Modeling Power Consumption and Temperature in TLM Models

Matthieu Moy[1], Claude Helmstetter[2], Tayeb Bouhadiba[3], and Florence Maraninchi[4]

1 **Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France**
CNRS, VERIMAG, F-38000 Grenoble, France
http://orcid.org/0000-0002-6054-8882
matthieu.moy@imag.fr

2 **Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France**
CNRS, VERIMAG, F-38000 Grenoble, France
http://orcid.org/0000-0002-2323-6919
claude.helmstetter@imag.fr

3 **Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France**
CNRS, VERIMAG, F-38000 Grenoble, France
http://orcid.org/0000-0003-1694-6709
tayeb.bouhadiba@imag.fr

4 **Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France**
CNRS, VERIMAG, F-38000 Grenoble, France
http://orcid.org/0000-0003-0783-9178
florence.maraninchi@imag.fr

Abstract

Many techniques and tools exist to estimate the power consumption and the temperature map of a chip. These tools help the hardware designers develop power efficient chips in the presence of temperature constraints. For this task, the application can be ignored or at least abstracted by some high level scenarios; at this stage, the actual embedded software is generally not available yet.

However, after the hardware is defined, the embedded software can still have a significant influence on the power consumption; i.e., two implementations of the same application can consume more or less power. Moreover, the actual software power manager ensuring the temperature constraints, usually by acting dynamically on the voltage and frequency, must itself be validated. Validating such power management policy requires a model of both actuators and sensors, hence a closed-loop simulation of the functional model with a non-functional one.

In this paper, we present and compare several tools to simulate the power and thermal behavior of a chip together with its functionality. We explore several levels of abstraction and study the impact on the precision of the analysis.

2012 ACM Subject Classification Hardware - Power and energy - Power estimation and optimization - Chip-level power issues

Keywords and phrases Power consumption, Temperature control, Virtual prototype, SystemC, Transactional modeling

Digital Object Identifier 10.4230/LITES-v003-i001-a003

Received 2015-07-10 **Accepted** 2016-04-29 **Published** 2016-06-29

1 Introduction

Reducing power consumption of Systems-on-a-Chip is an important challenge. Clearly, portable devices should save energy to maximize battery life, but other issues like heat dissipation [21], voltage drop [11] or faster aging [14] due to overheating are also of growing importance.

With modern CMOS processes, static power consumption, due to leakage current, is increasing. Power-saving techniques like clock-gating, that act only on dynamic consumption are no longer sufficient. More heavy-weight techniques like DVFS and power-gating are necessary to control the consumption of the chip. These mechanisms use sensors for non-functional properties (temperature, voltage, ...) to control the chip's clock generators and power supply [37, 3]. The power management policy itself is usually implemented in software.

Developing such power management policies requires an execution platform, including models of sensors, actuators, and taking into account the feedback loop between actuators and sensors. Low-level (RTL, gate-level or even SPICE) simulations are possible, but too late in the design flow and their simulation is not fast enough for system-level simulation including non-trivial software. On the other hand, an accurate model of power consumption and temperature requires a realistic model of timing.

Virtual prototyping of Systems-on-a-Chip is a well-established practice. The standard technology for fast and early functional simulation is SystemC/TLM [1], implemented as a C++ library. Several abstraction levels are possible: one can model the timing-behavior precisely, requiring a relatively detailed model of the micro-architecture. Early in the design-flow, more abstract simulations that completely abstract the micro-architecture and with a very loose notion of timing are more applicable.

Estimating power-consumption and temperature on early virtual prototypes is needed to allow early power-aware software development and optimize the power-consumption at the system-level. A solution for system-level power and temperature analysis must satisfy several criteria: it must be fast, to allow simulating non-trivial scenarios on the whole hardware+software system. It must be applicable early in the design flow, hence should require a low development effort, and it must be applicable on early, hence abstract, virtual prototypes.

This paper presents several solutions satisfying these three criteria to augment a functional SystemC/TLM model with non-functional information, and to perform power and thermal analysis using a dedicated solver. The tool LIBTLMPWT [25] uses a tight coupling between the functional simulation and the non-functional solver, and exploits this tight coupling for performance optimizations (`pwt` stands for *PoWer and Temperature*). We also developed an approach that allows using an external solver as black box and in a separate process [10], and showed how the cosimulation approach could be used with loosely-timed models [9], validated by a case-study [16]. The present paper gives a global but detailed presentation of these different approaches. It includes both improvements in presentation of previously published content and unpublished results.

We focus on the *cosimulation* of an existing thermal model (Hotspot [28], ATMI [38] or Aceplorer [31]) with a functional SystemC/TLM simulation. The internals of the thermal model are out of the scope of this paper: we study the coupling of simulators, and the impact of the modeling style in SystemC/TLM on the accuracy of the result.

More specifically, the contributions of this paper are:

- A detailed presentation of the tool LIBTLMPWT. Although the tool itself was published with a technical report [25], it was never presented in a peer-reviewed publication. The tool provides a new method for power instrumentation with low performance overhead. It take into account standard SystemC/TLM performance optimizations like temporal decoupling and direct memory interface (DMI). The annotation mechanism is based on *activity ratios*, and takes into account the frequency changes automatically. It is available as free software [27], including the instrumentation API, the cosimulation engine, and a graphical user interface.
- An analysis of different levels of abstraction, and their consequences on possible cosimulation techniques, with a comprehensive state of the art.

- A method to model abortion following an interrupt in loosely-timed systems.
- New experimental studies, including an analysis of the development cost.

2 Modeling Power Management Policies

2.1 Power Management in Modern System-on-Chips

Power consumption of a silicon chip comes mainly from two sources:

dynamic power comes from switching activity. One can see transistors and wires as small capacitors that are filled-in and emptied when the corresponding value switches from 0 to 1 or 1 to 0. This consumption therefore depends on the capacitance of physical elements (hence, of the physical process), and on the frequency at which values change, i.e. how much computation is being performed by the component.

static power comes from leakage current. For a given hardware component, it depends on the state of the power supply (on or off, and voltage), but is independent of the computation being performed. The leakage current tends to increase when the size of transistors is reduced. Since heating silicon increases its conductivity, the static power also increases with temperature.

A simple way to reduce dynamic power consumption is *clock gating*, which stops the switching activity by stopping the clock. Clock gating can be applied automatically by the synthesizer in many cases, and the problem has been studied extensively for decades (e.g. [2, 6]). Clock-gating, however, is not sufficient in modern Systems-on-Chip, where static power is non-negligible, if not dominant.

To reduce static power, heavier techniques have to be used. *Dynamic Voltage and Frequency Scaling (DVFS)* allows reducing the voltage, hence both static and dynamic power, but requires a lower frequency. Switching from a voltage/frequency couple to another takes time, hence the policy behind DVFS is non-trivial, and is usually implemented in software (for example, the `cpufreq` drivers in Linux count for around 20,000 lines of code). Another technique is *power gating*, which consists in disabling the power supply of unused components. Power gating also takes time, and the transitions from a mode to another may take time and consume a lot of energy (typically, before shutting down a component completely, a part of the internal state may have to be saved to some retention registers), hence the policy to chose whether to power-gate a component or not is again non-trivial. In the sequel, we call *power controller* the set of hardware components that directly control these physical mechanisms, and *power manager* the implementation of the policy, usually done at least partially in software. Bugs related to power management, or *power bugs*, can drastically increase the power consumption of a system, and discharge the battery of a mobile phone in a couple of hours [40]. Worse, serious bugs in the power management policy can lead to deadlocks, e.g., waiting for interrupts from a component which has been completely switched off.

The need for low-power is obvious for battery-powered devices, but it is also important for permanently plugged ones like set-top-boxes: in addition to being environment-friendly, a low-power system needs cheaper packaging and avoids the need for a noisy cooling system. Power consumption cannot be evaluated precisely independently from temperature, as hotter silicon consumes more (and consuming more in turn increases temperature).

Temperature peaks can physically damage the chip. Since high temperature increases silicon conductivity, hence static power consumption, a temperature peak can lead to a consumption peak, and therefore a voltage drop. If the voltage goes below the appropriate threshold, the system may stop working. It is therefore critical to have thermal estimations before manufacturing the real system to dimension the packaging and cooling system appropriately.

Also, high temperature gradients lead to faster aging of the chip [14]. To avoid peaks and minimize gradients, the chips are usually equipped with several temperature sensors (typically

one or more per core on a many-core platform [37, 3]), allowing the implementation of a software control loop to regulate temperature.

2.2 Virtual Prototyping for Power Aware Systems

2.2.1 Non-functional Models

A chip's non-functional aspects can be modeled by specifying the power consumption of individual components, the layout of components (called the *floorplan*), and the physical parameters influencing heat dissipation (from a component to another, and from the chip to its environment).

For system-level simulations, modeling individual transistors precisely is clearly too low-level and too slow. The physical parameters have to be abstracted into a set of operating modes for each component, often called *power-state* [5, 7]. A power-state defines the power consumption (in Watts, or the current intensity in Amperes), possibly as a function of temperature. The energy consumption (in Joules, or the charge transported in Coulombs) is obtained by integrating the power over time. When the power consumption of a power-state is constant, the integration is simply a multiplication by a duration. A power-state is defined by several parameters:

Fixed parameters depend on the platform, but do not vary at runtime (we do not use the word "static" to avoid confusion with "static power consumption"):

- *floorplan parameters:* location of the module on the chip; used to compute the area of the module, and the neighbor relations.
- *technology dependent parameters:* physical values that depend on the technology used to produce the physical chip; this includes capacitance, leakage current, influence of temperature on the leakage, etc. Most of these values are common to all modules.

Runtime parameters depend on the platform's activity and configuration:

- *Electrical state* is the voltage and frequency of the component. It is influenced by DVFS and power-gating, and controlled by the power-controller.
- *Activity* defines the computation performed by the component (e.g., waiting, computing, ...). The total power consumption related to activity can be expressed as $K \times F \times V^2 \times \alpha$, where K is a fixed constant (depending on the number of gates and the capacitance of each gate) taken into account in the *technology dependent parameters*, F is the frequency, V the voltage, and α is the *activity ratio*. The activity ratio expresses the proportion of gates involved. It varies from 0 (no activity) to 1 (all gates are active at each cycle).
- *Traffic* is the amount of data processed by unit of time (e.g. number of transactions routed for a bus, number of reads and writes for a memory, ...).

Note that different parameters correspond to different kinds of models. Electrical state fit very well in the power-state model: one only needs to model transitions from state to state. Activity can be modeled in several ways: either an operating mode is modeled as a power-state with a given activity ratio independently from the functionality, or the activity of the functional model is observed during simulation and used in the non-functional model. Traffic has to be modeled separately, since the traffic of a component usually comes from the activity of another component. We cannot define power-states for traffic a priori and need to take into account the traffic of the functional simulation. The solutions presented in this paper take all these parameters into account.

Depending on the stage in the design-flow, the power consumption associated with a power-state can come from different sources. At early stages of development, they can be target values (i.e., that further development will consider as an objective), or extrapolation from previous

```
// SystemC thread
void compute() {
    while (true) {
        set_activity(0); // Enter IDLE mode
        wait(event);
        set_activity(ACTIVITY_RATIO_RUN); // enter RUN mode
        f();
        wait(100, SC_US);
        send_irq();
    }
}
```

Figure 1 Simple Power-model of a Hardware Component (pseudo-code).

generations of the same component. Later in the design flow, they can be more precise estimations based on RTL or gate-level simulations, and physical measurements when a prototype of the chip is available. Our contribution is not to provide these parameters to the user, but to allow exploiting these per-component parameters for a *system-level* simulation including power and thermal management policy and software. Because of heat dissipation, and of the relationship between power and temperature, the composition of the component's parameters is non-trivial.

Tools like *Docea Power*'s *Aceplorer* [31] allow modeling a chip, taking into account both the power consumption and the thermal aspects. The traditional way to use Aceplorer for simulation is to define scenarios that define the sequence of modes for each component. The tool computes the power consumption and the evolution of temperature (including the feedback of temperature on power described above). Scenarios can be provided by hand, using UML's activity diagrams, or can be produced by a SystemC/TLM or RTL simulation.

In the scenario-based usage, the model does not allow the execution of power-aware software: the SystemC or RTL simulation provides non-functional stimuli (typically dumped in a VCD file for offline power/thermal analysis), but has no access to the result of the power and thermal simulation. A temperature sensor, or battery monitor component could not be modeled. Our contribution is to allow such closed-loop simulation.

2.2.2 Functional and Non-functional Models cosimulation

To illustrate the cosimulation concepts, let us first consider a very simple component, with 2 activity states. The component waits for an event in `IDLE` mode, and on reception of the event performs a computation `f` that lasts 100 microseconds in mode `RUN`. A SystemC thread describing this behavior is provided in Figure 1.

The principle of cosimulation is illustrated in Figure 2. The instrumented functional model computes, at each point in time, the power state of each component. This power state information is transmitted (a) to the power model, which uses this power state information together with the temperature T obtained from the thermal solver (c) to compute the power consumption of each component. The power consumption is used by the thermal solver (b) solver to compute the derivative of the temperature. It is used together with the thermal model to compute the evolution of temperature. The temperature (and possibly other non-functional information) are transmitted back to the functional part and embedded software (d) through models of physical sensors.

Note that the scheme illustrated in Figure 2 contains two loops. We focus on the Functional/Non-functional loop ((a) and (d) on the figure: the behavior is influenced by temperature sensors, but

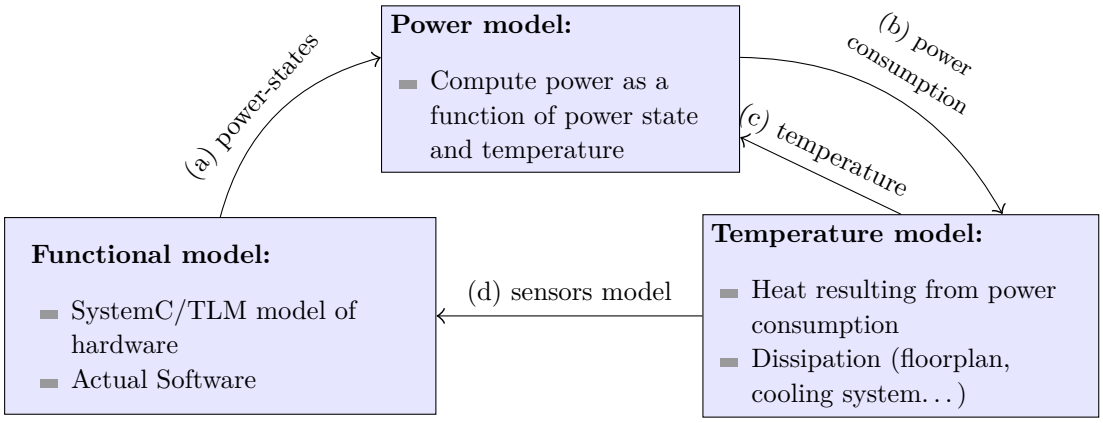

Figure 2 Functional, Power and Thermal Models.

the temperature depends on the behavior). A second loop appears with Power/Temperature ((b) and (c) on the figure: temperature increases as a consequence of power consumption, but static power consumption increases when the temperature increases). This second loop is also managed by our approach: when cosimulating with a black-box power/thermal solver like Aceplorer, the solver deals with it internally. In LIBTLMPWT, the computation of power density can use the temperature (it will actually use the temperature computed at the previous ATMI step).

2.2.3 Power-Aware Software Execution on a Virtual Prototype

To validate these power and thermal managers early in the design flow, one needs virtual prototypes that allow execution of power-aware software. Among these models, various degrees of precision can be achieved:

Purely functional: Even purely functional prototypes need models of temperature sensors: if the embedded software reads a value from one of its registers, then the simulated platform should include a component mapped at this address and returning a sensible value (possibly an arbitrary constant).

Non-functional contracts: Some basic, but yet serious mistakes like reading from or writing to a component which is switched off can be caught by assertions in the model. The assertions form the contract [35] of the components, or of the hardware platform with respect to software.

Scenario-based models: To test some basic power management policies, one may need the non-functional inputs to take different values. For example, if a platform triggers an emergency stop when temperature goes above some threshold, then testing this functionality requires a scenario where the temperature returned by the sensor crosses this threshold. This can be done by returning a pre-defined sequence of values in the temperature sensor model.

Approximate models: When the policy to be tested is non-trivial, manually writing scenarios is not feasible anymore. Not only the scenarios are too complex to be written by hand, but realistic scenarios cannot be generated offline [41]. One needs an automated way to get reasonable sequences of values. A simple thermal model can be sufficient: it will typically let the temperature decrease when the system saves energy and vice versa. For example, a software developer implementing a simple hysteresis policy (switch to power saving mode when temperature is too high, and switch back to normal mode when the temperature crosses a low threshold) can use this model to test that the mode switches are correctly performed. These

models allow detecting some of the non-functional bugs in the embedded software (failure to enter a low-power mode, polling instead of explicit wait for an interrupt, ...).

Precise models: To validate the parameters of a power management policy and get the actual values for maximal temperature peaks and gradient, one needs a precise model. This implies precision in the timing of the platform, and on the thermal model of the chip. Some degree of precision is also needed to compare the efficiency of several power management policies.

We are interested in the last two. The following sections describe new tools to allow power and thermal modeling on SystemC/TLM at different levels of timing granularity. In both cases, the approach enriches a functional model with some power-state information, and cosimulates it with a dedicated power and thermal solver.

The rest of the paper is organized as follows. We review related work in Section 2.3, and then present our tools. Section 3 presents a cosimulation method implemented in LIBTLMPWT, where the thermal solver is embedded in the simulation. Section 4 presents our techniques to deal with loosely timed models without introducing simulation artifacts. In Section 5, we present a prototype distinct from LIBTLMPWT based on similar ideas, that allows cosimulating a SystemC simulation with an external power and temperature solver. We present our experimental results in Section 6, and conclude in Section 7.

2.3 Related Work

2.3.1 Power-state Model

The power-state model we use has already been applied to SystemC in the TLM Power library presented in [34, 55, 23]. TLM Power models a system-on-a-chip in SystemC/TLM, running the actual software on top of a simulated hardware. The objective is to validate a power management policy. We borrowed some ideas from TLM Power, but the latter does not allow temperature management. Another approach using the power-state model on SystemC programs is presented in [24], with advanced techniques for software integration like source-level simulation with back-annotations. We extended the idea to support cosimulation with a thermal solver, including closed-loop cosimulation where the software has access to non-functional values through sensors. Also, [34, 55, 23, 24] target precisely timed models while we allow an analysis on temporally decoupled or loosely timed models.

The power-state model is also used in [18] where the authors perform a thermal analysis to evaluate and optimize the mean time to failure of a chip. [18] uses an abstraction of the software as a task graph, unlike our approach which uses the concrete, actual software (i.e. a cross-compiled binary executable).

The power-state model can also be used at a more abstract level, where the hardware and the software are modeled with automata. Indeed, if the set of power-states is finite, then the underlying formal model is the one of *linear-priced timed automata* [4]. On a simple enough model, formal analysis is possible. For example, [19] formulates the thermal analysis problem as a hybrid automata reachability verification problem, and solves it using model checking. [32] defines a system-level *analytical* model to capture the consumption and thermal behavior of a chip with *Power Variability Curves*, based on the framework of real-time calculus. These approaches cannot be applied on a model precise enough to execute the actual software. Instead, the software is modeled as a set of tasks, and the hardware architecture is not detailed. Power consumption is considered to be a function of the executing software task, hence only processors are considered. [32] computes guaranteed bounds on temperature peaks. The method is intended to be used at a very early stage of the design. As opposed to this, we use SystemC/TLM models that *model* the

hardware, but *execute* the actual embedded software (using either instruction set simulators or source-level simulation [44]).

A concrete study of power-state models in Synopsys Platform Architect and Aceplorer is given in [22]. Unlike the present paper, [22] focuses on the power/thermal analysis and does not consider the cosimulation with a functional simulator nor software integration.

2.3.2 Low-Level Power Analysis

Our approach uses the power-state model, which assumes that the non-functional characteristics like power-consumption are given for each component and each power-state. Other techniques, or physical measurements, have to be used to find these values for each component. The power-state model is used to compose the results at the system-level. The following techniques are lower-level analysis techniques that can be used to calibrate a high-level power-state model.

[39] presents an analytical approach to dynamic power estimation of RTL descriptions. The authors use the concept of entropy (from information theory) to estimate the average activity factor. They consider only the combinatorial parts of a circuit.

The work described in [52] is a simulation method including the functionality and power consumption. It focuses on instruction-level power consumption for software execution, describing in details how to get the parameters of the power-model, with measures. It is limited to power-consumption and does not take temperature into account. A case-study from Intel is described, and the simulation results compared to measures on the real chip. The analysis is based on Hamming distance, and this information is not available in most TLM models because they abstract away details about transaction interleaving, signals switching and arbitration.

[13] provide a methodology for power characterization of the AMBA AHB communication Bus. The resulting power characterization is based on monitoring some parameters of the functional simulation. Accuracy of power estimation is gained at the expense of the functional model abstraction level, and therefore at the expense of the simulation speed. Like for [52], it relies on information that may not be available in TLM.

A technique for instruction level power modeling of processors is described in [51]. The intuitive idea is to run repeatedly one instruction in a loop and measure the average power. The work takes into account inter-instruction effect (switching from an instruction to another). Analysis of assembly code can then associate power values with each basic block of the program. The modeling approach targets software power optimization trough power-aware compilation. A similar approach is described in [42].

[12] describes a methodology for deriving instruction-based dynamic power models from gate-level simulations. The derived power models (in the form of look-up tables) are meant to be used in system-level simulation models to allow for faster power simulation.

Industrial tools like Synopsys's PrimeTime PX [49] allow a very detailed power analysis, but these tools take as input the netlist of the circuit, which is not available at early stages of the design-flow. They perform a detailed analysis hence run orders of magnitude slower than a TLM model.

Several approaches are dedicated to the definition of power-state machines of individual IPs, from data collected with low-level simulations. In some approaches, the structure of the machine has to be known in advance, and the analysis of low-level simulations provides the consumption values to be attached to the predetermined states. [17] proposes to run functional simulations of an IP together with gate-level simulations, in order to synthesize both the structure of the power state machine, and the consumption values attached to the states.

2.3.3 Thermal Analysis

Several thermal solvers are available. Hotspot [28] can be used either as a standalone tool or as a library. It takes as input the physical characteristics of the chip and a power-trace (power consumption of each component as a function of time), and computes a thermal trace. ATMI [38] follows a similar approach, with a focus on simplicity of integration. It is available only as a library. 3D-ICE [48] uses a similar model dedicated to 3D stacked chips. These tools do only thermal analysis: in Figure 2, they can be used for step "Temperature model", but need an additional cosimulation engine to perform the complete simulation.

CTherm [33] proposes a cosimulation including functional (SystemC), power and thermal (3D-ICE) models similar to ours (the thermal solver is triggered periodically from SystemC at a user-defined pace). It does not allow Direct Memory Interface (DMI), temporal decoupling or loose timing. However, it has some features like thermal checkpointing and automatic thermal model generation that would be interesting to add to our tools.

Aceplorer [31] is an industrial tool which, coupled with AceThermalModeler, does both power and thermal analysis (i.e. does both "Power model" and "Temperature model" in Figure 2). Initially, it could get some power-state traces from an external SystemC/TLM model, but not feed non-functional values back to the model. It did take into account the power/temperature loop (arrows (b) and (c) on Figure 2), but did not allow a feedback loop from the non-functional side to the functional model (arrow (d)). The AceTLMConnect [43] extension was added to the tool based on our research prototype (as part of the joint project HeLP[1]).

These thermal solvers solve several problems. They can perform a *steady-state* analysis (i.e., compute the state of the system after a long period of time with constant load), or *transient* analysis (i.e., model the response of a system to a change). The steady-state analysis is useful to analyze the overall characteristics of a chip, but is not sufficient to execute a software power management policy, hence is not sufficient to solve the problem addressed in this paper. A steady-state analysis does not need the detailed behavior hence it does not need our cosimulation technique. We use the *transient* analysis of these tools, and feed it with information produced by the functional simulation.

2.3.4 Standards and Formats

Some common languages/formats (e.g., UPF [29] and CPF [45]) for low power design are emerging and allow to describe power intents (power domains, power network, power switches, etc.) of a circuit. These formats can be the entry point of a code generation flow [36] that avoids hardcoding non-functional values into the SystemC/TLM code, and could use a cosimulation like ours to run the simulation.

2.3.5 System-Level Power and Thermal Simulation

A widely used tool for system-level simulation is gem5 [8]. The principles behind gem5 and SystemC/TLM are very similar: both are event driven simulators, and gem5 can actually cosimulate with SystemC. gem5 is meant to be usable out of the box, and includes several instruction set simulators and memory models, while SystemC only provides the building blocks to write such simulators. gem5 can be configured with different speed Vs accuracy trade-offs, but even the speed-oriented configurations are more precise than the loosely-timed models we are targeting in this paper. For example, gem5 does not support temporal decoupling.

[1] ANR Arpège, 2009-2013, see `http://www-verimag.imag.fr/PROJECTS/SYNCHRONE/HELP/`

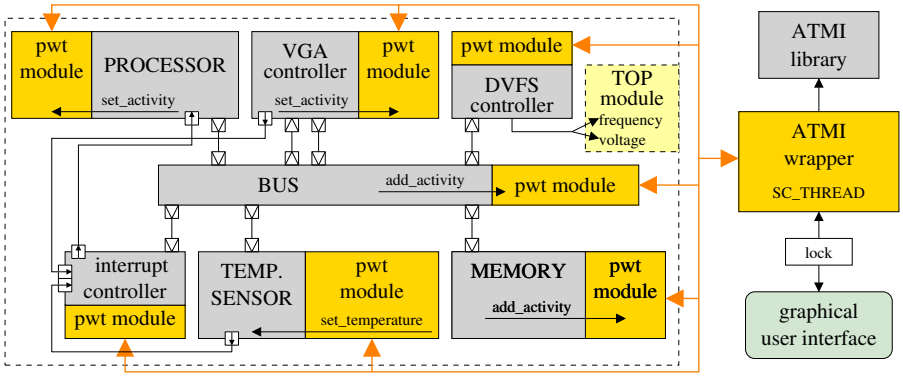

Figure 3 Example of a minimal SoC model with its power and temperature extensions.

gem5 can be augmented with a power model and cosimulate with a thermal solver like Hotspot as done in [50]. Similarly to the present paper, [50] allows executing software that interact with power-related sensors and actuators on the chip. However, the abstraction level is very different. Our techniques allow for common optimizations on loosely-timed models: coarse-grained timing with temporal decoupling (see Section 4.2), direct memory interface (see Section 3.3), including traffic models. Also, to the best of our knowledge, gem5 is not able to cosimulate with an external, black-box, power and thermal solver (see Section 5).

3 Standalone Power and Temperature Modeling

3.1 Architecture Overview

We now present the approach followed by LIBTLMPWT, in which all power and temperature computations are integrated into the SystemC/TLM model. As shown on Figure 3, each module contains code to estimate its power consumption, then a centralized temperature solver gathers power information and evaluates the temperature of each module. An optional graphical user interface (GUI) allows monitoring and controlling the simulation.

Temperature evaluation is done using the ATMI tool [38]. Basically, ATMI takes as input a floorplan, i.e. a list of areas described by their coordinates, and the initial temperature. During simulation, ATMI computes the temperature of each area based on its power consumption, expressed as a power density. This computation is done at a regular pace, such as once every millisecond (SystemC time). The temperature computation must be centralized, because the temperature of any area depends on the temperature of neighbor areas.

Since ATMI is packaged as a C library, it can be directly called from SystemC code. This is done by a SystemC module, called the ATMI wrapper. At elaboration time, we register to this module each SystemC module that is mapped to an ATMI area. This module contains a SystemC thread that calls the ATMI library according to the ATMI pace during simulation.

The ATMI wrapper algorithm is schemed by Figure 4. Given the power densities, computing the module temperatures is just a function call to the ATMI library. The resulting temperature is propagated to the SystemC side by the wrapper after the call to ATMI. For any SystemC module, it is possible to provide a callback method that is called each time the temperature is set. For example, the temperature sensor module defines a callback method that raises an interrupt when the temperature reaches some thresholds (for a concrete example of sensor providing this functionality, see the High/Low-Temperature Interrupt Enable bits of the `IA32_THERM_INTERRUPT` Register on Intel Architecture [30]). The main issue is to estimate the average power density consumed during the last step elapsed. This estimation is done in the new `pwt_module` class.

```
while (true) {
    wait(atmi_step_duration); // SystemC's wait
    for each module, compute its average power density during the last elapsed step.
    atmi_simulator_step(atmi_instance, power_densities) // call ATMI
    set module temperatures and call associated callbacks to trigger non-functional events
}
```

Figure 4 Basic algorithm of the ATMI wrapper.

In our model, the temperature transmitted to the SystemC modules, in particular to models of the thermal sensors, is the temperature at the corresponding instant. If the actual sensor exposes only a sampling of the temperature (e.g. the physical sensor samples temperature only every second), then this sampling can be implemented in SystemC by the temperature sensor. Letting the SystemC module chose the sampling rate makes the approach very flexible.

Each SystemC module that is mapped to an ATMI area must inherit from the `pwt_module` class. This class stores several parameters, categorized following the classification given in Section 2.2.1 (*fixed* parameters: floorplan and technology dependent parameters; *runtime* parameters: voltage, frequency, activity ratio).

In order to define frequency and power domains, we allow the existence of PWT modules not mapped on the floorplan. Such modules only provide the voltage and frequency parameters, which are forwarded to their children modules. Other methods are disabled; in particular, they have no power densities. For example, the chip on Figure 3 has one DVFS (*Dynamic Voltage and Frequency Scaling*) controller and a single power domain. So, in TLM, the model of the DVFS controller is bound to the top module; the DVFS controller TLM module calls the methods `set_frequency` and `set_voltage` of the top module, which in turn calls the `set_frequency` and `set_voltage` methods of all its children PWT modules.

Given these parameters, the `pwt_module` class computes the power density, or more precisely, its average value during the last step elapsed. The power consumption is computed as the sum of the static and the dynamic power:

- The *static* power is due to the leakage current, and it is proportional to the voltage and the leakage current intensity. The intensity itself increases when the temperature increases; in the current implementation, we use a linear approximation of the temperature effect, but a more elaborated physical model would be easy to integrate in the tool.
- The *dynamic* power corresponds to the cost of voltage changes in gates. It is proportional to the frequency, to the number of gates involved (i.e., the activity ratio), and to the square of the voltage. Moreover, it is proportional to a constant that depends on the capacitance per gate and on the gate density.

In consequence, the general formula is of the form:

$$P = P_{static} \quad + \quad P_{dynamic} = V \times K_1 \times (1 + K_2 \times T) \quad + \quad F \times V^2 \times \alpha \times K_3,$$

where V is the voltage, T is the temperature, F is the frequency, α is the activity ratio, and K_i are static parameters depending on the module area and on the synthesis technology. Because PWT modules contain the general formula, which involves explicitly the frequency and the voltage, the power model manages DVFS by construction. Indeed, other approaches [34] let model developers provide the actual power value, using their own function that may or may not take into account that the voltage or the frequency may change. If needed, a module that has a specific power model can also redefine the method that computes its power density and use an arbitrary formula written in C++.

3.2 Setting the Activity Ratio

Using the approach we present in this section, the main task to extend a TLM model with power consumption and temperature estimations is to set dynamically the activity ratio of each module. The `pwt_module` class provides two ways to set this ratio (`sc_time_stamp()` is the SystemC function returning the current simulated time):

1. *level-based: change the activity level, until the next call*
   `void set_activity(float ratio, sc_time now = sc_time_stamp())`
   Set the activity ratio starting from *now* and until another level is set.
2. *action-based: add some extra-activity for a fixed duration*
   ```
   void add_activity(float ratio_increment,
                     unsigned nb_cycles,
                     sc_time now = sc_time_stamp())
   ```
   Add some activity to the current level (more precisely, the level at date *now*), for a short duration defined by a number of clock cycles.

In general, the first method is best suited for initiator modules whereas the second is better for interconnects and target modules. For example, a processor TLM model will call the `set_activity` method when it enters an *idle* state and when it becomes *busy* again. Using this method, we get an activity-state based power model, as in [34]. If more accuracy is needed, one can develop an instruction-based power model (like the one of [20]) on top of this API. The model would use the second method and call `add_activity` for each instruction, with a ratio depending on the instruction kind and the register values. Obviously, the second approach requires additional manpower and will slow down the simulation.

Concerning interconnect and target modules (that receive transaction initiated by others, like a memory component), the best solution is to call the `add_activity` method once per transaction. In general, the *ratio increment* depends on the command (`READ` or `WRITE`) whereas the duration (`nb_cycles`) depends on the transaction size. Note that it would be harder to use the activity-state based method, because there is no local activity state (indeed, the activity depends on the external initiators).

A naive implementation of the `add_activity` method would be to use two calls of the `set_activity` method, as follow: remember the current activity level as `current_ratio`, set the activity level to `current_ratio + ratio_incr`, increase the local date, and finally re-set the activity to `current_ratio`. However, this implementation would not be reentrant, and pretty slow. Reentrancy is mandatory for a bus or memory module: they can receive transactions from several initiators whose non-functional effect overlap. On the contrary, our implementation of the `add_activity` method is reentrant and fast.

The rationale for the parameter "`sc_time now`" of methods `set_activity` and `add_activity` is to ensure the compatibility with temporal decoupling [53], when using the coding rules defined in [1]. When temporally decoupled, the local date of a process is expressed as "`sc_time_stamp() + local_offset`" (instead of "`sc_time_stamp()`"), allowing to advance the local time by executing a low-cost "`local_offset += T`" instead of a costly "`wait(T)`". This local offset is part of all transactions, so it can be used by interconnects and target modules too. Consequently, to set the activity ratio at the right date, the methods `set_activity()` and `add_activity` must be called with the parameter `now` set to "`sc_time_stamp() + local_offset`". Since this parameter has a default value, it can be safely ignored for all processes that are not temporally decoupled.

The "`now`" parameter passed to the activity methods may be further in time than the next ATMI step boundary. Indeed, some TLM modules modeled at a coarse-grained may simulate a slice of time much longer than the ATMI time step before yielding back to the scheduler.

Consequently, each PWT module contains a list of activity counters. The list head contains the activity counter of the current ATMI step, and successive list elements store the activity of future steps. The ATMI wrapper pops the front element once every step. The traffic model of [9] can be implemented on top of this, but it is currently not part of the tool presented here.

3.3 Direct Memory Interface (DMI) Management

Motivated by simulation speed issues, some TLM modules use a technique called *Direct Memory Interface* (DMI). The goal is to accelerate memory accesses. and the idea is to provide the initiator (e.g., an ISS) with a pointer to the memory array. So, when accessing memory, the initiator will directly use the memory pointer instead of generating a transaction. Given that a transaction involves many indirect function calls plus routing in the interconnects, the speed gain is significant. DMI is functionally correct but may bypass some side-effects, because the code related to timing and power into the interconnects and the memory is no longer executed. For the timing issue, the SystemC standard [1] suggests to provide the initiator with two durations: the read latency and the write latency. Thus, the initiator can add the latency to its local offset when a memory access is simulated. Because the latencies depend on the frequency, the *DMI descriptor* must be updated every time the frequency is changed.

We use the same idea for power modeling. However, providing a single activity ratio increment per transaction is not enough, since the activity increment must be added to each module involved in the transaction. Indeed, if the additional activity was assigned to the initiator, then the initiator temperature would be overestimated whereas the target components (e.g. bus and memory) temperatures would be underestimated. Our solution is to add into the DMI descriptor a pointer list of all PWT modules that are on the transaction path. Our DMI manager class provides a method `apply_side_effects(command, size)` that increases the local time offset according to the latency and call the `add_activity` method of all the PWT modules in the pointer list.

3.4 Graphical User Interface

We have developed a graphical user interface, providing basic simulations controls and some monitoring features (see Figure 5). This GUI is implemented using the Qt framework, and run in a POSIX thread distinct from the SystemC simulation. So, the SystemC simulation is slowed down only when the GUI takes the main lock to update its values. Those value updates are done ten times per second (wall-clock time).

On the control side, the GUI allows to pause the simulation (done by keeping the lock), or to reduce the SystemC simulation speed (done by adding Unix `usleep` calls in the Posix thread running the SystemC kernel).

For monitoring, the GUI provides the current temperatures as text values, a graph of the floorplan where each module is coloured with respect to its temperature (from blue for cold, to red for hot module). Additionally, a plot provides either the temperature or the power consumption or the power density of each module depending on the time.

4 Thermal Analysis for Loosely Timed Models

4.1 Loosely Timed Models

Loose timing requires special attention. An obvious limitation of any power-state based model is that if the timing is too imprecise, then a precise power analysis is not possible (as the power-state model intrinsically needs timing to integrate the power values over time). Still, the timing can be loose and be a reasonable approximation of reality.

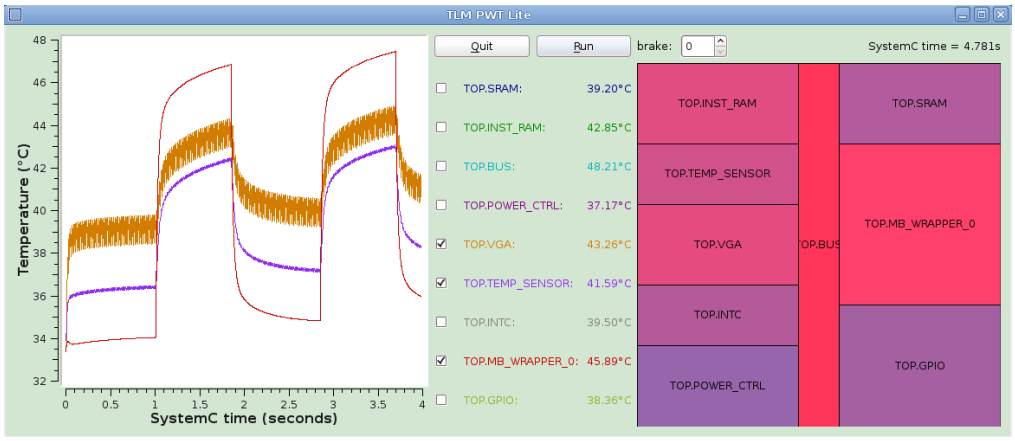

Figure 5 Graphical user interface.

In fine-grained models, the duration of a task execution is usually *computed*. For example, an ISS is a way of computing a software task duration: the real binary (and thus the real algorithm) is simulated instruction by instruction and memory access per memory access; the duration of the task is then a sum of short durations corresponding to each instruction execution and memory access. These short durations are defined in many modules: instruction cost in the ISS module and memory access latencies and bandwidths in interconnect and target modules.

As opposed to this, loosely timed models can abstract durations completely and consider that computations are done in zero-time, or use *calibrated* durations. This happens for example when the algorithm used by the future hardware is still unknown, or when the simulated platform uses an algorithm that is different from the hardware. For an image processing application, for example, a coarse-grained model that considers an image as atomic can be calibrated with some timing values close to the actual ones (e.g. considering that decoding one image takes X ms). This estimated duration may possibly depend on some parameters, such as the image size.

Calibrated models cannot be very precise, as the actual performance can depend on interactions that cannot be taken into account in calibration (e.g. conflicts on a shared bus). However, using calibrated models is not less precise than using hand-written scenarios, which is a common practice for power/thermal modeling in the industry. This section presents a set of techniques to write approximate power and thermal models based on loosely timed functional models. We obviously cannot recover information which is abstracted away by loose timing, but we avoid simulation artifacts that would result from it.

In this section, we consider the case of coarse-grained modules; that is to say, a module which contains abstract tasks whose duration is declared instead of computed. A typical example is an hardware accelerator for graphical computation, such as image encoding or decoding. Such TLM module models the functionality using an algorithm (such as the legacy code algorithm) which is not the same as one used in the hardware.

The power consumption of a coarse-grained module can be defined easily by a discrete set of activity states: for example one for idle and another for busy. However, because a coarse-grained module performs memory accesses, it has an impact on memory and interconnect consumption that must also be evaluated. Figure 6 shows a typical code for a coarse-grained module task.

More generally, the code is usually written as a sequence of computations, each of them being of the form `compute(); wait(...); commit();` . `compute()` is done in zero simulated time. `wait(...)` lets the simulated time elapse. Its argument is the time the computation would take on the real system. `commit()` is the functional effect of the computation, typically an interrupt or a transaction

```
1  void compute() { // SC_THREAD
2      while (true) {
3
4          wait(start_event); // wait for a new request
5
6          in_data = read_block(...); // read all useful data (e.g., 1 image)
7          out_data = compute_task(in_data); // call legacy code
8          write_block(out_data, ...); // write all new data
9          wait(duration_task); // wait the duration task
10         end_event.notify(); // notify the client that the request is done
11
12     }
13 }
```

Figure 6 Example Code with Loose Timing.

to a control register. `commit()` is also called a *synchronization point*. More details can be found in [15].

With a naive implementation, the `read_block` and `write_block` transactions are done at a single SystemC time, thus creating an infinite peak of power consumption in the interconnect and memory, followed by no additional consumption during `duration_task`. As a consequence, in the model, the chip temperature will climb very fast and immediately fall back as fast. This temperate peak may cross a threshold of the temperature sensors, thus generating an event for the temperature manager. This event is an issue, because on the physical system there is likely no such temperature peak, but instead a slow increase of the temperature spread among the whole task execution.

Avoiding unrealistic peaks is relatively easy on the initiator's side. The solution is to instrument the complete `compute(); wait(...); commit();` block of code. In the example of Figure 6 this would mean adding calls to `set_activity()` at lines 3, 5 and 11.

4.2 Transactions from Coarse-Grained Modules

To illustrate the problems caused by loose timing, Figure 7 shows a possible execution of two iterations of the loop. Vertical arrows represent transactions. The first line shows the transactions that the actual system would perform. The second line shows how these transactions are simulated on a loosely-timed model. In a naive model (Figure 7.(a)), the dynamic energy consumption of the bus routing these transactions would be modeled at the simulated time when the transactions are executed, hence we would get an instantaneous energy consumption at SystemC instants (for clarity, the figure shows cumulative energy instead of power, which would be infinite). The temperature model would therefore compute a peak that does not exist on the real system.

If the model uses a precise temporal decoupling, then the local dates of transactions can be used instead of the global SystemC time. However, on a loosely timed system, this local clock only provides a lower bound of the transaction start: the actual time of the transaction may be at any time between the previous timing annotation and the next one, which is not known yet. In the example of Figure 6, the TLM model has a clear read/compute/write separation, but the actual system may use a pipelined algorithm where reads, computations and writes are interleaved. This cannot be reflected by only adding annotations to the code. The best we can do is to assume that transactions will be evenly distributed on the interval of time where they happen. This is illustrated on Figure 7.(b): the analysis counts transactions over each interval of time, and when

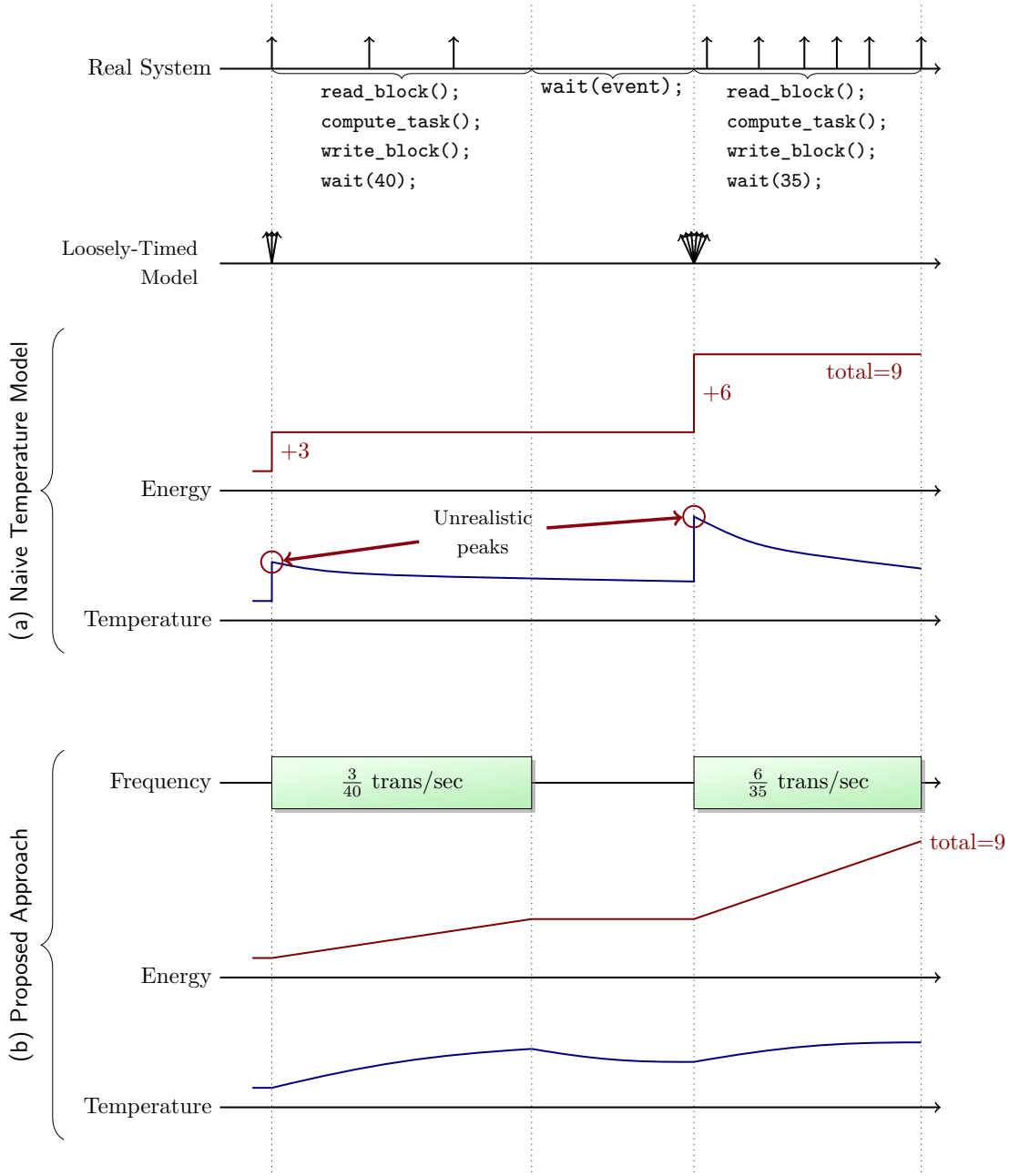

Figure 7 Elimination of Simulation Artifacts.

reaching a synchronization point, a frequency is computed, and this frequency is used instead of individual transactions in the analysis.

We presented a first implementation of this principle in [9]. We describe below our new implementation of this principle integrated in LIBTLMPWT, with an accurate management of classical optimizations like DMI.

Firstly, interconnect and targets modules (mainly memory modules) must be able to recognize such coarse-grained transactions. For this goal, we have defined a new extension type (following the ASI TLM guidelines for ignorable extensions). When a module receives a transaction with this extension, its behavior is changed as follow:

- the code to add a delay to the initiator local date is skipped
- the extra-activity due to transaction is stored until the end of access date is known.
- the current module is registered to a *delayed-access manager*.

Secondly, when the task is done and its execution duration is known, the coarse grain module informs the delayed-access manger, which in turn calls back each module currently storing delayed accesses. Actually, this is done by adding a line `DelayedActivityManager::commit(task_duration)` just before the statement `wait(duration_task)`. When interconnect and target modules are called back, they simply spread the stored activity between the start of access and end of access date.

Additionally, the new extension for coarse-grained transactions allows to define a secondary transaction size. The idea is that because the algorithm is not the real one, the access size in the physical chip may be different from the one of the TLM platform. For a simple example, consider a hardware module computing the next step of the Game of Life. In TLM, the implementation will read then write the whole image. However, a hardware implementation will have to read some pixel many times if it cannot buffer the whole image; the most naive implementation would even read each pixel 9 times. On the write side, if the image is modified in place, then the hardware may decide to write only pixels that change. Thanks to the secondary size provided with the extension, the extra-activity can be computed on a better estimated size without changing the functional model.

4.3 Interrupt Management

As explained in Section 4.1 and [15], a loosely-timed model can still be functionally faithful in the presence of interrupts. For example, in Figure 6, in the execution depicted on Figure 7, if an interrupt is received by the component executing `compute` at time 20, then in the real system, the interrupt may abort any of `read_block()`, `compute_task()` or `write_block()`. In the loosely-timed model, these 3 functions are executed in zero-time. The interrupt is received during the call to `wait(duration_task)`, and a check for pending interrupts is done after this `wait` statement. The interrupt is therefore processed after the `wait` statement terminates. In some sense, it is taken into account later than the real system, but the model is still faithful because the interrupt is received before `end_event.notify()`, hence the functional effect of the computation interrupted is not yet visible to other components. It is therefore acceptable to take the interrupt into account at the next SystemC instant (at time 40 in our case) from the functional point of view.

However, from the non-functional point of view, interrupts in loosely-timed models raise several issues: even though we can manage them in a functionally faithful way, the timing of the SystemC simulation does not match the one of the physical system. Consider a component executing `compute(); wait(50); commit();` starting at time t, to model a computation that takes 50 units of time. Two cases are problematic:

1. If the processor receives an interrupt at time $t + 20$, and executes an Interrupt Service Routine (ISR) `isr()` for 10 units of time at this point. The functional model executes `compute()` and then `isr()`, but we need to model `isr()` as being executed in interval $[t + 20, t + 30]$.
2. If the processor receives an interrupt at time $t + 20$ which aborts the computation, then `compute()` is executed completely in the functional model, and only a part (20/50) of its execution must be taken into account in the non-functional analysis. In other words, we do not need to actually cancel the functional computation because its functional effect is not yet visible, but we must not consider the power consumption of a computation that is not done in the real system.

To solve issue 1, we consider ISR as a special-case of software execution. First, we need to execute the ISR at the right point in time, hence checking for the presence of pending interrupt after the `wait` statement is no longer acceptable. Our approach replaces the call to `wait(time)` between a computation and its functional effect with a `wait(event, time)`, that either completes when enough simulated time has elapsed, or can be interrupted by an event triggered by an interrupt. This way, the interrupt service routine can be executed at the right point it time. The frequency of transactions due to the normal computation is kept unchanged, but is not taken into account during the execution of the ISR.

To solve issue 2, we need to instrument the abortion in the embedded software, and when we encounter a task abortion, we reset the frequency for the processor (technically, this is implemented by instrumenting `longjmp` to write to a magic address that is caught by the SystemC model and transmitted to the power-model). This is implemented in the tool presented in [9] but not yet in LIBTLMPWT.

5 Cosimulation with an External Power and Temperature Solver

5.1 Cosimulation Interface

When using the non-functional solver as a black-box it is no longer possible to execute a SystemC process for each of its internal steps, since we cannot know when the steps should take place. In this case, another cosimulation strategy has to be applied.

We implemented a cosimulation interface, presented in details in [10], that allows running a functional SystemC/TLM platform with power annotations with an external non-functional solver running in a separate process. We did our experiments with wrappers around ATMI [38] and Hotspot [28] as external solvers, but the same interface was used to connect to the industrial tool Aceplorer. This interface is now used by the commercial extension AceTLMConnect [43] for Aceplorer. A small case study using the industrial implementation is presented in [16].

The cosimulation interface uses a simple request/response protocol, using Thrift [46] for interprocess communication. The principle is the following: the simulation starts on the SystemC side, and SystemC's time drives the simulation. From time to time, the SystemC program requests a non-functional simulation on a time interval. The non-functional solver performs the simulation and returns the relevant values at the end of the time interval. In some case, the non-functional simulation may return early, because the temperature crossed a given threshold, and a SystemC event can be generated at the time the threshold is crossed.

5.2 Strategies Using the Interface

The same interface can be used with multiple strategies. In the *lockstep strategy* (Figures 8 and 9), the functional/non-functional synchronization is performed at the end of each SystemC simulated instant. A non-functional simulation is requested for the time interval between the current instant

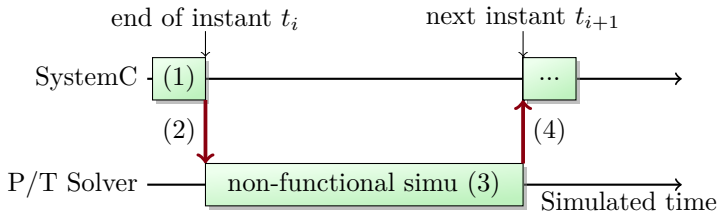

Figure 8 *lockstep* cosimulation strategy (for clarity, simulated instants are represented with a non-null width; boxes on the SystemC line correspond to simulated instants and boxes on the P/T solver line to intervals between instants).

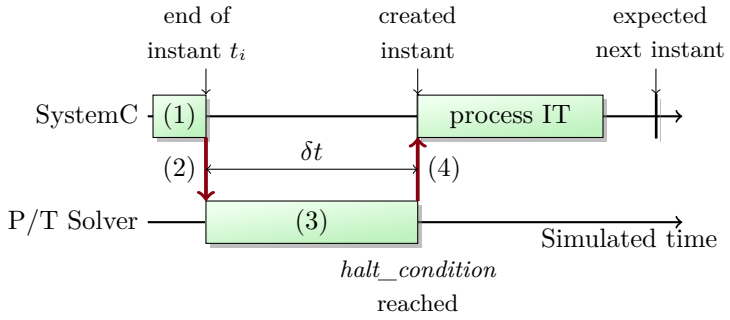

Figure 9 *lockstep* cosimulation strategy with interrupt (IT).

and the expected next simulation instant. In case a non-functional event is triggered (Fig 9), the non-functional simulation stops before the end of the requested time interval, and a SystemC event is notified, which creates a SystemC instant during which the functional part can react (e.g., by triggering an emergency stop if the temperature is too high).

When one can guarantee that no non-functional event is possible (either because the hardware sensors are only passive components and cannot trigger events, or because we know for sure that the condition will never be met), another strategy is possible: the *functional-ahead strategy*, illustrated in Figure 10. In this case, the SystemC side runs without launching the non-functional simulation until an access to a sensor is performed. When this happens, a non-functional simulation is requested up to the current SystemC instant. This strategy is less flexible, but requires considerably less request/response round trips, hence can be faster (especially when SystemC and the non-functional solvers run on different machines).

A *parallel strategy* is also presented in [10].

6 Experiments

6.1 Development Cost

For the development and the evaluation of the LIBTLMPWT approach, we have developed a demonstration platform based on a small FPGA system. The main components are a processor (MicroBlaze) and a VGA controller. There are two memories, one for instructions and the other for data, plus the usual devices: timers, UART, interrupt controller, etc. Compared to the initial FPGA system, we have added in the TLM model a temperature sensor and a DVFS controller. The whole TLM model uses the blocking TLM interfaces of [1], with the generic payload plus an ignorable extension for DMI configuration. We reuse some open-source TLM code from SoCLib [47] and SimSoC [26].

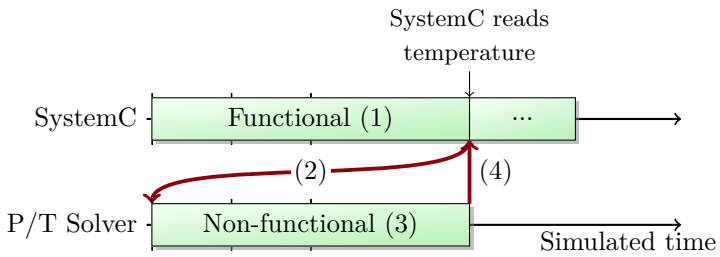

Figure 10 *functional ahead* strategy, in the absence of interrupt.

We have applied some classic optimizations in the TLM model, so that the base simulation speed is similar to the simulation speed of an industrial TLM model. In particular, we use *temporal decoupling* in all places where it is useful; the processor and VGA controller models use the *Direct Memory Interface* mechanism. When the processor is busy, the simulation speed is around 50 MIPS (million instructions per second).

The TLM model without power and temperature counts 5000 lines of code, and uses some general development kits counting in total 1400 lines of code. In this version, the temperature sensor and the DVFS controller modules are included but they have no behavior.

For the instrumentation of the platform itself, we have added about 100 lines of code. Note that this is quite small compared to the platform size, showing that once the tools and data are available, instrumenting an existing TLM model for power and temperature estimations requires a very little cost. One must provide the physical values used in the power and thermal model; this calibration task is out of this paper scope.

The core classes we have developed for power and temperature modeling counts 700 lines of code (not including the ATMI library, which is 2700 lines long). Additionally, the graphical user interface counts close to 600 lines.

6.2 Simulation Speed Overhead

To evaluate the simulation speed, we use our demonstration platform and make it run a benchmark application. In this benchmark, the processor is periodically computing: it waits one second and then computes during about 0.8 seconds (SystemC time). The application computes the "game of life", waiting 1 second between images. Additionally, the VGA controller is active and loads the image buffer 60 times per second. Between two consecutive reloads, the VGA controller remains idle for a few milliseconds.

The first time the model is simulated, the ATMI library computes some data in advance in order to accelerate the simulation itself. Those data are cached in a file for future simulations. Modifying the floorplan or some technology dependent parameters requires to compute this file again. This computation takes about two minutes.

Simulating 10 seconds (SystemC time) takes:

- 3.4 seconds (wall-clock time) for the initial functional TLM model
- 6.4 seconds with power and temperature estimations (6.6 seconds with GUI), assuming that the ATMI cache file was ready.

So, the simulation duration overhead is about **+88%**. We consider that it is a significant but acceptable overhead: the performance remain in the same order of magnitude, and it should be noted that the LIBTLMPWT approach is compatible with the common TLM abstractions and optimizations, which allowed gaining several orders of magnitude compared to lower-level models like RTL.

Table 1 Effect of the ATMI step duration in LIBTLMPWT.

ATMI step	simulation duration	standard deviation (σ)			
		processor	VGA	bus	temp. sensor
0.25 ms	15.3 s (+139%)	*reference*	*reference*	*reference*	*reference*
0.5 ms	8.9 s (+39%)	0.01 °C	0.05 °C	0.09 °C	0.00 °C
1 ms	6.4 s (*ref.*)	0.03 °C	0.11 °C	0.12 °C	0.01 °C
2 ms	5.2 s (-19%)	0.06 °C	0.23 °C	0.15 °C	0.03 °C
4 ms	4.6 s (-28%)	0.13 °C	0.42 °C	0.19 °C	0.05 °C

For example, DMI would need to be disabled if we did not take it into account. As expected, running the PWT simulation with DMI but without the extension is quick (≈5 seconds) but incorrect: we have observed errors of more than 1 degree Celsius. If we disable DMI, then the functional simulation consumes 9.1 seconds and the PWT simulation consumes 12.8 seconds. In other words, without our extended DMI, the total overhead would be 9.4 seconds, i.e. +276% (5.7 seconds for disabling the DMI plus 3.7 seconds for power and temperature computations).

Looking at the profile obtained with `callgrind`+`kcachegrind` [54], we notice that there are two performance-consuming spots: 1. computations internal to the ATMI library (≈28% of total simulation time), 2. applications of transaction side effects when using DMI (≈12% of total simulation time). The `pwt_module` class has been implemented with the optimization of this second performance-consuming spot in mind. These numbers show that the performance overhead comes mainly from the temperature simulation (about 2/3 of the overhead), not from the interfacing (about 1/3 of the overhead).

Concerning the time spent in the ATMI library, the user may optimize it by adapting the ATMI step duration. The values above are given for a step duration of one millisecond. As shown by Table 1, the longer the ATMI step is, the faster the simulation, at the cost of a loss of accuracy. The temperature error is higher in modules whose power density changes at a fast pace.

6.3 Applications

Using the "game of life" benchmark previously presented, we can observe that temperature plots are as expected. Figure 11 shows those plots at different time scales. Looking at a short time range, we see that the VGA temperature fluctuates with an amplitude slightly above 1 °C; as a consequence, the temperature sensor fluctuates too, but with a smaller amplitude. On the second plot, we see that the processor temperature fluctuates at a slower pace, since it is computing about one second every two. Moreover, other module temperatures evolve according to the processor temperature. Finally, the third plot shows that the whole system takes about 100 seconds to reach its maximum temperature. It is one reason why simulators must be fast enough: several minutes of SystemC time can be needed to observe the relevant behavior.

One possible application of our tool is to detect non-functional errors in embedded software, such as polling a device register instead of using idle mode and interrupts. Figure 12 shows what happens if the previous benchmark uses polling instead of interrupts. The functional behaviour is exactly the same, but we immediately see that the temperatures keep increasing and that the real chip would overheat. Note that the bus temperature is higher during polling than during frame computation due to the high polling traffic. The bug is obvious with temperature analysis, even if the analysis is imprecise. It would be much harder to find without it.

Another application is the development and validation of the voltage and frequency management. One simple solution to avoid overheating is to switch between two modes: a default fast mode

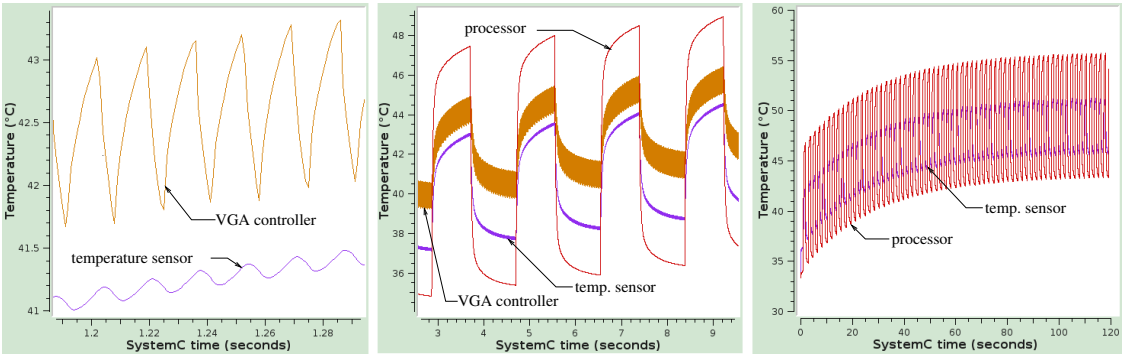

Figure 11 Temperature plots for the "game of life" benchmark, with different time scales.

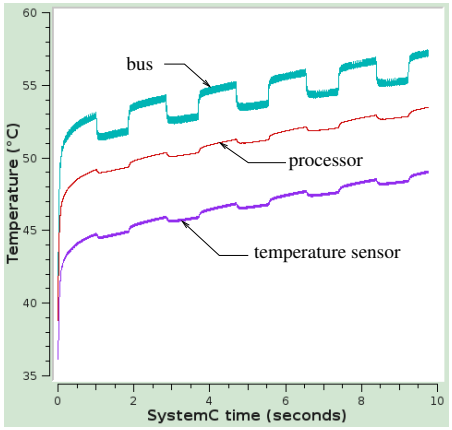

Figure 12 Variant of the "game of life" benchmark using polling.

where frequency and voltage are high, and a backup low-power mode where voltage and frequency are low. The power manager (i.e., a part of the embedded operating system) programs the interrupts of the temperature sensor module according to two thresholds: the high threshold is used to avoid overheating and causes the switch to the low-power mode, whereas the low threshold determines when to switch back to the fast mode. Testing this algorithm on a pure functional TLM model is impossible (see discussion in Section 2.2.3). It would require at least a scenario-based model, but in this case, temperature sensor interrupts would occur at unrealistic dates which would make the test more difficult. On the contrary, using the extended TLM model and its GUI allows to check easily the power manager's behaviour, as shown on Figure 13. Again, note that the simulation must be at least 20 seconds long to be useful, which shows that simulation speed matters.

Such temperature management based on low and high thresholds has some drawbacks; one is that the frequent temperature changes may raise the failure rate of the system [56]. Another approach is to use a PID controller. We have implemented this approach in the embedded software of our demonstration platform. As shown by Figure 14, we see that the PID-controlled temperature curve is smoother than the previous threshold-managed temperature curve. The first plot shows a simulation with a badly tuned PID controller, where the VGA controller temperature oscillations are amplified instead of smoothed, meaning that the gain parameters are likely too high.

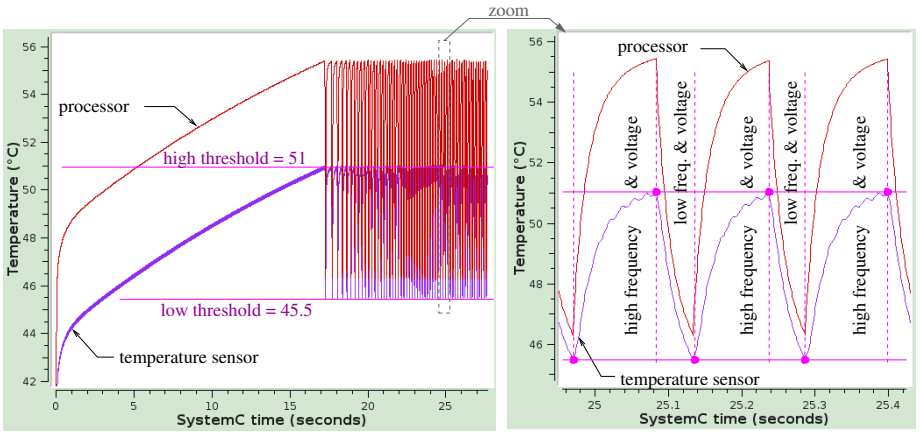

Figure 13 Temperature management with *low* and *high* thresholds: the software manager toggles between low and high power modes according to the temperature.

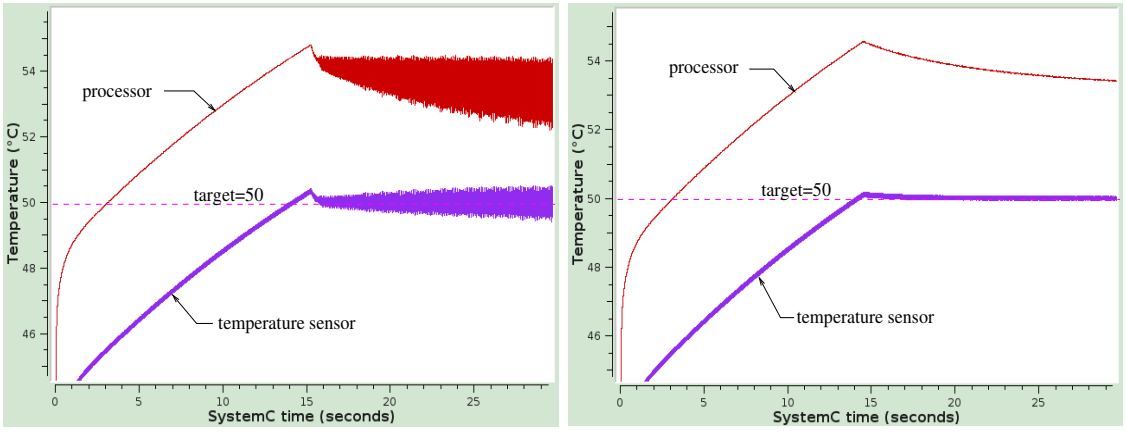

Figure 14 Temperature management using a PID controller, with distinct gain parameters.

6.4 Influence of Loose Timing on Performance and Precision

This section describes cosimulation techniques implemented in the tool presented in [9, 10], but not yet in LIBTLMPWT.

6.4.1 Granularity of the functional models

Figure 15 illustrates another experience with power consumption and temperature of the MEMORY component. This experiment was done using the cosimulation approach presented in Section 4, distinct from LIBTLMPWT, hence the performance results are not directly comparable with the previous sections. The functional behavior of the SoC is such that the memory receives a lot of traffic in the first half of the simulation period represented, and nothing in the second half. Fig. 15-(a) is the instruction-accurate simulation, on which the effect of this two-phase behavior is clearly visible: the temperature increases because of high power consumption, and then decreases slowly. Both Fig. 15-(b) and Fig. 15-(c) are obtained with a coarse-grained simulation, but we can see that only (c) reproduces the profile of (a). The difference is the following: in (b), the simulation is made in one big step (because there is no simulation instant in the middle), and the power consumption is spread over the whole interval; in (c), the simulation is split into two

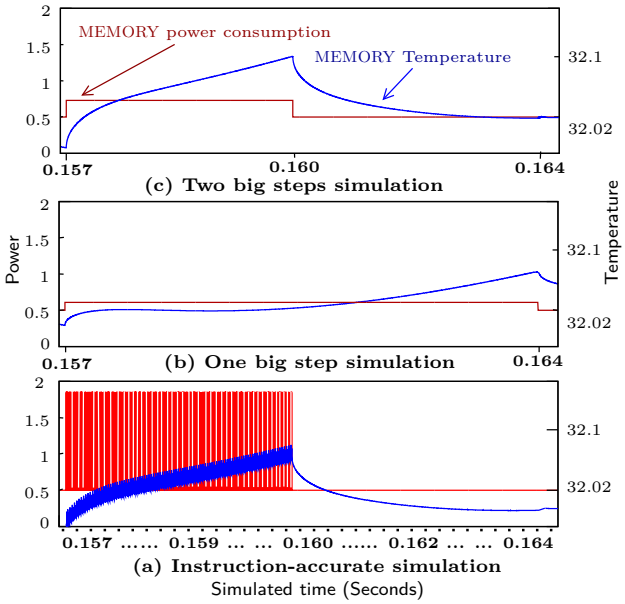

Figure 15 The effect of coarse granularity on memory power consumption.

simulation intervals. What happens is that in (b) the *functional and timed model* itself is too coarse. If the memory receives traffic in two well-differentiated patterns, it is probably because one of the components has two distinct running modes that have been ignored in the model. It means that the modeling of a component (like the software) can benefit from an explicit distinction between running modes, even if the impact is not on the *activity* model of component itself, but on the *traffic* model of another component.

6.4.2 Simulation speed

Table 2 is a summary of simulation speeds of the tool presented in [9, 10]. 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). The first line corresponds to an instruction-accurate simulation, the second one to simulation where we execute 100 instructions in a row, and the last one is the coarse-grained simulation based on logical synchronization points and explicit `purge` statements. We observe a factor 7 between the first and the third ones. It is also interesting to see that a lot can be gained on the side of the temperature simulator. ATMI is a fixed-step simulator, and we may obtain better results with a variable-step simulator. Note however that ATMI already uses a well optimized model, pre-computing the temperature response to a pulse, and using a sophisticated optimization called *event compression* [38]. Moreover, ATMI does not allow to ask for the evolution of temperature on time intervals smaller than $1\mu s$. When we execute the SystemC model one instruction at time, it represents a time interval of around 20 ns. It is useless to call ATMI for each of these small steps; a cache mechanism could be implemented in the connection, to call ATMI only when several steps with a total of more than $1\mu s$ have been executed on the SystemC side.

6.4.3 Interrupt Management

In Section 4.3 we explained how to model the abortion of a computation due to an interrupt faithfully with respect to power consumption. Figure 16 illustrates the effect of the method on

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

	Time spent in				
	Total	SC	ATMI	Connection	Number of exchanges
1-inst.	1028s	48.8%	41.2%	11%	15.7E+6
100-inst.	185s	5.2%	94.5%	0.03%	0.15E+6
coarse-grained	139s	5%	95%	$\approx$ 0%	128

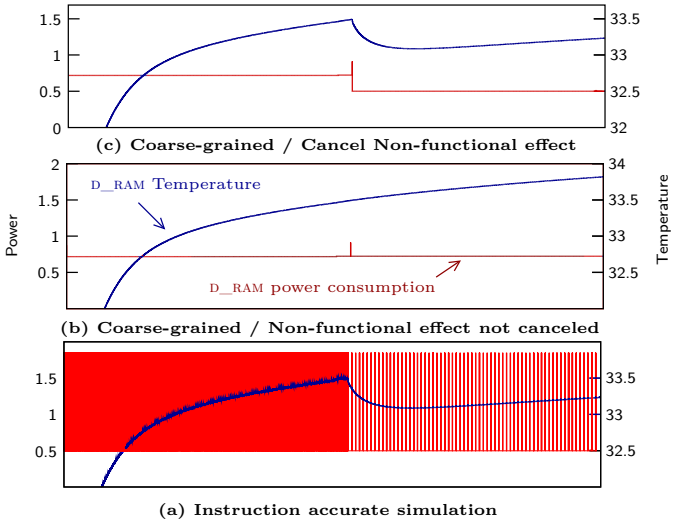

Figure 16 Modeling abortion.

MEMORY's temperature and power, due to the traffic it receives. We use a variant of the case-study where the power management policy just checks whether the temperature reaches the upper threshold and reacts by canceling the current operation of the software; this results in a much lower traffic on the memory. Fig. 16-(a) is the instruction-accurate simulation, which shows a high power consumption before the interrupt, and then a much lower one. Fig. 16-(b) is what we obtain with a coarse-grained simulation if we do not implement our method for modeling abortion faithfully. Fig. 16-(c) is what we obtain if we do implement it: we correctly observe that the temperature decreases when some functional behavior is aborted, resulting in a lower power consumption on the memory.

7 Conclusion

We presented several methods for cosimulation of a functional SystemC/TLM model with a power and a thermal model. LIBTLMPWT is a lightweight, integrated solution, that embeds ATMI as a temperature solver. It is available publicly as free software [27]. The tool presented in [9, 10] allows a cosimulation with an external solver, using inter-process communication to communicate with it.

Obviously, none of these tools allows recovering precision that was lost by raising the level of abstraction. If a component has a complex access pattern to a bus, and this pattern is not modeled, then none of our techniques can accurately model it. This is not surprising, as this would require *guessing* instead of *modeling*. Our contribution is, given a possibly loosely-timed functional platform, to provide modeling tools to create a non-functional model that is as accurate as it can be given the abstraction level of the functional platform.

The validation of these models is a difficult task. Ideally, we should compare the result of the models with a real system, but this is much harder than it seems. Measuring the temperature precisely can be done only on a chip without its packaging, which cannot run at full speed without overheating. This is a non-trivial task for silicon manufacturers, and clearly out of reach for an academic laboratory. Power-consumption measurements of individual components would require an instrumented version of the chip (with more pins than the actual system). Instead of validating our models against the real system, we compared several models at several levels of abstraction, considering the lower-level ones as the reference. Indeed, our contribution is not to provide power and thermal models, but to cosimulate them with a SystemC/TLM model. In other words, we assume the availability and accuracy of a non-functional model, and plug the input and output of this model on a functional simulation.

We experimented on a small but representative platform containing both hardware IPs and a processor. It would be interesting to experiment on a larger platform containing a large number of hardware IPs and/or a large number of processors (like many-core embedded processors or high-end general-purpose processors for data-centers where power and temperature are also important concerns). We do not foresee any fundamental issue with our cosimulation techniques: both the SystemC/TLM functional simulation and the power/thermal solvers we use are already used industrially at very large scales, and the cosimulation itself adds only a small overhead.

Using ATMI limits us to 2D designs. We are currently extending the interface to support Hotspot [28], which is able to manage 3D chips. It would be interesting to test other thermal solvers such as 3D-ICE [48]. Also, we focused on the cosimulation scheme, but a more integrated design-flow can be envisioned, where dedicated tools would be used for individual components, and the result used in the generation of our model. This is currently easy in theory but partly manual.

We are now working on extending the approach to support loose power and temperature annotations: when the precise values are not known, allow specifying an interval instead of possible values for each parameter.

References

1 Accellera Systems Initiative. *IEEE 1666 Standard: SystemC Language Reference Manual*, 2011. URL: `http://www.accellera.org/`.

2 Mazhar Alidina, José C. Monteiro, Srinivas Devadas, Abhijit Ghosh, and Marios C. Papaefthymiou. Precomputation-based sequential logic optimization for low power. *IEEE Trans. VLSI Syst.*, 2(4):426–436, 1994. `doi:10.1109/92.335011`.

3 Andrea Bartolini, Matteo Cacciari, Andrea Tilli, and Luca Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Trans. Parallel Distrib. Syst.*, 24(1):170–183, 2013. `doi:10.1109/TPDS.2012.117`.

4 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001. `doi:10.1007/3-540-45351-2_15`.

5 Luca Benini, Robin Hodgson, and Polly Siegel. System-level power estimation and optimization. In Anantha Chandrakasan and Sayfe Kiaei, editors, *Proceedings of the 1998 International Symposium on Low Power Electronics and Design, 1998, Monterey, California, USA, August 10-12, 1998*, pages 173–178. ACM, 1998. `doi:10.1145/280756.280881`.

6 Luca Benini and Giovanni De Micheli. Automatic synthesis of low-power gated-clock finite-state machines. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(6):630–643, 1996. `doi:10.1109/43.503933`.

7 Reinaldo A. Bergamaschi and Yunjian Jiang. State-based power analysis for systems-on-chip. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pages 638–641. ACM, 2003. `doi:10.1145/775832.775992`.

8 Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 sim-

ulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011. doi:10.1145/2024716.2024718.

9 Tayeb Bouhadiba, Matthieu Moy, and Florence Maraninchi. System-level modeling of energy in TLM for early validation of power and thermal management. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 1609–1614. EDA Consortium San Jose, CA, USA / ACM DL, 2013. doi:10.7873/DATE.2013.327.

10 Tayeb Bouhadiba, Matthieu Moy, Florence Maraninchi, Jérôme Cornet, Laurent Maillet-Contoz, and Ilija Materic. Co-simulation of functional systemc TLM models with power/thermal solvers. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*, pages 2176–2181. IEEE, 2013. doi:10.1109/IPDPSW.2013.206.

11 Dennis D Buss. Technology in the internet age. In *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, volume 1, pages 18–21. IEEE, 2002.

12 M. Caldari, M. Conti, P. Crippa, G. Nuzzo, S. Orcioni, and C. Turchetti. Instruction based power consumption estimation methodology. In *Electronics, Circuits and Systems, 2002. 9th International Conference on*, volume 2, pages 721 – 724 vol.2, 2002. doi:10.1109/ICECS.2002.1046270.

13 Marco Caldari, Massimo Conti, Massimo Coppola, Paolo Crippa, Simone Orcioni, Lorenzo Pieralisi, and Claudio Turchetti. System-level power analysis methodology applied to the AMBA AHB bus. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 20032–20039. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10234.

14 Chang-Chih Chen and Linda Milor. System-level modeling and microprocessor reliability analysis for backend wearout mechanisms. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 1615–1620. EDA Consortium San Jose, CA, USA / ACM DL, 2013. doi:10.7873/DATE.2013.328.

15 Jérôme Cornet. *Separation of Functional and Non-Functional Aspects in Transactional Level Models of Systems-on-Chip*. PhD thesis, Institut National Polytechnique de Grenoble, 2008.

16 Jérôme Cornet, Laurent Maillet-Contoz, Ilija Materic, Sylvian Kaiser, Hela Boussetta, Tayeb Bouhadiba, Matthieu Moy, and Florence 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*, page SESSION 10U: USER TRACK, San Francisco, États-Unis, Jun 2012.

17 Alessandro Danese, Graziano Pravadelli, and Ivan Zandona. Automatic generation of power state machines through dynamic mining of temporal assertions. In Luca Fanucci and Jürgen Teich, editors, *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 606–611. IEEE, 2016. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=7459383.

18 Anup Das, Akash Kumar, and Bharadwaj Veeravalli. Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.*, 27(3):869–884, 2016. doi:10.1109/TPDS.2015.2412137.

19 Dipankar Das, P. P. Chakrabarti, and Rajeev Kumar. Thermal analysis of multiprocessor soc applications by simulation and verification. *ACM Trans. Design Autom. Electr. Syst.*, 15(2), 2010. doi:10.1145/1698759.1698765.

20 Nagu R. Dhanwada, Ing-Chao Lin, and Vijaykrishnan Narayanan. A power estimation methodology for systemc transaction level models. In Petru Eles, Axel Jantsch, and Reinaldo A. Bergamaschi, editors, *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2005, Jersey City, NJ, USA, September 19-21, 2005*, pages 142–147. ACM, 2005. doi:10.1145/1084834.1084874.

21 Mohammad Javad Dousti and Massoud Pedram. Power-efficient control of thermoelectric coolers considering distributed hot spots. In Wolfgang Nebel and David Atienza, editors, *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 966–971. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2757037.

22 Bernhard Fischer, Christian Cech, and Hannes Muhr. Power modeling and analysis in early design phases. In Gerhard Fettweis and Wolfgang Nebel, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6. European Design and Automation Association, 2014. doi:10.7873/DATE.2014.210.

23 David Greaves and Mehboob Yasin. Tlm power3: Power estimation methodology for systemc tlm 2.0. In *Models, Methods, and Tools for Complex Chip Design*, pages 53–68. Springer, 2014.

24 Kim Grüttner, Philipp A. Hartmann, Tiemo Fandrey, Kai Hylla, Daniel Lorenz, Stefan Stattelmann, Björn Sander, Oliver Bringmann, Wolfgang Nebel, and Wolfgang Rosenstiel. An ESL timing & power estimation and simulation framework for heterogeneous socs. In *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2014, Agios Konstantinos, Samos, Greece, July 14-17, 2014*, pages 181–190. IEEE, 2014. doi:10.1109/SAMOS.2014.6893210.

25 Claude Helmstetter, Tayeb Bouhadiba, Matthieu Moy, and Florence Maraninchi. Fast and modular transaction-level-modeling and simulation of power and temperature. Technical report, Verimag Research Report, 2014. URL: http://www-verimag.imag.fr/~moy/?LIBTLMPWT-Model-Power-Consumption.

26 Claude Helmstetter, Vania Joloboff, and Hui Xiao. Simsoc: A full system simulation software for embedded systems. In *Open-source Software for Sci-*

entific Computation (OSSC), 2009 IEEE International Workshop on, pages 49–55, Sept 2009. doi:10.1109/OSSC.2009.5416870.

27 Claude Helmstetter and Matthieu Moy. LIBTLMPWT: Model power-consumption and temperature in systemc/tlm. Distributed under the terms of the GNU General Public License version 2, 2013. URL: http://www-verimag.imag.fr/~moy/?LIBTLMPWT-Model-Power-Consumption.

28 Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R. Stan. Hotspot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. VLSI Syst.*, 14(5):501–513, 2006. doi:10.1109/TVLSI.2006.876103.

29 IEEE. Ieee 1801-2009 — unified power format (upf), 2009. URL: http://standards.ieee.org/develop/project/1801.html.

30 Intel. *Intel? 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System Programming Guide, Part 2*, order number: 253669-057us edition, December 2015.

31 Sylvian Kaiser, Ilija Materic, and Rabih Saade. ESL solutions for low power design. In Louis Scheffer, Joel R. Phillips, and Alan J. Hu, editors, *2010 International Conference on Computer-Aided Design, ICCAD 2010, San Jose, CA, USA, November 7-11, 2010*, pages 340–343. IEEE, 2010. doi:10.1109/ICCAD.2010.5653615.

32 Pratyush Kumar and Lothar Thiele. System-level power and timing variability characterization to compute thermal guarantees. In Robert P. Dick and Jan Madsen, editors, *Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, Taiwan, 9-14 October, 2011*, pages 179–188. ACM, 2011. doi:10.1145/2039370.2039400.

33 Sumeet S. Kumar, Amir Zjajo, and René van Leuken. Ctherm: An integrated framework for thermal-functional co-simulation of systems-on-chip. In Masoud Daneshtalab, Marco Aldinucci, Ville Leppänen, Johan Lilius, and Mats Brorsson, editors, *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, Turku, Finland, March 4-6, 2015*, pages 674–681. IEEE Computer Society, 2015. doi:10.1109/PDP.2015.56.

34 Hugo Lebreton and Pascal Vivet. Power modeling in systemc at transaction level, application to a DVFS architecture. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2008, 7-9 April 2008, Montpellier, France*, pages 463–466. IEEE Computer Society, 2008. doi:10.1109/ISVLSI.2008.71.

35 Ons Mbarek, Alain Pegatoquet, and Michel Auguin. A methodology for power-aware transaction-level models of systems-on-chip using UPF standard concepts. In José L. Ayala, Braulio García-Cámara, Manuel Prieto, Martino Ruggiero, and Gilles Sicard, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation - 21st International Workshop, PATMOS 2011, Madrid, Spain, September 26-29, 2011. Proceedings*, volume 6951 of *Lecture Notes in Computer Science*, pages 226–236. Springer, 2011. doi:10.1007/978-3-642-24154-3_23.

36 Ons Mbarek, Alain Pegatoquet, and Michel Auguin. A methodology for power-aware transaction-level models of systems-on-chip using UPF standard concepts. In José L. Ayala, Braulio García-Cámara, Manuel Prieto, Martino Ruggiero, and Gilles Sicard, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation - 21st International Workshop, PATMOS 2011, Madrid, Spain, September 26-29, 2011. Proceedings*, volume 6951 of *Lecture Notes in Computer Science*, pages 226–236. Springer, 2011. doi:10.1007/978-3-642-24154-3_23.

37 Diego Melpignano, Luca Benini, Eric Flamand, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy, and Denis Dutoit. Platform 2012, a many-core computing accelerator for embedded socs: performance evaluation of visual analytics applications. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, pages 1137–1142. ACM, 2012. doi:10.1145/2228360.2228568.

38 Pierre Michaud and Yiannakis Sazeides. ATMI: analytical model of temperature in microprocessors. *Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2007.

39 Farid N. Najm. Towards a high-level power estimation capability. In Massoud Pedram, Robert W. Brodersen, and Kurt Keutzer, editors, *Proceedings of the 1995 International Symposium on Low Power Design 1995, Dana Point, California, USA, April 23-26, 1995*, pages 87–92. ACM, 1995. doi:10.1145/224081.224097.

40 Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In Pascal Felber, Frank Bellosa, and Herbert Bos, editors, *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012*, pages 29–42. ACM, 2012. doi:10.1145/2168836.2168841.

41 Pascal Raymond, Xavier Nicollin, Nicolas Halbwachs, and Daniel Weber. Automatic testing of reactive systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*, pages 200–209. IEEE Computer Society, 1998. doi:10.1109/REAL.1998.739746.

42 Björn Sander, Jürgen Schnerr, and Oliver Bringmann. ESL power analysis of embedded processors for temperature and reliability estimations. In Wolfgang Rosenstiel and Kazutoshi Wakabayashi, editors, *Proceedings of the 7th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2009, Grenoble, France, October 11-16, 2009*, pages 239–248. ACM, 2009. doi:10.1145/1629435.1629469.

43 Tanguy Sassolas, Chiara Sandionigi, Alexandre Guerre, Alexandre Aminot, Pascal Vivet, Hela Boussetta, Luca Ferro, and Nicolas Peltier. Early

design stage thermal evaluation and mitigation: The locomotiv architectural case. In Gerhard Fettweis and Wolfgang Nebel, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–2. European Design and Automation Association, 2014. doi:10.7873/DATE.2014.327.

44 Jürgen Schnerr, Oliver Bringmann, Alexander Viehl, and Wolfgang Rosenstiel. High-performance timing simulation of embedded software. In Limor Fix, editor, *Proceedings of the 45th Design Automation Conference, DAC 2008, Anaheim, CA, USA, June 8-13, 2008*, pages 290–295. ACM, 2008. doi:10.1145/1391469.1391543.

45 Si2. Common power format specification (CPF) 2.1, 2014.

46 M. Slee, A. Agarwal, and M. Kwiatkowski. Thrift: Scalable cross-language services implementation. *Facebook White Paper*, 2007.

47 An open platform for virtual prototyping of multiprocessors system-on-chip. URL: http://www.soclib.fr/.

48 Arvind Sridhar, Alessandro Vincenzi, Martino Ruggiero, Thomas Brunschwiler, and David Atienza. 3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling. In Louis Scheffer, Joel R. Phillips, and Alan J. Hu, editors, *2010 International Conference on Computer-Aided Design, ICCAD 2010, San Jose, CA, USA, November 7-11, 2010*, pages 463–470. IEEE, 2010. doi:10.1109/ICCAD.2010.5653749.

49 Synopsys. Primetime PX, 2015. URL: https://www.synopsys.com/apps/support/training/primetimepx_fcd.html.

50 Federico Terraneo, Davide Zoni, and William Fornaciari. An accurate simulation framework for thermal explorations and optimizations. In Gianluca Palermo, Daniel Gracia Pérez, Morteza Biglari-Abhari, Daniel Chillet, Smaïl Niar, and Adam Morawiec, editors, *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO@HiPEAC 2015, 21 January, 2015, Amsterdam, The Netherlands*, pages 5:1–5:6. ACM, 2015. doi:10.1145/2693433.2693438.

51 Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. VLSI Syst.*, 2(4):437–445, 1994. doi:10.1109/92.335012.

52 Ankush Varma, Eric Debes, Igor Kozintsev, Paul Klein, and Bruce L. Jacob. Accurate and fast system-level power modeling: An xscale-based case study. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008. doi:10.1145/1347375.1347378.

53 Emmanuel Viaud, François Pêcheux, and Alain Greiner. An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles. In Georges G. E. Gielen, editor, *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*, pages 94–99. European Design and Automation Association, Leuven, Belgium, 2006. doi:10.1109/DATE.2006.244003.

54 Josef Weidendorfer. Sequential performance analysis with callgrind and kcachegrind. In Michael M. Resch, Rainer Keller, Valentin Himmler, Bettina Krammer, and Alexander Schulz, editors, *Tools for High Performance Computing - Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*, pages 93–113. Springer, 2008. doi:10.1007/978-3-540-68564-7_7.

55 MY Yasin, C Koch-Hofer, Pascal Vivet, and DJ Greaves. Tlm power 3.0 (cbg) user manual version: Cbg 3.2 alpha draft manual-updated 1q2015-rev f, 2015.

56 Francesco Zanini, David Atienza, Luca Benini, and Giovanni De Micheli. Multicore thermal management with model predictive control. In *European Conference on Circuit Theory and Design (ECCTD 2009)*, volume 1, pages 90 – 95. IEEE Press, 2009. doi:10.1109/ECCTD.2009.5275073.