Proiect Cache Controller

Autori:

Vintan Iulia – Grupa 3.2 C

Simion Vlad – Grupa 2.2 C

Profesor coordonator:

Bozdog Alexandru

An universitar 2024–2025

Cuprins

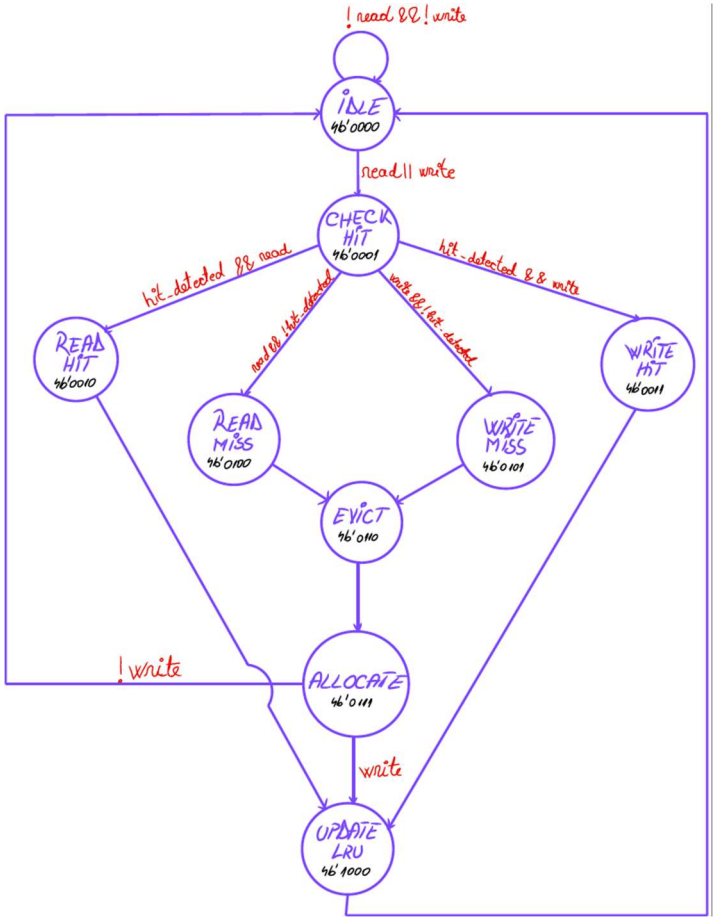
Prezentare generala si obiective

Proiectul Cache Controller are ca scop proiectarea, implementarea și testarea unui controler de memorie cache destinat unui sistem de calcul simplificat. Controlerul este realizat utilizând limbajul de descriere hardware Verilog, fiind construit pe baza unei mașini cu stari finite (FSM) pentru a gestiona eficient operațiile de citire, scriere și actualizare a memoriei cache.

Proiectul a fost dezvoltat și testat folosind platforma EDA Playground, care a permis simularea codului HDL și observarea comportamentului controllerului în diferite scenarii de acces la memorie.

Arhitectura memoriei cache are o dimensiune totală de 32 KB, cu 128 de seturi și o dimensiune a blocului de 64 de octeți, suportând un write policy de tip write-back și write-allocate.

Finite State Diagram



IDLE

În această stare, controllerul așteaptă inițierea unei operații de citire (read) sau scriere (write). Nu se face nicio verificare sau acces asupra memoriei cache în acest punct.

CHECK_HIT
Se verifică dacă adresa cerută este deja prezentă în cache. Se parcurg toate cele 4 căi ale setului corespunzător pentru a detecta o potrivire pe tag și un bit de validare activ. Dacă este găsită o potrivire, se consideră *hit*, altfel *miss*.

READ_HIT
Dacă cererea este de tip citire și s-a găsit o potrivire în cache, datele sunt extrase din calea corespunzătoare și returnate pe read_data. Apoi se trece la actualizarea politicii LRU.

WRITE_HIT
Dacă cererea este de tip scriere și adresa este găsită în cache, datele sunt scrise în blocul identificat, iar bitul de *dirty* este activat. După aceea, se face actualizarea LRU.

READ_MISS
În cazul unei cereri de citire fără *hit*, se selectează calea ce urmează a fi înlocuită (conform LRU) și se continuă către faza de *evict*.

WRITE_MISS
Similar cu READ_MISS, dar pentru operații de scriere. Se va înlocui o linie din cache, urmând politica de alocare la scriere (*write-allocate*).

EVICT
În această stare se verifică dacă linia aleasă pentru înlocuire are bitul de *dirty* setat. Dacă este, ar trebui scrisă înapoi în memoria principală (acțiune simulată în cod, dar nu efectuată complet). Bitul de *dirty* este pus pe 0.

ALLOCATE
Sunt setate bitul de validare și tag-ul, și în funcție de operație, sunt scrise datele sau se simulează o accesare a memoriei principale. Dacă este o scriere, linia devine imediat *dirty* si bitul corespunzator este setat.

UPDATE_LRU
Se actualizează contorii LRU ai setului. Linia folosită recent primește valoarea 0, iar celelalte își cresc contorul dacă sunt mai mici decât cel al liniei active. Aceasta asigură selectarea celei mai puțin recent utilizate la următoarea înlocuire.

Implementare

În implementarea controllerului de cache, a fost necesară împărțirea adresei pe 32 de biți în trei componente esențiale: *offset, index* și *tag*. Această împărțire permite identificarea precisă a locației datelor în structura cache-ului, conform arhitecturii specificate: cache 4-way Set-Associative, de 32 KB, cu blocuri de 64 de bytes și 128 de seturi.

Dimensiunea unui bloc: 64 bytes = 2^6 bytes => *6 biti pentru offset*

Număr total de seturi: 128 = 2^7 bytes => *7 biti de index*

Restul de *19 biți* sunt utilizați pentru tag

Modulul cache_controller reprezintă componenta centrală de control a memoriei cache, implementată în Verilog, și are rolul de a intermedia comunicarea dintre CPU și memoria principală (RAM). Acesta gestionează în mod autonom citirile și scrierile procesorului către memorie, având ca scop reducerea latenței de acces prin memorarea temporară a datelor frecvent utilizate într-o memorie cache.

Semnalele read și write determină tipul operației solicitate de procesor, în timp ce address conține adresa completă de 32 de biți care urmează să fie accesată.

Datele care trebuie scrise în cache sunt transmise prin semnalul *write_data*, care are o lățime de 64 de biți. În cazul unei operații de citire, rezultatul este furnizat prin semnalul de ieșire *read_data*, care returnează datele preluate din cache către CPU. Pe lângă date, cache controllerul semnalează starea accesului prin semnalele *hit* și *miss*, indicând dacă datele solicitate au fost găsite sau nu în cache.

Într-o arhitectură hardware reală, acest modul s-ar conecta direct la magistrala internă dintre CPU și memoria principală. Semnalele de intrare (cum ar fi address, read, write și write_data) ar proveni direct din procesor. Ieșirile (read_data, hit, miss) ar fi interpretate de CPU pentru a continua execuția instrucțiunii curente, fie în caz de succes, fie declanșând o accesare a memoriei principale în caz de miss2. Operațiile de alocare și evict a datelor în cache, atunci când este necesar, ar presupune transferuri către și dinspre memoria principală, simulate în această implementare prin simpla inițializare sau resetare a valorilor interne, fără acces explicit la RAM.

Testare

Pentru a valida funcționalitatea corectă a modulului cache_controller, a fost realizat un testbench în Verilog, care simulează comportamentul unui procesor ce inițiază accesări de memorie. Acest testbench a fost utilizat pentru a genera o serie de semnale de citire și scriere asupra unor adrese specifice, permițând observarea modului în care controllerul răspunde în diferite situații. Au fost testate atât scenarii de tip *cache hit*, în care datele se aflau deja în cache, cât și situații de *cache miss*, în care datele au trebuit aduse din memoria principală simulată. Prin aceste teste, s-a

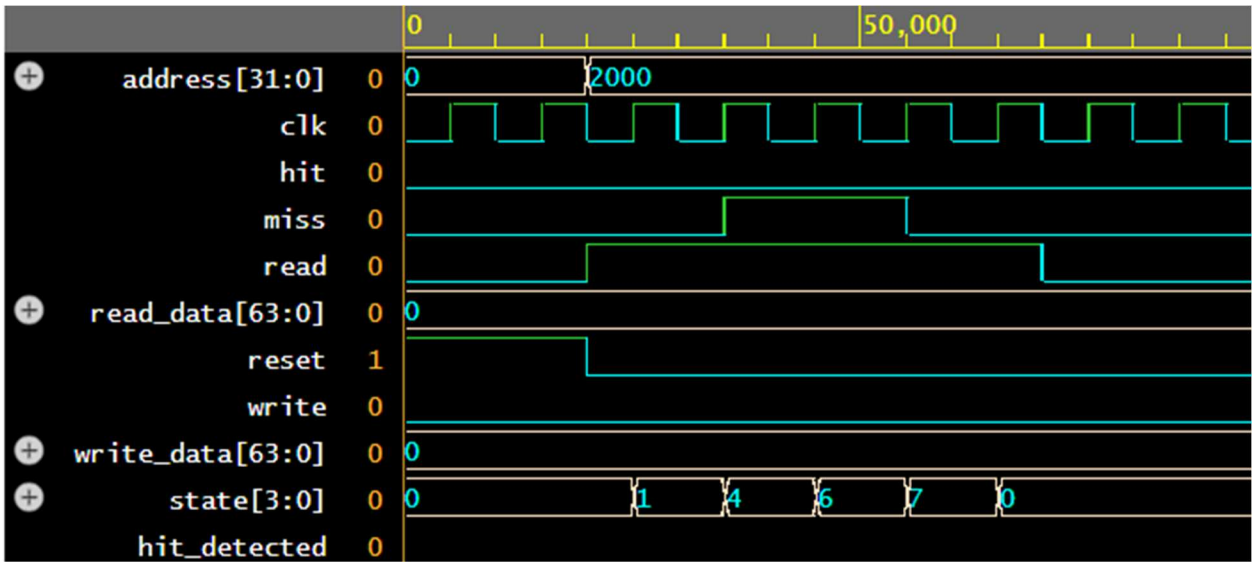
verificat funcționarea corectă a tranzițiilor între stări în FSM, comportamentul politicii de înlocuire LRU, și respectarea politicilor de scriere de tip write-back cu write-allocate. Rezultatele obținute confirmă că modulul răspunde corespunzător cerințelor specificate și îndeplinește obiectivele proiectului.

Scenariul 1 – read miss

Se trimite o cerere de citire pentru o adresă ce nu a fost încă accesată. Datele nu se găsesc în cache.

Traseu FSM:

- IDLE
- CHECK_HIT (nu se detectează hit)
- READ_MISS
- EVICT (se alege o linie de cache conform LRU)
- ALLOCATE (blocul este adus în cache)
- IDLE



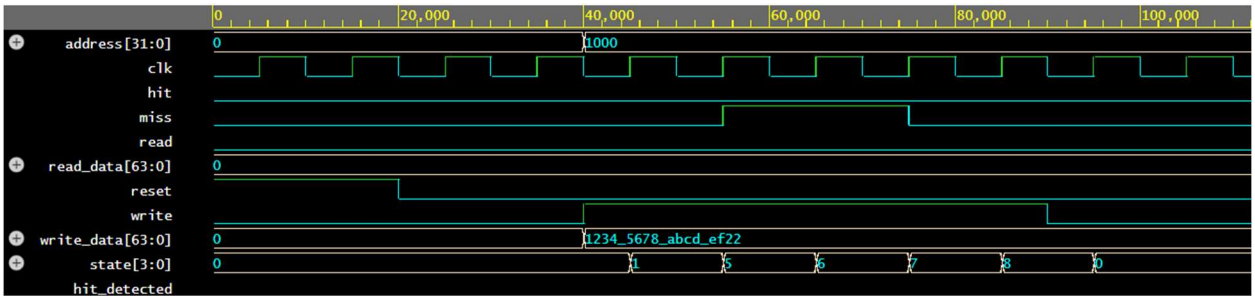
Scenariul 2 – write miss

Se trimite o cerere de scriere pentru o nouă adresă. Blocul nu este în cache, deci trebuie adus înainte de scriere.

Traseu FSM:

- IDLE
- CHECK_HIT (nu se detectează hit)
- WRITE_MISS

- EVICT
- ALLOCATE
- UPDATE_LRU (marcăm blocul ca recentry used)
- IDLE

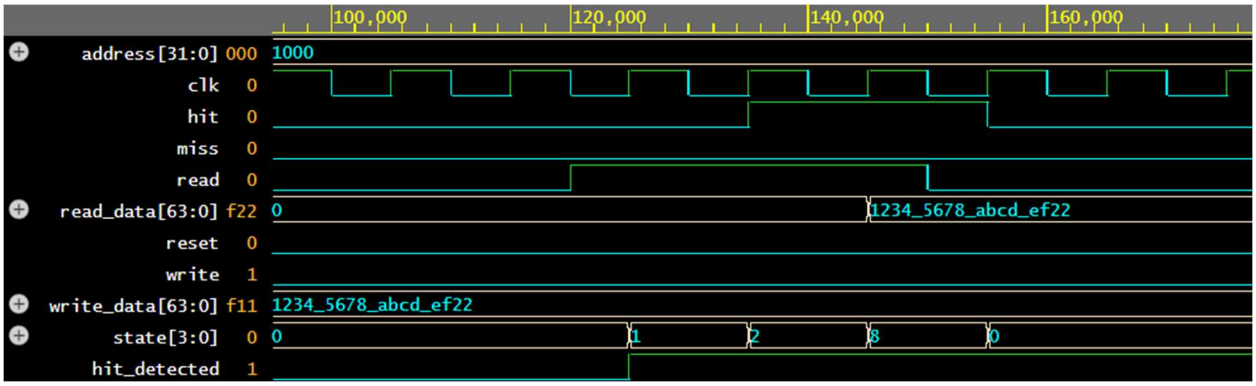


Scenariul 3 – read hit

Se citesc datele de la o adresă care a fost deja adusă în cache anterior (in cazul nostru folosim adresa din scenariul anterior).

Traseu FSM:

- IDLE
- CHECK_HIT (hit detectat)
- READ_HIT (datele sunt citite din cache)
- UPDATE_LRU
- IDLE



Scenariul 4 – write hit

Se scrie într-o adresă care este deja prezentă în cache.

Traseu FSM:

- IDLE
- CHECK_HIT (hit detectat)
- WRITE_HIT (scrierea este efectuată în cache)
- UPDATE_LRU
- IDLE