Implementation of a Cache Controller

Hărdălău Elvis Costinel

Iorga Răzvan Vlad

Cache Controller Specifications

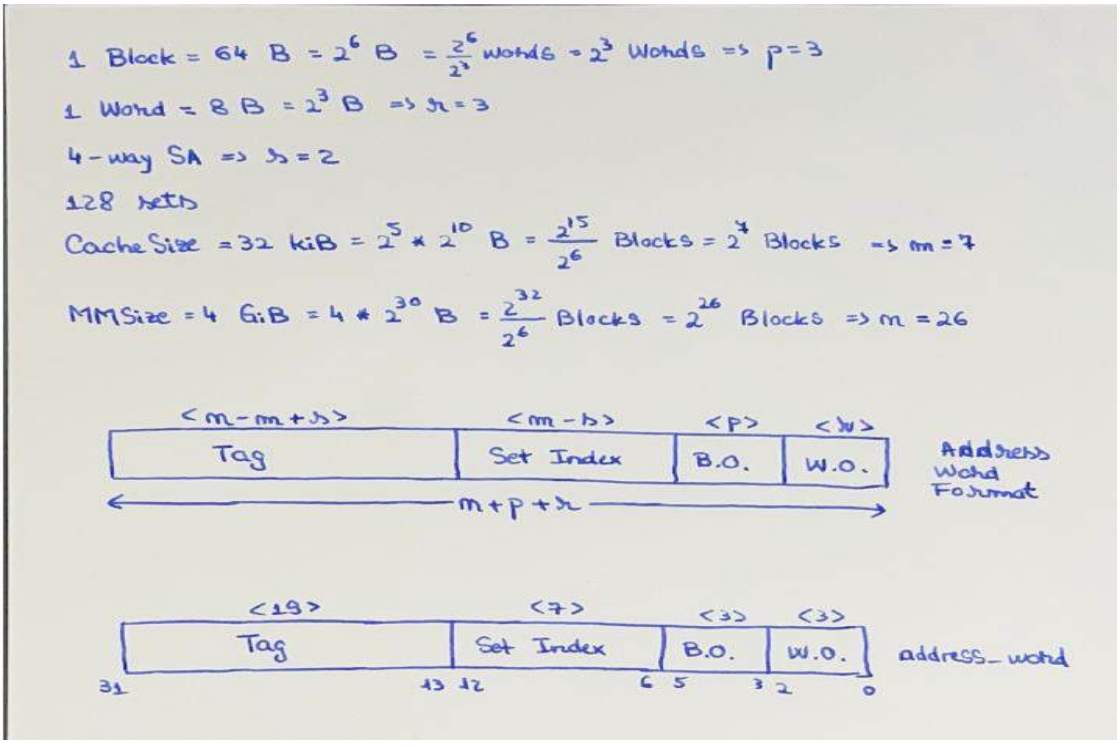
The cache controller to be designed must adhere to the following specifications:

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

Due to the fac that a block has 64 bytes, the offset will be on 6 bits. Furthermore, knowing that the cache size is 32 KiB and a block has 64 bytes, with an easy divison, for a DM cache, the index would be on 9 bits. However, we have a 4-way SA with 128 sets, meaning that the index will be on 7 bits.

As discussed at the laboratory, the main memory size will be at our choice. We will implement it on 4 GiB.

As a result, the tag will be on 19 bits. This is the Address format and the size for each field. Moreover, having LRU replacement Policy, we will need 2 bits for the age register. In addition, we will need a dirty bit for the write back policy.



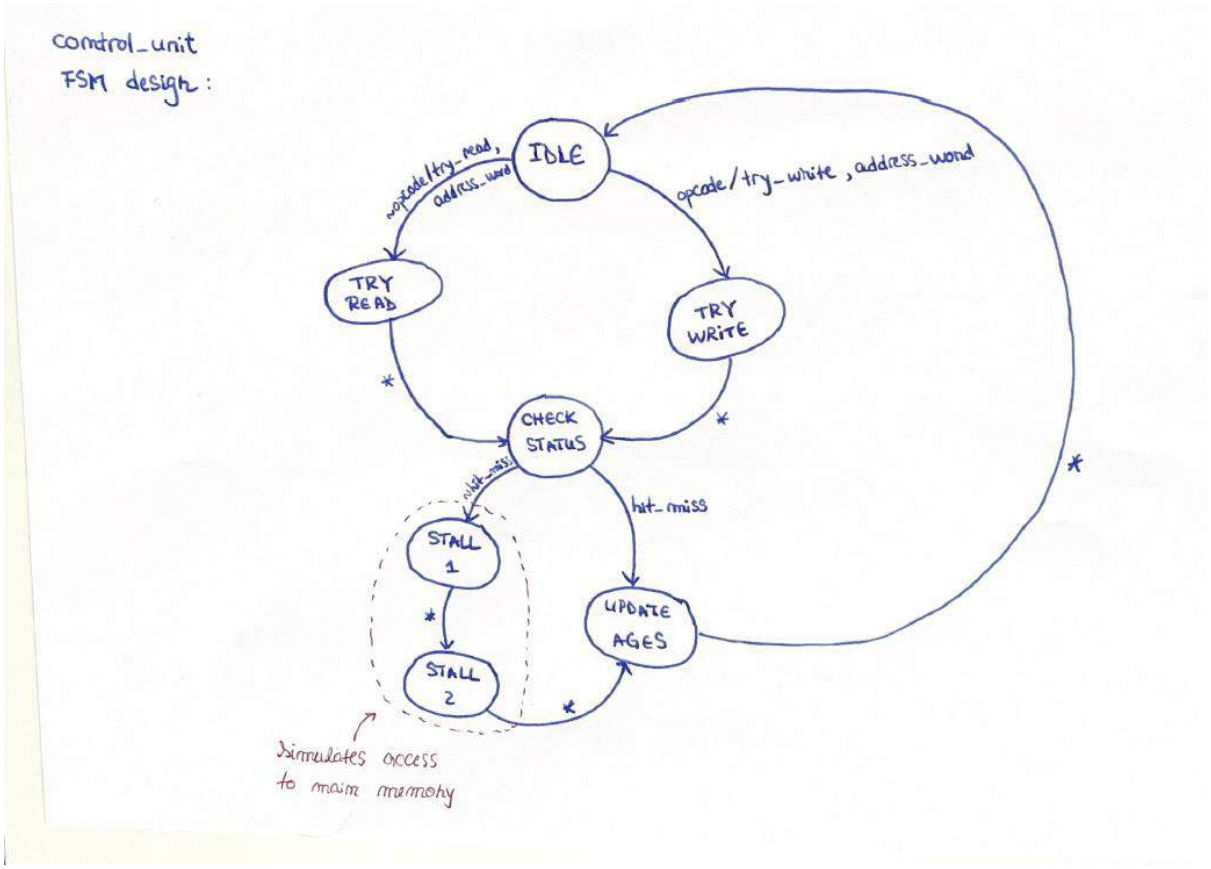
FSM-Based Design

The cache controller should be implemented as a finite state machine with clearly defined states including, but not limited to:

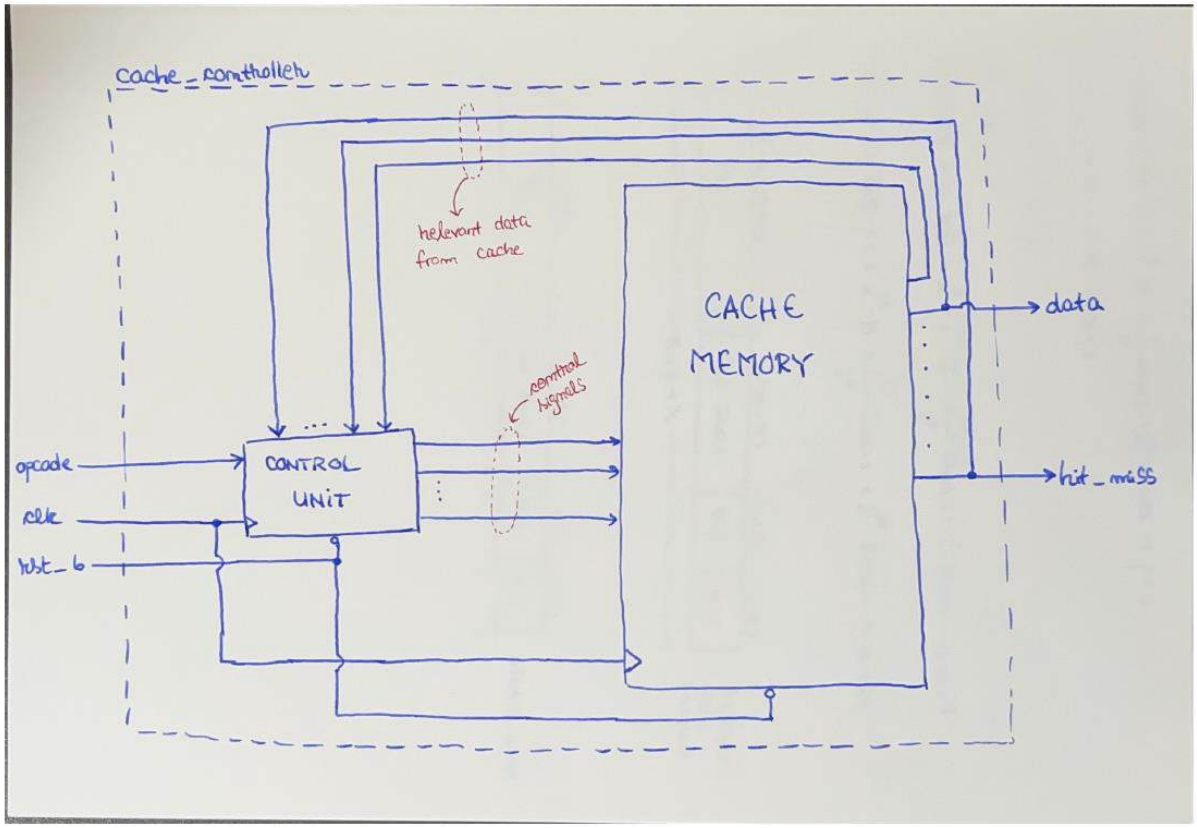
- IDLE: Waiting for a new request
- TRY READ: Requesting a read operation
- TRY WRITE: Requesting a write operation
- CHECK STATUS: Searching for the target data if it is in a cache or not, resulting either in a miss or a hit
- STALL: Simulates access to main memory
- UPDATE AGES: Update age register accordingly

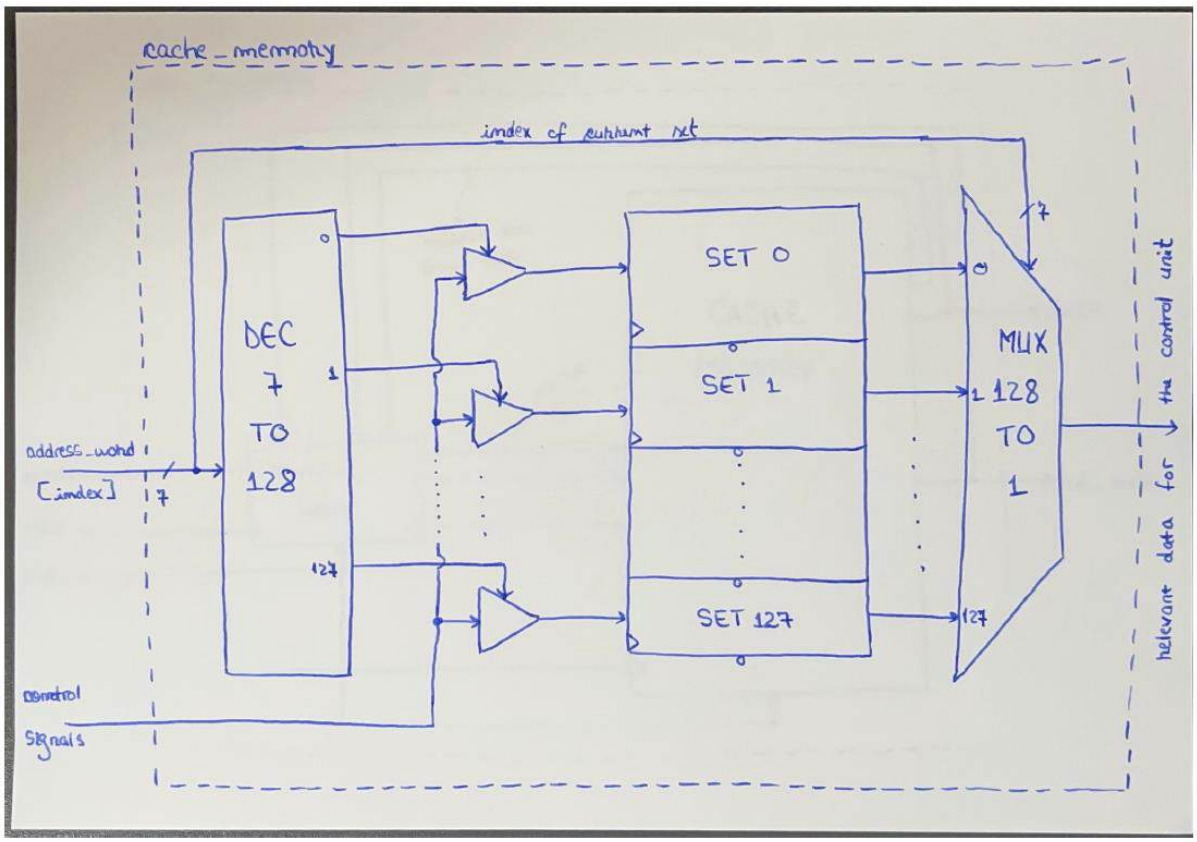
Each state is handled and tested into the HDL implementation and the cache is updated accordingly.

The detailed FSM diagram showing all states and transitions is displayed beneath:



As for the cache controller, the functional block diagram which describes it perfectly is:





Hardware Description Language

Verilog HDL is one of the most widely used languages for describing and simulating electronic systems, especially in the design of digital circuits and systems. Verilog allows designers to create a model of an electronic system at a high level, detailing the behavior and structure of the system in code that can be synthesized into physical hardware. It offers a syntax similar to C, making it relatively easy for those familiar with programming to learn and use.

More specific benefits which led us to Verilog:

- Verilog is extensively used in the industry, making skills in Verilog highly valuable for students as they enter the workforce.
- Compared to other HDLs like VHDL, Verilog has a simpler syntax and structure, which is less verbose and more straightforward.
- Verilog is supported by almost all simulation and synthesis tools, including the recommended ModelSim for wave simulation.
- Verilog is well-suited for designing and implementing finite state machines (FSMs), which is a central requirement for the cache controller project. Its capabilities to handle state transitions, timing, and concurrency are robust, making it ideal for managing the complex interactions and states in a cache controller.
- Verilog not only supports the basic design and simulation of digital logic but also allows for behavioral modeling and gate-level description. This flexibility makes it suitable for both educational purposes and real-world applications, where designs might start at a high level and then be refined down to the physical gate level.

The complete code, including professional comments explaining important sections from it, as well as the corresponding simulation, is uploaded on the GitHub link: https://github.com/elvis89b/Proiect_CN_CACHE

Moreover, it is packed in the archive sent as an attachement.

Final report

This project involved designing and implementing a cache controller for a simple computer system using Verilog HDL. The primary goal was to develop a 4-way set associative cache with a 32 KiB size and 64 bytes block size. Detailed specifications including policies for replacement and write operations were established from the request. Moreover, the address format was being discussed about previously.

The cache controller was structured around a finite state machine (FSM) to manage its operations and state transitions effectively.The design process began with the definition of the FSM states needed to handle various cache operations. Each state

was carefully programmed to manage the interactions with the cache based on the operation type and current state of the cache.

The FSM and other cache logic were coded in Verilog. Modules were created for different parts of the cache controller, such as cache memory, cache management logic, and interface logic. ModelSim was used for wave simulation to verify the correctness of the design under various scenarios and to ensure that all states transitioned correctly.

Several technical challenges arose during the project:

- ✓ Handling Concurrency: Ensuring that the cache operations did not lead to race conditions or deadlocks was challenging. This was addressed by carefully designing the FSM to manage concurrent accesses and by using non-blocking assignments in Verilog to avoid race conditions.
- ✓ Complexity of LRU Implementation: Implementing the Least Recently Used (LRU) policy for block replacement was complex due to the need to track the usage history of each cache block. A specific module was designed to update and check the LRU status efficiently.
- ✓ Write Back and Write Allocate Policies: Implementing these policies required precise control over data write-back to memory and allocation of new blocks on misses. These operations were managed through dedicated states in the FSM, ensuring that data integrity and coherence were maintained.
- ✓ In a set, data must be read/written: 4 lines in a set, we didn't want all the lines, it was a difficult time to figure out on what line is possible a hit or not. It tried to write on the all lines, and we want to redirect it directly to the source.

The performance of the cache controller was evaluated through extensive simulations. The following metrics were collected and analyzed:

- ❖ Hit Rate: The percentage of access requests that were successfully met by the cache without needing to fetch data from main memory.
- ❖ Miss Penalty: The average time taken to handle cache misses, including the time to fetch data from main memory and any necessary evictions.
- ❖ Throughput: The number of requests handled per unit time, which reflects the overall efficiency of the cache controller.

The simulations demonstrated that the cache controller achieved a high hit rate, thereby reducing the average memory access time significantly. The implementation of the LRU policy effectively minimized the miss penalty by ensuring that the least recently used blocks were replaced. The throughput measurements indicated that the cache controller could handle a high volume of requests efficiently, confirming the effectiveness of the FSM-based design.

Conclusion:

The project successfully met its objectives by designing and implementing a functional cache controller using Verilog HDL. The FSM-based approach proved to be effective in managing the complex dynamics of cache operations. The technical challenges were adequately addressed through meticulous design and careful implementation, leading to a robust and efficient cache system. The performance analysis confirmed the theoretical benefits of the chosen design and implementation strategies, showcasing significant improvements in cache performance metrics. This project not only enhanced our understanding of cache memory mechanisms but also provided valuable experience in the practical aspects of hardware module design using HDL.