UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Unitate de management pentru controlul unei memorii cache

Nume: Hădărău Elena

Grupa: 30233

CUPRINS

1. INTRODUCERE

Memoria cache este o componentă esențială a unui sistem de calcul și a fost creată pentru a îmbunătăți performanța procesorului și a altor componente ale calculatorului, asigurând un access rapid la datele frecvent utilizate, reducând latența, economisind energie și îmbunătățind gestionarea datelor. Mai exact, ea asigură o comunicare eficientă între unitatea centrală de procesare (CPU) și memoria principală DRAM (Dynamic Random-Access Memory).

Cache-ul este construit folosind o tehnologie de memorie diferită, cunoscută sub numele de Static Random Access Memory (SRAM). SRAM este mai rapid, dar cu o capacitate mai mică și, prin urmare, mai scump decât DRAM. SRAM și DRAM reprezintă două straturi ale ierarhiei de memorie. Astfel, memoria cache este o zonă de stocare rapidă și temporară, de capacitate mai redusă, care stochează date frecvent utilizate, pentru a le face disponibile cu o latență foarte mică sau aproape instantaneu.

Mai mult, pentru utilizarea cât mai eficientă a memoriei cache, se dispune proiectarea unei unități de management pentru controlul memoriei, care se dovedește a fi o parte critică a dezvoltării sistemelor hardware (procesoarele sau sistemele încorporate). Mai precis, unitatea de management pentru controlul unei memorii cache are următoarele roluri: de a se asigura că datele stocate în memoria cache sunt actualizate corect, că sunt eliminate datele mai puțin utile pentru a face loc altora și că memoria cache funcționează în mod optim pentru a îmbunătăți performanța sistemului.

Așadar, aspectele importante în proiectarea unei unități de management pentru controlul unei memorii cache sunt:

- Deciziile de stocare și înlocuire: regăsirea informațiilor se face cu memorie asociativă (alocare dinamică) sau pe baza de adresa (alocare statica)
- Politicile de scriere: scrierea în așteptare (write-back) sau scrierea imediată (write-through)

Conform cerințelor, scopul proiectului este implementarea unei unități de gestiune a memoriei cache, unde memoria cache este mapată direct și politica de scriere este scrierea imediată (write-through).

Interesată fiind de hardware și arhitectura sistemelor de calcul, proiectarea unei unități de control a memoriei cache este o oportunitate excelentă de a explora și înțelege în profunzime aceste aspecte. Mai exact, acest proiect implică o serie de competențe tehnice, cum ar fi proiectarea hardware, programarea „low level" și înțelegerea profundă a arhitecturii de calcul, motiv pentru care, această experiență mă poate ajuta să-mi dezvolt competențele specificate și să dobândesc cunoștințe valoroase. În plus, proiectarea unei unități de management pentru controlul memoriei cache implică o abordare interdisciplinară, în care se combină cunoștințe din domenii precum arhitectura hardware, proiectarea software și ingineria sistemelor. Această diversitate poate fi captivantă și provocatoare.

2. STUDIU BIBLIOGRAFIC

Tehnologia folosită în dezvoltarea proiectului este limbajul VHDL, care este un limbaj de descriere a hardware-ului (Hardware Description Language) utilizat în automatizarea proiectării electronice pentru a descrie sisteme digitale și mixte, cum ar fi FPGA-urile și Circuitele Integrabile. VHDL este un standard internațional reglementat de IEEE.

Avantajele VHDL:

• Permite descrierea/modelarea și verificarea/simularea comportamentului sistemului necesar înainte ca instrumentele de sinteză să traducă design-urile în hardware real.

• Permite descrierea sistemelor concurente; VHDL este un limbaj de flux de date, spre deosebire de limbajele de programare procedurale precum C, limbajul de asamblare, etc., care rulează secvențial, o instrucțiune la un moment dat.

• Orice proiect VHDL este portabil și poate fi mutat pe o altă bază de elemente (de exemplu, VLSI).

În ce privește proiectarea unității de gestiune a memoriei cache, sunt câteva aspecte importante care trebuie luate în considerarea în dezvoltarea sa. Astfel, aspectele care necesită importanță sunt:

- În ce privește politica de înlocuire, fiind vorba de o memorie cache mapată direct, nu este nevoie de abordarea unei politici de înlocuire precum LRU (Least Recently Used), LFU

(Least Frequently Used), deoarece un bloc din memoria principală poate fi scris într-o singură locație în memoria cache. Astfel, în situația în care un bloc nou se mapează la o locație deja scrisă, blocul nou îl va înlocui pe cel anterior.

- Principiul localizării va fi localizarea spațială- se bazează pe faptul că, dacă este nevoie de unele date, în curând va fi nevoie probabil de date aflate în apropierea lor în memorie (principiul vecinătăţii – cache nears). De aceea o linie din memoria Cache corespune unui bloc de cuvinte din memoria principală.
- Politica de scriere folosită este Write-through - Când procesorul emite o cerere de scriere, un cache write-through actualizează datele din cache, executând, eventual, operația de fetch (dacă datele nu se aflau în cache), dar actualizează și datele din memoria principală.

- Avantaj: implementare simplă, asigură ușor consistența dintre memorii: dacă memoria cache și cea principală au mereu același conținut, ele sunt consistente.

- Dezavantaj: atunci când datele sunt actualizate des și la intervale scurte de timp, devine ineficient accesul repetat la memoria principală pentru efectuarea operațiilor de scriere, ducând la scăderea performanței

- În cazul în care datele ce vin de la procesor nu se află în cache, am ales opțiunea de implementare a scrierii în cache: write-allocate, mai exact, se alocă o linie în cache pentru datele respective.

- Maparea directă – deoarece este o dependență unu-la mai multe, memoria principală fiind mai mare și existând mai multe locații care se mapează exact pe același cuvânt din cache, va fi necesar ca în implementarea mecanismului de accesare a conținutului memoriei cache să se facă distincția între acestea cu diferite soluții. Mai precis, se pot calcula următoarele componente:

WORDOFFSET= A mod Cwords, LINEOFFSET = [A div Cwords] mod Clines = nr.bloc în memorie mod Clines

Pentru a face distincția între blocuri mapate pe aceeași linie, în memoria cache fiecare linie va primi o etichetă: TAG = nr.bloc din memoria principală=[A div Cwords] , A – adresa din memoria principală

Cwords – nr.cuvinte/linie (bloc) din memoria cache, Clines – nr.linii din memoria cache

3. ANALIZĂ ȘI DESIGN

3.1. ANALIZĂ

Proiectul implementează funcționalități care permit simularea unui sistem cu memorie cache la o scară redusă față de sistemele reale. Astfel, sunt posibile următoarele operații:

- Scriere de date în memorie, conform unei adrese;

- Citire de date din memorie, conform unei adrese;

- Tratarea unei situații de miss, adică atunci când adresa la care se dorește a face o operație de scriere/citire nu se află în memoria cache și trebuie adusă din memoria principală

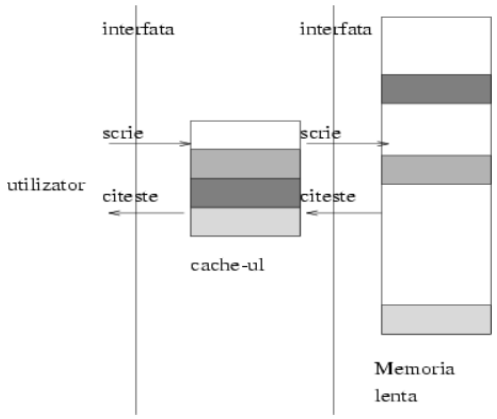


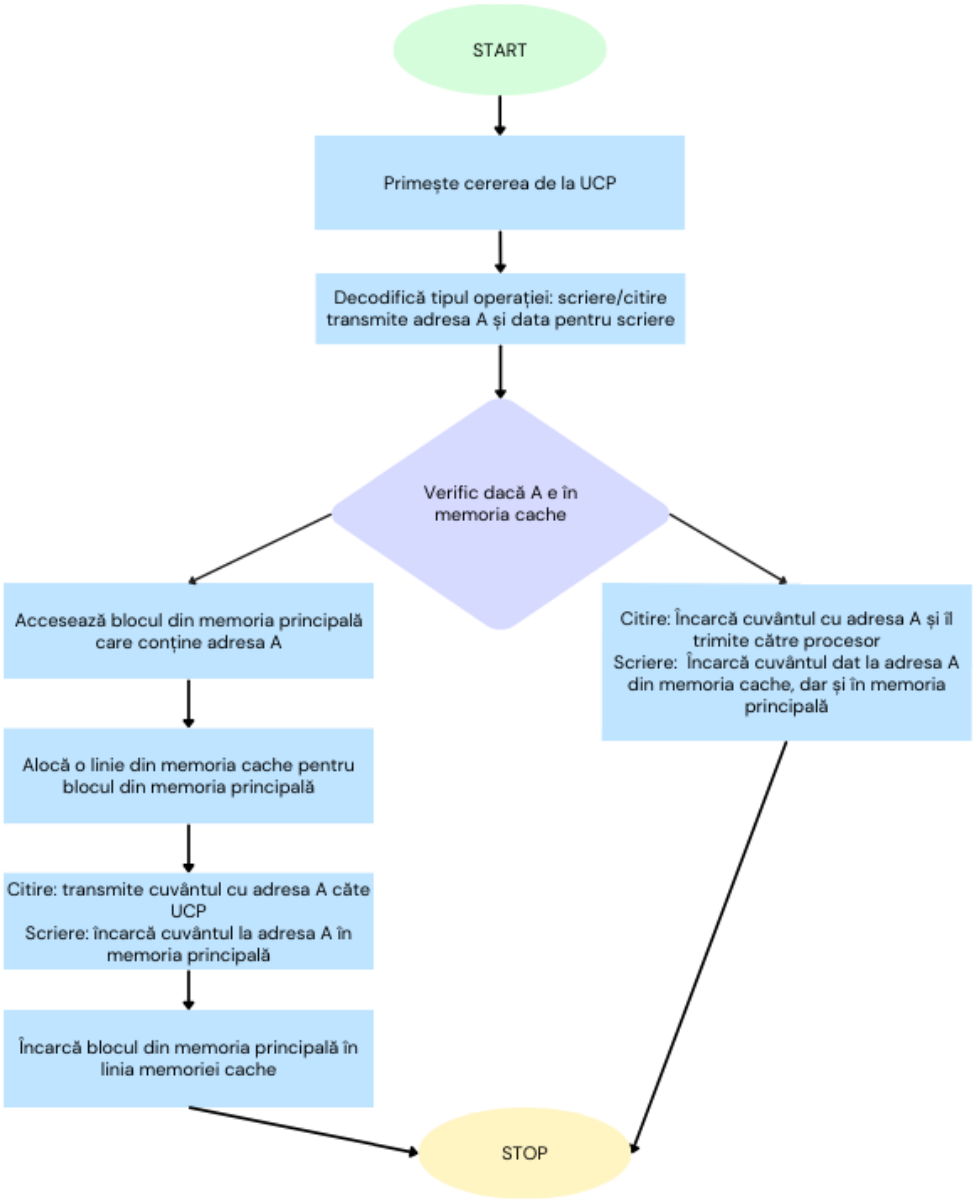
Figure 1: Principiul cache-ului

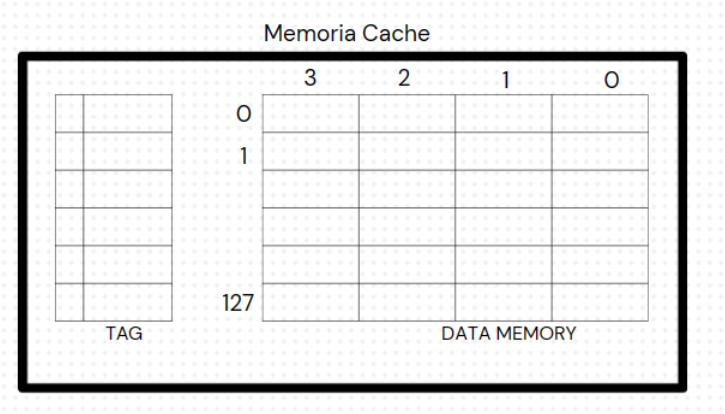
Se consideră că unitatea de gestiune a memoriei cache primește ca date de intrare o adresă care trebuie translatată pentru a identifica locația la care se dorește efectuarea unei operații de citire sau scriere și un cuvânt pentru scriere, în cazul în care se va efectua o operație de scriere.

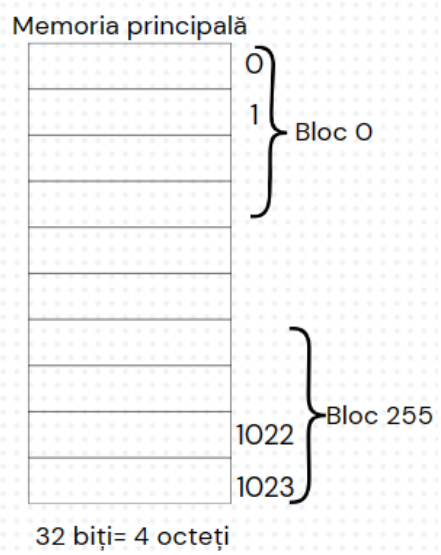
Unitatea de gestiune se ocupă cu gestionarea cererilor venite dinspre procesor. Mai exact, decodifică dacă e o operație de citire sau scriere și transmite adresa și data spre memoria cache și spre memoria principală. Dacă adresa căutată se află în memoria cache, atunci are loc un hit și data cerută este afișată sau scrisă în funcție de cererea venită. Dacă cererea corespunde unei operații de scriere, atunci are loc și scrierea în memoria principală, datorită implementării metodei de scriere imediată. În caz contrar, dacă se detectează un MISS, atunci datele de la adresa respectivă trebuie aduse din memoria principală.

Astfel, pentru a trata un MISS, se accesează memoria principală, se caută blocul care conține adresa cerută, se alocă o linie în memoria cache și, mai apoi, blocul din memoria principală se încarcă în linia alocată din memoria cache. În cazul în care linia la care se mapează locul respectiv, este deja ocupată, datorită caracteristicii de mapare directă a memoriei cache, datele se suprascriu pe acea linie.

Mai mult, după ce blocul de date din memoria principală este adăugat, se efectuează operația de citire sau scriere conform pașilor descriși mai sus.



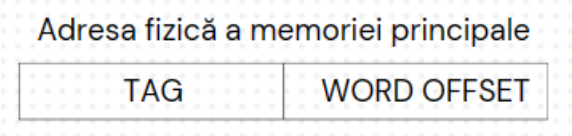




- 4 cuvinte/bloc => 256 blocuri în memoria principală
- O linie din cache corespunde unui bloc de cuvinte din memoria principală => dimensiunea unei linii = 32*4=128 biți

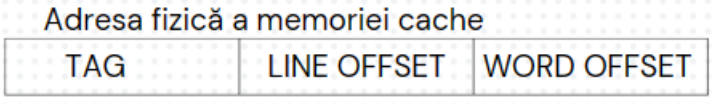
Codificarea adreselor primite ca date de intrare pentru accesarea datelor:

a) Pentru memoria principală



Adresa corespunzătoare memoriei principale este cea care vine de la procesor. Având în vedere ca dimensiunea aleasă pentru memoria principală este de 1024 de cuvinte de câte 32 de biți, adresa de memorie va fi codificată pe 10 biți, rezultând valori între 0 și 1023, corespunzătoare fiecărui cuvânt din memorie. Câmpul TAG este constituit din primii 8 cei mai semnificativi biți și reprezintă numărul blocului din memoria principală, în timp ce câmpul WORD OFFSET este constituit din ultimii 2 biți cei mai nesemnificativi care indică al câtelea cuvânt din bloc mă interesează.

b) Pentru memoria cache

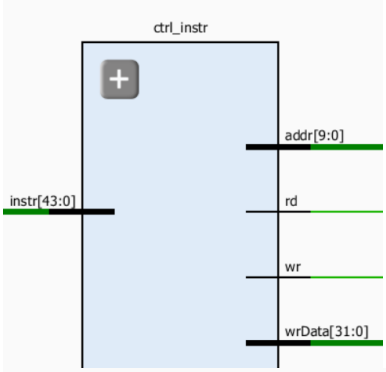


Adresa corespunzătoare memoriei cache este tot pe 10 biți. WORD OFFSET-ul care indică al câtelea cuvânt mă interesează este identic cu cel din memoria principală. În schimb, câmpul TAG al memoriei principale nu are aceeași semnificație cu cel din memoria cache. Mai exact, câmpul LINE OFFSET este constituit din ultimii 7 biți cei mai nesemnificativi (128 de linii în memoria cache) ai câmpului TAG din memoria principală, iar câmpul TAG al memoriei cache împreună cu cel LINE OFFSET formează câmpul TAG al memoriei principale.

3.2. DESIGN

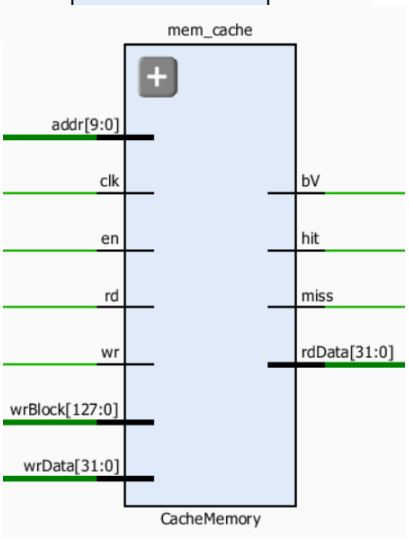
În ce privește design-ul aplicației, principalele componente utilizate sunt următoarele:

- ctrl_instr



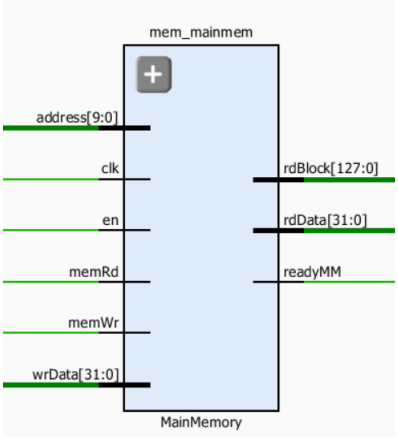
- se ocupă cu translatarea datelor de intrare, mai exact, partiționează input-ul primit pe semnalul instr în: data transmisă, adresă și tipul operației
- Este o componentă asincronă

- CacheMemory



- Memoria Cache reprezintă o memorie cu semnal de enable, citire asincronă și scriere sincronă
- Dimensiunile sunt semnificativ reduse față de dimensiunile din sistemele reale – 128 linii x 4 cuvinte= 512 ; o linie având 128 de biți
- Input: addr – adresa data de procesor
  wrData - cuvântul ce se dorește a fi scris
  wrBlock – blocul din memoria principală
- Mai are 2 semnale de read (rd) și write (wr) pentru a efectua operațiile

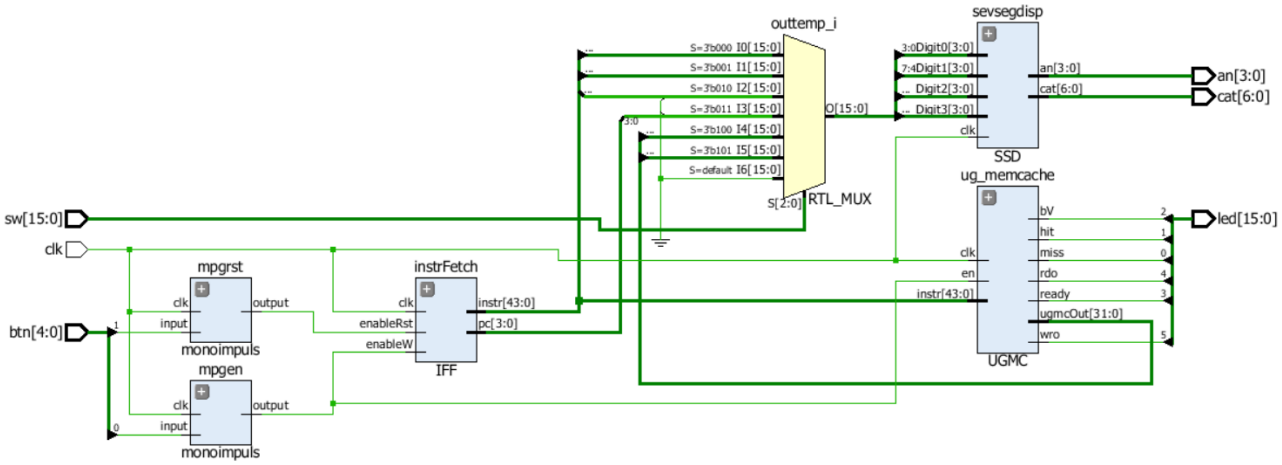
- MainMemory



- Reprezintă memoria principală de date cu semnal de enable, citire asincronă și scriere sincronă
- Dimensiunile alese sunt semnificativ reduse față de cele din sistemele reale: 1024 de cuvinte x 32 de biți
- Input: addr – adresa data de procesor
  wrData - cuvântul ce se dorește a fi scris
- Output: rdBlock – blocul din memoria principală ce se va scrie în linia memoriei cache
  rdData – cuvântul de la adresa dată de processor
  readyMM – un semnal care anunță că are loc o scriere în memoria principală

4. IMPLEMENTARE

Schema:

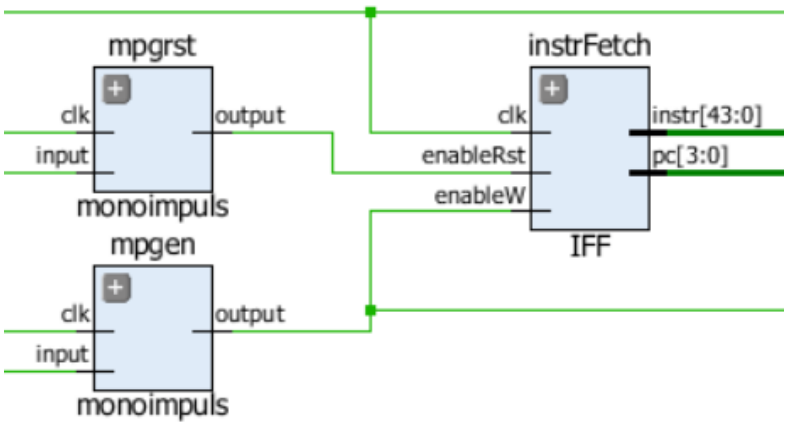


Pentru a stoca adresele și datele care se doresc a fi citite sau scrise, am decis să folosesc o memorie de adrese de dimensiune redusă, 16 cuvinte de câte 44 de biți. În această memorie, citirea este asincronă.

Astfel, codificarea celor 44 de biți este:

- Primii 2 biți vor codifica operația: citire (”00”) sau scriere (”01”)
- Următorii 10 biți sunt biții pentru adresa
- Ultimii 32 de biți sunt biții de date, cei pe care se vor transmite datele ce se doresc a fi scrise în memorie

Pentru a prelua datele din această memorie se folosește un numărător pe 4 biți și 2 generatoare de monoimpuls pentru butoanele de Enable și Reset. Când se apasă butonul de Reset, atunci se va citi din memorie de la adresa 0, iar dacă butonul de Reset nu este apăsat, dar se apasă cel de Enable, atunci numărătorul se incrementează și se citește de la adresa următoare din memorie.



Pentru afișarea datelor citite, utlizez afișorul 7 segment display al plăcuței FPGA BASYS3. Pentru a afișa numerele pe 32 de biți pe cele 4 cifre ale afișorului, am ales să codific numerele în baza 16 (hexazecimal) și să afișez în funcție de valorea switch-urilor 2,1 și 0 de pe plăcuță, diferite valori pentru a verifica corectitudinea programului. Astfel, pentru combinațiile „000”, „001”, „010” voi afișa instrucțiunea venită din memoria de instrucțiuni, iar pentru combinațiile „011” și „101” voi afișa partea HIGH, respectiv partea LOW a cuvântului căutat.

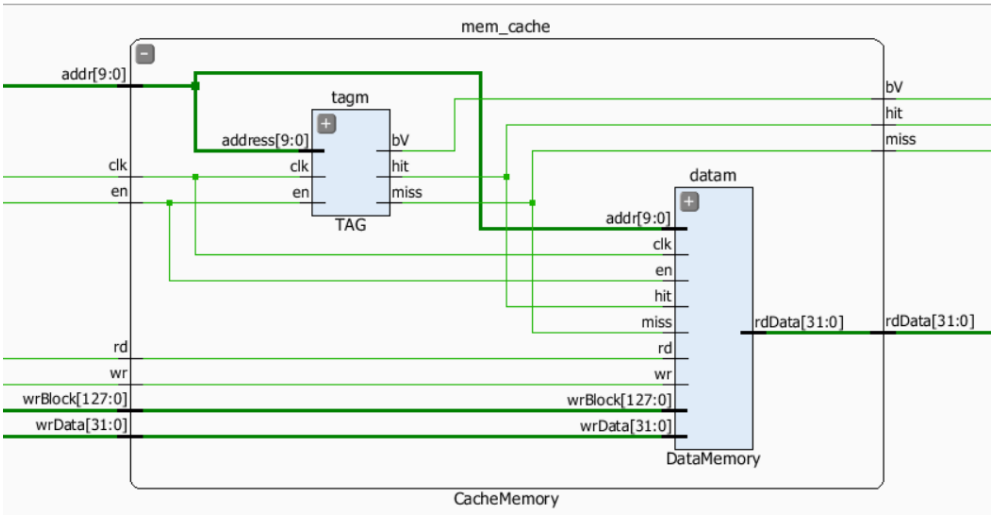
```
process(sw)
 begin
case sw(2 downto 0) is
     when "000" => outtemp(15 downto 0)<=instr(15 downto 0); --instr primele 4 cifre hexa
     when "001" => outtemp(15 downto 0)<=instr(31 downto 16); --instr urm 4 cifre hexa
     when "010" => outtemp(15 downto 0)<="0000"&instr(43 downto 32); --instr urm 3 cifre hexa
     when "011" => outtemp(15 downto 0)<=ugmc_out(15 downto 0);
     when others=> outtemp(15 downto 0)<=ugmc_out(31 downto 16);
end case;
end process;
```

Pentru a se putea urmări valoarea unor semnale de control am ales să atribui valoarea semnalelor la leduri:

- Pe ledul(0) se va putea vizualiza dacă adresa căutata nu a fost prezentă în memoria cache: MISS
- Pe ledul(1) se va putea vizualiza dacă adresa căutata a fost prezentă în memoria cache: HIT
- Pe ledul(2) se va putea vizualiza dacă linia din memoria cache a mai fost scrisă sau nu
- Pe ledul(3) se va putea vizualiza dacă se face scriere în memoria principală
- Pe ledul(4) se va putea vizualiza dacă s-a ales efectuarea unei operații de citire
- Pe ledul(5) se va putea vizualiza dacă s-a ales efectuarea unei operații de scriere

```
led(0)<=miss;
led(1)<=hit;
led(2)<=bv;
led(3)<=ready;
led(4)<=rd;
led(5)<=wr;
```

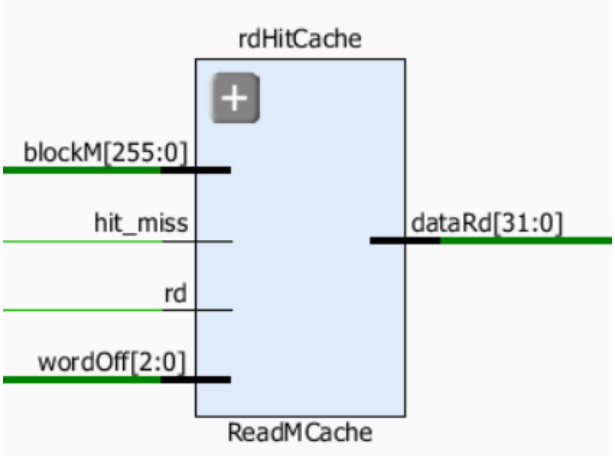
Pentru a implementa memoria Cache, am folosit 2 componente: memoria TAG și memoria de date (DataMemory).



În memoria TAG, pe fiecare linie se reține numărul blocului din memoria principală (biții 9-2 din adresă) care este stocat pe linia respectivă din memoria cache. Pentru a se verifica dacă o adresă căutată se află în memoria cache (HIT), este nevoie ca biții de etichetă (câmpul TAG, biții 9-2) ai adresei căutate să coincidă cu biții de etichetă (7-0) ai liniei din memoria TAG (componentă a memoriei cache) la care se mapează blocul, dacă linia a fost scrisă cel puțin o dată (adică bitul 8 să fie ‘1’). În caz contrar, dacă linia nu a fost scrisă niciodată (bitul 8=’0') sau dacă biții de etichetă nu coincid, atunci are loc un MISS.

```
process(address)
variable lineTag:integer:=0;
variable bloc:std_logic_vector(8 downto 0):=(others=>'0');
begin
lineTag:=conv_integer(address(8 downto 2));
bloc:=mtag(lineTag);
if bloc(8)='1' then
    if bloc(7 downto 0)=address(9 downto 2) then
        temp_hit<='1';
        temp_miss<='0';
    else
        temp_hit<='0';
        temp_miss<='1';
    end if;
else
    temp_hit<='0';
    temp_miss<='1';
end if;
bV<=bloc(8);
end process;
```

Pentru a citi un anumit cuvânt dintr-o linie a memoriei cache, am ales să folosesc o componentă ReadMCache.



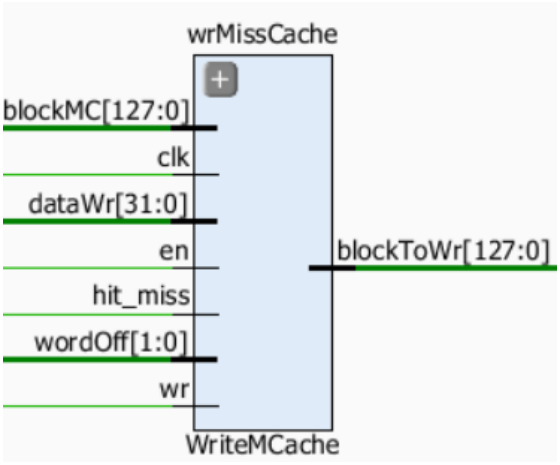
Input: hit_miss – componenta este folosită pentru citire atât în caz de HIT, cât și în caz de MISS, dar blocul de cuvinte primit diferă

- rd – semnalul de citire
- blockM – pentru HIT, va fi blocul din memoria cache; pentru MISS – blocul din memoria principală
- wordOff – care cuvânt din bloc

În funcție de aceste semnale, se va returna cuvântul dorit (dataRd)

```
process(hit_miss,rd,blockM,wordOff)
begin
if hit_miss='1' then
    if rd='1' then
        case wordOff is
            when "000" =>  dataRd <= blockM(31 downto 0);
            when "001" =>  dataRd <= blockM(63 downto 32);
            when "010" =>  dataRd <= blockM(95 downto 64);
            when "011" =>  dataRd <= blockM(127 downto 96);
            when "100" =>  dataRd <= blockM(159 downto 128);
            when "101" =>  dataRd <= blockM(191 downto 160);
            when "110" =>  dataRd <= blockM(223 downto 192);
            when others =>  dataRd <= blockM(255 downto 224);
            end case;
    end if;
end if;
end process;
```

Pentru a scrie un anumit cuvânt într-o linie a memoriei cache, am ales să folosesc o componentă WriteMCache.



Input: hit– componenta este folosită pentru scriere atât în caz de HIT, cât și în caz de MISS, dar blocul de cuvinte primit diferă

- wr – semnalul de scriere
- blockMC – pentru HIT, va fi blocul din memoria cache; pentru MISS – blocul din memoria principală
- wordOff – care cuvânt din bloc
- dataWr – cuvântul care se dorește a fi scris

În funcție de aceste semnale, se va returna blocul de cuvinte modificat care va fi scris în memoria cache (blockToWr)

```
process(hit,wordOff,wr,clk,en,dataWr,blockMC)
variable varBlockMC: std_logic_vector(255 downto 0):=(others=>'0');
begin
varBlockMC:=blockMC;
if hit='1' then
     if wr='1' then
          case wordOff is
               when "000" => varBlockMC(31 downto 0) := dataWr;
               when "001" => varBlockMC(63 downto 32) := dataWr;
               when "010" => varBlockMC(95 downto 64) := dataWr;
               when "011" => varBlockMC(127 downto 96) := dataWr;
               when "100" => varBlockMC(159 downto 128) := dataWr;
               when "101" => varBlockMC(191 downto 160) := dataWr;
               when "110" => varBlockMC(223 downto 192) := dataWr;
               when others => varBlockMC(255 downto 224) := dataWr;
          end case;
          blockToWr<=varBlockMC;
     end if;
end if;
end process;
```

5. TESTARE ȘI CONCLUZII

5.1. TESTARE

Trasare	“000” Instr(15-0)	“001” Instr(31-16)	“010” Instr(43-32)	“011” Pccnt	“100” Data(15-0)	“101” Data(31-16)	Leduri w/rd/ry/bv/h/m	Hit/Miss
Pas 0	x”0000”	x”0000”	x”0001”	0	x”0002”	x”0000”	010001	Miss
Pas 1	x”0000”	x”0000”	x”000A”	1	x”000F”	x”0000”	010001	Miss
Pas 2	x”0000”	x”0000”	x”0001”	2	x”0002”	x”0000”	010110	Hit
Pas 3	x”0000”	x”0000”	x”000A”	3	x”000F”	x”0000”	010110	Hit
Pas 4	x”001A”	x”0000”	x”0414”	4	x”000F”	x”0000”	101001	Miss
Pas 5	x”001E”	x”0000”	x”049b”	5	x”000F”	x”0000”	101001	Miss
Pas 6	x”0000”	x”0000”	x”009b”	6	x”001E”	x”0000”	010110	Hit
Pas 7	x”0000”	x”0000”	x”0014”	7	x”001A”	x”0000”	010110	Hit
Pas 8	x”0000”	x”0000”	x”0002”	8	x”0003”	x”0000”	010110	Hit
Pas 9	x”0000”	x”0000”	x”0000”	9	x”0001”	x”0000”	010110	Hit

Pentru testare, am folosit plăcuța FPGA BASYS 3. Conform rezultatelor obținute, programul funcționează conform așteptărilor.

5.2. CONCLUZII

În final, scopul proiectului a fost implementarea unei unități de gestiune a memoriei Cache. Conform cerințelor, memoria cache este mapată direct, iar politica de scriere impusă a fost scrierea imediată. Dificultățile întâmpinate au fost legate de implementarea transferurilor de date între memoria Cache și memoria principală și de identificarea componentelor necesare implementării.

6. REFERINȚE

- Universitatea Politehnică din București : site: elf.cs.pub.ro documentul: seminar2.pdf
- Universitatea Politehnică din București: Arhitectura sistemelor de calcul Lucrarea de laborator nr.5 „Memoria cache. Gestiunea memoriei cache”
- Baruch, Arhitectura Calculatoarelor, capitolul 8, subcapitolul 5 „Memoria Cache” Link: users.utcluj.ro/~baruch/book_ac/AC-Memoria-Cache.pdf
- Lucrarea de laborator nr.3 „L3 - Programming elements in VHDL.pdf” din cadrul cursului Structura Sistemelor de Calcul, Universitatea Politehnică din Cluj
- Lucrarea de laborator nr.11 „L11.pdf” din cadrul cursului Structura Sistemelor de Calcul, Universitatea Politehnică din Cluj
- https://www.geeksforgeeks.org/cache-memory-in-computer-organization/