LABORATOR 3

Proiectarea comparatoarelor
Proiectarea circuitelor aritmetice (sumatoare, ALU, multiplicatoare)

1. Proiectarea comparatoarelor

Comparatoarele digitale sunt extrem de diverse ca implementare şi ca funcţii de ieşire. Ieşirile pot indica egalitatea, ne-egalitatea, relaţia de ordine (mai mare, mai mare sau egal, etc). în principiu, la implementarea circuitelor de comparare stau porţile XOR şi XNOR. De aceea, într-o primă parte a acestui paragraf ne vom referi la modelarea funcţiilor XOR şi a circuitelor de detecţie a parităţii.

A . Porţi XOR şi circuite de detecţie a parităţii

VHDL dispune de primitive XOR şi XNOR pentru a fi folosite în modelele mai ample. De exemplu, un circuit de detecţie a parităţii implementat cu porţi XOR are structura din Figura 1 (74x280).

Modelul incomplet pentru generatorul de paritate de 9 biţi este prezentat în Figura 2. Se generează ‚1' la ieşirea ODD dacă numărul de biţi de ‚1' este impar şi se generează ‚1' la ieşirea EVEN dacă numărul de biţi de ‚1' de la intrare este par.

1. Completaţi programul din Figura 2, compilaţi-1 şi simulaţi funcţionarea circuitului. Utilizaţi facilitatea *Language Templates* din ISE Xilinx.
2. Elaboraţi un raport scris care să conţină modelul VHDL complet (comentat) şi formele de undă (adnotate) care demonstrează funcţionarea corectă a circuitului în toate cazurile posibile.

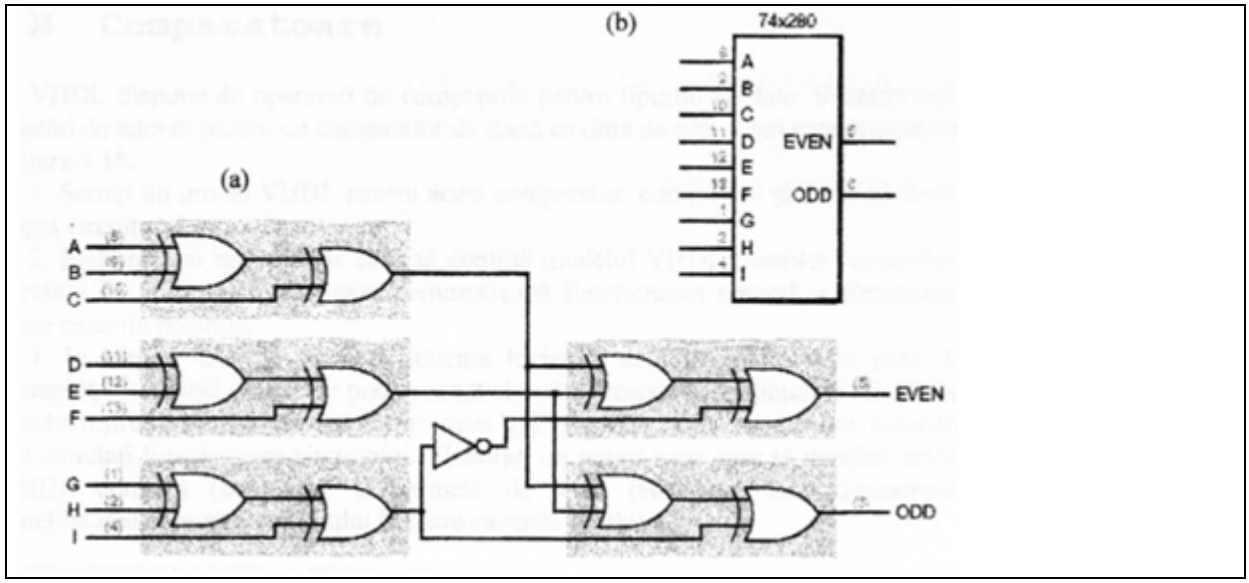


Fig. 1 Circuitul generator de paritate de 9 biţi 74x280. a) schema logică; b) schema bloc.

```
 entity parity9 is
  port  (
    I:.....;  — o intrare de 9 biţi
    EVEN, ODD:   ...  — doua ieşiri de tip ???
    ) ;
end entity;

architecture parity9a of parity9 is begin
process  (I)

variable p:   ;— aceasta variabila va conţine
               --indicaţia de paritate: 1 pentru impar, 0 par

begin
    p := I(1); — se citeşte valoarea I(1)
    for j in 2 to 9 loop

      if I(j) = ... then p := ... ;  —  se  citesc  pe  rand  biţii  de  la
      intrare si se actualizează p

      end if;
      end loop;
      ODD <= p;
    EVEN <= .....; — ODD si EVEN au valori diferite
    end process;
    end architecture parity9a;
```

Fig. 2 Modelul VHDL incomplet al circuitului generator de paritate de 9 biţi 74x280.

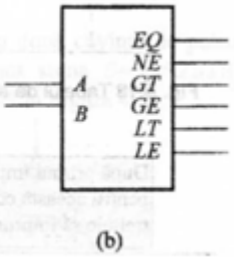
A. Comparatoare

VHDL dispune de operatori de comparaţie pentru tipurile de date. Schema bloc şi tabelul de adevăr pentru un comparator de două cuvinte de câte 8 biţi este prezentată în Figura 12.

1. Scrieţi un model VHDL pentru acest comparator, compilaţi-1 şi simulaţi funcţionarea circuitului. **Utilizaţi facilitatea *Language Templates* din ISE Xilinx.**
2. Elaboraţi un raport scris care să conţină modelul VHDL complet (comentat) şi formele de undă (adnotate) care demonstrează funcţionarea corectă a circuitului în toate cazurile posibile.
3. În Figura 13 se prezintă schema logică a unui comparator de patru biţi. Completaţi un tabel de adevăr pentru acest circuit şi descrieţi funcţionarea lui. De ce s-a notat ieşirea cu DIFF? Scrieţi un program VHDL pentru acest comparator, compilaţi-1 şi simulaţi funcţionarea circuitului. Elaboraţi un raport scris care să conţină modelul VHDL complet (comentat) şi formele de undă (adnotate) care demonstrează funcţionarea corectă a circuitului în toate.

Intrări	Ieşiri					
	EQ	NE	GT	GE	LT	LE
A=B	1	0	0	0	0	0
A/=B	0	1	0	0	0	0
A>B	0	0	1	0	0	0
A>=B	0	0	0	1	0	0
A<B	0	0	0	0	1	0
A<=B	0	0	0	0	0	1

(a)



(b)

Fig. 3 Comparator de două cuvinte de câte **8** biți. a) tabelul de adevăr; b) schema bloc

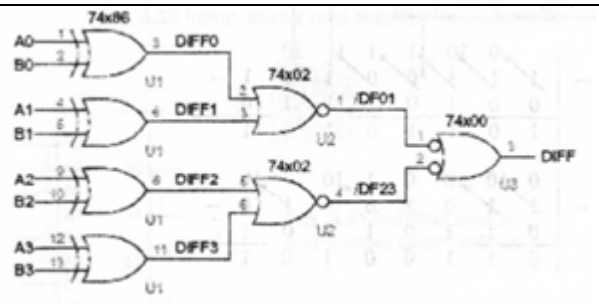


Fig. 4**.** Comparator de două cuvinte de câte **4** biți realizat cu XOR

2. **Proiectarea circuitelor aritmetice**

A Sumatoare şi scăzătoare

În figura 5 se prezintă tabelul de adevăr care descrie adunarea şi scăderea numerelor binare. Scăderea binară se efectuează în mod similar cu adunarea, dar folosind semnalele de împrumut $\mathbf{b_{in}}$ şi $\mathbf{b_{om}}$ în locul semnalelor de transport **(carry);** se obține bitul diferență **d.**

c_{in} sau b_{in}	x	y	c_{out}	s	b_{out}	d
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Fig. 5 Tabelul de adevăr pentru adunarea şl scăderea numerelor binare

O utilizare obişnuită a operației de scădere în calculatoarele numerice este realizarea comparației între două numere. De exemplu, dacă operația **X - Y** produce un **borrow out** la poziția bitului cel mai semnificativ, atunci **X** este mai mic decât **Y ;** în caz contrar, **X** este mai mare sau egal cu **Y.**

Un operator complet de scădere (scăzător complet) are ca intrări cei doi biți care trebuie scăzuți, **X** (scăzătorul) şi **Y** (descăzutul), şi împrumutul **BIN**; ieşirile sunt diferența **D** şi

împrumutul **BOUT.** Din tabelul de adevăr se pot deduce ecuațiile logice ale semnalelor de ieşire:

$$D = X \oplus Y \oplus BIN$$
$$BOUT = \overline{X} \cdot Y + \overline{X} \cdot BIN + Y \cdot BIN$$

Aceste ecuații pot fi scrise sub următoarele forme echivalente:

$$D = X \oplus \overline{Y} \oplus \overline{BIN}$$
$$\overline{BOUT} = X \cdot \overline{Y} + X \cdot \overline{BIN} + \overline{Y} \cdot \overline{BIN}$$

Aceste ultime ecuații indică faptul că putem realiza un scăzător complet dintr-un sumator complet, după cum se arată în Figura 6.

Un sumator tipic integrat este 74x283, sumator pentru două cuvinte de patru biți A=A3A2A1A0 şi B=B3B2B1B0. Ieşirile circuitului sunt suma S=S3S2S1S0 şi transportul C4. Schema bloc a acestui circuit este prezentată în Figura 7.

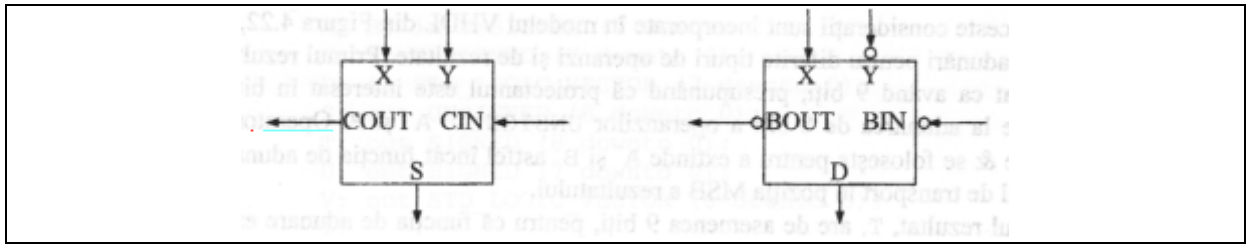


Fig. 6 Interpretarea unui scăzător ca un sumator.

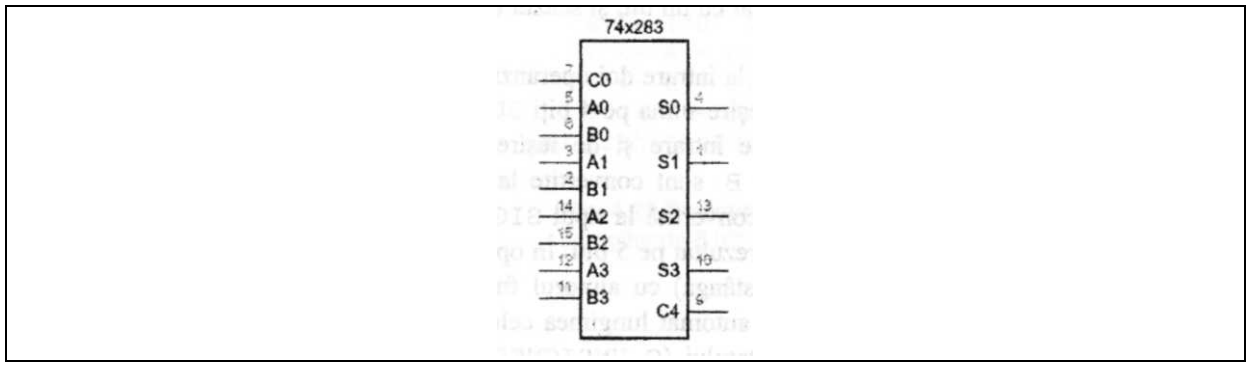


Fig. 7 Schema bloc a sumatorului binar 74x283

Deşi VHDL are operatori de adunare (+) şi de scădere (-), ei funcționează numai cu tipurile întregi, reale şi enumerare. Aşa cum au fost ei definiți, nu funcționează cu tipurile BIT_VECTOR sau STD_LOGIC_VECTOR. în schimb, alte pachete standard definesc aceşti operatori. . Pachetul **IEEE_std_logic_arith** defineşte două noi tipuri, SIGNED şi UNSIGNED. De asemenea, în pachet se definesc operațiile de adunare şi de scădere pentru aceste noi tipuri de date precum şi pentru STD_LOGIC şi STD_ULOGIC pentru operanzi de un bit.

type SIGNED **ia array** (NATURAL **range <>) of** STD_LOGIC;

type UNSIGNED **is array** (NATURAL **range <>) of** STD_LOGIC;

Tipul SIGNED se foloseşte pentru aritmetica în complement faţă de doi, iar tipul UNSIGNED se foloseşte pentru aritmetica ce conţine numere binare fără semn. Pachetul conţine operatori aritmetici şi operaţionali supraîncărcaţi pentru cele două noi tipuri, precum şi numeroase funcţii de conversie de tip: SIGNED şi UNSIGNED la INTEGER şi STD_LOGIC_VECTOR, etc.

În mod normal, dacă oricare dintre operanzi este de tipul SIGNED, rezultatul este SIGNED, în caz contrar rezultatul este UNSIGNED. Totuşi, dacă rezultatul este asignat unui semnal sau unei variabile de tipul STD_LOGIC_VECTOR, atunci rezultatul SIGNED sau UNSIGNED este convertit la acel tip. Lungimea oricărui rezultat este, de obicei, lungimea celui mai lung operand. Dar, dacă un operand UNSIGNED este combinat cu unul SIGNED sau INTEGER, lungimea creşte cu 1 pentru a include bitul de semn. Aceste consideraţii sunt incorporate în modelul VHDL din Figura 8, unde se prezintă adunări pentru diferite tipuri de operanzi şi de rezultate. Primul rezultat, S, este declarat ca având 9 biţi, presupunând că proiectantul este interesat în bitul de transport de la adunarea de 8 biţi a operanzilor UNSIGNED A şi B. Operatorul de concatenare & se foloseşte pentru a extinde A şi B astfel încât funcţia de adunare va returna bitul de transport în poziţia MSB a rezultatului.

Următorul rezultat, T, are de asemenea 9 biţi, pentru că funcţia de adunare extinde operandul UNSIGNED A atunci când acesta se combină cu operandul SIGNED C. În a treia adunare, un STD_LOGIC_VECTOR de 8 biţi D este convertit la tipul SIGNED şi combinat cu C pentru a obţine un rezultat SIGNED de 8 biţi U. În ultima instrucţiune, D este convertit la UNSIGNED, extins automat cu un bit, şi scăzut din C, pentru a conduce la un rezultat V pe 9 biţi.

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

     entity ex_add is port  (
         A,  B:  in UNSIGNED  (7 downto 0);
         C:  in  SIGNED  (7 downto 0);
         D:  in STD_LOGIC_VECTOR  (7 downto 0);
         S:  out UNSIGNED  (8 downto 0);
         T:  out SIGNED  (8 downto 0);
         U:  out SIGNED  (7 downto 0);
         V:  out STD_LOGIC_VECTOR  (8 downto 0);
         ) :
     end entity ex_add;

     architecture add_arch of ex_add is

     begin
     S <=  ( '0'& A) + ( '0'  & B) ;
     T <= A + C;
     U <= C + SIGNED  (D);
     V <= C + UNSIGNED  (D);
end architecture;
```

Fig. 8 Programul VHDL pentru adunarea şi scăderea întregilor de 8 biţi de diferite tipuri.

Primul model de sumator propus spre implementare are la intrare doi operanzi de 4 biți, A şi B, o intrare de transport, CIN, şi formează la ieşire suma pe 4 biți SUM şi transportul către rangul următor, CAR_OUT. Tipurile de intrare şi de ieşire sunt STD_LOGIC sau STD_LOGIC_VECTOR. Intrările A şi B sunt convertite la tipul SIGNED; intrarea CIN este completată cu trei biți 0 şi convertită la tipul SIGNED. Adunarea celor trei operanzi (A, B şi CIN) va produce un rezultat pe 5 biți. în operația de adunare, A este extins la 5 biți (se adaugă un 0 la stânga) cu ajutorul funcției CONV_UNSIGNED. Operatorul supraîncărcat + ajustează automat lungimea celorlalți operanzi. Cei mai puțin semnificativi patru biți ai rezultatului (C_UNSIGNED) vor reprezenta ieşirea sumă SUM. Bitul cel mai semnificativ al lui C_UNSIGNED devine CAR_OUT. De menționat că cele trei variabile intermediare sunt formate pentru că o funcție de conversie de tip nu poate determina tipul unui operand fără o astfel de declarație.

Programul parțial este prezentat în Figura 9.

1. Completați programul din Figura 9, compilați-1 şi simulați funcționarea circuitului.

2. Elaborați un raport scris care să conțină modelul VHDL complet (comentat) şi formele de undă (adnotate) care demonstrează funcționarea corectă a circuitului în toate cazurile posibile.

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

entity SIMP_ADD is
port(A,B:   ....; -- doua intrări de cate 4 biti
     CIN:....; -- intrarea de transport
     C:    .. ; -- rezultatul pe 4 biti
     CAR_0UT:.....); -- ieşirea de transport
end SIMP_ADD;

architecture ALG of SIMP_ADD is begin
  P1:process(A,B,CIN)
    variable PADDED_CIN:  STD_L0GIC_VECT0R(3 downto 0);
    variable A_UNSIGNED: UNSIGNED(3 downto 0);
    variable C_UNSIGNED: UNSIGNED(4 downto 0);
    variable temp: std_logic_vector...;
    begin
     A_UNSIGNED := UNSIGNED(A);
     PADDED_CIN    :=... ;  -- concatenarea lui CIN cu 000
     C_UNSIGNED    := CONV_UNSIGNED(A_UNSIGNED,5) + UNSIGNED(B) +
      UNSIGNED(PADDED_CIN);
     C    <= CONV_STD_LOGIC_VECTOR () ;
     Temp := ... -- se obține din C_UNSIGNED
     CAR_OUT   <=  temp(4);
   end process;
   end ALG;
```

Fig. 9 Programul VHDL incomplet pentru adunarea a două numere de **4** biți.

Al doilea circuit aritmetic este un circuit care poate efectua fie o adunare fie o scădere, între numere binare de 32 biți, a şi b. Ieşirea s reprezintă suma sau diferența celor două numere.

Intrarea de control mode determină operația care va fi realizată de circuit. Dacă mode este 0, se realizează suma dintre a şi b, dacă mode este 1 se scade b din a. În Figura 10 se prezintă programul VHDL incomplet pentru acest operator.

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

entity adder_subtractor is
    port  (a, b:       ; — doua intrări de cate 32 biţi
    mode:  .....; -- determina adunarea sau scăderea
    s :    ..; — rezultatul pe ... biti               ,
end entity adder_subtractor;

architecture behavioral of adder_subtractor is begin
  behav: process  (a, b)  is
    constant Tpd_in_out : ......;
    variable op2: std_ulogic_vector  (b'range);
    variable carry_in: std_ulogic;
    variable carry_out; std__ulogic;
    begin
     carry_out := mode;
       if mode  = '1'  then
       op2 := not b;
       else
       op2  := b;
       end if;
     for index in 0 to 31 loop
     carry__in := carry_out;
       s(index) <= a(index) xor op2 (index) xor carry_in after Tpd_in_out;
       carry_out := (a(index)  and op2(index))or  (carry_in and (a(index)  xor
       op2(index)));
       end loop;
     s(32)  <= a(31)  xor op2(31)  xor carry_out    after Tpd_in_out;
   end process;
   end architecture;
```

Fig. 10 Entitatea şi corpul arhitectural pentru un modulul de adunare/scădere.

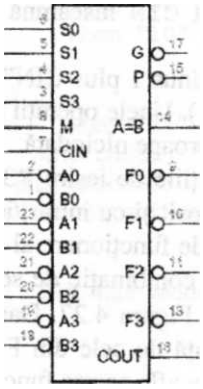
1. Completați programul din Figura 10, compilați-1 şi simulați funcționarea circuitului.
2. Elaborați un raport scris care să conțină modelul VHDL complet (comentat) şi formele de undă (adnotate) care demonstrează funcționarea corectă ă circuitului în toate cazurile posibile.

B. Unități aritmetice şi logice

O unitate aritmetică şi logică (ALU) este un circuit combinațional care poate realiza un anumit număr de operații aritmetice şi logice asupra a două cuvinte binare. O unitate MSI tipică are operanzi de 4 biți şi are 3...5 intrări de selecție a funcției de realizat, permițând astfel până la 32 operații.

În Figura 11 se prezintă schema bloc şi tabelul de adevăr pentru ALU de 4 biți 74x181.

Tipul de operație efectuată de 74x181 este selectat prin M (operații aritmetice sau logice). Operația concretă în cadrul unui grup este stabilită prin intrările S3S2S1S0. Identificatorii A, B şi F din tabelul din Figura 11se referă la cuvintele de 4 biți A3A2A1A0, B3B2B1B0 şi respectiv F3F2F1F0. Simbolurile . şi + se referă la operațiile logice AND, OR.



Inputs				Function	
S3	S2	S1	S0	M = 0 (arithmetic)	M = 1 (logic)
0	0	0	0	F = A minus 1 plus CIN	F = A′
0	0	0	1	F = A·B minus 1 plus CIN	F = A′ + B′
0	0	1	0	F = A·B′ minus 1 plus CIN	F = A′ + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B′) plus CIN	F = A′·B′
0	1	0	1	F = A·B plus (A + B′) plus CIN	F = B′
0	1	1	0	F = A minus B minus 1 plus CIN	F = A⊕B′
0	1	1	1	F = A + B′ plus CIN	F = A + B′
1	0	0	0	F = A plus (A + B) plus CIN	F = A′·B
1	0	0	1	F = A plus B plus CIN	F = A⊕B
1	0	1	0	F = A·B′ plus (A + B) plus CIN	F = B
1	0	1	1	F = A + B plus CIN	F = A + B
1	1	0	0	F = A plus A plus CIN	F = 0000
1	1	0	1	F = A·B plus A plus CIN	F = A·B′
1	1	1	0	F = A·B′ plus A plus CIN	F = A·B
1	1	1	1	F = A plus CIN	F = A

Fig. 11 Schema bloc şi tabelul de funcționare pentru ALU **74x181**

Dacă M = 1, se selectează operații logice; fiecare ieşire Fi este o funcție numai de datele corespunzătoare de intrare Ai şi Bi. între ranguri nu se propagă semnale de transport **(carry)** iar intrarea CIN este ignorată. Intrările S3-S0 selectează o operație logică particulară; se poate selecta oricare dintre cele 16 funcții logice de tip combinațional care se pot realiza asupra a două variabile.

Dacă M=0, se selectează operații aritmetice, se propagă transport **(carry)** între ranguri, iar CIN este interpretat ca intrare de transport pentru rangul cel mai puțin semnificativ. Pentru operații asupra unor cuvinte mai mari de 4 biți, pot fi cascadate mai multe ALU 74x181, conectând COUT de la fiecare unitate la CIN al unitäp'i următoare (mai semnificativă). Tuturor unităților cascadate li se aplică aceleaşi semnale de control (M, S3-S0).

Pentru a realiza adunarea în complement față de doi, se selectează „A plus B plus CIN" din S3-S0. Intrarea CIN a celei mai puțin semnificative ALU este setată la zero pe durata operațiilor de adunare. Pentru a realiza scăderea în complement față de doi. se selectează minus B minus CIN" din S3-S0. în acest caz, intrarea CIN a celei mai puțin semnificative ALU este setată la unu, întrucât CIN înseamnă împrumut **(borrow)** în cazul scăderilor.

74x181 oferă şi alte operații aritmetice, cum ar fi minus 1 plus CIN", care sunt utile în aplicații de procesare (cum ar fi decrementarea cu 1). Unele operații sunt destul de ciudate „AB plus (A + B) plus CIN" şi nu se folosesc aproape niciodată.

Intrările A3_L-A0_L şi B3_L-B0_L, precum şi funcțiile de ieşire F3__L-F0_L ale 74x181 sunt toate active L. Totuşi, 74x181 poate fi folosit şi cu intrări/ieşiri active H. în acest caz trebuie să se ia în considerare un alt tabel de funcționare al circuitului.

Pachet de operații logice

Pentru a implementa unele dintre primitivele care au rămas, este util să se definească unele funcții şi proceduri de bază. În figura 12 este descris un pachet care conține aceste funcții Toate funcțiile operează asupra bit vectorilor a căror lungime este nerestricționată. în fiecare caz se presupune că aceşti bit vectori reprezintă un număr binar şi că bitul cel mai puțin semnificativ este în dreapta. în fiecare caz se formează o variabilă internă care are gama descrescătoare, iar valoarea parametrului se asignează acestei variabile. Deci, parametrii de apelare pot avea fie gamă crescătoare fie descrescătoare, atâta timp cât bitul cel mai puțin semnificativ este în dreapta.

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

package PRIMS is
function INC(X : BIT_VECTOR) return BIT_VECTOR;
function DEC(X : BIT_VECTOR) return BIT_VECTOR;
function INTVAL(VAL : BIT_VECTOR) return INTEGER;
procedure ADD(A,B: in BIT_VECTOR; CIN: in BIT; SUM: out
BIT_VECTOR; COUT: out BIT);
end package PRIMS;

package body PRIMS is

procedure ADD(A,B: in BIT_VECTOR; CIN: in BIT;
SUM: out BIT_VECTOR; COUT: out BIT) is
variable SUMV,AV,BV: BIT_VECTOR(A'LENGTH-1 downto 0);
variable CARRY: BIT;
begin
AV := A;
BV := B;
CARRY := CIN;
for I in 0 to SUMV'HIGH loop
SUMV(I)  := AV(I) xor BV(I) xor CARRY;
CARRY := (AV(I) and BV(I)) or (AV(I) and CARRY) or (BV(i) and
CARRY);
end loop;
COUT := CARRY;
SUM := SUMV;
end procedure ADD;

function INC(X : BIT_VECTOR) return BIT_VECTOR is
variable XV: BIT_VECTOR(X'LENGTH-1 downto 0);
begin XV := X;
  for I in 0 to XV'HIGH loop
   if XV(I) = '0' then XV(I) := '1';
    exit;
   else XV(I) := '0';
   end if;
```

```
  end loop;
 return XV;
 end function INC;

function DEC(X : BIT_VECTOR)
return BIT_VECTOR is
variable XV: BIT_VECTOR(X'LENGTH -1 downto 0);
begin XV := X;
for I in 0 to XV'HIGH loop
 if XV(I) = '1' then XV(I)  := '0';
  exit;
 else XV(I) := '1';
 end if;
end loop;
return XV;
end function DEC;

function INTVAL ( VAL: BIT_VECTOR) return INTEGER is
variable VALV: BIT_VECTOR(VAL'LENGTH - 1 downto 0);
variable SUM: INTEGER := 0;
begin
VALV := VAL;
for N in VALV'LOW to VALV'HIGH loop
 if VALV(N) = '1' then SUM := SUM + (2**N);
 end if;
end loop;
return SUM;
end function INTVAL;

end PRIMS;
```

Figura 12. Pachet pentru operații logice

Acest pachet se adaugă la proiect în ISE astfel: *New source – VHDL Package*, pachetu se va numi *PRIMS.* Sursa se va putea vizualiza în fereastra *Sources*, tabul *Libraries*.
Pachetul va fi declarat în fişierul sursă folosind sintaxa

```
use work.PRIMS. all;
```

Într-o **primă etapă** se cere să se proiecteze o ALU simplă, care poate efectua patru operații între operatori de un bit. ALU are trei intrări de date: A şi B sunt bitvectori, iar CI este intrarea de transport (carry). Există două ieşiri de date: F este un bit vector care reprezintă rezultatul operației curente iar COUT este o ieşire de transport. Intrarea care selectează operația (funcția) care se execută este FSEL. în acest caz FSEL este un bit vector cu două componente, iar ALU poate efectua patru operații:

F = A,
F = not (A) ,
F = A + B
(unde + reprezintă adunarea în complement față de doi. Intrarea CI este inclusă în sumă şi se formează COUT.),

şi
F = A and B.

Operațiile mai complicate pot fi realizate prin trecerea repetatăa datelor prin ALU şi memorarea în registre a rezultatelor intermediare.

1. Completați programul din Figura 13, compilați-1 şi simulați funcționarea circuitului.
2. Folosind modelul de la punctul 1 şi cunoştințele anterioare legate de proiectarea unui circuit digital, construiți modelul pentru ALU de 4 biți din Figura 11. Simulați funcționarea acestei ALU.
3. Elaborați un raport scris care să conțină modelele VHDL complete (comentate) şi formele de undă (adnotate) care demonstrează funcționarea corectă a unităților aritmetice şi logice proiectate (în toate cazurile posibile).

```
use work.PRIMS. all;
entity ALU is
  generic(DEL: TIME);
port(A,B: in BIT_VECT0R(3 downto 0); CI: in BIT;
FSEL: in BIT_VECTOR(1 downto 0);
F: out BIT_VECTOR(3 downto 0); COUT: out BIT);
end entity ALU;

architecture ALG of ALU is
begin
  process (A,B,CI,FSEL)
    variable   FV:.. ; — imaginea rezultatului
    variable COUTV:... ; — imaginea bitului de transport
  begin
    case FSEL is
      when "00" => F ... after .... ;
      when "01" => F ... after .... ;
      when "10" => ADD (A, B, CI, EV, COUTV) ;
                   F .... after ___ ;
                   COUT . after____ ;
when "11" => F <= A and B after DEL;
end case;
end process;
end architecture ALG;
```

Fig. 13 Modelul VHDL al unei unități aritmetice şi logice simple