A New Paradigm and Associated Tools for TLM/T Modeling of MPSoCs

Emmanuel Viaud
Laboratoire LIP6
Université Pierre & Marie Curie
Email: emmanuel.viaud@lip6.fr

François Pêcheux
Laboratoire LIP6
Université Pierre & Marie Curie
Email: francois.pecheux@lip6.fr

***Abstract*— The paper presents a way to speed-up simulation of complex MPSoCs at the TLM/T (Transaction Level Modeling with Time) level and a way to switch from abstraction level "at run time". The hardware part of the platform is described in standard SystemC. Rather than using statistical models for adding timing, to obtain an accurate view of the platform dynamic contention is taken into account. This allows to reach a speed-up of up to 50 versus a corresponding BCA simulation with a low timing error. The state saving method gives the ability to save the state of a platform at the TLM/T level and move it to the same platform described at the BCA level. This allows fast and accurate debugging of the embedded software.**

I. Introduction

Reaching high-simulation speed is of crucial importance with the advent of more and more complex SoCs. Even if they give the most valuable informations, low abstraction levels are unmanageable as early simulation levels as they are slow and they need a very precise description of the architecture of the SoC, which is not available at the beginning of the design phase. Moreover, the degree of accuracy provided may be overkill for the needs.

Higher abstraction levels exist that provide different trade-off between simulation speed, accuracy of the results and the quantity of information one can get. From all the levels available, a fairly recent one coined Transaction Level Modeling (TLM) seems to attract much interest from the community. Its main characteristic is that it "abstracts the pin-level communication in the physical model to the level of media access or individual protocol word/frame transactions"[1]. It means that the computational part and the communication part of a component are separated; whereas the computational part can be more or less precise, the communication does not represent a real protocol but only an abstracted view of it, represented in term of read and write accesses of packet composed of multiple words. Being an abstract view of the SoC, the TLM level can be used early in the design process, thus providing a way for software developers to quickly test their program on a representative model of the platform and for hardware architects to quickly explore the design space.

TLM allows a developer to answer questions such as "is my SoC functional ?" but, being non-timed, some questions pertaining to the performance or some real-time constraints remain unsolved. Giving reliable answers to those problems implies to model many more aspects of the architecture, most notably the interactions between the different IPs found in the SoC (dynamic effects of the contention in the interconnect, ...). Thus, adding timing informations to TLM is necessary in order to increase its usability. In the rest of this paper, this updated level is called TLM/T (TLM with Time). Another term used in the literature is PVT (Programmer's View with Time) in opposition to PV (Programmer's View, for TLM).

Usually, adding more precision to an abstraction level is often synonymous of decreasing performance of the simulation. The aim of this paper is to show that, on the contrary, it is possible to get meaningful timing informations from a simulation with only a small loss of performance, and that the data we get are meaningful.

The paper is composed of six sections. After a brief introduction, section II presents the relevant work in the TLM and TLM/T modeling domains. section III briefly explains the methodology, section IV shows a way to switch between TLM/T level and BCA level and section V presents the experiments we led. Finally, section VI comments the previous results and gives some research perspectives.

II. Previous work

Even if the term is often used, getting a precise and common definition of TLM is pretty hard. More than a unique level of abstraction, it rather represents a set of levels all linked by their approach of communication abstraction. Cai and Gajski in [2] or Donlin in [3] proposed a taxonomy, trying to clear the ground for some clean foundations. On that base, some standards have been proposed for a concrete definition of TLM. The most significant in our context is the one made by the Open SystemC Initiative in [4]. It presents an API and a set of channel that aim at easing the development of platforms at the TLM level. Unfortunately, the proposal does not address the problem of timing at all. Other approaches for TLM have been proposed. A very efficient one is described in [5]. Even if limited to untimed simulation, some concepts (most notably the *passive components*) are shared with our proposed methodology.

Nevertheless, approaches managing time exist. An example is the CCATB (Cycle Count Accurate at Transaction Boundaries) level presented in [6]. Not formally a TLM approach it still proposes to abstract the communication and, as in our

approach, to only keep pertinent timing information about the begin and end time of a transaction. But, in that approach, we are still close to the BCA level and, thus, the expected speed-up over BCA level simulations is rather limited (about a factor 2).

III. The methodology

Classically, time issues are handled through the `wait` construct of SystemC and the use of `SC_THREAD`s. But using that techniques has proved to notably slow the simulations down [7]. But, as using `SC_THREAD`s eases the development of the simulation models, keeping them in a methodology is definitely a plus. As the main loss of speed comes from context switching between threads, one must find a way to reduce their numbers. Our key idea is to reduce the number of logical events that are at their origin. So, instead of waking up the different components every cycle even if they have nothing to do, it is far better to find a way to only transmit the smallest quantity of information, and only when needed. These needs correspond to the ones addressed by the Parallel Discrete Event Simulation (PDES) domain.

The approach presented herein is a direct application of the classical PDES Chandy-Misra-Bryant algorithm ([8] and [9]) to a SystemC simulation. Each component is seen as an independent thread with its own clock, called local clock. There is therefore no global clock in the system. Instead, every time a packet representing an elementary data transfer between an initiator (producer) and a target (consumer) is sent, it is piggy-backed with the local time of the producer, in order to synchronize the component with the remaining parts of the platform. To preserve causality, a component is allowed to increase its local time only when it has sufficient information on all the components that feed it with data, i.e. all its input channels contain data [10]).

As an example, we will now describe the structure of a master component which plays the role of a processor with its associated cache (instruction and data). It should be noted that the proposed architecture is only an example, i.e. other descriptions can be used (use of `SC_METHOD` rather than `SC_THREAD` would lead to a faster simulation at the expense of the easiness of writing the component model, but could be managed with tools such as the one presented in [11]). In the proposed structure, the main loop of the component is a `SC_THREAD` that can be described by the following pseudo-SystemC code:

```
while(1) {
  if (icache_hit(PC)) {
    ir = icache[PC]
  } else {
    send_request(PC, icache_word_per_line)
    wait(response)
  }
  switch(ir) {
    case ADD:
    ...
    break;

    case LOAD:
    if (dcache_hit(effective_addr_reg)) {
      dest_reg = DCACHE[effective_addr_reg]
    } else {
      send_request(effective_addr_reg,
                   dcache_word_per_line)
      wait(response)
    }
    break;
    ...
  }
}
```

This code shows that communication with the rest of the platform is done only when necessary, through the send_request function call. As long as the component finds directly its local instruction and data (instruction and data available in the cache), it works autonomously without giving hand to the scheduler, thus decreasing the number of necessary context switches. When communicating with the outside, the component goes to sleep to give the possibility to the other components to receive the packet and send their answer. Once the response packet received, the component posts an event to itself to wake up the main thread and to continue its task.

IV. Switching between different abstraction levels

TLM/T is used for architectural exploration and the validation of the embedded software. Even if it provides a useful way to develop embedded code early, some tasks (like finding some race conditions in the embedded code) are impossible to fulfill due to the relative inaccuracy of the simulations and need to be done at a lower level of abstraction (BCA as an example). But a lower level of abstraction means a slow simulation and thus, simulation time may become unacceptable if errors occur late in the simulation. Hence, we propose a method that allows to shift from TLM/T abstraction level to BCA abstraction level. The idea is to take advantage of both levels, the speed of TLM/T and the accuracy of BCA. To do this, the TLM/T simulation is stopped just before the problematic point and the BCA simulation is restarted.

Two problems appear. The first is linked to the methodology. As each component has its own clock, stopping the simulation at a common given time is impossible. Some algorithms (like [12]) solve that problem by taking a “snapshot” of the state system that is guaranteed to be at least “coherent”, which means that no message will be forgotten or counted multiple times. In our case though, we can simplify the problem a bit. As a matter of fact, whereas in the general case one should take great care not to forget some messages still in the communication channels, in our methodology the channels are either function calls (and thus, no message can stay in it as the SystemC scheduler is not preemptive) or logical modules (in which case, saving state is simple). So, we can use a

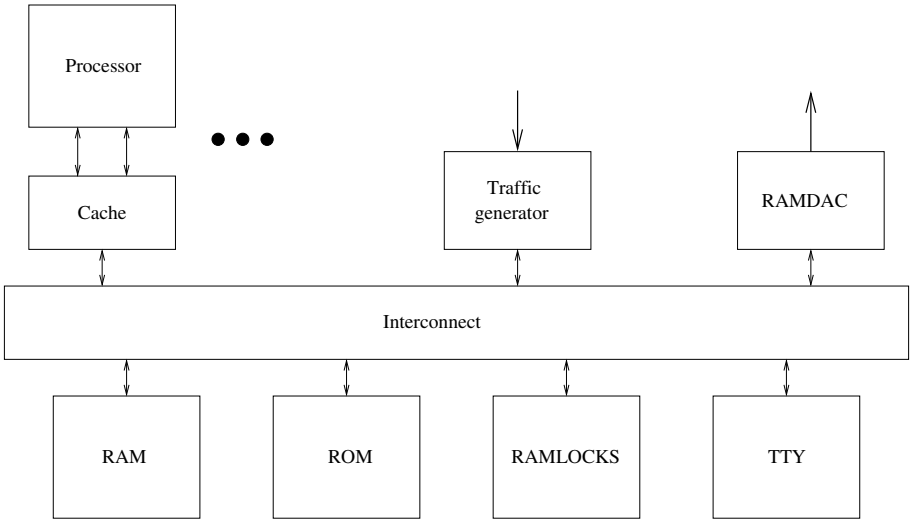

Fig. 1. Architecture of the MJPEG platform

degenerated version of the algorithm which only consists in saving the state of the different modules found in the simulated platform. The second, inherent to the TLM level, is that the components, being described in a more abstract way, do not fully represent the internal architecture of the IP. Thus, it is not possible to directly represent the state of the component at the BCA level when simulating it at the TLM/T level: a mechanism to convert from TLM/T state to BCA state is mandatory.

The solution to both problems is in fact linked. We decided to create a tool that would, given the TLM/T state of a component, generate the corresponding BCA state. Obviously, the informations pertaining to the elements not represented in the TLM/T version of the component cannot be obtained from the simulation and must be determined in a way or another. Rather than giving them randomly a value, we decided to limit the impact of this arbitrary decision by allowing to stop the TLM/T simulation only in some states which minimize the role of the undescribed parts of the architecture. For the interconnect, it means that it must be stopped when it is empty in order to be put in a clean state in the BCA simulation. For a cache with a write buffer, we must not have any pending transaction and the write-buffer must be empty. Thanks to that constraint, we can put their initial values in the different unknown registers and then launch one cycle of simulation to correctly set the outputs of the different components before continuing the normal BCA simulation.

Stopping the simulation with the previously given constraints is possible because the components are at different times. This advantage also has a drawback: how to choose the initial time of the BCA simulation ? At the moment, we put the simulation time at the lowest time of the master components. This obviously generates an error in the timing but, by limiting the advance of the clock of the components with the lookahead parameter [10], we can limit the resulting error.

V. Experimental results

All the following experiments were made on an Intel Pentium IV running at 2.66 GHz running Linux.

In order to test the approach, we modeled a MJPEG decoder platform (shown in figure 1). It contains a generic VCI/OCP compliant interconnect [13], a variable number of processors, two hardware coprocessors in charge of the I/O (one reads a MJPEG stream and sends it to the first task, the other one reads the decoded pixels and send them to a screen) and some other needed components (ROM, RAM, ...)

The platform is modeled first at the BCA level using SoCLib [14] components and then at the TLM/T level. The comparison between the two platforms is made on two points:

- the simulation speed
- the accuracy of the results

Here, the accuracy is measured by the difference of the time needed to decode an image in the BCA and TLM/T simulations (we suppose that the BCA is the correct decoding time). To test the efficiency of the approach, we modify the number of processors available in the platform.

TABLE I
Simulation speed and error for both platforms (rounded values)

	1 proc	2 procs	4 procs
BCA (cycle/s)	36725	26345	17923
TLM/T (cycle/s)	2012814	901412	445958
speed ratio ($\frac{TLM/T}{BCA}$)	55	34	25
error	2 %	4 %	3 %
cycles/δ-cycle	15	8	6

The results on Table I show that the methodology offers a speed-up between 20 and 50 over the corresponding BCA simulation, with an error of less than 5 %. One also see that our previously stated goal of diminishing the number of events provoking descheduling of threads is also reached as

one can see that for every delta-cycle, the equivalent global clock has increased of around 10 cycles, which partly explains the observed speed-up. The decrease in the efficiency of the methodology can be explained by the needed wait states in the interconnect to synchronize the different components.

Another comparison can be made concerning the state switching method. Rather than just comparing the simulation speed, which has already been done, we will rather concentrate on the error between a pure BCA simulation and a TLM/T-BCA one. The simulated platform is simpler that the previous one. It consists in 4 processors and caches, a ROM and a RAM, 4 TTYs to display text informations and a timer that will periodically interrupt each processor. The comparison will be made on the interrupt time displayed by each processor on its TTY.

TABLE II
SIMULATION SPEED AND ERROR FOR THE STATE SWITCHING METHODOLOGY

	200,000 cycles	2,000,000 cycles
$\frac{TLM/T}{BCA}$ speed-up ratio	4	5
timing error (cycles)	≈ 10	≈ 150

One can see (Table II) that switching state give a simulation speed-up of only 5 for that platform. That is explained by the fact that, as the processors are mostly idle, waiting for events coming for each processor is a very long phase, and thus, many delta-cycles are spent. In that case, the cycle/delta-cycle ratio is only a bit more than one. The increase in the speed-up is explained by the fact that the state saving phase is included in the simulation time. As that time is a constant one for a given platform, longer simulations implies that the cost of saving is better amortized. Even if the gain is low, one can also see that the timing error is also low. Obviously, both previous figures depend on the simulated platform.

We also finally made some experiments to compare our simulation speed with some existing TLM methodologies to see the loss of performance brought by adding time to the simulation. As we dispose of only generic components for the classical TLM approach (we used the TLM package from OSCI), we had to model a very simple platform using two masters doing read and write access to a single RAM (it is the example 4.2 from the OSCI TLM package). Each master repeats a pattern of twenty accesses to the RAM. We will use the number of times we encounter that pattern as our metric.

TABLE III
SIMULATION SPEED FOR A TLM AND THE PROPOSED APPROACH

	TLM	TLM/T
number of pattern for master 1	0	29853
number of pattern for master 2	48376	29853
simulation time	2.84 s	6.53 s
simulation speed	≈ 17000	≈ 9100

There is no access to the RAM for the master 1 as the arbitration scheme used in the TLM example allows starvation. Thus, we see that, adding timing information to the simulation led to a speed that is only half the one of the TLM simulation. It has to be noted that the components in TLM/T are way more complex than the one in the TLM platform which explains a part of the simulation speed loss.

VI. CONCLUSION

We showed in this paper some tools that exploit the interest of the TLM/T level of modeling. They provide an efficient mean to model complex SoCs and explore the different architectures available for a system but also some practical ways to develop and debug the associated embedded software, allowing a fast simulation of the correct parts of the software and a "temporal zoom" on the faulty ones. The timing error resulting from that method can be kept fairly low if the model of the different components are accurately written.

Rather than replacing the whole platform when switching between abstraction levels, an interesting improvement could be to replace only a few components, meaning that we would have a way to easily do co-simulation without having to rely on third-party softwares.

REFERENCES

[1] A. Gerstlauer, D. Shin, R. Dömer, and D. Gajski, "Systel-level communication modeling for network-on-chip synthesis," in *ASP-DAC 2005*. IEEE Computer Society, 2005, p. 45.

[2] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. ACM Press, 2003, pp. 19–24.

[3] A. Donlin, "Transaction level modeling: flows and use models," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM Press, 2004, pp. 75–80.

[4] A. Rose, S. Swan, J. Pierce, and J.-M. Fernandez, "Transaction Level Modeling in SystemC," http://www.systemc.org, January 2005.

[5] Z. Kadi and P. Klein, "Efficient passive-TLM and transaction management," in *First North American SystemC User's Group Conference*, 2004.

[6] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in *Proceedings of the 41st annual conference on Design automation*, 2004, pp. 113–118.

[7] P. Garg, S. Shukla, and R. Gupta, "Efficient usage of concurrency models in an object-oriented co-design framework," in *DATE '01: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2001.

[8] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM*, vol. 24, no. 4, pp. 198–206, 1981.

[9] R. E. Bryant, "Simulation of packet communication architecture computer systems," Tech. Rep., 1977.

[10] E. Viaud, F. Pêcheux, and A. Greiner, "An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles," in *DATE '06: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2006.

[11] N. Savoiu, S. Shukla, and R. Gupta, "Automated concurrency reassignment in high level system models for efficient system-level simulation," in *DATE '02: Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA: IEEE Computer Society, 2002, p. 875.

[12] K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985.

[13] *Virtual Component Interface Standard*, http://www.vsi.org.

[14] SoCLIB, "A modelisation & simulation platform for system on chip," 2003, http://soclib.lip6.fr.