Fully Associative Mapped Cache Memory System

David Murphy, Brent Simmons, Marc-Andre Couturier

University of New Brunswick Fredericton

ECE 3242

Winter 2016

Table of Contents

[Abstract 3](#_Toc446175259)

[Reference System 4](#_Toc446175260)

[Specifications 4](#_Toc446175261)

[Design 4](#_Toc446175262)

[Matrix addition 5](#_Toc446175263)

[Enhanced System 6](#_Toc446175264)

[Specifications 6](#_Toc446175265)

[Design 6](#_Toc446175266)

[References 7](#_Toc446175267)

[Appendix A 8](#_Toc446175268)

[Figures 9](#_Toc446175269)

# Abstract

The scope of this document is to describe in a concise way, the design, implementation and benchmarking of two systems. The first system, reference, is partially provided by the professor of the course. This system needs to be modified to meet the reference system requirements. The second system, enhanced, must implement a fully associative mapped cache to the reference. The two architectures must then be benchmarked to provide concrete evidence of: both architectures behaving as expected and to quantify the performance difference between the two architectures.

**Problem Statement**

We were tasked to design, simulate and implement a fully associative cache memory system in VHDL using Altera Quartus II and to compare it to a reference system without cache memory to verify the performance enhancement.

# 

# Reference System

This system is meant to be the main building block of the cached system as well as a benchmark to assess the improvement achieved by the cached system.

## Specifications

1. Main memory size should be 4kbytes of 16 bits words (address width of 12 bits). See appendix for detailed steps.
2. The speed of memory module is controlled by the memory clock, which has to be changed to 1/8 of the CPU clock.
3. System modification will require changes to the microprocessor module (instantiate the new memory module and new address width) and to the controller module (to deal with the new memory timing).

Design

The m9k memory was implemented as indicated in the project manual provided. We have used the “.mif” file format to implement the memory. A for loop was used in the top level entity to reduce the memory clock to 1/8th of the system clock. Delays had to be introduced in the controller as to assure memory was read properly. That was necessary because of the assumption that the original systems’ memory would read/write in only 1 clock cycle. However, the m9k memory needs 1 extra clock cycle to operate properly. The system was then tested and proper behavior was observed.

The m9k memory does operate at 1/8th the speed of the system clock, but it is always running. This means that the memory access time has the potential to take a variable number of clock cycles, but this kind of memory access is the same for both the reference and enhanced system! In the code for memory access, a memory ready flag is set to observe when the memory has completed reading and writing. This memory ready flag waits for two rising edges as the m9k memory takes this long to process the command. Therefore, the time it takes to access memory depends on when the read and write flags are triggered. Since the memory ready flag could be triggered at any time relative to the 1/8th memory clock, the range of cycles memory access can take is between 9 and 15 cycles per access. This is kept consistent for both the reference and enhanced systems though.

***Note*.** Things to be noted about the design choices made, ambiguities and other relevant information.

Matrix addition

The matrix addition program implements the m9k memory module with the required operations to perform a 5 x 5 matrix addition. Two new operational codes were implemented. These were necessary to provide us with the most optimized coding tools to do matrix addition on the system. The two operational codes are as follows:

1. Code 8: It is called mov5 and will read the memory location specified in a register and write the information read into a register. The pseudo code for the operation looks like this: RF[r2] <= mem[RF[r1]].
2. Code 9: It is called jz2 and will jump to a memory address regardless of the value of RF[1].

# Enhanced System

This architecture was implemented using the reference system as a base. It is to be tested using the same benchmark that was used to test the reference system. The cache system is expected to provide faster execution time compared to the reference system. Through testing, it was found that the enhanced cache system executes the benchmark program faster than the reference system.

## Specifications

1. Main memory size should be 4kbytes of 16 bits words (address width of 12 bits).
2. Memory should be implemented with M9K units as before.
3. Cache size should be 32 words of 16 bits (8 lines of 4 words).
4. Mapping scheme: fully associative.
5. Writing scheme: write back.
6. CPU will not access main memory directly but through the cache.

Design

Cache. The cache block was implemented in three parts: the cache controller, SRAM and TRAM. The cache controller coordinated all behaviors exhibited by the cache block. The TRAM holds the values of the tags’ of each lines present in cache. The SRAM holds the cached data. The general structure described above is shown in Figure 1.

***Cache controller.*** The cache controller is the interface between the CPU and its memory. All memory accesses are handled by the cache controller. The way this is implemented is through a handshaking procedure. When memory access is needed, the controller in the CPU waits and triggers a flag for memory to be accessed. The cache controller processes the request and then triggers a flag that memory is ready to be used by the CPU.

The cache controller has finite state machine. This is the way it differentiates between the incoming read/write signals and whether it can access cache memory or main memory. The following is a description of the state machine implemented in the cache controller. When a memory access request is made, the state machine starts. If memory is not needed, the cache controller resets to state S0.

State S0 reads from the TRAM tag table to check if the tag is there. A cache hit flag is set based upon the result.

State S1 decides what memory to access (cache or main) and what the operation is (read or write). This is based off cache hit flag and the read/write enable operations.

If the cache hit flag says that the tag is in cache, the proper read/write operations are set and the cache memory is accessed. The state goes to S2. State S2 is a wait state that allows for the operation to be performed on the cache memory and, importantly, signals that memory is ready for the CPU to use. S2 goes to S0 to restart the process again.

If the cache hit flag says that the tag is not in cache, this triggers a cache miss. First, the tag to be replaced is checked to see if it is ‘dirty’; this check is to see if any of the data contained in the tag has been written to/altered. If it has, this has to be written back to main memory first. If not, main memory can be accessed normally.

On a cache miss and the tag is not ‘dirty’, this means that a block of main memory needs to be brought into cache memory. This triggers a read operation and the state is called S\_mem1. There are two wait states that allow for the memory to be read called S\_mem1b and S\_mem1c. Then state S\_mem2 which writes the data obtained from memory into the cache. This state then goes to S0 to repeat the process. A cache hit should be triggered at this time.

On a cache miss and the tag is ‘dirty’, this means that a block of main memory needs to first written to memory and then the new data needs to be brought into cache memory. The state main\_write\_state is entered. This waits until for main memory and then passes to main\_write\_stateb which allows for the finishing of the write operation. This state then goes to S0 to repeat the process. A main memory read should be triggered at this time.

***Memory access.***

Replacement policy. A first in, first out policy is used when replacement is necessary.

Note. Things to be noted about the design choices made, ambiguities and other relevant information.

**References**

[1] Intel. (n.d.). Cache Architecture. Retrieved February 26, 2016, from [*http://download.intel.com/design/intarch/papers/cache6.pdf*](http://download.intel.com/design/intarch/papers/cache6.pdf)

[2] Stallings, W. (2006). *Computer organization and architecture: Designing for performance* (8th ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

**Appendix A**

**Figures**

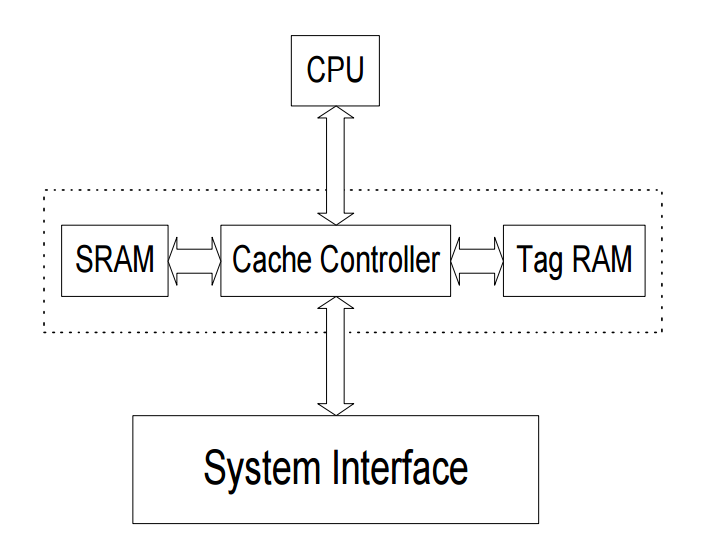


Figure 1: Look through cache structure [1]

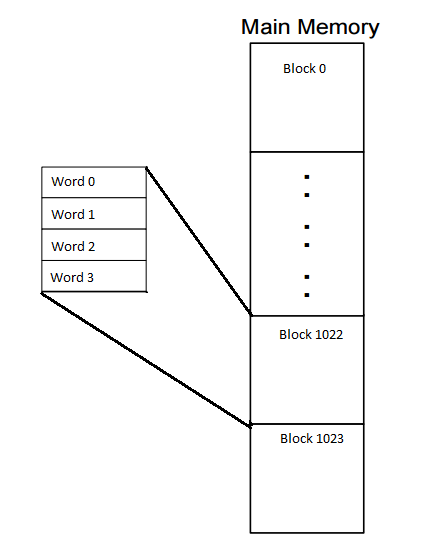


Figure 2: Block structure of main memory and cache memory [1]

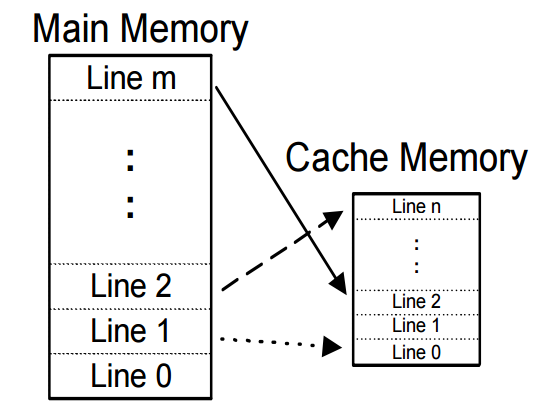


Figure 3: Full associative cache line mapping [1]