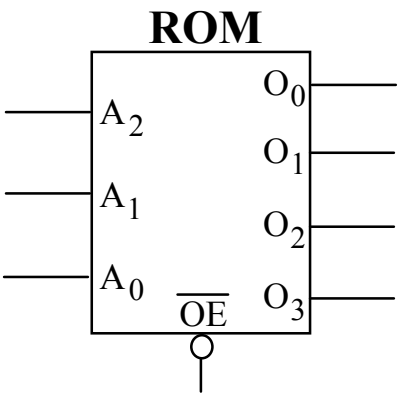
8 STRUCTURI PROGRAMABILE

8.1 Memoria ROM

Memoria ROM (**R**ead **O**nly **M**emory) este un circuit combinațional care stochează permanent date binare, iar această informație poate fi numai citită. Această structură este de obicei definită ca un convertor de cod compus dintr-un decodificator şi un codificator. Vectorul de intrare în decodificator este interpretat ca o **adresă**, iar datele obținute la ieşirea codificatorului reprezintă informația memorată la adresa respectivă.

În figura 8.1 s-a luat un exemplu de memorie ROM care conține 8 cuvinte de câte 4 biți. O combinație binară care se aplică pe cele 3 intrări de adresă, A_2, A_1 şi A_0, selectează unul din cele 8 cuvinte, iar cei 4 biți de date ai cuvântului selectat sunt disponibili la ieşirile O_0, O_1, O_2 şi O_3, cu condiția ca intrarea $\overline{OE}$(**O**utput **E**nable) să fie activată (în exemplul nostru activarea se face pe 0 logic). Dacă $\overline{OE}=1$ ieşirile memoriei sunt în starea de înaltă impedanță (high Z). Tabelul de adevăr din figură este numai un exemplu care arată o posibilitate de implementare a 4 funcții binare de câte 3 variabile.



A_2	A_1	A_0	$\overline{OE}$	O_0	O_1	O_2	O_3
0	0	0	0	1	1	1	0
0	0	1	0	1	1	0	1
0	1	0	0	1	0	1	1
0	1	1	0	0	1	1	1
1	0	0	0	0	0	0	1
1	0	1	0	0	0	1	0
1	1	0	0	0	1	0	0
1	1	1	0	1	0	0	0
x	x	x	1	high Z	high Z	high Z	high Z

Fig. 8.1 *Memorie ROM de 8 cuvinte de 4 biți şi harta memoriei*

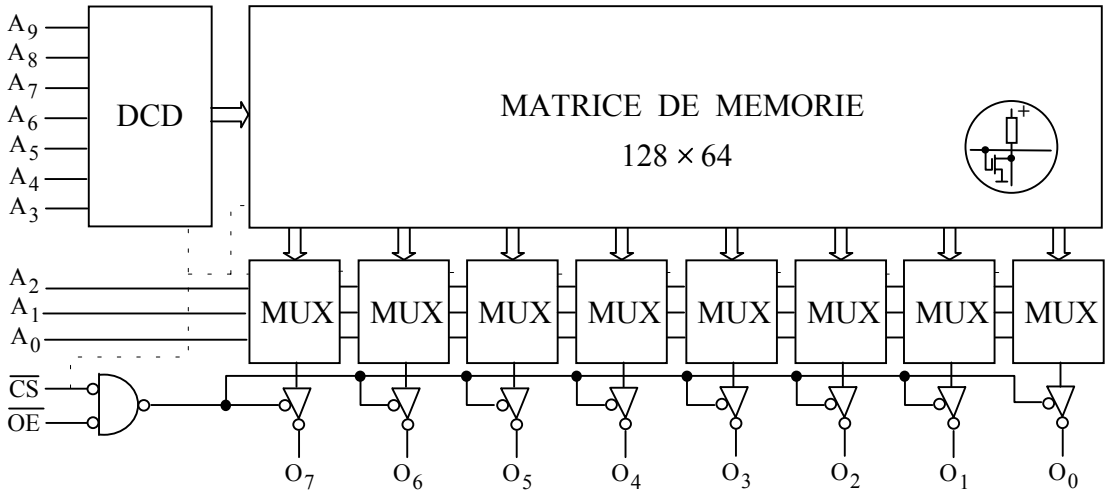


Fig. 8.2 *Structura unei memorii ROM de 1024 cuvinte de 8 biţi*

Structura internă a unui cip de memorie ROM organizat pe 1024 cuvinte de câte 8 biţi, adică $1K \times 8$, este prezentată în figura 8.2. Matricea de memorie se construieşte sub o formă cât mai apropiată de cea a unui pătrat. Liniile matricei sunt selectate cu ajutorul decodificatorului cu 7 intrări de selecţie, iar coloanele cu ajutorul celor 8 multiplexoare cu câte 3 intrări de selecţie. Intrarea suplimentară $\overline{CS}$ (**C**hip **S**elect), activă pe 0 logic, pregăteşte cipul de memorie în vederea unei operaţii de citire a datelor. Prin dezactivare, acţionează asupra circuitelor din structură în scopul reducerii consumului. Desenul mai conţine şi aspectul unui element din matricea de memorie, dacă aceasta este realizată în tehnologie MOS.

Există mai multe tipuri constructive de memorie ROM. Memoriile ROM **programabile prin mască**, sau **mask ROM**, sunt circuite la care harta memoriei se introduce în procesul de fabricaţie al circuitului integrat. Ele se realizează la comandă, în serie mare, pentru că realizarea unei măşti "la cerere" costă câteva mii de dolari.

Memoriile **PROM** (**P**rogrammable **ROM**) se fabrică cu toţi biţii poziţionaţi pe un anumit nivel logic. Există însă posibilitatea ca utilizatorul să-şi înscrie singur, acolo unde doreşte, valorile logice complementare. Acest lucru se face prin arderea unui fuzibil, o peliculă subţire de CrNi, care se vaporizează la trecerea unui curent suficient de mare prin el. Programarea memoriei constă în selecţia adresei şi a liniei de date şi aplicarea unui impuls de tensiune (10-30V) pe un pin special de programare. Evident că programarea nu se poate face decât o singură dată.

Memoriile **EPROM** (**E**rasable **PROM**) sunt programabile ca şi PROM-urile, dar de mai multe ori, pentru că ele pot fi "şterse" prin expunere la radiaţii ultraviolete cu o lungime de undă specifică. Matricea de memorie conţine **tranzistoare FAMOS** (**F**loating **A**valanche Injection **MOS**) adică tranzistoare care conţin o poartă metalică flotantă, izolată de substrat prin 10^{-7} m de SiO_2. Prin aplicarea unui impuls negativ de amplitudine mare pe drenă, joncţiunea *pn* constituită de drenă şi substrat este străpunsă prin avalanşă şi, prin efect tunel, electronii sunt injectaţi în poarta flotantă. Numărul lor depinde de amplitudinea şi durata impulsului. Această sarcină electrică stocată în poartă determină crearea unui canal *p* în substrat şi tranzistorul conduce. Radiaţia ultravioletă crează perechi electroni-goluri şi permite descărcarea porţii ([Valachi, 1986]). Însă izolaţia nu este perfectă şi în timp se pierd electroni, dar se pare că circa 70% din sarcină este regăsită şi după 10 ani.

Memoriile **EEPROM** (**E**lectrically **EPROM**) sunt similare EPROM-urilor, numai că ştergerea se poate face pe cale electrică. Izolaţia porţii este mult mai subţire decât la EPROM-uri, şi sarcina electrică negativă poate fi eliminată prin aplicarea unei tensiuni de polaritate inversă pe poarta tranzistorului care nu este flotantă ([Wakerly,1990]).

Memoriile maskROM şi PROM se realizează atât în tehnologie bipolară cât şi în tehnologie MOS, în timp ce memoriile EPROM şi EEPROM nu se pot realiza decât în tehnologie MOS.

Figura 8.3 prezintă formele de undă tipice pentru un ciclu de citire a datelor din memorie. Unul din parametrii cei mai importanţi ai memoriei ROM este **timpul de acces**, definit ca intervalul de timp scurs din momentul aplicării adresei pe intrările circuitului şi până ce datele memorate la adresa respectivă sunt disponibile la ieşiri, în condiţiile în care semnalele $\overline{CS}$ şi $\overline{OE}$ sunt activate. Aceşti timpi de acces sunt mai mici de 100ns la memoriile bipolare, dar pot atinge aproape 500ns la memoriile MOS.

Progresele tehnologice au determinat modificări importante în performanţele memoriilor ROM. Dacă memoriile EPROM în tehnologie NMOS au timpi de acces de aproape 500ns, cele realizate în tehnologie CMOS au timpi de acces mai mici de 100ns. O memorie EPROM poate fi ştearsă şi reprogramată de circa 100 ori, iar timpul de stocare a informaţiei poate depăşi 10 ani. O memorie EEPROM poate fi ştearsă şi reprogramată de circa 10000 ori, iar timpul de stocare a informaţiei poate fi de 100 ani. În sfârşit, o memorie **FLASH**(un EEPROM de capacitate mare) poate avea un timp de acces de 50ns, 100000 cicluri garantate de programare/ştergere şi timp de stocare a datelor de peste 20 ani. Notaţia **E^2PROM** este uneori folosită pentru memoriile EEPROM ([AMD, 1996]).

Revenind la formele de undă din figura 8.3, observăm existenţa unui **timp de reţinere** a datelor la ieşiri, după ce adresa a fost modificată, notat cu t_{hold} , un **timp de menţinere** a datelor după dezactivarea semnalului $\overline{OE}$, notat cu t_{od}, şi un timp de menţinere a ieşirilor în "high *Z*" după activarea semnalului $\overline{OE}$, notat cu t_{oe}. De obicei sunt date în catalog valorile maxime pentru aceşti timpi, iar pentru t_{od} şi t_{oe} se specifică uneori şi valorile minime.

Memoriile ROM pot avea dimensiuni variabile, de la cele mici cum ar fi 32 cuvinte de câte 8 biţi, până la 1M cuvinte de câte 8 biţi, adică 2^{20} cuvinte de 8 biţi. Dacă avem nevoie de o capacitate de memorie mai mare decât cea oferită de un singur circuit integrat, atunci trebuie să facem o **extindere a capacităţii de memorie**.

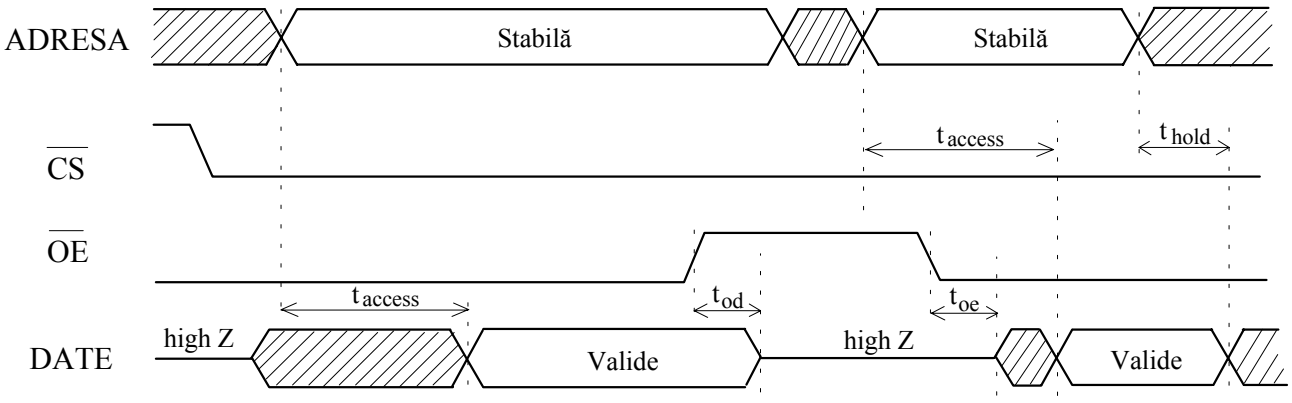


Fig. 8.3 *Forme de undă pentru ciclul de citire a datelor*

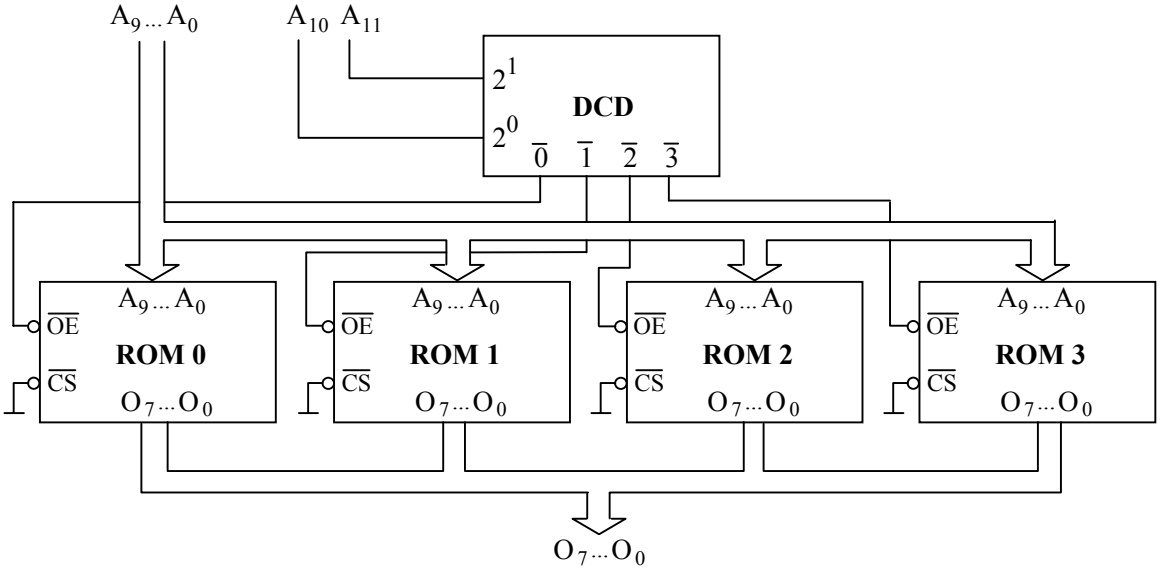


Fig. 8.4 *Extinderea numărului de cuvinte de la 1K la 4K*

Extinderea capacității de memorie se poate face în sensul extinderii numărului de cuvinte, sau al numărului de biți pe cuvânt. Schema logică din figura 8.4 exemplifică extinderea numărului de cuvinte de la $1K$ la $4K$, folosind cipuri de $1K$. Intrările $A_0...A_9$ ale celor 4 cipuri identice de memorie ROM sunt conectate în paralel. Biții suplimentari de adresă A_{10} şi A_{11} se aplică unui decodificator, care activează prin $\overline{OE}$ numai unul din cele 4 cipuri. Ieşirile lor sunt conectate în paralel, dar ieşirile circuitelor neselectate sunt în "highZ" şi datele din circuitul selectat sunt disponibile pe ieşirile $O_7...O_0$.

Extinderea numărului de biți pe cuvânt se face conectând în paralel intrările de adrese ale tuturor cipurilor selectate şi alăturând ieşirile lor până ce obținem lungimea dorită a cuvântului de date. Evident că, de cele mai multe ori, trebuie să facem o extindere mixtă a capacității de memorare.

Una din aplicațiile memoriilor ROM constă în stocarea programelor permanente în sistemele cu microprocesoare. Componenta BIOS, de exemplu, este un set de programe încorporate în calculator, care furnizează operațiunile fundamentale, primare de control şi supraveghere a acestuia. Aceste programe sunt de fapt secvențe binare care sunt stocate în cipuri de memorie ROM.

O altă aplicație, care este şi cea mai importantă din punctul nostru de vedere, este implementarea funcțiilor binare, fără a mai fi necesare operații de minimizare a lor. Este suficient să introducem tabelul de adevăr al funcției în memorie. Am mai arătat că cipul de memorie ROM din figura 8.1 poate fi folosit pentru implementarea unui număr de 4 funcții binare distincte, fiecare de câte 3 variabile independente. În afară de comoditatea implementării funcțiilor binare cu memorii ROM, mai există o serie de avantaje indiscutabile: timpul de acces este de multe ori mai mic decât în cazul altor soluții, memoria poate fi oricând reprogramată pentru alte aplicații, prețurile memoriilor ROM sunt în continuă scădere ([Wakerly,1990]).

Dezavantajul implementării funcțiilor binare cu memorii ROM este dat mai ales de creşterea fantastică a capacității de memorie odată cu creşterea numărului de intrări în circuit, datorită rigidității posibilităților de adresare. Aceasta este însă o caracteristică a structurii interne a memoriei ROM şi a modului în care se accesează informația.

8.2 Structuri PLD

Structurile PLD (**P**rogrammable **L**ogic **D**evices) conţin porţi logice şi, în unele cazuri, circuite bistabile, aranjate în aşa fel încât interconectările între componente să poată fi modificate pentru a implementa diverse funcţii binare.

Structurile PLD combinaţionale conţin numai porţi logice cu conexiuni programabile, care permit implementarea comodă a funcţiilor binare reprezentate în formă disjunctivă şi înlătură dezavantajul semnalat la memoria ROM. Circuitele reprezentative din această categorie sunt **structurile PLA** (**P**rogrammable **L**ogic **A**rrays) şi **structurile PAL** (**P**rogrammable **A**rray **L**ogic). Acestea din urmă sunt marcă înregistrată a firmei AMD.

Structura PLA pentru n intrări şi m ieşiri este reprezentată în figura 8.5. Ea conţine 2 nivele de logică, o matrice de porţi ŞI şi o matrice de porţi SAU. Matricea ŞI este un decodificator condiţionat de dimensiuni reduse faţă de numărul mare de intrări, iar matricea SAU are rol de codificator. Un alt parametru important al circuitului este numărul de porţi ŞI din structură. Acest număr, pe care îl notăm cu p, este în general mult mai mic decât 2^n, care este numărul total al mintermenilor unei funcţii de n variabile, iar acest lucru limitează serios complexitatea funcţiilor ce pot fi implementate. Porţile ŞI au câte $2n$ intrări care pot fi conectate prin programare la oricare din cele n variabile de intrare, sub formă directă sau complementată. Porţile SAU au câte p intrări care pot fi conectate prin programare la ieşirile oricărei porţi ŞI. Un astfel de circuit relativ simplu ar permite implementarea unui număr de m funcţii binare, fiecare de câte n variabile independente, cu condiţia ca numărul mintermenilor să nu-l depăşească pe p. Este evident că, pentru implementarea funcţiilor binare, structura prezentată este superioară memoriei ROM.

Un circuit integrat tipic realizat în această configuraţie este 82S100, fabricat de firma Signetics. Circuitul are $n = 16$ intrări, $m = 8$ ieşiri şi $p = 48$ porţi ŞI. Matricea de porţi ŞI conţine $2\times16\times48=1536$ comutatoare programabile, iar matricea de porţi SAU numai $8\times48=384$ comutatoare programabile. Programarea se face o singură dată, prin arderea unui fuzibil, ca la memoria PROM.

Circuitul din figura 8.5 este destul de complicat din cauza reprezentării. O reprezentare simplificată a unui circuit PLA cu 4 intrări şi 3 ieşiri este dată în figura 8.6.

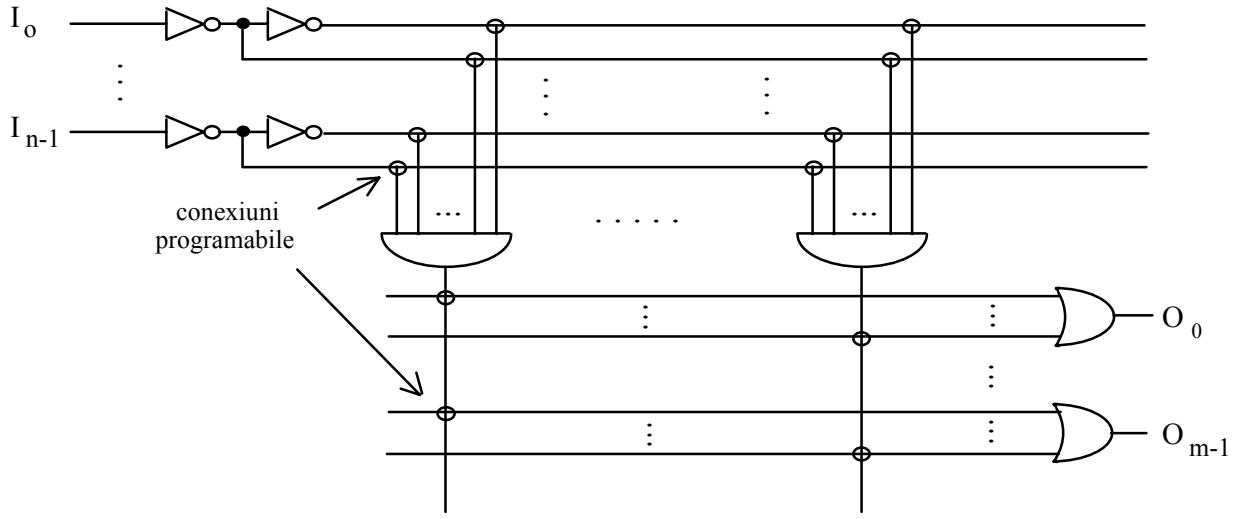


Fig. 8.5 *Structura internă a unui circuit PLA*

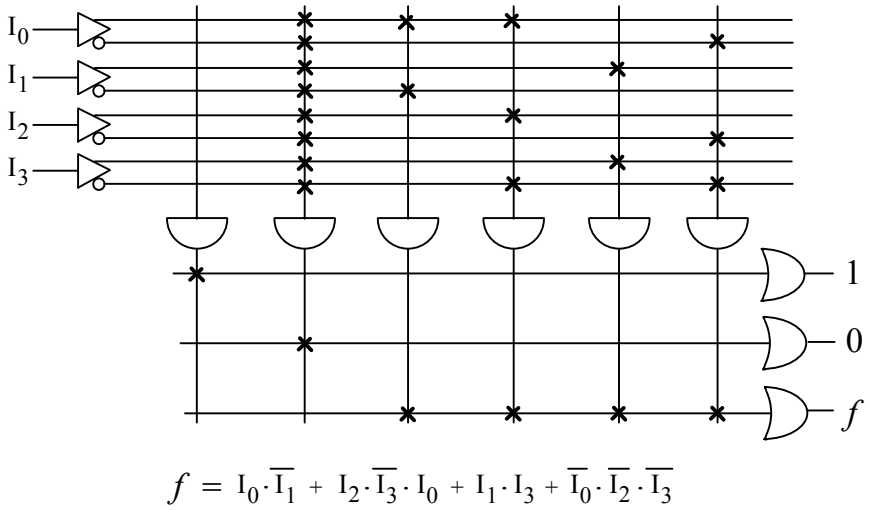


Fig. 8.6 *Reprezentare simplificată de PLA cu 4 intrări şi 3 ieşiri*

Circuitul a fost folosit la implementarea unor funcţii binare: funcţia unară, sau constanta 1, funcţia nulară, sau constanta 0 şi funcţia a cărei ecuaţie este dată în figură. Cred că modul de realizare a conexiunilor este destul de clar.

Structura PAL este asemănătoare cu structura PLA, dar matricea SAU are conexiuni fixe. Numărul porţilor ŞI este redus la 8, dar matricea de porţi ŞI este multiplicată, existând câte una pentru fiecare din porţile SAU. Ieşirile sunt prin intermediul unor porţi cu 3 stări şi din acest motiv, ele pot fi configurate în unele situaţii ca intrări.

Am constatat că aceste structuri conţin un număr mare de conexiuni programabile. Cum stabilim care sunt conexiunile pe care trebuie să le modificăm prin programare? Deşi acest lucru se poate face şi manual, bănuim că eficientizarea acestei sarcini se poate face prin programare. Există limbaje de programare specializate care, pornind de la o descriere formală a circuitului, furnizează prin compilare harta de conexiuni a lui. Unul din cele mai cunoscute limbaje de programare folosite în acest scop este **limbajul ABEL** (**A**dvanced **B**oolean **E**quation **L**anguage).

Structura fişierului sursă ABEL este dată în figura 8.7. Programul începe cu antetul "module", urmat de un nume şi se termină cu "end", care trebuie să aibă ataşat acelaşi nume. Expresia "title" este urmată de un şir de caractere incluse între ghilimele simple. Declaraţia "device" identifică circuitul care urmează să fie programat. Circuitul specificat trebuie să fie acceptat de compilator. Pot apare aici unele directive ca "@ALTERNATE" care permite recunoaşterea unui set alternativ de simboluri pentru operaţiile logice. Urmează declararea pinilor circuitului ca intrări şi ieşiri. După "equations" se scriu ecuaţiile funcţiilor binare, sau tabelul de adevăr, sau versiunea text a unei diagrame a stărilor, dacă este cazul.

module *numele modulului*
title *string*
identificator **device** *tipul dispozitivului*
declararea pinilor
alte declaraţii
equations
ecuaţiile
end *numele modulului*

Fig. 8.7 *Structura fişierului sursă ABEL*

```
module   Exemplu
title    ' Acesta este un exemplu '
CHIP_1   device   ' P16L8 ' ;
@ALTERNATE
" Input pins
I0, I1, I2, I3          pin 2, 3, 4, 5;
" Output pins
O1, O2, O3              pin 18, 17, 16;
equations
O1 = 1;
O2 = 0;
O3 = I0 * /I1 + I2 * /I3 * I0 + I1 * I3 + /I0 * /I2 * /I3;
end   Exemplu
```

Fig. 8.8 *Exemplu de program ABEL pentru circuitul din figura 8.6*

În figura 8.8 se dă un exemplu de fişier sursă scris în limbajul ABEL pentru circuitul din figura 8.6. Am ales descrierea logică a funcţiilor binare prin expresii boolene.

Structurile PLD secvenţiale au două forme constructive: fie la structurile PLD combinaţionale se adaugă câteva bistabile de tip D şi se obţine o structură de tip registru paralel, fie se adaugă o serie de **macrocelule programabile**. Macrocelulele sunt circuite cu funcţii selectabile. După cum se observă pe schema din figura 8.9, la pinul de ieşire se poate trimite una din ieşirile bistabilului sau o cale care ocoleşte bistabilul. Astfel de bistabile se numesc **bistabile îngropate** în macrocelule ([Wilkinson, 2002]).

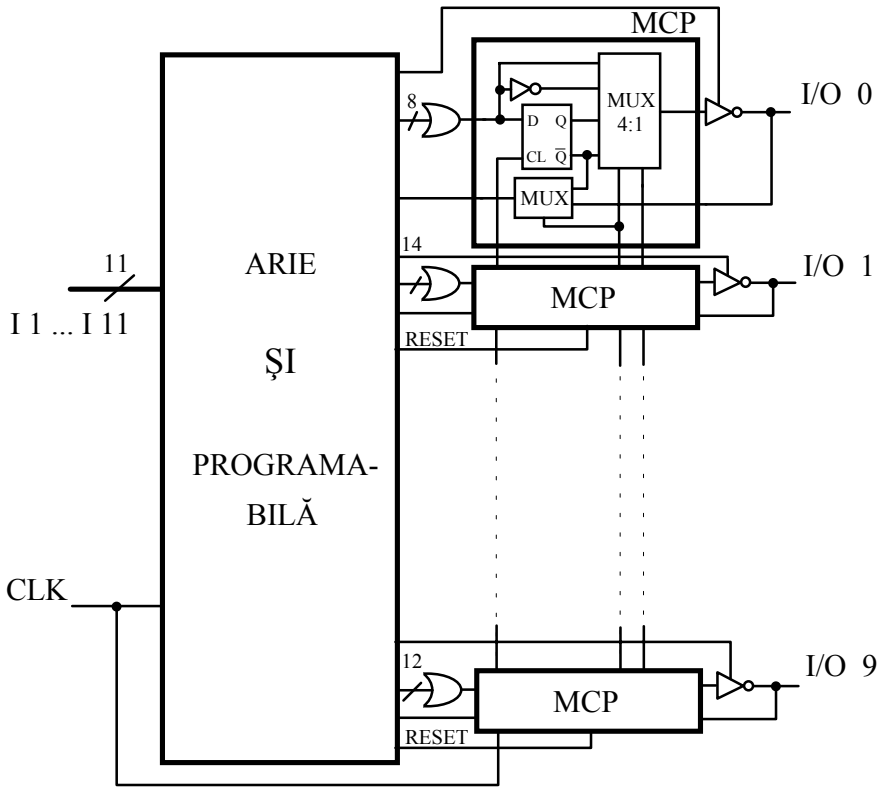


Fig. 8.9 *Structura simplificată a circuitului PAL22V10*

Un exemplu de circuit PLD secvențial de tip registru este circuitul integrat PAL16R4 care conține 16 intrări/ieşiri şi 4 bistabile de tip D. Schema completă a circuitului este dată în figura 8.10 şi vom utiliza acest circuit în exemplul care urmează. Un exemplu de circuit cu macrocelule este PAL22V10, care conține 22 intrări/ieşiri şi 10 macrocelule. Schema simplificată a circuitului este prezentată în figura 8.9.

Exemplul 8.1

Ne propunem să proiectăm o interfață calculator-microcalculator folosind circuitul PAL16R4. Interfața este de fapt un automat finit cu 4 intrări şi 4 ieşiri. Intrările sunt START şi T/R (Transmit/Receive) de la calculator, respectiv READY şi CYEND (Cycle End) de la microcalculator. Ieşirile sunt INACTIVE şi F/E (Full/Empty) spre calculator şi ATTENTION şi CYCLE spre microcalculator.

Diagrama stărilor este reprezentată în figura 8.11. Activarea semnalului START determină tranziția într-o stare în care se generează ATTENTION către microcalculator şi se aşteaptă răspunsul lui prin READY. Se aşteaptă activarea semnalului READY,

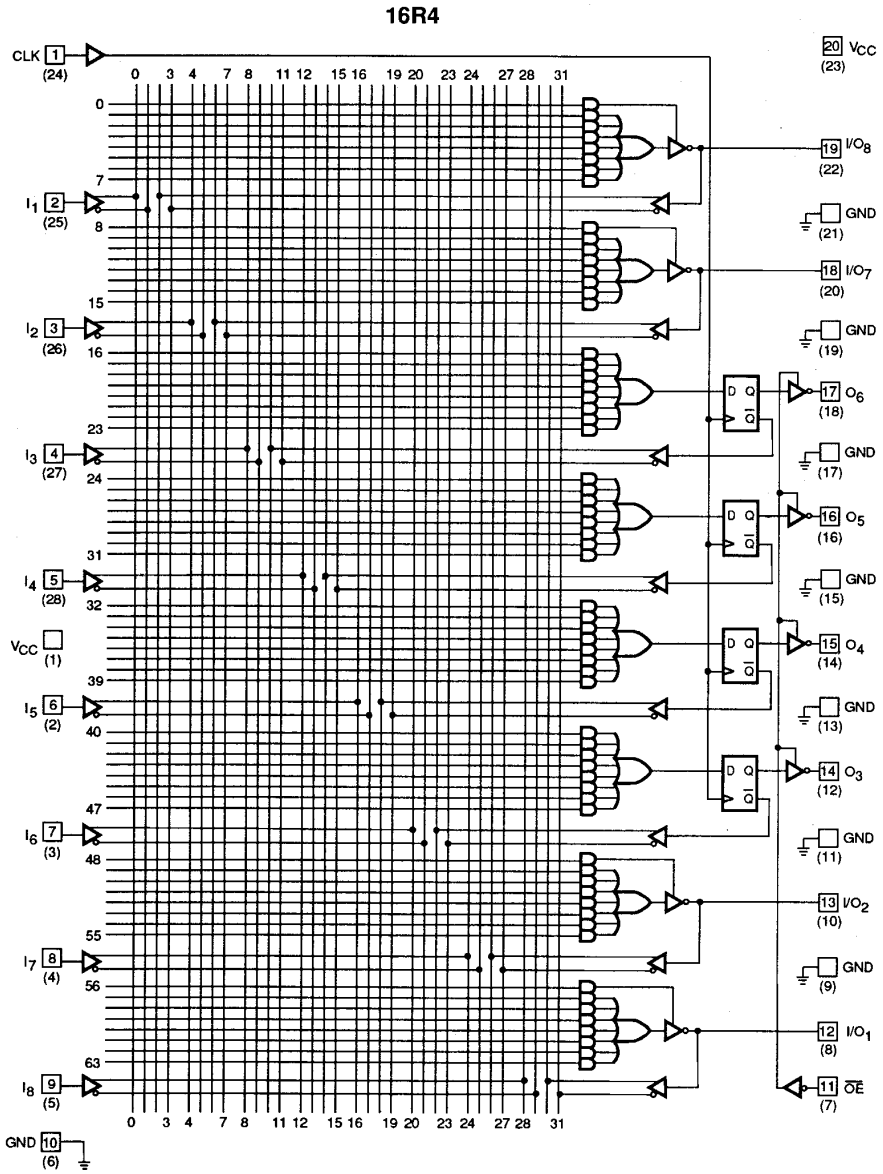


Fig. 8.10 *Structura circuitului PAL16R4*

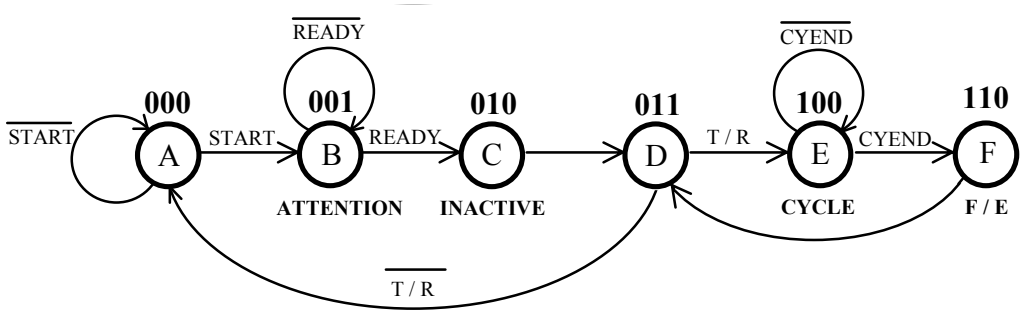


Fig. 8.11 *Diagrama stărilor interfeței calculator-microcalculator*

după care se trece în starea următoare, stare în care se trimite spre calculator semnalul INACTIVE, un semnal care durează o singură perioadă de ceas. În starea următoare, notată cu D, se testează semnalul T/R prin care calculatorul indică tipul operației dorite, transmisie sau recepție. Nu ne preocupă aici sensul de transmisie a datelor ci numai modul de aplicare a semnalelor de comandă. Dacă semnalul T/R este activat, interfața comandă trecerea într-o nouă stare în care se generează CYCLE şi se testează CYEND, semnal care anunță terminarea operației solicitate. Dacă CYEND este activat se trece în starea următoare, în care se comunică calculatorului, prin activarea semnalului F/E pe durata unei perioade de ceas, dacă s-au completat sau golit regiştrii de date.

Dacă alegem codurile stărilor indicate în diagramă, avem nevoie de 3 bistabile de tip D şi de o logică combinațională pentru a genera funcțiile de excitație şi ieşirile circuitului. Dacă folosim circuitul din figura 8.10, observăm că ieşirile bistabilelor sunt trecute prin inversoare cu 3 stări înainte de a ajunge la pinii de ieşire ai circuitului. Atribuim ieşirile O_6 lui Q_2, O_5 lui Q_1 şi O_4 lui Q_0. Ecuațiile stărilor următoare şi ale ieşirilor circuitului sunt deduse cu ajutorul tabelului tranzițiilor, construcția lui fiind lăsată în seama cititorului:

$$\overline{Q}_2^+ = \overline{Q}_2 \cdot \overline{Q}_1 + Q_1 \cdot \overline{Q}_0 + \overline{T/R} \cdot Q_1$$

$$\overline{Q}_1^+ = \overline{Q}_2 \cdot \overline{Q}_1 \cdot \overline{Q}_0 + Q_1 \cdot Q_0 + \overline{READY} \cdot Q_0 + \overline{CYEND} Q_2 \cdot \overline{Q}_1$$

$$\overline{Q}_0^+ = Q_2 \cdot \overline{Q}_1 + Q_1 \cdot Q_0 + READY \cdot Q_0 + \overline{Q}_1 \cdot \overline{Q}_0 \cdot \overline{START}$$

Semnale de ieşire ATTENTION, INACTIVE, CYCLE şi F/E sunt atribuite ieşirilor I/O_8, I/O_7, I/O_2 şi respectiv I/O_1, care au şi ele inversoare cu 3 stări pe ieşire. Ecuațiile lor devin:

$$\overline{ATTENTION} = Q_1 + \overline{Q}_0 \ ; \qquad \overline{INACTIVE} = Q_2 + \overline{Q}_1 + Q_0 \ ;$$

$$\overline{CYCLE} = \overline{Q}_2 + Q_1 \ ; \qquad \overline{F/E} = \overline{Q}_2 + \overline{Q}_1 \ ;$$

Semnalele de intrare START, READY, T/R şi CYEND sunt atribuite intrărilor I_1, I_2, I_3 şi respectiv I_4, iar pe I_5 se aplică intrarea RESET, absolut necesară aici, pentru inițializarea bistabilelor înainte de verificarea sintezei prin analiză Pspice. Schema electrică a interfeței realizată cu circuitul PAL16R4 este dată în figura 8.12. Ieşirea OUT4 nu este folosită, iar pe intrarea CLK se aplică semnalul de ceas, notat aici cu CLOCK.

Sigur că acest circuit nu are nici o valoare dacă nu arătăm care este starea celor 2048 de conexiuni programabile din matricea de conexiuni. Starea normală a acestor conexiuni este 1 logic, iar prin arderea fuzibilelor respective, ele trec în 0 logic. Programarea circuitului se poate face o singură dată. Structura interfeței este memorată în aria combinațională a circuitului integrat prin starea fuzibilelor.

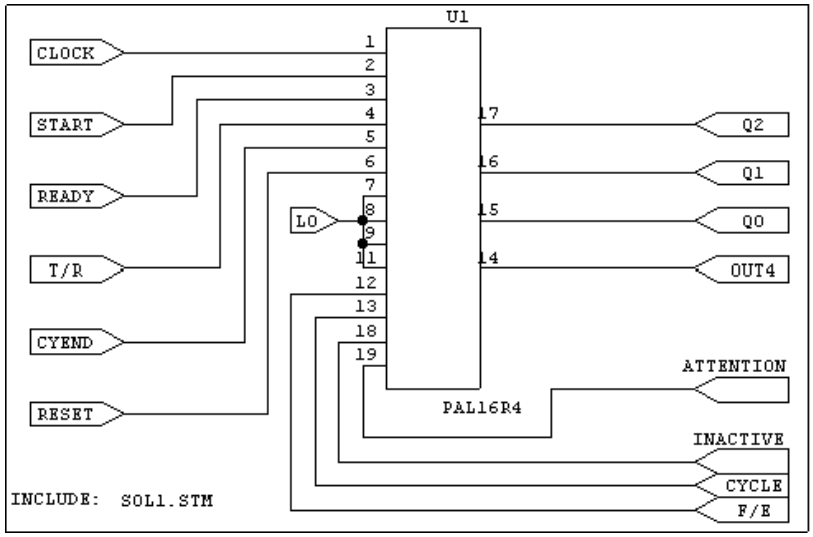


Fig. 8.12 *Schema electrică a interfeței realizată cu PAL16R4*

Fişierul care conține informația asupra stării fuzibilelor este un fişier în **format standard JEDEC** (**J**oint **E**lectronic **D**evice **E**ngineering **C**ouncil), fişier care este ataşat de simbolul grafic al circuitului PAL16R4 din biblioteca PSpice. Figura 8.13 arată conținutul acestui fişier pentru exemplul considerat.

Fişierul PAL.JED începe cu caracterul 02H (start of text) şi se termină cu caracterul 03H (end of text) şi este divizat în câmpuri, separate prin asterisc (*). Primul câmp este de identificare şi conține numele circuitului, atribuirea pinilor şi alte informații. D este un identificator pentru tipul circuitului, G este fuzibilul de siguranță, QF indică numărul total de fuzibile, iar F reprezintă starea implicită a fuzibilelor. L este un identificator pentru lista fuzibilelor, numerotate de sus în jos şi de la dreapta la stânga, începând de la L0000.

```
□$DEVICE
        PAL16R4;
$PIN
        1=CLOCK;
        2=START;
        3=READY;
        4=T/R;
        5=CYEND;
        6=RESET;
        12=F/E;
        13=CYCLE;
        18=INACTIVE;
        19=ATTENTION;
$END
*
D1234*
G0*
QF2048*
F0*
L0000 11111111111111111111111111111111*
L0032 11111111111111011111111111111111*
L0064 11111111111111111110111111111111*
L0256 11111111111111111111111111111111*
L0288 11111111110111111111111111111111*
L0320 11111111111111110111111111111111*
L0352 11111111111111111101111111111111*
L0512 11111111111011101111111111111111*
L0544 11111111111111011101111111111111*
L0576 11111111101110111111111111111111*
L0608 11111111111111110111111111111111*
L0768 11111111111011101110111111111111*
L0800 11111111111111011101111111111111*
L0832 11111011111111111101111111111111*
L0864 11111111110110101111111111111111*
L0896 11111111111111110111111111111111*
L1024 11111111110111101111111111111111*
L1056 11111111111111011101111111111111*
L1088 11110111111111111101111111111111*
L1120 10111111111111101110111111111111*
L1152 11111111111111110111111111111111*
L1536 11111111111111111111111111111111*
L1568 11111111110111111111111111111111*
L1600 11111111111111011111111111111111*
L1792 11111111111111111111111111111111*
L1824 11111111110111111111111111111111*
L1856 11111111111111101111111111111111*
□
```

Fig. 8.13 *Fişierul PAL.JED care conține harta conexiunilor pentru circuitul de interfață*

```
UCLOCK STIM(1,1) $G_DPWR $G_DGND
+       CLOCK
+       IO_STM TIMESTEP=0.5u
+       0c      0
+       label=loop
+       1c      1
+       2c      0
+       3c      goto loop -1 times

URESET STIM(1,1) $G_DPWR $G_DGND
+       RESET
+       IO_STM
+       TIMESTEP=0.5u
+       0c      1
+       2c      0
+       6c      1

USTART STIM(1,1) $G_DPWR $G_DGND
+       START
+       IO_STM
+       TIMESTEP=0.2u
+       0c      0
+       29c     1
+       57c     0

UREADY STIM(1,1) $G_DPWR $G_DGND
+       READY
+       IO_STM
+       TIMESTEP=0.5u
+       0c      0
+       20c     1
+       25c     0

UT/R STIM(1,1) $G_DPWR $G_DGND
+       T/R
+       IO_STM
+       TIMESTEP=0.2u
+       0c      0
+       53c     1
+       84c     0

UCYEND STIM(1,1) $G_DPWR $G_DGND
+       CYEND
+       IO_STM
+       TIMESTEP=0.5u
+       0c      0
+       40c     1
+       50c     0
```

Fig. 8.14 *Fişierul SOL1.STM care descrie semnalele de intrare în circuit*

Schema din figura 8.12 conţine un fişier inclus, prin directiva INCLUDE. Numele lui este SOL1.STM şi este un fişier de stimuli destinat generării semnalelor de intrare. Un exemplu de aplicare a acestor semnale este dat în figura 8.14.

Rezultatul analizei circuitului prin simulare Pspice, cu intrările stabilite prin fişierul SOL1.STM din figura 8.14, sunt date în figura 8.15. Pentru diferite modificări ale semnalelor de intrare, ieşirile circuitului răspund conform diagramei stărilor. Se observă că prin activarea RESET-ului ieşirile bistabilelor se resetează pe frontul crescător al ceasului.

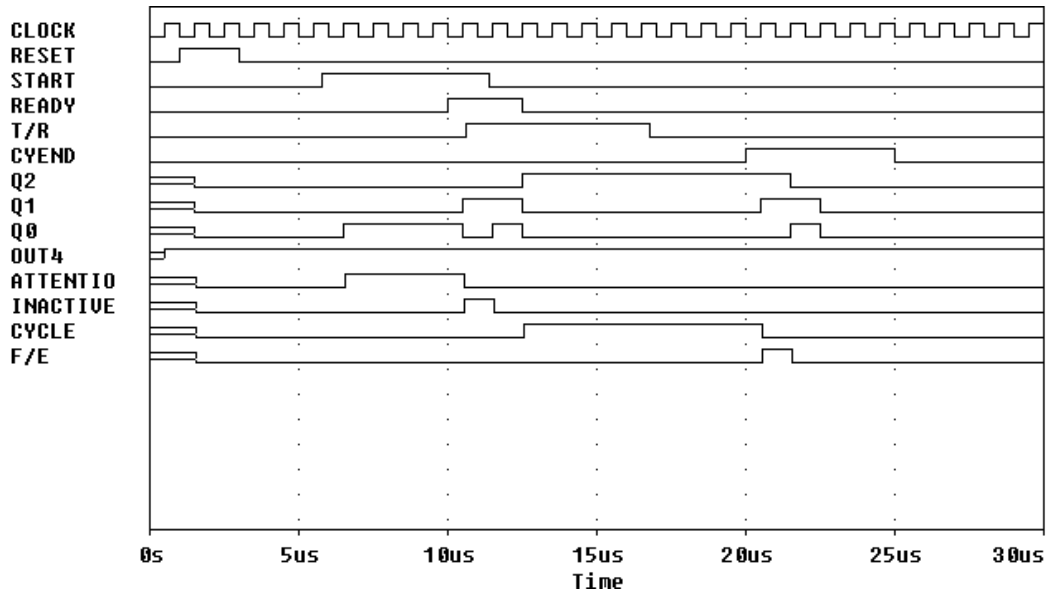


Fig. 8.15 *Formele de undă care descriu funcţionarea circuitului*

8.3 Structuri FPGA

Structurile FPGA (**F**ield **P**rogrammable **G**ate **A**rrays) sau **ariile logice reconfigurabile** oferă un grad sporit de integrare şi sunt cele mai răspândite circuite programabile. Ciclul redus de proiectare-fabricaţie şi densitatea elementelor integrate fac ca aceste circuite să fie preferate în proiectarea sistemelor numerice ([Vásárhelyi, 1998]).

Succesul acestor circuite este legat de arhitectură şi tehnologie. Ideea arhitecturii FPGA este prezentată în figura 8.16. Există trei elemente constructive de bază care se repetă de câte ori este necesar în structură: blocul logic, blocul de intrare-ieşire, şi resursele de interconectare a blocurilor, de fapt matrici de comutatoare programabile, numite şi **switchbox**-uri. Blocul logic poate conţine sute sau mii de porţi logice şi poate fi configurat diferit în funcţie de aplicaţie. El poate avea simplitatea unui tranzistor sau complexitatea unui microprocesor. Realizarea interconexiunilor, sau **rutarea**, permite o utilizare superioară a resurselor logice faţă de structurile PLD, unde legăturile se stabilesc de obicei prin intermediul unui singur switch. O variantă tipică de rutare constă din segmente de legătură de lungime variabilă, interconectabile prin switch-uri programabile electric. Alegerea numărului optim de segmente pentru legături este un procedeu dificil care nu este definitiv pus la punct. Alegând un număr nepotrivit de segmente de legătură, numai o mică parte din blocurile logice pot fi utilizate, ceea ce duce la irosirea inutilă a resurselor oferite de circuit. Pe de altă parte, o lungime necorespunzătoare a lor poate afecta performanţele de frecvenţă ale structurii ([Gontean, 1997]).

Implementarea switch-urilor programabile se realizează prin una din cele trei tehnologii majore: **SRAM** - în care comutatorul este un tranzistor controlat de un bit de memorie RAM, **antifuzibil** – un element de circuit cu două terminale între care rezistenţa este foarte mare în stare neprogramată şi devine foarte mică prin programare, **EPROM** – care foloseşte un tranzistor cu poarta flotantă, controlat prin injecţia de sarcină pe poartă.

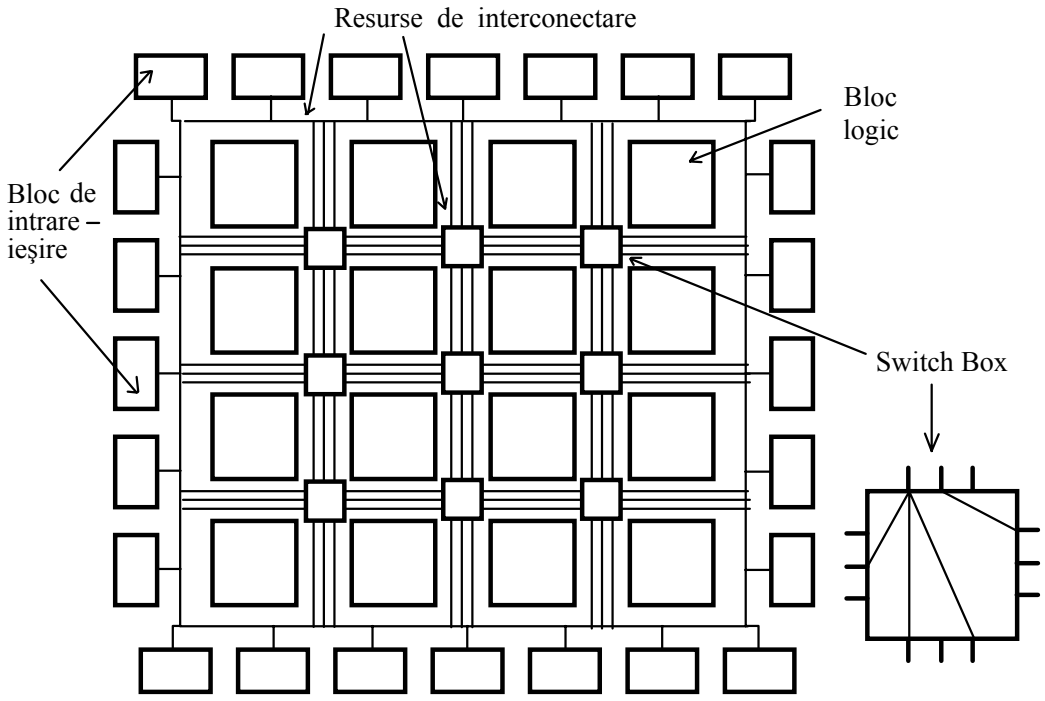


Fig. 8.16 *Arhitectura FPGA*

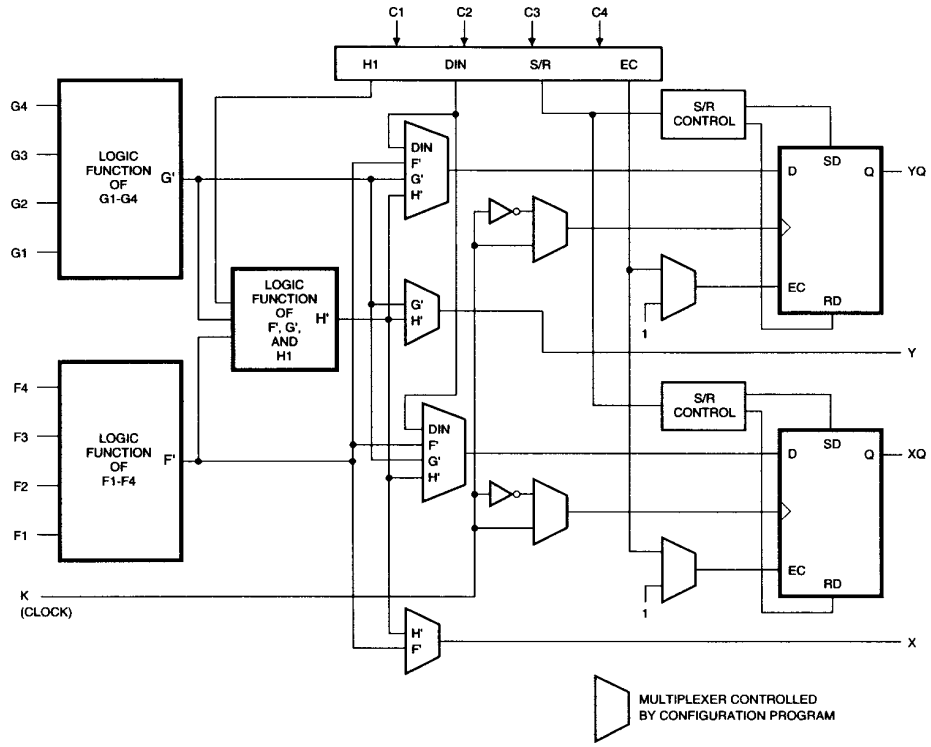


Fig. 8.17 *Schema simplificată a blocului logic la circuitul XC4000*

Structura blocului logic al circuitului XC4000 produs de firma Xilinx este dată în figura 8.17. Modulele F şi G sunt generatoare de funcţii binare programabile cu câte 4 intrări, iar împreună cu modulul H, care este tot un generator de funcţii binare, permit implementarea unor funcţii cu 9 variabile independente. Blocul logic mai conţine o logică combinaţională de selecţie şi 2 bistabile de tip D pentru stocarea rezultatelor date de generatoarele de funcţii. Ieşirile generatoarelor de funcţii se pot utiliza independent de ieşirile elementelor de stocare de tip registru ([Xilinx, 1993]).

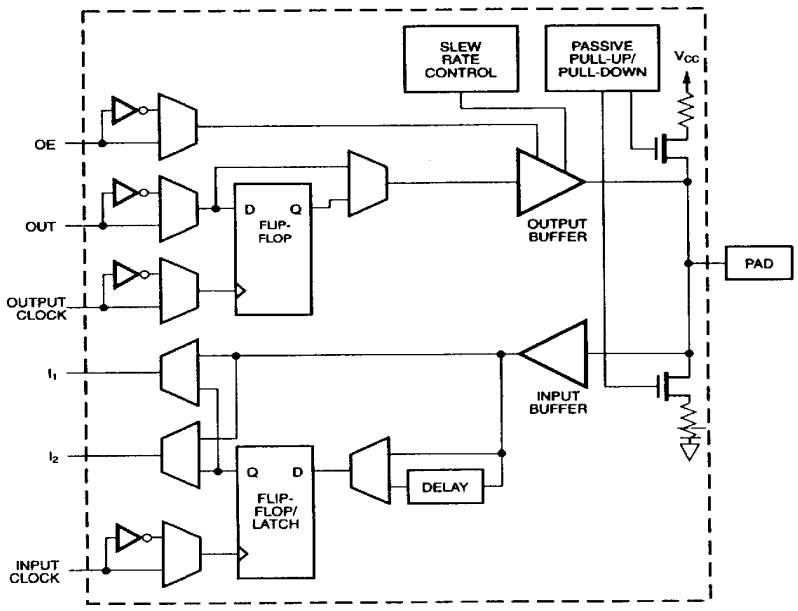


Fig. 8.18 *Schema simplificată a blocului de intrare/ieşire la circuitul XC4000*

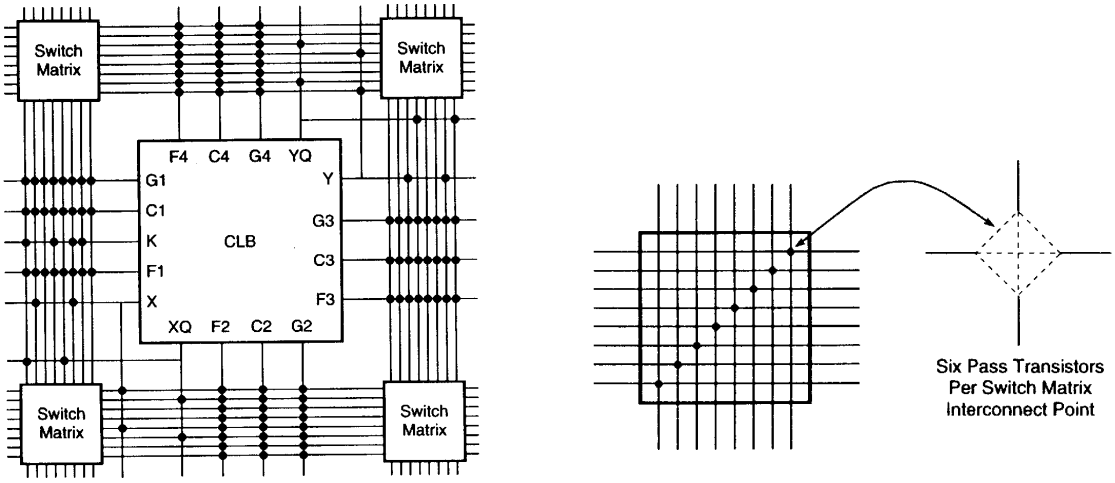


Fig. 8.19 *Conexiuni ale blocului logic cu switchbox-uri vecine şi structura unui switchbox*

Structura blocului de intrare/ieşire al circuitului XC4000 este dată în figura 8.18. Fiecare bloc de acest fel controlează un pin al circuitului integrat (pad). Blocul se poate configura ca port de intrare, port de ieşire sau port bidirecţional. Semnalele de intrare I_1 şi I_2 se pot aplica direct sau prin bistabilul de intrare. Semnalele de ieşire, care se pot inversa în interiorul blocului, se pot conecta direct la ieşirea pinului sau la bistabilul de ieşire. Slew rate-ul fiecărui buffer de ieşire este redus pentru a minimiza consumul şi perturbaţiile pe tensiunea de alimentare. Există şi un circuit intern de forţare a nivelului logic pe fiecare pin.

Toate conexiunile interne sunt compuse din segmente metalice cu puncte de cuplare programabile şi matrice de conexiuni, după cum se vede în exemplul din figura 8.19. Matricea de cuplare programabilă, sau switchbox-ul, este alcătuită din tranzistori de trecere utilizaţi pentru realizarea conexiunilor între linii. Observăm în figura 8.19 că la fiecare intersecţie a două linii există 6 tranzistoare care au rolul unor comutatoare ce pot fi deschise sau închise. În acest fel se poate realiza orice configuraţie posibilă de conexiuni.

Seria de circuite FPGA XC5200 aduce în plus o creştere a complexităţii blocurilor logice şi o modalitate mai eficientă de realizare a interconexiunilor. Celula logică are o configuraţie mai simplă decât cea prezentată în figura 8.17, dar blocul logic este format din 4 celule logice care pot fi interconectate folosind o matrice de conexiuni locale. Aceasta este conectată la matricea de interconexiuni generale prin intermediul a 24 noduri bidirecţionale. Cele 2 nivele de interconectare îmbunătăţesc granularitatea arhitecturii.

Dar circuitul care marchează trecerea la a doua generaţie de circuite FPGA este XC6200. Arhitectura acestui circuit poate fi privită ca o structură ierarhică. Celula de bază nu diferă prea mult faţă de cea de la XC5200, dar o arie de 4×4 celule de bază formează un bloc, o arie de 4×4 formează un grup de blocuri, iar o arie de 4×4 grupuri formează o unitate logică. Fiecare nivel ierarhic posedă resurse de conectare proprii, care au direcţii orientate spre cele 4 puncte cardinale ([Vásárhelyi,1998]).

Dar poate că lucrul cel mai important, mai ales prin prisma viitoarelor aplicaţii posibile, este faptul că acest circuit se poate reconfigura foarte rapid şi de un număr nelimitat de ori. Reconfigurarea circuitului se poate face în totalitate sau parţial, pe secţiuni, fără a afecta funcţionarea secţiunilor de circuit care nu se reconfigurează. Aceste circuite au fost folosite cu succes în experimente de hardware evolutiv, unde sunt lăsate să se reconfigureze, în scopul adaptării la mediu sau al autoreparării, urmărind o strategie evolutivă de reconfigurare ([Popa, 1998]).

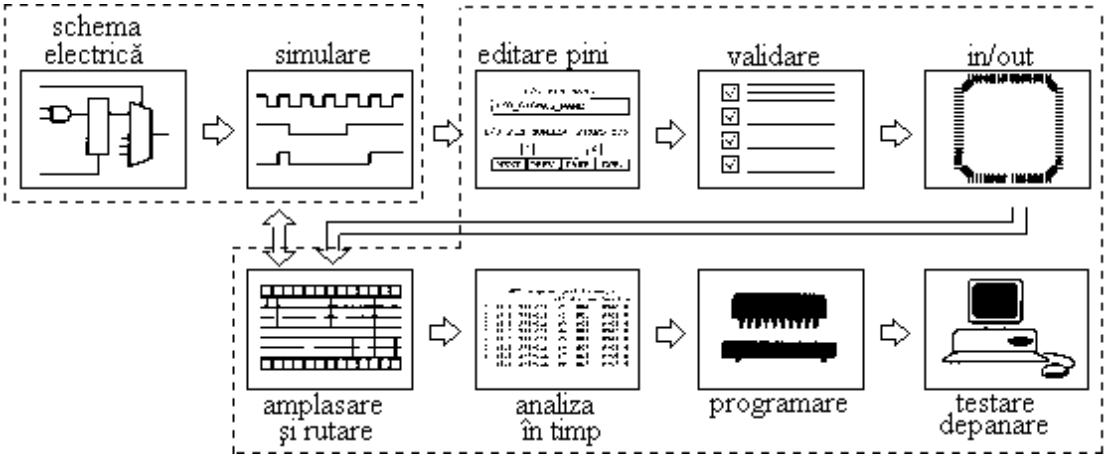


Fig. 8.20 *Etapele implementării cu FPGA-uri a unui sistem numeric*

Complexitatea actuală a structurilor FPGA a depăşit momentul proiectării manuale. În prezent performanţele implementărilor cu circuite FPGA depind esenţial de software-ul de proiectare şi de cel de plasare şi rutare utilizat.

Etapele de implementare cu circuite FPGA a unui sistem numeric sunt prezentate în figura 8.20. După editarea proiectului într-un mediu integrat CAD, cum ar fi OrCAD, Protel, Viewlogic sau altele, se face o simulare de tip Pspice a funcţionării. Modelul software al circuitului este livrat dispozitivului de programare recomandat de firma constructoare a circuitul FPGA respectiv. Urmează etapele de asignare a pinilor, validare a schemei electrice, asignarea automată a pinilor care nu au fost luaţi încă în considerare, şi operaţia de plasare şi rutare a blocurilor logice. Întârzierile introduse de comutatoarele programabile sunt evaluate printr-o nouă simulare PSpice a circuitului. Se face o nouă analiză în timp a circuitului şi se programează conform unui protocol recomandat de firmă. În final se face testarea structurii hardware rezultate prin programare.

8.4 Memoria RAM

Memoria RAM (**R**andom **A**ccess **M**emory) este un circuit care stochează biţi de informaţie într-o matrice de memorie, la fel ca memoria ROM. Diferenţa constă în faptul că informaţia utilă memorată în RAM trebuie mai întâi să fie "scrisă" acolo, înainte de a fi citită. Termenul RAM se traduce prin "memorie cu acces aleator", care sugerează faptul că timpul pentru citirea/scrierea unui bit nu depinde de poziţia bitului în matricea de memorie. Din acest punct de vedere şi memoria ROM este tot o memorie cu acces aleator, dar denumirea de RAM s-a păstrat numai la acest tip de memorie, din considerente istorice.

Există două tipuri constructive de memorie RAM: **RAM static** sau **SRAM**, în care biţii de date, odată ce au fost înscrişi, sunt memoraţi atât timp cât circuitul integrat este alimentat cu tensiune, şi **RAM dinamic** sau **DRAM**, în care datele memorate trebuie să fie mereu reîmprospătate prin citirea şi apoi rescrierea lor periodică în locaţiile respective de memorie, în caz contrar ele pierzându-se definitiv.

Cele mai multe memorii RAM sunt volatile, adică îşi pierd datele la întreruperea alimentării. Există şi memorii RAM nevolatile, numite **NOVRAM** (**No**n**v**olatile Static **RAM**) care dublează matricea de memorie RAM cu una de memorie EEPROM.

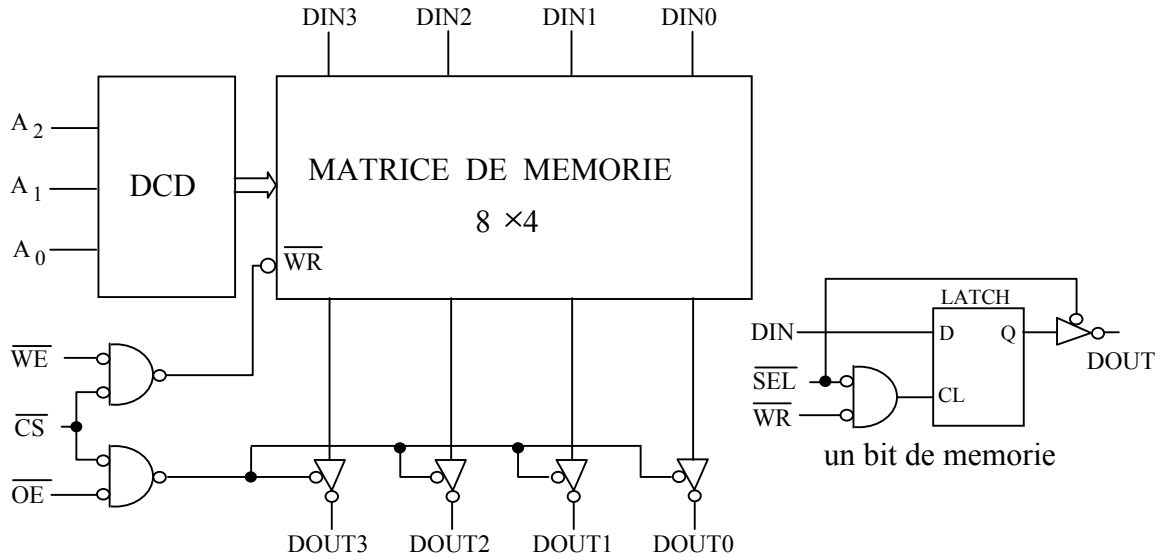


Fig. 8.21 *Structura unei memorii SRAM de 8 cuvinte de 4 biţi*

Structura unei memorii SRAM este asemănătoare cu cea a unei memorii ROM. Apare în plus semnalul $\overline{\text{WE}}$(**W**rite **E**nable) care, odată ce este activat pe 0 logic, memorează datele de pe intrările de date la adresa indicată de intrările de adresă. Se poate vedea în figura 8.21 că celula de memorie de un bit conţine un latch de tip D, iar memorarea datelor se face pe palierul de 1 logic al ceasului, adică când sunt activate semnalele $\overline{\text{WR}}$ şi $\overline{\text{SEL}}$, acesta din urmă fiind una din ieşirile decodificatorului liniilor de adresă. Activarea semnalului $\overline{\text{WR}}$ este o consecinţă a activării semnalelor de intrare $\overline{\text{WE}}$ şi $\overline{\text{CS}}$.

Ciclul de citire a datelor din memorie este identic cu cel de la memoria ROM, semnalul $\overline{\text{WE}}$ fiind bineînţeles dezactivat. Formele de undă pentru ciclul de scriere sunt date în figura 8.22. La activarea semnalului $\overline{\text{WE}}$ adresele trebuie să fie stabile cu cel puţin un timp $t_{address\ setup}$ înainte şi trebuie să se mai menţină stabile încă cel puţin un timp t_{hold} după dezactivarea lui $\overline{\text{WE}}$. Datele trebuie să fie stabile cu cel puţin un timp $t_{data\ setup}$ înainte de frontul crescător al semnalului $\overline{\text{WE}}$ şi trebuie să mai fie menţinute încă un timp minim $t_{data\ hold}$ după dezactivarea lui $\overline{\text{WE}}$. Se dă şi o durată minimă a pulsului de scriere $t_{write\ puls}$. Trebuie menţionat că se pot scrie date şi prin activarea temporară a semnalului $\overline{\text{CS}}$.

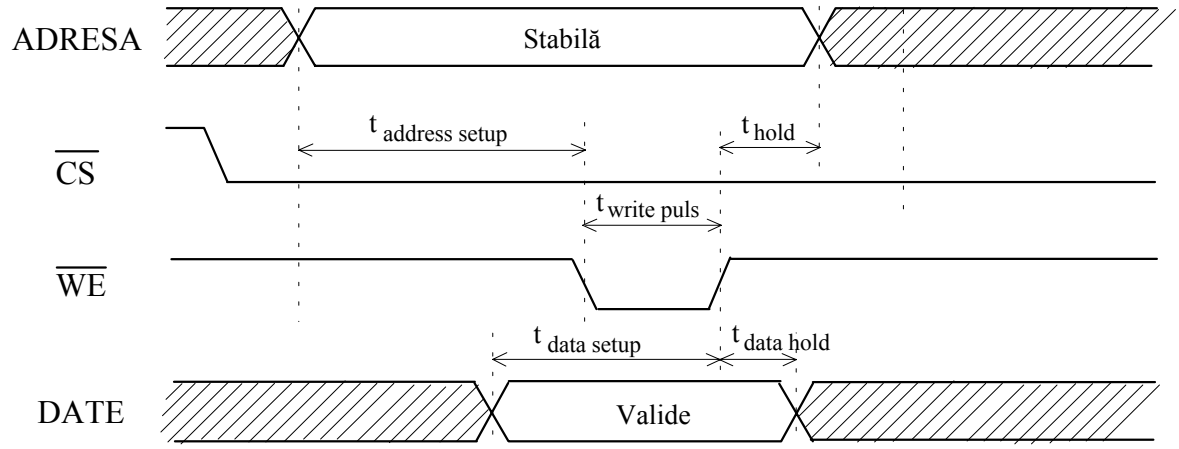


Fig. 8.22 *Forme de undă pentru ciclul de scriere a datelor la memoria SRAM*

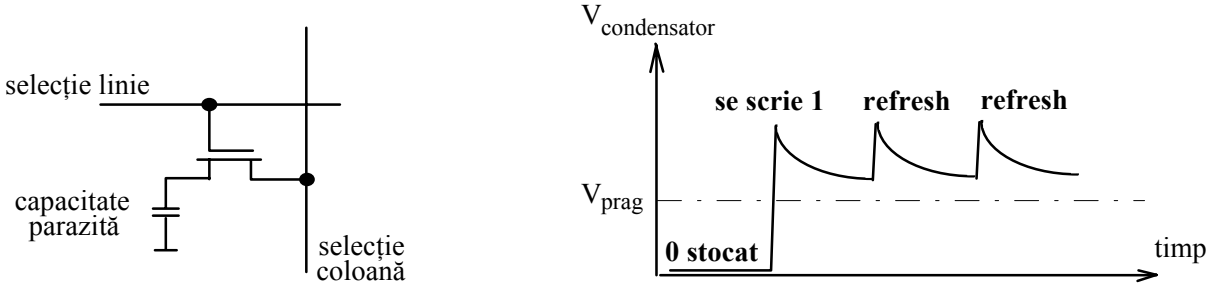


Fig. 8.23 *Celula de memorie DRAM de 1 bit*

Memoria DRAM a fost concepută pentru mărirea densității de integrare pe cip. Celula de memorie DRAM este dată în figura 8.23. Selecția celulei activează prin 1 logic "selecție linie", care deschide tranzistorul, iar capacitatea parazită se încarcă cu informația binară prezentă pe "selecție coloană". Dacă s-a memorat un bit de 0 logic, nu avem nici o problemă, dar dacă s-a memorat 1 logic, atunci informația trebuie reîmprospătată periodic prin **refresh**, altfel "condensatorul" se descarcă şi informația se pierde. Această reîmprospătare a informației memorate se face prin citirea periodică a tensiunii pe "condensator" şi refacerea ei prin reîncărcarea "condensatorului" la valoarea lui 1 logic, dacă se constată că tensiunea citită depăşeşte o anumită tensiune de prag. Un automat finit extern circuitului integrat generează aceste cicluri de refresh o dată la câteva milisecunde.

Memoriile DRAM au de obicei capacitate mare, deci multe linii de adresă. S-a constatat însă că citirea sau înscrierea datelor pe coloană se face după selecția liniei, deci biții mai semnificativi ai adresei pot fi generați cu o oarecare întârziere. De aici a apărut ideea multiplexării liniilor de adresă şi deci înjumătățirea lor la intrarea în circuitul integrat. Structura cipului de memorie DRAM este dată în figura 8.24.

Observăm că apar suplimentar două structuri noi de memorare de tip LATCH, comandate de două semnale suplimentare: $\overline{\text{RAS}}$ (**R**ow **A**ddress **S**trobe) şi $\overline{\text{CAS}}$ (**C**olumn **A**ddress **S**trobe). Matricea de memorie este întotdeauna pătrată, deci orice creştere de capacitate se face prin dublarea densității pe fiecare dimensiune din plan, noua capacitate de memorie fiind de 4 ori mai mare decât precedenta. Amplificatorul bidirecțional asigură generarea nivelului logic corespunzător, pornind de la tensiunea existentă pe "condensatorul" selectat, atât la operațiile de citire/scriere cât şi pentru generarea ciclului de refresh.

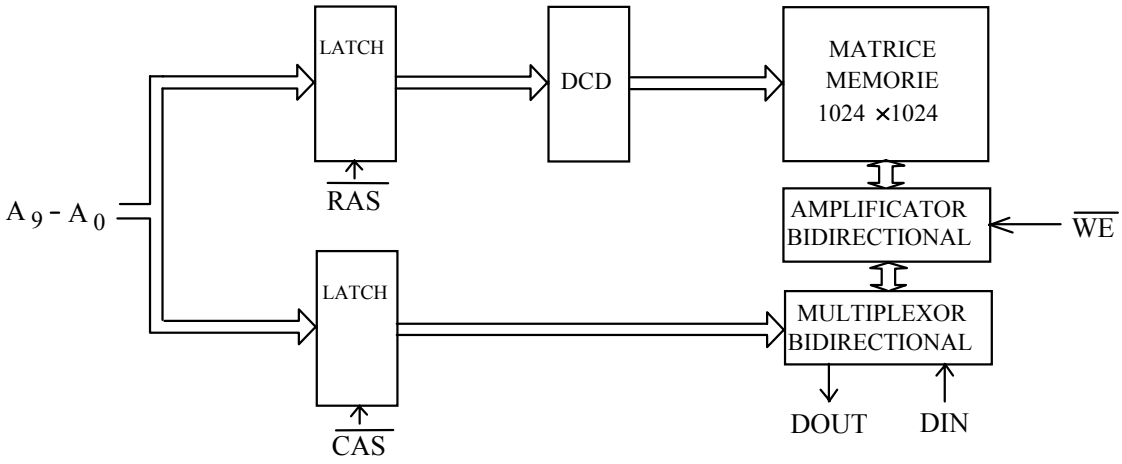


Fig. 8.24 *Structura unei memorii DRAM de 1M*

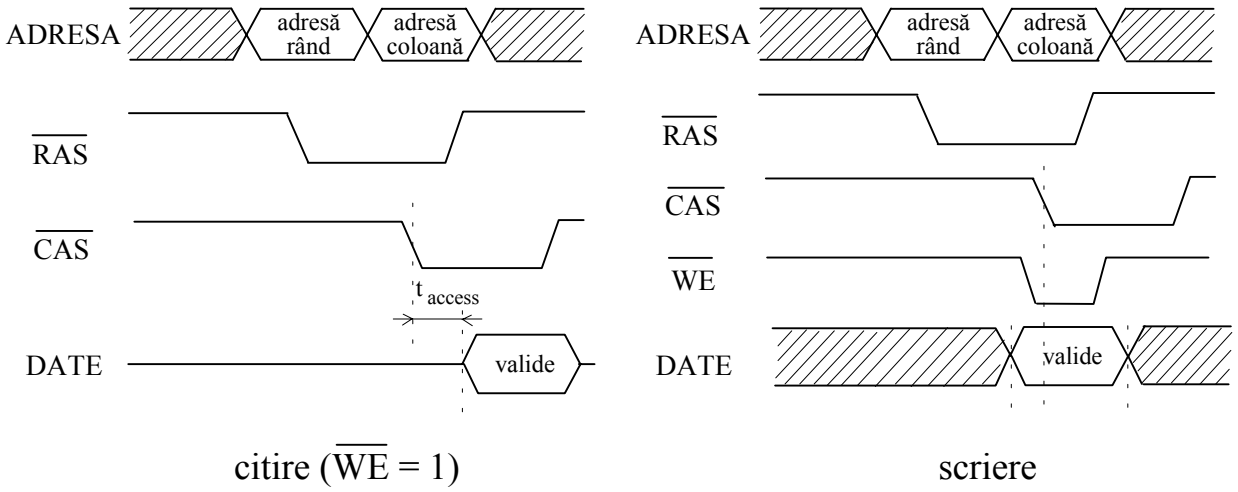


Fig. 8.25 *Forme de undă tipice pentru citirea/scrierea datelor*

Timing-ul, sau ordinea şi timpii de activare ai semnalelor, la bornele unui cip de memorie dinamică este restricționat de funcționarea relativ complexă pe care o are acest tip de circuit. Ciclurile de bază folosite la memoria DRAM sunt: ciclul de refresh, ciclul de citire şi ciclul de scriere.

În ciclul de refresh, sau reîmprospătare, după fixarea adresei de rând se activează semnalul $\overline{RAS}$. Conținutul liniei selectate din matricea de memorie este citit de amplificatorul bidirecțional şi reînscris în aceleaşi poziții. Semnalul $\overline{CAS}$ nu trebuie activat pentru că regenerarea locațiilor de memorie se face pe linie. Perioada maximă de refresh este o dată de catalog pentru fiecare cip de memorie. Dacă la cipurile DRAM de 64K era de cel mult 2ms, la memoriile de mare capacitate recente se constată creşterea acestui interval de timp (până la 64ms) şi implementarea unui algoritm special numit "hidden refresh". Acest ciclu presupune doar aplicarea semnalului $\overline{RAS}$, adresa fiind furnizată de un numărător intern care se autoincrementează pe fiecare front descrescător de $\overline{RAS}$.

Formele de undă tipice pentru ciclul de citire sunt date în figura 8.25. După fixarea adresei de linie se activează semnalul $\overline{RAS}$, moment în care adresa de linie este stocată în latch-ul corespunzător. Se aplică pe urmă adresa de coloană şi se activează semnalul $\overline{CAS}$. Pe frontul descrescător al semnalului $\overline{CAS}$ adresa de coloană este stocată în latch-ul corespunzător. Datele devin disponibile după un timp de la activarea semnalului $\overline{CAS}$. Acest timp este timpul de acces la memorie. Pentru preluarea datelor din memorie se poate folosi frontul crescător al semnalului $\overline{CAS}$. Pe toată durata ciclului $\overline{WE}$ rămâne dezactivat.

Pentru un ciclu de scriere, adresele şi semnalele $\overline{RAS}$ şi $\overline{CAS}$ se aplică la fel ca şi pentru un ciclu de citire, dar semnalul $\overline{WE}$ trebuie să fie activat înainte de frontul descrescător al semnalului $\overline{CAS}$. Intrările de date trebuie să fie şi ele stabile cu un timp minim înainte de activarea semnalului $\overline{CAS}$ şi încă un timp minim după aceea. Toți aceşti timpi sunt date importante de catalog pentru fiecare cip de memorie în parte.

Mai există şi alte cicluri care eficientizează lucrul cu memoria DRAM: ciclul de citire-modificare-scriere, folosit pentru citirea datelor, prelucrarea lor şi memorarea rezultatului prelucrării la aceeaşi adresă, ciclul de citire în mod pagină, care după activarea semnalului $\overline{RAS}$ activează în mod repetat semnalul $\overline{CAS}$ şi permite citirea mai rapidă a informației coloană cu coloană, sau ciclul de citire în mod nibble, asemănător cu cel în mod pagină, numai că acum generarea unor secvențe de 4 impulsuri de $\overline{CAS}$ se face automat.

Probleme

8.1 Să se implementeze o memorie ROM de 128 cuvinte a câte 8 biți, folosind cipuri de 256 biți, organizați în 32 cuvinte a câte 8 biți.

8.2 Să se implementeze o memorie ROM de 64 cuvinte a câte 16 biți, folosind cipuri de 32 cuvinte a câte 8 biți.

8.3 Să se implementeze cu memorie ROM şi apoi cu o structură PLA următorul set de funcții binare:

$$f_1 = P_1 + P_3 + P_4 + P_8 + P_{12} + P_{14} + P_{15}$$
$$f_2 = P_0 + P_1 + P_5 + P_8 + P_{12}$$
$$f_3 = P_2 + P_4 + P_5 + P_8 + P_{13} + P_{15}$$

([Friedman, 1986])

8.4 Să se implementeze cu memorie ROM un generator al funcției *sinus*, ştiind că argumentul funcției variază între 0 şi π/2 cu paşi de π/512. Se cere rezultatul cu 4 zecimale exacte în cod BCD.

8.5 Să se realizeze o memorie EPROM de 64 Kbiți ($64K \times 1$), folosind o memorie EPROM de 8 Kocteți ($8K \times 8$) şi un multiplexor cu 3 intrări de selecție.

([Mureşan, 1996])

8.6 Folosind o memorie EPROM să se implementeze un circuit care realizează înmulțirea $M \times N = P$, unde M, N şi P sunt exprimate în cod BCD, iar M şi N sunt cuprinse între 0 şi 9.

([Mureşan, 1996])

8.7 Să se implementeze o memorie EPROM de $2K \times 32$, folosind o memorie EPROM de $64K \times 1$. Se pot utiliza circuite auxiliare SSI/MSI şi se presupune că dispunem de un semnal de ceas cu o perioadă ceva mai lungă decât timpul de acces la memoria de $64K \times 1$. Care este timpul de acces al memoriei de $2K \times 32$?

([Wakerly, 1990])

8.8 Să se explice conexiunile programate la implementarea într-un PAL a funcției SAU-EXCLUSIV negat.

([Mureşan, 1996])

8.9 Să se proiecteze un numărător binar reversibil pentru controlerul unui lift într-o clădire cu 20 etaje, folosind un singur circuit PAL16R6. Numărătorul trebuie să aibă o intrare care permite numărarea şi o intrare care stabileşte sensul de numărare. La numărarea înapoi trebuie să se blocheze în starea 1, iar la numărarea înainte trebuie să se blocheze în starea 20. În orice sens de numărare se sare peste starea 13. Reprezentați diagrama stărilor şi ecuațiile scrise în ABEL.

([Wakerly, 1990])

8.10 Să se proiecteze un divizor cu trei sincron, implementabil într-un PLD.
([Mureşan, 1996])

8.11 Folosind memorii SRAM de $8K \times 8$ biţi să se realizeze în două variante, o memorie de $16K \times 12$ biţi.
([Mureşan, 1996])

8.12 Să se proiecteze o schemă de adresare şi comandă a reîmprospătării pentru o memorie DRAM de $64K \times 1$ bit.
([Mureşan, 1996])

8.13 Un sistem de prelucrare şi afişare a imaginilor posedă o memorie video de $64K \times 8$ biţi realizată cu memorii DRAM. Indicaţi modul în care este cel mai convenabil să se execute adresarea memoriei DRAM astfel încât să nu mai fie necesară reîmprospătarea memoriei. Se ştie că imaginea conţine 256×256 pixeli cu 256 nivele de gri.
([Mureşan, 1996])

8.14 Dispunem de memorii ROM de 256 cuvinte de 8 biţi astfel programate încât realizează funcţia unui înmulţitor de două numere reprezentate pe câte patru biţi. Utilizând circuite adiţionale (pentru sumare), să se construiască un înmulţitor de cuvinte de 8 biţi.
([Ştefan, 1992])

8.15 Să se proiecteze un automat finit destinat semaforizării unei intersecţii simple folosind un circuit programabil PAL16R4.

8.16 Explicaţi dacă este posibil ca schema logică a unui automat finit implementat cu circuite SSI/MSI să fie diferită de cea obţinută prin implementarea cu circuite FPGA. Care sunt criteriile de proiectare în fiecare din cele două cazuri?

8.17 Faceţi o comparaţie între cele două soluţii posibile de proiectare a unui sistem numeric: soluţia cablată, care foloseşte circuite SSI/MSI, şi soluţia programată, care foloseşte circuite FPGA. Analizaţi cele două soluţii din punctul de vedere al gabaritului, consumului, fiabilităţii, posibilităţilor de modificare ulterioară, timpului de execuţie, costului.

8.18 Arătaţi care este importanţa posibilităţii de reconfigurare de un număr foarte mare de ori a unei structuri FPGA. Ce aplicaţii posibile se întrevăd în viitorul apropiat datorită acestor tehnologii excepţionale?