CURS 7. Procesoare Orientate pe Aplicati

7. Implementarea în FPGA a procesoarelor aritmetice partiționate/nepartiționate în unitate de comandă şi unitate de execuție.

După cum s-a arătat într-unul din paragrafele precedente, implementarea sistemelor numerice în FPGA se poate realiza pe baza specificării algoritmului de operare într-un limbaj de descriere hardware (HDL). La rândul ei specificarea algoritmului se poate face cel puțin în doua maniere: *cu detalierea secțiunilor de execuție şi comandă* sau *fără detalierea acestora*.

În primul caz, abordarea are un caracter tradițional, proiectantul fiind obligat să analizeze secțiunile de execuție şi control din punctul de vedere al: intrărilor/ieşirilor, resurselor hardware implicate, stărilor automatului de comandă, semnalelor de comandă şi de condiții, ceea ce, în cazul sistemelor numerice complexe, ridică probleme. Astfel, se poate prelungi durata proiectării, în condițiile obținerii unor soluții mai puțin avantajoase sub aspectul vitezei de operare (timpul de execuție a algoritmului), cât şi al cantității de componente hardware utilizate.

În continuare se prezintă, *din punct de vedere metodologic*, două descrieri ale algoritmului cmmdc în variantele menționate mai sus.

7.1. Algoritmul cmmdc partiționat pentru descrierea operării secțiunii de comandă cmmdcc şi a secțiunii de execuție cmmdce.

Partiționarea s-a efectuat conform schemei bloc prezentate în fig.7.1. Atât secțiunea de comandă, cât şi secțiunea de execuție primesc, de la mediul extern, semnalele de intrare: clk, clr, start, în plus secțiunea de execuție acceptă şi operanzii xi, yi. Automatul de comandă este un automat cu stări finite şi tranziții condiționate (în principiu poate fi de tip Mealy sau Moore). Automatul transmite secțiunii de execuție informații de stare, care evocă, în această unitate anumite, operații elementare: încărcarea registrelor cu operanzii de la intrare, scăderea operanzilor, interschimbul operanzilor. La rândul său secțiunea de execuție transmite secțiunii de comandă privind condițiile îndeplinite de operanzii stocați în registre: xeqy (egali), xney (neegali), xgty (mai mare). Procesul ia sfârşit dupa un număr de paşi, când secțiunea de comandă generează semnalul gata şi secțiunea de execuție valoarea cmmdc.

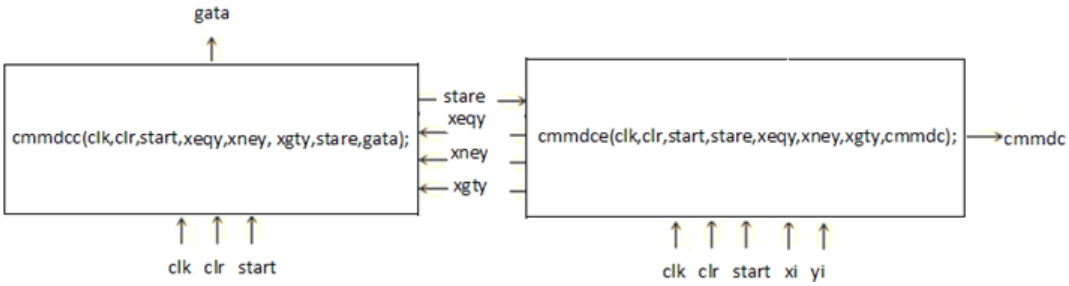


Fig. 7.1. Schema bloc a implementării algoritmului cmmdc partiționat în secțiunile de comandă cmmdcc şi de execuție cmmdce.

/*1. Exemplu de modul pentru calculul celui mai mare divizor comun **(**cmmdc**),** în varianta pariționării în unitate de comandă module CMMDCC(clk,start,clr,xeqy,xney,xgty,gata,stare); şi unitate de execuție: module CMMDCE(clk,start,clr,stare, xi, yi, xeqy,xney,xgty,cmmdc); */

/* Calculează cmmdc pentru numerele Xi[15:0], Yi[15:0], activează clr pentru inițializare, apoi dezactivează, pentru a începe execuția. Când răspunsul este disponibil, de la modulul CMMDCC[15:0], este activat semnalul gata. */

```
module CMMDCC(clk,start,clr,xeqy,xney,xgty,gata,stare);
 input clk,clr,start,xeqy,xney,xgty;
 output gata; output [2:0] stare;
 reg[2:0] stare; reg gata;          // fortat 1 când calculul a luat sfârşit.
 always @(posedge clk)              // secțiune inițializată pe frontal pozitiv al semnalului clk.
begin: ciclu_CMMDC
  if(clr)                           //  clr=1– încarcă registrele interne şi resetează indicatorul gata.
   begin
  gata<=0; stare <= 1;
   end
  else                             // clr=0 – ciclează în stările 1, 2, 3.
  case(stare)
  1: begin
     if(start)                     // start=1 – se trece în starea 2.
     begin
     gata<=0; stare <= 2;
      end
      else
      begin                        // start = 0 – se rămane în starea 1.
      stare <= 1;
      end
     end
  2: begin
    if(xeqy)                       // xeqy =1 – se trece în starea 1 şi gata =1.
     begin
        gata <= 1; stare <= 1;
     end
    else if(xney)                  //  xney = 1 – se trece în starea 3.
```

```
                    begin
                     stare <= 3;
                     end
                   end
                3: begin
                        stare <= 2;        // fara condiții se trece în starea 2.
                     end
               default:  stare <= 0;
              endcase
              end
             endmodule
//=================================================================================
             module CMMDCE(clk,start,clr,stare, xi, yi, xeqy,xney,xgty,cmmdc);
             input clk,start,clr;
             input [2:0] stare;
             input [15:0] xi, yi;
             output[15:0] cmmdc;
             output xeqy,xney,xgty;
             reg [15:0] x,y,cmmdc;
             reg xeqy,xney,xgty;
             always @(posedge clk)
             begin
              xeqy<= (x==y);
              xney<= (x!=y);
              xgty <= (x>y);
                 if(clr)
                   begin
              cmmdc<=0;
                   end
                 if ((stare==1)&start)
                    begin
                      x<=xi; y<=yi;
                    end
                 if ((stare==2) & (x==y))
                      begin
                        cmmdc <= x;
                      end
                 if ((stare==3) &(x>y))
                      begin
                        x<=x-y;
                      end
                 if ((stare==3)&(x<y))
                      begin
                        y<=y-x;
                      end
              end
```

```
                endmodule
//=============================================================================

                // test principal -- nesintetizabil.
                 module principal;
                 reg clk,clr,start;      // intrări pentru circuitul de test
                 reg [15:0] xi,yi;

                wire [2:0] stare;
                 wire [15:0] cmmdc;     // ieşiri de la circuitul de test
                 wire gata,xeqy,xney,xgty;
                /* instanțe ale modulelor CMMDCC/CMMDCE, numite "icmmdcc"/"icmmdce",
                  în care primul argument este ataşat primului port al modulului etc;
                  argumentele şi porturile au aceleaşi nume */

                     CMMDCC icmmdcc(clk,start,clr,xeqy,xney,xgty, gata,stare);

                     CMMDCE icmmdce(clk,start,clr,stare, xi, yi, xeqy,xney,xgty,cmmdc);

                // semnal de ceas periodic : perioada 2 şi  ciclul de lucru: 50%
                     always #1 clk = ~clk;
                // afişează rezultetele pe frontul căzător al semnalului de ceas
                   always @(posedge clk)
                    begin
                    $display("Timp=",$time, "  clr=", clr,"  start=",start,"  stare =",stare," cmmdc=",cmmdc,"
                            gata=",gata," x=",xi,"  y=",yi);

                    $vw_dumpvars();
                    $vw_group("all",clk,xi,yi,clr,start,stare,cmmdc,gata);

                   end
                // când este activat gata se termină simularea
                 always @(negedge clk)
                  if (gata) $stop;
                //Se execută la începutul simulării şi  inițializează: clk, clr, xi, yi
                  initial begin
                    clk = 0; clr = 0; start=0; xi = 64; yi = 12;
                    #1 clr=1;
                    #2 clr = 0;
                    #2start =1;
                    #2 start=0;
                  end
                endmodule
```

```
C1> .
Timp=  0 clr=0 start=0 stare =z cmmdc=  x gata=x x=  64 y=  12
Timp=  2 clr=1 start=0 stare =1 cmmdc=  0 gata=0 x=  64 y=  12
Timp=  4 clr=0 start=1 stare =1 cmmdc=  0 gata=0 x=  64 y=  12
Timp=  6 clr=0 start=0 stare =2 cmmdc=  0 gata=0 x=  64 y=  12
Timp=  8 clr=0 start=0 stare =2 cmmdc=  0 gata=0 x=  64 y=  12
Timp= 10 clr=0 start=0 stare =3 cmmdc=  0 gata=0 x=  64 y=  12
Timp= 12 clr=0 start=0 stare =2 cmmdc=  0 gata=0 x=  64 y=  12
================================================================================
Timp= 34 clr=0 start=0 stare =3 cmmdc=  0 gata=0 x=  64 y=  12
Timp= 36 clr=0 start=0 stare =2 cmmdc=  0 gata=0 x=  64 y=  12
Timp= 38 clr=0 start=0 stare =3 cmmdc=  4 gata=0 x=  64 y=  12
Timp= 40 clr=0 start=0 stare =2 cmmdc=  4 gata=0 x=  64 y=  12
Timp= 42 clr=0 start=0 stare =1 cmmdc=  4 gata=1 x=  64 y=  12
Stop at simulation time 42
```

Rezultatele simulării arată că după 42 de unităţi de timp sunt stabile:

- rezultatul: cmmdc = 4;
- semnalul de sfârşit de operaţie: gata = 1.

7.2. Algoritmul nepartiţionat cmmdc pentru secţiunile de comandă şi de execuţie.

Algoritmul este văzut ca un singur modul cmmdc, cu intrările şi ieşirile prezentate în fig.7.2.

Fig.7.2. Schema bloc a implementării algoritmului nepartiţionat cmmdc.

/*2. Exemplu de modul pentru calculul celui mai mare divizor comun (cmmdc**), în varianta nepariţionării în unitate de comandă şi unitate de execuţie.***/

/* calculează cmmdc pentru numerele XI[15:0], Y[15:0], activează clr pentru iniţializare, apoi dezactivează clr, pentru a începe execuţia; gata este activat atunci când răspunsul este disponibil de la modulul CMMDCC[15:0] */

```
module CMMDC(clk,xi,yi,clr,start,cmmdc,gata);
 input clk,clr,start;
 input [15:0] xi,yi;
 output gata;
 output [15:0] cmmdc;
 reg [15:0] cmmdc;      // rezultatul  stabilit la ultima iteraţie;
 reg gata;              // forţat 1 când calculul a luat sfârsit
```

```
 always @(posedge clk) begin: ciclu_CMMDC
     reg [15:0] x,y;        //  registre interne care stochează stările intermediare;
   if (clr)    //  clr <= 1  încarcă registrele interne şi anulează indicatorul gata;
     begin
     cmmdc<=0;
     gata <= 0;
      end
      else
        begin
   if(start)
      begin
      x <= xi; y <= yi;
      end
    else if (!gata)
     begin
      if (x == y)
       begin
          cmmdc <= x; gata <= 1;
       end
     else if (x > y) x <= x - y;
         else y <= y - x;
      end
     end
end
endmodule

 // test principal -- nesintetizabil;
   module principal;
   reg clk,clr,start;              // intrări pentru circuitul de test;
   reg [15:0] xi,yi;
   wire [15:0] cmmdc;     // ieşiri de la circuitul de test;
   wire gata;

  /* instanță a modulului CMMDC module, numită "icmmdc";
     primul argument este ataşat primului port al modulului;
     argumentele şi porturile au aceleaşi nume */

       CMMDC icmmdc(clk,xi,yi,clr,start,cmmdc,gata);

  // semnal de ceas periodic : perioada 8;  ciclul de lucru: 50%;
       always #1 clk = ~clk;

  // afişeaza rezultetele pe frontul căzător al semnalului de ceas;
      always @(negedge clk)
       begin
       $display(" time=", $time, " clr=", clr," start=", start," cmmdc=",cmmdc," gata=",gata, "
```

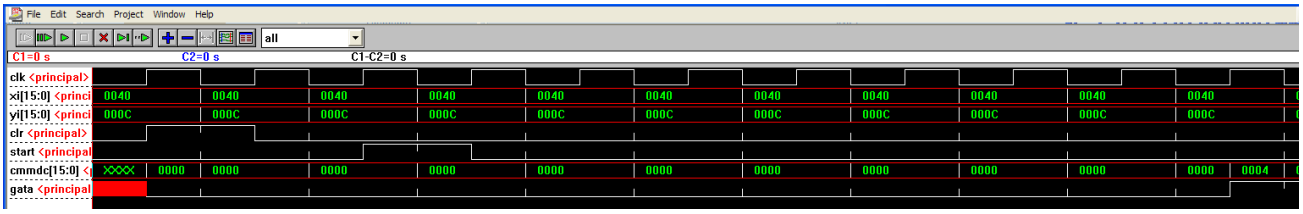
```
                x=",xi, " y=",yi);
    $vw_dumpvars();
    $vw_group("all",clk,xi,yi,clr,start,cmmdc,gata);
    end

  // când este activat gata se termina simularea;
    always @(posedge clk)
    if (gata) $stop;

  // se execută la începutul simulării şi  inițializează: clk, clr,xi, yi;
    initial begin
    clk = 0; clr = 0; start=0;xi = 64; yi = 12;
    #1 clr=1;
    #2 clr = 0;
    #2start =1;
    #2 start=0;
   end
  endmodule
```

```
C1> .
 time=  0  clr=0  start=0  cmmdc= z  gata=z  x= 64  y= 12
 time=  2  clr=1  start=0  cmmdc=  0  gata=0  x= 64  y= 12
 time=  4  clr=0  start=0  cmmdc= 0  gata=0  x= 64  y= 12
 time=  6  clr=0  start=1  cmmdc= 0  gata=0  x= 64  y= 12
 time=  8  clr=0  start=0  cmmdc= 0  gata=0  x= 64  y= 12
==================================================================
 time= 20  clr=0  start=0  cmmdc= 0  gata=0  x= 64  y= 12
 time= 22  clr=0  start=0  cmmdc= 4  gata=1  x= 64  y= 12
Stop at simulation time 23
```

Tab. 7.1. Rezultatele simulării.



Rezultatele simulării arată că după 22 de unități de timp se obțin stabile:

- rezultatul: cmmdc = 4;
- semnalul de sfârşit de operație: gata = 1.

Conform rezultatelor simulării se constată că soluția bazată pe implementarea algoritmului nepartiționat în unitate de comandă şi unitate de execuție este de circa două ori mai rapidă, în cazul

de față. Astfel, se poate face *recomandarea* ca, *în cazurile în care nu există o cerința majoră de a pune în evidență secțiunile de comandă şi de execuție ale algoritmului, să se renunțe la partiționare, întrucât se va obține un sistem numeric mai rapid şi cu mai puține componente hardware.*

7.2.1. Implementarea algoritmului cmmdc nepartiționat pe platforma experimentală.

Platforma pe care s-a realizat experimentul a avut la bază placheta Spartan-3A/3AN FPFA, la care s-au conectat o tastatură PS/2 şi un monitor VGA. După cum se poate constata din fig.7.3. au fost folosite module Verilog pentru descrierea interfețelor cu tastatura şi monitorul, cât şi pentru o unitate aritmetica-logica generica (alu.v), care instanțiază modulul Verilog cmmdc (gcd.v).

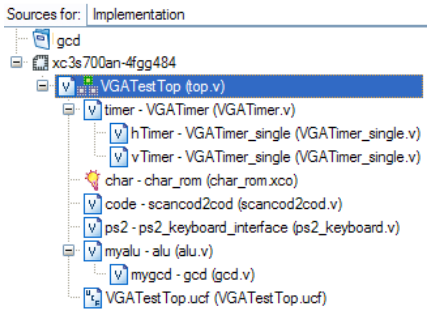


Fig.7.3. Structura ierarhică a modulelor Verilog.

7.2.1.1. Unitatea aritmetica-logica generică module alu(clk, clr, x, y, z, op).

Unitatea aritmetică-logica generică module alu (fig.7.4.) a fost concepută pentru a simplifica cuplarea la platformă a unui număr de 1– 4 sisteme numerice (procesoare aritmetice) diferite, care pot fi selectate cu ajutorul intrării **op.**

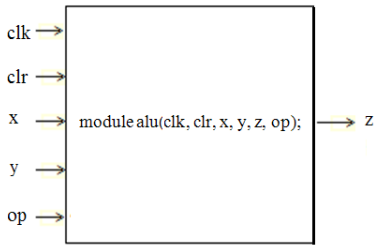


Fig.7.4. Schema bloc a componentei module alu(clk, clr, x, y, z, op);

```
////////////////////////////////////////////////////////////////////////////////
                `timescale 1ns / 1ps
               module alu(clk, clr, x, y, z, op);
                parameter width = 32;
                  input clk;
                  input clr;
                  input [width-1:0] x;
                  input [width-1:0] y;
                  output [width-1:0] z;
                  input [1:0] op;
                            reg [width-1:0] z;
                            wire done;
                            wire [15:0] gcdout;

                            gcd mygcd(clk,x,y,clr,gcdout,done);

                            always @(posedge clk)

                          case (op)
                              2'b00 : z = gcdout;
                                        2'b01 : z = y;
                                        2'b10 : z = x;
                                        2'b11 : z = y;
                                        default: z = 0;
                            endcase
               endmodule
////////////////////////////////////////////////////////////////////////////////
```

Implementarea automată, a modulului Verilog alu(clk, clr, x, y, z, op), cu ajutorul aplicației ISE 10.1.01 – WebPack, conduce la schema bloc prezentată în fig.7.5.

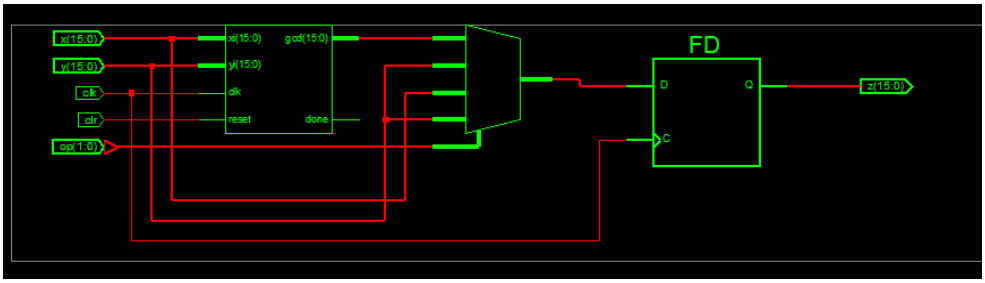


Fig.7.5. Schema bloc a implementării modulului alu(clk, clr, x, y, z, op), cu ajutorul aplicației ISE 10.1.01 – WebPack.

Modulul alu.v, la nivel structural, conține:

- o instanță mygcd a modulului gcd.v;

- un multiplerxor, cu 4 intrări, care poate colecta ieşirile z ale altor instanțe selectate prin intrarea op;
- un registru de ieşire, pentru rezultat.

7.2.1.2. **Modulul** cmmdc (gcd.v).

Modulul cmmdc a fost implementat pe platformă în maniera în care nu mai a apărut semnalul start, modulul fiind denumit, pe scurt, gcd.v (greatest common divizor):

```
////////////////////////////////////////////////////////////////////////////////
                module gcd(clk,xi,yi,reset,gcd,done);
                 input clk,reset;
                 input [15:0] xi,yi;
                 output done;
                 output [15:0] gcd;

                 reg [15:0] gcd;      // rezultatul se obține la ultima iterație;
                 reg done;            //  done = 1, când se termină execuția algoritmului;
                 always @(posedge clk) begin: gcd_loop
                  reg [15:0] x,y;     // registrele interne stochează stările intermediare;
                  if (reset)          // daca reset =1, se încarcă registrele x, y şi se anulează indicatorul done;
                    begin x <= xi; y <= yi; done <= 0; end
                  else if (!done) begin
                    if (x == y)
                      begin gcd <= x; done <= 1; end
                    else if (x > y) x <= x - y;
                    else y <= y - x;
                  end
                 end
                endmodule
////////////////////////////////////////////////////////////////////////////////
```

Implementarea automată, a modulului Verilog cmmdc (gcd.v), cu ajutorul aplicației ISE 10.1.01 – WebPack, conduce la schema bloc prezentată în fig.7.6.

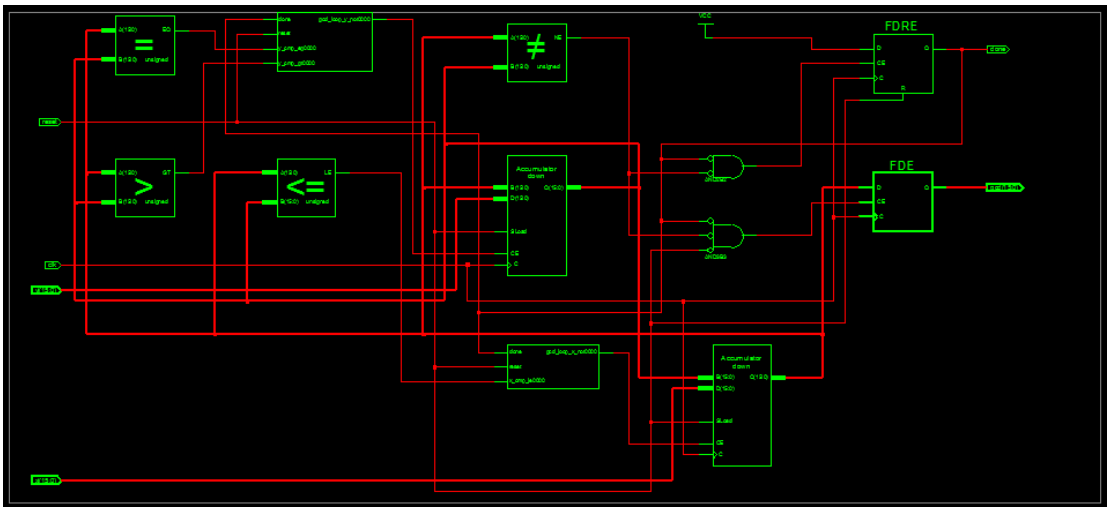


Fig7.6. Schema bloc a implementării modulului cmmdc (gcd.v), cu ajutorul aplicației ISE 10.1.01 – WebPack.

Modulul gcd.v, la nivel structural, conține:

- 4 blocuri comparatoare (=, >, ≠, <=);
- 2 blocuri de tip registru/acumulator, pentru operanzii x şi y;
- 2 circuite de implicate în operațiile: x <= x - y; y <= y - x;
- 1 registru de ieşire pentru rezultat gcd (cmmdc);
- 1 bistabil pentru semnalul de stare done (gata).

În fig.7.7. sunt ilustrate exemple de circuite întâlnite în schema din fig.7.6, cu specificarea porturilor de intrare şi iesire: (a) şi (b) - comparatoare, (c) – registru/acumulator, (d) – circuit implicat în operația: x <= x - y, (e) – registrul rezultatului, (f) - bistabilul de stare:

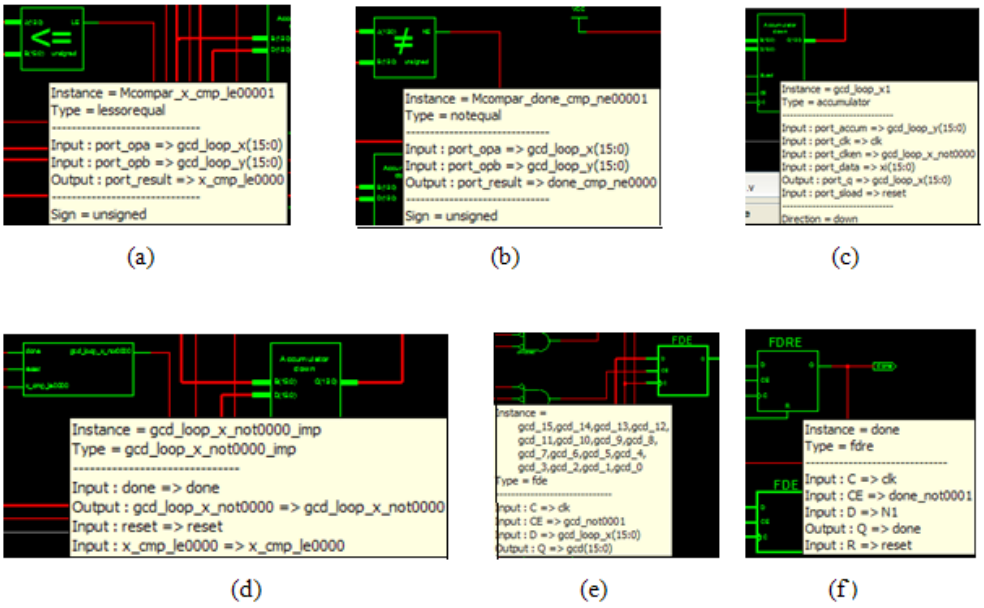


Fig.7.6. Exemple de circuite întâlnite în schema din fig.7.5, cu specificarea porturilor de intrare şi ieşire.

Unitatea aritmetică-logică generică module alu(clk, clr, x, y, z, op) a fost concepută pentru a simplifica cuplarea la platformă a unui numar de 1 – 4 sisteme numerice (procesoare aritmetice) diferite, care pot fi selectate cu ajutorul intrarii op:

```
`timescale 1ns / 1ps
module alu(clk, clr, x, y, z, op);
 parameter width = 32;
   input clk;
   input clr;
   input [width-1:0] x;
   input [width-1:0] y;
   output [width-1:0] z;
   input [1:0] op;
            reg [width-1:0] z;
            wire done;
            wire [15:0] gcdout;

            gcd mygcd(clk,x,y,clr,gcdout,done);

            always @(posedge clk)

         case (op)
            2'b00 : z = gcdout;
                     2'b01 : z = y;
                     2'b10 : z = x;
                     2'b11 : z = y;
                     default: z = 0;
             endcase
endmodule
```

Modulul care se instanțiază trebuie sa respecte numărul, semnificațiile şi ordinea parametrilor în declarația de modul. Selectarea modulului, care se instanțiază, se realizează cu ajutorul a două comutatoare de pe plachetă, care asigură intrarea op. În cazul particular dat, modulul gcd este instanțiat pentru op = 2'b00.

În Tab.7.2. se prezintă situația utilizării dispozitivelor disponibile în circuitul FPGA xc3s700an4fgg484.

Tab.7.2. Situația utilizării dispozitivelor disponibile în circuitul FPGA

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	174	11,776	1%	
Number of 4 input LUTs	345	11,776	2%	
Logic Distribution				
Number of occupied Slices	237	5,888	4%	
Number of Slices containing only related logic	237	237	100%	
Number of Slices containing unrelated logic	0	237	0%	
Total Number of 4 input LUTs	388	11,776	3%	
Number used as logic	344			
Number used as a route-thru	43			
Number used as Shift registers	1			
Number of bonded IOBs	29	372	7%	
IOB Flip Flops	2			

Cu toate că, pe lângă modulul gcd, au fost implementate şi modulele de interfață pentru tastatura PS/2 şi monitorul VGA, proiectul complet a solicitat o parte infimă din resursele hardware disponibile pe circuitul FPGA (a se vedea coloana 3, din Tab.5.2.)

7.3. Implementarea unor procesoare asociative orientate pe aplicații specifice.

După cum s-a văzut în capitolul 3, existența tehnologiei hardware - FPGA, cât şi a uneltelor de descriere, simulare şi implementare a algoritmilor direct in hardware oferă posibilitatea dezvoltarii şi experimentării unor noi algoritmi, în particular a algoritmilor asociativi.

În cele ce urmează se vor prezenta două procesoare asociative dedicate unor aplicații specifice. În prima aplicație se pune problema căutarii unor obiecte într-un perimetru dat, iar în a doua se cere găsirea unor cluster-e de de valori, care gravitează în jurul unei valori date. Aplicațiile prezentate au un rol demonstrativ-metodologic.

7.3.1. Procesor asociativ pentru căutarea obiectelor aflate într-un perimetru dat.

Teoria legată de operarea procesorului asociativ, pentru căutarea obiectelor aflate într-un perimetru dat a fost prezentată într-un curs anterior.

Pentru cazul particular, de față, se consideră un set *S* de 8 puncte stocate sub formă de octeți într-un fişier CMASOCI.txt : 9a 8b 0c 0d ff a0 a3 86. Punctele sunt plasate într-un spațiu 2D, fiecare octet fiind constituit din doua tetrade: tetrada din stânga şi tetrada din dreapta, care corespund coordonatelor *y* şi *x* ale punctelor specificate în fisierul CMASOCI.txt. Utilizatorul poate fi interesat să afle care dintre punctele specificate în fişier, prin coordonatele lor, se află în interiorul unui dreptunghi definit de abscisele 05, 0c şi de coordonatele 70 şi e0. Inspectând rezultatele simularii obținute prin programul Verilog de mai jos, se poate constata că în interiorul dreptunghiului definit sunt plasate 3 puncte, de coordonate: 9a, 8b şi 86.

7.3.1.1. Simulare

```
/*Procesor Asociativ Multicomparand 1 (PAMC_1), parametrizat ca lungime cuvântului "n" şi
capacitate a tabloului asociativ TA - "m". Pe baza comparanzilor Cj şi a măstilor Mj, procesorul
generează respondenții pe m biți Rjg[i] şi Rjl[i], care exprimă relațiile ">" şi "<" între vectorii
"Mj&Cj" şi TA[i]&Mj"; */
module PAMC_1;
        parameter n=8;
        parameter m=8;
        integer i;
        reg [n:1] TA[1:m]; // tabloul asociativ în care se vor stoca datele;
        reg [1:m] R1g,R1l,R2g,R2l,Rz; // rezultatele comparațiilor;
        reg [n:1] M1,C1,M2,C2,C3,C4; // măştile şi comparanzii;
initial begin
        i=1;
        R1g='h00; R1l='h00; R2g='h00; R2l='h00; Rz='h00; // inițializare rezultate comparații;
        M1='h0f;  M2='hf0; // inițializare măsti;
        C1='h05; C2='h0c; C3='h70; C4='he0; // inițializare comparanzi;
           $readmemh("CMASOCI.txt",TA);
            end

   always
           begin
             while(i<m+1)
               begin
                 R1g[i]=((M1&C1)<(TA[i]&M1)); // Rezultate pentru comparația "> 05";
```

```
            R1l[i]=((M1&C2)>(TA[i]&M1));  // Rezultate pentru comparația “< 0c”;
            R2g[i]=((M2&C3)<(TA[i]&M2)); // Rezultate pentru comparația “> 70”;
            R2l[i]=((M2&C4)>(TA[i]&M2));  // Rezultate pentru comparația “< e0”;

            Rz[i] = R1g[i]&R1l[i]&R2g[i]&R2l[i];  // Rezultate finale;

$display($time,, "M1=%h C1=%h M2=%h C2=%h C3=%h C4=%h TA[i]=%h R1g[i]=%b R1l[i]=%b
R2g[i]=%b R2l[i]=%b Rz[i]=%b i=%0d", M1,C1,M2,C2,C3,C4,TA[i],R1g[i],R1l[i],R2g[i],R2l[i],Rz[i],i);
      i=i+1;
   end
  $stop;
end
endmodule
```

```
Entering Phase II...
Entering Phase III...
2 warnings in compilation
No errors in compilation
Top-level modules:
   MASMC
C1> .
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=9a R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=1
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=8b R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=2
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=0c R1g[i]=1 R1l[i]=0 R2g[i]=0 R2l[i]=1 Rz[i]=0 i=3
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=0d R1g[i]=1 R1l[i]=0 R2g[i]=0 R2l[i]=1 Rz[i]=0 i=4
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=ff R1g[i]=1 R1l[i]=0 R2g[i]=1 R2l[i]=0 Rz[i]=0 i=5
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=a0 R1g[i]=0 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=0 i=6
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=a3 R1g[i]=0 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=0 i=7
M1=0f C1=05 M2=f0 C2=0c C3=70 C4=e0 TA[i]=86 R1g[i]=1 R1l[i]=1 R2g[i]=1 R2l[i]=1 Rz[i]=1 i=8
```

Rezultatele obținute prin simulare arată că punctele situate în interiorul dreptunghiului definit mai sus au coordonatele: 9a, 8b şi 86.

7.3.1.2. Implementare.

Pentru implementarea procesorului s-a rescris modulul PAMC_1, utilizând facilitățile oferite de Verilog 2001, în legătură cu posibilitatea de paralelizare a ciclurilor, folosind construcția *generate.*

În esență construcția *generate*, din Verilog 2001, “desface” o construcție de control *for* într-un număr de fire paralele, implementate în hardware, egal cu numarul ciclurilor *for*. Astfel, complexitatea algoritmului se reduce cu un factor egal cu numărul ciclurilor *for.* În cazul de față există un fir de execuție hardware pentru fiecare cuvânt/punct, dintre cele *m* stocate în memoria TA.

Descrierea Verilog a modulului PAMC_1, în vederea simulării, în noua variantă, este dată mai jos:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
```

```
// Company: Quadrilogic
// Engineer: Iacob Petrescu
// Create Date:    10:09:37 11/24/2008
// Design Name:
// Module Name:    PAMC_1
// Project Name:
// Target Devices: xc2s200e-6pq208
// Tool versions: Xilinx  ISE Design Suite 9.203i
//////////////////////////////////////////////////////////////////////////////////
module PAMC_1(M1,M2,C1,C2,C3,C4,TA,R1g,R1l,R2g,R2l,Rz);
  parameter n=8;
   parameter m=8;
   input [n:1] M1, M2;
   input [n:1] C1, C2, C3, C4;
   input [m*n:1] TA;
   output [1:m] R1g, R1l, R2g, R2l;
   output [1:m] Rz;
   reg [1:m] R1g, R1l, R2g, R2l;
   wire [1:m] Rz;
   assign Rz = R1g&R1l&R2g&R2l;
   genvar i;
   generate
    for (i=1; i <= m; i=i+1)
    begin: slice
     always @ *
       begin
        R1g[i]=((M1&C1)<(TA[(m-i+1)*n:(m-i)*n+1]&M1));
        R1l[i]=((M1&C2)>(TA[(m-i+1)*n:(m-i)*n+1]&M1));
        R2g[i]=((M2&C3)<(TA[(m-i+1)*n:(m-i)*n+1]&M2));
        R2l[i]=((M2&C4)>(TA[(m-i+1)*n:(m-i)*n+1]&M2));
             end
    end
   endgenerate
endmodule
module tbw_PAMC;
            parameter n=8;
            parameter m=8;
            reg [m*n:1] D;
            wire [1:m] R1g,R1l,R2g,R2l,Rz;
            reg [n:1] M1,C1,M2,C2,C3,C4;
                    PAMC_1  uut(M1,M2,C1,C2,C3,C4,TA,R1g,R1l,R2g,R2l,Rz);
    initial begin
```

```
        M1='h0f; M2='hf0;
        C1='h05; C2='h0c; C3='h70; C4='he0;
        TA ='h9a_8b_0c_0d_ff_a0_a3_86;
  $stop;
end
endmodule
```

Pentru implementarea propriu-zisă s-a renunțat la modulul tbw_PAMC şi s-au adăugat, la descrierea modulului PAMC_1, modulele de interfață pentru tastatura PS/2 şi pentru display-ul VGA. Cu ajutorul tastaturii se asigură introducerea unor noi valori pentru comparanzi, măşti şi conținutul tabloului asociativ. Rezultatele execuției algoritmului de către procesor se vizualizează pe ecranul display-ului.

Implementarea procesorului s-a realizat cu ajutorul platformei Xilinx ISE Design Suite 10.1.03i,, având ca țintă circuitul FPGA Xilinx xc2s200e-6pq208, cu 200.000 porți. Standul experimental este prezentat în fig. 7.7.

Fig. 7.7. Standul experimental pentru procesorul PAMC_1.

Rezumatul raportului referitor la utilizarea resurselor circuitului FPGA Xilinx xc2s200e-6pq208, furnizat de platforma Xilinx ISE Design Suite 10.1.03i, arată că în total s-au folosit un numar de 19.961 porți echivalente, pentru implementarea modulelor: PAMC_1, interfața cu tastatura PS/2 şi interfața cu display-ul VGA.

Performanța procesorului poate fi estimată plecând de la constatarea că algoritmul operează într-un singur ciclu de ceas, în cadrul modelului de calcul de tip “transferul datelor inițiale de la registrul sursă la registrul destinație, prin intermediul rețelei logice, care asigură prelucrarea datelor”. Platforma Xilinx ISE Design Suite 10.1.03i raportează o intârziere maximă în calea/rețeaua

combinațională, $T_{comb\_path\_delay}$, egală cu 15,56 ns. Datele de catalog, pentru circuitul xc2s200e-6pq208, indică următoarele valori ale parametrilor T_{CtoQ} maxim şi T_{setup} minim: $T_{CtoQ} = 0{,}7$ns şi $T_{setup} = 3{,}1$ ns. În aceste condiții o estimare [66] a perioadei ceasului sistemului, T_{clock}, este dată de expresia (7.1):

$$T_{clock} \geq T_{CtoQ} + T_{comb\_path\_delay} + T_{setup} \qquad (7.1)$$

Introducând valorile numerice corespunzătoare, în relația (7.1) rezultă $T_{clock} \geq 19{,}36$ ns, ceea ce corespunde cu situația reală a unui ceas cu frecvența de 50 MHz, ($T_{clock} = 20$ ns). În literatura de specialitate nu este semnalată o implementare mai rapidă a algoritmului de mai sus.

7.3.2. Procesor asociativ pentru căutarea obiectelor caracterizate prin valori grupate în cluster-e în jurul unor valori date.

Procesorul asociativ, care implementeaza căutarea obiectelor caracterizate prin valori grupate în cluster-e în jurul unor valori date, manipulează informații structurate ca tripleți de forma:

$$\mathcal{T} = < y, x, z > \qquad (7.2)$$

în care *y*, *x*, *z* sunt întregi pozitivi corespunzători coordonatelor obiectelor în 2D, iar z reprezintă o valoare numerică asociată punctului de coordonate (*y*, *x*).

În cazul de față se consideră un set S de 16 obiecte, fiecare obiect având asociat un cuvânt structurat ca un triplet de forma (7.2). Cele 16 cuvinte sunt stocate într-un fisier CTA1.txt al cărui conținut în hex este dat mai jos:

```
//CTA1.txt
01a11_02a12_03a13_04534_56565_57776_58f27_59f28_6af29_69f2a_78f2b_9d82c_aa35d_
bb46e_ccabf_ddff0
```

Se poate constata faptul că cele două cifre hex din stânga reprezintă coordonatele *y* şi *x* ale punctelor, iar ultimele trei cifre corespund valorii numerice asociate punctului dat.

S-a presupus că utilizatorul este interesat de gruparea valorilor sub forma de cluster-e în jurul valorilor a1x şi f2x, unde x (notație Verilog pentru valoare necunoscută) poate lua valori cuprinse între 'h0 şi 'hf. Astfel, "diametrul" unui cluster poate fi cel mult de 'hf.

7.3.2.1. Simulare.

Simularea la nivel comportamental a procesorului asociativ pentru problema formulată mai sus s-a efectuat în doua variante şi cu două simulatoare Verilog diferite. În prima variantă s-a utilizat construcția de control while, pentru a descrie ciclurile necesare calculului biților succesivi ai rezultatelor Re1 şi Re2, cât şi simulatorul Veriwell. În varianta a doua s-a facut apel la construcția generate, introdusă in versiunea Verilog 2001, construcție care permite paralelizarea ciclurilor, prin reducerea acestora la fire de execuție paralelă a corpului ciclului. Simularea s-a realizat cu ajutorul aplicației Modelsim XE III 6.1e.

Simularea comportamentala in Veriwell

/*Procesor Asociativ Multicomparand 2 (PAMC_2), parametrizat ca lungime a cuvântului "*n*" şi capacitate a tabloului asociativ TA - "*m*". Pe baza comparanzilor Cj şi a măstii Mj, procesorul generează respondenții pe m biți Re1[i] şi Re2[i], care exprimă relația "=="între vectorii "Mj&Cj" şi Mj &TA[i]" */

```
module PAMC_2;
parameter n=20;  // număr biți cuvânt TA;
parameter m=16; // număr cuvinte în TA;
integer i;
reg[n:1]TA[1:m];  // declarație tablou asociativ în care sunt stocate datele inițiale;
reg[n:1]TR1[1:m]; // declarație tablou memorie rezultate comparand C1;
reg[n:1]TR2[1:m]; // declarație tablou memorie rezultate comparand C2;
reg[n:1]C1,C2;       // declarații comparanzi C1, C2;
reg[n:1]M1;          // declarație mască M1;
reg[1:m]Re1,Re2;  // declarație registre rezultate comparații R1, R2;
initial
begin
 i=1;
 Re1='h0000;  Re2='h0000;   // inițializare registre R1, R2;
 C1='h00a10; C2='h00f20;    // inițializare comparanzi C1, C2;
 M1='h00ff0;                // inițializare mască M1;
 $readmemh("CTA1.txt",TA); // stocare date în tabloul asociativ TA;
end
always
 begin
  while(i<m+1)
   begin
    Re1[i]=((M1&C1) == (M1&TA[i]));   // evaluare secvențială a biților rezultatului Re1;
          if(Re1[i]==1)
      begin
```

```
          TR1[i]=TA[i];
        end
       else
          begin
           TR1[i]='hxxxxx;
         end
      Re2[i]=((M1&C2) == (M1&TA[i]));    // evaluare secvențială a biților rezultatului Re2;
             if(Re2[i]==1)
        begin
         TR2[i]=TA[i];
        end
       else
          begin
           TR2[i]='hxxxxx;
         end

   $display("M1=%h C1=%h C2=%h TA[i]=%h Re1[i]=%h TR1[i]=%h Re2[i]=%h TR2[i]=%h
i=%0d",M1,C1,C2,TA[i],Re1[i],TR1[i],Re2[i],TR2[i],i); // afişare rezultate;
     i=i+1;
    end
  $stop;
 end
endmodule
```

Rezultatele simulării în Veriwell sunt prezentate în tabelul 7.3., care conţine masca M1, comparanzii C1 şi C2, tabloul asociativ TA, respondenții Re1, Re2 şi fişierele TR1, TR2 cu obiectele ce formează Cluster-ul1 şi Cluster-ul2.

Tab. 7.3. Rezultatele simularii in Veriwell.

În fig. 7.8. sunt ilustrate grafic punctele care alcătuiesc cele două cluster-e: Cluster-ul1 şi Cluster-ul2. Se poate face observația că punctele în jurul căroara se caută alte puncte, care ar intra în

compunerea unor cluster-e, s-ar putea alege ca fiind punctele având asociate valoarea maximă şi/sau valoarea minimă.

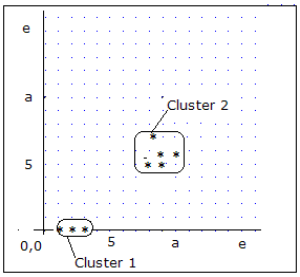


Fig.7.8.. Reprezentarea 2D a celor doua cluster-e

Simularea comportamentală în Verilog 2001.

Simularea comportamentală în Verilog 2001 s-a realizat în două variante. În prima variantă tabloul asociativ s-a inițializat cu ajutorul unui fişier distinct "CTA1.txt", utilizând construcția: $readmemh("CTA1.txt",TA); . Cea de-a doua variantă a presupus inițializarea lui TA sub forma unui vector cu 320 de biți structurați ca tetrade 'h, în cadrul unei construcții initial:

```
TA='h01a11_02a12_03a13_04534_56565_57776_58f27_59f28_6af29_69f2a_78f2b_9d82c_aa35d_bb46e_c
cabf_ddff0;
```

Prima varianta:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:  Quadrilogic
// Engineer:    Iacob Petrescu
// Create Date:    20:05:21 11/30/2008
// Design Name:
// Module Name:    PAMC_2
// Project Name:
// Target Devices: xc2s200e-6pq208
// Tool versions: ISE Design Suite 10.1.03i
//////////////////////////////////////////////////////////////////////////////////

module PAMC_2; // generate în loc de while
parameter n=20;  // număr biți cuvânt TA;
parameter m=16; // număr cuvinte în TA;
integer i;
reg[n:1]TA[1:m];   // declarație tablou asociativ în care sunt stocate datele inițiale;
```

```
reg[n:1]TR1[1:m]; // declarație tablou memorie rezultate comparand C1;
reg[n:1]TR2[1:m]; // declarație tablou memorie rezultate comparand C2;
reg[n:1]C1,C2;       // declarații comparanzi C1, C2;
reg[n:1]M1;          // declarație mască M1;
reg[1:m]Re1,Re2;  // declarație registre rezultate comparații R1, R2;
initial
begin
 i=1;
 R1='h0000;  R2='h0000;   // inițializare registre R1, R2;
 C1='h00a10; C2='h00f20; // inițializare comparanzi C1, C2;
 M1='h00ff0;                    // inițializare mască M1;
 $readmemh("CTA1.txt",TA); // stocare date în tabloul asociativ TA;
end

   genvar j,k;
   generate    // construcția generate pentru calculul respondenților Re1 şi Re2;
    for (j=1; j <= m; j=j+1)
    begin: Re_slice
     always @ *
       begin
           Re1[j]=((M1&C1) == (M1&TA[j]));  // calculul respondentului Re1;
           Re2[j]=((M1&C2) == (M1&TA[j]));   // calculul respondentului Re2;
       end
    end
   endgenerate

   generate      // construcția generate pentru selectarea elementelor cluster-elor 1 şi 2;
    for (k=1; k <= m; k=k+1)
    begin: TR_slice
     always @ *
       begin
         TR1[k] = (Re1[k])? TA[k]: 'hxxxxx;  // selectarea elementelor Cluster-ului1 din tabloul asociativ TA;
         TR2[k] = (Re2[k])? TA[k]: 'hxxxxx;  // selectarea elementelor Cluster-ului2 din tabloul asociativ TA;
       end
    end
   endgenerate

always
 begin
  while(i<m+1)
   begin
  $display("M1=%h C1=%h C2=%h TA[i]=%h Re1[i]=%h TR1[i]=%h Re2[i]=%h TR2[i]=%h
i=%0d",M1,C1,C2,TA[i],Re1[i],TR1[i],Re2[i],TR2[i],i);    // afişare rezultate;
    i=i+1;
    end
 $stop;
```

```
  end
endmodule
```

Varianta a doua:

```
module PAMC_2(M1,C1,C2,TA,Re1,Re2,TR1,TR2);
  parameter n=20;
   parameter m=16;
   input [n:1] M1;
   input [n:1] C1, C2;
   input [m*n:1] TA;
   output [1:m] Re1, Re2;
   output [m*n:1] TR1,TR2;
   reg [1:m] Re1, Re2;
   reg [m*n:1] TR1,TR2;

   genvar j,k;   // construcția generate pentru calculul respondenților Re1 şi Re2;
   generate
    for (j=1; j <= m; j=j+1)
    begin: Re_slice
     always @ *
       begin
// selectarea elementelor Cluster-ului1 din tabloul asociativ TA;
           Re1[j]=((M1&C1) == (M1&TA[(m-j+1)*n:(m-j)*n+1])); //calculul respondentului Re1;
           Re2[j]=((M1&C2) == (M1&TA[(m-j+1)*n:(m-j)*n+1]));  // calculul respondentului Re2;
       end
    end
   endgenerate
   generate
    for (k=1; k <= m; k=k+1)
    begin: TR_slice
     always @ *
       begin
// selectarea elementelor cluster-elor 1 si 2 din tabloul asociativ TA;
         TR1[(m-k+1)*n:(m-k)*n+1] = (Re1[k])? TA[(m-k+1)*n:(m-k)*n+1]: 'hxxxxx;
         TR2[(m-k+1)*n:(m-k)*n+1] = (Re2[k])? TA[(m-k+1)*n:(m-k)*n+1]: 'hxxxxx;
       end
    end
   endgenerate
endmodule

module tbw_PAMC_2;  // “test bench” PAMC_2;
            parameter n=20;
            parameter m=16;
            reg [m*n:1] TA;
            wire [1:m] Re1,Re2;
```

```
        wire [m*n:1] TR1,TR2;
        reg [n:1] M1,C1,C2;

        PAMC_2 #(20,16) uut(M1,C1,C2,TA,Re1,Re2,TR1,TR2); // se instanțiază PMAC_2;
initial begin
        C1='h00a10; C2='h00f20; // inițializare comparanzi C1, C2;
        M1='h00ff0;             // inițializare mască M1;
TA='h01a11_02a12_03a13_04534_56565_57776_58f27_59f28_6af29_69f2a_78f2b_9d82c_aa35d_bb46e_c
cabf_ddff0; // inițializează tabloul asociativ TA sub forma unui vector binar cu 320 de biți;
  $stop;
end
endmodule
```

In fig. 7.9. se prezintă în hexazecimal rezultatele simulării (aplicația Modelsim XE III 6.1e.) comportamentale a PAMC_2. În ordine sunt afişate TA, Re1, Re2, TR1, TR2, M1, C1 şi C2:

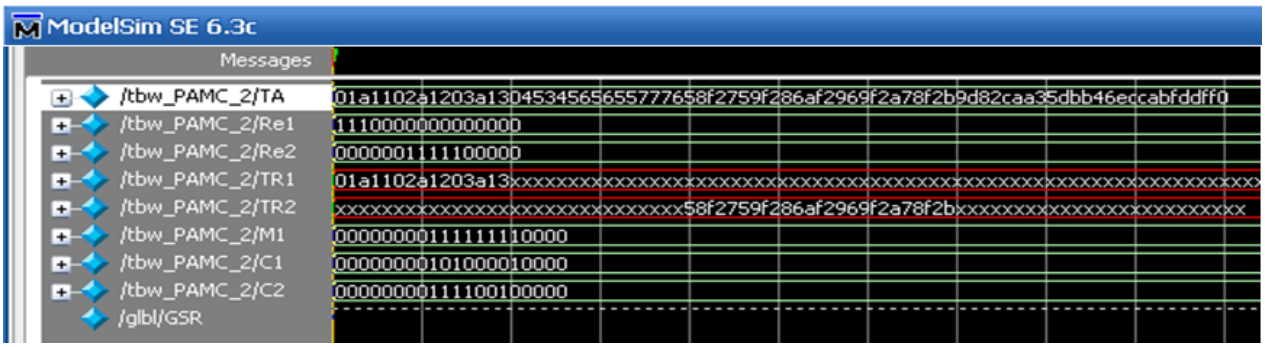


Fig.7.9. Rezultatele simulării comportamentale a PAMC_2.

7.3.2.2 Implementare.

Pentru implementarea procesorului PAMC_2 s-a folosit platforma de test descrisă anterior.

În vederea simplificării introducerii datelor, acestea au fost considerate ca având un suport hardware sub forma de registre inițializate. Rezultatele execuției algoritmului de către procesor se vizualizează pe ecranul display-ului.

Implementarea procesorului s-a realizat cu ajutorul platformei Xilinx ISE Design Suite 10.1.03i, având ca țintă circuitul FPGA Xilinx xc2s200e-6pq208, cu 200.000 porți. Standul experimental este prezentat în fig. 7.10.

Rezumatul raportului (Tab.7.4.), referitor la utilizarea resurselor circuitului FPGA Xilinx xc2s200e-6pq208, furnizat de platforma ISE Design Suite 10.1.03i, arată că în total s-au folosit un număr de 1875 porți echivalente (LUT-uri cu 4 intrari), din cele 4704 disponibile, adică 39%, pentru implementarea modulelor: PAMC_2, interfața cu tastatura PS/2 şi interfața cu display-ul VGA.

```
`timescale 1ns / 1ps
 module VGATestTop(clk, AorB, sw, ps_data, ps_clk, hsync, vsync, r2, r1, r0, g2, g1, g0, b2, b1, b0, led);
   input clk;
            input AorB;
            input [2:0] sw;
            inout ps_data,  ps_clk;
   output hsync, t vsync;
   output r2, r1, r0, g2, g1, g0, b2, b1, b0;
            output [7:0] led;
            localparam width = 380;
            reg r2, r1, r0, g2, g1, g0, b2, b1, b0;
   wire active;
   wire [11:0] x,  y;
   wire [7:0] led;
 //M1_C1_C2_TA
            reg [width-1:0]
A='h00ff0_00a10_00f20_01a11_02a12_03a13_04534_56565_57776_58f27_59f28_6af29_69f2a_78f2b_9d8
2c_aa35d_bb46e_ccabf_ddff0;
            wire [width-1:0] B, C;
            wire [0:3] display_byte,display;
            wire [2:0] dtx,dty;
            wire [7:0]  tmp;
            wire [8:0] addrc;
            wire [0:7] doutc;
            wire bord,number;
            wire new_scancode,new_data;
            wire [9:0] dsb0,dsb1,dsb2,dsb3;
            wire [7:0] scancode;
            wire [3:0] data_from_ps2;
            reg saved;
           reg rx_read;
            wire rx_extended,rx_released,rx_shift_key_on,tx_write_ack_o,tx_error_no_keyboard_ack;
            wire [7:0] rx_ascii;
   VGATimer #(800,64,120,56, 600,23,6,37, 1) timer (clk, hsync, vsync, active, x, y);
            char_rom char(addrc,~clk,doutc);
            scancod2cod code(scancode,rx_released,data_from_ps2,new_data);
 ps2_keyboard_interface
ps2(clk,1'b0,ps_clk,ps_data,rx_extended,rx_released,rx_shift_key_on,scancode,rx_ascii,new_scancode,rx_r
ead,8'd0,1'b0,tx_write_ack_o,tx_error_no_keyboard_ack);
//   alu #(width) myalu(clk, A, B, C, sw);
//   MASOCI #(width, width) memasoc(A,B,sw,C);
//    PAMC_2 #(20,16) pamc(M1,C1,C2,TA,Re1,Re2,TR1,TR2);
      PAMC_2 #(20,16)
pamc(A[379:360],A[359:340],A[339:320],A[319:0],B[379:364],C[379:364],B[319:0],C[319:0]);
```

```
        assign B[363:320]=0;
        assign C[363:320]=0;
        always @(posedge clk)
          rx_read <= (new_data)? saved: ~saved;
        always @(posedge clk)
          if (new_data) begin
                    if (~saved) begin
                     saved <= 1;
                     /*if (AorB) begin
                      B[width-1:4] <= B[width-5:0];
                      B[3:0] <= data_from_ps2;
                     end else */ begin
                      A[width-1:4] <= A[width-5:0];
                      A[3:0] <= data_from_ps2;
                     end
                    end else saved <= 0;
      assign addrc = {2'b11,display,dty};
      assign tmp=width/4-1-x[9:3];
      assign dsb0={tmp,2'b00};
      assign dsb1={tmp,2'b01};
      assign dsb2={tmp,2'b10};
      assign dsb3={tmp,2'b11};
      assign display_byte= (y[8]) ? {B[dsb3],B[dsb2],B[dsb1],B[dsb0]}:{A[dsb3],A[dsb2],A[dsb1],A[dsb0]};
      assign display= (y[9]) ? {C[dsb3],C[dsb2],C[dsb1],C[dsb0]}:display_byte;
      assign dtx=x[2:0];
      assign dty=y[2:0];
      assign bord=((x==0)|(x==799)|(y==0)|(y==599)|(y==500));
      assign number = (x<width*2)&&(y[7:0]<8);
      always @(posedge clk)
        begin
      r2 <= active&&(doutc[dtx]&&number);
                r1 <= active&&(doutc[dtx]&&number);
      r0 <= active&&(doutc[dtx]&&number);
      g2 <= active&&(doutc[dtx]&&number);
                g1 <= active&&(doutc[dtx]&&number);
                g0 <= active&&(doutc[dtx]&&number);
      b2 <= active&&(doutc[dtx]&&number);
                b1 <= active&&(doutc[dtx]&&number);
      b0 <= active&&(doutc[dtx]&&number);
        end
       assign led = scancode;
endmodule
```

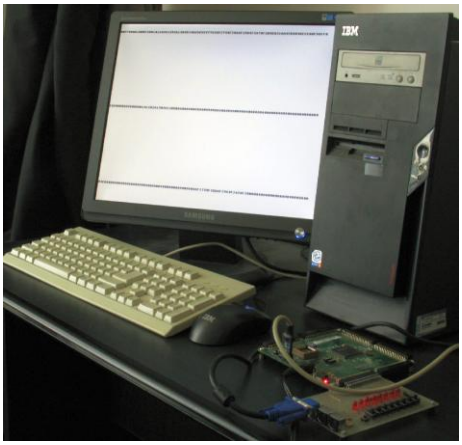


Fig. 7.10. Standul experimental pentru procesorul PAMC_2.

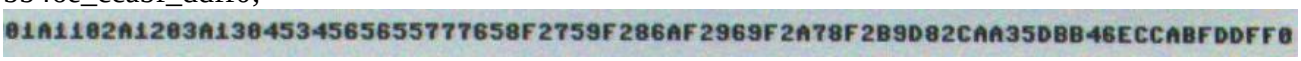
Tab.7.4. Raportul referitor la utilizarea resurselor circuitului FPGA Xilinx xc2s200e-6pq208

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	467	4,704	9%	
Number of 4 input LUTs	1,827	4,704	38%	
Logic Distribution				
Number of occupied Slices	1,167	2,352	49%	
Number of Slices containing only related logic	1,167	1,167	100%	
Number of Slices containing unrelated logic	0	1,167	0%	
Total Number of 4 input LUTs	1,875	4,704	39%	
Number used as logic	1,827			
Number used as a route-thru	47			
Number used as Shift registers	1			
Number of bonded IOBs				
Number of bonded	25	142	17%	
IOB Flip Flops	2			
Number of Block RAMs	1	14	7%	
Number of GCLKs	1	4	25%	
Number of GCLKIOBs	1	4	25%	

.

Pe ecranul display-ului au fost afişate rezultatele execuției algoritmului de căutare în tabloul asociativ TA, a punctelor grupate la o distanță data, ca valoarea asociată, în jurul a doua puncte date. Imaginile informațiilor fotografiate de pe ecran sunt date mai jos:

TA='h01a11_02a12_03a13_04534_56565_57776_58f27_59f28_6af29_69f2a_78f2b_9d82c_aa35d_
bb46e_ccabf_ddff0;



TR1='h01a11_02a12_03a13;

TR2='h58f27_59f28_6af29_69f2a_78f2b

7.4. Concluzii.

Pentru a implementa cu succes noi algoritmi în FPGA, utilizatorul trebuie să aibe în vedere modalitățile de efectuare a operatiilor de I/E, pentru date, comenzi şi stări. Aceste operații pot fi asistate de un PC gazdă sau de un hardware dedicat.

Primul caz este tipic pentru platformele de dezvoltare, care oferă avantaje constând în: universalitate, elasticitate, interfață prietenoasă cu utilizatorul, modificările relativ facile ale proiectului.

Cazul al doilea este recomandat pentru soluțiile stabile şi încorporate, care necesită rate ridicate de transfer.

Implementările de sisteme numerice, bazate pe tehnologiile FPGA, semnifică: absența desenelor circuitelor, a măştilor/şabloanelor, a cheltuielilor pentru fabricația circuitelor orientate pe aplicații (ASIC), costuri medii, cât şi o mare flexibilitate. În comparație cu tehnologiile ASIC, tehnologiile FPGA micşoreaza costurile nerecurente cât şi timpul de plasare pe piață al produsului.