Demonstration of 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 Crétes BP 193 F-06904 Sophia Antipolis, France
Email: {daniel.knorreck, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

Abstract—In the scope of the demonstration, we present 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 [10]. Our new achievement comprises an additional module which enables 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. UML models representing the application are animated and simulated based on the semantics of a separate architecture model. Finally, this description 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. DEMONSTRATION DESCRIPTION

A. 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 ([5], [20]). 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 ([2], [4]) aims at guiding the user through the aforementioned design flow. 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 which allow for the detection of deadlock situations and the analysis of 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.

The demonstration puts emphasis 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 in terms of usual control commands (loops, conditions, 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 comprises a set of interconnected hardware nodes on which tasks, channels, events and requests are mapped.

In the scope of our previous contribution [10], 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.
B. Interactive Simulation Environment

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 she/he intends 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.

C. Related work

Some of current state of the art UML modeling tools ([18], [17], [15], [7] amongst others) are able to simulate UML models, mostly based on class diagrams, composite structure diagrams and state machines. Simulations can only be performed based on purely functional models in an untimed fashion. Our previously described interactive simulator however also accounts for architecture semantics like arbitration of shared resources, speed or data throughput of devices, etc.

Related work in the field of system level modeling and simulation often suffers from one of the following problems: Off the shelves solutions like [3] and [1] mostly do not permit an orthogonalization of functionality and architecture. Detailed RTL models of HW components and the final software code must be at hand to perform co-simulation. Thus, only little abstraction may be applied to for instance inter component communication (SystemC TLM [16], etc). Some academic approaches enable the design of distinct models for architecture and application ([14], [19], [11]). In this case, the level of abstraction is often not pushed high enough in order to explore a representative subset of the HW/SW design space in a reasonable amount of time. Some application models do not exhibit data/functional abstractions or application models are not capable of reflecting indeterminism and data dependencies ([12], [13], [21]). Sometimes, the simulation strategy does not leverage abstractions and models have to be refined before being executed ([6]). Purely analytical approaches ([9], [8]) however require tasks models to be simplistic, merely reflecting key figures like worst case/best case execution time, etc. As opposed to our approach, control flow within tasks cannot be considered at all. For that reason it may be tedious if not impossible to model tasks exhibiting data dependent or irregular behavior.

D. Conclusions and future directions

In conclusion it can be said that we extended our simulation environment with a module providing an interactive control of the simulation procedure. In contrast to state of the art UML model simulators

- simulation semantics is tailored to the needs of Design Space Exploration of Systems-on-Chip
- and the user is enabled to closely interact with the simulator and to guide simulation runs.

The new 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 possible executions
- Pruning the design space by guiding the simulation.

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

Trading off accuracy against model complexity of hardware components will remain subject to our research. For
example, instruction cache-misses and data cache-misses have been accounted for by static probabilities so far. As algorithmic details are represented by symbolic instructions, the real code of the application is not available thus making state of the art cache models unsuited. Furthermore, the accuracy of bus and memory models have to be validated against real embedded systems. A fair comparison with a real implementation will reveal whether a set of parameters can be found to limit the inaccuracy to a reasonable percentage. To simplify the modeling of systems making extensive use of DMA engines, a respective UML stereotype could be introduced. This way, the designer would not have to model DMA transfers explicitly using a dedicated execution unit. In addition to technical improvements of the simulator, future work will also include the automatic exploration of several alternative executions in order to enhance the simulation coverage. The exploration of some control flow branches could be privileged or abandoned based on certain criteria (CPU usage, resource contention, etc.). When taking into account different executions, recurring system states should be tracked so as to be able to merge similar simulation runs.

REFERENCES