Configurable Virtual Platform Environment Using SID Simulator and Eclipse*

Hadipurnawan Satria, Baatarbileg Altangerel, Jin Baek Kwon, and Jeongbae Lee

Department of Computer Science, Sun Moon University
Kalsan 100, Tangjeong, Asan, Chungnam, South Korea
hadi198@yahoo.com, a_bbileg@yahoo.com, jbkwon@sunmoon.ac.kr, jblee@sunmoon.ac.kr

Abstract. For designing and testing embedded software, simulation tools have been used to keep pace with the rapid development of customized hardware parts. SID is a framework for building computer system simulations and SID is made for debugging, testing and verifying embedded software. Though, it is difficult for developers to use SID for their work. In this work, we developed an integrated virtual platform environment based SID simulation framework for a simulator engine and Eclipse for development platform. The proposed system avoids users to manually write the configuration file, and aids loading and connecting components on the fly. We also developed an image file builder and an automation tool for running SID simulation with GDB debugger. Furthermore, users can also monitor/probe the status of all the active components in the target virtual platform during the simulation

Keywords: embedded software, development tools, virtual platform, full system simulator.

1 Introduction

Nowadays, embedded system products are found everywhere and are becoming more and more advanced. To keep up with the market competition, the products must be more sophisticated and feature rich, but manufacturers also require a shorter time to market. Nevertheless, the improvements to the design and testing tools have not kept pace with the rapid development of customized hardware parts. Simulation tools have been designed to help close the gap and meet the needs of embedded software developers. The simulation of the target environment or virtual platform enables embedded software developers to analyze and test their software, even in the absence of the physical hardware.

From the perspective of full system simulation and emulation, there are number of software systems that support a wide rage of devices[1][3][4][5][6][7][8][9]. However, SID is specifically made for debugging, testing and verifying embedded

* This research was supported by Ministry of Information and Communication, Korea, under the ITRC(IT Research Center) support program supervised by Institute of Information Technology Assessment.

R. Obermaisser et al. (Eds.): SEUS 2007, LNCS 4761, pp. 394–398, 2007.

software. Since our work focuses on an environment for building embedded hardware simulators with simulated components, SID is a better fit for our work.

Although SID is a well-designed framework and environment of building a new virtual platform, it is difficult for embedded software developers to use SID for their work. To use it for an actual development, the users should configure a target platform by editing a configuration file with a text editor, run the virtual platform by typing a command in console, write and build a binary image to be run on the target, run a debugger such as GDB, and load and run the binary image to the active virtual platform. Since each step is manually done with independent tools, the users of today who get used to user-friendly user interface should endure considerable inconvenience. Therefore, it is desirable to integrate the tools and automate the usage procedure with graphical user interfaces.

In this work, we developed an integrated virtual platform environment based SID simulation framework for a simulator engine and Eclipse[2] for development platform. Eclipse is an open source development environment having extensible architecture, where the environment can be extended by adding plug-ins. Thus, our system is developed as Eclipse plug-ins. The proposed system avoids users to manually write the configuration file, and aids loading and connecting components on the fly. We also developed an image file builder and an automation tool for running SID simulation with GDB debugger. Furthermore, the users can also monitor/probe the status of all the active components in the target virtual platform during the simulation.

2 Background

2.1 SID Simulation Framework

In SID, a simulation is comprised of a collection of loosely coupled *components*. Simulated systems may range from a CPU's instruction set to a large multi-processor embedded system. SID defines a small component interface which serves to tightly encapsulate them. Components may be written in C++, C, Tcl or any other language to which the API is bound. C++ is the main language used, and for additional language a special component, a *bridge*, is required. During simulation start-up, components are instantiated, interconnected, and configured as necessary to represent some specific system. SID is suitable for consideration as an integration platform for other simulators by interconnecting models from different simulators, as has been done with Bochs[1], and also with a live Verilog system.

The components and their relationships are described in a configuration file, therefore required to run a simulation. The configuration file describes all components to be loaded and which component connected to which component. The SID simulator engine loads the components and connects them according to the configuration file. The SID framework provides a few ways for components to communicate with each other, i.e. pin, bus, attribute and relation mechanisms. All of these communication mechanisms may also be set up in the configuration file. Although there is an auto-configuration file builder for some typical target boards, in general users have to manage the configuration file content for new target platform by editing the file.

2.2 Eclipse Platform

The Eclipse[3] platform is designed for building integrated development environments (IDEs) as an open source project. One of the key benefits of the Eclipse Platform is realized by its use as an integration point. Building a tool or application on top of Eclipse Platform enables the tool or application to integrate with other tools and applications also written using the Eclipse Platform. The Eclipse Platform is turned in a Java IDE by adding Java development components (e.g. the JDT[5]) and it is turned into a C/C++ IDE by adding C/C++ development components (e.g. the CDT[2]). It becomes both a Java and C/C++ development environment by adding both sets of components. Eclipse Platform integrates the individual tools into a single product providing a rich and consistent experience for its users [4].

Eclipse platform has a plug-in architecture, where a plug-in is the smallest unit that can be developed and delivered separately. Plug-ins are coded in Java. Each plug-in has a plug-in manifest declaring its interconnections to other plug-ins. The interconnection model is simple: a plug-in declares any number of named extension points, and any number of extensions to one or more extension points in other plug-ins.

3 Architecture

In this section, we describe the overall architecture of our system, which the modules implementing the functions mentioned above are developed as plug-ins over Eclipse platform. Fig. 1 shows the architecture.

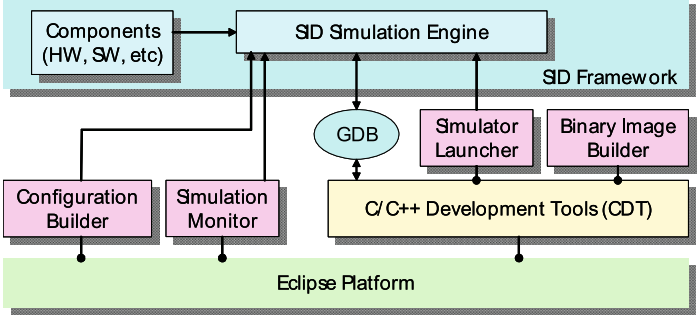

Fig. 1. Overall Architecture

As shown the figure, the architecture is based on SID framework and Eclipse platform, and includes CDT that is a set of plug-ins for IDE for C/C++. CDT consists of an editor, building tools such as compiler and linker, a debugger front-end connecting to GDB, etc. We extended it by adding two plug-ins, the binary image builder and the simulator launcher, to cooperate with SID. And, the configuration builder and the simulation monitor are plugged directly in Eclipse platform.

CDT does not support a cross-development environment. An embedded software development environment should provide the cross-development environment, where

an image file built in a host is run on the target system. That is why the *binary image builder* was developed. It provides the cross-development environment to CDT with GNU cross toolchains, e.g., arm-elf-gcc, arm-elf-as, etc. Therefore, users can build an image file to be run on a virtual target platform with the binary image builder.

The SID simulator requires a configuration file that describes the target platform. Basically, the file should be manually edited by a developer. In order to eliminate the troublesomeness, the *configuration builder* automatically generates a configuration template file for a target platform by checking the components to be used and inserting some values such as memory addresses on a GUI.

CDT has a debugger user interface interacting GDB. And, SID has a built-in component that performs the equivalent function of a GDB remote stub. Hence, CDT can load and debug an image on the virtual platform through GDB. The *simulator launcher* activates the virtual platform described by the configuration file selected when a debugging session begins.

SID also provides a built-in but experimental system monitor written in Tcl/Tk, to monitor a running simulation. The system monitor lists the components in the active virtual platform, showing specific component attributes such as pins, registers, etc. Since our system is based on the Eclipse framework, the system monitor should also be made to an Eclipse plug-in, which must be written in Java. However, SID cannot support components written in Java directly without a Java bridge component. Instead of developing the bridge component, we connected the simulation monitor plug-in and SID over a socket communication.

4 Implementation

The simulation monitor is implemented by interacting between the SID simulator and the Eclipse plug-in. Thus, it is implemented in two parts, one as SID component and another as an Eclipse plug-in.

The configuration builder is implemented as a new file wizard along with SID file types. Using the wizard, users can choose the target processor, e.g., ARM, and they can also select the components of the target platform SID supports many kinds of components, we currently only provide some of most important components on this wizard. The configuration builder generates a configuration template file according to the user's choice. Then the user can edit the template manually for a finer configuration.

The binary image builder provides the cross-development environment to CDT with GNU cross toolchains, e.g., arm-elf-gcc, arm-elf-as, etc. In CDT, the set of the tools and their settings to be used for build process is determined by selecting "Build Target." Thus, we add new build targets for ARM processor. By this way, the binary image file can be successfully built with the default tools and build settings. Users can also further modify the build settings as needed.

CDT has a debugger user interface interacting GDB. And, SID has a built-in component that performs the equivalent function of a GDB remote stub. The simulator launcher activates the virtual platform described by the configuration file selected when a debugging session begins. Eclipse has a general debug configuration window, where users can select different kinds of debug configuration template. After

they choose the proper template, they can configure the debugger based on that template. We implemented the "C/C++ Virtual" configuration template for GDB debugging session with the virtual platform. In this configuration, the users can select an SID configuration file describing a target machine. The simulator launcher activates the virtual platform on SID when the debugging starts, and also deactivates it when the debugging stops.

5 Conclusion

In this work, we developed an integrated virtual platform environment based SID simulation framework for a simulator engine and Eclipse for development platform. Our system is developed as Eclipse plug-ins. The proposed system avoids users to manually write the configuration file, and aids loading and connecting components on the fly. We also developed an image file builder and an automation tool for running SID simulation with GDB debugger. Furthermore, users can also monitor/probe the status of all the active components in the target virtual platform during the simulation.

References

1. The Bochs IA-32 Emulator Project, http://bochs.sourceforge.net
2. Eclipse C/C++ Development Tools (CDT), http://www.eclipse.org/cdt/
3. Eclipse Platform, http://www.eclipse.org
4. Eclipse Platform Technical Overview (2006), http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf
5. Eclipse Java Development Tools (JDT), http://www.eclipse.org/jdt/
6. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A Full System Simulation Platform. IEEE Computer 35(2), 50–58 (2002)
7. PearPC: PowerPC Architecture Emulator, http://pearpc.sourceforge.net
8. QEMU: A Generic and Open Source Processor Emulator, http://fabrice.bellard.free.fr/qemu
9. System, S.I.D.: Simulator, http://sourceware.org/sid
10. SimOS: The Complete Machine Simulator, http://simos.stanford.edu
11. SkyEye: An Embedded Simulation System, http://www.skyeye.org
12. Witchel, E., Rosenblum, M.: Embra: Fast and Flexible Machine Simulation. ACM SIGMETRICS Performance Evaluation Review 24(1), 68–79 (1996)