6 ANALIZA ŞI SINTEZA AUTOMATĂ

Aplicațiile din acest capitol îşi propun să prezinte posibilități de analiză şi sinteză a sistemelor numerice folosind programe de calculator concepute în acest scop. Un standard pentru analiza circuitelor prin simulare pe calculator, fie ele analogice sau numerice, este programul **SPICE(Simulation Program with Integrated Circuit Emphasis)**, elaborat la Universitatea Berkeley din California şi perfecționat pe parcursul câtorva decenii. Limbajul **VHDL(Very High Speed Integrated Circuit Hardware Description Language)**, folosit astăzi pe scară largă de proiectanții de sisteme numerice, este destinat sintezei structurilor numerice de mare complexitate, fiind standardizat de IEEE în 1993.

6.1 Considerații teoretice

6.1.1 Analiza circuitelor prin simulare PSPICE

Pachetul de programe SPICE nu a fost inițial conceput pentru calculatoare personale. Odată cu apariția PC-urilor, au apărut şi programe de analiză a circuitelor pe PC, similare cu programul SPICE, cunoscute de obicei sub numele **PSPICE** (**PC SPICE**).

Programul PSPICE folosit este **Design Center 5.2**, un mediu integrat sub Windows 3.X, produs de firma americană **MicroSim**. Acest mediu conține subprograme de *editare*, *analiză* şi *prezentare* a rezultatelor obținute. Cea mai importantă parte a programului de simulare o reprezintă subprogramul de analiză, care execută analizele de circuit specificate în fişierul editat, ieşirile din acest subprogram furnizând date pentru a fi utilizate ulterior de subprogramul de prezentare a rezultatelor, care materializează rezultatele sub formă de grafice şi texte. Subprogramul de analiză conține procedeele numerice ale reprezentării matematice a circuitului. Pentru a trece de la circuitul propriu-zis la un sistem matematic de ecuații, elementele de circuit (rezistoare, condensatoare, surse, diode, tranzistoare, porți logice, bistabile, registre etc.) sunt reprezentate prin *modele matematice*. Sistemul de ecuații care descrie întregul circuit este determinat de ecuațiile modelului fiecărui element şi

relaţiile topologice care sunt date de interconectarea elementelor. Relaţiile topologice au la bază legile lui Kirchhoff iar comportarea generală a circuitului este descrisă printr-un sistem de ecuaţii diferenţiale, ale cărui soluţii se obţin prin analiza circuitului, pentru diferite cazuri particulare de abordare: analiza de curent continuu (**.DC**), analiza de curent alternativ (**.AC**), analiza regimurilor tranzitorii (**.TRAN**) şi altele. Aceste analize se realizează pe baza unor metode numerice, care presupun formularea ecuaţiilor, rezolvarea ecuaţiilor liniare, a ecuaţiilor neliniare şi integrarea numerică.

Pe măsură ce creşte experienţa lucrului cu PSPICE se pun tot mai clar în evidenţă avantajele simulării. În afară de faptul că simularea este mult mai ieftină decât realizarea experimentală a circuitului, ea permite efectuarea unor analize imposibil de realizat pe model experimental: cum am putea măsura de exemplu tensiunea într-un nod din interiorul unui circuit integrat sau comportarea unui tranzistor la temperatura de 120 grade Celsius?

Deşi primele variante ale programelor PSPICE au fost concepute în exclusivitate pentru analiza circuitelor analogice, Design Center 5.2 şi variantele ulterioare permit şi analiza circuitelor numerice. Opţiunea **DIGITAL SIMULATION** permite modelarea comportării unui număr mare de dispozitive numerice (porţi, bistabile, registre, numărătoare, dispozitive logice programabile etc.), numite primitive. Aceste primitive sunt folosite de o bibliotecă numerică **DIGITAL.LIB** pentru a modela un număr mare de componente care pot fi introduse direct în circuit printr-un apel de subcircuit.

PSPICE recunoaşte trei tipuri de noduri: analogice, numerice şi de interfaţă. Dacă la un nod sunt conectate numai dispozitive analogice, atunci el este analogic. Dacă sunt conectate numai dispozitive digitale, atunci el este digital. Dacă la nod sunt conectate atât dispozitive analogice cât şi digitale, atunci avem nod de interfaţă. PSPICE separă automat nodurile de interfaţă în analogice şi numerice, inserând una sau mai multe circuite de interfaţă analog/numerice.

Nivelele logice utilizate în PSPICE nu trebuie să fie neapărat tensiuni. Ele sunt:

0	LOW
1	HIGH
R	RISE (crescător, de la 0 la 1)
F	FALL (descrescător, de la 1 la 0)
X	necunoscut

În situaţia conectării mai multor ieşiri logice împreună, pentru determinarea nivelului logic corect al nodului, s-a asociat fiecărei ieşiri câte o intensitate, a cărei valoare este determinată în raport cu intensităţile celorlalte ieşiri. Nodurile conduse de ieşiri cu aceeaşi intensitate, dar de nivele diferite, vor avea nivelul logic X. Pspice are 64 de nivele de intensitate, cea mai slabă fiind Z (înaltă impedanţă), iar cea mai puternică, intensitatea de forţare (scurtcircuitul). Aceste nivele se fixează prin parametrii **DIGDRVZ** şi **DIGDRVF** ai comenzii **OPTIONS**.

Programul de simulare numerică PSPICE defineşte trei noduri numerice globale, având următoarele nume şi valori:

$D_HI	1
$D_LO	0
$D_X	X

Ele sunt folosite pentru a menţine un pin al unui dispozitiv sau subcircuit, la nivelul logic dorit, pe tot parcursul simulării. La aceste noduri nu se conectează dispozitive analogice. În simularea numerică, tot nod global este considerat şi nodul analogic de masă (potenţial nul sau nod logic 0).

Pentru a furniza semnalele de intrare necesare simulării funcționării unui circuit numeric, programul PSPICE pune la dispoziția utilizatorului două tipuri de dispozitive. Primul dispozitiv este un generator de impuls, care permite obținerea unei game largi de semnale numerice, în mod asemănător cu cele generate la simularea circuitelor analogice. Al doilea dispozitiv este un fișier de impulsuri, care permite obținerea unui număr oricât de mare de forme de undă dintr-un fișier extern. În aplicațiile noastre vom utiliza un fișier de impulsuri pentru generarea semnalelor de intrare.

Dacă circuitul supus analizei conține atât dispozitive analogice cât şi numerice, cum este cazul oscilatorului din figura 6.1, se face o simulare mixtă analog/numerică. Figura 6.2 arată formele de undă rezultate în urma simulării. În partea de sus sunt reprezentate formele de undă numerice, iar în cea de jos tensiunile în diferite noduri importante ale circuitului. Semnalul RESET este aplicat prin U4 la intrarea CLEAR a bistabilului JK, iar prin inversorul cu colector în gol U3 la nodul 1 al oscilatorului, pentru stabilirea condițiilor inițiale de simulare.

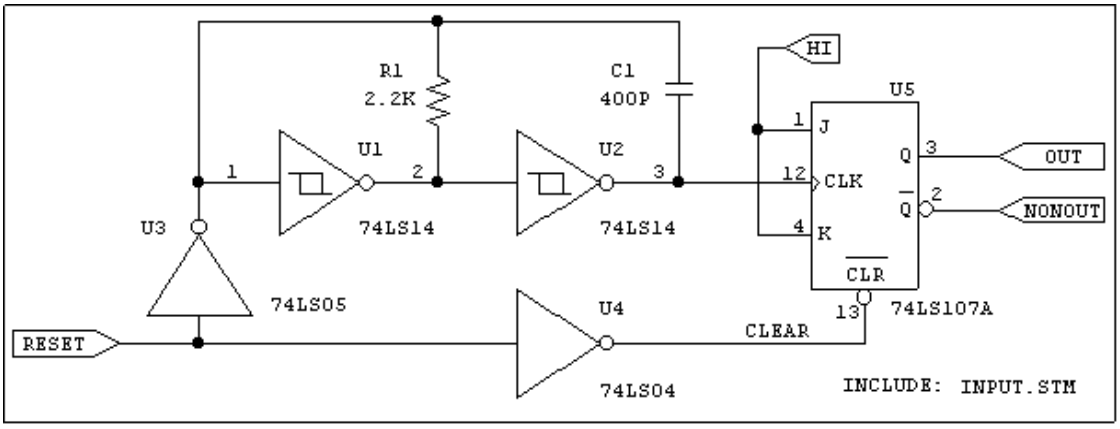

Fig. 6.1 *Schemă electrică editată în Design Center 5.2*

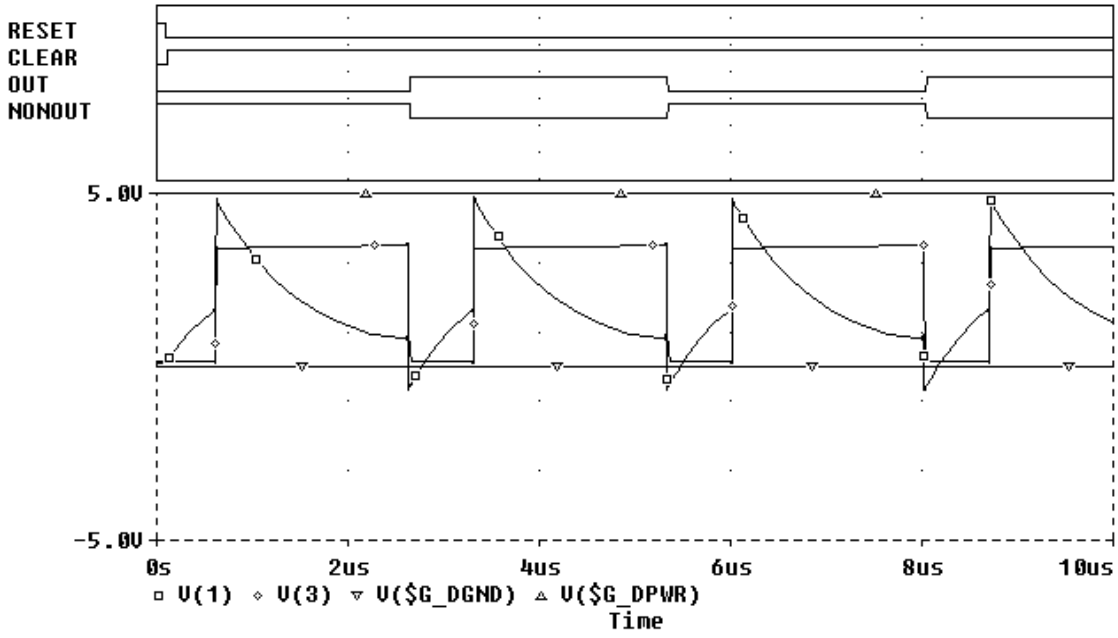

Fig. 6.2 *Formele de undă rezultate în urma simulării*

```
URESET STIM(1,1) $G_DPWR $G_DGND
+       RESET
+       IO_STM
+       TIMESTEP=10n
+       0c      1
+       10c     0
```

Fig. 6.3 *Fişierul text INPUT.STM*

Figura 6.3 prezintă fişierul text pentru generarea semnalului RESET. Numele fişierului este INPUT.STM şi este introdus în schema electrică prin directiva INCLUDE (vezi figura 6.1). Circuitul realizat fizic funcţionează şi fără semnal de reset, dar simulatorul are nevoie de stări iniţiale, înainte de a începe analiza. În lipsa semnalului de reset, se porneşte de la o stare logică necunoscută şi analiza nu se poate face.

Fişierul text INPUT.STM se editează cu un simplu editor de text, de exemplu NOTEPAD. Formatul generatorului de impuls pentru un semnal este următorul:

U<nume> **STIM**(<lăţime>,<format>)
+ <nod alimentare numerică> <nod masă numerică>
+ <nod>
+ <nume (Model de I/E)>
+ [**IO_LEVEL** = <valoare (selectare subcircuit interfaţă)>]
+ [**TIMESTEP** = <pas>]
+ <(comandă)>

unde <lăţime> specifică numărul semnalelor de ieşire furnizate de generator, iar <format> specifică formatul valorilor utilizate în definirea impulsului. <nume (Model de I/E)> este de cele mai multe ori modelul IO_STM. IO_LEVEL este un parametru opţional prin care se pot selecta subcircuitele interfaţă numeric analogice; el a fost fixat înainte de analiză la valoarea recomandată 3 (prin comanda **ANALYSIS**, urmată de **SETUP**, şi **OPTIONS** din meniul principal, s-a fixat parametrul **DIGIOLVL** = 3). TIMESTEP reprezintă timpul pe un ciclu de tact sau un pas, iar <comandă> este o descriere a semnalului prin valorile logice ale acestuia la valori ale timpului care sunt multipli de TIMESTEP.

După desenarea schemei electrice a circuitului, se selectează **ANALYSIS** din meniul principal şi apoi **ANNOTATE** (numai dacă se doreşte o renumerotare a componentelor din circuit) şi **ELECTRICAL RULE CHECK**. Dacă există erori în schemă, programul anunţă acest lucru înainte de a merge mai departe. Dacă nu există erori, se lansează **CREATE NETLIST** şi pe urmă se alege tipul de analiză dorit. Acest lucru se face prin comanda **ANALYSIS**, urmată de **SETUP** şi **TRANSIENT**. Pentru analiza tranzitorie s-a fixat o durată totală de 10 μsec, cu un pas de 0,1 μsec. Nu uitaţi selectarea opţiunii **ENABLED** înainte de a părăsi fereastra de setare a tipului de analiză.

Urmează rularea programului de analiză, cu comanda **RUN PSPICE**. La terminarea rulării, care poate dura de la fracţiuni de secundă la zeci sau sute de secunde, funcţie de complexitatea circuitului, tipul de analiză, numărul de paşi ales şi viteza sistemului de calcul, apare ecranul mediului **PROBE** cu un alt meniu specific. Comanda **TRACE**, urmată de **ADD** afişează o fereastră cu toate semnalele din circuit. Se selectează semnalele pe care vrem să le vizualizăm şi obţinem formele de undă în timp, ca cele din figura 6.2.

Ne propunem în continuare să proiectăm o interfață calculator-microcalculator cu 4 intrări şi cu 4 ieşiri, sincronă cu ceasul microcalculatorului. Semnalele sunt reprezentate în figura 6.4, iar organigrama de funcționare a circuitului, în figura 6.5. Vom verifica dacă sinteza este corectă, făcând analiza circuitului rezultat prin simulare PSPICE.

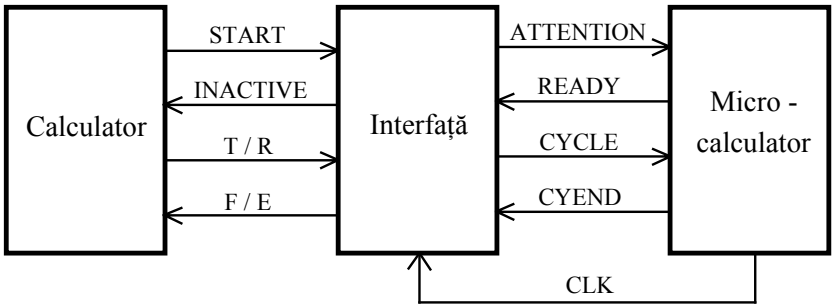

Fig. 6.4 *Semnalele interfeței calculator-microcalculator*

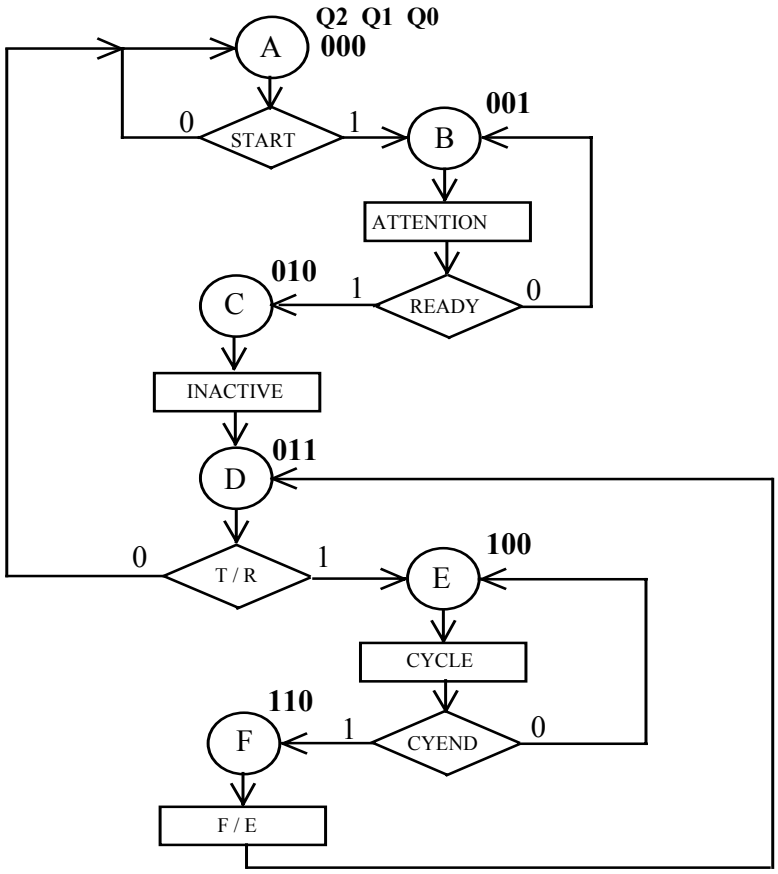

Fig. 6.5 *Organigrama interfeței*

Cele 6 stări ale automatului finit sunt codificate binar minimal folosind 3 biți. Semnalele de ieşire sunt generate pe stare, iar codurile stărilor succesoare stărilor în care sunt testate intrările asincrone START şi T/R sunt adiacente. Vom implementa circuitul într-o primă variantă folosind 3 bistabile de tip JK şi porți logice, iar pe urmă vom face o implementare cu un singur circuit PLD de tip PAL. Ne ocupăm în această secțiune numai de sinteza schemelor logice, iar analiza schemelor electrice, prin simulare PSPICE, va fi făcută în secțiunea 6.2.

Start	Ready	T/R	Cyend	Q2	Q1	Q0	Q2+	Q1+	Q0+	Attention	Inactive	Cycle	F/E
0	x	x	x	0	0	0	0	0	0	0	0	0	0
1	x	x	x	0	0	0	0	0	1	0	0	0	0
x	0	x	x	0	0	1	0	0	1	1	0	0	0
x	1	x	x	0	0	1	0	1	0	1	0	0	0
x	x	x	x	0	1	0	0	1	1	0	1	0	0
x	x	0	x	0	1	1	0	0	0	0	0	0	0
x	x	1	x	0	1	1	1	0	0	0	0	0	0
x	x	x	0	1	0	0	1	0	0	0	0	1	0
x	x	x	1	1	0	0	1	1	0	0	0	1	0
x	x	x	x	1	1	0	0	1	1	0	0	0	1

Fig. 6.6 *Tabelul tranziţiilor şi al ieşirilor*

Tabelul tranziţiilor şi al ieşirilor este dat în figura 6.6. Dacă construim în continuare coloanele funcţiilor de excitaţie pentru implementarea cu bistabile JK şi minimizăm aceste funcţii folosind diagrame Veitch-Karnaugh, obţinem următoarele ecuaţii:

$$J_2 = T/R \cdot Q_1 \cdot Q_2 \qquad \text{şi} \qquad K_2 = Q_1$$

$$J_1 = READY \cdot Q_0 + CYEND \cdot Q_2 \qquad \text{şi} \qquad K_1 = Q_0$$

$$J_0 = \overline{Q_2} \cdot START + Q_1 \qquad \text{şi} \qquad K_0 = READY + Q_1$$

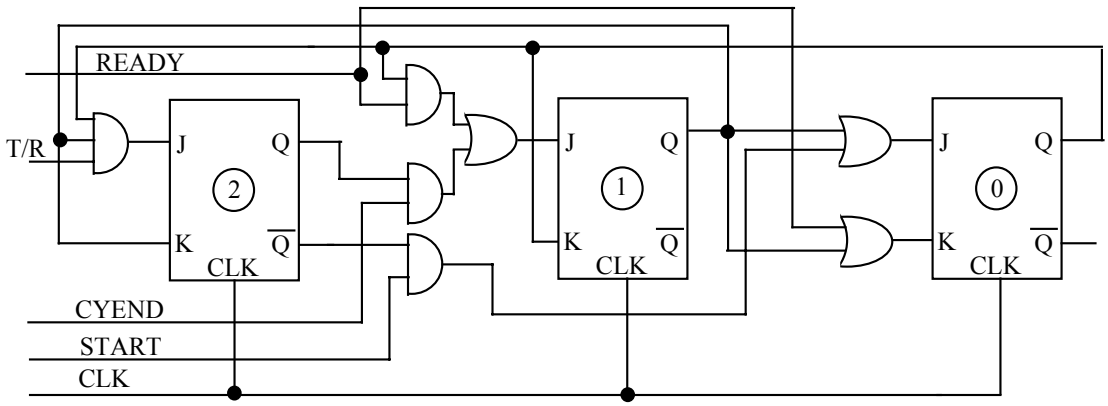

Fig. 6.7 *Schema logică a interfeţei implementate cu bistabile JK şi porţi*

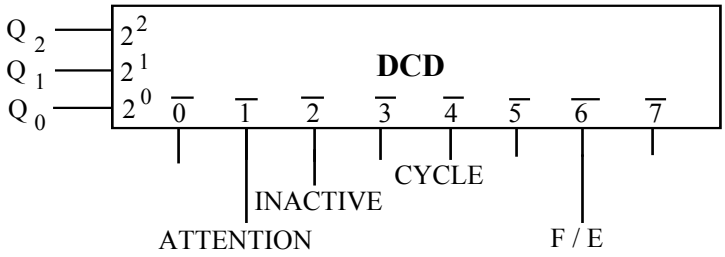

Fig. 6.8 *O soluţie posibilă pentru sinteza ieşirilor*

Folosind ecuațiile de mai sus rezultă schema logică a circuitului, reprezentată în figura 6.7, iar schema din figura 6.8 oferă o soluție de implementare cu decodificator a funcțiilor de ieşire, fără a mai fi necesară minimizarea lor.

Pentru a face sinteza cu PLD a circuitului propus mai sus, trebuie să alegem o structură de circuit din catalog. Circuitul **PAL16R4** de la AMD are foaia de catalog dată în figura 6.18. Observăm că ieşirile bistabilelor sunt trecute prin inversoare cu 3 stări înainte de a ajunge la pinii de ieşire ai circuitului. Atribuim ieşirea O6 lui Q2, O5 lui Q1 şi O4 lui Q0. Aria combinațională implementează funcțiile D2, D1 şi D0 cu porți ŞI-SAU şi conține 2048 de fuzibile ce pot fi arse o singură dată. Ecuațiile lor, deduse cu ajutorul tabelului din figura 6.6, sunt:

$$\overline{Q_2^+} = \overline{Q_2} \cdot \overline{Q_1} + Q_1 \cdot \overline{Q_0} + \overline{T/R} \cdot Q_1$$

$$\overline{Q_1^+} = \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} + Q_1 \cdot Q_0 + \overline{READY} \cdot Q_0 + \overline{CYEND} \cdot Q_2 \cdot \overline{Q_1}$$

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

Celelalte 4 semnale de ieşire ATTENTION, INACTIVE, CYCLE şi F/E sunt atribuite ieşirilor I/O8, I/O7, I/O2 şi respectiv I/O1, 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 I1, I2, I3 şi respectiv I4, iar pe I5 se aplică intrarea RESET, absolut necesară şi aici, din motivul inițializării bistabilelor la începutul simulării.

6.1.2 Sinteza circuitelor folosind limbajul VHDL

Limbajul VHDL este cel mai cunoscut şi cel mai puternic limbaj de descriere hardware a circuitelor. Pe lângă modelarea şi simularea sistemelor numerice, el permite sinteza structurilor numerice la orice nivel, de la structuri alcătuite din câteva porți logice până la un sistem complet cu microprocesor, de exemplu.

Proiectele pot fi descompuse ierarhic, iar VHDL oferă un cadru de lucru de bună calitate pentru definirea modulelor şi a interfețelor lor, precum şi pentru completarea ulterioară a detaliilor. După scrierea propriu-zisă a codului VHDL pentru fiecare dintre elementele menționate mai sus, se compilează proiectul, iar dacă nu avem erori, se trece la etapa de simulare. De fapt, simularea este doar un fragment al unei etape mai ample, numită verificare. Este vorba de o verificare funcțională, în care se verifică logica circuitului, fără a ține seamă de aspectele de temporizare (întârzierile introduse de porți se consideră nule), urmată de o verificare temporală, care are un caracter preliminar. După verificare se trece la stadiul de finalizare a proiectului. Descrierea VHDL se transpune într-un set de primitive ce pot fi asamblate în tehnologia propusă. Aceste primitive se aplică resurselor de dispozitive disponibile, folosind un instrument de aplicare, iar în final, se face o verificare temporală finală a circuitului rezultat după aplicare.

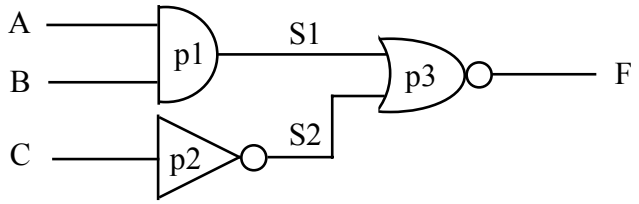

Fig. 6.9 *Schema logică a unei structuri combinaţionale*

Pentru a introduce câteva dintre conceptele limbajului VHDL vom considera circuitul foarte simplu din figura 6.9. Pentru verificarea codului creat şi simularea circuitului am folosit versiunea freeware a programului **VHDL Simili 2.2**, realizat de firma **Symphony EDA.** Această versiune are unele limitări funcţionale şi poate fi folosită timp de câteva luni. La depăşirea acestui timp, se poate descărca din Internet o versiune actualizată freeware a programului.

Un cod posibil VHDL pentru circuitul din figura 6.9 este dat în figura 6.10. După declararea bibliotecilor folosite, se declară *entitatea* FUNC şi apoi *arhitectura* asociată entităţii. În exemplul din figura 6.10 s-a făcut o descriere *comportamentală* a arhitecturii circuitului, în care ansamblul celor 3 porţi logice este văzut ca o *componentă*, numită GATES, căreia i se specifică intrările şi ieşirile, iar relaţia dintre ele este dată de ecuaţia F <= not((A and B) or (not C)). Mai există o declaraţie a semnalelor A, B, C şi F, precum şi variaţiile lor în vederea simulării. În final aceste semnale sunt *mapate* pe intrările şi ieşirile componentei GATES, în ordinea în care au fost declarate.

Componenta **Sonata** din **VHDL Simili 2.2** oferă un mediu IDE (**I**ntegrated **D**evelopment **E**nvironment) prietenos. Se crează de la început un nou proiect prin **File** şi **NewWorkspace**. Din acest moment toate acţiunile din meniul principal devin posibile şi se

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity FUNC is
end;

architecture VAR1 of FUNC is

        component GATES
                port (A, B, C: in STD_LOGIC;
                        F: out STD_LOGIC);
        end component;

        signal A, B, C, F: STD_LOGIC;

        begin

                A <= '0', '1' after 100 NS, '0' after 300 NS;
                B <= '0', '1' after 200 NS, '0' after 400 NS;
                C <= '1', '0' after 350 NS;
                F <= not((A and B) or (not C));

                M: GATES port map (A, B, C, F);

         end VAR1;
```

Fig. 6.10 *Codul VHDL pentru descrierea şi simularea circuitului din figura 6.9*

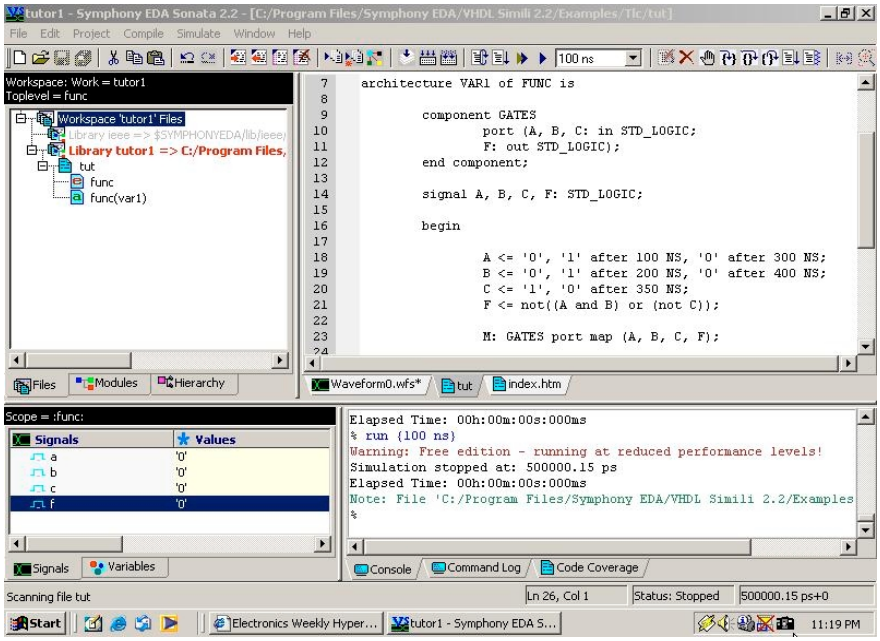

Fig. 6.11 *Mediul Sonata şi câteva ferestre reprezentative*

crează două fişiere cu extensiile .sws şi .sym. Se asociază un set de fişiere VHDL cu biblioteca curentă (fişierul sursă tut.vhd în exemplul nostru), iar modulele *entitate* şi *arhitectură* apar ca subcomponente ale fişierului tut.vhd în workspace. În figura 6.11 sunt prezentate patru ferestre reprezentative ale mediului Sonata: workspace, editorul de text, lista semnalelor vizualizate şi fereastra de consolă, care informează în permanență programatorul asupra acțiunilor desfăşurate în mediul prezentat.

Se face compilarea setului de fişiere ataşat, sau a fişierului în exemplul nostru, şi dacă nu sunt semnalate erori în fereastra de consolă, se poate trece la faza de simulare. Semnalele declarate sunt prezente în fereastra **Scope** şi ele pot fi trimise în fereastra **Signals** prin procedeul **Drag and Drop.**

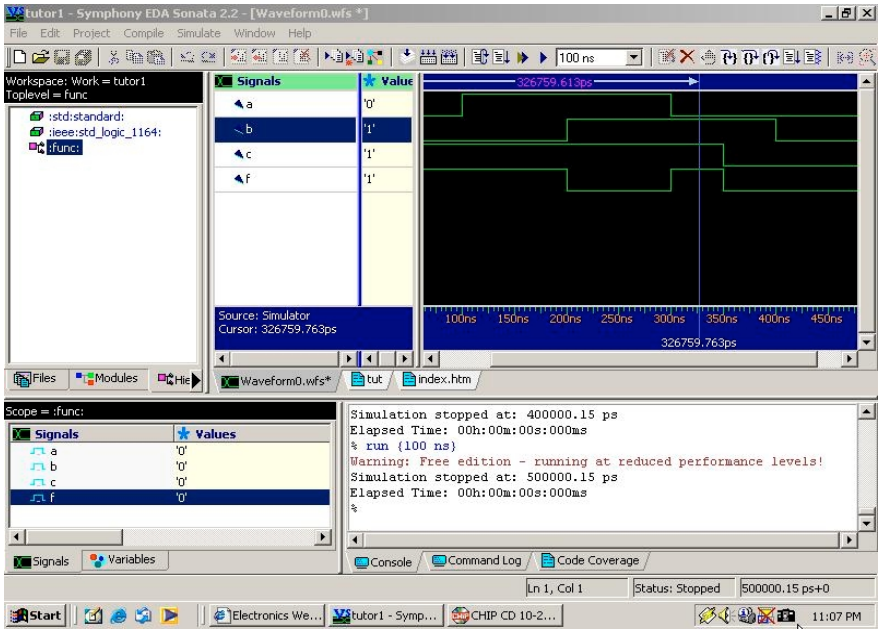

Fig. 6.12 *Mediul Sonata şi fereastra Waveforms*

Simularea se poate face fie în regim continuu, până la valoarea de timp stabilită de proiectant, fie în paşi care au o anumită durată ce poate fi selectată după dorință. Formele de undă din figura 6.12 (dreapta, sus) arată că funcționarea circuitului este corectă. Este adevărat că în această simulare s-au neglijat timpii de propagare prin porți.

Porțiunea de cod din figura 6.13 generează o *arhitectură structurală* a circuitului, făcând o descriere la nivel de componente şi interconexiuni. Circuitul este descris ca o interconectare a unor blocuri sau componente disponibile (porți logice, multiplexoare, memorii etc.). În acest context clasa **signal** va putea fi interpretată ca interconexiune. Descrierea structurală este echivalentă schemei electrice. Fiecare poartă logică este introdusă prin directiva *component*, se declară semnalele de intrare/ieşire A, B, C, F, dar şi semnalele interne S1 şi S2 (vezi figura 6.9). Se precizează apoi variația semnalelor de intrare, iar semnalele de intrare/ieşire pentru fiecare componentă de circuit sunt *mapate* pe intrările/ieşirile fiecărei componente, conform schemei electrice a circuitului.

Declarațiile de tip de componente sunt însă fără conținut deoarece lipsesc informațiile despre obiectele la care se referă. Aceste declarații asigură denumiri formale proprii fiecărui fişier sursă. Atribuirea conținutului se face prin asocierea numelor componentelor declarate cu entitățile utilizate pentru fiecare componentă. În acest fel se păstrează rezultatele anterioare la eventuale modificări de context, asigurându-se modularitatea proiectării, un mare avantaj al limbajului VHDL ([Burdia,1999]). Alte avantaje remarcabile ale limbajului VHDL sunt portabilitatea, independența proiectării de tehnologia de integrare şi simularea comportamentală a circuitelor.

```
architecture VAR2 of FUNC is

        component and2gate
              port (A, B: in STD_LOGIC;
                    F: out STD_LOGIC);
        end component;

        component invgate
              port (A: in STD_LOGIC;
                    F: out STD_LOGIC);
        end component;

        component nor2gate
              port (A, B: in STD_LOGIC;
                    F: out STD_LOGIC);
        end component;

        signal A, B, C, F: STD_LOGIC;
        signal S1, S2: STD_LOGIC;

        begin

              A <= '0', '1' after 100 NS, '0' after 300 NS;
              B <= '0', '1' after 200 NS, '0' after 400 NS;
              C <= '1', '0' after 350 NS;

              p1: and2gate port map(A, B, S1);
              p2: invgate port map(C, S2);
              p3: nor2gate port map(S1, S2, F);

        end VAR2;
```

Fig. 6.13 *O altă descriere VHDL pentru circuitul din figura 6.9*

6.2 Demonstrații practice

6.2.1 Se lansează în execuție editorul grafic **Schematics** din mediul **Design Center 5.2**. Se desenează schema oscilatorului din figura 6.1 şi se salvează într-un fişier cu extensia .sch. Se editează în NOTEPAD fişierul text **Input.stm**, dat în figura 6.3, şi se salvează în acelaşi director cu fişierul sursă *.sch. Se alege analiza **.TRAN** pe o durată de 10 μs, folosind un pas de 100ns, şi, dacă nu sunt erori, apare mediul **PROBE** din care se aleg formele de undă care ne interesează. Se verifică dacă ele corespund cu cele din figura 6.2 şi se repetă simularea cu modificarea unor parametri (constanta de timp RC, durata simulării, pasul simulării etc.). □

6.2.2 Se desenează în editorul grafic **Schematics**, pe aceeaşi foaie de lucru, schemele din figurile 6.7 şi 6.8. Se editează în NOTEPAD un fişier de stimuli după modelul din figura 6.14. Acest fişier, care are extensia *.stm, se salvează în acelaşi director cu fişierul sursă *.sch şi se include în desen prin directiva INCLUDE. Se face analiza **.TRAN** şi se confruntă rezultatul obținut cu cel din figura 6.15. Observați modul în care este generat semnalul de ceas în fişierul de stimuli. Observați modul în care sunt generate celelalte semnale de intrare. Explicați de ce s-au luat paşi diferiți pe scara timpului pentru semnalele **START şi T/R**. Explicați de ce nivelele logice ale ieşirilor nu sunt cunoscute în primele momente de timp de la începutul simulării şi arătați care este rolul semnalului **RESET**, semnal care nu apare în organigramă. Verificați dacă formele de undă reprezintă absolut toate tranzițiile posibile din organigramă şi, în caz contrar, construiți un alt fişier de stimuli care să evidențieze şi acele tranziții care lipsesc. Se repetă simularea şi pentru alte semnale de intrare, punând în evidență asincronismul intrărilor **START** şi **T/R.**

```
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. 6.14 *Fişier de stimuli*

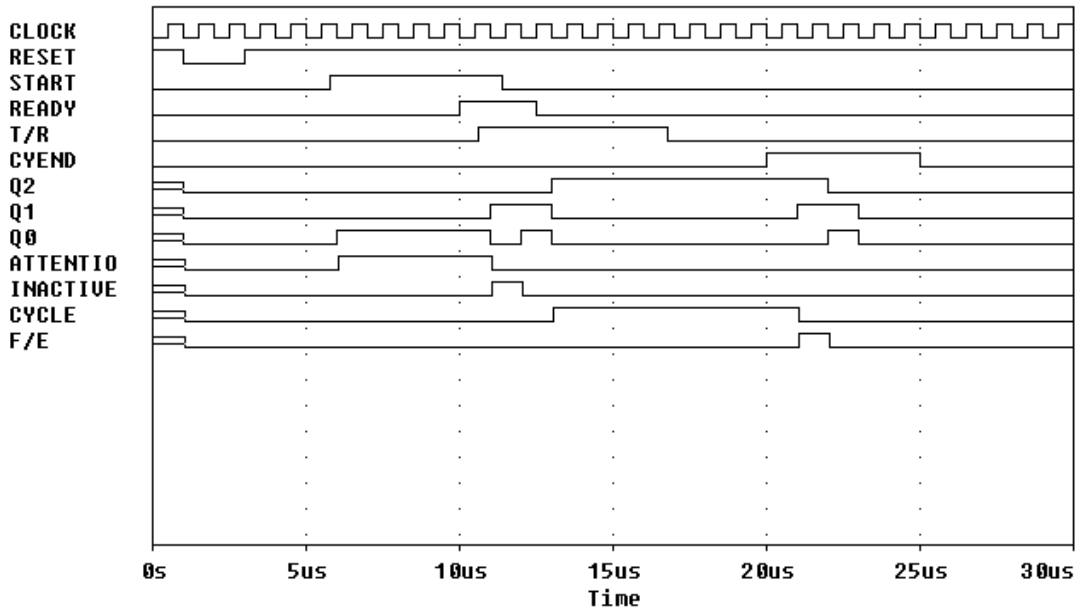

Fig. 6.15 *Simularea PSPICE a funcționării interfeței* □

6.2.3 Se analizează schema din figura 6.16 şi se discută problema metastabilității. Structura interfeței este memorată în aria combinațională prin starea fuzibilelor. Fişierul care conține această informație se numeşte PAL.JED şi este un fişier în format standard **JEDEC** (**J**oint **E**lectronic **D**evice **E**ngineering **C**ouncil). Figura 6.17 arată conținutul acestui fişier pentru exemplul considerat. Formele de undă obținute în urma simulării sunt practic identice cu cele obținute în figura 6.15, cu excepția semnalului RESET, care se activează de această dată pe 1 logic. 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.** Structura internă a circuitului PAL 16R4 este dată în figura 6.18.

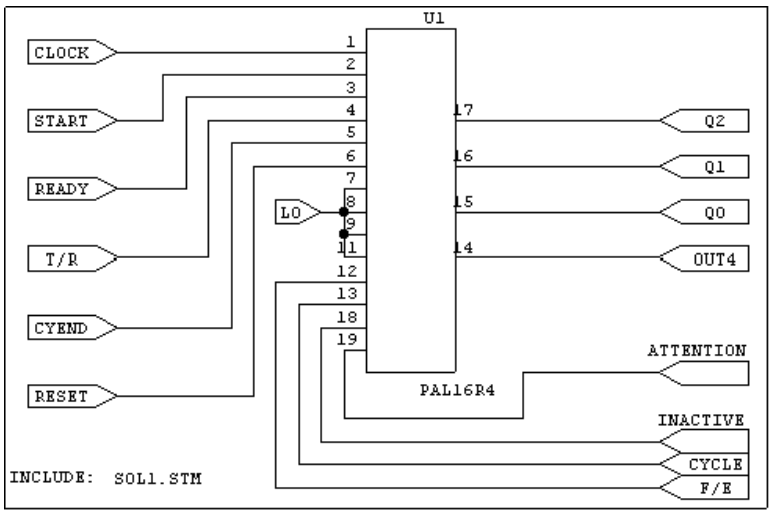

Fig. 6.16 *Schema electrică a interfeței realizată cu PAL16R4*

```
$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 11111111111111111101111111111111*
L0256 11111111111111111111111111111111*
L0288 11111111101111111111111111111111*
L0320 11111111111111101111111111111111*
L0352 11111111111111111101111111111111*
L0512 11111111111011101111111111111111*
L0544 11111111111111011110111111111111*
L0576 11111111101111011111111111111111*
L0608 11111111111111101111111111111111*
L0768 11111111110111011101111111111111*
L0800 11111111111111011101111111111111*
L0832 11111011111111111101111111111111*
L0864 11111111110110101111111111111111*
L0896 11111111111111101111111111111111*
L1024 11111111101111011111111111111111*
L1056 11111111111111011101111111111111*
L1088 11110111111111111101111111111111*
L1120 10111111111111011101111111111111*
L1152 11111111111111101111111111111111*
L1536 11111111111111111111111111111111*
L1568 11111111110111111111111111111111*
L1600 11111111111111011111111111111111*
L1792 11111111111111111111111111111111*
L1824 11111111110111111111111111111111*
L1856 11111111111111101111111111111111*
```

Fig. 6.17 *Conținutul fișierului PAL.JED* □

6.2.4 Se lansează în execuție mediul **Sonata** din programul **Symphony EDA**, grupul **VHDL Simili 2.2** şi se introduce numele unui nou proiect, selectând din meniul **File** opțiunea **NewWorkspace**. Se verifică faptul că toate acțiunile din meniu devin posibile (compilarea, simularea etc.). Se editează în fereastra de editare codul VHDL propus în figura 6.10 şi se salvează într-un fişier .vhd, fişier care se asociază cu biblioteca curentă. Se compilează fişierul creat (prima opțiune din meniul **Compile**) şi dacă nu există erori în fereastra de consolă, atunci se poate simula funcționarea circuitului. După alegerea pasului de simulare (de exemplu 100ns), din meniul **Simulate** se alege opțiunea **Go** şi la apariția ferestrei formelor de undă se selectează prin Drag and Drop semnalele pe care dorim să le vizualizăm din lista de semnale existente în fereastra **Scope**. În continuare, la fiecare apăsare de buton se desenează noile forme de undă pe durata pasului de simulare. Verificați că semnalele obținute coincid cu cele din figura 6.12. Complicați fişierul sursă, introducând şi timpii de întârziere prin porți şi reluați simulările de mai sus. □

6.2.5 Se modelează automatul finit descris prin organigrama din figura 6.5. Modelarea se poate face folosind instrucțiunea **case**. Variantele definite în instrucțiunea **case** modelează comportamentul în fiecare stare. Starea la un moment dat poate fi memorată într-un semnal. Sursa VHDL pentru o arhitectură comportamentală este dată în figura 6.19. Semnalele de intrare pentru simularea entității *interfața* se pot introduce separat cu ajutorul unui fişier de comenzi, sau pot fi descrise într-un alt fişier .vhd, care se compilează împreună cu fişierul care descrie circuitul. Vizualizați formele de undă şi verificați comportarea automatului pentru toate tranzițiile posibile. Încercați să construiți un nou fişier sursă care să realizeze o descriere structurală a automatului, asemănătoare celei din figura 6.13. Verificați prin simulare funcționarea corectă a structurii.

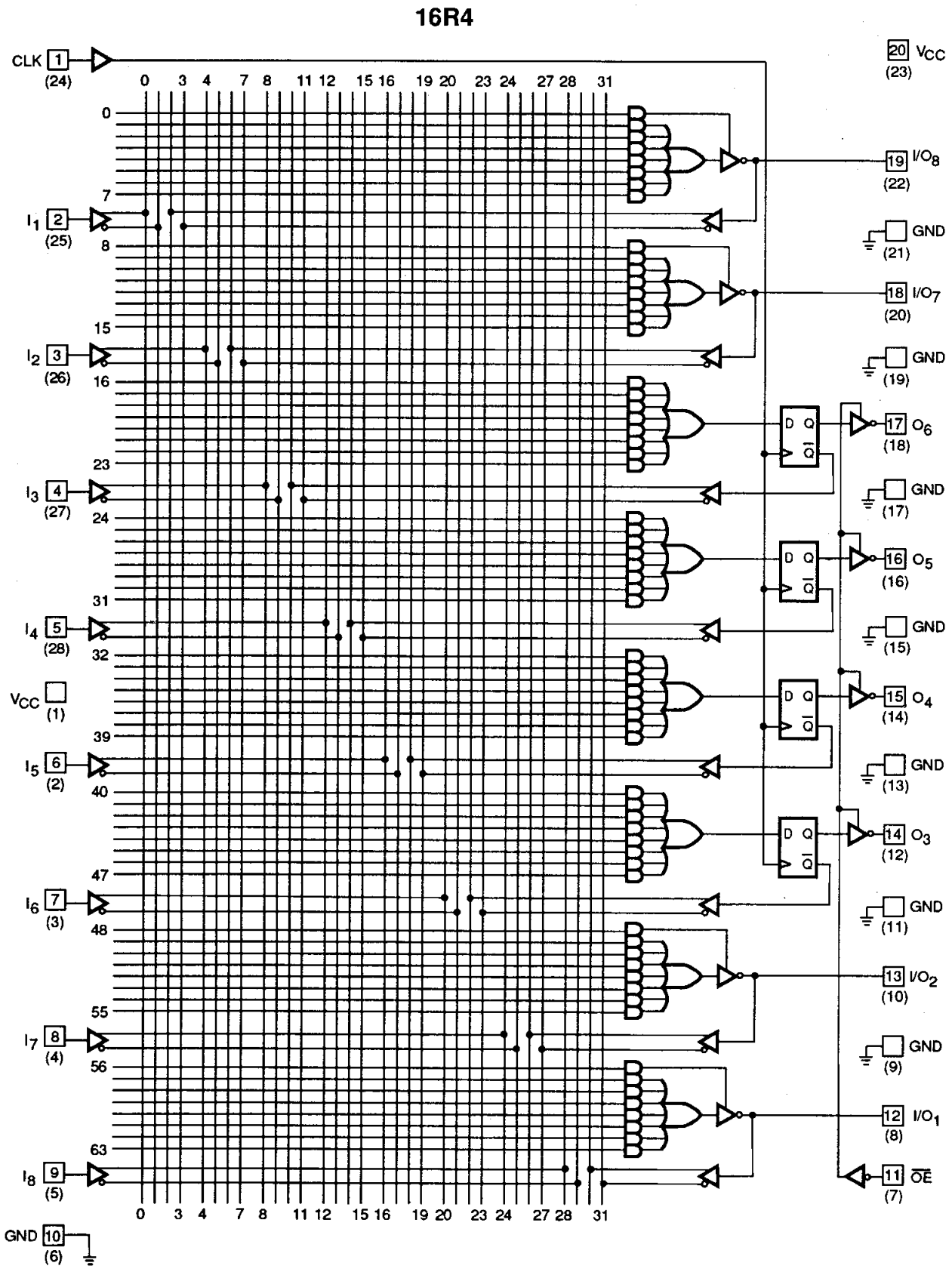

Fig. 6.18 *Structura internă a circuitului PAL16R4*

```
library IEEE;
use IEEE.std_logic_1164.all;

entity interfata is
        port(CLOCK, START, READY, TR, CYEND: in std_logic;
                ATTENTION, INACTIVE, CYCLE, FE: out std_logic);
end interfata;

architecture automat_finit of interfata is
        type state_type is (A, B, C, D, E, F);
        signal state: state_type;
        begin
                process(CLOCK)
                begin
                        if CLOCK'event and CLOCK = '1' then
                                case state is
                                        when A =>
                                                if (START = '1') then
                                                        state <= B;
                                                else
                                                        state <= A;
                                                end if;
                                        when B =>
                                                if ( READY = '1') then
                                                        state <= C;
                                                else
                                                        state <= B;
                                                end if;
                                        when C =>
                                                state <= D;
                                        when D =>
                                                if (TR = '1') then
                                                        state <= E;
                                                else
                                                        state <= A;
                                                end if;
                                        when E =>
                                                if ( CYEND = '1') then
                                                        state <= F;
                                                else
                                                        state <= E;
                                                end if;
                                        when F =>
                                                state <= D;
                                end case;
                        end if;
                end process;

                ATTENTION <= '1' when (state = B) else '0';
                INACTIVE <= '1' when (state = C) else '       0';
                CYCLE <= '1' when (state = E) else '0';
                FE <= '1' when (state = F) else '0';
        end automat_finit;
```

Fig. 6.19 *Codul VHDL pentru automatul descris în figura 6.5* □

6.3 Probleme rezolvate

6.3.1 Nivelele logice ale semnalelor de ieşire reprezentate în figura 6.15 nu sunt cunoscute în prima microsecundă de la începutul simulării. Explicaţi de ce şi arătaţi cum se poate rezolva această problemă.

Rezolvare:

Este exact aceeaşi problemă semnalată la oscilatorul din figura 6.1. Bistabilele din structura PAL nu sunt iniţializate, deci valorile ieşirilor Q_2, Q_1 şi Q_0, precum şi valorile ieşirilor ATTENTION, INACTIVE, CYCLE şi F/E, care la rândul lor sunt generate de ieşirile bistabilelor, nu sunt cunoscute la începutul simulării. Aici nu este vorba de starea de înaltă impedanţă (high Z), ci de nivele logice 0 sau 1, dar care sunt necunoscute pentru simulator.

Soluţia constă în introducerea unui semnal de RESET, care aplică un nivel logic de 0 pe intrările bistabilelor din structură, iar la activarea semnalului de ceas acestea se resetează. Din acest moment toate ieşirile automatului sunt cunoscute şi simulatorul poate analiza funcţionarea circuitului. Spre deosebire de bistabilele discrete, cele din structura PAL nu sunt prevăzute cu intrări asincrone de SET şi RESET.

Este evident că la circuitele reale această problemă nu apare pentru că ieşirile bistabilelor au o valoare logică fermă la cuplarea alimentării, deci ele sunt automat iniţializate, iar comportarea lor viitoare depinde de semnalele aplicate pe intrările circuitului. □

6.3.2 Scrieţi o secvenţă de cod VHDL pentru generarea unui semnal periodic de ceas necesar sistemelor secvenţiale sincrone.

Rezolvare: (după [Burdia, 1999])

Cea mai simplă metodă de generare a unei secvenţe periodice este utilizarea instrucţiunii de atribuire concurentă pentru semnale:

```
a <= not a after 10 ns;
```

Rezultatul este o formă de undă periodică cu perioada T = 20 ns şi factor de umplere de 50%, dacă semnalul este de tip **bit**. Pentru tipul **std_logic** valoarea de iniţializare nu mai este ‘0’, ca la tipul bit, ci ‘U’, adică necunoscută. Deci expresia not ‘U’ rămâne tot ‘U’ şi semnalul apare ca fiind necunoscut pe toată durata simulării.

O altă variantă de generare a acestui semnal este utilizarea într-un proces a instrucţiunii **wait**, cu observaţia că trebuie să existe o limitare explicită a timpului simulat printr-o introducere condiţionată fără limită de timp a lui **wait**. Codul pentru acest proces este dat mai jos:

```
process
    begin
        wait for 10 ns;
        a <= ‘1’;
        wait for 20 ns;
        a <= ‘0’;
    end process;
```

□

6.3.3 Scrieţi un program VHDL pentru un decodificator BCD – 7 segmente.

Rezolvare: (după [Cîrstea, 2001])

Considerăm că vectorul aplicat pe cele 7 segmente ale afişajului este **abcdefg**, păstrând notaţiile din problema 2.3.2. Codul VHDL pentru acest circuit este dat în figura 6.20.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity decodificator is
        port (CNTIN: in STD_LOGIC_VECTOR (3 downto 0);
              CNTOUT: out   STD_LOGIC_VECTOR (6 downto 0));
end decodificator;

architecture dec_arch of decodificator is
        begin
                WITH CNTIN SELECT
                CNTOUT <= "1111110" WHEN "0000",
                          "0110000" WHEN "0001",
                          "1101101" WHEN "0010",
                          "1111001" WHEN "0011",
                          "0110011" WHEN "0100",
                          "1011011" WHEN "0101",
                          "1011111" WHEN "0110",
                          "1110000" WHEN "0111",
                          "1111111" WHEN "1000",
                          "1111011" WHEN "1001",
                          "0000000" WHEN OTHERS;
        end dec_arch;
```

Fig. 6.20 *Un program VHDL pentru decodificatorul BCD – 7 segmente*

Entitatea "decodificator" este formată dintr-un bloc care are 4 biți de intrare grupați în magistrala CNTIN şi 7 biți de ieşire grupați în magistrala CNTOUT. Aceste semnale sunt tratate ca vectori. Arhitectura circuitului este de tip comportamental, urmărind cazurile posibile descrise în tabelul de adevăr al decodificatorului BCD-7segmente.

Instrucțiunea **WITH CNTIN SELECT** specifică faptul că valorile celor patru elemente componente ale vectorului CNTIN se vor utiliza împreună pentru a selecta o valoare care va fi încărcată în cele 7 elemente ale vectorului CNTOUT.

Observăm că LED-urile afişajului sunt aprinse pentru valoarea logică 1 aplicată segmentului respectiv, deci pentru celelate combinații binare care sunt interzise în codul BCD segmentele afişajului sunt stinse. □

6.3.4 Scrieți un program VHDL pentru un automat finit cu două intrări sincrone A şi B şi cu o ieşire Y. Ieşirea trebuie să fie 1 dacă numărul de valori 1 aplicate pe cele 2 intrări, după ultima reinițializare, este multiplu de 4; iar în celelalte cazuri ieşirea trebuie să fie 0.

Rezolvare: (după [Wakerly, 2002])

O variantă de program VHDL care rezolvă problema dată este prezentată în figura 6.21. În cadrul arhitecturii se declară un subtip COUNTER, care este o valoare UNSIGNED de 2 biți. Apoi declarăm un semnal COUNT de acest tip, pentru a păstra numărul valorilor 1, şi o constantă ZERO, de acelaşi tip, pentru inițializarea şi verificarea valorii COUNT.

În cadrul procesului, se verifică apariția frontului crescător al ceasului, iar clauza "if" face o reinițializare sincronă, în timp ce "else" adaugă la COUNT un 0, 1 sau 2, după cum apar biții de 1 pe intrările A şi B. Expresia de tipul "(0,X)" este o variabilă literală tablou, iar tipul ei este compatibil cu UNSIGNED. Deci se poate face operația de adunare, definită în pachetul std_logic_arith. În exteriorul procesului, instrucțiunea simultană de atribuire a semnalelor impune valoarea 1 la ieşirea Y atunci când COUNT este zero.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity numarator is
        port(CLOCK, RESET, A, B: in std_logic;
             Y: out std_logic);
end numarator;

architecture automat_finit of numarator is
        subtype COUNTER is UNSIGNED (1 downto 0);
        signal COUNT: COUNTER;
        constant ZERO: COUNTER := "00";
        begin
                process(CLOCK)
                begin
                        if CLOCK'event and CLOCK = '1' then
                                if RESET = '1' then COUNT <= ZERO;
                                    else COUNT <= COUNT + ('0', A) + ('0', B);
                                end if;
                        end if;
                end process;
        Y <= '1' when COUNT = ZERO else '0';
        end automat_finit;
```

Fig. 6.21 *Un program VHDL pentru automatul de numărare a valorilor de 1 pe intrări* □

6.3.5 Reluați problema 6.3.4 şi modificați procesul VHDL pentru acelaşi automat, arătând care sunt opțiunile posibile folosind o instrucțiune **case**.

Rezolvare: (după [Wakerly, 2002])

Vom prezenta în figura 6.22 numai modificarea adusă procesului. Prin formularea corespunzătoare a opțiunilor dintr-o instrucțiune **case**, devine posibilă funcționarea în paralel a celor două circuite de incrementare, iar pentru selecția uneia dintre ieşiri se poate folosi un multiplexor.

```
process(CLOCK)
variable ONES: STD_LOGIC_VECTOR (1 to 2);
        begin
            if CLOCK'event and CLOCK = '1' then
                ONES := (A, B);
                 if RESET = '1' then COUNT <= ZERO;
                 else case ONES is
                              when "01" | "10" => COUNT <= COUNT + "01";
                              when "11"        => COUNT <= COUNT + "10";
                              when  others     => null;
                         end case;
                end if;
            end if;
        end process;
```

Fig. 6.22 *Un alt proces VHDL pentru automatul de numărare a valorilor de 1 pe intrări* □