ARMSim# – a Customizable Simulator for Exploring the ARM Architecture

R. N. Horspool, W. Lyons and M. Serra
Department of Computer Science, University of Victoria, Victoria, B.C., Canada V8W 3P6

Abstract – *ARMSim# is a simulator for the ARM architecture intended for use in both teaching and research. The simulator provides some standard features found in debuggers such as breakpoints, single-step execution and watch-points. Its innovative features include support for software plug-ins which can implement external devices and new instructions. Such plug-ins can be used to prepare students for working on actual development boards, and by researchers for experiments with simulated custom I/O boards and custom extended instructions. Another powerful feature of ARMSim# is its ability to select a cache configuration and then observe the cache hit ratios and execution times as measured in clock cycles*

Keywords: Simulator, ARM, Instruction Set Architecture

1 Introduction

The ARM processor has become a major force, excellent both as a product and for its associated development tools. It accounts for a large portion of the design market in embedded systems which represent 90% of all processors sold. It is important for both students and researchers to be able to work easily and to experiment with various designs deploying the ARM and its features. Some texts on computer architecture now cover the ARM, including [1,2]. However, ARM development boards and the commercial software development environments needed to work with the ARM can be relatively expensive, especially for use in undergraduate courses, and not necessarily suitable.

The SPIM simulator [3] for MIPS has been extremely popular for RISC architecture courses, as well as in other related projects. Our software tool, ARMSim#, was inspired by SPIM. However it is a total re-design, featuring a powerful graphical user interface, extra functionality and features not currently available in most such platforms.

The first function of ARMSim# is indeed to be a powerful simulator for the ARM7TDMI architecture. The more exciting part is the introduction of new features intended to help instructors and students: These include:

- cache simulation, where the cache characteristics can be customized for performance experiments;
- program timing measured in clock cycles;
- linking with cross compiled C code;
- compatibility with a commercial ARM-based board for embedded systems design [4];
- automated testing of ARM programs;
- easily created extra software plug-ins for visual simulation of custom I/O and for extensions to the user interface;
- ability to introduce custom new ARM instructions.

ARMSim# is implemented in C# and requires the .NET Framework. It runs on Windows, MacOS and Linux. The latter two operating systems require the mono implementation of the .NET framework.

2 The ARMSim# Basic Design

The main functionality of ARMSim# is as a simulator for ARM programs. In this context it includes assembling, debugging and executing, with the most important goal being ease of use even for students or users with little experience. A user simply opens an assembly language source code file, without having to go through the separate steps of assembling and linking a program. In accepting source input in this manner, it follows the SPIM paradigm.

However, ARMSim# can also execute programs composed from multiple files. These files can be either source code files or object code files (*.o files). The latter are in ARM Elf format and can be generated by the assembler or the gcc C compiler included in the CodeSourcery GNU toolchain for ARM processors [5]. ARMSim# also has the capability of searching libraries (*.a files) for object code files needed to define unresolved symbols, in the same manner as a linker.

Although ARMSim# is not primarily intended as a debugger, it provides many of the same features, including:

- A source code display window which shows the next instruction to execute; it allows breakpoints to be set on instructions.
- Various modes of execution – run, step into, step over and step out.
- Windows which display the register contents, regions of memory, and an easy-to-interpret dynamically changing view of the program stack.

- A watch window for program variables which allows the user to see the changing contents of data regions of memory as the user single-steps through the program.

Figure 1 shows a snapshot of ARMSim# in operation, just after the first instruction of the program has been executed in single-step mode. The current instruction, which will be the next one to be executed, is highlighted in the large source code window. The window at the left shows the registers, with the ones changed by the previous instruction displayed in red.

In the Windows version of the program, all the windows are *docking windows*, customizable for size and placement to enhance visibility. (Limitations of the mono environment prevent this feature from working on MacOS and Linux.) Effort has been expended on developing a User Interface which follows the best guidelines in HCI for GUI and usability criteria, in order to make the functionalities of ARMSim# be as effective as possible and extremely user-friendly.

Using these basic capabilities, a student can learn to use and experiment with the instruction set of the ARM. Input for the program can be obtained from pre-defined data variables and results can be stored into memory locations. At the end of execution, the contents of memory can be observed to verify that the correct results have been obtained.

For more ambitious and useful coding, ARMSim# provides predefined system functions, accessed via the ARM *swi* (software interrupt) instruction. With them, any program can read from the keyboard ("stdin"), write text to a separate display window ("stdout"), perform operations on text files (open, close, read and write), display messages in alert boxes, obtain the current internal clock time, and allocate blocks of storage from the heap.

The additional features to simulate custom I/O devices accessed through a bus are described below.

3 Cache Simulation and Timing

Few simulators provide a faithful emulation of the cache, and development boards do not provide facilities for inspecting the contents of the cache. This is unfortunate because the cache is often a very difficult concept to understand and experiment with vis-à-vis performance issues. One of our goals with ARMSim# was to rectify that omission. By displaying the cache contents while a program is being single-

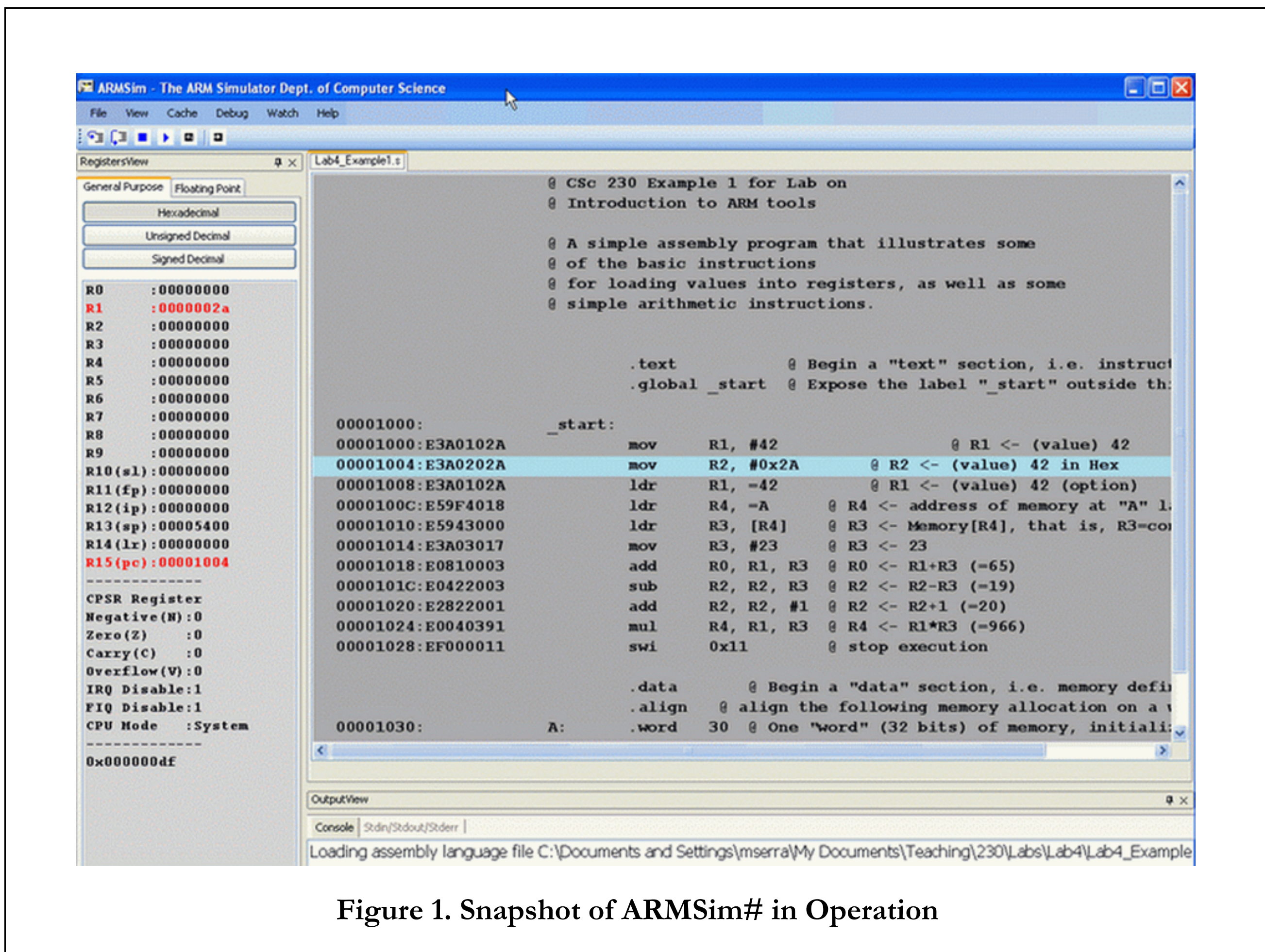

Figure 1. Snapshot of ARMSim# in Operation

stepped, the ARMSim# can help a student understand what is going on.

A more advanced user may be more interested in the different possibilities for cache design. Should it have separate *I* and *D* caches, or should they be combined? Should the cache be direct mapped or set-associative? What line size and set size should be used? What happens as the cache size is changed? What replacement strategy is used with set-associative or fully-associative cache? All these design choices can be explored with ARMSim#. Moreover ARMSim# provides a wizard to navigate through them and select a consistent set of parameters for the cache.

With the cache in operation, the user can stop the program at various points to check the cache contents, can watch the cache updates take place, or can simply run the program to completion and observe both the cache hit ratio and the simulated execution time (measured in both clock cycles and real time).

Figure 2 shows a screen snapshot of the ARMSim# menu which comes up when setting the parameters of the data cache.

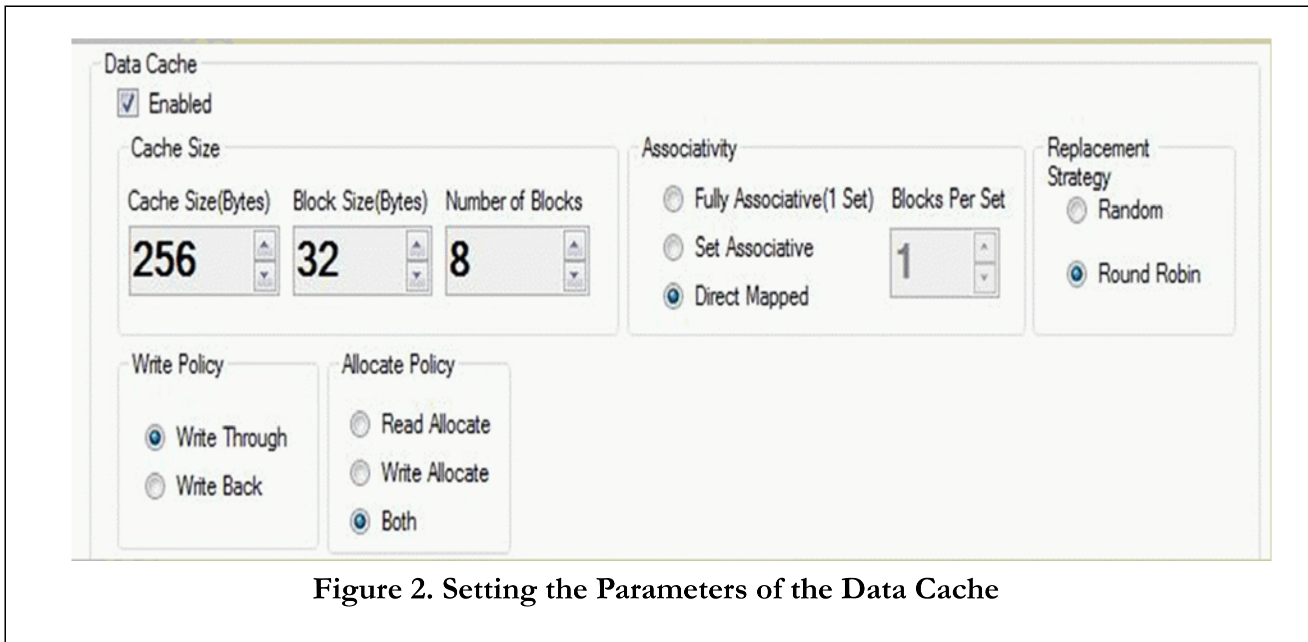

Figure 2. Setting the Parameters of the Data Cache

During and after execution, detailed timing statistics are also collected. The cycles used in the various modes of execution provide the data needed to support any future design decision. Especially when running larger programs, the effect on performance of the different cache configuration become apparent through experimentation, yet still within the framework of an easy-to-use interface and engine.

4 Custom I/O Devices and Boards

A processor can be uninteresting when used in isolation; it needs to be connected to I/O devices before any useful programming is possible. Although a keyboard and monitor can be attached via an interface card, it is more likely that other kinds of devices are used in an embedded system. Which devices are used and how they are accessed by ARM instructions depends on the manufacturer of the system.

To provide a high degree of flexibility in system configuration, ARMSim# supports software plug-ins. It is relatively easy for a course instructor to create a plug-in file as a *dll* file located in the same folder as ARMSim#. The plug-in can simulate buttons, sensors, timers, LED lights, buzzers, etc, to name just a few of the possibilities. A recent project asked the students to control the traffic lights at a street intersection where pedestrian buttons could be pressed and affect the sequence of lights. The newly designed custom plug-in displayed the traffic lights and provided the pedestrian buttons.

The laboratory for the computer architecture course at the University of Victoria provides ARM development boards attached to PCs for the students to use. The process of debugging a program on the board is a frustrating task for a novice student. The IDE provided with the board itself is powerful, yet not easy to learn quickly and use effectively, especially by a novice. We made life much easier for the students by implementing a simulated version of the development board as a plug-in for ARMSim#. A snapshot of the actual board and of its simulated equivalent are shown in Figure 3. The board has a small LCD screen, a keypad with 4 rows and 4 columns, two buttons, an 8-segment LED display, and two LED lamps. These input and output devices have all been simulated using controls available with Windows Forms (i.e. a text window, buttons, and image icons).

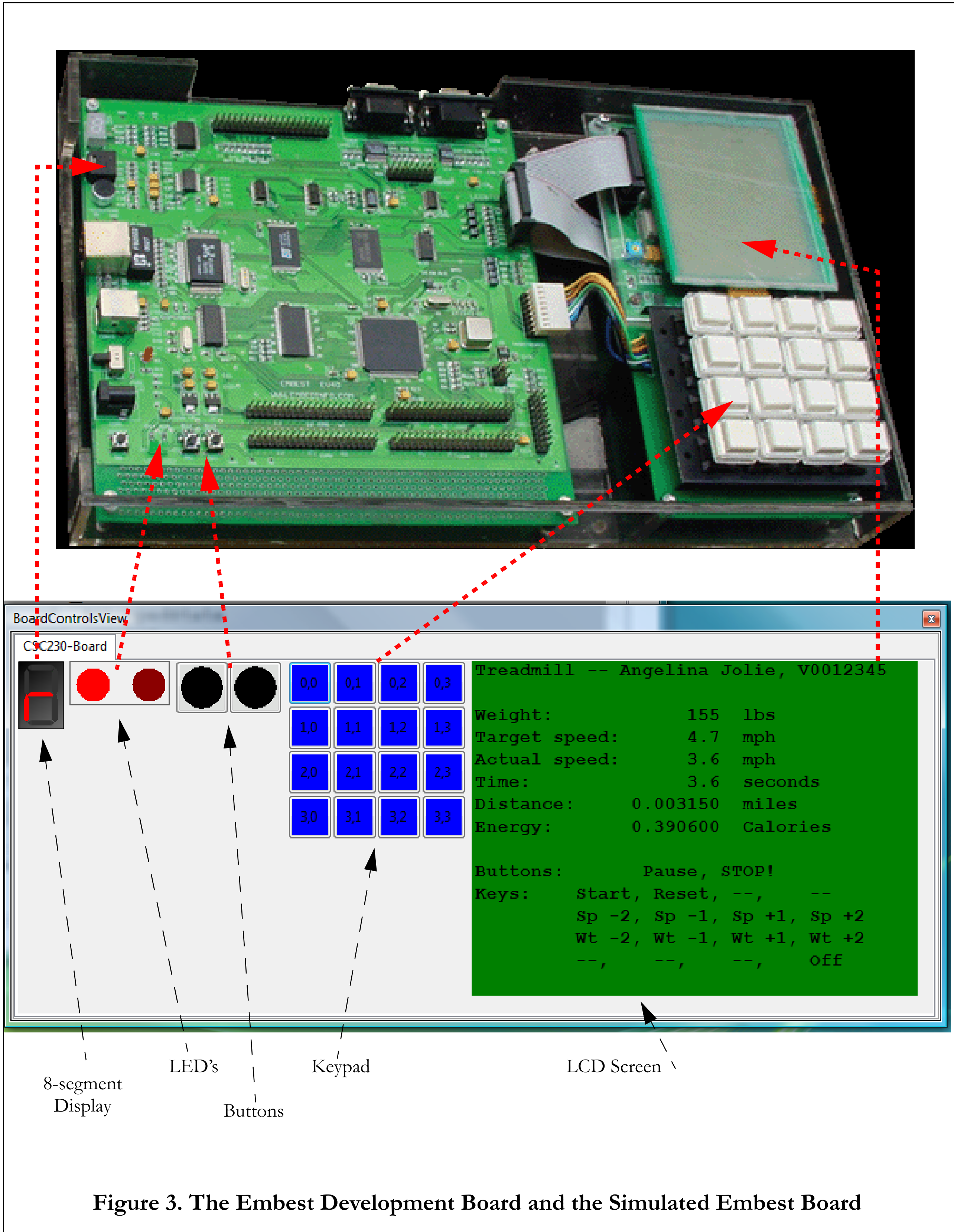

Figure 3. The Embest Development Board and the Simulated Embest Board

5 Simulating Hardware

Most simulators simply display windows which contain information about the executing program and in that ARMSim# is similar. However, when a plug-in is created, accesses to the memory-map are intercepted by ARMSim# and control is passed to the plug-in callback (delegate). The plug-in can simulate virtually any type of hardware device that uses memory-mapped I/O (including hardware timers). Plug-ins have been written to date to simulate a wide range of hardware. They include the I/O devices of the Embest development board, as described above, and the push buttons and lights of the traffic light simulation.

A powerful feature of the plug-in pattern is the ability for the end user to extend the functionality of ARMSim# without requiring modification of the simulator itself. ARMSim# uses a C# interface that defines the communication between the simulator engine and the plug-ins. The interface creates a strict contract that allows the two entities to interact with little coupling. Future versions of ARMSim# can be published and users can upgrade without fear of losing the time invested in developing a plug-in.

6 Extending the ARM Instruction Set

In addition to simulating hardware, plug-ins can also be developed to simulate new, hypothetical, ARM instructions. The ARM instruction set has some patterns that are not recognized by the ARM processor and can be used to define new instructions [6, page A3-27]. When a new instruction using one of the unrecognized instruction patterns has been defined, the ARMSim# engine will intercept any attempt to execute it and pass control to the responsible plug-in. The plug-in can inspect the opcode and perform a custom action for the new instruction.

For example, a plug-in can be written to control a hypothetical 64-bit general purpose register. A set of instructions can be defined to manipulate this register (add, subtract, multiply/accumulate, move contents etc). Binary opcode patterns can be assigned to these new instructions from the pool of unused opcodes of the ARM instruction set and a plug-in written to handle these instructions. Once deployed, the ARMSim# will pass control to the new plug-in every time one of these new instructions is executed. The plug-in maintains an internal 64-bit value and manipulates it based on the instructions executed. As another convenience, the plug-in can implement a user interface extension to visualize the 64-bit register as an extra docking window inside the ARMSim# user interface.

In addition to defining new instructions for the ARMSim# engine, instruction mnemonics can be inserted into the parsing tables for the build-in assembler. This allows new instructions to not only be simulated, but also parsed and assembled in source code files.

The most important aspect to note in all of the above is the ease with which the extensions can be developed and configured as part of ARMSim#, without extensive programming or needing to touch at all the main module.

7 Developing Plug-Ins

Plug-ins are implemented as ordinary .NET assemblies with types that implement a defined ARMSim# interface. A developer of a plug-in can use any .NET language in conjunction with Microsoft Visual Studio to create the assembly.

Since the .NET framework was used as the development platform of ARMSim#, the Forms services are available to plug-in writers for user interface extensions. Using the Forms editor, developers can quickly create user interface elements to represent simulated hardware.

A starter kit is included with the ARMSim# installer that allows rapid development of plug-ins within Visual Studio. The starter kit generates a skeleton plug-in in either C# or VB.NET with documented interface members and examples of plug-in implementation. A developer with limited experience can be up and going developing ARMSim# plug-ins quickly.

In addition to starter kits, the ARMSim# website has some example plug-ins that can be studied and used as the basis for a new plug-in. The use of the Embest board as an experiential basis has made the main board plug-in the most natural and useful. Programs which will eventually be downloaded to the board can be fully tested with ARMSim# and the very realistic interface given by the plug-in (without worrying about key bounces, for example). After the final version is finished, the code can be imported into the IDE used for developing programs for the hardware board, the program can be compiled and the executable file downloaded to the board for execution.

8 Use of ARMSim#

8.1. In the Classroom

We have deployed ARMSim# in the course laboratory and have made the program available to students for installation on their own computers. Allowing students to have their own copies of ARMSim# reduces pressure on laboratory resources while also being a great convenience.

While the design and use of ARMSim# in a course is mainly aimed towards pedagogical goals, it also has extra features incorporated to simplify life for the instructor.

1. ARMSim# can be executed as a batch program with a script that can check the contents of memory locations in the ARM program at the end of execution. This permits an instructor to automate the testing of submitted code.
2. The instructor can pre-compile C subroutines or pre-

assemble ARM source code subroutines and package them in an archive library. Students can use these subroutines in their programs.

8.2. For Research and Experimentation

Any IDE or simulator can be obviously used in many domains for a variety of purposes. An ARM simulator could indeed confine itself to the compilation and execution of appropriate programs, together with all sorts of extra features to support the development environment. Its use in research may be seen as simply one of the many tools.

It is important to emphasize though that the extra features of ARMSim# related to the cache manipulation and to the timing statistics can be an extremely useful asset in research experiments and in case studies for higher level learning environments. Few simulators provide such support (SPIM doesn't) and certainly not with the same level of flexibility and ease for configuration.

9 Conclusions

ARMSim# has been in use for more than two years at the University of Victoria in a Computer Science course which is offered 3 times per year. The students learn the basic ARM assembly language and C programming, and combinations of the two in a single program in the labs attached to the introductory course on computer architecture. The hands-on practice at the low level enhances the understanding of concepts for computer organization and system architecture. An architectural variant of the ARM known as ARM7TDMI is supported, as well as Vector Floating Point (VFP) instructions. It is an extremely useful instrument to convey most of the concepts of architecture together with practical implementations and developments.

Simulation is not always a substitute for the real thing. If boards and IDE's can be purchased, they can be great learning tools, yet sometimes it takes a while to become familiar with a commercial IDE. The great advantage of ARMSim# is that it is easily extended to display a simulated development board functioning exactly like the real thing. Even when actual boards are available (as in our course), it is valuable to gain experience and verify correctness first with the simulator, before tackling the added complexity of a commercial development environment, the frustrations of cross-compiling and the sometimes tricky steps of downloading to the real board.

Moreover being able to attach a plug-in representing a more interesting (or fun) interface can bring added value to a course. In the example of a traffic light controller, the functionality was expressed effectively through the use of black and blue buttons, blinking LED lights and the 8-segment LED display. However the (fun) graphical plug-in which showed a simulated traffic light using the same *swi* instructions for controlling the light's operation made for a more realistic project. It was a project which indeed seemed much closer to a real embedded control system, even for novice users.

The ARMSim# program is freely available for academic use by contacting the authors of this paper and following the links from their web pages.

10 References

[1] C. Hamacher, Z. Vranesic, S. Zaky, *Computer Organization*, fifth edition. McGraw-Hill, 2002.

[2] A. Clements, *Principles of Computer Hardware*, fourth edition. Oxford University Press, 2006.

[3] J. Larus. SPIM – A MIPS32 Simulator. URL: `http://pages.cs.wisc.edu/~larus/spim.html` (checked 24 Feb 2009).

[4] Embest Info & Tech Co., Ltd. Embest S3CEV40 Evaluation Board. URL: `http://www.armkits.com/product/s3cev40.asp` (checked 24 Feb 2009).

[5] CodeSourcery. GNU Toolchain for ARM Processors. URL: `http://www.codesourcery.com/sgpp/lite/arm/` (checked 24 Feb 2009).

[6] D. Seal (editor). *ARM Architecture Reference Manual*, second edition. Addison-Wesley, 2001.