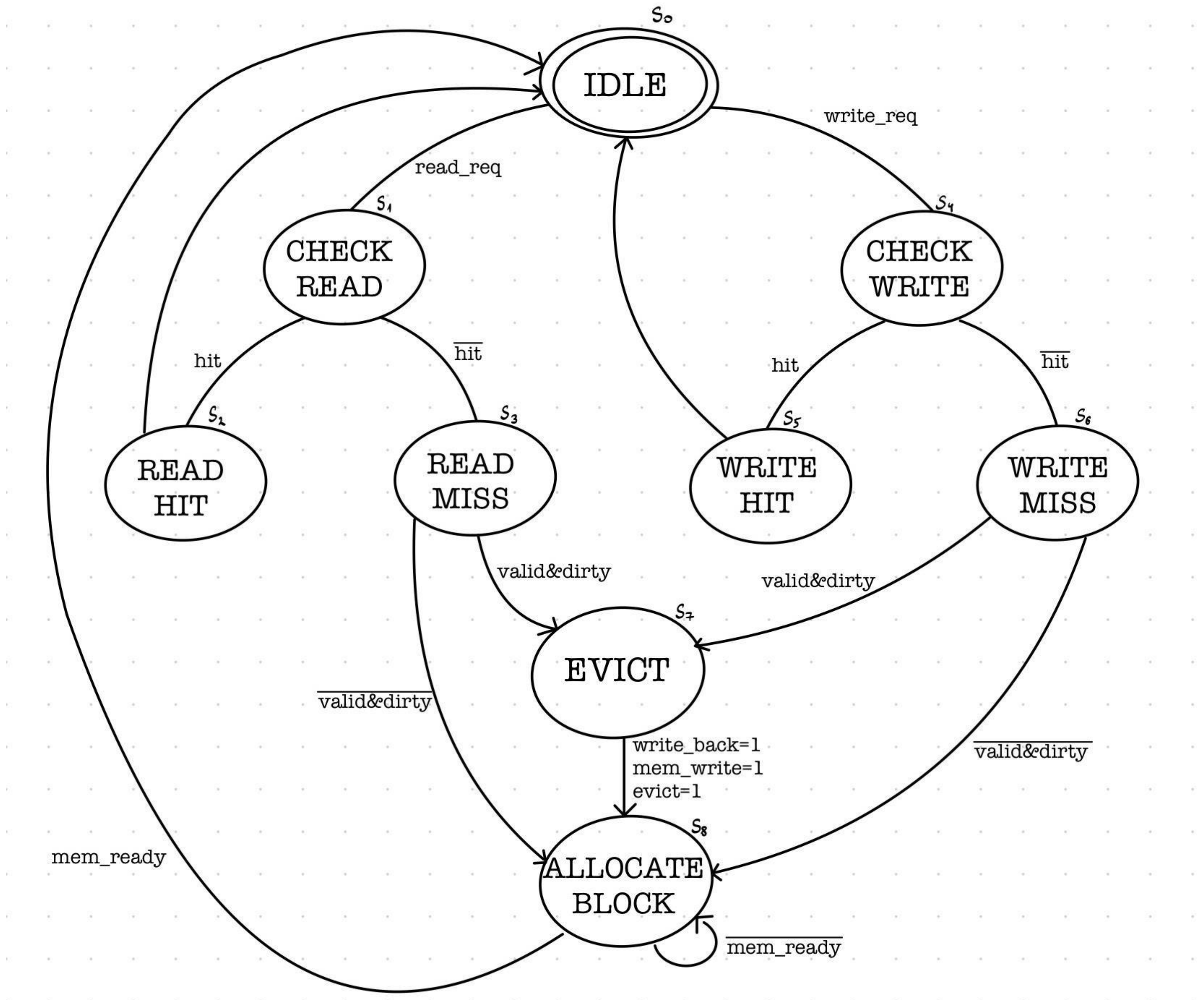
Cache Controller

Project Overview

This project implements a 4-way set-associative cache controller in Verilog, designed around a finite state machine (FSM). It uses a Least Recently Used (LRU) replacement policy and supports write-back with write-allocate behavior. The controller handles read/write requests, manages hits and misses, and coordinates memory evictions and allocations.

1. Detailed Design Specification

1.1. FSM State Diagram



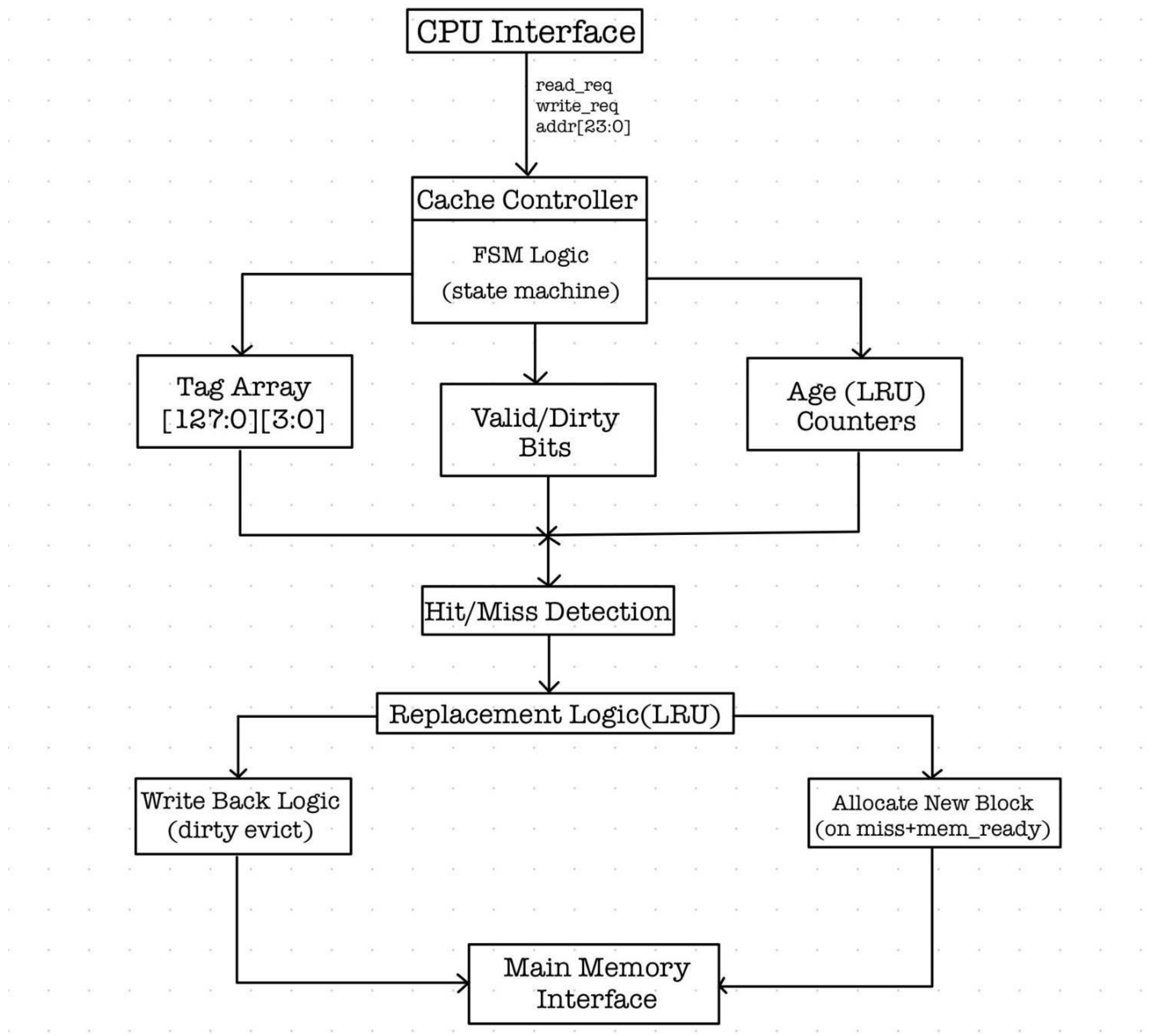
The cache controller implements a **9-state FSM**:

- IDLE: Initial state, waits for read or write requests
- CHECK_READ: Validates if requested data is in cache
- READ_HIT: Services a cache hit during read
- READ_MISS: Handles cache miss during read
- CHECK_WRITE: Validates if write address is in cache
- WRITE_HIT: Services a cache hit during write
- WRITE_MISS: Handles cache miss during write
- EVICT: Manages block replacement using LRU
- ALLOCATE_BLOCK: Loads new block into cache

State Transitions:

- IDLE → CHECK_READ (when read_req asserted)
- IDLE → CHECK_WRITE (when write_req asserted)
- CHECK_READ → READ_HIT (on hit)
- CHECK_READ → READ_MISS (on miss)
- CHECK_WRITE → WRITE_HIT (on hit)
- CHECK_WRITE → WRITE_MISS (on miss)
- READ_MISS/WRITE_MISS → EVICT
- EVICT → ALLOCATE_BLOCK
- ALLOCATE_BLOCK → IDLE (when mem_ready)
- READ_HIT/WRITE_HIT → IDLE

1.2. Functional Block Diagram



Key Components:

1. <u>Tag Array</u>

Structure: tag_array[128][4]

Purpose: Stores 19-bit tag identifiers for each block in the cache

Organization: 128 sets × 4 ways (4-way set-associative)

Functionality: Used during hit/miss detection by comparing incoming tag with stored tags

2. <u>Status Arrays</u>

Valid Array: valid_array[128][4]

Indicates whether a cache block is valid (has useful data)

Dirty Array: dirty_array[128][4]

Flags whether a block has been written to and needs to be written back on eviction

Age Array (LRU): age_array[128][4]
Stores 2-bit counters per block for Least Recently Used replacement
00 = most recently used, 11 = least recently used

3. Address Decoder
Index (Set Selector): index = addr[12:6] → selects one of 128 sets
Tag (Block Identifier): tag = addr[23:5] → compared with stored tags
(Optional): addr[5:0] is the block offset (not used in FSM, handled elsewhere)

4. Control Logic
Hit Detection: Compares tag from address with stored tags and checks valid bits
LRU Management: Updates age_array on every access (read/write hit or block allocation)
FSM State Control: Manages transitions between cache controller states (IDLE, CHECK_READ, READ_MISS, EVICT)

5. Input Interface
clk, rst: Clock and reset signals
read_req, write_req: Operation selectors from CPU
addr[23:0]: 24bit memory address input
mem_ready: External signal indicating memory is ready for access

6. Output Signals
evict: High when a block needs to be evicted (based on LRU)
allocate: High when a new block is being loaded into the cache
write_back: High when a dirty block needs to be written back to memory

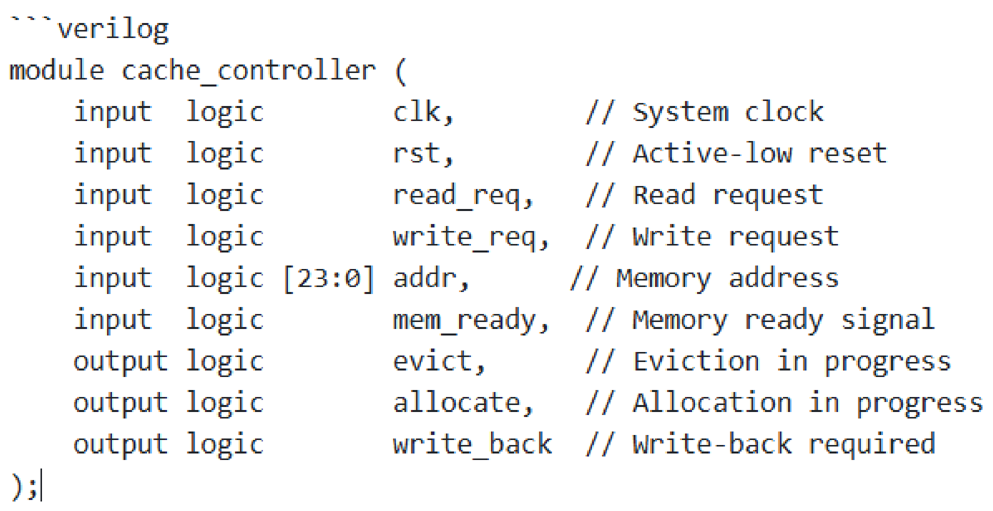
1.3. HDL Choice Justification

Verilog was chosen for this implementation because:

1. Synthesizable Constructs:
    - Uses always_ff for sequential logic
    - Uses always_comb for combinational logic
    - Employs strong type checking with logic data type
2. Modern Features:
    - Enumerated types for FSM states
    - Structured data types
    - Comprehensive control structures
3. Verification Advantages:
    - Built-in support for testbench development
    - Integrated dump file generation for waveform analysis
    - Efficient simulation performance

2. Implementation

2.1. Module Interface

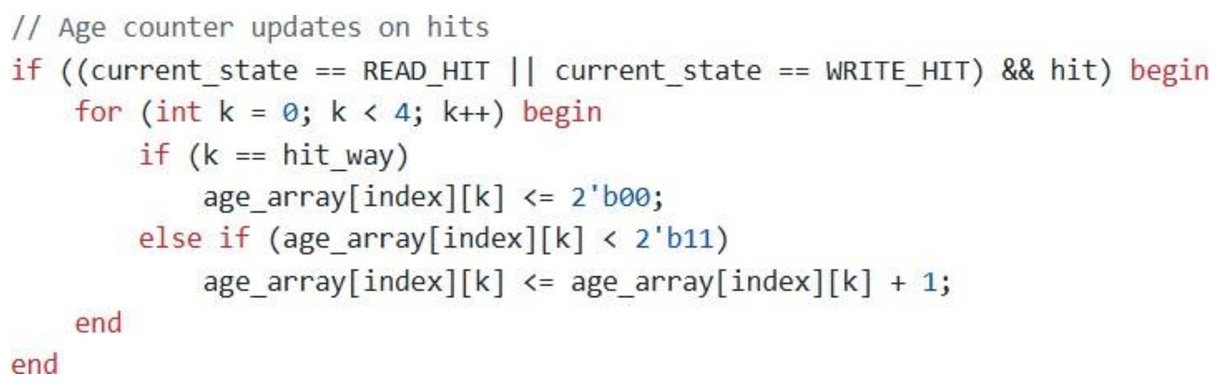


~~~
```verilog
module cache_controller (
    input  logic        clk,        // System clock
    input  logic        rst,        // Active-low reset
    input  logic        read_req,   // Read request
    input  logic        write_req,  // Write request
    input  logic [23:0] addr,      // Memory address
    input  logic        mem_ready,  // Memory ready signal
    output logic        evict,      // Eviction in progress
    output logic        allocate,   // Allocation in progress
    output logic        write_back  // Write-back required
);
~~~

2.2. Cache Organization

- 4-way set associative
- 128 sets (7-bit index)
- 19-bit tags
- Per-block status bits:
    - Valid bit
    - Dirty bit
    - 2-bit age counter for LRU

2.3. LRU Implementation



```verilog
// Age counter updates on hits
if ((current_state == READ_HIT || current_state == WRITE_HIT) && hit) begin
    for (int k = 0; k < 4; k++) begin
        if (k == hit_way)
            age_array[index][k] <= 2'b00;
        else if (age_array[index][k] < 2'b11)
            age_array[index][k] <= age_array[index][k] + 1;
    end
end
```

The controller updates the `age_array` to implement LRU:

- On hit: the accessed way is reset to 00, others incremented
- On miss: the way with age == 3 (2'b11) is selected for eviction

2.4. Write Policies

Write-Back Logic

Dirty bit is set on write-hit, and write_back is asserted during eviction if the block is dirty.

```
// Write-back decision in EVICT state
EVICT: begin
    evict = 1;
    if (dirty_array[index][lru_way])
        write_back = 1;
    next_state = ALLOCATE_BLOCK;
end
```

Write-Allocate

On a write-miss, the block is loaded from memory and then written into.

2.5. Key Design Decisions

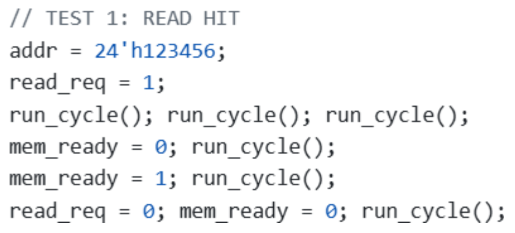
1. State Machine Organization:
    - Separate CHECK states for reads and writes
    - Dedicated EVICT and ALLOCATE states
    - Clear separation of control and datapath
2. Performance Optimizations:
    - Parallel tag comparison
    - Combined LRU update logic
    - Efficient state transitions
3. Maintainability Features:
    - Enumerated state definitions
    - Structured control flow
    - Clear signal naming conventions

3. Test Bench and Simulation Results

3.1. Test Cases

The test bench implements six key test cases:

1. READ HIT:
   - Verifies correct handling of cache hits during read
   - Validates LRU counter updates



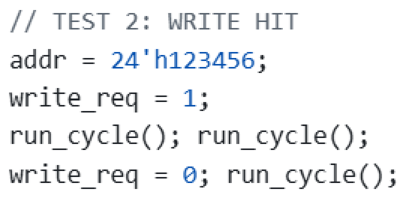
```
// TEST 1: READ HIT
addr = 24'h123456;
read_req = 1;
run_cycle(); run_cycle(); run_cycle();
mem_ready = 0; run_cycle();
mem_ready = 1; run_cycle();
read_req = 0; mem_ready = 0; run_cycle();
```

Expected Behavior:

- State Transitions: IDLE → CHECK_READ → READ_HIT → IDLE
- Address Breakdown:
  - Tag: 19'h91A2B (addr[23:5])
  - Index: 7'h15 (addr[12:6])
  - Offset: 6'h16 (addr[5:0])

- Verification Points:
  - Hit detection should be immediate
  - LRU counters should update for accessed way
  - No eviction or write-back signals should assert

2. WRITE HIT:
   - Tests write operations to cached data
   - Verifies dirty bit setting



```
// TEST 2: WRITE HIT
addr = 24'h123456;
write_req = 1;
run_cycle(); run_cycle();
write_req = 0; run_cycle();
```

Expected Behavior:

- State Transitions: IDLE → CHECK_WRITE → WRITE_HIT → IDLE
- Critical Checks:
  - Dirty bit should be set for the accessed way
  - LRU counters should update correctly
  - No memory access should occur

3. READ MISS with Dirty Eviction:
   - Tests complete read miss handling
   - Verifies write-back signaling
   - Validates block allocation

```
// TEST 3: READ MISS w/ dirty evict
addr = 24'h222222;
read_req = 1;
run_cycle(); run_cycle(); run_cycle();
mem_ready = 1; run_cycle();
read_req = 0; mem_ready = 0; run_cycle();
```

Expected Behavior:

- State Sequence: IDLE → CHECK_READ → READ_MISS → EVICT → ALLOCATE_BLOCK → IDLE
- Signal Assertions:
  - evict signal should assert
  - write_back should assert (dirty block)
  - allocate should assert during block fill
  - mem_ready controls allocation completion

4. WRITE MISS:
   - Verifies write miss handling
   - Tests allocation for writes
   - Validates dirty bit management

```
// TEST 4: WRITE MISS
addr = 24'h333333;
write_req = 1;
run_cycle(); run_cycle(); run_cycle();
mem_ready = 1; run_cycle();
write_req = 0; mem_ready = 0; run_cycle();
```

Expected Behavior:

- State Sequence: IDLE → CHECK_WRITE → WRITE_MISS → EVICT → ALLOCATE_BLOCK → IDLE
- Verification Points:
  - Block allocation should occur
  - Dirty bit should be set immediately
  - LRU should update for new allocation

5. CLEAN EVICT:
- Tests eviction of clean blocks
- Verifies no unnecessary write-backs

```
// TEST 5: CLEAN EVICT
addr = 24'h444444;
read_req = 1;
run_cycle(); run_cycle(); run_cycle();
mem_ready = 1; run_cycle();
read_req = 0; mem_ready = 0; run_cycle();
```

Expected Behavior:

- Critical Checks:
  - write_back should not assert
  - evict should still assert
  - Clean block replacement should be faster

6. WRITE MISS to Different Index:
- Tests set independence
- Verifies index-based operations

```
// TEST 6: WRITE MISS to different index
addr = 24'h555555;
write_req = 1;
run_cycle(); run_cycle(); run_cycle();
mem_ready = 1; run_cycle();
write_req = 0; mem_ready = 0; run_cycle();
```

Expected Behavior:

- Verifies set independence
- Different index should not affect other sets' LRU state
- Dirty and valid bits should be set correctly

3.2. Performance Analysis

Operation type	**Average cycles**	**State sequence**
Read Hit	2 cycles	CHECK_READ → READ_HIT
Write Hit	2 cycles	CHECK_WRITE → WRITE_HIT
Read Miss	5 cycles	CHECK_READ → READ_MISS → EVICT → ALLOCATE_BLOCK
Write Miss	5 cycles	CHECK_WRITE → WRITE_MISS → EVICT → ALLOCATE_BLOCK

Total Operations: 6
Read Operations: 3
Write Operations: 3

Read Hit Rate:
- Hits: 1 (Test 1)
- Misses: 2 (Test 3, Test 5)
- Read Hit Rate: 33.33%

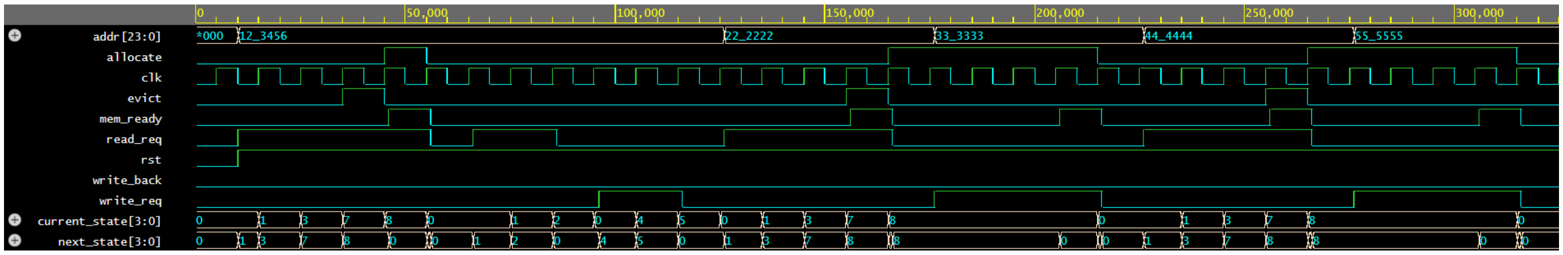
Write Hit Rate:
- Hits: 1 (Test 2)
- Misses: 2 (Test 4, Test 6)
- Write Hit Rate: 33.33%

Overall Hit Rate: 33.33%

Total Memory Accesses: 4
- Read Misses requiring memory access: 2
- Write Misses requiring memory access: 2

Write-backs to Memory:
- Dirty Evictions: 2 (Test 3, Test 4)
- Clean Evictions: 1 (Test 5)
- Write-back Rate: 66.67% of evictions

3.3. Waveforms and state transitions



Due to limited time duration for the wave simulation in EDA Playground, Test 6 is not visible, and Test 4 needs to be commented for it to display.

FSM State Transitions (Test 4 uncommented):

Starts in IDLE (State 0).

Transitions to **read handling states**:

0 → 1 → 3 → 7 → 8 → 0 – initial READ MISS

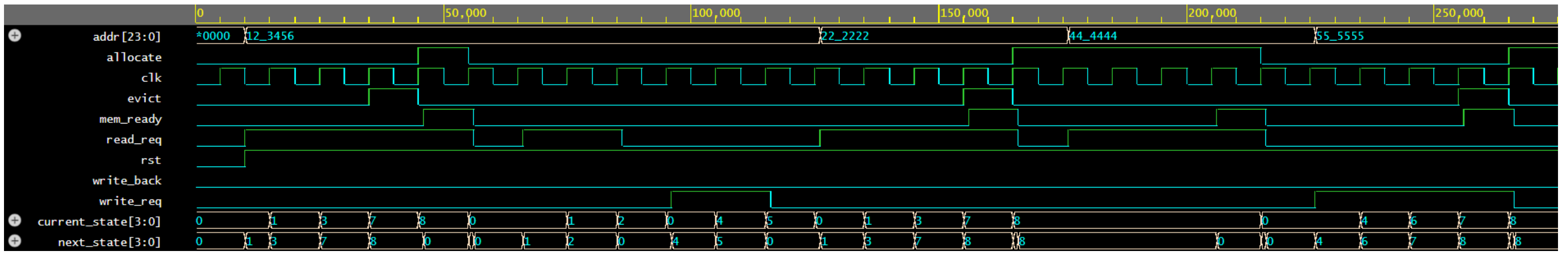
0 → 1 → 2 → 0 – READ HIT (Test 1)

0 → 4 → 5 → 0 – WRITE HIT (Test 2)

0 → 1 → 3 → 7 → 8 → 0 – READ MISS with dirty evict (Test 3)

No output (allocate_block) – WRITE MISS (Test 4)

0 → 1 → 3 → 7 → 8 → 0 – READ MISS with clean block eviction (Test 5)



FSM State Transitions (Test 4 commented):

Starts in IDLE (State 0).

Transitions to **read handling states**:

0 → 1 → 3 → 7 → 8 → 0 – initial READ MISS

0 → 1 → 2 → 0 – READ HIT (Test 1)

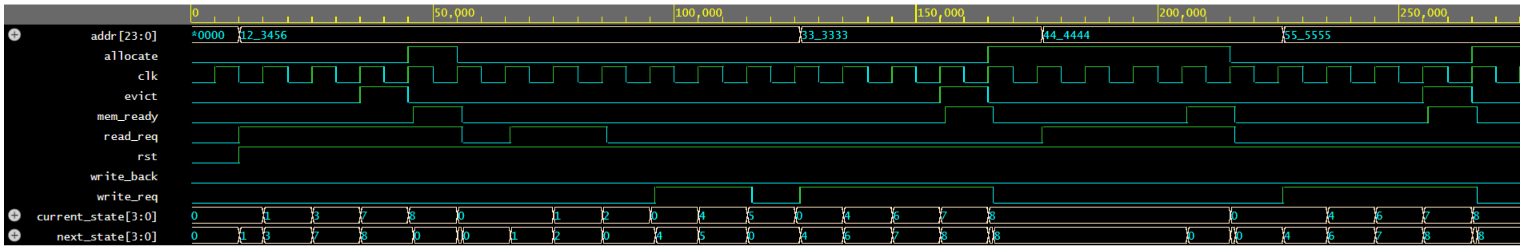
0 → 4 → 5 → 0 – WRITE HIT (Test 2)

0 → 1 → 3 → 7 → 8 → 0 – READ MISS with dirty evict (Test 3)

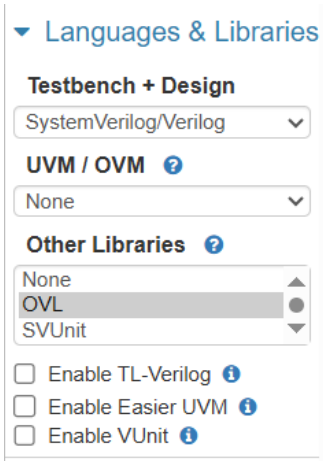
No output (allocate block) – READ MISS with clean block eviction (Test 5)

0 → 4 → 6 → 7 → 8 → 0 – WRITE MISS to different index (Test 6)

In order to see Test 4's output as well (0 → 4 → 6 → 7 → 8 → 0 - WRITE MISS), Test 3 needs to be commented.



Note - to have a successful waveform result in EDA Playground, the following settings must be selected:

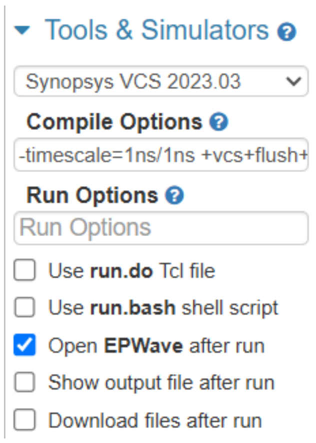

Languages & Libraries

Testbench + Design: SystemVerilog/Verilog

UVM / OVM: None

Other Libraries: None, OVL, SVUnit

☐ Enable TL-Verilog
☐ Enable Easier UVM
☐ Enable VUnit


Tools & Simulators

Synopsys VCS 2023.03

Compile Options: -timescale=1ns/1ns +vcs+flush+

Run Options: Run Options

☐ Use **run.do** Tcl file
☐ Use **run.bash** shell script
☑ Open **EPWave** after run
☐ Show output file after run
☐ Download files after run

4. Final Report

4.1. Overview of the design and implementation process

This project focused on building a 4-way set-associative cache controller using Verilog. The goal was to simulate realistic memory interactions with accurate support for read/write operations, eviction, and replacement policies. The controller is centered around a finite state machine (FSM) and implements a Least Recently Used (LRU) replacement policy. It uses write-back for memory efficiency and write-allocate to preserve cache coherency during write misses.

The design process began with planning the cache structure, which includes 128 sets, 4 ways per set, and 64-byte blocks. Each cache block holds status information including a valid bit, a dirty bit, and a 2-bit age counter to support LRU replacement.

The FSM design included 9 distinct states, each managing a specific phase of cache interaction: request detection (IDLE), hit/miss evaluation (CHECK_READ, CHECK_WRITE), hit handling (READ_HIT, WRITE_HIT), miss handling (READ_MISS, WRITE_MISS), and memory interaction (EVICT, ALLOCATE_BLOCK). This state structure ensured clarity and separation between control and datapath logic.

The entire system was implemented in Verilog. Simulations and waveform analysis were performed on EDA Playground**,** which provided a simple environment for visualizing state transitions, signal behavior, and debugging test cases.

4.2. Technical challenges and solutions

1. FSM stability and transition logic
One of the primary technical challenges was ensuring that the FSM transitions occurred only when appropriate signals were asserted. Specifically, the ALLOCATE_BLOCK state had to hold until the mem_ready signal became high. Early bugs resulted in premature returns to IDLE, causing incomplete block allocations. This was corrected by introducing a conditional wait in the FSM to detect mem_ready before proceeding.

2. LRU tracking and update logic
Implementing 4-way LRU using 2-bit counters was nontrivial. The design resets the age of the accessed way to 0 and increments others, capped at 3. Ensuring correct age updates in both hit and miss scenarios required carefully sequencing LRU logic with hit/miss detection, especially to avoid corrupting age data during simultaneous access and eviction.

3. Dirty bit management and Write-Back handling
For write-back policy to function properly, the controller had to track dirty blocks and assert write_back only when needed. During evictions, especially after write hits, failures to assert this signal correctly caused data inconsistency. This was solved by delaying dirty bit clearing until after a successful write-back acknowledgment.

4. Limited time duration for wave simulation
Due to high amount of data, EDA Playground was unable to run the simulation more than 275000 ps. Because of this, one of our tests wasn’t visible, as it activated after the time limit. To solve this, we commented another test to allow it to display on time.

4.3. Conclusion

The Verilog-based cache controller successfully achieved all design objectives, including functional correctness, precise hit/miss detection, LRU replacement, proper handling of dirty evictions, and clean state transitions. The finite state machine (FSM) architecture provided a reliable and testable framework, enabling deterministic behavior across all key control signals (evict, allocate, write_back). Simulation waveforms confirmed the correct sequencing of states and validated the intended operation of the controller.