ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/319423724

An Android-based MESI Cache Coherence Simulator

Conference Paper · June 2017

DOI: 10.18638/scieconf.2017.5.1.422

CITATIONS
0

READS
88

2 authors, including:

Dimitris Kehagias
University of West Attica

20 PUBLICATIONS **26** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

MESI cache coherence simulator View project

All content following this page was uploaded by Dimitris Kehagias on 01 September 2017.

The user has requested enhancement of the downloaded file.

An Android-based MESI Cache Coherence Simulator

Dimitris Kehagias
Department of Informatics
TEI of Athens
Athens, Greece
dkehayas@teiath.gr

Ioannis Raptis
Department of Informatics
TEI of Athens
Athens, Greece
cs130003@teiath.gr

Abstract—**In multi-processor systems data can reside in multiple levels of cache, as well as in main memory. The problem of keeping the data consistent among all caches and memory is known as the cache coherence problem. There are different protocols to solve this problem. The MESI (Modified-Exclusive-Shared-Invalid) cache coherence protocol is one of them. In this paper, an Android-based educational MESI cache coherence simulator is presented that shows with animation how the MESI protocol works. This work is a continuation of our previous desktop implementation of a MESI cache coherence simulator. It is targeted to be used for teaching and learning the cache memory coherence in advanced computer architecture courses. The simulator enables interactive communication with students and is implemented in Unity Engine and Visual Studio IDE using scripts in C#.**

Keywords- MESI, Coherence protocol, Simulator, Computer architecture, Interactive animation

I. Introduction And Motivation

This paper complements our previous work in the development of a MESI cache coherence simulator [1] by presenting an Android version of the simulator. In [1], we presented the implementation of a desktop-based MESI cache coherence simulator. The Android version of the simulator will enable students to comprehend the principle of cache coherence anytime, anywhere.

In multiprocessor architectures caching plays a very important role and it is actually the key to the performance of the processor [8, 12]. However, cached architectures introduce the problem of cache coherence. The cache coherence problem arises from the possibility that more than one cache of the architecture may maintain a copy of the same memory block. The protocols to maintain coherence are called cache coherence protocols. There are two classes of cache coherence protocols in use: *Snooping and directory based*. Directory based protocols employ a directory where information about each block of physical memory is stored. This way the directory has full control over which core has loaded which cache block. In snooping based protocols, rather than keeping the information about each block in a single directory, every cache that has a copy of the data from a block of physical memory could track the sharing status of the block.

Snooping protocols because of simplicity and low overhead are most commonly used in commercial multiprocessors. The bus that connects cores, caches and shared memory constitutes a convenient broadcast medium to implement the snooping protocols. One of the most commonly used types of snooping protocols is the MESI cache coherence protocol.

For students in an undergraduate advanced computer architecture course, cache coherence protocols are often confusing as they are not always that distinct. That's why cache coherence simulation tools are used to support learning [14, 15, 16].

Our intention to build a MESI simulator was motivated by the fact that many students, in the undergraduate advanced computer architecture course offered by the Informatics department of the Technological Educational Institute (T.E.I.) of Athens, exhibit difficulties understanding the cache coherency problem.

Although there are already similar tools created to simulate the MESI protocol [2, 5, 11, 13], our aim was to develop a simulator which on one hand can easily be used by the instructor to effectively teach concepts related to the internal functions of the MESI protocol and on the other hand to provide an attractive and easily understandable tool for the students to assimilate these concepts. The major driving force was to introduce animation to aid the students to really understand the inner workings of the MESI protocol. The simulator supports:

- Interaction with the user for configuring the blocks in memory.
- Animation during a read or a write (hit or miss) in a cache line.
- Written explanations in every animation step.
- Twelve ready scenarios that implement all the functioning of the MESI protocol.
- Simulated execution to proceed in variable-speed timed mode with interactive display update speed adjustment.

The rest of the paper is organized as follows. Section II explains the MESI protocol. Section III presents an overview of simulator implementation, its functioning and features. Section IV concludes the paper and discusses possible future work.

II. MESI Coherence Protocol

The MESI protocol (known also as Illinois protocol due to its development at the University of Illinois at Urbana-Champaign [6]) is a widely used cache coherence protocol. It is the most common protocol which supports write-back cache where a cache line can be written multiple times before the memory is updated. It is based on the four states that a cache

eISSN: 1339-9071, cdISSN: 1339-3561
ISBN: 978-80-554-1337-2

line can be. These four states are the abbreviations for MESI: Modified, Exclusive, Shared and Invalid [9, 10].

Modified (M): The value of the cache line has been modified and is different from the copy located in memory. A write back must be performed in future, before permitting any other read of the memory.

Exclusive (E): The cache line is present only in the current local cache and is the same with the copy located in memory. It may be changed to Share at any time, in response to a read request. It may also be changed to Modified state when writing to it.

Shared (S): The cache line may be replicated in more than one cache and is the same with the copy located in memory.

Invalid (I): The value of the cache line is not valid, so it should not be used.

The state diagram in Fig. 1 shows the possible state transitions of a cache line. Here are some explanations. When the block is first read by a core, if a valid copy exists in another cache (condition PrRd/BusRd(S), in Fig. 1), then it enters the core's cache in shared state. However, if no other cache has a copy at the time (condition (not s), in Fig. 1), it enters the cache in exclusive state. Then if this block is written by the same core (PrWr/-), it changes to modified state meaning that the block which is in main memory is different to it. In a read request from another cache for an exclusive block (BusRd/Flush) causes a state transition from exclusive to shared. A write request from another cache (BusRdX/Flush) causes the invalidation of the block.

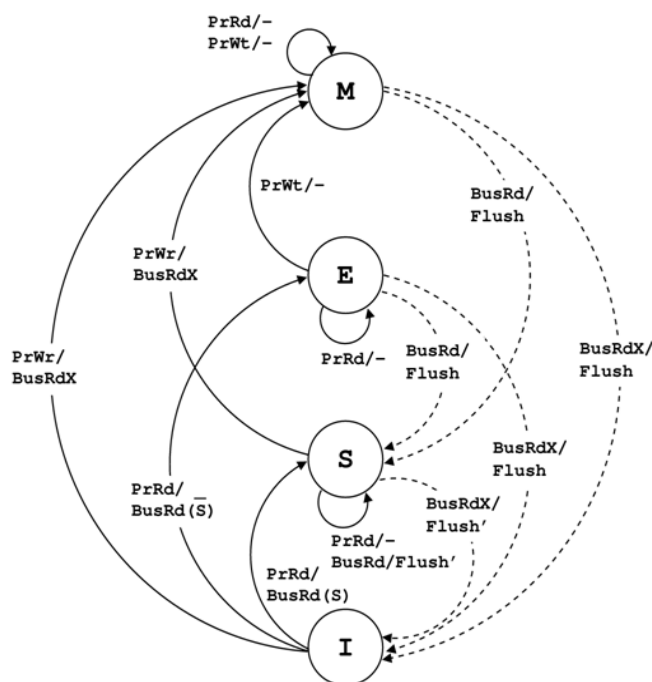

Figure 1. [4]: MESI state diagram.

III. The MESI Simulator

A. Functional Description

The operation of the simulator is described by examining the events that take place in a local core. Events may be either due to local core activity because of cache access (read hit/miss, write hit/miss) or due to bus activity as a result of snooping. Both kinds of activities are requests to a local cache controller which takes the appropriate actions.

Bus transactions are all initiated by cache controllers responding to requests from their associated cores. There are three bus transactions 'BusRd', 'BusRdX' and 'BusWB', post by cache controllers. With the BusRd (Bus Read) transaction the controller asks for a copy with no intent to modify it and the data could come from memory or another cache, while with the BusRdX (Bus Read-Exclusive) transaction asks for a copy with intent to modify it. In the BusRdX transaction must invalidate all other caches copies. With the BusWB (Writeback) transaction the controller puts a copy on the bus and the memory is updated.

Fig. 2 shows the activity diagram of the simulator in case of a Read Hit/Miss.

Local Read Hit: In case of a local read, a core creates a "PrRd" (processor read) request to a cache controller. If the controller finds the requested word in local cache then the read is handled as a "Read Hit". This can happen when the cache line containing the requested word is in one of the M, E or S state. The word is read by the core with no state change in cache line. (Fig. 2)

Local Read Miss: In a local read, when the controller doesn't find the requested word in cache or the cache line that contains the word is in state I then the read is handled as a "Read Miss" where the cache controller posts the bus transaction "BusRd" asking for a copy with no intent to modify it. There may be the following cases:

(a) *Only one cache contains the requested copy in state E.* Snooping cache puts the copy on the bus; the local core caches the copy and both cache lines set to state S.

(b) *There is no other copy in caches.* The bus request "BusRd" is intended to memory and the copy read from memory to local cache, marked E.

(c) *Several caches have the requested copy in state S.* One cache (arbitrarily) puts the copy on the bus; the local core caches the copy, marked S, with other copies remaining in state S.

(d) *Only one cache contains the requested copy in state M.* Snooping cache puts the copy on the bus and the local core caches the copy, tagged S. In addition, snooping cache posts the bus transaction "BusWB", writes back its copy to memory and changes its copy from state M to S.

Figs 3 and 4 show the activity diagrams of the simulator in case of a Write Hit/Miss.

Local Write Hit: In case of a local write, a core makes a "PrWr" (processor write) request to a cache controller. If the controller finds the requested word in local cache then the write

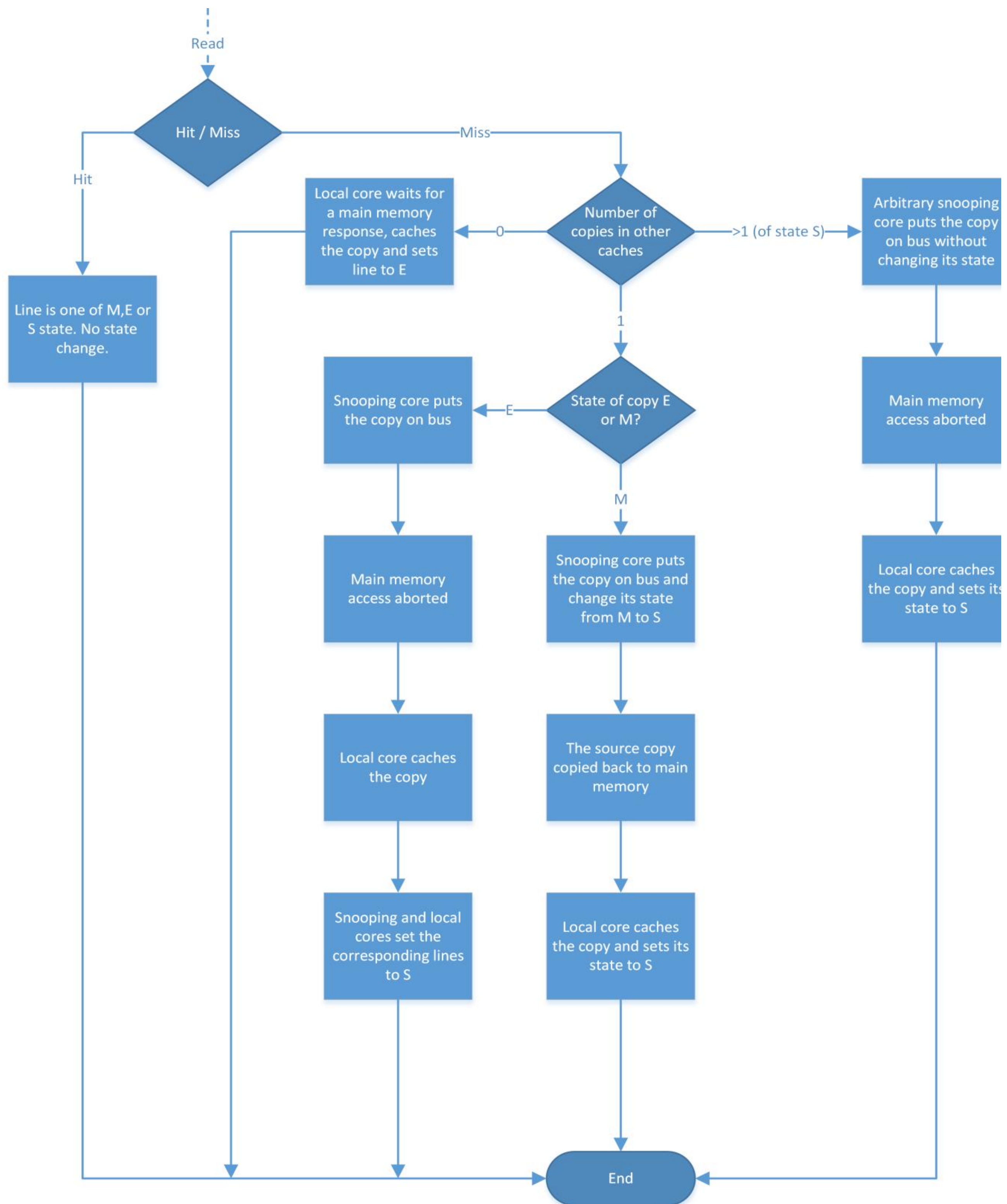

Figure 2. Activity diagram of the simulator in case of a Read Hit/Miss

is handled as a “Write Hit”. This can happen when the cache line containing the requested word is in one of the M, E or S state. If the cache line is in M state the local core just updates its value with no state change, while in state E updates its value with state change from E to M. When the cache line is in state S the cache controller broadcasts invalidate on the bus, the snooping cores with an S copy change state from S to I and the local cache value is updated with state change from S to M.

eISSN: 1339-9071, cdISSN: 1339-3561
10.18638/scieconf.2017.5.1.422

ISBN: 978-80-554-1337-2

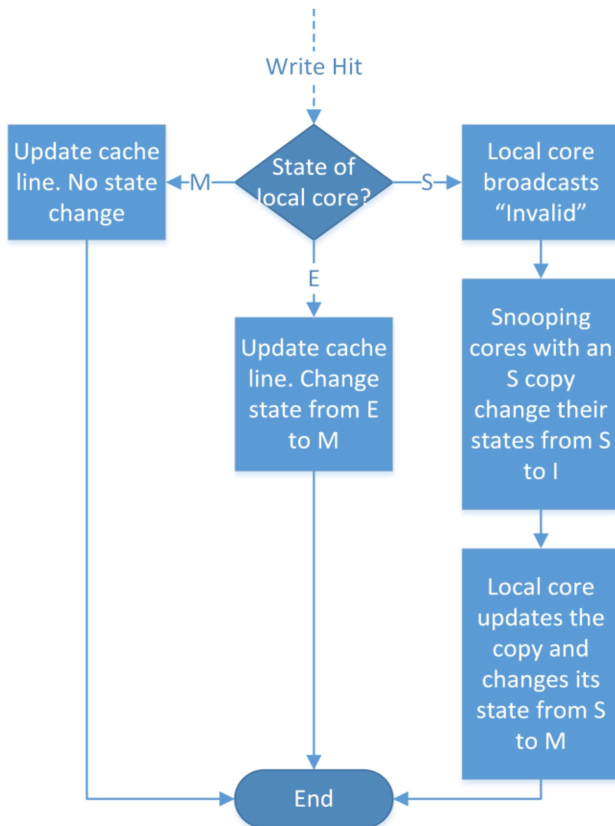

Figure 3. Activity diagram of the simulator in case of a Write Hit

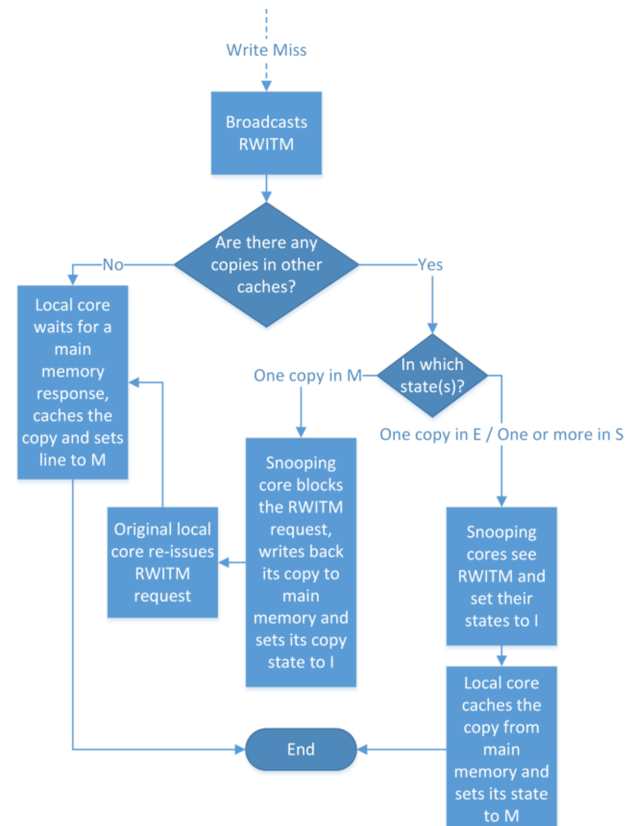

Figure 4. Activity diagram of the simulator in case of a Write Miss

Local Write Miss: In a local write, when the controller does not find the requested word in cache or the cache line that contains it is in state I then the write is handled as a "Write Miss" where the cache controller posts the bus transaction "BusRdX" asking for a copy with intent to modify it. There may be the following cases:

(a) *There is no other copy in caches.* The bus request "BusRdX" is intended to memory; the copy read from memory to local cache, updated and marked M.

(b) *Only one cache contains the requested copy in state E or several caches have the requested copy in state S.* The bus request "BusRdX" is intended to memory; the copy read from memory to local cache, updated and marked M. The snooping cores see the "BusRdX" request and change their copy state from E or S to I.

(c) *Only one cache contains the requested copy in state M.* Because of "BusRdX" request, the snooping core posts the "BusWB" request, writes back its copy to memory and changes its copy state from M to I. The local controller re-issues "BusRdX" request; the copy read from memory to local cache, updated and marked M.

B. Overview of Simulator Implementation

The simulator was developed in Unity Engine Version 5.3.4f1 personal edition [3,7]. All scripts were written in C# using Microsoft Visual Studio Community 2015. Unity Engine was chosen because of the attractive GUI system that provides.

GameObject is the base class for all entities in Unity scenes. MonoBehaviour is the base class every script derives from. The following classes were used to develop the simulator.

- *Public class terminal: MonoBehaviour.* This is the most important class directly connected with READ and WRITE buttons, it is the class which selects and triggers the appropriate methods in accordance with the user input.
- *Public class busrd_c1: MonoBehaviour, public class busrd_c2: MonoBehaviour, public class busrd_c3: MonoBehaviour, public class busrd_m : MonoBehaviour.* These four classes include methods to illustrate the various signals generated within the processor. Game Objects of these classes are subject to movement using the Box2D physics system that is integrated into the Unity Engine.
- *Public class credits:* MonoBehaviour. GUI Buttons handler: Open/Close the Info window.
- *Public class dme: MonoBehaviour.* GUI Buttons handler: Open/Close the Scenario and Help windows. Triggers Initialize. Scene change. Application quit.
- *Public class glow: MonoBehaviour.* Adds the flickering line attribute to the selected blocks and lines.
- *Public class init_msi_snoopy: MonoBehaviour.* It bears the Initialize and Random Words insertion method.

10.18638/scieconf.2017.5.1.422
eISSN: 1339-9071, cdISSN: 1339-3561
ISBN: 978-80-554-1337-2

- *Public class msi_snoopy_button: MonoBehaviour.* GUI Button's handler: Scene change from Main Menu to main program screen.
- *Public class scenario: MonoBehaviour.* GUI Buttons handler: Choose a scenario menu.

C. Simulator Features

Fig. 5 shows the graphical interface of the simulator whose features include:

- *Three cores with local caches.* The caches are direct mapped with a write back policy. The local cache of each core has four cache lines (LN0-LN3) and the cache-line/block size is four words. The column titled STATE displays the current state (M, E, S, or I) of each cache line. The simulator doesn't concern itself with byte addressing within words, word alignment and so on. The simulator accepts its input from users. A user specifies a word in the "Enter Word…." frame and starts a read/write transaction on the specified word by pressing the READ/WRITE button. A word consists of a letter (A-Z) and a digit (0-9). In order to start a read/write operation the contents of memory must previously be set by pressing the RND Words button. The words are set randomly.
- *A main memory* containing sixty four words organized as sixteen memory blocks (BL0-BL15) with four words each. Memory is addressed at block level and data addresses start from zero. Thus, address zero indexes the first block in memory, address one the second block, and so on.
- The local caches and the main memory are connected by *a bus* that acts as a communication network.
- *LOG info panel* that shows which read or write request is being executed.
- With the *Initialize button* the contents of all caches and main memory are cleared, while with the *RND Words button* the contents of memory are specified randomly.
- The simulator permits simulated execution to proceed in variable-speed timed mode with interactive display update speed adjustment using the *frame Speed: (5 to 100).* Speed of 10, 50 and 100 correspond to 10, 2 and 1 second(s) per step respectively.
- By selecting the *Scenario button* twelve ready case studies are displayed that implement all the functioning parts of the MESI protocol.
- Description of the main interface is given by selecting the *Help button*.
- *Main menu button* returns to main menu.

IV. Conclusions And Future Work

A tool to aid students and teachers in an undergraduate advanced computer architecture course was presented. This tool, an Android-based MESI simulator for a write-back cache, implements the MESI snoopy cache coherence protocol. Each

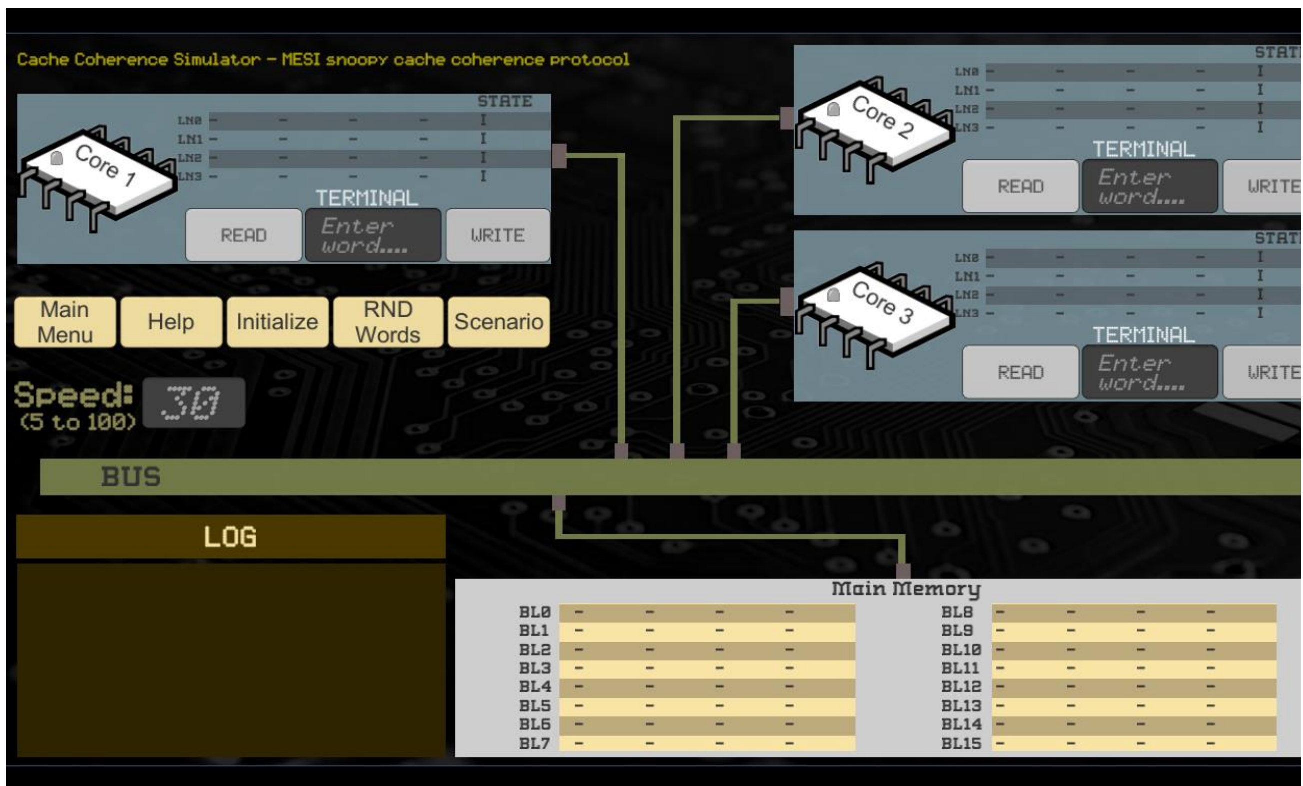

Figure 5. Graphical Interface

10.18638/scieconf.2017.5.1.422
eISSN: 1339-9071, cdISSN: 1339-3561
ISBN: 978-80-554-1337-2

step of the simulation is represented with animation and with reference to a text area in order to give a clear picture of which read or write request is being executed. Initial use of the simulator has shown learning effectiveness. In near future the simulator will be evaluated in the classroom through student surveys. Lastly, both versions one for English language and one for Greek language will be integrated into one version.

REFERENCES

[1] D. Kehagias and I. Raptis, "An Interactive MESI Cache Coherence Simulator for Educational Purposes", In the ACM Conference Proceedings of the 20th Pan-Hellenic conference on Informatics (PCI 2016), Patra Greece, doi>10.1145/3003733.3003765, Nov. 10-12, 2016.

[2] Gomez-Luna, J., Herruzo, E. and Benavides, J. I., "MESI Cache Coherence Simulator for Teaching Purposes". CLEI ELECTRONIC JOURNAL. 12, 1, 2009.

[3] Unity User Manual: http://docs.unity3d.com/Manual/index.html.

[4] https://en.wikipedia.org/wiki/MESI_protocol.

[5] Laguens, A. A., Mir, S.B. and Quintana Orti, E.S., "An Interactive Animation for Learning How Cache Coherence Protocols Work". In proceedings of INTED2011 Conference, 7-9 March, 2011, Valencia Spain.

[6] Papamarcos, M. S. and Patel, J. H., "A low-overhead coherence solution for multiprocessors with private cache memories". In ISCA '84: Proceedings of the 11th annual international symposium on Computer architecture, ACM , New York, NY, 348-354, 1984. DOI= http://doi.acm.org/10.1145/800015.808204.

[7] Joe Hocking, "Unity in Action: Multiplatform Game Development in C# with Unity 5". Manning Publications, 2015.

[8] Hennessy J.L. and Patterson D.A., "Computer Architecture: A Quantitative Approach". Morgan Kaufmann Publishers Inc., Fifth Edition.

[9] MESI Two Level. Available at http://www.m5sim.org/MESI Two Level, Oct. 2016.

[10] The gem5 Simulator System, "A modular platform for computer system architecture research". Available at http://www.gem5.org, Oct. 2016.

[11] VivioJS - Interactive Reversible E-Learning Animations for the WWW. Available at https://www.scss.tcd.ie/Jeremy.Jones/vivio/vivio.htm. Oct. 2016.

[12] Patterson, D., Hennessy, J., "Computer Organization and Design (5th ed.)", Morgan Kaufmann, 2014.

[13] Somdip Dey and Mamatha S. Nair, "Design and Implementation of a Simple Cache Simulator in Java to Investigate MESI and MOESI Coherency Protocols". International Journal of Computer Applications, Vol. 87, No. 11, pp. 6-13, 2014.

[14] A. Misev, M. Gusev, "Simulators for Courses in Advance Computer Architecture". International Journal of FACTA UNIVERSITATIS (series: Electronics and Energetics), Vol.18, No.2, pp. 237-252, 2005.

[15] M. Gusev, A. Misev, and G. Popovski, "Simulation of superscalar processor," in proc. of ITI'98, Pula, Croatia, pp. 169–174, 1998.

[16] A. Misev and M. Gusev, "Supersim v2.0 ilp processor visual simulator," in Computation Intelligence and Information Technologies, Proceedings, R. Stankovi´c, Ed. Nis, Yugoslavia: Faculty of Electonic engineering, June 20-21, 2001, pp. 161–166.

View publication stats