Fast Simulation Techniques for Design Space Exploration

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@eurecom.fr, ludovic.apvrille@telecom-paristech.fr, renaud.pacalet@telecom-paristech.fr

***Abstract*— In this paper, we present our current work on a UML based environment providing efficient means for system level design space exploration. First of all, a brief introduction to the concepts of DIPLODOCUS gives a general overview of our methodology. A cycle-based simulation strategy for the application modeled in terms of abstract tasks was introduced previously. This paper emphasizes a new approach which is currently subject of our research activity. A more coarse grained simulation of the modeled application can significantly reduce the simulation time. The basic idea is to merge several clock cycles and to process them as a whole whenever possible. Finally, this article gives an idea of possible enhancements of the simulation environment and longer term objectives.**

I. Extended Abstract

System level design space exploration in a System-on-chip (SoC) design cycle is an issue of great concern in today's rapidly growing and heavily constrained design process. The increasing complexity of SoC requires a complete re-examination of design and validation methods prior to final implementation. We believe that the solution to this problem lies in developing an abstract model of the system intended for design, on which fast simulations and static formal analysis could be performed in order to test the satisfiability of both functional and non functional requirements.

In this context, we have previously introduced a UML-based environment, named DIPLODOCUS. 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.

This paper focuses on the post-mapping simulation of tasks mapped onto a given hardware architecture. A task is described by means of usual operators (loops, tests, variable settings, etc.), of communication operators (reading/writing abstract data samples in channels, sending/receiving events and requests), and of computational cost operators (EXECx instructions). A mapping is described using a set of interconnected hardware nodes (CPUs, buses, hardware accelerators, etc.) on which tasks, channels, events and requests are mapped.

An important issue to solve, at post-mapping simulation step, is the scheduling of operations performed on hardware nodes, and more particularly on CPUs and on buses. A first simulation environment, based on SystemC, was developed previously and is detailed in [BHA07I]. In that simulator, a SystemC process is assigned to each hardware node, and a cycle-based simulation approach is used. Unfortunately, scheduling of those processes (process switching time, cycle-based approach) implies low performance, even if tasks are described with high-level instructions. A second approach described in that paper is a transaction-based simulator, using only one main process. Transactions are defined according to tasks' commands (i.e. tasks instructions). Those transactions are meant to last several cycles, but in case they last only one cycle, then, the simulator automatically falls back to a cycle-based behavior. Thus, the modeling granularity directly impacts the simulation performance.

More precisely, a transaction refers to a portion of a command. Thus, the READ task command allows for an inter task communication by reading a given number of samples from an abstract channel. For example let us consider the expression READ 3. This command could be broken down into two transactions, the first one reading one sample and the second one reading the two remaining samples. The basic idea is to try at first to schedule a transaction having the same length as the command (3 in our example). If the attempt fails, the transaction is subsequently split into smaller parts which can be scheduled. We first suppose that the command can be executed as a whole. Then, that command is cut into sub-transactions if some other transactions interfere with the first one. That way the simulation overhead decreases if the amount of inter task dependency is low. The worst case of that

speculative approach is when scheduling policies of CPUs and buses are based on a strict alternation between tasks: in that case, the simulation environment gets back to a cycle based simulation because every transaction is automatically reduced to the length of 1. This scenario could be avoided with the aid of bus scheduling policies relying on atomic burst transfers which cannot be interrupted. As an example, let us consider the scenario depicted below: there are two CPUs, referred to as CPU1 and CPU2. A CPU merges both an abstract hardware component and an abstract real time operating system.
On figure 1, both CPUs (CPU1 and CPU2) have already executed a transaction of a task T_{11} and T_{21} respectively. The transaction on CPU1 finished at t_{SI}, the one on CPU2 finished at t_{SII}. The schedulers of both CPUs subsequently propose the next transaction to execute. The simulation algorithm selects the transaction which will terminate first because its execution could have an impact on other tasks (making them runnable for instance). This decision is in line with traditional mechanisms used for discrete event simulation. As we have to be sure that during the execution of the selected transaction no events will occur, special care has to be taken when considering communication-related commands (read to channel, write to channel, wait on event, notify event, ...). The length of these transactions are therefore calculated based on the number of samples to read, the number of samples to write, the content and the size of the channel as well as the size of atomic burst transfers on the bus.
After T_{12} has been scheduled, t_{SI} is set to the end time of T_{12}, referred to as t_{SInew}. If the completed transaction T_{12} causes a task to become runnable on CPU2, T_{22} is truncated at t_{SInew} and the remaining transaction is scheduled on CPU2. As a channel always links only two tasks, revealing dependencies becomes trivial. If a portion of T_{22} has been added to the schedule, t_{SII} is changed accordingly ($t_{SII} = t_{SInew}$). After this, all schedulers of CPUs which have executed a transaction are invoked. The algorithm has now reached its initial state where all schedulers have selected a potential next transaction. Again, the transaction which finishes first is scheduled...

The main components which are involved in a scheduling round are the following: tasks, CPUs buses and the main scheduler. The sequence of the entities of the aforementioned list also reflects their hierarchy during the scheduling process: tasks are settled at the top layer and the main scheduler constitutes the lowermost layer. Based on the knowledge of their internal behavior, tasks are able to determine the next transaction which has to be executed within their scope. This proposal is forwarded to the scheduler of the respective CPU on which the task was mapped. The latter scheduler in turn selects one task proposal and transfers it to the dedicated bus in case the current transaction demands bus access. During this stage, delays due to bus contention are taken into account. As several CPUs may be connected to the same bus, a scheduling decision has to be taken once again. Finally, the main scheduler is in charge of assuring the causality of the simulation which is achieved as stated previously.

In conclusion it can be said that the two main contributions of this paper are on the one hand the extension of the traditional discrete event simulation with a hierarchical scheduling procedure comprising execution nodes and communication nodes. On the other hand, simulation granularity is automatically adapted to the requirements of the application thanks to the transaction based approach. When experimenting with a model of an MPEG decoder, we experienced performance gains in terms of execution time up to factor 30 as compared to a cycle based SystemC simulator. It should be reemphasized that the gain depends on the application model.

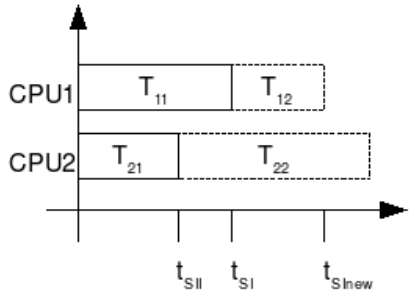

Fig. 1. Scheduling scenario

The status quo of the simulator leaves room for improvements. The bus model for example could be enhanced in a future version of the simulator. For the time being, the bus determines the length of transactions it has to deal with (READ and WRITE transactions). The underlying assumption is that for fast CPUs, the bus represents the bottleneck and the CPU is slowed down due to I/O operations. However for CPUs working at a low frequency as compared to the bus, the latter assumption does not hold. In this case, the CPU is not able to keep the bus busy and thus represents the bottleneck.

In addition technical improvements of the simulator, advanced simulation capabilities could be integrated in the future. For example, one could think of specifying functional requirements which are checked during simulation (for example: if a client requests the bus, access is granted within 10ms). In this case, it would also be very interesting to explore several branches of control flow in order to enhance the coverage of a simulation. Several tasks could examine different branches or a single task could process them consecutively. In the latter case, a mechanism has to be conceived which allows to return to a system state in the past. To achieve this, it would be sufficient to store member variables of classes which characterize the system state: filling level of channels, end time of the last scheduled transaction on CPUs, progress and current transactions of

commands,... The object oriented model could be based on the memento pattern.

REFERENCES

[HUS07] Sorin A. Huss, *Advances in Design and Specification Languages for Embedded Systems*, Springer, 2007

[APV06] Ludovic Apvrille, Renaud Pacalet, Axelle Apvrille, Pierre de Saqui-Sannes, *Un environnement de conception de systèmes distribués basé sur UML* System-On-Chip laboratory, GET/ENST, 2006

[MUH] Waseem Muhammad, Ludovic Apvrille, Rabéa Ameur-Boulifa, Sophie Coudert, Renaud Pacalet, *Abstract Application Modeling for System Design Space Exploration* System-On-Chip laboratory, GET/ENST

[MUH06] Waseem Muhammad, Ludovic Apvrille, Rabéa Ameur-Boulifa, Sophie Coudert,Renaud Pacalet, *A UML-based Environment for System Design Space Exploration*
System-On-Chip laboratory, GET/ENST, 2006

[BHA07I] Muhammad Khurram Bhatti, Ludovic Apvrille, *Modeling and Simulation of SoC Hardware Architecture for Design Space Exploration* System-On-Chip laboratory, GET/ENST, 2007

[BHA07II] Muhammad Khurram Bhatti, Ludovic Apvrille, Renaud Pacalet, *Designing SoC with data abstraction in mind* System-On-Chip laboratory, GET/ENST, 2007

[BLA04] David C. Black, *SystemC: From the ground up* Kluwer Academic Publishers, 2004

[GROT02] Thorsten Grotker, Stan Liao,Grant Martin, *System Design with SystemC* Kluwer Academic Publishers, 2002

[GHE05] Frank Ghenassia, *Transaction Level Modeling with SystemC* Springer, 2005