An Interactive System Level Simulation Environment for Systems on Chip

Daniel Knorreck, Ludovic Apvrille, Renaud Pacalet
System-on-Chip Laboratory (LabSoC), Institut Telecom, Telecom ParisTech, LTCI CNRS
2229, Routes des Crêtes BP 193 F-06904 Sophia Antipolis, France
Email: {daniel.knorreck, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

Abstract— **This article presents an interactive simulation environment for high level models intended for Design Space Exploration of Systems-On-Chip. Our coarse grained simulation methodology which allows for efficient system level performance estimations was detailed in the scope of our last year's submission [4]. Our new achievement comprises an additional module which allows the designer to control the simulation in real time by performing step wise execution, saving and restoring simulation states as well as providing live feedback to a graphical interface. Finally, this article gives an idea of possible enhancements of the simulation environment in order to automatically assess several possible executions of tasks and thus enhance simulation coverage.**

I. Introduction

The increasing complexity of today's embedded systems requires an evaluation of benchmark data as early as possible in the design flow. At early design stages, low level models of the Systems-On-Chip are not available yet. Design Space Exploration aims at identifying the most suitable hardware / software platform complying to given constraints. This stage is accomplished based on high level models of the target system, on which fast simulations and static formal analysis can be performed in order to verify the satisfiability of both functional and non functional requirements.

In this context, we have previously introduced a UML-based environment, named DIPLODOCUS ([3], [5]). The strength of our approach relies on formal verification capabilities, and fast simulation techniques. DIPLODOCUS design approach is based on the following fundamental principles:

- Use of a high level language (UML).
- Clear separation between application and architectural matters.
- Data abstraction.
- Use of fast simulation and static formal analysis techniques, both at application and mapping levels.

Moreover, DIPLODOCUS includes the following 3-step methodology:

1) Applications are first modeled using tasks with communication capabilities.
2) Targeted hardware architectures are modeled independently from applications. A set of usual hardware components has been defined (e.g. CPUs, buses, etc.).
3) A mapping process defines how applications may be mapped onto a given architecture.

An open source modeling and validation framework called TTool ([1], [2]) is meant to guide the designer while performing the aforementioned steps. After having conceived application and architecture/mapping models, the designer may launch formal verification (based on LOTOS/UPPAAL) or simulation at the push of a button. Formal verifications may be conducted in order to verify the absence of deadlock situations and liveness properties. In order to obtain key figures characterizing the respective architecture, the graphical model is automatically converted to a C++ representation. The latter is compiled and run subsequently in the framework of the provided simulation engine.

This paper focuses on the interactive simulation which may be carried out after the designer has established an association between entities of the application model (like tasks and channels) and generic parametrizable hardware components (like CPUs, buses, memories, etc.). A task is described by means of usual commands (loops, tests, variable settings, etc.), of communication commands (reading/writing abstract data samples in channels, sending/receiving events and requests), and of computational cost commands (EXECx instructions). A mapping is described using a set of interconnected hardware nodes on which tasks, channels, events and requests are mapped.

In the scope of our last years' contribution [4], we proposed an efficient simulation environment implemented in pure C++ which leverages the characteristics of our high level application model. It renounces to a cycle-based analysis of application tasks by processing bunches of clock cycles (hereafter referred to as transactions) as a whole. Thus, a transaction stands for a portion of a command which belongs to a specific task of the application model. Transactions are initially defined according to those commands but inter task synchronization or component parametrization might induce truncations of transactions. Thus, modeling granularity as well as the amount of inter task communications solely impact the simulation performance regardless of the specific amount of clock cycles to be simulated.

II. Introducing Interactivity

The above mentioned simulation environment has been enhanced and it henceforth allows for an interactive exploration of the application based on a particular architecture. A TCP connection provides the simulator with commands intended for directing the simulation. For example, the following simulation commands have already been implemented:

- Different flavors of run commands: a given amount of transactions, commands or time units is simulated...
- ...likewise simulation may be interrupted when a given element processes a transaction (a CPU, a Bus, a Memory, etc.).
- Reset the simulation.
- Save and restore the simulation state, especially useful when several branches of control flow are to be looked into.
- Simulation traces can be output in several formats (text, VCD, HTML).
- Breakpoint related commands like setting and deleting breakpoints.
- Commands to obtain information about the progress of the simulation. These are used to animate UML diagrams within TTool.

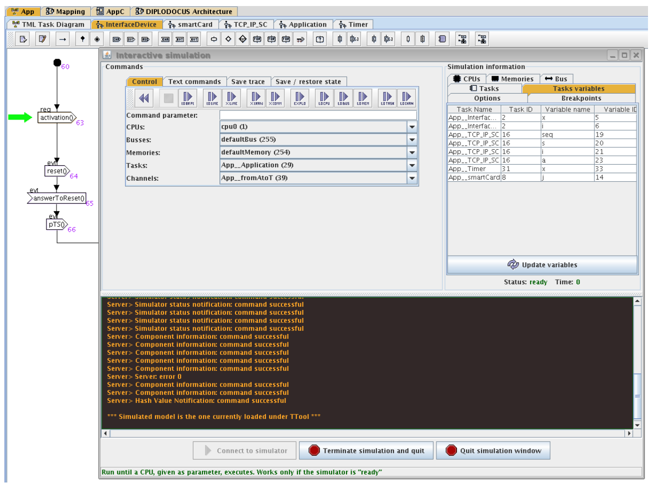
TTool encompasses a graphical interface to direct the simulation (Figure 1) and thus unburdens the user from familiarizing with a low-level simulation language. The feedback from the simulation engine is exploited by the graphical user interface and used to animate UML application diagrams. For instance, the current command of a task is highlighted for following simulation progress on each task. As a simple example, let us consider an algorithm having two main branches which significantly differ in terms of execution time and resource usage in general. For the performance evaluation of a specific architecture, it would be crucial to try out both alternatives. Hence, the coverage of the simulation should be enhanced. As a first step, the designer could benefit from the various conditional run commands so as to get a more intuitive view of the behavior of the application and the interaction of hardware components. The next step could be to reset the simulation and to set a breakpoint on the branch command which is crucial for the continuation of the simulation. The simulation will stop at the previously defined choice command therefore allowing the user to specify which branch he means to explore. In combination with the feature of capturing simulation states, complex scenarios can be evaluated and meaningful traces be recorded. In our example, the user would certainly save the simulation state when reaching the choice command so that it can be restored to study other alternative executions.

III. Conclusions and Future Directions

In conclusion it can be said that the contribution of this paper is on the one hand the extension of our simulation environment with a module providing an interactive control of the simulation procedure. On the other hand, the new simulation features have been tightly coupled to our integrated development environment TTool so that simulation progress is directly visualized within the UML diagrams representing the application model. Thus, a powerful toolbox is provided to the designer which is helpful when performing Design Space Exploration. It may alleviate considerably the process of

- Debugging applications
- Accessing intermediate simulation results
- Returning to previous system states
- Enhancing the coverage of the simulation by exploring several control flow branches.

Hence an intuitive insight into complex interrelationships of the system can be gained with ease.



Command parameter:
CPUs: cpu0 (1)
Busses: defaultBus (255)
Memories: defaultMemory (254)
Tasks: App__Application (29)
Channels: App__fromAtoT (39)

Task Name	Task ID	Variable name	Variable ID
App__Interfac...	2	x	5
App__Interfac...	2	i	6
App__TCP_IP_SC	16	seq	19
App__TCP_IP_SC	16	s	20
App__TCP_IP_SC	16	i	21
App__TCP_IP_SC	16	a	23
App__Timer	31	x	33
App__smartCard	8	j	14

Status: ready Time: 0

```
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Server: error 0
Server> Component information: command successful
Server> Component information: command successful
Server> Hash Value Notification: command successful

*** Simulated model is the one currently loaded under TTool ***
```

Run until a CPU, given as parameter, executes. Works only if the simulator is "ready"

Fig. 1. Screenshot of the Simulation Interface

In addition to technical improvements of the simulator, future work will include the automatic exploration of several alternative executions in order to enhance the simulation coverage. Furthermore, advanced simulation capabilities will be subject to further research activities. For example, one could think of specifying functional requirements which are checked during simulation (for example: if a client requests the bus, access is always granted within 10ms). The exploration of some branches could be privileged or abandoned based on certain criteria (CPU usage, resource contention, etc). When accounting for different executions, recurring system states should be tracked so as to be able to merge the control flow again.

References

[1] TTool, the Turtle Toolkit: http://labsoc.comelec.enst.fr/turtle.

[2] L. Apvrille. TTool for DIPLODOCUS: An Environment for Design Space Exploration. In *Proceedings of the 8th Annual International Conference on New Technologies of Distributed Systems (NOTERE'2008)*, Lyon, France, June 2008.

[3] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A UML-based environment for system design space exploration. *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pages 1272–1275, Dec. 2006.

[4] R. Pacalet D. Knorreck, L. Apvrille. Fast simulation techniques for design space exploration. *SAFA08*, December 2008.

[5] M. Waseem, L. Apvrille, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. Abstract application modeling for system design space exploration. *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 331–337, 0-0 2006.