An Application Development Environment for a SoC with a Simulator in SystemC

Augusto José de Oliveira Martins
augustoom@yahoo.com.br

José Camargo da Costa
camargo@ene.unb.br

Universidade de Brasilia - Faculdade de Tecnologia - Departamento de Engenharia Eletrica - Caixa Postal 4386 - Brasilia - DF - CEP 70904-970

Abstracts

An application's development environment which integrates an editor, an assembly tool, a simulator, and a debug tool aimed to a SoC is described. The SoC has a 16 bits RISC processor, memories, an RF transceiver, digital interface and A/D interface. The main feature of the development environment is to provide the possibility to test some piece of code even in the absence of a fully functional prototype of the SoC. The simulator is implemented with SystemC; the editor and the debug tool are being developed as Eclipse plug-ins; the assembly tool is not presented here.

1. Introduction

Our University, together with seven other institutions, is developing a system to control irrigation on crops [1]. The system is composed by a base station, field stations and nodes. The field stations gather information from the nodes through a wireless link, accurately identify areas of moisture deficiency and send instructions to the nodes, determining which ones should act on latch solenoid valves, in order to deliver the required amount of water to the plants. Each node is composed by a CMOS 0.35µm SoC (System on Chip), a tensiometer with a solid state pressure sensor, an actuator which controls the water flow through a solenoid valve, a solar powered power supply, an RF antenna, and embedded softwares. The SoC consists of a RISC microprocessor, memories, an RF transceiver, digital interface and A/D interface.

In this context, a development environment is being implemented to provide programmers with tools to produce and test their piece of software for that SoC. An assembly tool has been implemented, which is basically a two step assembler. It was made in such a way that it is possible to change, add or delete the pseudo-instructions just by editing a text file.

Two technologies are being used to implement the other parts of the development environment: the Eclipse Platform [2] and the SystemC Library [3]. The first is an open platform for tool integration that provides an open, extensible, general purpose Integrated Development Environment which can be extended by adding plug-ins. The Eclipse is visually organized by perspectives. Those perspectives are groups of graphic elements (like, windows, menus, and toolbars) which are functionally related. The other technology, SystemC, is a library implemented in standard C++ which provides hardware-oriented construction using C++.

2. The development environment architecture

In Figure 1 is shown the global architecture of the development environment. There are three main programs: the visual environment, the simulator, and the assembly tool. The first software provides a graphic interface to the user. It is a stand-alone program, but it is dependent on the others to be fully operational. The user does not need to interact directly with the simulator and the assembly tool. The visual environment starts them, when they are necessary, and specifies what they should do.

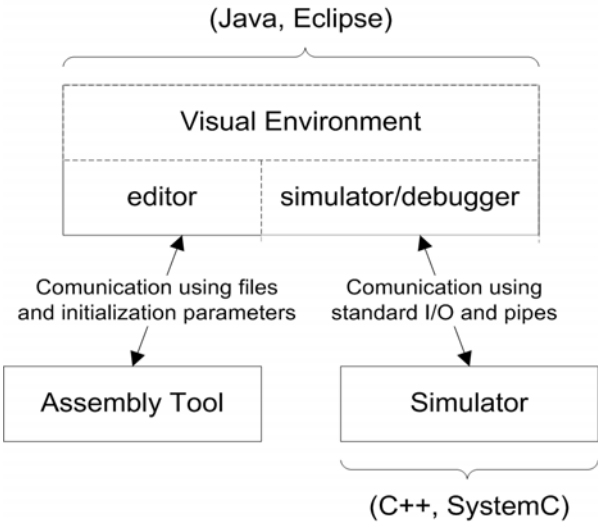

Figure 1. The Development Environment Architecture

The communication with the assembly tool is achieved by initialization parameters and files. The commands needed to accomplish an interactive simulation must use a more complex data communication approach. In order to do so, the standard input/output streams in the simulator were defined as channels to receive the commands and reply them; and pipes were used by the visual environment to send the commands and read their answers.

That solution is more flexible and easier to implement and debug than other inter process communications as shared memory or sockets. Other desired feature made

possible is that the simulator can be a stand-alone program.

2.1. The simulator

The simulator is a translation from VHDL description [4] to SystemC. It provides a register transfer level model of the SoC for test, debug, and validation of pieces of software and hardware's functionalities.

The simulator's interface defines a set of commands and answers which controls the simulation, informs about simulation's events, and synchronizes the simulator with the virtual environment. Two desired features of that set are: they hide most of the SoCs internal details, and make it possible to replace the simulator without affecting the visual environment. Therefore, it is possible to have SoCs different versions working with the same visual environment.

Figure 2 shows a partial sequence diagram of communication between the visual environment and the simulator generate by the user's interaction. The first arrow to the simulator represents its launch. The next two are commands to it. The simulation's answers are the dashed arrow. The "ok" message is for synchronization, and means "I have done what you asked. Now, you can send me another command".

Whether, during a period of clock, a register or memory word is changed or an instruction is started or finished, the simulator will send messages to inform those events.

The command on the bottom Figure 2, "serialize", is used to get all necessary information to save the state of a specific instant. To do the complementary operation, "deserialize" is executed with the data that had been sent. Those commands make possible to save how a simulation is at a moment and come back when it is wanted.

2.2. The visual environment

The visual environment is developed as plug-in to the Eclipse Platform in Java. It provides a graphic interface for the user which is divided in two perspectives: the editor, the simulator/debugger.

The editor perspective is like a usual code editor integrated with a compiler. The simulator/debugger perspective provides an interface to the simulator that automated all functionalities. The user can step through instruction by instruction of his program or run the simulation for undefined number of instruction. In this mode it is possible to command a stop or set break points where the simulation will be stopped automatically.

A textbox with program's code shows the line which is being executed. Statistic information related to the simulation as time consumed; the number of executed instruction; the amount of usage memory; are available to the user. A table with the value of the whole memory's words, and other with the registers is also available. Other feature provided is the possibility of save whole SoCs internal state at one specific instant and restore it later.

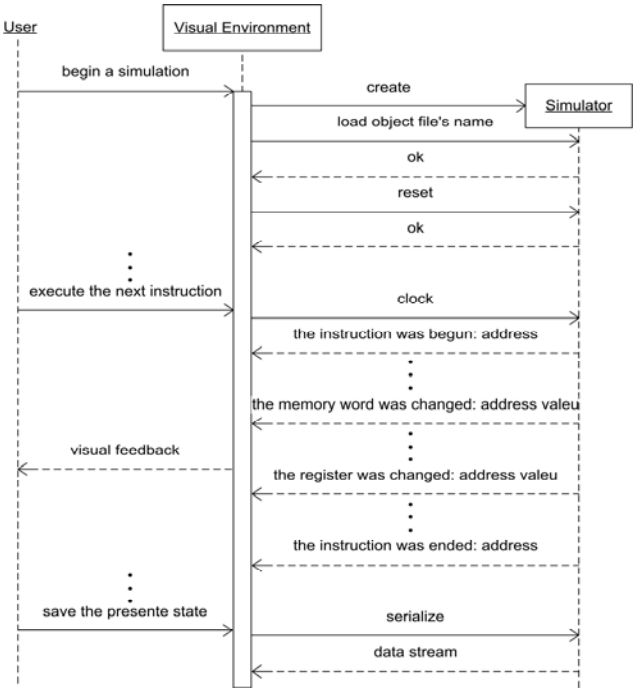

Figure 2. A sequence diagram of commands and the resulted events

3. Conclusions

The simulator was successfully developed for a simplified SoC version, and is now under testing. The SoC test plan [4] was used to validate it. The visual environment has been fully described and the object-oriented analysis has been made. In the simulator's next version, the modules RF transceiver, digital interface, and A/D will be implemented.

4. Acknowledgments

The authors would like to acknowledge CAPES, CNPq and PADCT/Millenium Institute (Brazilian government agencies) for financial support.

5. References

[1] J. D. Costa et all., "Projeto de Estruturas de um Processador RISC para Aplicação em um SoC para Controle de Irrigação", Proceedings X Iberchip Workshop, Iberchip, Cartagena de Indias, Colombia, 2004 (in Portuguese).

[2] http://www.eclipse.org

[3] http://www.systemc.org.

[4] J. D. Costa, "Implementação de um Processador RISC 16-bits CMOS num Sistema em Chip", Master Thesis in Electrical Engineering, University of Brasília, Brazil, 2004 (in Portuguese).