”Unii trăiesc fără nici o țintă în viață; trec prin lume ca nişte fire de paie pe faţa unui râu; nu merg ei, ci îi duce curentul.”

Lucius Annaeus Seneca

Arii Logice Programabile

(GAL-uri)

Laboratoarele 1 & 2

Structuri Logice Programabile (GAL-uri) – Lab 1 & 2

1. Obiectivele laboratorului

Scopul acestui laborator este de a prezenta teoretic, în prima parte, arhitectura unei structuri logice programabile cu referire directă la circuitul produs de firma americană LATTICE GAL16V8 care ulterior va fi utilizat în cadrul laboratorului. În partea a doua se vor realiza două circuite un multiplexor şi un decodificator 7 segmente. Cu aceste ocazii se va trece prin toate etapele necesare dezvoltării şi implementării circuitului: **1.** de la dezvoltarea teoretică, conceptuală, **2.** descrierea funcționării software a circuitului în limbajul ABEL şi **3.** simularea software a lui, până la **4.** obținerea fişierului de ardere a acestuia (JEDEC-file), **5.** programarea circuitului şi **6.** testarea funcționării acestuia pe placa de test.

2. Introducere

Structurile Logice Programabile (PLD, **P**rogramable **L**ogic **D**evice) au apărut la începutul anilor ’70 şi cunosc în prezent o deosebită amploare. Firma Xilinx, de exemplu, a vândut în ultimul deceniu peste zece milioane de circuite FPGA (spre comparare, aproximativ în acelaşi interval, s-au vândut circa treizeci de milioane de microcontrolere din familia 8051, provenind de la firma Intel, în principal, dar şi de la sursele secundare Philips, Siemens, Matshushita). Față de implementarea clasică cu circuite TTL, PLD-urile oferă o serie de avantaje:

- cost redus şi fiabilitate ridicată;
- flexibilitate mare a structurii;
- densitate mare de integrare;
- durata ciclului proiectare-producție se reduce de la câteva luni la câteva săptămâni

3. Arhitectura

Circuitul GAL 16V8 este o structură logică programabilă realizat în tehnologie UltraMOS putând fi atât scris cât şi şters electric şi putând lucra până la frecvențe maxime de 250 MHz. În **Figura 1.** Se remarcă arhitectura matricii de porți ŞI programabile cu 64 de linii şi 32 de coloane. Opt macrocelule OLMC (*Output Logic Macrocell*) cu o arhitectură complexă sunt disponibile pentru cei 8 pini de ieşire. Fiecare macrocelulă este configurabilă de utilizator (intrare, ieşire, inclusiv reacția pentru aria logică de intrare). Astfel, celulele OLMC pot fi configurate în trei topologii distincte (detalii despre aceste moduri de lucru vor fi prezentate în paginile următoare):

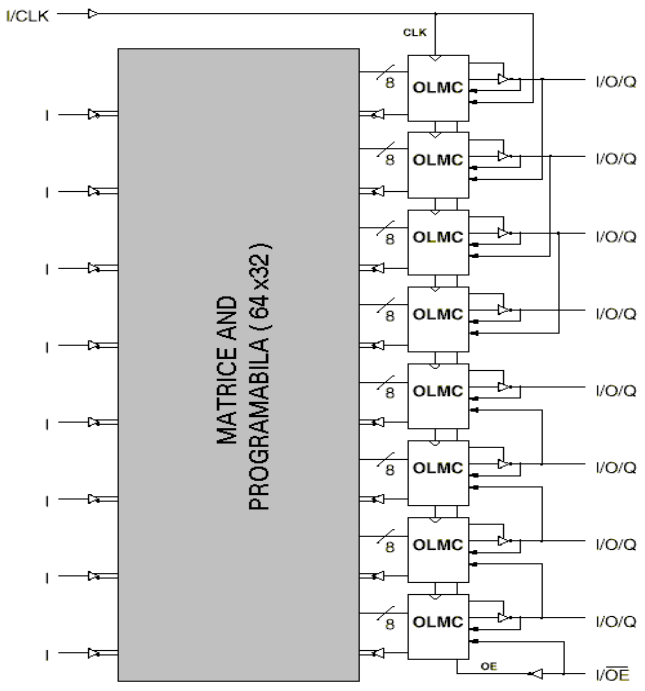


Figura 1. Arhitectura internă a circuitului GAL 16V8

- modul *simplu* de lucru
- modul *complex* de lucru
- modul de tip *registru*

Alegerea configurației dorite este dată de doi biți globali SYN şi AC0 care se găsesc într-un registu intern circuitului şi care controlează modul de lucru pentru toate macrocelulele.

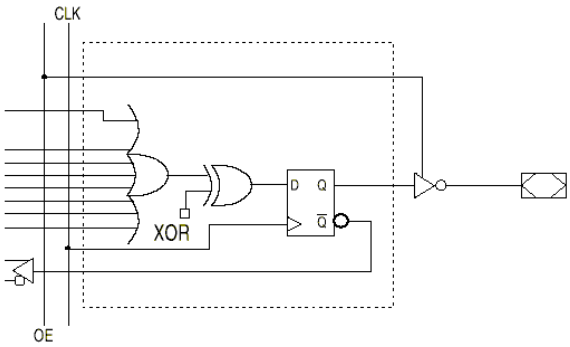
Totodată există şi un bit XOR care este specific fiecărei celule în parte şi cu ajutorul căruia se controlează polaritatea ieşirii pentru toate cele trei moduri prezentate, în timp ce bitul AC1 controlează configuraţia de intrare/ieşire a celulelor. Harta memoriei interne şi poziţionarea acestor biţi se arată în **ANEXA 1**.

Arhitecturi PAL emulate de GAL 16V8	Configuraţia celulei OLMC pentru GAL 16V8
16R8	Registru
16R6	Registru
16R4	Registru
16RP8	Registru
16RP6	Registru
16RP4	Registru
16L8	Complexă
16H8	Complexă
16P8	Complexă
10L8	Simplă
12L6	Simplă
14L4	Simplă
16L2	Simplă
10H8	Simplă
12H6	Simplă
14H4	Simplă
16H2	Simplă
10P8	Simplă
12P6	Simplă
14P4	Simplă
16P2	Simplă

Figura 2. Echivalenţe cu alte circuite

Cu aceasta arhitectură evoluată a macrocelulii de ieşire circuitul GAL 16V8 poate emula la aceeaşi viteză şi la un consum substanţial redus de putere (există două tipuri de dispozitive care se împart după consumul de la sursă: 75 mA – Low Power, 45 mA – Quarter Power) majoritatea circuitelor PAL bipolare cu 20 de pini – cu excepţia seriei X realizată cu porţi SAU-EXCLUSIV.

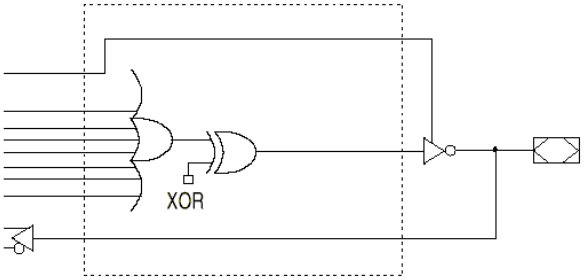
În ***modul de lucru registru*** toate macrocelulele sunt configurate ca ieşiri dedicate de tip registru (**Figura 3**) sau ca pini de intrare / ieşire (**Figura 4**) combinaţionali. Toate celulele au acelaşi semnal de tact şi acelaşi pin de validare a ieşirii. În cazul alegerii configuraţiei de tip registru de ieşire se observă că avem opt termeni de tip produs care pot fi selectaţi la intrare registrului şi şapte termeni produs care pot fi selectaţi în cazul configuraţiei combinaţionale intrare / ieşire. Pinii 1 şi 11 vor fi configuraţi tot



Registered Configuration for Registered Mode

- SYN=0.
- AC0=1.
- XOR=0 defines Active Low Output.
- XOR=1 defines Active High Output.
- AC1=0 defines this output configuration.
- Pin 1 controls common CLK for the registered outputs.
- Pin 11 controls common $\overline{OE}$ for the registered outputs.
- Pin 1 & Pin 11 are permanently configured as CLK & $\overline{OE}$.

Figura 3. Configuraţia de tip registru a macrocelulei în modul *registru* de lucru



Combinatorial Configuration for Registered Mode

- SYN=0.
- AC0=1.
- XOR=0 defines Active Low Output.
- XOR=1 defines Active High Output.
- AC1=1 defines this output configuration.
- Pin 1 & Pin 11 are permanently configured as CLK & $\overline{OE}$.

Figura 4. Configuraţia combinaţională a macrocelulei în modul *registru* de lucru.

timpul, în acest mod de lucru, ca intrări de tact (CLK) şi de validare a ieşirii ($\overline{OE}$) aceşti pini neputând fi configuraţi ca pini de intrare dedicaţi. Adresa fiecărei fuzibile din arie cât şi a zonelor User Electronic Signature (UES) cât şi Product Term Disable (PDT) sunt prezentate în **Figura 5**.

În *modul* ***complex de lucru***, macrocelulele pot fi configurate numai ca ieşiri combinaţionale simple cu posibilitate de trecere a lor în înaltă impedanţă (**Figura 7**) sau ca ieşiri combinaţionale cu posibilitate de reacţie a ieşirii în matricea ŞI aşa cum se observă în **Figura 6**. Ca o restricţie existentă trebuie observat că dintre cele 8 OLMC numai 6 putând fi conectate în acest mod după cum se poate vedea din **Figura 11**. Pinii 1 şi 11 în acest mod devin pini de intrare şi se folosesc de căile de întoarcere în structură a pinilor 19 şi 12 astfel macrocelulele corespunzătoare acestor pini nu mai prezintă cale de reacţie aşa cum s-a prezentat şi mai sus. În cazul în care aplicaţia ne impune să avem

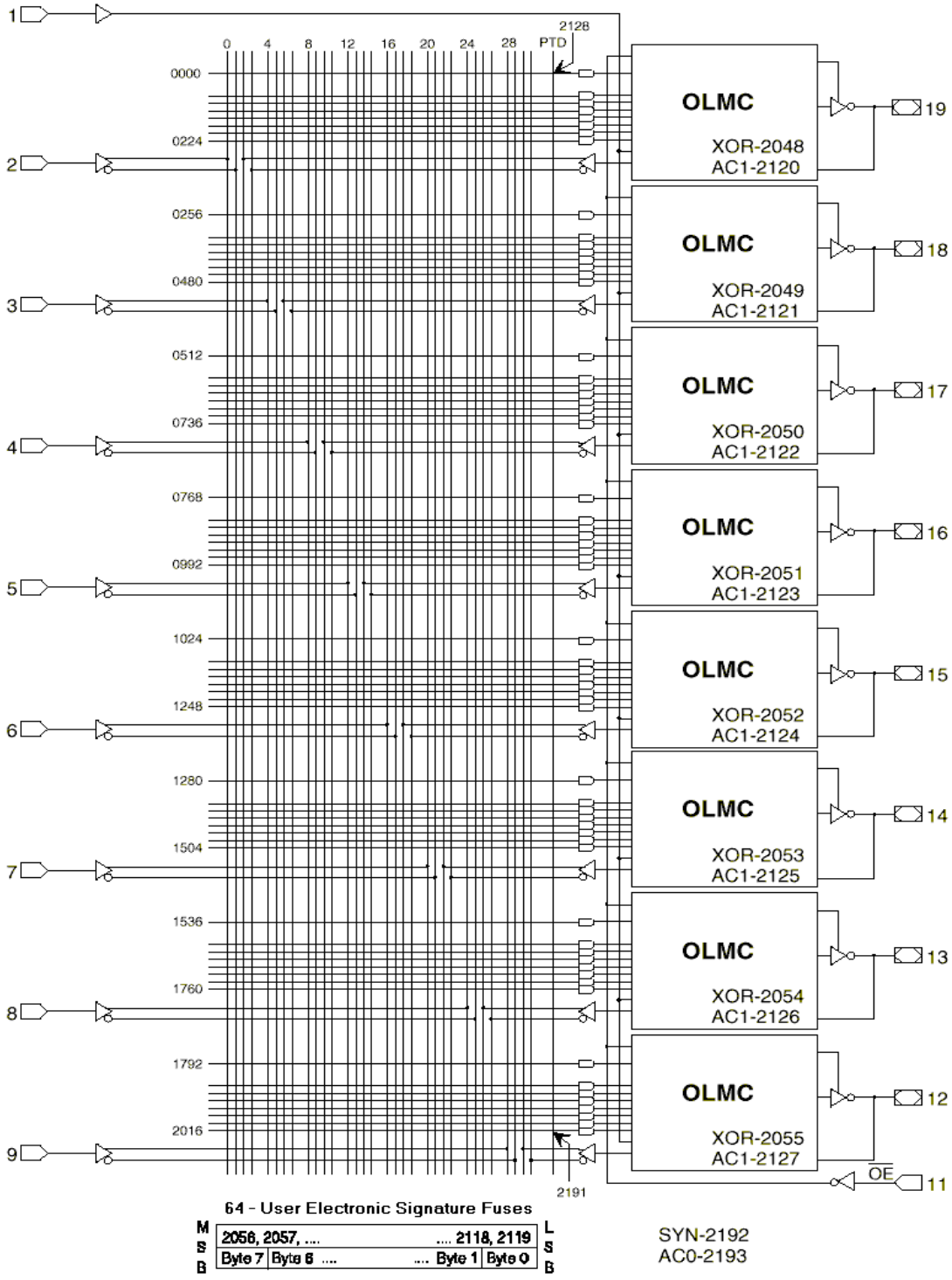


Figura 5. Topologia internă a circuitului în modul *registru* de lucru

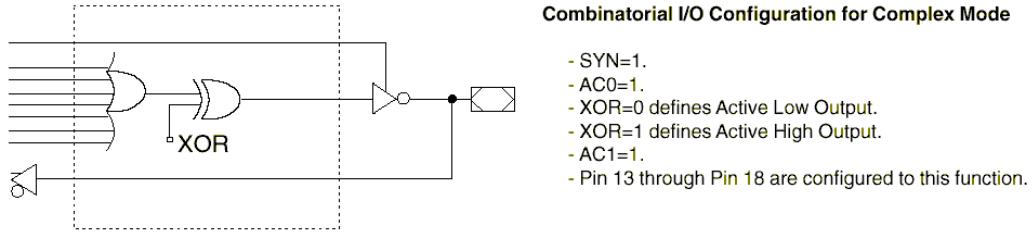


Figura 6. Configurația de tip Intrare / Ieşire combinațională în modul *complex* de lucru

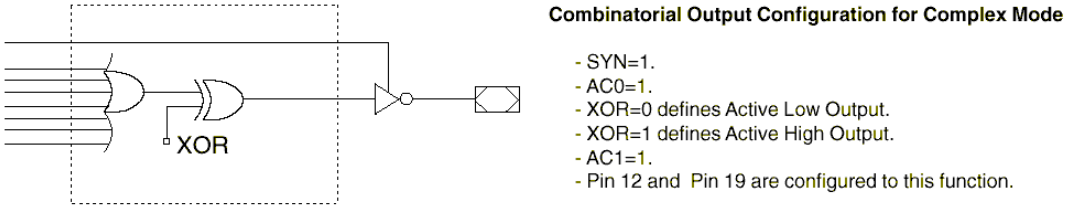


Figura 7. Configurația de tip ieşire combinațională în modul *complex*

cale de reacție pentru toate macrocelulele suntem obligați să folosim modul de lucru de tip registru. Ieşirea combinațională poate depinde de 7 termeni de tip produs (AND) deoarece un termen de tip produs este folosit pe post de trecere a ieşirii în înaltă impedanță.

În modul ***simplu de lucru*** macrocelulele pot fi configurate ca intrari (**Figura 10**) sau ieşiri cu (**Figura 8**) sau fără (**Figura 9**) cale de reacție în structură. Unul dintre avantajele acestui mod de lucru este dat de faptul prezenței a opt termeni de tip produs. Pinii 1 şi 11 sunt disponibili ca pini de intrare

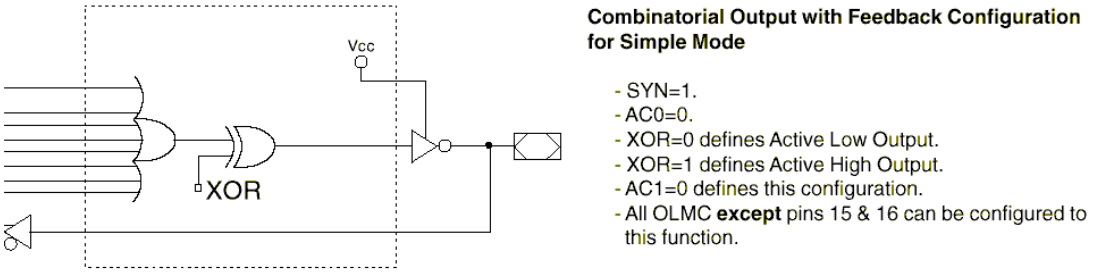


Figura 8. Configurație a macrocelulei în modul *simplu* în care se observă prezența reacției în structură

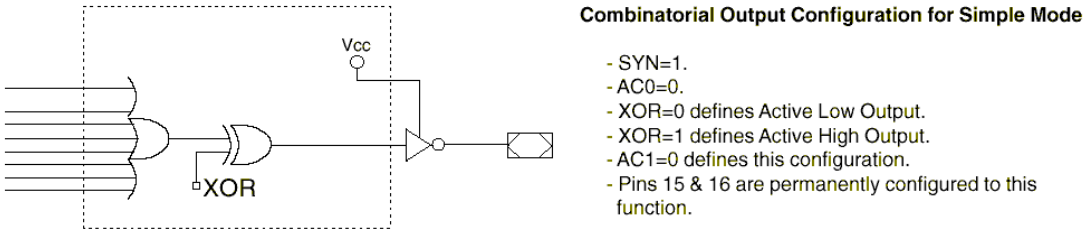


Figura 9. Configurație de tip ieşire combinațională în modul *simplu* de lucru

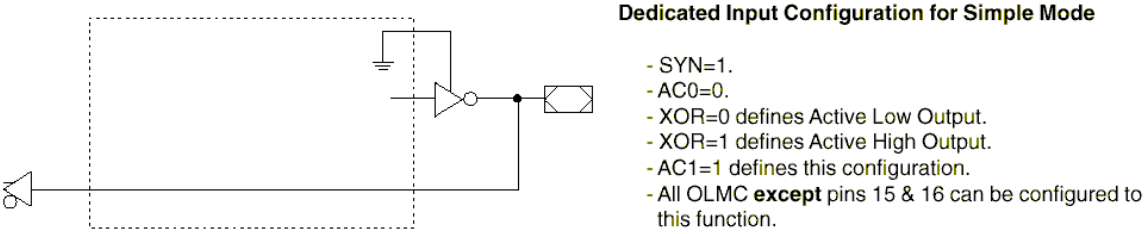


Figura 10. Configurarea unei macrocelule ca intrare dedicată în modul *simplu* de lucru.

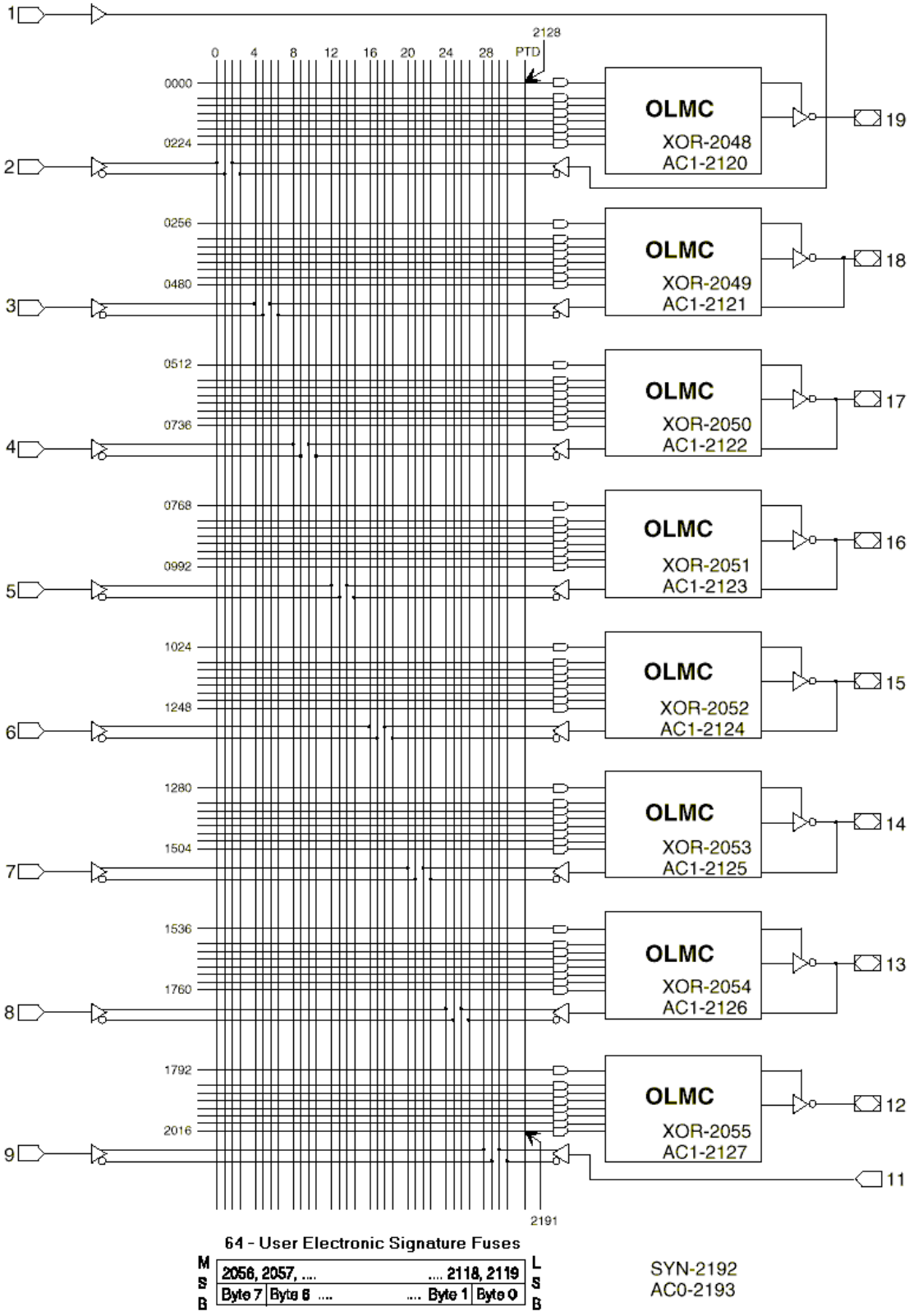


Figura 11. Arhitectura globală a întregului circuit în modul *complex* de lucru

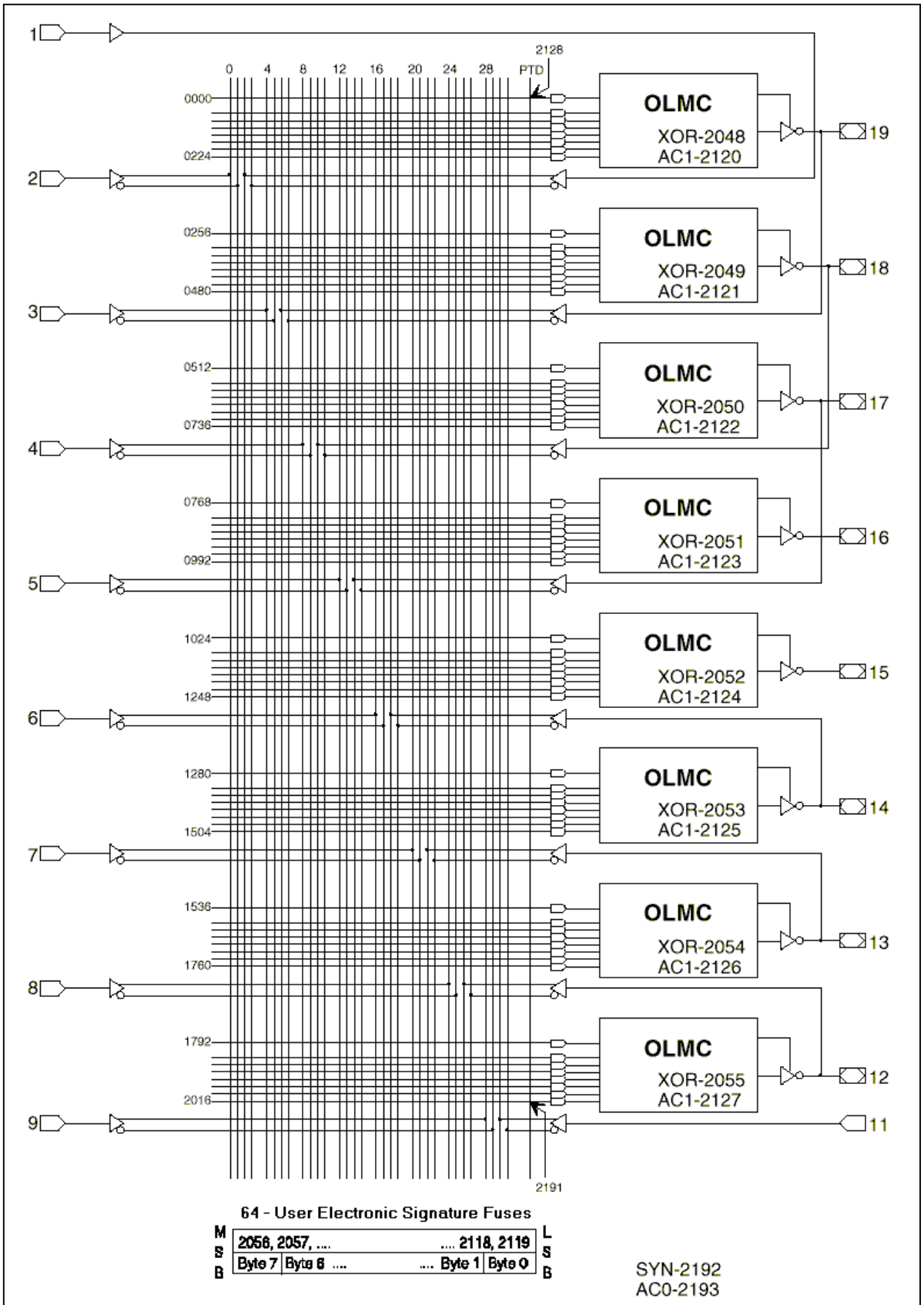


Figura 12. Arhitectura circuitului în modul *simplu* de lucru.

4. Exemple

4.1. Un simplu exemplu

Temă: Să se implementeze într-un GAL 16V8 funcţia SAU-EXLUSIV (XOR).

În continuare vom urmări paşii ce trebuie executaţi pentru atingerea acestui obiectiv. Scopul final este acela de a obţine un fişier JEDEC cu ajutorul căruia programatorul să fie capabil să scrie informaţia ce va configura şi descrie funcţionarea porţii XOR într-un circuit de tipul GAL16V8.

Prima etapă consta în scrierea, redactarea textului sursă prin care se descrie: funcţionarea circuitului (cu ajutorul ecuaţiilor de funcţionare sau a altor artificii de limbaj care au ca scop să ne uşureze munca), echivalenţa între pinii circuitului şi variabilele de intrare sau ieşiri, precizarea tipului de circuit folosit, introducerea unor vectori de test folosiţi în testarea *software* a rezultatului etc.

În cadrul acestui laborator ne vom folosi de limbajul ABEL pentru a descrie funcţionarea circuitului. În urma compilării acestui fişier vor rezulta mai multe fişiere cele mai multe dintre ele ţinând de diferite rapoarte în care ne sunt prezentate diferite informaţii, cum ar fi: gradul de utilizare al circuitului, mesaje de eroare legate de nerespectarea sintaxei limbajului, date referitoare la momentul şi durata compilării, numărul de termeni produs rezultaţi în urma minimizării etc. Cel mai important fişier rezultat fiind cel de tip JEDEC care conţine şi rezultatul compilării, acela care va fi utilizat de programator.

În cadrul limbajul ABEL se definesc patru zone distincte:

- Hederul
- Zona declaraţiilor
- Descrierea logică a circuitului
- Zona vectorilor de test (această ultimă zonă poate fi implementată în mod separat într-un fişier specific destinat testării funcţionării circuitului, fişier ce va avea extensia „**abv**” şi va fi inclus în proiectul aplicaţiei)

Prezentarea acestora se face pe codul sursă a XOR-ului:

Hederul

```
MODULE XOR
TITLE 'Test pentru crearea unui xor'        “linie optională
```

Declaraţiile

```
declarations
“declar pinii de intrare
        A, B pin 2, 3;
“declar pinul de ieşire
        Q pin 19 istype 'com';
        x=.X.;

@alternate;
```

Descrierea logica

```
equations
        Q = /A & B + A & /B;
```

Vectorii de test

```
test_vectors 'test'
        ([A, B] -> [Q])
         [0, 0] -> [x];
         [0, 1] -> [x];
         [1, 0] -> [x];
         [1, 1] -> [x];
```

Sfârşit

```
END
```

Din punctul de vedere al aplicaţiei, descrierea funcţionării circuitului se va face într-un fişier ce va purta extensia „**abl**” şi va fi parte integrantă a proiectului.

Fişierul JEDEC al implementării de mai sus generat în urma compilării este:

```
Synario 3.00 Data I/O Corp.
JEDEC file for: P16V8AS V
Created on: Sun Dec 17 02:20:29 2000

Test pentru crearea exemplului cu xor
*
QP20* QF2194* QV4* F0*
 X0*
NOTE Table of pin names and numbers*
NOTE PINS A:2 B:3 Q:19*
L0000 01110111111111111111111111111111*
L0032 10111011111111111111111111111111*
L2128 11111111111111111111111111111111111111111111111111111111111111*
L2192 1*
N test*
V0001 X00XXXXXXNXXXXXXXXXN*
V0002 X01XXXXXXNXXXXXXXXXN*
V0003 X10XXXXXXNXXXXXXXXXN*
V0004 X11XXXXXXNXXXXXXXXXN*
C0FBE*
□802B
```

Implementarea funcției XOR în circuitul GAL 16V8 este prezentată mai jos (încercați să faceți echivalarea între fișierul JEDEC, implementare hard din **Figura 13**:

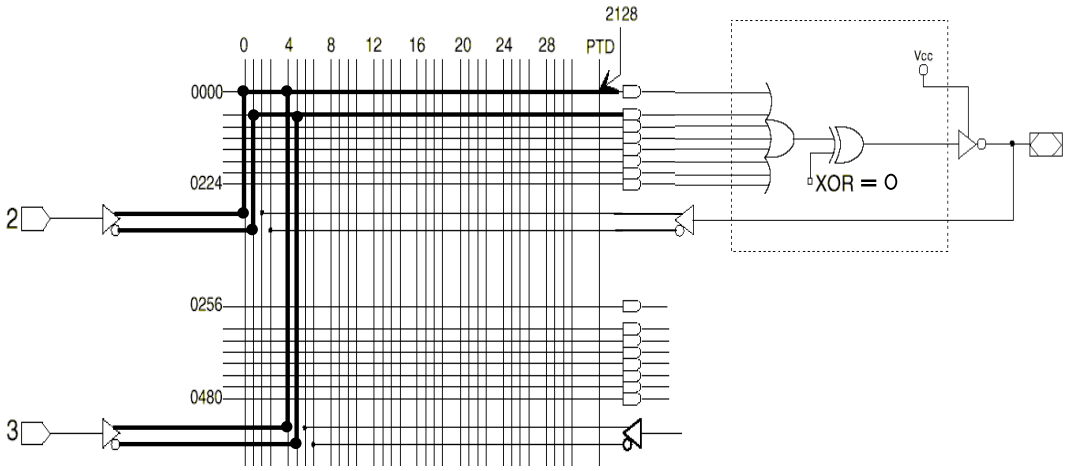


Figura 13. Implementarea hard a circuitului XOR

Exercițiul 1. Explicați aparenta neconcordanță între ecuația scrisă în fișierul sursă şi implementarea hard a aceste-ia.

4.2. Multiplexor

Temă: Să se realizeze şi să se testeze funcționarea unui circui multiplexor realizat în interiorul unei structuri programabile GAL 16V8.

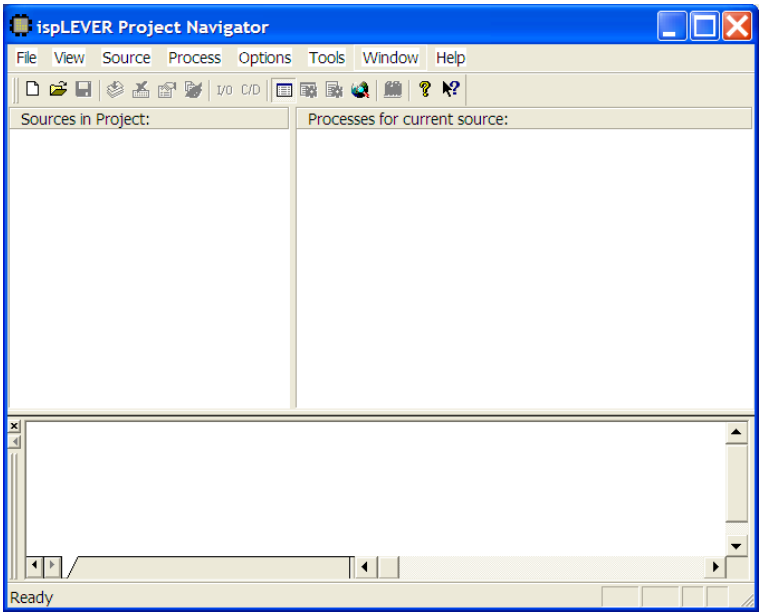
Textul sursă al programului este:

```
MODULE Mux_4To1
TITLE 'Realizare'


declarations
"intrari de control
                    S1, S0 pin 5, 6;
"intrari date
                    A0, A1, A2, A3 pin 1, 2, 3, 4;
"iesire
                    Out pin 19 istype 'com';
x=.X.;

equations
                    Out=(!S1 & !S0 & A0) # (!S1 &  S0 & A1) #
                          ( S1 & !S0 & A2) # ( S1 &  S0 & A3);
test_vectors
                    ([S1, S0, A0, A1, A2, A3]->[Out])
                     [0,  0,  1,  0,  0,  0 ]->[x];
                     [0,  1,  0,  1,  0,  0 ]->[x];
                     [1,  0,  0,  0,  1,  0 ]->[x];
                     [1,  1,  0,  0,  0,  1 ]->[x];
END
```

Exercițiul 2. Scrieți diagrama Veigh-Karnaugh ce descrie funcționarea circuitului şi de unde a rezultat ecuația logică a acestuia.

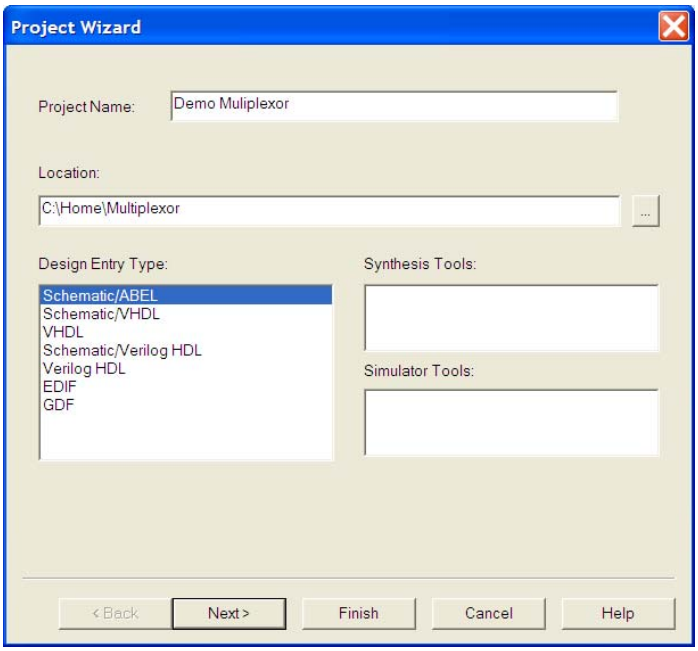
Paşii care trebuie urmați pentru realizarea proiectului sunt:

1. Lansați în execuție mediul de dezvoltare Lattice Semiconductor ispLEVER Classic. Fereastra care se va deschide o dată cu lansarea aplicației este următoarea:

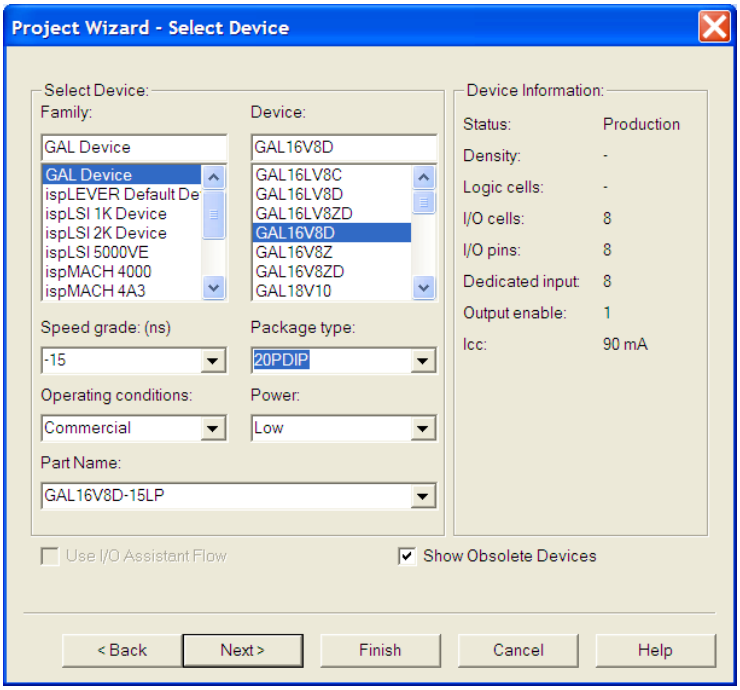


2. Din meniul *„File”* selectați opțiunea *„New Project”*. Automat se va lansa o aplicație de tipul „**Wizard**” care vă va ghida paşii în vederea dezvoltării şi configurării corecte a proiectului acestei aplicații. În fereastra curentă, prima a acestui „**Wizard**”, dați un nume generic proiectului (de exemplu *„Demo_multiplexor”*) şi introduceți calea unde va fi salvat acesta, în cazul nostru

introducem structura de directoare „C:\Home\Multiplexor”. Anterior introducerii acestei structuri creaţi directorul „Multiplexor”. Totodată selectaţi tipul proiectului: Schematic/ABEL (în câmpul „*Design Entry Type*”). La final, apăsaţi butonul „*Next*” pentru a trece la etapa următoare.



3. În această etapă a configurării proiectului trebuie ales tipul circuitului şi introduse date specifice ce descriu acest circuit (tip capsulă, putere consumată, viteza de propagare etc.). Aceste informaţii sunt necesare în special atunci când se obţine fişierul JEDEC şi în pasul anterior, atunci când se simulează circuitul. Realizaţi următoarele selecţii conform figurii de mai jos.

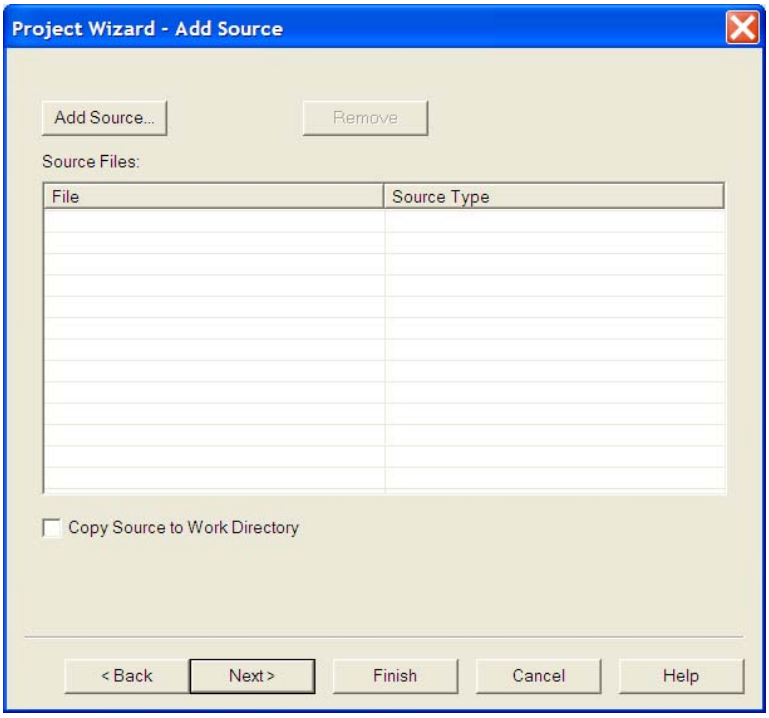


Cea mai importantă setare ţine de alegerea circuitului GAL16V8D şi a capsulei de tip DIP. Chiar dacă circuitele cu care se va lucra în cadrul laboratorului vor fi de tip PALCE16V8, GAL16V8B sau GAL16V8C (circuite produse de alte companii sau predecesoare ale circuitului GAL16V8D)

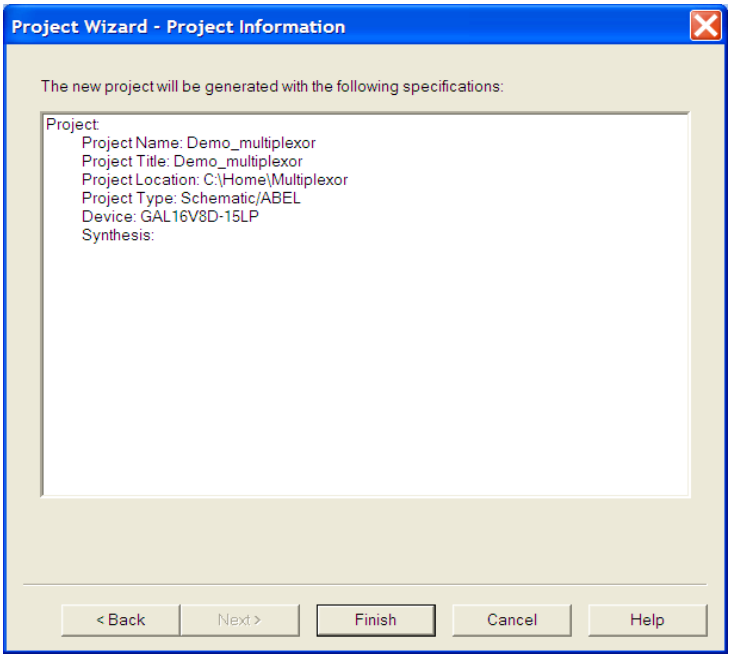
aceast fapt nu va influența în mod negativ realizarea proiectului; toate aceste circuite sunt funcțional identice.

Totodată selectați tipul capsulei (Package type) DIP (*dual in-line packaged*). Deoarece circuitele cu care lucrăm sunt de tip DIP vom putea face o echivalare directă între circuitul dezvoltat şi cel existent în realitate.

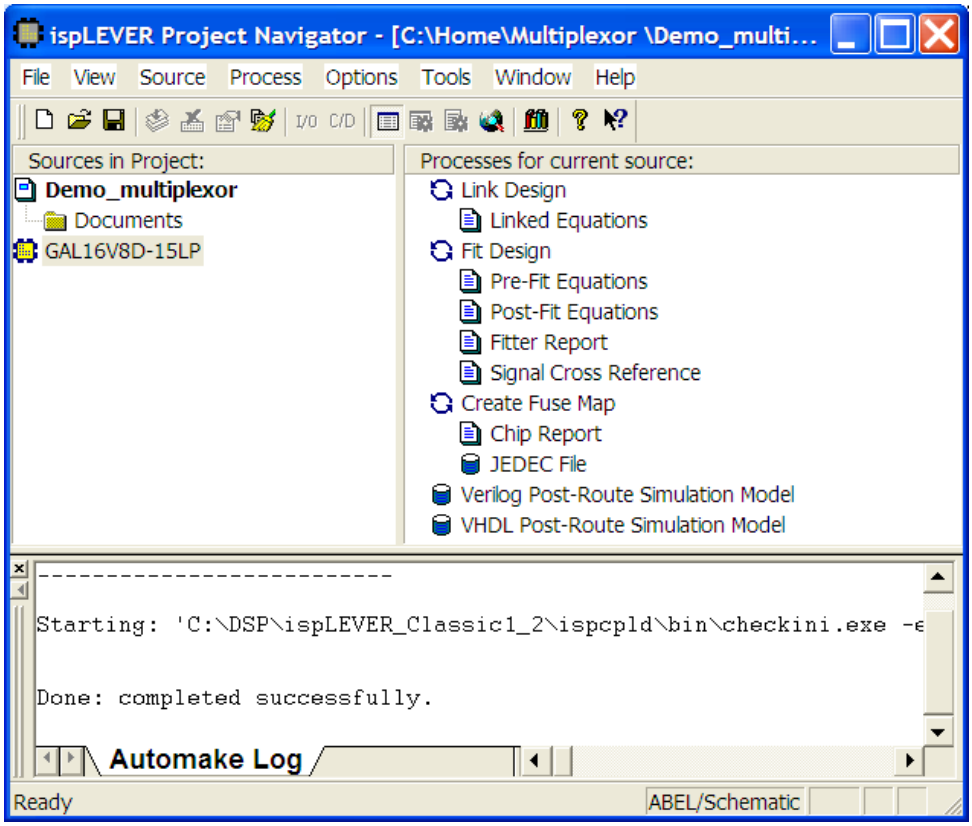
4. Prin apăsarea butonului „*Next*" se va ajunge la penultima fereastră a aplicației „**Project Wizard**" care ne permite să includem în proiect diferite fişiere specifice şi necesare acestuia care au fost anterior dezvoltate sau preluate din alte surse.



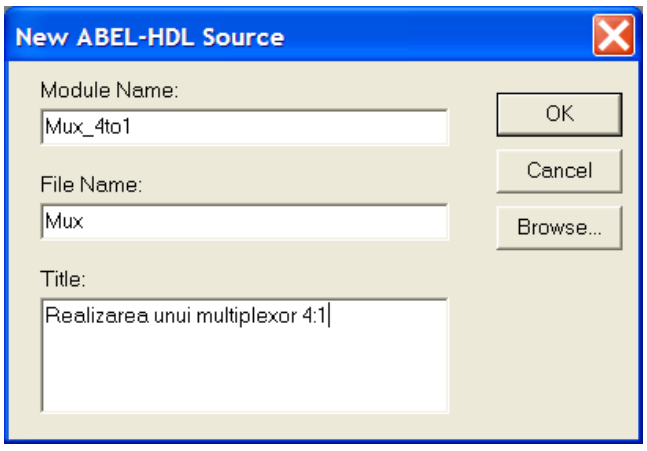
Deoarece în cazul nostru particular nu avem astfel de fişiere mergem mai departe prin apăsarea butonului „*Next*".



5. Ultima fereastră este una în care ne este prezentat un raport asupra tuturor opțiunilor alese anterior, permițându-ne, în cazul în care s-au strecurat greşeli, să revenim asupra deciziilor anterioare prin apăsarea butonului „*Back*”. Dacă nu au intervenit erori apăsăm butonul „*Finish*” – vezi figura de mai sus. În această etapă fereastra principală a aplicației arată ca mai jos;

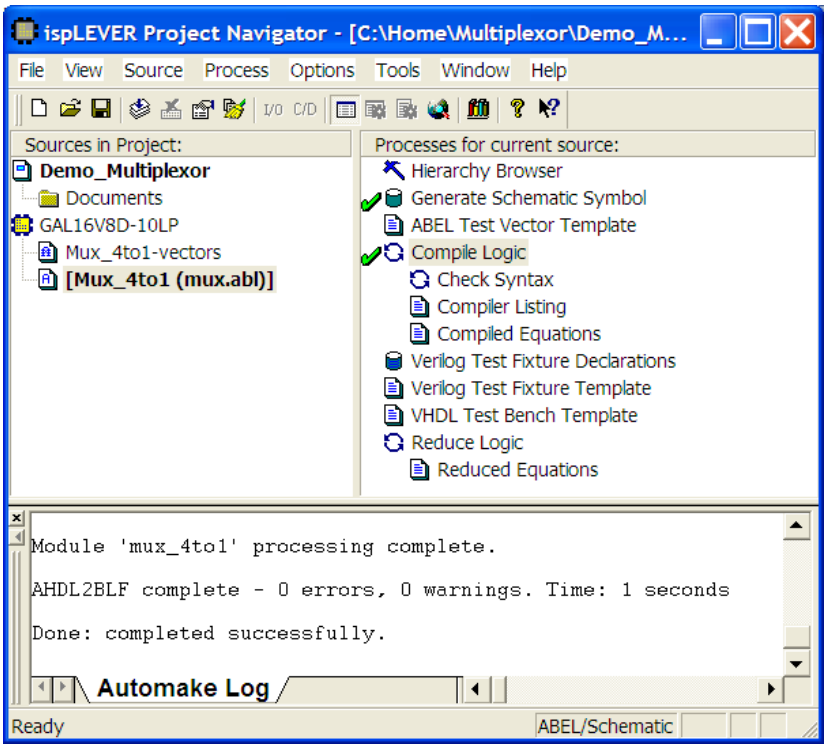


6. În continuare din fereastra principală a aplicației selectați opțiunea „*Source”* urmată de „*New…”*. În fereastra care se deschide se selectează tipul de fișier ce va fi inclus în proiect. În cazul nostru *ABEL-HDL Module*. Fereastra care se afișează va fi una similară cu cea prezentată în figura de mai jos:

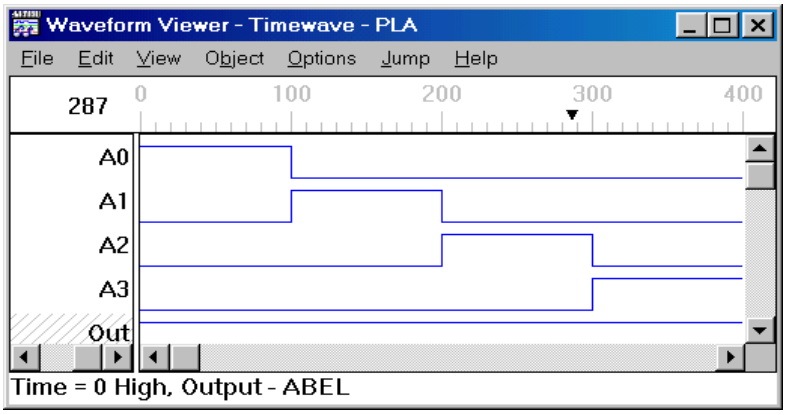


7. În cadrul acestei ferestre de dialog se vor completa rubricile: *Module Name* (numele modulului – în același proiect putând exista mai multe module), *File Name* (numele fișierului sursă ABEL-HDL – numele sub care va fi salvat fișierul pe *hardisk*) și *Title* (opțional).
8. Se va completa listingul fișierului conform sursei prezentate mai sus. La sfârșit se salvează fișierului și se iese din editor.

9. Compilarea aplicației se va realiza prin selectarea opțiunii „*Compile Logic*” urmată de dublu-click pe aceasta. Dacă codul este corect şi compilarea se va termina cu succes se va obține ceva asemănător cu imaginea de mai jos, în caz contrar se vor citi mesajele de eroare şi se vor face corecțiile necesare.



10. Pentru simularea soft se selectează *mux_4to1-vectors* şi se execută „*Simulate JEDEC File*”. Pentru vizualizarea formelor de undă se selectează opțiunea JEDEC Simulation Waveform care lansează aplicația Waveform Viewer.



11. În vederea vizualizării formelor de undă, acestea se aleg din meniul rezultant în urma selecției opțiunilor „*Edit*” urmată de „*Show…*” (în fereastra rezultantă se da dublu-click pe numele fiecărei intrări sau ieşiri ce se doreşte a fi vizualizată).
12. Pentru obținerea fişierului JEDEC se selectează *GAL 16V8* din box-ul „*Sources in Project”* ulterior se dă dublu-click pe „*Create Fuse Map*” (o opțiune mai generală care determină anterior diferite verificări şi teste – de. ex. se verifică dacă ecuațiile logice de descriere a sistemului logic încap în circuitul fizic) sau, mai direct, dând dublu-click pe opțiunea „*JEDEC File*” din box-ul „*Processes for Current Source*” rezultatul obținut va fi un fişier cu numele mux.jed ce se găseşte în directorul proiectului.

13. Următorul pas constă în "arderea" circuitului. Pentru aceasta urmați instrucțiunile prezentate în **ANEXA 3.**

Atenție în momentul în care se cere introducerea circuitului în soclu, pinii inferiori ai circuitului (pinii 10 și 11) se vor poziționa la baza soclului – se urmăresc indicațiile prezentate pe cutia programatorului.

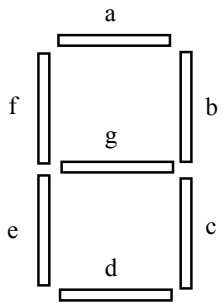
14. În ultimul pas se testează funcționarea circuitului. Intrările S0 și S1 se conectează la comutatoarele S0 și S1de pe placa de test, pe intrarile A0,A1,A2,A3 conectați SW1, CLK, S7 și SW2, iar din ieșirea circuitului duceți-vă pe unul din LED-urile plăcii de test, de exemplu LED-ul Q3. Schimbând starea comutatoarelor S0 și S1 observați efectul intrărilor selectate asupra ieșirii.

4.3. Decodificator binar – 7 segmente

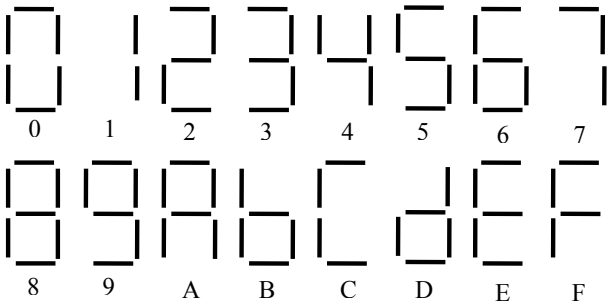
Temă: Să se implementeze și să se realizeze fizic un decodificator binar 7 segmente pentru o cifră hexazecimală. Se va ține cont că elementul de afișaj este de tipul anod comun (anodurile tuturor elementelor de afișare, a LED-urilor, sunt legate în comun, deci pentru aprinderea unui segment va trebui pus un potențial de 0L pe pinul asociat).

Un afișaj 7 segmente este prezentat mai jos, odată cu valorile care se doresc afișate:

Pentru obținerea ecuațiilor între ieșire și intrare ne vom folosi de tabelul de mai jos care a fost scris ținnd cont de **Figura 14 (b).** și diagramele VK.



a. identificarea segmentele



b. corespondența între valoarea afișată și codul hexazecimal trimis

Figura 14. Element de afișaj de tipul digit

Cod Hexa	Intrări				Ieșiri spre afișaj						
	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
A	1	0	1	0	1	1	1	0	1	1	1
B	1	0	1	1	0	0	1	1	1	1	1
C	1	1	0	0	1	0	0	1	1	1	0
D	1	1	0	1	0	1	1	1	1	0	1
E	1	1	1	0	1	0	0	1	1	1	1
F	1	1	1	1	1	0	0	0	1	1	1

Codul sursă al programului este prezentat mai jos:

```
MODULE bin7seg
TITLE 'circ decodificator pt. afisaj cu 7 segmente'

@alternate

declarations
 A, B, C, D pin 2, 3, 4, 5;
 /a, /b, /c, /d, /e, /f, /g pin 18, 17, 16, 15, 14, 13, 12 istype ' com ';

equations

    a = /A & /C + B & /D + B & C + /A & D + /B & /C & D + A & C & /D;
    b = /A & /C + /C & /D + A & B & /D + A & /B & D + /A & /B & /D;
    /c = /A & B & /C & /D + B & C & D + /A & C & D;
    d = /B & D + /A & /C & /D + A & B & /C + A & /B & C + /A & B & C;
    e = /A & /C + B & D + C & D + /A & B;
    f = /A & /B + B & D + /C & D + /A & C + /B & C & /D;
    g = /A & B + B & /C + A & D + /C & D + /B & C & /D;

END
```

Exercițiul 3. Pentru segmentul „f” aflați singuri ecuația. Înlocuiți ecuația similară existentă în exemplul de mai sus cu cea obținută de dumneavoastră.

Exercițiul 4. Înainte de a trece la programarea circuitului implementați codul sursă necesar a fi scris în zona vectorilor de test. Cu ajutorul acestui cod testați corectitudinea implementării ecuației logice a segmentului „f”, scrisă de dumneavoastră, dar și a celorlalte ecuații logice[1].

O realizare alternativă a aceluiași cod sursă dar, de data aceasta, folosindu-ne de artificiile limbajului ABEL-HDL este prezentată mai jos:

```
MODULE bin7seg

@alternate
 A, B, C, D pin 2, 3, 4, 5;
 /a, /b, /c, /d, /e, /f, /g pin 18, 17, 16, 15, 14, 13, 12 istype 'com';

equations
truth_table
        ([D, C, B, A]->[a, b, c, d, e, f, g])
        [0, 0, 0, 0]->[1, 1, 1, 1, 1, 1, 0];
        [0, 0, 0, 1]->[0, 1, 1, 0, 0, 0, 0];
        [0, 0, 1, 0]->[1, 1, 0, 1, 1, 0, 1];
        [0, 0, 1, 1]->[1, 1, 1, 1, 0, 0, 1];
        [0, 1, 0, 0]->[0, 1, 1, 0, 0, 1, 1];
        [0, 1, 0, 1]->[1, 0, 1, 1, 0, 1, 1];
        [0, 1, 1, 0]->[1, 0, 1, 1, 1, 1, 1];
        [0, 1, 1, 1]->[1, 1, 1, 0, 0, 0, 0];
        [1, 0, 0, 0]->[1, 1, 1, 1, 1, 1, 1];
        [1, 0, 0, 1]->[1, 1, 1, 1, 0, 1, 1];
        [1, 0, 1, 0]->[1, 1, 1, 0, 1, 1, 1];
        [1, 0, 1, 1]->[0, 0, 1, 1, 1, 1, 1];
        [1, 1, 0, 0]->[1, 0, 0, 1, 1, 1, 0];
        [1, 1, 0, 1]->[0, 1, 1, 1, 1, 0, 1];
```

[1] Firma Lattice producătoare a circuitelor din familia GAL garantează un număr de 100 de cicluri de scriere/ștergere a acestor dispozitive. Prin testarea corectitudinii ecuațiilor ce descriu funcționarea circuitului evitați ciclurile inutile de scriere/ștergere și prelungiți „viața” unui astfel de circuit.

```
        [1, 1, 1, 0]->[1, 0, 0, 1, 1, 1, 1];
        [1, 1, 1, 1]->[1, 0, 0, 0, 1, 1, 1];
END
```

Exercitiul 5. Concepeți şi încercați să realizați practic un mod prin care în momentul în care un pin îşi schimbă starea elementul de afişare să se stingă.

Ultimul exemplu al aceluiaşi cod sursă de mai sus este prezentat în continuare:

```
module  bcd7seg
title 'decodificator 7 segmente făcut noaptea tîrziu'

D, C, B, A      pin 2,3,4,5;
a,b,c,d,e,f,g   pin 13,14,15,16,17,18,19 istype 'com';

bcd     = [D,C,B,A];

ON,OFF  = 0,1;                  " pentru afişaje cu anod comun
L,H,X,Z = 0,1,.X.,.Z.;

@dcset
truth_table ( bcd -> [ a , b , c , d , e , f , g ] )
                0  ->  [ ON,   ON,   ON,   ON,   ON,   ON,   OFF  ];
                1  ->  [ OFF,  ON,   ON,   OFF,  OFF,  OFF,  OFF  ];
                2  ->  [ ON,   ON,   OFF,  ON,   ON,   OFF,  ON   ];
                3  ->  [ ON,   ON,   ON,   ON,   OFF,  OFF,  ON   ];
                4  ->  [ OFF,  ON,   ON,   OFF,  OFF,  ON,   ON   ];
                5  ->  [ ON,   OFF,  ON,   ON,   OFF,  ON,   ON   ];
                6  ->  [ ON,   OFF,  ON,   ON,   ON,   ON,   ON   ];
                7  ->  [ ON,   ON,   ON,   OFF,  OFF,  OFF,  OFF  ];
                8  ->  [ ON,   ON,   ON,   ON,   ON,   ON,   ON   ];
                9  ->  [ ON,   ON,   ON,   ON,   OFF,  ON,   ON   ];
end
```

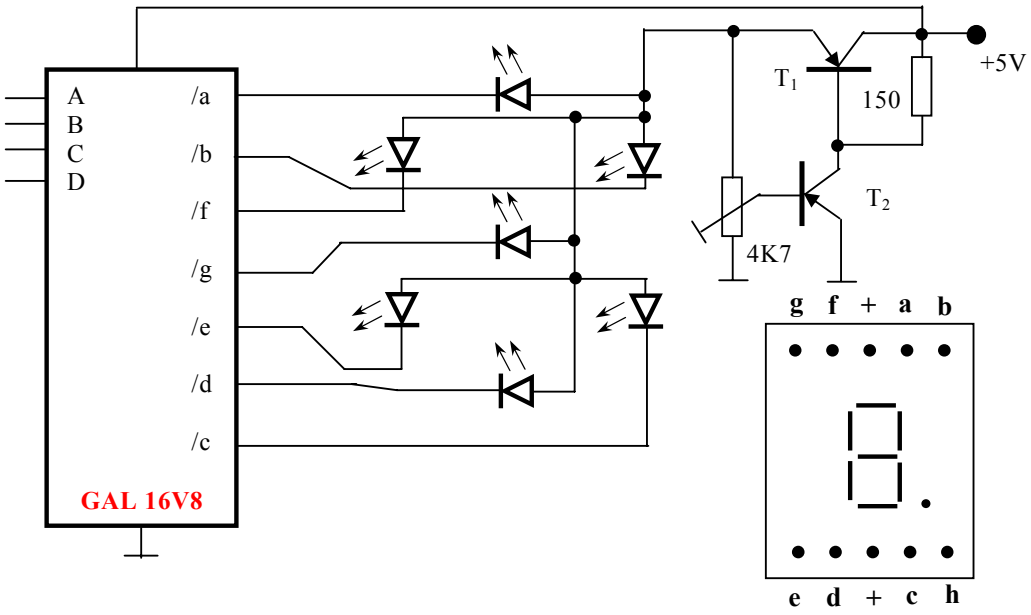


Figura 15. Conectarea circuitului la elementul de afişare; dispunerea pinilor acestuia (vedere de sus)

Pentru testarea funcționării decodificatorului implementat în una din cele trei variante prezentate vom folosi conexiunile din schema prezentată în **Figura 15.** Cu ajutorul tranzistoarelor T_1 şi T_2 se formează un stabilizator de tensiune care furnizează 2V şi astfel nu mai suntem obligați să introducem în circuit 7 rezistoare (ştiindu-se că pe un LED tensiunea maximă este de 2V). Intrările A, B, C, D se conectează la comutatoarele S0, S1, S2, S3 de pe placa de test. Pinii elementului de afişare sunt prezentați în aceeaşi figură în partea dreaptă jos.

ANEXA 1

Harta Memoriei

GAL 16V8	*Funcţia*
0000…2047	Matricea ŞI
2048…2055	XOR-ul pt. fiecare OLMC
2056…2119	UES (User Electronic Signature)
2120…2127	AC1 pt. fiecare OLMC
2128…2191	PTD (Product Term Disable)
2192	SYN
2193	AC0

ANEXA 2

Instrucţiuni, directive şi simboluri ale limbajului ABEL-HDL:

@alternate

Aduce un set alternativ de operanzi faţă de cei standard ai limbajului ABEL care rămân şi ei activi. Operatorii alternativi rămân în vigoare până se foloseşte directiva **@standard** sau până când se ajunge la sfârşitul modulului.

Standard	Alternativi	
!	/	NOT
&	*	AND
#	+	OR
$	:+:	XOR
!$	:*:	XNOR

Singura problemă care apare cu utilizarea setului alternativ de operanzi ţine de faptul că aceştia suprascriu operaţiile de înmulţire, adunare şi împărţire.

@dcset

Permite utilizarea stărilor "indiferent" în mod favorabil optimizării funcţiilor rezultate, dându-le valori de zero sau unu (În ultimul exemplu care apare stările începând cu 10 nu ne mai interesează, dar ele există şi vor fi folosite de compilator pentru minimizarea funcţiilor de ieşire).

.Z.

Înaltă impedanţă.

.X.

Stare pinului, a nodului nu contează

com Semnalul dat este obţinut exclusiv prim metode combinaţionale. Nu are nici un registru asociat cu el şi orice posibil registru compilatorul trebuie să îl excludă.

istype Sintaxă:

semnal [PIN | NOD] număr **ISTYPE** 'atribut';

semnal - un identificator de pin sau nod
număr - parametru opţional; numărul unui pin sau număr
atribut - un şir de caractere ce specifică tipul pinului sau nodului; în cazul mai multor caracteristici acestea se despart prin virgulă

Este folosit în mod special în momentul în care se fac proiecte independente de arhitectură. Chiar când se specifică dispozitivul de la început aceste atribute asigură

portabilitate şi consistenţă codului în momentul unei reproiectări sau când se doreşte înlocuirea dispozitivului.

test_vectors

Sintaxa:

```
TEST_VECTORS [notă]
(intrări   -> ieşiri   )
in_vaori -> out_valori ;
...
```

notă - un text care comentează vectorii de test
intrări - un identificator sau un număr de identificatori care specifică numele semnalelor de intrare în dispozitiv
ieşiri - identificator sau identificatori care dau ieşirile din dispozitiv
in_valori - valoare sau valori ce vor fi prezentate la intrare
out_valori - valoare sau valori care rezultă din datele de intrare

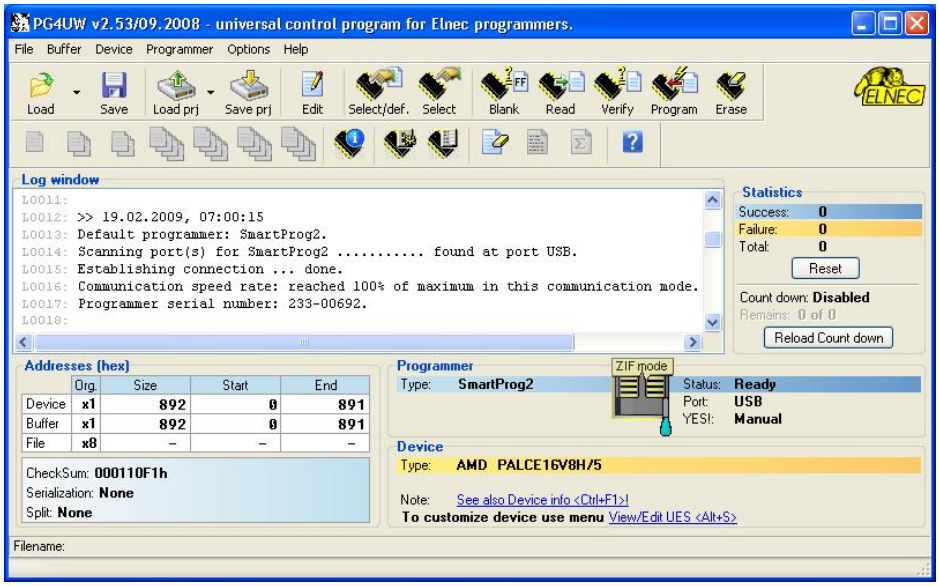
Caracterizează funcţionarea unui dispozitiv definind o funcţie din spaţiul intrărilor în spaţiul ieşirilor. Folosit în simulări a unui model dintr-un dispozitiv şi pentru testare funcţională a unui dispozitiv programat.

ANEXA 3

În vederea programării unui circuit de tip GAL se vor urmări paşii:

1. **Se lansează în execuţie aplicaţia ce controlează programatorul.** Pentru atingerea acestui obiectiv se urmăresc paşii: „*Start*”, „*Programs*”, „*Elnec*” şi în final se lansează aplicaţia „*Pg4uw*”.

   Fereastra grafică a aplicaţiei va fi una similară cu cea de mai jos:

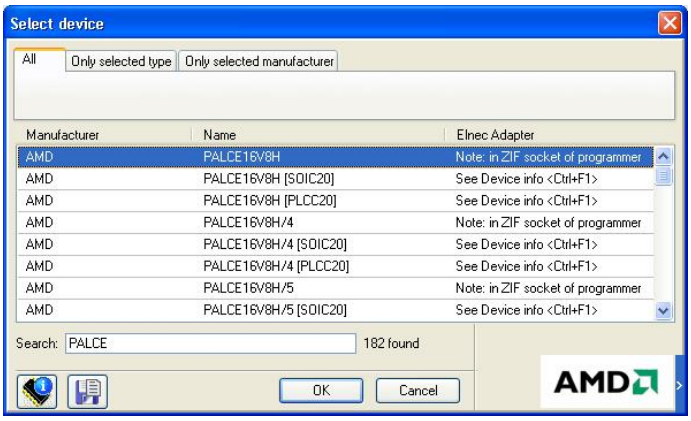


   Este foarte important ca pachetul *software* să găsească echipamentul *hardware* cu ajutorul căruia se face programarea dispozitivelor GAL. În fereastra de log („***Log window***” în figura anterioară) trebuie să găsiţi un mesaj similar cu cel prezentat în figură.

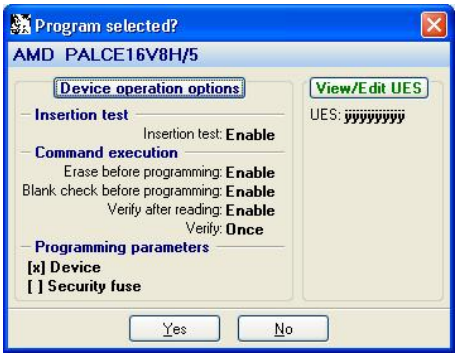
2. **Se alege tipul circuitului integrat ce va fi programat.** Prin apăsarea butonului „*Select*” de pe interfaţa grafică. Fereastra de dialog ce va apare va fi similară cu cea din figura de mai jos.

   Atenţie ! Trebuie ales atât tipul circuitului ce va fi programat cât şi producătorul acestuia. De exemplu, sub acelaşi nume GAL16V8C sunt produse circuite şi de către firma Lattice dar

şi de compania National Semiconductor. Deşi circuitele sunt similare ca funcţionare şi pini (funcţie a acestora) există diferenţe în algoritmii de programare.

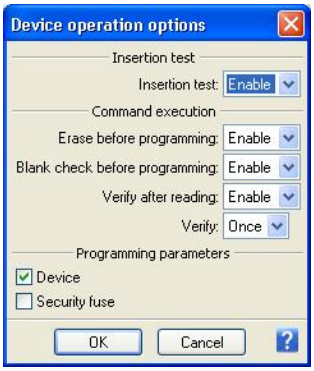


3. **Se încarcă fişierul JEDEC** apasând butonul „*Load*" de pe interfaţa grafică principală a programului.
4. **Se iniţializează procedura de programare** prin apăsarea butonului „*Program*" de pe interfaţa grafică principală aplicaţiei software de programare. Această acţiune determină apariţia următoarei ferestre.



Pentru ca programarea să reuşească fără nici o problemă trebuie ca toate opţiunile ce pot fi selectate să fie similare cu cele preuentate în figura de mai sus. În mod obligatoriu alegeţi ca înaintea scrierii să se facă o ştergere a dispozitivului iar parametrii de programare să fie preluaşi din dispozitiv (aceştia sunt particulari fiecărui circuit în parte şi fiecărui producător al aceluiaşi circuit în parte).

5. Dacă opţiunile de programare nu sunt corect alese le puteţi modifica prin apăsarea butonului „*Device operation options*" care determină apariţia unei ferestre de dialog care va permite modificarea tutror opţiunilor şi paşilor ce se vor executa atât înaintea cât şi ulterior programării (vezi figura de mai jos).



6. **Se porneşte procesul de programare a circuitului.** În cazul efectuării unei programării reuşite acest lucru va fi menţionat printr-un mesaj specific.