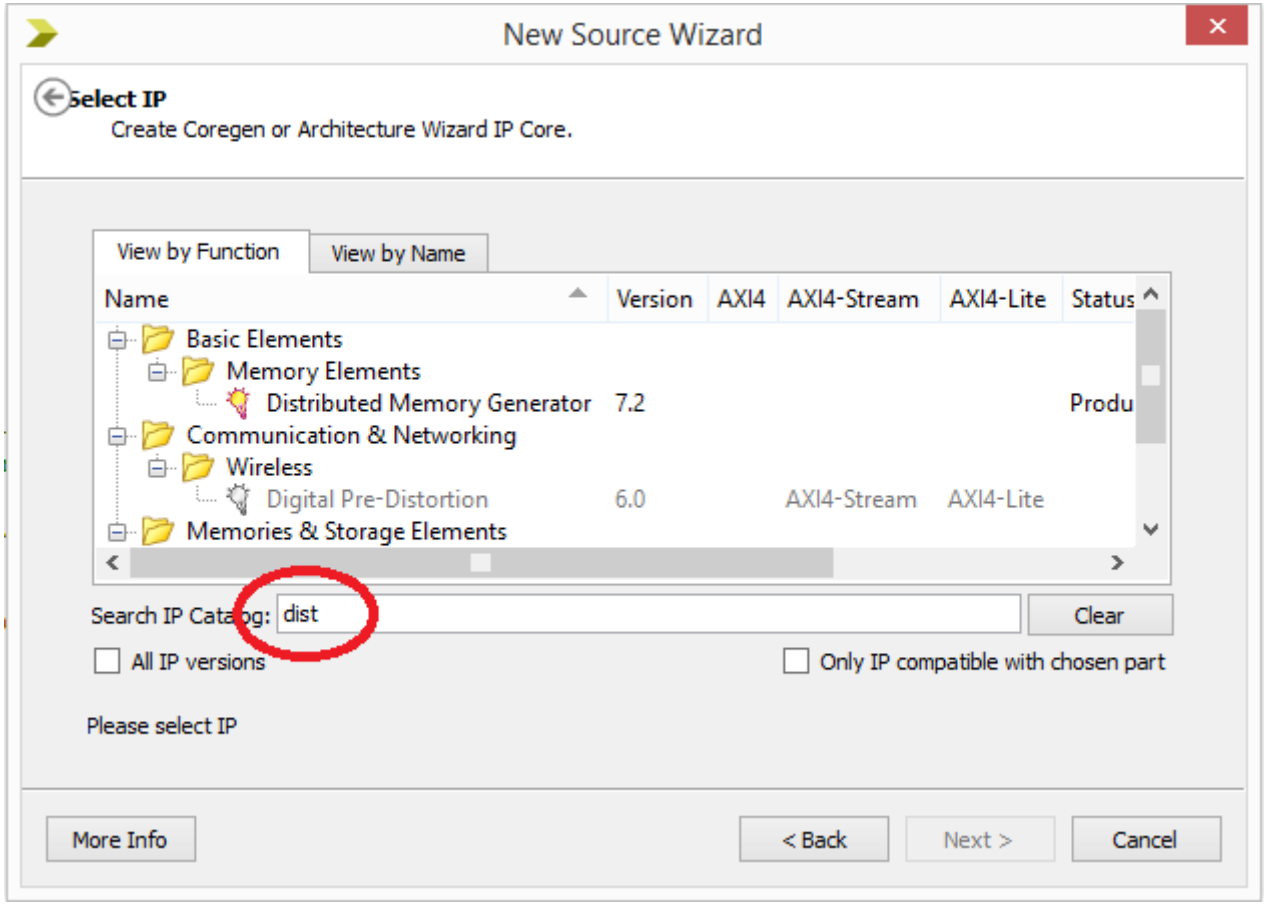
Folosirea blocurilor IP



1. Adăugați o nouă sursă de tip IP Core.
2. Numele sursei poate fi „dist_rom16”
3. Apăsați Next.

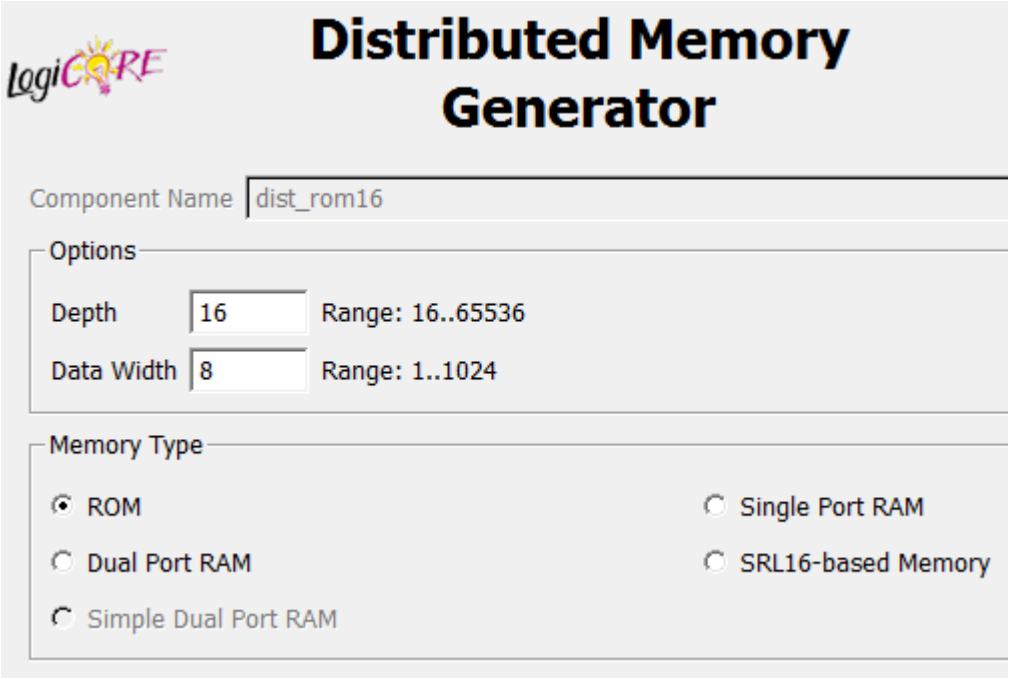
Distributed RAM/ROM

- Alegeți tipul *Distributed Memory Generator*.



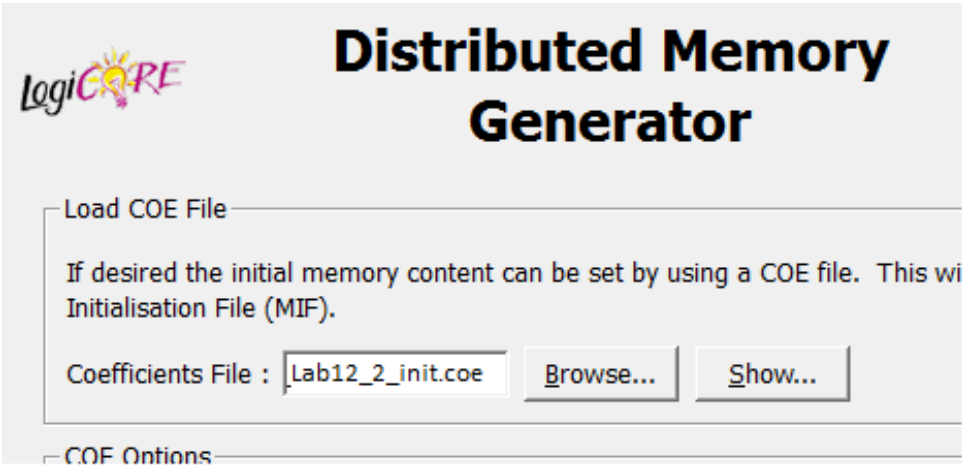
Setarea parametrilor memoriei

- Depth:16
- Data width: 8
- Type: ROM



Adăugarea fișierului de inițializare

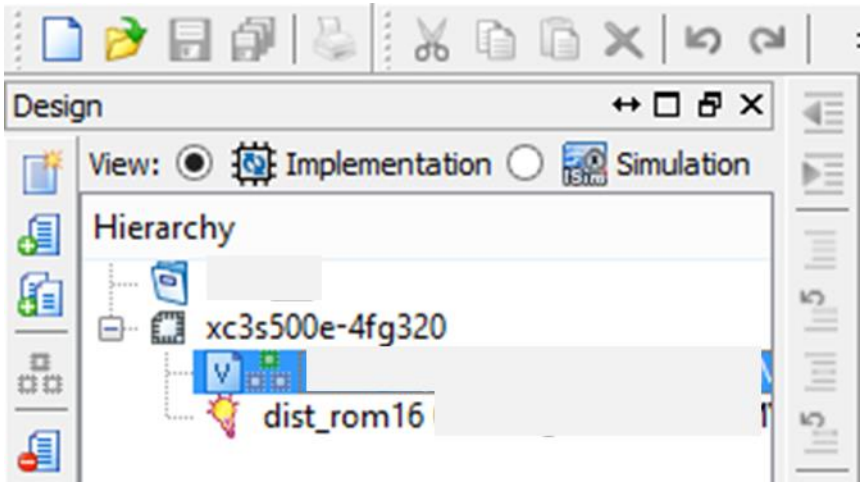
- Localizați fișierul Lab12_2_init.coe

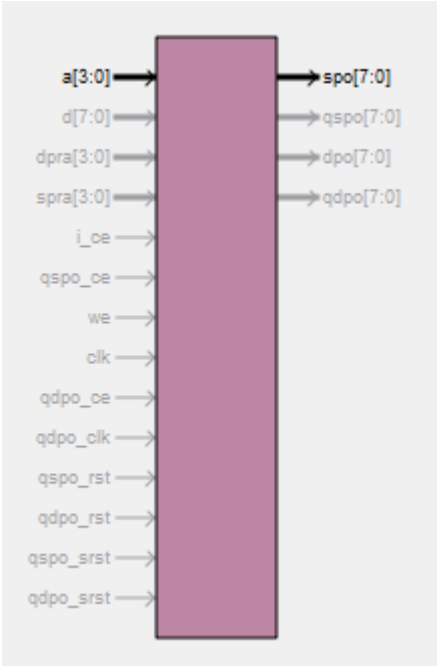


- Apăsând Show se poate vizualiza conținutul fișierului.
- Dacă numele fișierului este cu font roșu, fișierul conține erori.

Generarea și instanțiera memoriei

Apăsați Generate și așteptați până modulul va fi generat.





Instanțiera memoriei

```
module Lab12_2top(
        input [3 : 0] addr,
        output [7 : 0] spo
    );

dist_rom16 MEM(
  .a(addr), // input [3 : 0] a
  .spo(spo) // output [7 : 0] spo
);

endmodule
```

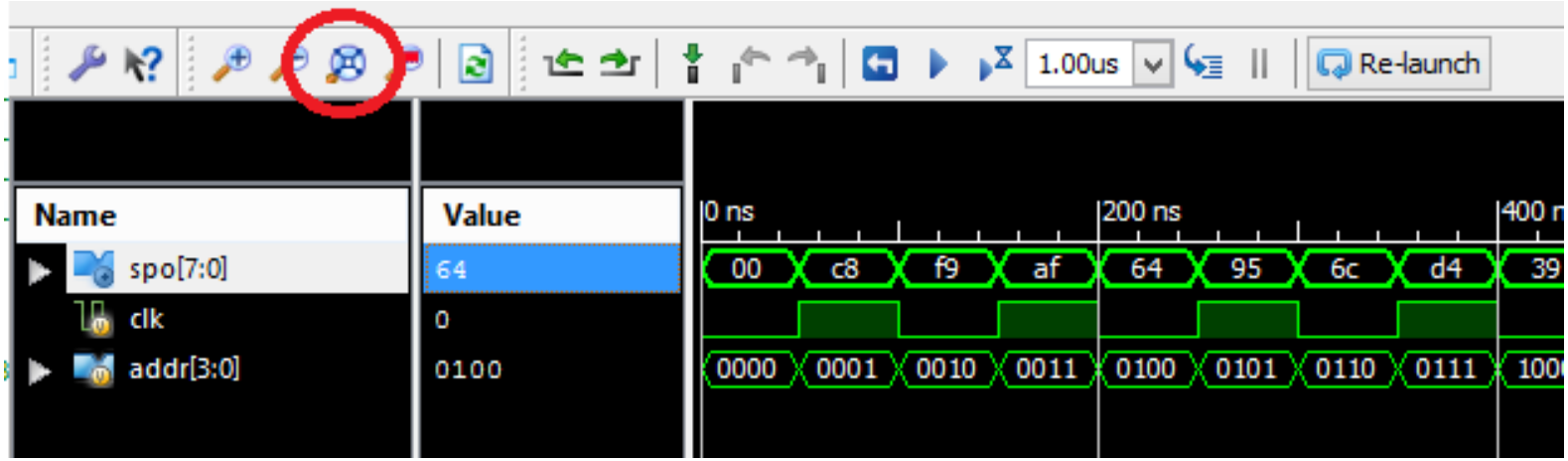
Adăugarea unui modul de test

- Adăugați un modul de test: Verilog Test Fixture
- Generați semnalele de test ca în exemplul următor

```
initial begin
    // Initialize Inputs
    clk = 0;
    addr = -1;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here

end

always@(*)
    begin
        #50
        clk <= ~clk;
    end

always@(clk)
    begin

        addr = addr + 1;
    end
```

Simulare și implementare

- Alegeți reprezentarea hexazecimală pentru conținutul memoriei (Radix – Hexadecimal)



- Comparați rezultatul simulării cu conținutul fișierului .coe

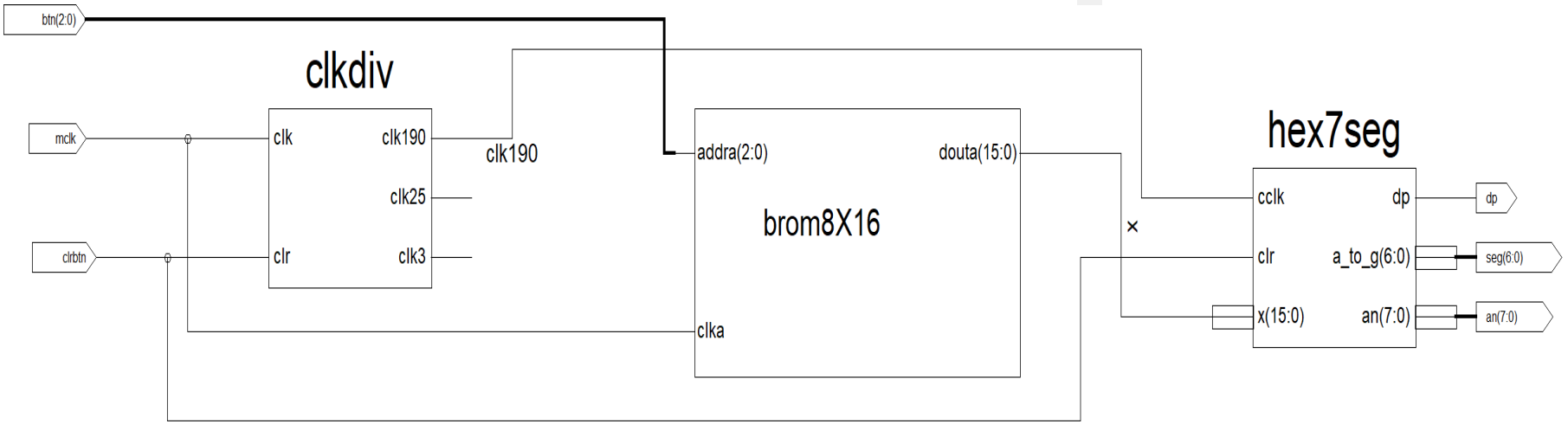
Implementare

- Adăugați fișierul de constrângeri .ucf
- Implementați, încărcați în placă și testați
- Generați adresele folosind comutatoarele.
- Verificați pe leduri conținutul de la adresa specificată

Lab12_3 BLOCK ROM

1. Creați un proiect nou (Lab12_3) și implementați o memorie de tip Block ROM
1. Adăugați un fișier Verilog top modul
2. Adăugați porturile ca în figura următoare

```
//////////////////////,
module  Lab12_3_top (
input wire mclk ,
input wire [3:0] btn ,
input wire clrbtn ,
output wire [6:0] seg ,
output wire dp ,
output wire [7:0] an
) ;
```



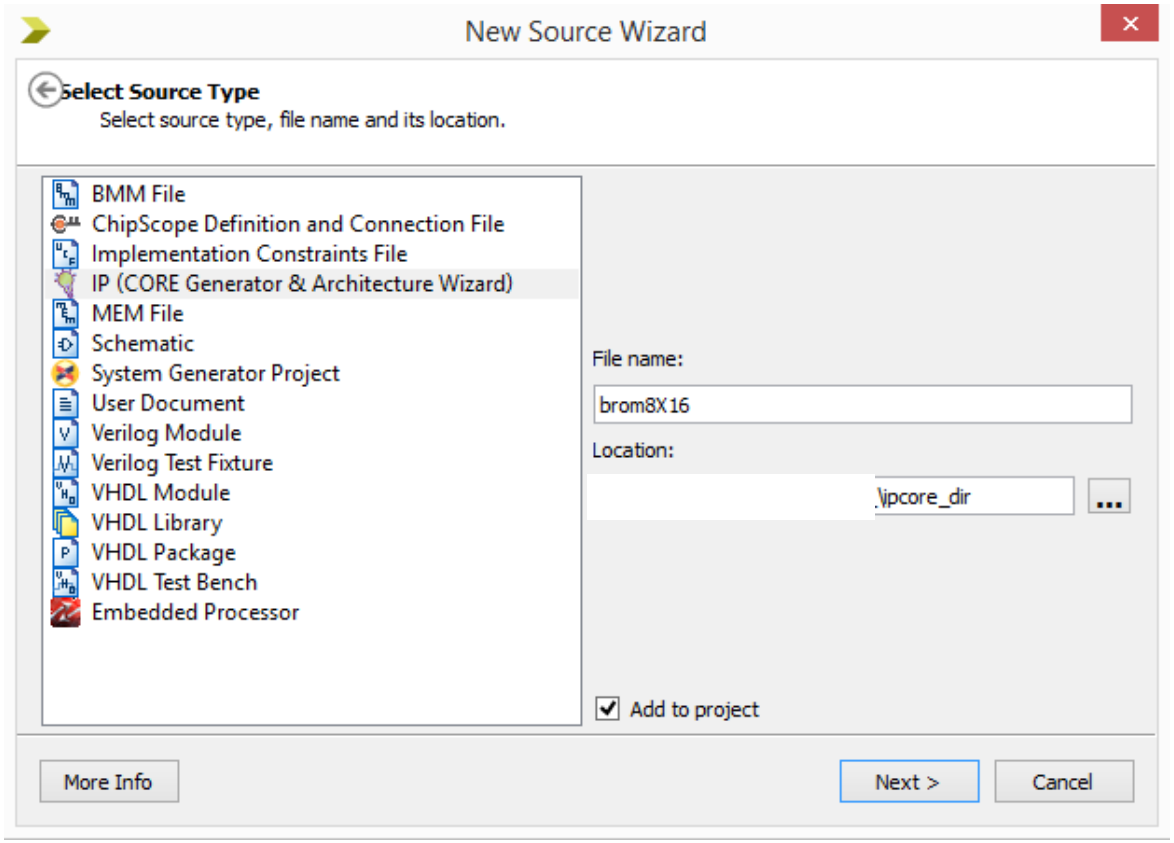
Fișierul de inițializare a memoriei

- Fișier de inițializare a memoriei (MIF) se poate crea folosind un editor de texte (de exemplu Notepad++)
- Creați fișierul cu conținutul prezentat mai jos:

```
; Example Initialization file for a 8x16 block ROM
memory_initialization_radix = 16;
memory_initialization_vector =
0000 1111 2222 3333
4444 5555 6666 7777;
```

- Salvați fișierul cu numele, Lab12_3_init.coe.
- Verificați să aibă extensia .coe!

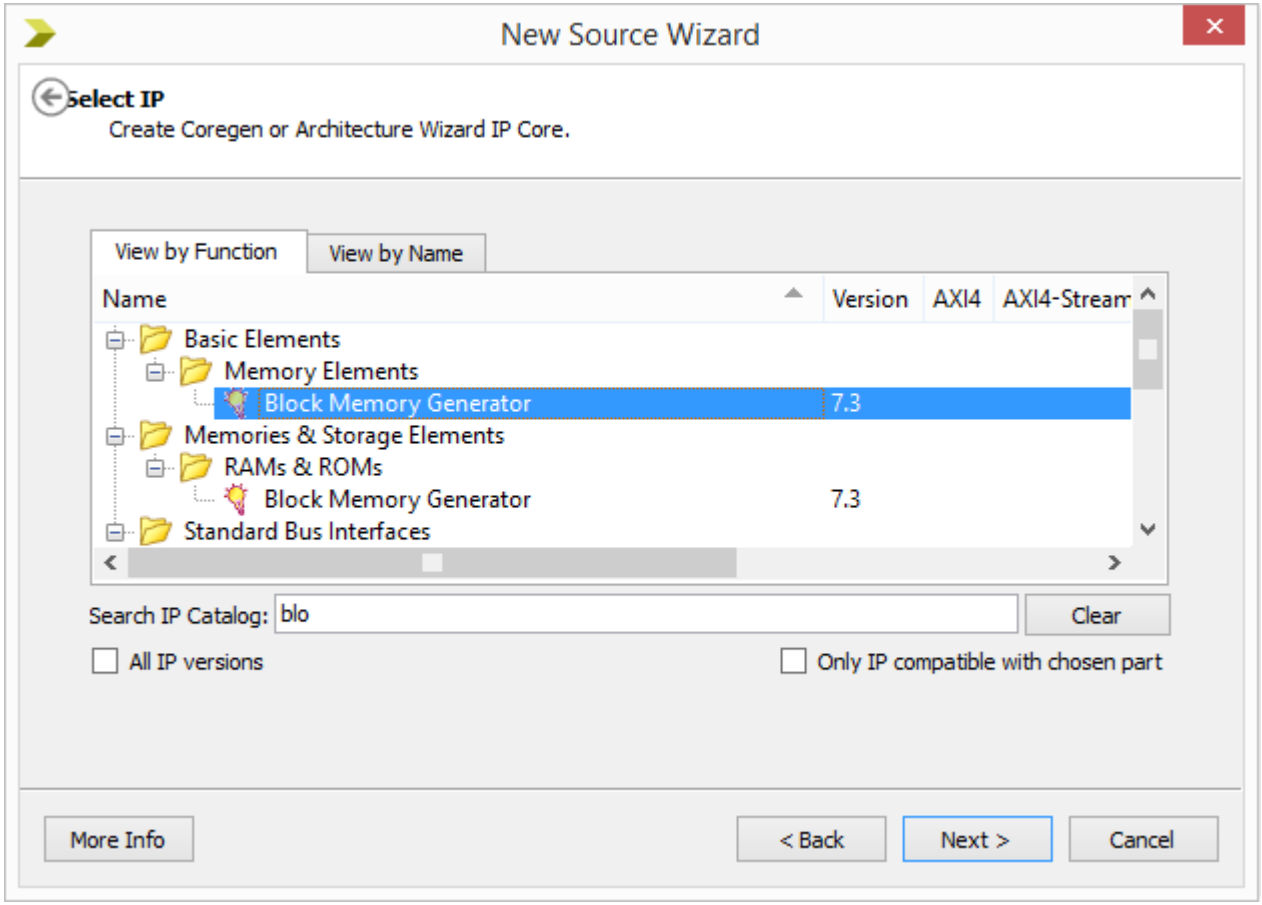
Core Generator



1. Adăugați o sursă nouă de tip IP Core
2. Introduceți numele ***brom8x16*** apoi apăsați Next.

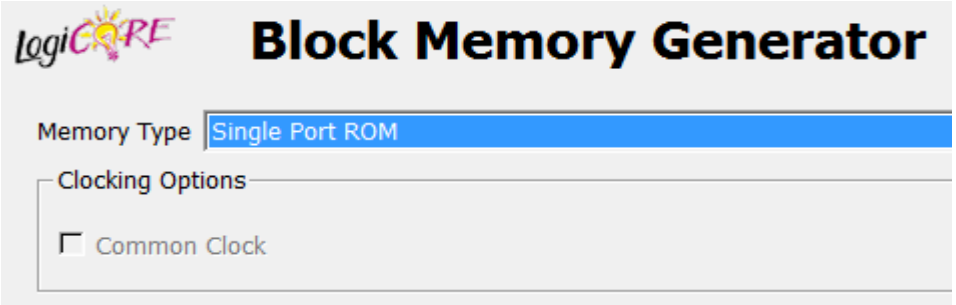
Crearea memoriei Block RAM/ROM

Începeți să tastați „blo” pentru a căuta în catalogul de blocuri IP și alegeți *Block Memory Generator*.

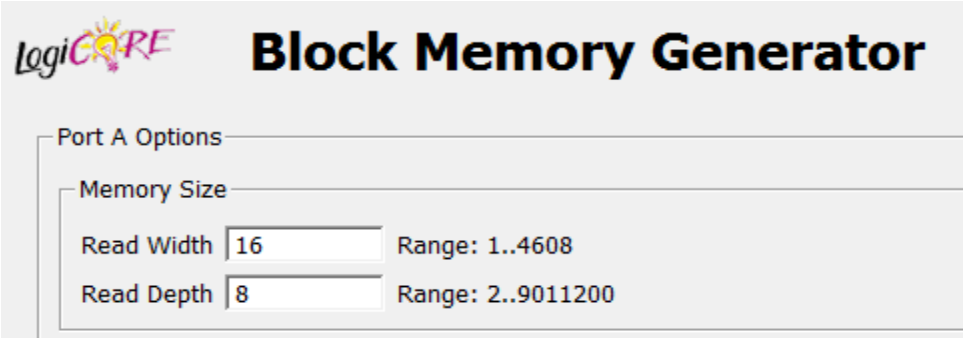


Setarea parametrilor memoriei

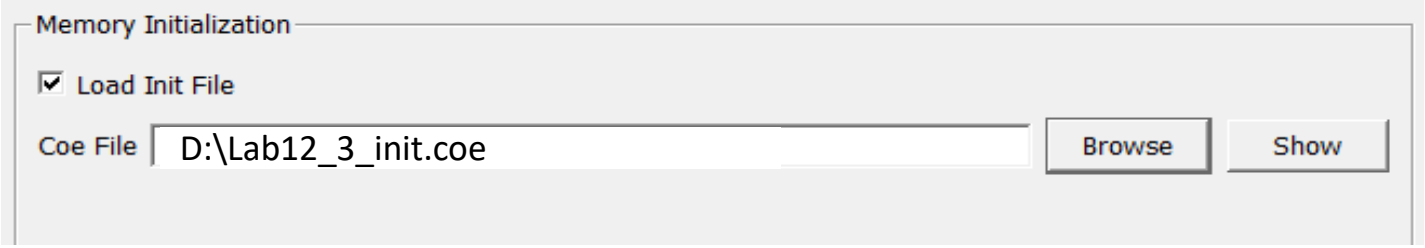
- Pe pagina a doua alegeți „Single Port ROM”



- Pe pagina 3 setați capacitatea memoriei 16x8 biți



- Pe pagina 4 indicați locația fișierului de inițializare cu extensia .coe



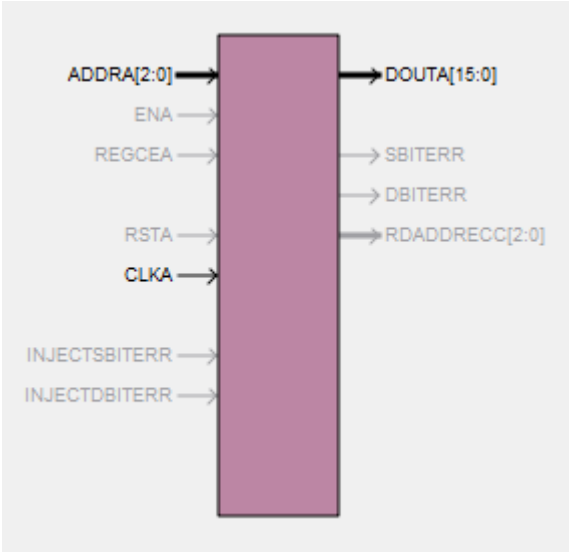
Generarea și instanțiera memoriei

Apăsați Generate și așteptați până modulul va fi generat.

Instanțiera memoriei în top modul.

```
wire [15:0] x;
wire [2:0] addr;
wire clk190;

clkdiv U1 (.clk(mclk),
    .clr (clrbtn),
    .clk190(clk190)
) ;

hex7seg X2 (.x(x),
.cclk(clk190),
.clr(clrbtn),
.a_to_g(seg),
.an(an),
.dp(dp)
) ;

brom8X16 U5 (
    .addra(btn), // Bus [2 : 0]
    .clka(mclk),
    .douta(x)// Bus [15 : 0]
);
endmodule
```

Porturile modulului de memorie



Modulul de divizare de clock

- Adăugați modul de divizare a semnalului de clock

```
20  //////////////////////////////////////////
21  module clkdiv (
22        input wire clk ,
23        input wire clr ,
24        output wire clk190 ,
25        output wire clk25 ,
26        output wire clk3
27     ) ;
28
29     reg [24:0] q;
30     // 25-bit counter
31     always @(posedge clk or posedge clr)
32     begin
33        if(clr == 1)
34           q <= 0;
35        else
36           q <= q + 1;
37     end
38     assign clk190 = q[17]; // 190 Hz
39     assign clk25 = q[0]; // 25 MHz
40     assign clk3 = q[23]; // 3 Hz
41  endmodule
```

Modulul Hex7seg (1)

- Adăugați modul hex7seg

```
//////////////////////////////////////////////////////////////////////////////////
module hex7seg (
input wire [15:0] x ,
input wire cclk ,
input wire clr ,
output reg [6:0] a_to_g ,
output reg [7:0] an ,
output wire dp
);
reg [1:0] s;
reg [3:0] digit;
wire [3:0] aen;
assign dp = 1;
// set aen[3:0] for leading blanks
assign aen[3] = x[15] | x[14] | x[13] | x[12];
assign aen[2] = x[15] | x[14] | x[13] | x[12] | x[11] | x[10] | x[9] | x[8] ;
assign aen[1] = x[15] | x[14] | x[13] | x[12] | x[11] | x[10] | x[9] | x[8] | x[7] | x[6] | x[5] | x[4] ;
assign aen[0] = 1; // digit 0 always on

// Quad 4-to-1 MUX: mux44
always @(*)
case(s)
   0: digit = x [3:0] ;
   1: digit = x [7:4] ;
   2: digit = x [11:8];
   3: digit = x [15:12] ;
   default: digit = x[3:0];
endcase
// 7-segment decoder: hex7seg
```

Modulul Hex7seg (2)

```
// 7-segment decoder: hex7seg
always @(*)
case(digit)
   0: a_to_g = 7'b1000000;
   1: a_to_g = 7'b1111001;
   2: a_to_g = 7'b0100100;
   3: a_to_g = 7'b0110000;
   4: a_to_g = 7'b0011001;
   5: a_to_g = 7'b0010010;
   6: a_to_g = 7'b0000010;
   7: a_to_g = 7'b1111000;
   8: a_to_g = 7'b0000000;
   9: a_to_g = 7'b0010000;
   'hA: a_to_g = 7'b0001000;
   'hb: a_to_g = 7'b1100000;
   'hC: a_to_g = 7'b0110001;
   'hd: a_to_g = 7'b1000010;
   'hE: a_to_g = 7'b0110000;
   'hF: a_to_g = 7'b0111000;
   default: a_to_g = 7'b0000001; // 0
endcase
// Digit select
always @(cclk)
   begin
      an = 4'b1111;
      if(aen[s] == 1)
         an[s] = 0;
   end
```

```
77  // 2-bit counter
78
79  always @(posedge cclk or posedge clr)
80     begin
81        if(clr == 1)
82        s <= 0;
83        else
84        s <= s + 1 ;
85     end
86
87  endmodule
```

Implementare și testare

Implementare

- Adăugați fișierul de constrângeri .ucf
- Implementați, încărcați în placă și testați
- Generați adresele folosind comutatoarele.
- Verificați pe afișajul cu 7 segmente conținutul de la adresa specificată

Bibliografie

[1] Introduction to Digital Design Using Digilent FPGA Boards — Block Diagram / Verilog Examples, LBE Books, Rochester, MI 2009, ISBN 978-0-9801337-9-0, Online Version

[2] Richard E. Haskell, Darrin M. Hanna: Advanced Digital Design, LBE Books, Rochester, MI 2009, ISBN 978-0-9801337-5-2

[3] Lung, C., Oniga, S., Joian, R., Gavrincea, C., Circuite integrate digitale - Îndrumător de laborator, Editura Universitarii de Nord, Baia Mare, 2008, ISBN 978-973-1729-86-2

[4] Dan Nicula, Verilog. Carte de învățătură, Editura Universității Transilvania Brașov, 2019 ISBN 978-606-19-1162-2

[5] Xilinx, HDL Coding Techniques. XST User Guide, UG627 (v 11.3) September 16, 2009

Resurse de învățare online

[1] Oniga, S., Pagina web a disciplinei de Circuite integrate digitale I / Proiectare logică: http://ece.ubm.ro/ea/cursuri/CID/

[2] Verilog Tutorial: http://www.asic-world.com/verilog/veritut.html

Anexe

- Șablon Raport de laborator
- Lista de verificare
- Coduri sursă

Șablon Raport de laborator

UTCN - CUNBM

RAPORT DE LABORATOR **Circuite digitale**

Laborator 1

Nume student:	Specializare/an/grupa:
Cadru didactic:	Rezultat:

Lab1 exercițiul 1 :

- Numele exercițiului .....
- Exercițiul a fost efectuat, a funcționat? ***Da*** ***Nu***

  Dacă nu, de ce? .....
- Rezultate: tabele, simulări (capturi de ecran), etc.

  .....
- Concluzii (comentariile voastre)

  .....

Lab1 exercițiul 2:

- .....

Lista de verificare (1)

UTCN - CUNBM

RAPORT DE LABORATOR **Proiectare logică / Circuite integrate digitale**

Lista de verificare laboratoare/exerciții efectuate

Nume student:	Specializare/an/grupa:
Cadru didactic:	Rezultat:

	Laborator 1	Obligatoriu / Facultativ	Raport laborator	Data	Rezultat
	Prezentare și testare plăci de dezvoltare				
	Laborator 2				
Lab2_1	Implementarea porților logice - schematic	O			
Lab2_2	Implementarea porților logice în limbaj Verilog	O			
Lab2_3	Implementare circuit logic 1 - schematic	F			
Lab2_3	Implementare circuit logic 1 - HDL	F			
Lab2_4	Implementare circuit logic2 - HDL	F			
Lab2_5	Implementare circuit logic 3 - HDL	F			
	Laborator 3				
Lab3_1a	Implementarea funcțiilor logice de 3 variabile - Asociativitate	O			
Lab3_1b	Implementarea funcțiilor logice de 3 variabile - Distributivitate	F			
Lab3_1c	Implementarea funcțiilor logice de 3 variabile – Regulile de absorbție	F			
Lab3_2a	Teorema lui De Morgan pentru funcții de 2 variabile	O			
Lab3_2b	Teorema lui De Morgan pentru funcții de 3 variabile	O			
Lab3_2c	Generalizarea teoremei lui De Morgan	F			
Lab3_3a	Simularea și implementarea funcțiilor AND, OR, XOR și NOR de 4 variabile	O2			

Lista de verificare (2)

	Laborator 4				
Lab4_1	Implementarea funcțiilor logice pe două nivele – funcția XOR	O			
Lab4_2	Decodificator BCD – 7 segmente, implementarea funcției pt. segmentul „a”	O			
Lab4_3	Afișajul cu 7 segmente	O			
	Laborator 5				
Lab5_1a	Codificator zecimal - BCD – descriere structurală	O			
Lab5_1b	Codificator 8:3descriere structurală. Simulare și implementaare.	F2			
Lab5_1c	Codificator prioritar – descriere comportamentală folosind funcția IF	O			
Lab5_2a	Decodificator binar pe 3 biți - descriere structurală	F			
Lab5_2b	Decodificator binar pe 3 biți - descriere comportamentală	O			
Lab5_3a	Multiplexor 2:1 - descriere comportamentală	O			
Lab5_3b	Multiplexor 4:1 - descriere structurală	F			
Lab5_3c	Multiplexor 4:1 - descriere comportamentală	O			
Lab5_3d	Multiplexor generic - descriere comportamentală	F			
	Laborator 6				
Lab6_1a	Comparator pentru numere pe 2 biți	O			
Lab6_2	Generator de paritate	F2			
Lab6_3a	Semisumator pe 1-bit - simulare	O			
Lab6_3b	Sumator complet pe 1-bit (folosind 2 semi sumatoare) - implementare	O			
Lab6_4	Sumator complet pe 1-bit - descriere comportamentală	O			
Lab6_5a	Sumator pe 4-biți - descriere structurală	F			
Lab6_5b	Sumator pe 4-biți - descriere comportamentală	O			
	Laborator 7				
Lab7_1a	Circuit de scădere pe 4-biți	F			
Lab7_2	ALU pe 1 bit	F			
Lab7_3	ALU pe 4 biți	O			
Lab7_3b	ALU pe 4 biți – simulare	F			
Lab7_4	ALU pe 4 biți cu afișare pe display cu 7 segmente	O			

Lista de verificare (3)

	Laborator 8				
Lab8_1a	Latch de tip D	O			
Lab8_2	Flip-flop de tip D (2 din 4, a-d)	O			
Lab8_3	Flip-flop de tip T cu reset asincron	O			
Lab8_3b	Flip-flop de tip T cu reset sincron	O			
	Laborator 9				
Lab9_1a	Numărător direct sincron pe 4-biți cu ștergere asincronă	O			
Lab9_1b	Numărător direct sincron pe 4-biți cu ștergere sincronă	F			
Lab9_2	Numărător invers sincron pe 4-biți cu ștergere sincronă	F			
Lab9_3	Numărător reversibil sincron pe 4-biți	F			
Lab9_4	Numărător zecimal direct cu încărcare	O			
Lab9_5	Numărător direct sincron pe N-biți cu ștergere asincronă	O			
	Laborator 10				
Lab10_1	Divizor de clock	O			
Lab10_2	Numărător pe 8-biți cu ieșire pe leduri	O			
Lab10_3	Numărător pe 8-biți cu afișaj cu 7 segmente	O			
Lab10_4	Numărător decadic	F			
	Laborator 11				
Lab11_1	Registru paralel-paralel pe 4 biți				
Lab11_2	Registru de deplasare pe 8 biți				
Lab11_3	Registru paralel-serie pe 8 biți				
Lab11_4	Registru serie-paralel pe 8 biți				
Lab11_5	Numărător în inel				
Lab11_6	Numărător Johnson				
	Laborator 12				
Lab12_1	Memorie ROM simpla în cod Verilog				
Lab12_2	Memorie RAM/ROM distribuită – simulare și implementare				
Lab12_3	Block RAM/ROM				