Team members:
Bardan Elena-Bianca
Golya Carmen-Mihaela
Hondola Paul
Iordachescu Vlad

Semigroup:
2.1, 2nd year, Fac.
Of Automation and Computers,
Specialization-Computers

Cache Controller Using HDL.

1. Project overview.

Modern computing systems rely heavily on an efficient memory hierarchy, where cache memory plays a crucial role in minimizing data access latency. Situated between the CPU and main memory, the cache stores frequently accessed data to speed up processing.

A cache controller manages data transfers between the CPU, cache, and main memory, ensuring coherence and improving overall performance. Designing such a controller requires a solid understanding of access patterns, replacement policies, and synchronization mechanisms.

Hardware Description Languages (HDLs) like Verilog and VHDL are commonly used to model, simulate, and implement cache controllers, enabling accurate and efficient hardware design.

2. Project description.

a. Main modules specifications.

The cache controller was designed using a modular approach, dividing the functionality into six main components. Each module handles a specific part of the cache operation to ensure clarity, maintainability, and efficient design.

- ***Control Unit***: This finite-state machine (FSM) manages the cache controller's operation. It processes read and write commands, monitors hit or miss responses and orchestrates LRU updates by coordinating address and data signals. The FSM transitions through specific states to initiate cache access and update replacement policies.
- ***Cache Memory:*** The cache_memory module implements a 128-set, 4-way set-associative cache. It takes a full address, read/write requests, and data to write as inputs. The module selects one cache set based on part of the address and routes control signals to it. It outputs the read data, age info for each way, and hit/miss status for the selected set.
- ***Cache Line:*** The cache_line module represents a single line in a set-associative cache, storing a block of data along with its tag, valid, dirty, and age bits. It takes in a memory address and read/write commands, compares the address tag to detect hits or misses, and outputs the requested data on a hit. On misses, it allocates a new line by resetting data and updating tags. It also manages an age counter for eviction decisions and exposes status signals like hit/miss, valid, and dirty.

- ***Cache Controller:*** The cache_controller module manages a 4-way set-associative cache by coordinating read and write requests based on an input command (opcode). It takes a clock and reset signal, generates addresses and data for operations, and communicates with the cache memory to perform tag lookups and data access. The outputs are the data read from the cache (on a hit) and a signal indicating whether the access was a hit or miss. It also handles cache line aging to implement the Least Recently Used (LRU) replacement policy.
- ***Bitwise Comparator:*** The bitwise_comparator module compares two input vectors bit-by-bit. It outputs eq high only if all corresponding bits in in_0 and in_1 are equal. It instantiates individual bit comparators for each bit, then uses a reduction AND to combine all bitwise results into a single equality signal.

Together, these modules form a complete 4-way set associative cache controller capable of efficient memory management using the Least Recently Used (LRU) replacement policy.

b. Architecture and Specifications.

This cache controller is designed as a 4-way set associative cache with the following specifications:

- Cache Size: 32 KB
- Block Size: 64 bytes
- Word Size: 4 bytes
- Number of Sets: 128
- Associativity: 4-way
- Replacement Policy: Least Recently Used (LRU)
- Write Policy: Write-back with write-allocate

The architecture consists of multiple cache sets, each containing four cache lines. The controller manages data movement between the CPU, cache memory, and main memory. It uses decoders to select sets, multiplexers to select cache lines, and modules that handle tag comparison, data storage, and cache line aging.

c. FSM-Based Design.

The control_unit module relies on a finite state machine (FSM) composed of seven states, encoded using 3-bit values. This FSM governs the control flow for issuing read/write operations to a 4-way set-associative cache and managing replacement logic based on the Least Recently Used (LRU) policy.

The FSM handles request initiation, response evaluation (hit or miss), and selectively updates age counters per cache way to enforce correct LRU behavior. State transitions are deterministic and triggered on the rising clock edge, with reset support to reinitialize the controller.

FSM state description:

- ***IDLE*:** The default waiting state. It deactivates all control signals and awaits a valid opcode input. Upon receiving an instruction, it transitions to either TRY_READ or TRY_WRITE depending on whether the operation is a read (0) or a write (1).

- ***TRY_READ*:** In this state, the controller asserts a single cycle try_read signal to initiate a read operation. A random address is generated and stored in address_word. The FSM then proceeds to CHECK_STATUS to evaluate the outcome of the request.
- ***TRY_WRITE*:** Like TRY_READ, but for write operations. It asserts try_write for one cycle, generates a random address and an 8-bit random value for write_data, and then transitions to CHECK_STATUS.
- ***CHECK_STATUS*:** A transitional state where the controller waits for one cycle to receive the cache response (hit_miss). If the access results in a hit, the FSM moves directly to UPDATE_AGES; if it's a miss, the FSM enters a simulated stall phase via STALL_1.
- ***STALL_1*:** First of two stall states used to simulate cache miss latency or main memory access delay. No control signals are asserted. Automatically advances to STALL_2 in the next cycle.
- ***STALL_2*:** Completes the second cycle of the stall sequence after a cache miss. Upon completion, the FSM transitions to UPDATE_AGES to perform the necessary LRU state updates for cache replacement.
- ***UPDATE_AGES*:** Performs LRU maintenance operations. It identifies the way that was accessed (on hit) or selected for replacement (on miss) using the one-hot hit_miss_set signal. The selected way's age is reset via reset_age, while any other way with a numerically lower age is flagged in increment_age for increment. The FSM then returns to IDLE to await the next operation.

3. Implementation and Testing.

a. *HDL Code Structure.*

The cache controller was implemented in Verilog, using a modular and hierarchical design. Each module was developed and tested independently to ensure correctness and reusability. The top-level module integrates all subcomponents, including the control unit, cache memory, cache lines, and comparator logic. All interconnections were verified to conform to the expected data flow and control signal protocols.

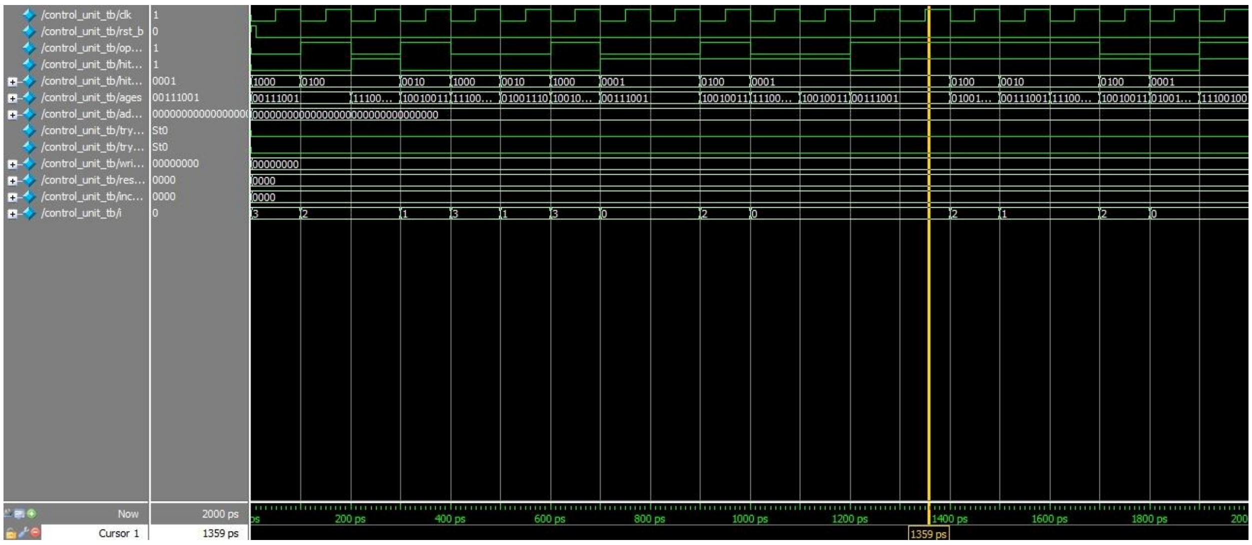
b. *Testbench Setup and Simulation Results.*

A testbench was developed to emulate realistic CPU activity by issuing random read and write commands to the cache controller. It generates random addresses and data, monitoring critical outputs like hit/miss status, data correctness, and LRU age updates.

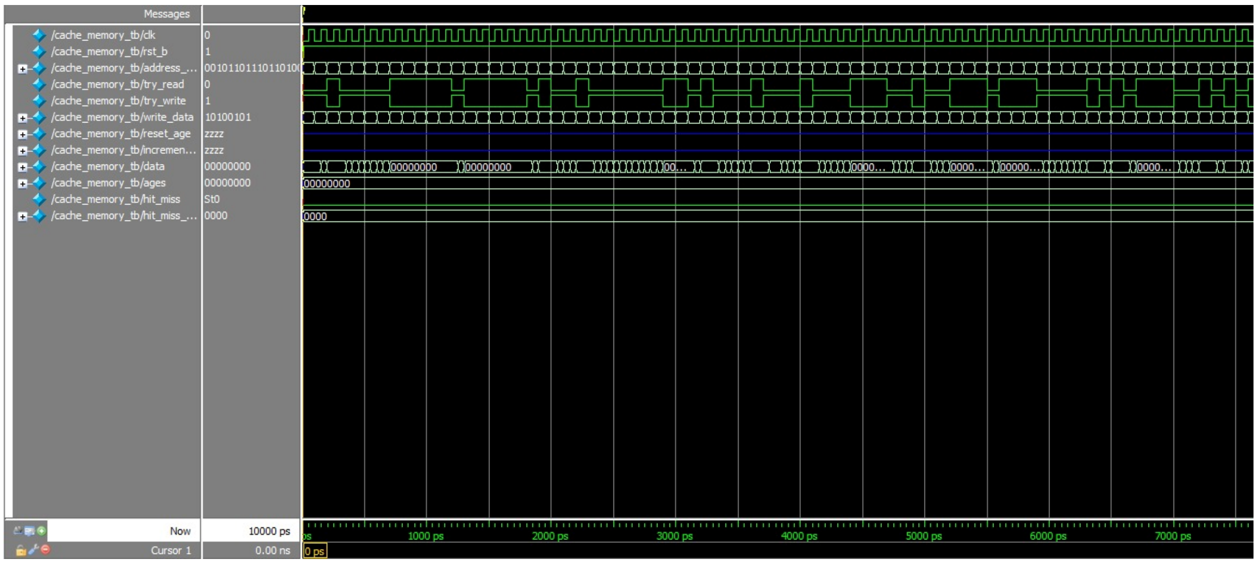
c. *Simulation results.*

The simulation showed the cache controller correctly handled random read/write commands, updating LRU ages and managing hits and misses. Hit/miss signals and stall states behaved as expected, validating the controller's functionality and timing.

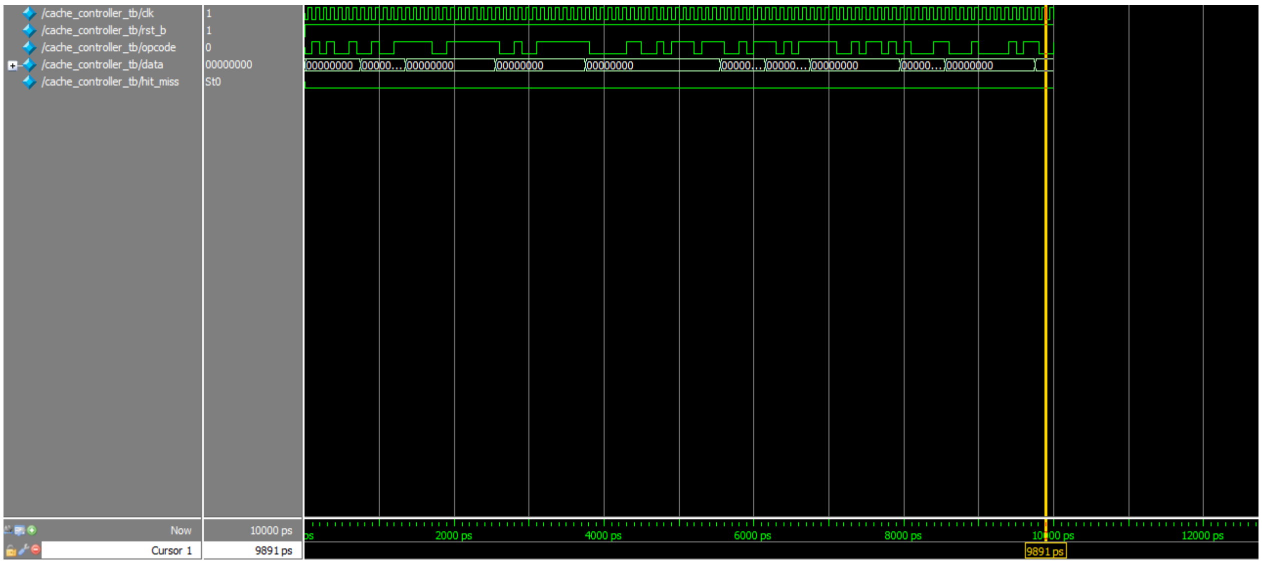
- **control_unit_tb:** The control unit testbench simulates clock, reset, opcode, hit/miss signals, and age values over multiple cycles. It verifies the FSM's response to random read/write operations and LRU updates. The waveform shows correct state transitions and signal behavior throughout the simulation.


/control_unit_tb/clk 1
/control_unit_tb/rst_b 0
/control_unit_tb/op... 1
/control_unit_tb/hit... 1
/control_unit_tb/hit... 0001
/control_unit_tb/ages 00111001
/control_unit_tb/ad... 0000000000000000
/control_unit_tb/try... St0
/control_unit_tb/try... St0
/control_unit_tb/wri... 00000000
/control_unit_tb/res... 0000
/control_unit_tb/inc... 0000
/control_unit_tb/i 0
Now 2000 ps
Cursor 1 1359 ps

- **cache_memory_tb:** The cache memory testbench applies random read/write commands with varied addresses and data inputs. It tests cache hits, misses, and age updates over multiple clock cycles. The waveform demonstrates proper data handling and hit/miss signal behavior during simulation.


Messages
/cache_memory_tb/clk 0
/cache_memory_tb/rst_b 1
/cache_memory_tb/address_... 0010110111011010
/cache_memory_tb/try_read 0
/cache_memory_tb/try_write 1
/cache_memory_tb/write_data 10100101
/cache_memory_tb/reset_age zzzz
/cache_memory_tb/incremen... zzzz
/cache_memory_tb/data 00000000
/cache_memory_tb/ages 00000000
/cache_memory_tb/hit_miss St0
/cache_memory_tb/hit_miss_... 0000
Now 10000 ps
Cursor 1 0.00 ns

- **cache_controller_tb:** The cache controller testbench simulates random read and write operations over multiple clock cycles. It verifies correct data output and hit/miss signaling. The waveform confirms proper controller response to varying opcodes.


/cache_controller_tb/clk 1
/cache_controller_tb/rst_b 1
/cache_controller_tb/opcode 0
/cache_controller_tb/data 00000000
/cache_controller_tb/hit_miss St0
Now 10000 ps
Cursor 1 9891 ps
2000 ps
4000 ps
6000 ps
8000 ps
10000 ps
12000 ps
9891 ps

4. Conclusion.

This project successfully demonstrated the design and implementation of a 4-way set-associative cache controller using HDL and an FSM-based approach. It provided practical experience in managing cache operations, including handling read/write hits and misses, eviction policies, and state transitions essential for cache performance.

This project also helped us develop a better appreciation for the complexity behind seemingly simple memory operations, and gave us confidence in writing modular, testable hardware logic using Verilog.