# Cache Controller Project: Final Report

## 1. Overview of the Design and Implementation Process

The cache controller project was developed to simulate and verify the behavior of a set-associative cache system, focusing on realistic CPU-cache-memory interactions. The design implements a 4-way set-associative cache with 128 sets, 16 words per block (512 bits), and a Least Recently Used (LRU) replacement policy. The controller manages read and write requests from the CPU, handles cache hits and misses, and coordinates with main memory for block allocation and eviction.

The implementation process began with a clear specification of the cache architecture, including parameterization for word size, block size, set count, and associativity. The main modules developed were:

* **cache\_controller.v**: The core finite state machine (FSM) that orchestrates cache operations, manages candidate lines, and interfaces with both the CPU and memory.
* **replacer.v**: A utility module for updating a specific word within a cache block, used during write operations.
* **cache\_controller\_tb.sv**: A comprehensive SystemVerilog testbench that simulates a variety of CPU access patterns, including hits, misses, write-backs, and edge cases.

The design emphasizes modularity and parameterization, allowing for easy adaptation to different cache configurations. The FSM in the controller ensures correct sequencing of operations, including hit detection, LRU updates, dirty block eviction, and block allocation from memory.

## 2. Technical Challenges Encountered and Solutions Implemented

### a. LRU Replacement Policy

**Challenge:** Implementing an efficient and correct LRU policy for a 4-way set-associative cache, ensuring that the “oldest” line is always selected for replacement on a miss.

**Solution:** Each cache line maintains a 2-bit age field. On every access, the controller updates the ages: the accessed line is set to 0, and all valid lines with a lower age are incremented. This logic is implemented combinationally and verified in the testbench. The LRU candidate is selected by finding the line with the maximum age.

### b. Handling Write-Backs and Dirty Blocks

**Challenge:** Correctly managing dirty blocks during eviction, ensuring that modified data is written back to memory before replacement.

**Solution:** The controller checks the dirty and valid bits of the LRU candidate on a miss. If eviction is required, the block is written to memory before the new block is allocated. The FSM includes explicit EVICT and ALLOCATE states to sequence these operations, and the testbench verifies correct memory transactions.

### c. Synchronization and Signal Timing

**Challenge:** Ensuring correct synchronization between the controller, cache, and memory, especially with respect to ready/enable handshakes and clocking.

**Solution:** The design uses registered signals and flip-flop modules to synchronize candidate data and control signals. The testbench provides realistic clocking and ready/enable pulses, and the controller FSM waits for appropriate ready signals before proceeding to the next state.

### d. Parameterization and Modularity

**Challenge:** Making the design flexible and reusable for different cache sizes and associativities.

**Solution:** All key parameters (word size, block size, set count, associativity, etc.) are defined as module parameters. The replacer and block\_selector modules are also parameterized, supporting easy scaling and adaptation.

### e. Comprehensive Verification

**Challenge:** Creating a testbench that covers all relevant scenarios, including hits, misses, write-backs, and edge cases (e.g., empty candidates).

**Solution:** The SystemVerilog testbench (cache\_controller\_tb.sv) includes tasks for read/write requests, candidate provisioning, and memory response simulation. It runs a suite of test cases covering read/write hits, misses with and without eviction, and operations with empty or partially filled sets. The testbench also generates VCD waveforms for detailed analysis.

## 3. Analysis of Performance Data Collected During Simulations

Simulation results were collected using the testbench, which exercises the cache controller with a variety of access patterns. Key performance metrics include hit rate, miss rate, and the number of memory transactions (reads/writes).

### a. Hit and Miss Behavior

* **Read/Write Hits:** The controller correctly identifies hits in any of the four candidates, updates the LRU ages, and returns data to the CPU with minimal latency (typically within a few cycles).
* **Misses Without Eviction:** When a miss occurs and there is an invalid (empty) candidate, the controller allocates the new block without requiring a write-back, reducing memory traffic.
* **Misses With Eviction:** If all candidates are valid and the LRU candidate is dirty, the controller performs a write-back to memory before allocating the new block. This is correctly sequenced and verified in the testbench.

### b. LRU Policy Effectiveness

Waveform analysis and testbench output confirm that the LRU policy is correctly maintained. After each access, the ages of the candidates are updated as expected, and the oldest line is always selected for replacement. This ensures optimal cache utilization and minimizes unnecessary evictions.

### c. Memory Traffic

The testbench logs show that memory transactions (reads and writes) occur only on misses and evictions, as expected. Write-backs are performed only for dirty blocks, reducing unnecessary memory writes.

### d. Latency and Throughput

* **Cache Hits:** Data is returned to the CPU with low latency, typically within 1-2 cycles after the request.
* **Cache Misses:** Misses incur additional latency due to memory access and possible eviction, but the FSM ensures correct sequencing and minimal stalling.
* **Throughput:** The controller can handle back-to-back requests, with the FSM returning to the IDLE state promptly after each operation.

### e. Edge Case Handling

The testbench includes cases with empty candidates and partially filled sets. The controller correctly identifies free slots and avoids unnecessary evictions, demonstrating robust handling of all scenarios.

## 4. Conclusion

The cache controller project successfully implements a parameterized, modular, and robust set-associative cache controller with LRU replacement and write-back support. The design addresses key technical challenges, including LRU management, dirty block handling, and synchronization. Comprehensive simulation and waveform analysis confirm correct functionality, efficient memory usage, and robust handling of all edge cases. The project provides a solid foundation for further exploration of cache architectures and performance optimization.