Lucrarea

10 Proiectarea unui multiplicator

10.1 Introducere

Proiectul are ca scop modelarea unui sistem riguros sincron care accepta la intrare doi vectori binari pe 4 biti si returneaza produsul acestora reprezentat pe 8 biti. Schema de nivel înalt a multiplicatorului este prezentata în figura 10.1.

Denumire port	Sens	Dim	Semnificatie
a	IN	4	primul operand
b	IN	4	al doilea operand
prod	OUT	8	rezultatul
Start	IN	1	startul operatiei de multiplicare
Ready	OUT	1	sfârsitul operatiei de multiplicare
Reset	IN	1	initializare asincrona
Ck	IN	1	semnal de ceas

Proiectul va trebui sa respecte constrângerile unui sistem sicron. Semnalul de ceas va fi aplicat tuturor registrelor folosite atât în calea de date cât si în calea de control. În plus, nu se vor folosi circuite de divizare a frecventei semnalului de ceas, ci circuite de generare a unor semnale de activare (enable). Descrierea sistemului se va face în întregime în subsetul VHDL propriu pachetului Alliance, instantiindu-se doua componente în descrierea de nivel înalt: cale de date si cale de control.

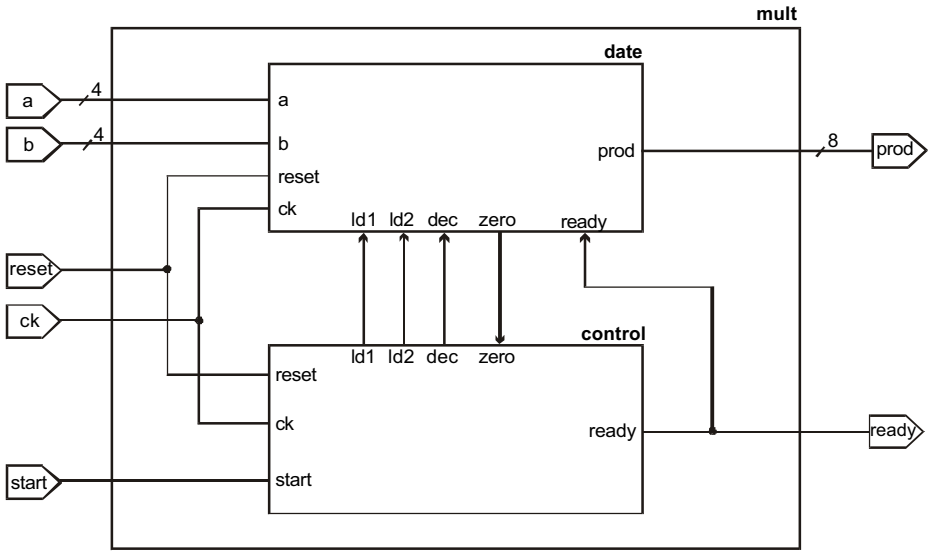

Figura 10.1 Schema bloc a multiplicatorului

Calea de date contine toate elementele necesare stocarii datelor (registre), cât si unitatea aritmetico-logica (ALU).
Calea de control este descrisa sub forma unui automat care va implementa un algoritm de înmultire prin adunari repetate.

10.2 Algoritmul de înmultire

Înmultirea prin adunari repetate constituie cel mai simplu algoritm de multiplicare cunoscut. Deînmultitul se aduna cu el însusi de un numar de ori reprezentat de înmultitor. În cazul concret al acestui multiplicator, cele doua numere sunt reprezentate pe 4 biti, produsul lor fiind pe 8 biti. Unul din cele doua numere (registrul A) va fi adunat cu el însusi în registrul P, cel de-al doilea (registrul B) urmând a fi decrementat pâna când ajunge la zero. Succesiunea operatiilor, împreuna cu un exemplu, este prezentata în continuare.

1. Reseteaza P (8 biti)
2.Încarca deînmultitul în A (4 biti)
3. Încarca înmultitorul în B (4 biti)
4. Repeta cât timp B este diferit de zero B=B-1
   P=P+A
5. P contine produs

```
7 x 10 = 70    7=00000111    10=00001010
Registrul  P   Registrul A   Registrul B
0000  0000         0111          1010
0000  0000                                  - iteratia 1
0000  0111
----  ----                        ----
```

```
      0000  0111                        1001

      0000  0111                              - iteratia 2
      0000  0111
      ----  ----                        ----
      0000  1110                        1000

      0000  1110                              - iteratia 3
      0000  0111
      ----  ----                        ----
      0001  0101                        0111

      0001  0101                              - iteratia 4
      0000  0111
      ----  ----                        ----
      0001  1100                        0110

      0001  1100                              - iteratia 5
      0000  0111
      ----  ----                        ----
      0010  0011                        0101

      0010  0011                              - iteratia 6
      0000  0111
      ----  ----                        ----
      0010  1010                        0100

      0010  1010                              - iteratia 7
      0000  0111
      ----  ----                        ----
      0011  0001                        0011

      0011  0001                              - iteratia 8
      0000  0111
      ----  ----                        ----
      0011  1000                        0010

      0011  1000                              - iteratia 9
      0000  0111
      ----  ----                        ----
      0011  1111                        0001

      0011  1000                              - iteratia 10
      0000  0111
      ----  ----                        ----
      0100  0110                        0000

Produs = 0100 0110 = 16x4 + 6 = 70
```

10.3 Unitati functionale

Multiplicatorul este structurat în:
- cale de date
- cale de control

Calea de control este implementata ca un automat finit pe baza algoritmului de înmultire cu adunari repetate. Calea de date contine restul de elemente necesare functionarii circuitului si anume registre, ALU etc. Comunicatia între cele doua unitati functionale va fi realizata prin semnale interne.

10.3.1 Calea de date

Pentru calea de date sunt necesari doi registri de patru biti în care sa fie încarcati cei doi operanzi la începutul operatiei de înmultire. Sunt necesare doua registre de câte 8 biti, unul pentru a memora rezultatul intermediar al adunarilor repetate, iar cel de-al doilea pentru a memora rezultatul la terminarea operatiei de multiplicare. Unitatea aritmetico-logica este un sumator pe 8 biti, având ca intrare 4 biti ai deînmultitului si 8 biti din registrul imtermediar. Rezultatul adunarii este încarcat din nou în registrul temporar la fiecare iteratie, exceptând-o pe ultima, în care rezultatul este încarcat în registrul de iesire, odata cu activarea semnalului **ready**.

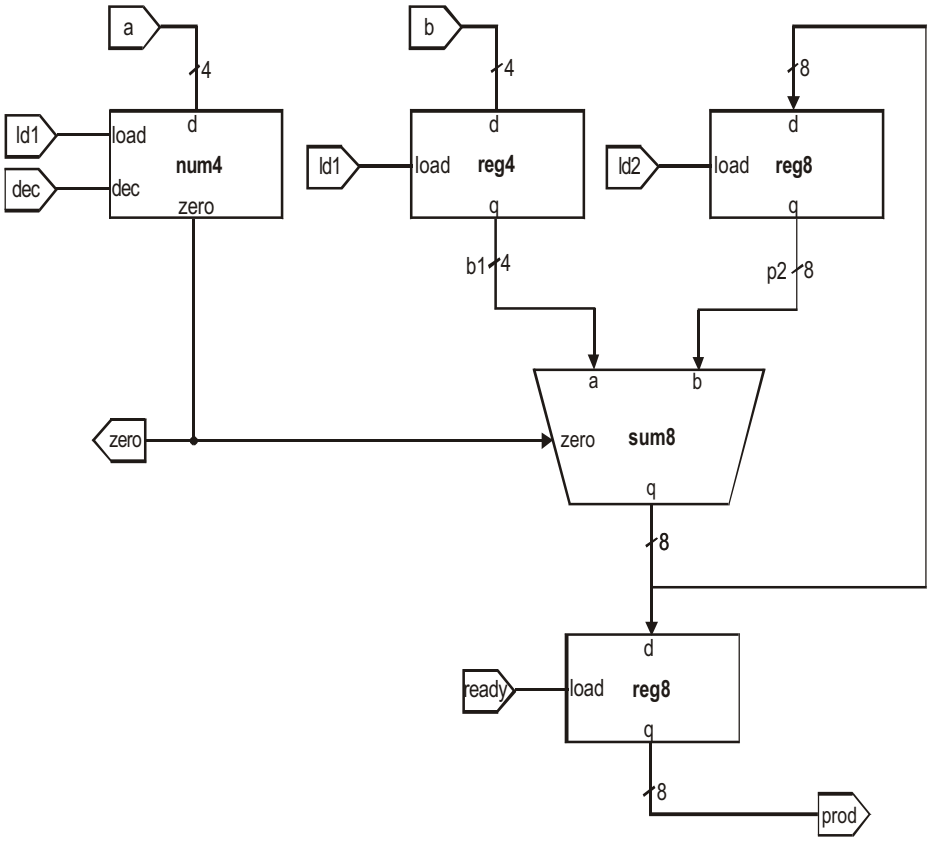

Figura 10.2: Calea de date a multiplicatorului

Operandul **a** este încarcat într-un registru la comada **start**. La fiecare iteratie, acest registru este decrementat. Când contorul registrului ajunge la ‘0’ semnalul **zero** anunta calea de control despre aparitia acestui eveniment. Calea de date este prezentata în figura 10.2.

Sumatorul ete modelat pe baza functiei logice, dupa cum urmeaza:

```
-- file: sum8.vbe
ENTITY sum8 IS
     PORT(  vdd :IN bit;
            vss :IN bit;
               a :IN bit_vector(3 DOWNTO 0);
               b :IN bit_vector(7 DOWNTO 0);
            zero :IN bit;
               q :OUT bit_vector(7 DOWNTO 0));
END sum8;

ARCHITECTURE behave OF sum8 IS

   SIGNAL carry:bit_vector(7 DOWNTO 0);
   SIGNAL q_s: bit_vector(7 DOWNTO 0);
   SIGNAL  a1: bit_vector(3 DOWNTO 0);

BEGIN
  WITH zero SELECT
     a1 <= a WHEN ,0‘, "0000" WHEN ,1‘;

  carry(0) <= ,0‘;
  q_s(0) <= (a1(0) XOR b(0)) XOR carry(0);
  carry(1) <= (a1(0) AND b(0))
            OR (a1(0) AND carry(0))
                 OR (b(0) AND carry(0));
  q_s(1) <= (a1(1) XOR b(1)) XOR carry(1);
  carry(2) <= (a1(1) AND b(1))
              OR (a1(1) AND carry(1))
              OR (b(1) AND carry(1));
  q_s(2) <= (a1(2) XOR b(2)) XOR carry(2);
  carry(3) <= (a1(2) AND b(2))
              OR (a1(2) AND carry(2))
              OR (b(2) AND carry(2));
  q_s(3) <= (a1(3) XOR b(3)) XOR carry(3);
  carry(4) <= (a1(3) AND b(3))
              OR (a1(3) AND carry(3))
              OR (b(3) AND carry(3));
  q_s(4) <=  b(4) XOR carry(4);
  carry(5) <= b(4) AND carry(4);
```

```
   q_s(5) <= b(5) XOR carry(5);
   carry(6) <= b(5) AND carry(5);
   q_s(6) <= b(6) XOR carry(6);
   carry(7) <= b(6) AND carry(6);
   q_s(7) <= b(7) XOR carry(7);

   q <= q_s;

   ASSERT((vdd = '1') and (vss = '0'))
      REPORT "Power supply is missing on sum8"
      SEVERITY WARNING;

END behave;
```

Semnalul **ld1** este aplicat atât registrului b, cât si numaratorului si este generat doar la începutul operatiei de înmultire pentru a încarca cei doi operanzi. Semnalul **ld2** este aplicat doar registrului intermediar pe 8 biti. Semnalul **zero** este generat de numarator dupa un numar de perioade de ceas egal cu înmultitorul si este aplicat atât sumatorului, cât si trimis mai departe la calea de control. Numaratorul este comandat de semnalul **dec** ce vine de la calea de control.

Registrul de iesire se încarca doar când semnalul **ready** (provenind de la calea de control) este activ, adica atunci când operatia de înmultire este încheiata, datele fiind asftel valide la iesirea circuitului. Toate registrele sunt comutate de acelas semnal de ceas si initializate de acelas semanl **reset**.

Descrierea comportamentala a unui registru poate fi facuta conform exemplului urmator. Extinderea la un registru de 8 biti se face prin simpla modificare a dimensiunilor porturilor si a vectorilor.

```
-- file: reg4.vbe

ENTITY reg4 IS
  PORT(   vdd :IN bit;
          vss :IN bit;
           d :IN bit_vector(3 DOWNTO 0);
        reset :IN bit;
           ck :IN bit;
         load :IN bit;
            q :OUT bit_vector(3 DOWNTO 0));
END reg4;

ARCHITECTURE behave OF reg4 IS
```

```
SIGNAL q_s: reg_vector(3 DOWNTO 0) register;

BEGIN
    block1: BLOCK(ck='1' and not ck'STABLE)
           BEGIN
          q_s <= GUARDED "0000"    WHEN reset='1'
                    ELSE      d    WHEN reset='0' and load='1'
                    ELSE    q_s;
          END BLOCK block1;
    q <= q_s;

ASSERT((vdd = '1') and (vss = '0'))
   REPORT "Power supply is missing on reg4"
   SEVERITY WARNING;

END   behave;
```

10.3.2 Calea de control

Calea de control are urmatoarele intrari si iesiri:
- intrarea **start** provenind din exteriorul circuitului si intrarea **zero** de la numarator;
- iesirile **ld1**, **ld2** si iesirea **ready** pentru registrul de iesire si deasemnea pentru exteriorul circuitului.

Schema caii de control este prezentata în figura 10.3.

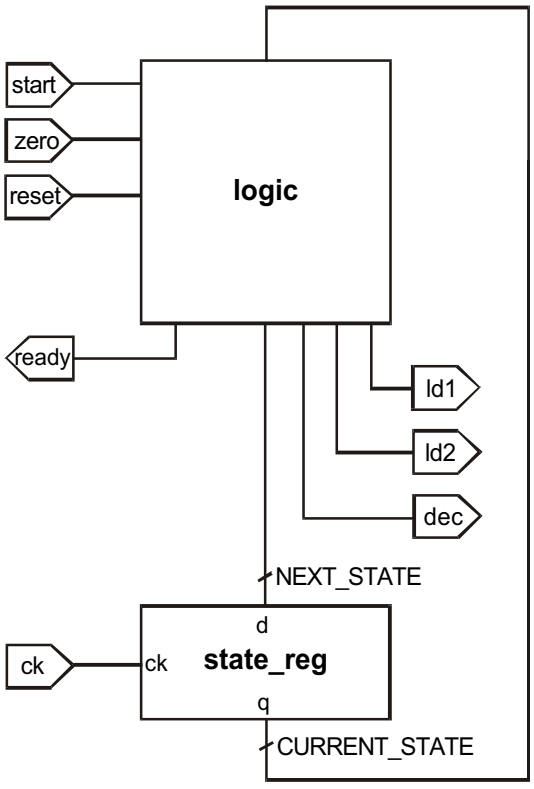

Figura 10.3: Calea de control si conexiunile cu calea de date

Numaratorul pe patru biti poate fi descris pe baza functiei logice, folosind un bloc cu garda având în lista de senzitivitati semnalul de ceas:

```
-- file: num4.vbe

ENTITY num4 IS
    PORT(   vdd :IN bit;
            vss :IN bit;
               d :IN bit_vector(3 DOWNTO 0);
          reset :IN bit;
             ck :IN bit;
           load :IN bit;
            dec :IN bit;
           zero :OUT bit);
END num4;

ARCHITECTURE behave OF num4 IS

      SIGNAL borrow: bit_vector(3 DOWNTO 0);
      SIGNAL b: bit_vector(3 DOWNTO 0);
      SIGNAL d1 : bit_vector(3 DOWNTO 0);
      SIGNAL d2 : reg_vector(3 DOWNTO 0) register;

BEGIN
  borrow(0) <= '0';

  WITH dec SELECT
  b           <= "0001" WHEN '1', "0000" WHEN '0';

  d1(0)       <= (d2(0) XOR b(0)) XOR borrow(0);
  borrow(1)   <=(((NOT d2(0)) AND b(0)) OR ((NOT d2(0))
                   AND borrow(0))OR (b(0) AND borrow(0)));
  d1(1)       <= (d2(1) XOR b(1)) XOR borrow(1);
  borrow(2)   <= (((NOT d2(1)) AND b(1))
                    OR ((NOT d2(1)) AND borrow(1))
                   OR (b(1) AND borrow(1)));
  d1(2)       <= (d2(2) XOR b(2)) XOR borrow(2);

  borrow(3)   <= (((NOT d2(2)) AND b(2)) OR ((NOT d2(2))
                   AND borrow(2)) OR (b(2) AND borrow(2)));
  d1(3)       <= (d2(3) XOR b(3)) XOR borrow(3);
```

```
  L1: BLOCK ((ck = '1') AND NOT ck'STABLE)
    BEGIN
       d2 <= GUARDED d WHEN reset = '0' and load = '1'
             ELSE   d1 WHEN reset = '0' and load = '0'
            ELSE   "1111";
    END BLOCK L1;

  zero <=  NOT (d2(0) OR d2(1) OR d2(2) OR d2(3));

  ASSERT((vdd = '1') and (vss = '0'))

     REPORT "Power supply is missing on num4"
     SEVERITY WARNING;

END behave;
```

Automatul este de tip Mealy imediat. Graful de tranzitie al automatului este prezentat în figura 10.3.
Trecerea din starea S0 în S1 se face la activarea semnalului **start**. Automatul ramâne în starea S1 pâna când semnalul **zero** devine activ, moment în care semnalul **ready** este activat, semnalizând terminarea operatiei de înmultire.

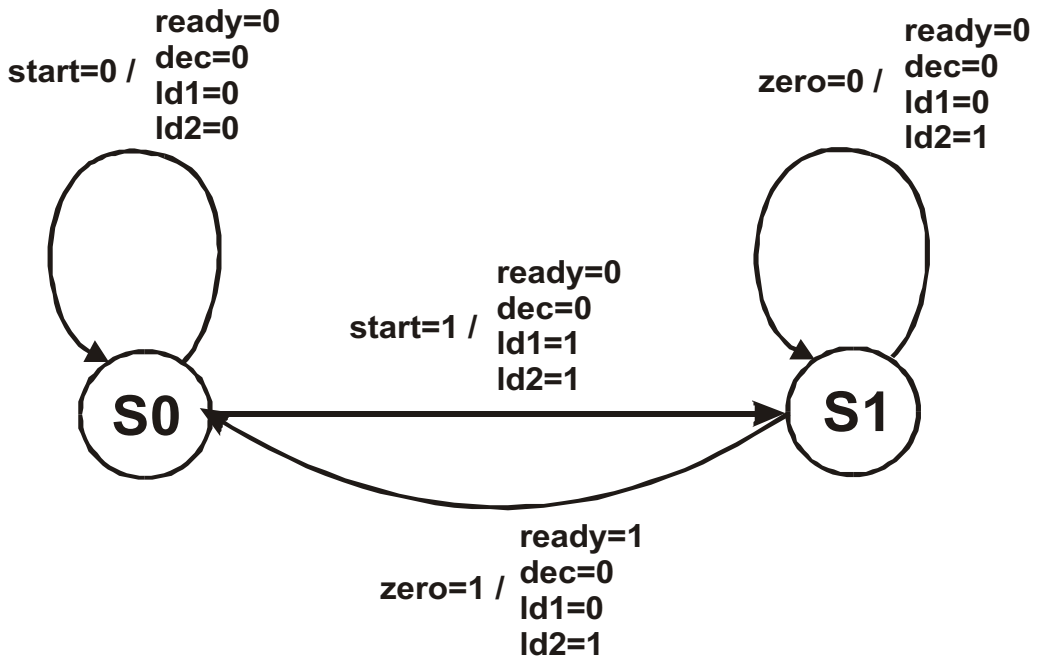

Figura 10.4: Graful de tranzitie al automatului

```
-- file: automat.fsm

ENTITY automat IS
  PORT(
         reset :IN bit;
         start :IN bit;
            ck :IN bit;
          zero :IN bit;
           vdd :IN bit;
           vss :IN bit;
           ld1 :OUT bit;
           ld2 :OUT bit;
           dec :OUT bit;
         ready :OUT bit
      );
END automat;


ARCHITECTURE automat_a OF automat IS

TYPE STATE_TYPE IS (S0, S1);

 — pragma CLOCK ck
 — pragma CUR_STATE CURRENT_STATE
 — pragma NEX_STATE NEXT_STATE


SIGNAL CURRENT_STATE, NEXT_STATE: STATE_TYPE;

BEGIN
   PROCESS (CURRENT_STATE, start, reset, zero)
  BEGIN

     IF(reset = '1') THEN
       NEXT_STATE <= S0;
       ld1 <= '0';
       ld2 <= '0';
       dec <= '0';
       ready <= '0';

    ELSE
      CASE CURRENT_STATE IS

        WHEN S0 =>
          IF(start = '1') THEN
            NEXT_STATE <= S1;
            ld1 <= '1';
```

```
          ld2 <= '1';
          dec <= '0';
          ready <= '0';
        ELSE
          NEXT_STATE <= S0;
          ld1 <= '0';
          ld2 <= '0';
          dec <= '0';
          ready <= '0';
        END IF;

       WHEN S1 =>
        IF(zero = '0') THEN
          NEXT_STATE <= S1;
          ld1 <= '0';
          ld2 <= '1';
          dec <= '1';
          ready <= '0';
        ELSE
          NEXT_STATE <= S0;
          ld1 <= '0';
          ld2 <= '0';
          dec <= '0';
          ready <= '1';
        END IF;

     END CASE;
    END IF;
  END PROCESS;

  PROCESS(ck)
  BEGIN
    IF(ck='1' and not ck'STABLE) THEN
      CURRENT_STATE <= NEXT_STATE;
    END IF;
  END PROCESS;
END automat_a;
```

10.3.3 Generarea descrierilor structurale

Pornind de la fisierele cu descrierile comportamentale (*reg4.vbe*, *reg8.vbe*, *sum4.vbe* si *count4.vbe*) se genereaza descrierile structurale (fisiere cu extensia vst). Operatiunea se face prin executarea utilitarului **scmap**.

```
scmap   reg4 reg4
```