Co-Simulation of Functional SystemC TLM Models with Power/Thermal Solvers

Tayeb Bouhadiba, Matthieu Moy, Florence Maraninchi, Jerome Cornet, Laurent Maillet-Contoz, Ilija Materic

Verimag Research Report n^o TR-2012-21

December 20, 2012

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Co-Simulation of Functional SystemC TLM Models with Power/Thermal Solvers

Tayeb Bouhadiba, Matthieu Moy, Florence Maraninchi, Jerome Cornet, Laurent Maillet-Contoz, Ilija Materic

December 20, 2012

Abstract

Modern systems-on-chips need sophisticated power-management policies to control their power consumption and temperature. These power-management policies are usually implemented partly in software, with hardware support. They need to be validated early, hence power and temperature-aware simulation techniques at the system-level need to be developed. Existing approaches for system-level power and thermal analysis usually either completely abstract the functionality (allowing only simple scenarios to be simulated), or run the functional simulation independently from the non-functional one.

The approach presented in this paper allows a coupled simulation of a SystemC/TLM model, possibly including the actual embedded software, with a power and temperature solver such as ATMI or the commercial tool ACEplorer. Power and temperature analysis is done based on the stimuli sent by the SystemC/TLM platform, which in turn can take decisions based on the non-functional simulation.

Keywords:

Reviewers:

How to cite this report:

```
@techreport {verimag-TR-2012-21,
  title = {Co-Simulation of Functional SystemC TLM Models with Power/Thermal Solvers},
  author = { Tayeb Bouhadiba, Matthieu Moy, Florence Maraninchi, Jerome Cornet, Laurent
Maillet-Contoz, Ilija Materic },
  institution = {{Verimag} Research Report},
  number = {TR-2012-21},
  year = {2012}
}
```

1 Introduction

Today's highly integrated electronic chips have millions of transistors and consume lot of power (couple of tens of Watts). This introduces several problems: most obvious are battery life for portable devices as well as heating of devices integrating these chips. In addition, even for devices that are continuously plugged to power supply, power consumption has an impact on operating costs and chip reliability.

Reducing power consumption and heating problems requires that these issues be taken into account throughout the design of the chip. This means estimating power and thermal behavior very early, but also designing a "power-aware" embedded software, that can react upon given power/temperature conditions (switching blocks off or degrading quality of service).

Early power and temperature estimation is tackled using standalone power/thermal model simulations. With such simulations, stimuli are produced through manually written high-level scenarios that depict activations, power-state changes, etc. For each component in the model, the stimuli describe for a given use-case the evolution of a set of parameters, including the electrical state (voltage, frequency) and the activity (kind of computation being performed, or traffic for components like buses and memories), which influences the current intensity.

Writing a power-aware embedded software requires more. Hand-written scenarios are too abstract: they can validate design choices, but not the actual software implementation. RTL simulators, on the other hand, are far too slow to execute non-trivial embedded software, even without power/temperature instrumentation. Also, they are not available in the early phases of the design flow. A TLM simulation is therefore needed; it has enough detail to execute the actual embedded software, but is still fast enough to run non-trivial applications. The TLM model must be augmented with non-functional aspects to show the developer the effect of running the embedded software from the power/thermal point of view. The result of TLM simulations can hardly be as precise as RTL or gate-level simulation, but they are the only option for early system-level modeling, and an improvement compared to handwritten scenarios.

TLM models can be instrumented to produce the stimuli traces described above, which can be imported in the power/thermal simulator to perform an offline analysis. A VCD (Value Change Dump) or other standard description may be used for that purpose. However, with this technique, it is not possible to test power management strategies, because there is no feedback loop between the functional behavior of the TLM simulation and power/thermal estimation.

In this work, we address this particular point: we propose a cosimulation method to integrate a TLM simulation with a power/thermal simulation done in parallel. Figure 1 is an architectural view of the cosimulation. The simulations act on each other: the TLM simulation (**a1**) provides functional stimuli (and possibly threshold temperatures) for the power simulation (**a2**); the thermal solver uses the non-functional model (actual consumption in different mode of operations for each component) to provide power values for the thermal simulation (**a3**) based on a thermal model (floorplan, thermal conductivity); the latter influences the functional behavior through temperature feedback, or interrupt notifications if threshold temperatures are reached. The exchange between (**a2**) and (**a3**) describes a possible temperature-dependent power analysis computation.

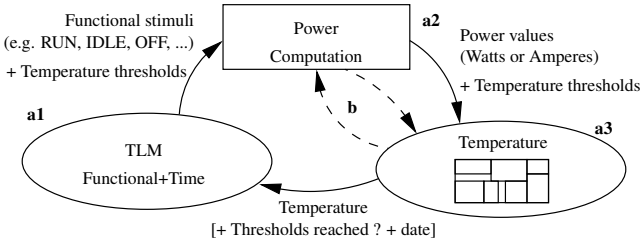


Figure 1: **a1**, **a2**, **a3**: one step of the co-simulation principle, **b**: temperature/power consumption feedback effect

The contributions of this paper are:

• We describe an interface to let a timed functional model of an SoC communicate with an external power/temperature (i.e., non-functional) solver. This is a cross-language interface, that might be used

locally or over the network. It is generic in two senses: it allows connecting to multiple non-functional solvers without changing the functional model, and it allows several strategies for coupled simulation, leaving room for optimizations and performance/precision trade-off. We consider SystemC functional models, but any discrete-event simulator with a notion of simulated time could be used instead.

• We present and compare techniques using this interface to run a coupled simulation, allowing the functional model to access power and temperature values computed by the solver to take decisions in the functional simulation. These techniques were implemented both with a simple temperature solver and an industrial power and temperature analysis tool. The implementation of the interface on the functional side is available as a library that can run on any SystemC implementation.

2 Related Work

The significant contribution of static power in today's SoCs consumption, and its dependency on temperature, makes it no longer possible to rely on simple power simulators without power/temperature feedback loop. Instruction-based TLM power simulators [1], or state-based ones [2], are fed by functional stimuli to estimate power consumption, independently from SoC temperature. We target thermal aware power analysis, like it is done in tools featuring power/temperature feedback [3].

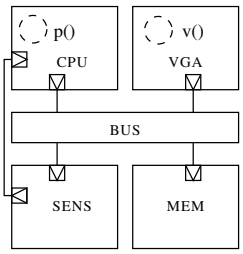
Thermal simulations rely on the well established duality of heat transfer and electrical phenomena. This led to modeling thermal properties by means of RC-circuits, and using numerical solvers to compute temperature evolution. In the context of TLM, this means that there is a need for mixing discrete simulations with continuous ones. The problem then falls in the domain of modeling and simulation of heterogeneous systems. Tools encompassing heterogeneous modeling have been proposed. Ptolemy [4], for instance allows to describe distinct models of computation, and to organize them into a hierarchy to model heterogeneity [5]. In Ptolemy, the embedded model of computation must synchronize its clock with the upper-level one, based on the time-stamp of the exchanged data. The authors in [6] propose heterogeneous modeling of synchronous reactive programs together with differential equations for modeling physical phenomena. VulcaNoCs [7] allows for modeling the functional behavior of Networks-on-Chips with cycle accurate SystemC-TLM, and relies on the Electrical Linear Network model of computation provided by SystemC-AMS [8] to implement the RC-Circuit modeling the thermal behavior. VulcaNoCs targets proactive thermal management. Therefore, there is no feedback of temperature in the functional model for the validation of *reactive* power/thermal management.

Contrary to VulcaNoCs, we use external, domain specific, tools for thermal simulation (e.g., HotSpot [9], ATMI [10], Aceplorer [3], etc.) and we need to feedback power/temperature in the functional model. Power/temperature modeling tools can provide more features and be less error prone than user written solvers based on general purpose tools. When using external tools, the traditional approach is to dump the power traces in a file (e.g. using VCD file), and to perform the analysis *offline*. Therefore, the result of power and temperature analysis cannot be used by the functional simulation. For that purpose, we propose the cosimulation of SystemC/TLM with external power/thermal simulators.

Cosimulation of SystemC and external tools (e.g., Simulink) has been proposed in [11, 12, 13]. We could use the proposed API in [12], but the type of the exchange between simulators in our approach is richer than simple signals. We also distinguish our work in the possible optimizations when synchronizing simulators. Indeed, their optimizations are guided by data exchange periodicity (e.g., sampling periods which are application-dependent), and may rely on the heavy mechanism of rollback in SystemC. Our improvements are better; in particular because of the type of the exchanges wich enable temporal decoupling of simulators, and even their parallelization. From the implementation point of view, we do not patch the SystemC scheduler as they do.

3 SystemC and TLM

Transaction Level Modeling (TLM) is an approach to virtual prototyping of Systems-On-a-Chip (SoCs). In TLM, hardware components and interconnects are modeled by means of modules that communicate through transactions. The reference implementation of TLM is provided as SystemC specific libraries and



(a)

```
1   vga::v () {
2     while (1) {
3       set_pwr("activity", "wait");
4       // wait for the next refresh
5       wait(8, sc_ms);
6       // load image and wait
7       // the time it takes
8       load_img();
9       set_pwr("activity", "load");
10      wait(3, sc_ms);
11    }
12  }
```

(b)

Figure 2: Example TLM model and SystemC code of the threads

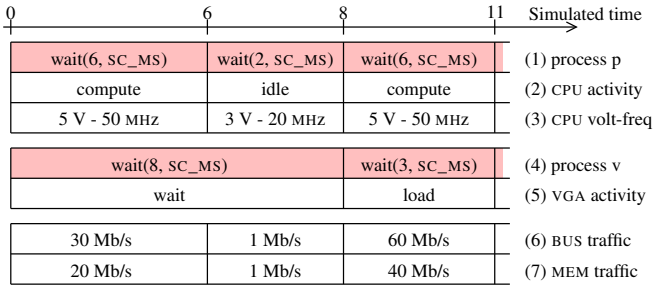


Figure 3: Power Parameter Trace of the Example in Section 3

templates. SystemC which is a C++ library, including a simulation engine, has become the reference in the industry as well as an IEEE standard [14].

Figure 2-(a) is an example model of a simple SoC made of a CPU, a VGA controller, a memory MEM, and a temperature sensor SENS. The sensor may be configured to send interrupts to the CPU (through the signal connecting the components) when the temperature reaches some threshold values. The behavior of the system consists in displaying images. The CPU writes images to the memory, and configures the VGA with the address of the image to be displayed.

The behavior of the simulation is defined by the execution of the threads as managed by the SystemC scheduler. The simulation produces a set of *simulation instants* $t_0 = 0, t_1, t_2, \ldots$; at each instant, some of the threads execute part of their behavior (the code between two wait(...) statements) atomically and suspend themselves by calling wait(...), which schedules them to be woken up later in the simulation (either at a particular time, or when a particular event is triggered). We use the term *simulation intervals* to refer to the successive adjacent intervals $[t_0, t_1], [t_1, t_2], \ldots$. Of course, since SystemC is a simulation language, the simulated time is different from the wall-clock time. It should be noted that computations occur only at simulation instants: the simulation intervals only correspond to the increment of simulated time in the scheduler.

Modeling an action that takes time (say, load_img(), taking 3 ms) is not directly possible in SystemC, which can express only instantaneous computations. The duration can be modeled with a wait(*time*) statement. One has to choose at which simulation instant the action should be executed. A common practice is to run the functional behavior first, followed by the wait statement (e.g. load_img(); wait(3, SC_MS);). If the duration of the action depends on the actual computation being performed, we would let load_img compute a duration t, and the following wait can be performed as wait(t). A particular case of this is TLM-2's *temporal decoupling* [14], where t is computed by maintaining a local clock, which is reset to 0 upon calling wait. In this paper, we follow these guidelines to model tasks that take time; i.e., "run the functional behavior first, and call wait afterwards to model its duration".

4 Power Instrumentation of TL-Models

Power instrumentation is out of the scope of the paper, this section gives an insight of the kind of information we rely on during the cosimulation, and how it is obtained.

Because the same functional platform may be used with several power models, we do not back-annotate SystemC code directly with numerical values, but instead apply the separation of concerns principle [15] and keep the numerical values in a separate model. We extend SystemC modules with a set of power parameters that might be of several types (voltage, frequency, circuit activity, etc.). The value of each parameter is set by the functional model at appropriate points in time. For example, the calls to set_pwr at lines 3 and 9 set the parameter activity to the corresponding mode, and allow producing line (5) in the execution trace of Figure 3.

Power instrumentation benefits from the guideline in which the *functional behavior* is simulated first in zero-time, then the *wait* statement declares its duration. During the simulation instant t and before the simulation jumps to the next instant, we are able to set power parameters, and compute the time Δt for which these states hold.

Instrumented SystemC/TLM models generate a trace of power parameters; that is, simulation intervals associated with components' power parameters. Fig. 3 depicts the power trace of the individual components of the platform in Fig. 2. Traces (2) and (3) in Fig. 3 depict the activity and electrical parameters of the CPU as set by the behavior of the process p() of the CPU. Trace (5) depicts the activity parameter of the VGA as set by the process v(). The execution of p() and v() generate transactions. The observed transactions on each of the BUS and MEM are quantified to computed average traffic frequencies resulting in traces (6) and (7) of Fig. 3.

From such traces, a power simulator can compute *power traces* that associate each component with its static and dynamic power consumption. Static and dynamic power consumption are functions of power states. Notice that static power computation may depend on system temperature.

5 SystemC and Power/Temperature Solver Cosimulation Interface

This section presents an approach where an external solver can be used, but without the drawback of offline analysis. We co-simulate the functional behavior with the power and temperature solver, which allows a bidirectional interaction between the functional behavior and the non-functional aspects: the functionality can for example change depending on the temperature of the system, and the temperature still depends on power consumption hence on the functionality.

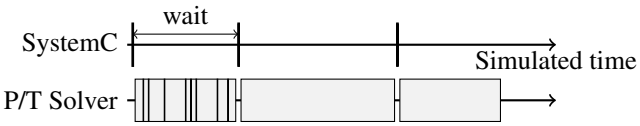


Figure 4: Simulated time in both simulators

Figure 4 illustrates the relationship between simulated time on the functional (SystemC) and non-functional (Power/Temperature) simulators. On the SystemC side, all computations are done at well-defined simulation instants. On the non-functional side, the behavior within simulation intervals is continuous. In most cases, this continuity is simulated using iterations over time internally, possibly with a variable step (as shown on the figure). We consider the simulation intervals as "black boxes" and do not want to rely on internal implementation details of the solver. Note that this relationship between *simulated* times does not necessarily force the order in which the simulations will be executed (see Section 6 for details).

Figure 5 illustrates the global view of the cosimulation architecture. On the left part is the SystemC functional model instrumented with power parameters. On the right-hand side of the figure is the temperature solver. The SystemC component SYNCHRO is added between the platform itself and the power/temperature solver to deal with data exchange and synchronization between simulators. The SYNCHRO

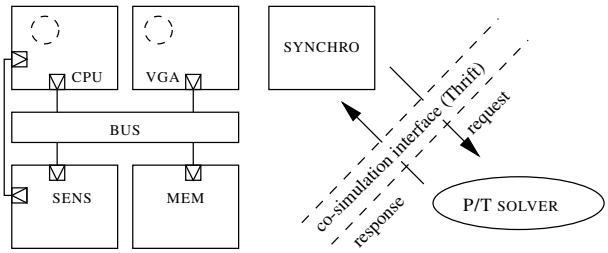


Figure 5: Architectural View of the Cosimulation

component implements several synchronization strategies; they are discussed in Section 6. The next section describes the type of exchanges that happen between the simulators.

5.1 Description and Implementation of the Interface

The discrete simulator (e.g., SystemC) produces the trace that is exploited by the continuous simulators (e.g., temperature solver). At some point in the cosimulation, the SystemC model sends a *request* describing the evolution of power parameters during an interval of simulated time (one, or possibly several simulation intervals). The non-functional solver will reply with a *response* that describes the result of simulation on the requested time interval.

The information contained in a request is similar to the one contained in execution traces used in offline analysis, but describe only the evolution of parameters during the time interval $[t_i, t_j]$ (one, or several simulation intervals); it is comparable to the information available in VCD files. The implementation of the co-simulation interface uses Thrift [16], an efficient, cross-language and cross-platform remote procedure call protocol. Figure 6 describes the type definition of the SystemC requests and the solver responses.

5.1.1 The request

A request is a `struct` composed of three main fields: (i) *until_date* is the time horizon to which the solver should advance in order to synchronize with the SystemC simulation; (ii) *value_changes* is a list of components' power states time-stamped with the date at which they changed their value; (iii) *halt_conditions* is a list of threshold values on which the simulation should be interrupted (e.g. temperature goes over a threshold, and the functional simulation should react immediately with an emergency stop). The solver should stop when one of these thresholds is crossed, and indicate the date at which it happened (i.e., zero-crossings).

5.1.2 The response

Based on the request it receives, the solver computes the components' power consumption and temperature over interval $[t_i, t_j]$ and construct a *response*. The response (see Fig. 6) is a `struct` composed of the following fields: (i) *halt_date* is the simulated time reached by the solver; (ii) *changed_values* is a list of power and temperature values time-stamped with the dates at which they were reached; (iii) *halt_causes* is a list of conditions causing the termination of the solver computation. The computed values at time t_j (a *response*) are sent back to the SystemC functional model. If the simulation stopped because of a *halt_condition*, the response mentions which condition was triggered.

The component SYNCHRO decides when the non-functional simulation should be performed, and is in charge of constructing the request, and dispatching non-functional values to the appropriate components (e.g., informing the SENS component of the new temperature value).

```
1  /* Request simulation up to until_date. */
2  struct power_simu_request {
3      1: required double until_date,
4      2: required list<value_change> value_changes,
5      3: required list<halt_condition> halt_conditions
6  }
7
8  /* Response from the power simulator */
9  struct power_simu_response {
10     1: required double halt_date,
11     2: required list<value_change> changed_values,
12     3: required list<halt_condition> halt_causes
13 }
14
15 service cosim_service {
16     power_simu_response simulate(1: power_simu_request rq);
17 }
```

Figure 6: Excerpt from the Thrift Specification of the Cosimulation interface

6 Cosimulation Strategies

Section 5 described the interface between the functional and non-functional solvers and the data-exchange between them, but left apart the question of synchronization: when should non-functional simulations be triggered, and on which time intervals. We now describe the various possible strategies.

6.1 Simulate Intervals One by One (Lockstep)

A possible cosimulation strategy is to synchronize simulators at the end of each simulation instant (*lockstep* strategy, Figure 7). Suppose the current simulation instant is t_i. All eligible processes at instant t_i execute (Fig 7.(1)). Before the SystemC scheduler jumps to instant t_{i+1}, the SYNCHRO component suspends the SystemC model, constructs a `request` according to power states and halt conditions. The time horizon of the requested simulation is set to the next SystemC instant t_{i+1} (7.(2)).

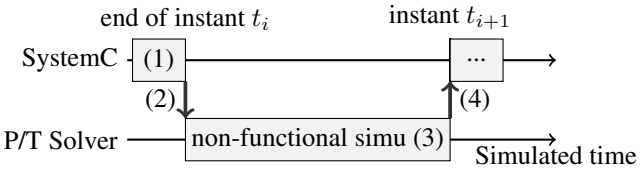


Figure 7: *lockstep* cosimulation strategy. For clarity, simulated instants are represented with a non-null width.

Upon receiving the request, the solver computes components' temperature according to the received power states (7.(3)). The temperature is computed until the time horizon t_{i+1} if no halt condition was encountered. The SYNCHRO component receives the response of the solver, updates registers of the temperature sensors, and resumes the SystemC execution (7.(4)). The updated temperature is valid for the SystemC instant t_{i+1}. Components executing at this instant will access the right temperature. "Suspending" and "resuming" the simulation is performed by sending the request and waiting for the response. Since SystemC simulation is sequential, waiting for the response effectively suspends the simulation.

6.1.1 Interrupt Triggered by Non-Functional Solver

In case the solver encounters a halt condition at time $t_i+\delta t$ (where $t_i+\delta t < t_{i+1}$), it stops the non-functional simulation at that time (See Fig 8). The response of the solver (8.(4)) gives component temperatures at $t_i + \delta t$ and the *halt_conditions* encountered. The SYNCHRO component receiving the solver response programs the temperature sensor to trigger an interrupt at $t_i + \delta t$ (using a timed event notification in SystemC). The consequence of such an interrupt, is to create a new SystemC instant to which the scheduler

will jump instead of resuming the simulation at t_{i+1}. This is possible without any backtrack in SystemC: indeed, the simulator was suspended at the end of simulation instant t_i and before jumping to the instant t_{i+1}, and can still notify an event at time $t_i + \delta t$.

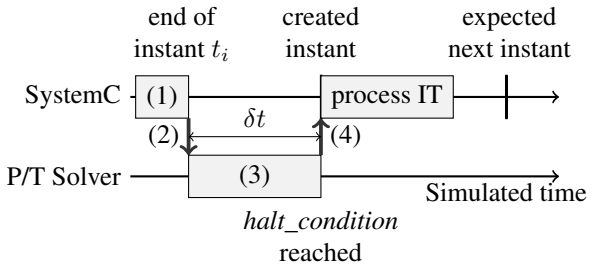


Figure 8: *lockstep* cosimulation strategy with interrupt

6.2 Optimization in the Absence of Non-Functional Interrupts

The benefit of the *lockstep* synchronization lies in the fact that the non-functional simulator runs *between* simulation instants, hence it is possible to interrupt the non-functional simulation on *halt_conditions*. These interrupts may change the future of the simulation without questioning its past (i.e., no backtrack). However, the *lockstep* strategy requires a round-trip between simulators for each simulation instant, which may result in non-negligible simulation overhead.

6.2.1 Functional Ahead Strategy

In the absence of non-functional interrupts, we can perform better, avoiding switching simulators at each SystemC instant (*functional ahead* strategy, see Figure 9). Synchronization is performed only when it is required. The SystemC simulation may run multiple instants ahead of the solver, until a non-functional value is required (e.g. there is a read access on a temperature solver, (9.(1)), and a single request is made for the set of time intervals corresponding to the instants executed (9.(2)). The non-functional simulator then simulates the trace (9.(3)). Note that in this case, the power parameters may change while the request is processed (this is the reason why a request contains a set of value changes with the associated simulation time, and not only a set of values). The response (9.(4)) contains the values that were required on the SystemC side.

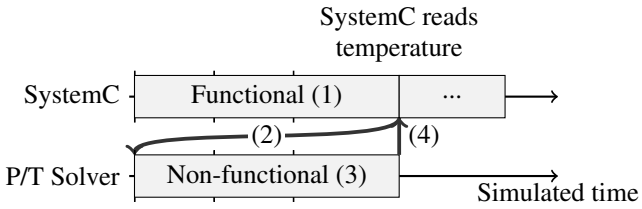


Figure 9: *functional ahead* strategy, in the absence of interrupt

6.2.2 Running both Simulations in Parallel

An improvement over the *functional ahead* strategy is to run the simulators in parallel. When the functional simulation does not need any non-functional values, the execution follows a simple producer-consumer scheme (the functional simulation produces requests that are consumed by the non-functional simulator). When the functional simulation requires a non-functional value, a synchronization is triggered: the functional simulator is blocked until it receives the last response, which contains the required values.

6.3 Dynamic Selection of Strategy

We propose a technique to select and change strategies at runtime, according to observations of the functional configuration. We may for instance exchange, at runtime, *lockstep* and *functional ahead* strategy during simulation. For the sake of cosimulation speed, *functional ahead* strategy is preferred, but the *lockstep* must be used to raise interrupts correctly.

The decision on which strategy to use starting from instant t relies on the information we have on interrupts. If we expect an interrupt for the next simulation interval then the *lockstep* strategy must be used, in order to prevent SystemC from advancing at a time greater than the occurrence date of the interrupt. Otherwise, select the *functional ahead* strategy. At the end of each simulation instant, the SYNCHRO component checks a sensor's register to update the strategy.

7 Implementation

We provide an implementation of the SYNCHRO component, as part of a synchronization library for SystemC/thermal solver cosimulation. The SYNCHRO component provides the above mentioned synchronization strategies. When to synchronize with the solver is up to the strategy being used; but this always happens at the end of a simulation instant.

In order to execute code at the end of simulation instants, we add a wrapper method for the sc_start() method of SystemC. The wrapper method performs instant-by-instant simulation: it calls sc_start(next_t) in a loop with next_t being the date of the nearest event. next_t is returned by the SystemC API.

The SYNCHRO component uses the thrift interface to call the thermal solver (e.g., ATMI [10], HotSpot [9] and Aceplorer [3]) locally or remotely over the network.

8 Experimental Results

8.1 Validation of Power/Thermal Management Policies

Figure 10 illustrates power/thermal plotting of the example of section 3. The CPU implements a power/thermal management policy, sensitive to temperature sensor interrupts. Interrupts notify two situations: i) the system is heating up (e.g., instants 0.78s and 1.33s in Figure 10); the CPU, then, scales down its voltage and frequency; and this impacts power consumption (see Figure 10); ii) the temperature is cooling down to a normal value (e.g., instant 1.17s in Figure 10). The CPU, then, scales up its voltage and frequency.

We use ATMI as a thermal solver, extended with a module to translate power states (those in Figure 3) into power values. Since the power management is sensitive to sensor interrupts, we use the *lockstep* strategy to synchronize simulators.

We compared the performances of different simulation strategies on a video decoding TL-Model, where the embedded software polls the sensor temperature to decide on the power configuration (no interrupts). Table 1 shows execution time for the 3 strategies described above with different simulation time step, using ATMI as a temperature solver. We can see that the *functional ahead* strategy performs similarly or better than *lockstep*. The benefit of *parallel* depends on the granularity, and the parallel version can actually perform worse than the sequential ones. Figure 2 is a summary of simulation speeds. We distinguish the time taken by the SystemC part (SC), the ATMI part, and the connection between them (between parentheses is the number of exchanges between the simulators). We can see that the overhead of connection is small, and even with thermal simulation, coarse-grain simulations remain much faster than low-level ones.

Table 3 summarizes the results of several cosimulations varying the length of simulation intervals of the TL-Model and the ATMI integration step. For example, with an average simulation interval length of the TL-Model of *200* μs (column 1), and a time-step of *100* μs for ATMI integration (row 1), parallelization performs worst taking 5% longer than the time taken by the lockstep strategy. With a finer time-step of ATMI (i.e, *50* μs), parallelization performs better (about 20%). The results show that the outcome of the parallelization is not always positive, and this is not a matter of implementation (of the strategy). At a first glance, parallel cosimulation would reach its optimal (i.e.; about 50% compared to lockstep), when both SystemC and the thermal solver take the same amount of time simulating a time interval δt. In

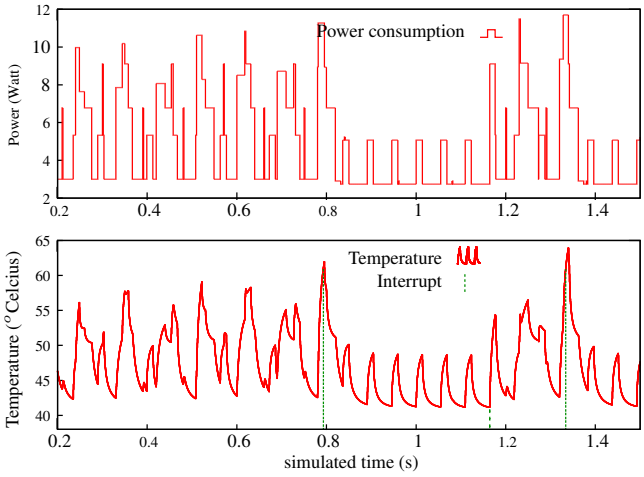


Figure 10: Power and thermal simulation for the case study of section 3

		strategy		
		lockstep	functional ahead	parallel
ATMI Step	1000 μs	0:59	0:59	1:17
	100 μs	1:45	1:48	1:07
	10 μs	11:30	10:57	9:53

Table 1: Timings for different strategies (minutes:seconds)

	Time	SystemC/TLM	ATMI	Connection
1-inst.	1028s	48.8%	41.2%	11% (15.7E+6)
100-inst.	185s	5.2%	94.5%	0.03% (0.15E+6)
Coarse-grain	139s	5%	95%	$\approx$ 0% (128)

Table 2: Execution times and contributions of the simulator parts, for simulating $0.5s$ of the system

depth performance analysis is in progress; it would allow to choose the optimal synchronization strategy depending on the TL-Model and the solver being used.

		TL-Model Simulation Interval Length			
		0.2 ms	1 ms	20 ms	200 ms
ATMI Step	100 μs	-5%	-11%	-13%	-10%
	50 μs	+20%	+17%	+8%	+13%
	10 μs	-16%	+9%	+16%	+11%
	1 μs	-12%	+8%	+1%	-5%

Table 3: Percentage of gain/loss of the Parallel Strategy Compared to the Lockstep one

We also performed experiments using Aceplorer as a power/temperature solver, which was modified to allow cosimulation through the Thrift interface. The SystemC/TLM platform models the video subsystem of an SoC. While the platform itself is relatively simple, it was written using STMicroelectronics's state-of-the-art development kits (several hundred thousands lines of code in total). Simulating 16 seconds of simulated time takes 1 hour 36 minutes, most of which is due to the thermal simulation: less than 2 minutes are needed if thermal modeling is deactivated. The case study was briefly presented in [17].

Simulation results are provided in Figures 11 and 12. One can see the effect of the power management/quality of service policy: when the temperature reaches a certain threshold, the video decoder is re-programmed to decode only one image out of 3. As a result, the temperature decreases, and the power management reactivates the normal behavior.

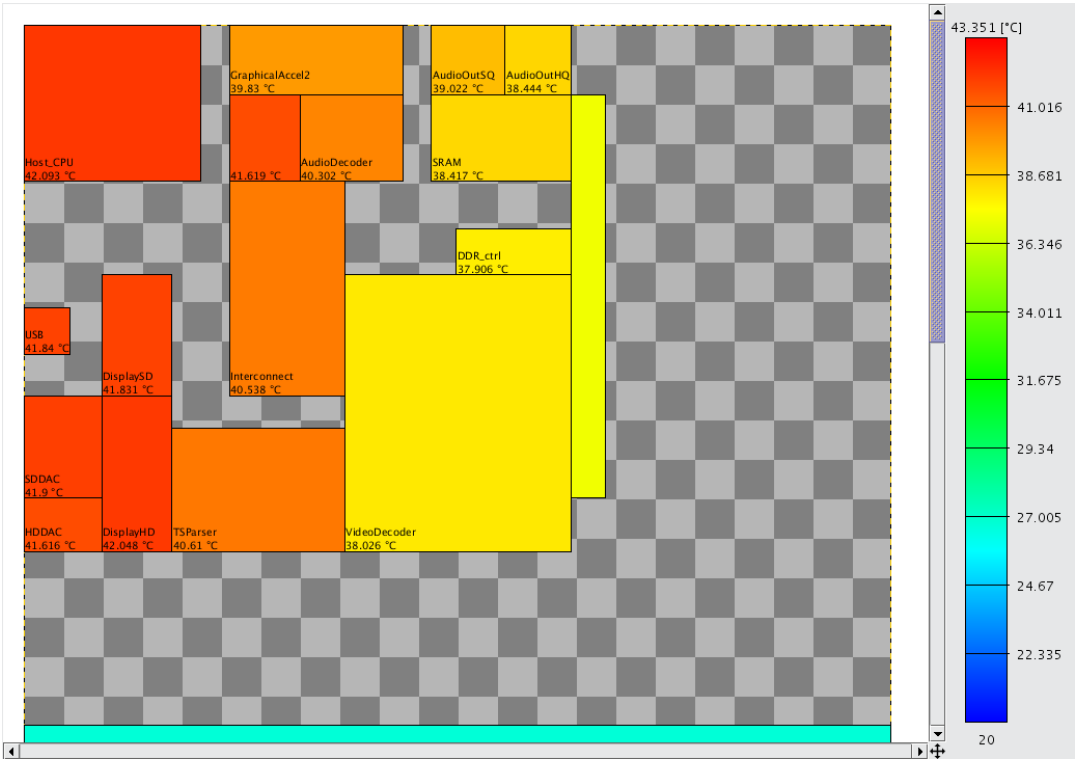


Figure 11: Floor-plan of the SoC model (Aceplorer screenshot)

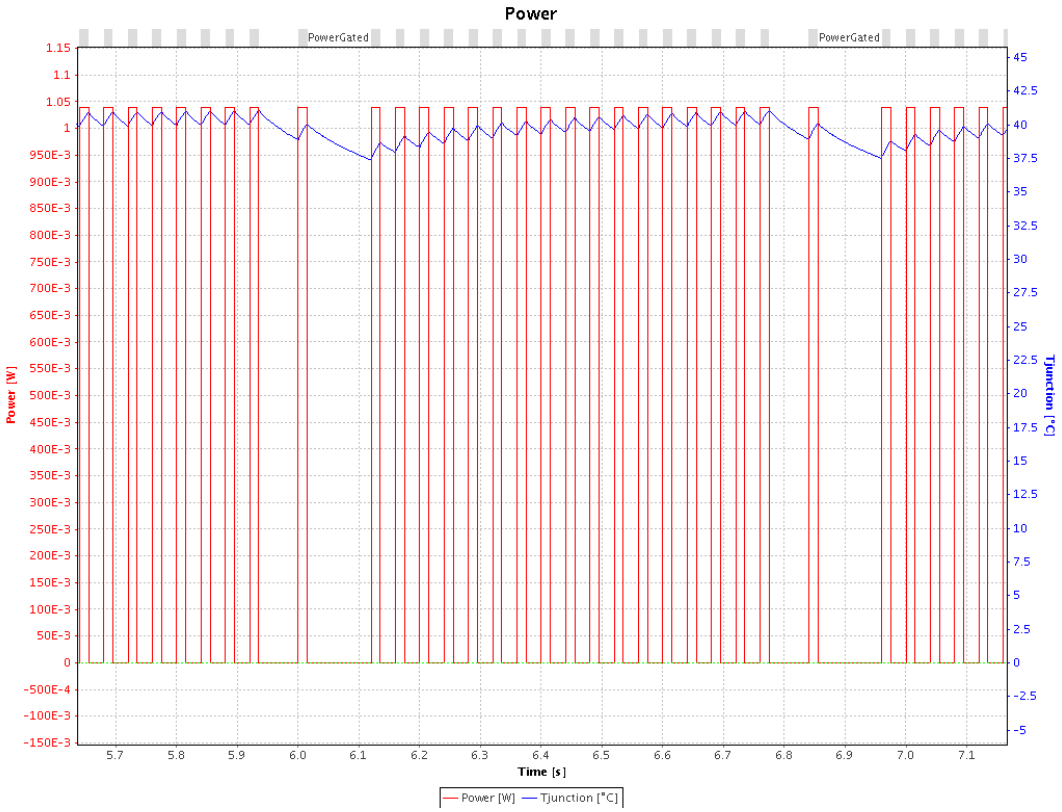


Figure 12: Evolution of power and temperature of the video decoder component (Aceplorer screenshot)

9 Conclusion

We present an approach for power/thermal simulation for functional TLM models. The cosimulation principles make it possible to take advantage of existing, domain specific tools and enable feedback of non-functional properties in the functional model. The approach features a cross-language cosimulation interface that enables local or remote execution of the solver in a transparent way.

The type of the exchange as defined by the cosimulation interface imposes separation of concerns when modeling power. The cosimulation framework is thus generic: the same functional model may be analyzed according to distinct power/thermal models, describing distinct physical and architectural parameters of the chip, possibly modeled with distinct domain-specific tools. The cosimulation interface is independent from the simulators synchronization, and was though in order to operate with multiple synchronization strategies. We presented these strategies and show some comparison results for local executions. Remote execution performance were not showed because of lack of space.

Future work will consider two directions: i) *performance analysis:* comprises the comparison of distinct power/thermal simulators, and an in-depth analysis of the cosimulation strategies both in local and remote executions. ii) *Extensions with other non-functional properties:* we want to extend the modeling with the behavior of battery level indicators, in order to validate power-saving strategies early, or to simulate the battery discharge profile and lifetime.

References

[1] N. Dhanwada, I.-C. Lin, and V. Narayanan, "A power estimation methodology for systemc transaction level models," in *Proceedings of the 3rd*, ser. CODES+ISSS '05. New York, NY, USA: ACM, 2005, pp. 142–147. 2

[2] H. Lebreton and P. Vivet, "Power modeling in SystemC at transaction level, application to a DVFS architecture," in *Symposium on VLSI. ISVLSI'08.* IEEE, 2008, pp. 463–466. 2

[3] http://www.doceapower.com. 2, 7

[4] http://ptolemy.eecs.berkeley.edu. 2

[5] J. Liu and E. A. Lee, "A component-based approach to modeling and simulating mixed-signal and hybrid systems," *ACM Trans. Model. Comput. Simul.*, vol. 12, no. 4, pp. 343–368, Oct. 2002. 2

[6] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet, "A hybrid synchronous language with hierarchical automata: static typing and translation to synchronous code," in *EMSOFT '11*. New York, NY, USA: ACM, 2011, pp. 137–148. 2

[7] T. Wegner, C. Cornelius, M. Gag, A. Tockhorn, and A. Uhrmacher, "Simulation of thermal behavior for networks-on-chip," in *NORCHIP, 2010*, nov. 2010, pp. 1 –4. 2

[8] *SystemC AMS LRM*, Accellera Systems Initiative, 2011. 2

[9] W. Huang, S. Member, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, M. R. Stan, S. Member, and S. Member, "Hotspot: A compact thermal modeling method for CMOS VLSI systems," *IEEE Transactions on VLSI Systems*, vol. 14, pp. 501–513, 2006. 2, 7

[10] P. Michaud and Y. Sazeides, "ATMI: analytical model of temperature in microprocessors," *Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2007. 2, 7

[11] F. Bouchhima, G. Nicolescu, E. M. Aboulhamid, and M. Abid, "Discrete-continuous simulation model for accurate validation in component-based heterogeneous soc design," in *IEEE International Workshop on Rapid System Prototyping*. IEEE Computer Society, 2005. 2

[12] F. Bouchhima, M. Briere, G. Nicolescu, M. Abid, and E. Aboulhamid, "A systemc/simulink co-simulation framework for continuous/discrete-events simulation," in *Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International*, sept. 2006, pp. 1 –6. 2

[13] L. Gheorghe, F. Bouchhima, G. Nicolescu, and H. Boucheneb, "Semantics for model-based validation of continuous/discrete systems," in *DATE*. IEEE, 2008, pp. 498–503. 2

[14] *IEEE 1666 Standard: SystemC Language Reference Manual*, Open SystemC Initiative, 2011. [Online]. Available: http://www.accellera.org/ 3, 3

[15] S. Kaiser, I. Materic, and R. Saade, "Esl solutions for low power design," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 340–343. 4

[16] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, 2007. 5.1

[17] J. Cornet, L. Maillet-Contoz, I. Materic, S. Kaiser, H. Boussetta, T. Bouhadiba, M. Moy, and F. Maraninchi, "Co-Simulation of a SystemC TLM Virtual Platform with a Power Simulator at the Architectural Level: Case of a Set-Top Box," in *Design Automation Conference*, San Francisco, États-Unis, Jun 2012, p. SESSION 10U: USER TRACK. 8.1