System-Level Modeling of Energy in TLM for Early Validation of Power and Thermal Management

Tayeb Bouhadiba
CNRS
Tayeb.Bouhadiba@imag.fr

Matthieu Moy
Grenoble INP
Matthieu.Moy@imag.fr

Florence Maraninchi
Grenoble INP
Florence.Maraninchi@imag.fr

Verimag (UMR CNRS 5104)
Grenoble, F-38041, France

***Abstract*—Modern systems-on-a-chip are equipped with *power architectures*, allowing to control the consumption of individual components or subsystems. These mechanisms are controlled by a power-management policy often implemented in the embedded software, with hardware support. Today's circuits have an important *static power* consumption, whose low-power design require techniques like DVFS or power-gating. A correct and efficient management of these mechanisms is therefore becoming non-trivial. Validating the effect of the power management policy needs to be done very early in the design cycle, as part of the architecture exploration activity. High-level models of the hardware must be annotated with consumption information. Temperature must also be taken into account since leakage current increases exponentially with it.**

Existing annotation techniques applied to loosely-timed or temporally-decoupled models would create bad simulation artifacts on the temperature profile (e.g. unrealistic peaks). This paper addresses the instrumentation of a timed transaction-level model of the hardware with information on the power consumption of the individual components. It can cope not only with power-state models, but also with Joule-per-bit traffic models, and avoids simulation artifacts when used in a functional/power/temperature co-simulation.

I. Introduction

A. Power Consumption and Temperature

Power consumption, usually expressed as the sum of *dynamic power* and *static power*, influences system temperature, which in turn, exponentially affects static power [1]. This interdependency makes it unavoidable to consider temperature when dealing with power.

Solutions for reducing power (and hence temperature) exist. For instance, *Clock gating* and *dynamic frequency scaling* are used to reduce *dynamic power* in an almost transparent way for the software. Because of the growing integration density of transistors (which increases static power), novel platforms offer solutions like *power gating* to reduce static power. The power architecture of such platforms consists in a set of *power domains* grouping components, and a *power controller* acting on *power switches* and *voltage controls* of the power domains.

Actions of the power controller are defined by a *power-management policy*, often implemented in the embedded software. The policy relies on physical (e.g., temperature, battery level) or logical (e.g., FIFO size) sensors to decide on the power configuration to apply. Validating the effect of the power management policy needs to be done very early in the design cycle, as part of the architecture exploration activity; hence the need for early system-level models of power consumption.

B. High-Level Models and Simulation

System-level modeling of energy consumption has become a very active field of research during the past ten years. We deal with models where actual software runs on a simulated hardware, before the RTL or gate models are available (simulating the software on such models is far too slow anyway). Transaction-Level Modeling (TLM) has been proposed to provide early-available abstract models that simulate fast. Methodologies based on SystemC [2] and the TLM library are now well accepted in the industry. TLM comes in several flavors, depending mainly on the precision of *timing* information. There has been a lot of work on the choice of the timing precision (or *granularity*) and its effect on the faithfulness of the models (e.g. [3]).

C. Early Validation of Power/Thermal Management

We deal with systems where the power management policy is sensitive to system temperature. For early validation of such a power management policy we rely on the cosimulation of timed SystemC-TLM models instrumented with power information, and a temperature solver (e.g., ATMI [4] or HotSpot [5]). The power/temperature solving can also be done with the industrial tool Aceplorer [6] using its cosimulation interface. The cosimulation mechanism is out of the scope of the paper. We focus on the power instrumentation of timed TLM models. We assume these models are appropriate for a functional analysis of the system, and that the information on the power consumption of individual components is given. For components like the CPU or the hardware accelerators, this information will be available as *power-state models* [7], [8]. For components like the bus, the available information will be a consumption in Joules per bit transmitted (called *traffic model* in the sequel). Even if this information is very accurate, the choice of the timing granularity (in order to gain simulation speed) has a strong impact on the estimation of power consumption and hence on temperature evolution.

Figure 1 illustrates the type of simulations we would like to obtain or avoid. The system under study alternates

This paper has been partially supported by the French ANR project HELP (ANR-09-SEGI-006).

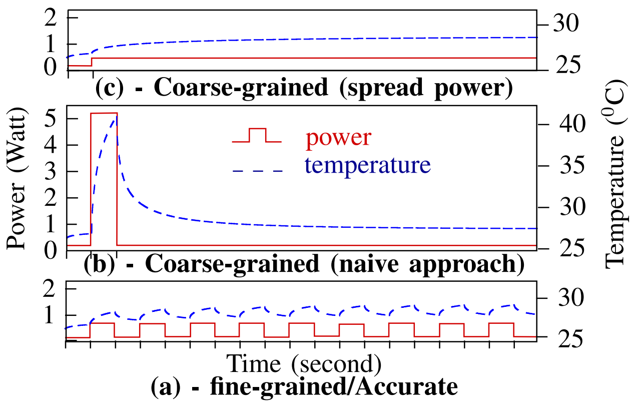

Fig. 1. Impact of timing granularity

short periods (e.g. one clock cycle) of high consumption with periods of low consumption. The temperature increases during the periods of high consumption, and decreases during the periods of low consumption, because of dissipation phenomena. The solid line represents energy consumption, and the dashed line represents the temperature, as given by a thermal model. Fig 1-(a) corresponds to a fine-grain model (e.g. cycle-accurate). The curves accurately describe the behavior of the chip. Fig 1-(b) corresponds to a coarse-grain model, where the functional behavior is simulated by executing large chunks of code instantaneously, interleaved with long periods of simulated time (this is a particular case of *temporal decoupling* [2]). If the energy consumption is naively related to the functional behavior, it will be considered to happen “at the same time”, thus producing a consumption peak, and an associated temperature peak, which are totally unrealistic. Moreover, if the power/thermal management policy implemented in the embedded software is meant to avoid peaks, the special behavior dedicated to this will be triggered when playing the software on the model, but not on the real circuit. A non-realistic power/thermal estimation will result in a non-realistic functional behavior.

The contribution of this paper is a power instrumentation method for TLM models with a focus on *traffic models*. Our approach produces the traces of Fig 1-(c), without the need for cycle-accurate models. The idea is to *spread* the power consumption associated with a given functionality, on the whole simulated period that simulates the time it takes. We apply our approach on a representative SoC, and we compare fine-grained models results with coarse-grained ones.

The rest of the paper is organized as follows: Section II discusses related work. Sections III introduces power modeling, and IV describes our contribution; Section V presents a small but representative case-study; Section VI describes the experimental setup and the results obtained for the case-study; Section VII concludes.

II. Related Work

A. Formal System-Level Models

In [7], we can find an early use of power-state models for the components of a chip at a very abstract level (e.g. no notion of temperature). A representation of the system with Binary-Decision-Diagrams, and model-checking exploration techniques are proposed. A similar modeling approach is that of [8]. These models can be extended with hybrid automata that also allow model-checking approaches [9].

In [10], a system-level *analytical* model (as opposed to *operational*, or state-based, as the two previous ones) is proposed, to capture the consumption and thermal behavior of a chip. It is based on the notion of a *Power Variability Curve*, which fits in the general framework of real-time calculus. The software is modeled as a set of tasks. Power consumption is considered to be a function of the executing task. This means that the consumption of hardware blocks other than processors is not considered. [10] computes guaranteed bounds on temperature peaks; the software has to be modeled, and the hardware architecture is not yet detailed; the method is intended to be used at a very early stage of the design.

Our solution is based on the same intrinsic principles as [7], [9], [8], augmented with the potential feedback of temperature on the functionality of the chip. We include all hardware components, and we can run the actual software. At this level of details, building a state model of the whole system and exploring it with model-checking is out of reach (or at least so costly that it can not serve as part of the architecture exploration activity).

B. Simulation System-Level Models

The approach in [11], [12] propose to model 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. Our proposal can be seen as a continuation of this work, providing: (i) higher-level abstractions (we can simulate much faster because we are not bound to using cycle-accurate models); (ii) the coupling with a temperature model and simulator; (iii) a more general methodology to build power models in SystemC/TLM. Current version of [12] allows logging energy per transaction, which is not sufficient for temperature analysis. The energy must be spread on appropriate time intervals to obtain power profiles from which realistic temperature curves may be computed (see Section I-C).

The work described in [13] is a system-level simulation method including the functionality and power consumption. The paper also describes in details how to get the parameters of the power-model, with measures. A complete case-study from Intel is described, and the simulation results compared to measures on the real chip. How to apply the same method to another case-study is not detailed, and there is no mention of temperature effects. Moreover, the paper does not detail the level of abstraction chosen for the simulation, especially on timing aspects, although it does mention transactions. Our proposal is meant to be more generic, and to allow the observation of power/temperature/functional effects, which happen with a power and thermal management policy.

III. System-Level Power Modeling in TLM

A realistic system-level model of energy consumption necessarily includes information on: 1) the energy consumption of the individual components, which depends on their current activity; 2) the power management policy, which observes the behavior of the system with physical (e.g., temperature) or logical (e.g., filling of a hardware FIFO) "sensors", and decides what to do regarding the electrical state of the components (clock, power, frequency); 3) the temperature model of the chip, relating the consumption and the temperature. In this section, we first describe the functional modeling practices our approach applies to, then we describe power instrumentation.

A. Modeling Function and Time in TLM

The intrinsic behavior of the SystemC discrete-event simulation engine is such that the simulation produces a sequence of *simulation instants* $t_0 = 0, t_1, t_2, ...$. We will also use the term *simulation intervals* to refer to the successive adjacent intervals $[t_0, t_1], [t_1, t_2], ...$

At high level of abstraction (loosely timed), a relatively large piece of code can be executed instantaneously (e.g. processing an image or a macro-block). To model time that should have elapsed during this action, one usually use a single `wait(`Δt`)` statement following it: an atomic execution of a SystemC process would have the form Δt`=compute_function(); wait(`Δt`);`, where `compute_function()` simulates a piece of behavior of the hardware and Δt is the time it would take to perform this behavior on the real circuit. TLM-2's *temporal decoupling* [2] produces this effect, but coarse-granularity models do not necessarily use a local clock to compute Δt. For instance, `compute_function()` may be be the reference sequential C for a video codec, while the hardware implementation would be a highly parallel hardware accelerator. Playing the SystemC model at a finer grain than what it is built for would be meaningless w.r.t. the real system.

B. Existing Work: Power-State Models

For a component whose energy consumption may vary in time, a power-state model [7] identifies a finite set of discrete *states*, each of them corresponding to a set of parameters from which a constant consumption value can be computed. In itself, a power state model of a component gives no information on why and when the component is in a given state. For instance, if we represent the DVFS operating points of a processor by a power-state model, then we also have to describe the part of the chip behavior that operates the DVFS command, thus changing the power state. By observing how long the component is in each of its power states, one can obtain the total consumption of a given scenario.

Power-state models of hardware components can be obtained by means of physical measures on the circuit. The value attached to a state can be given in Watts (Joules per second), or sometimes in Amperes, to keep the model unchanged even when the voltage changes.

Components like the CPU have two independent power-state models: their *electrical state* (power, frequency, clock); and their *activity state* related to the internal functionality (an image-processing hardware block can be either decoding an image, or just reading one); components like the bus and the memory will have a *traffic* model, and sometimes a power-state model for their electrical state.

Instrumenting a SystemC program with power-state information can be done by inserting appropriate function calls in the code (e.g. `set_state("activity", "Wait");` when the activity transitions to the state `Wait`). Part of the information needed for this instrumentation can be extracted from a UPF (Unified Power Format, IEEE 1801) specification.

IV. Joule-per-bit and Traffic Models

The functioning mode of a bus depends entirely on how it is solicited by the components connected to it, how the communications overlap or are sequentialized, etc. At the transaction-level of abstraction, we do not have sufficient information to drive an activity power-state of the bus. So we use an estimation of power consumption in the bus based on a Joule-per-bit abstraction [14]. Similarly, the power consumption of memory components depends on a lot of parameters not observable at the transaction-level modeling (e.g., access patterns, control logic, etc.). In the sequel, traffic models are described for the BUS components, but the same holds for memory components.

The *Joule-per-bit* abstraction is quite common for characterizing the consumption of a communication medium, in a wide variety of domains (from sensor networks to systems-on-a-chip). The energy consumed per bit transmitted can be obtained by performing measures on an implementation of the bus.

Any functional model can be instrumented in order to *count* the bits transmitted, hence providing an estimation of the total energy consumed during a scenario (if we ignore the feedback effects on temperature). As we explained in the introduction, if we want to detect peaks, and/or care about temperature, we need to produce realistic consumption *profiles* over time. This means that instrumenting the TLM model with a Joule-per-bit model of the bus is a bit more complex than just counting the bits transmitted for the whole scenario.

We can observe all the components connected to the bus, and count how many bits they send, and in which periods of time. For the bus, we gather all these contributions. The *traffic model* of the bus associates a transmission *frequency* F_i, given in bits per second, with each simulation interval $[t_i, t_{i+1}]$ as defined in Section III-A above. Equivalently, we can consider the number of transactions T_i in $[t_i, t_{i+1}]$, and $F_i = T_i/(t_{i+1} - t_i)$. The energy (in Joules) is $E_i = T_i \times s \times \gamma$ where γ is the cost of transmitting one bit, and s is the number of bits of a transaction; the power consumption (in Watts) is $E_i/(t_{i+1} - t_i) = T_i \times s \times \gamma/(t_{i+1} - t_i) = F_i \times s \times \gamma$.

A. Implementation in SystemC

The memories and the bus have a traffic model. Computing the transmission frequency in each global simulation interval is not trivial; the number of transactions on the bus is due to the contributions of all the components connected to it; these numbers have to be summed, and associated with the appropriate simulation interval.

Consider m components $C_1, C_2, ..., C_m$ connected to a bus. We explained in section III-A that the timed models are built with the behavior of each component being modeled by SystemC code of the form: Δt = `compute_function(); wait(`Δt`);`. Instrumenting for the traffic means that the computation of the functional effect

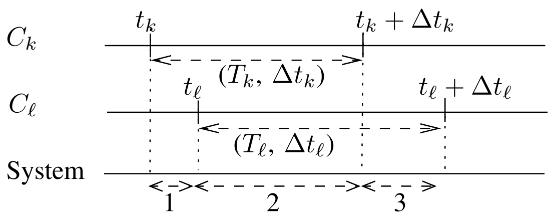

Fig. 2. Several components contributing to the traffic on the same bus

also delivers the number T of transactions produced: $(\Delta t, T)$ `= compute_function(); wait(`Δt`);`

Consider Figure 2. If such a piece of code, for a component C_k, is executed at the simulation instant t_k, we get a contribution of T_k transactions during an interval $[t_k, t_k{+}\Delta t_k]$. For another component C_ℓ connected to the bus, whose code is executed at the simulation instant t_ℓ, we might get a contribution of T_ℓ transactions during the interval $[t_\ell, t_\ell{+}\Delta t_\ell]$.

All the instants belong to the set of global simulation instants, and we should be able to associate a number of transactions with each global simulation interval of the system (denoted by 1, 2, 3 on the figure). We first transform the numbers of transactions into frequencies: $F_k = T_k/\Delta t_k$ and $F_\ell = T_\ell/\Delta t_\ell$. Then we see that in the simulation interval 1, only C_k is contributing, hence we have the frequency of transactions F_k, or a number of transactions $F_k \times (t_\ell - t_k)$; during the simulation interval 2, both components contribute, so the frequency is $F_k + F_\ell$ and the number of transitions is $(F_k + F_\ell) \times (t_k + \Delta t_k - t_\ell)$; during the simulation interval 3, only C_ℓ is contributing, hence a frequency F_ℓ and a number of transactions $F_\ell \times (t_\ell + \Delta t_\ell - (t_k + \Delta t_k))$.

In the SystemC implementation, each component that has a traffic model embodies data structures to map SystemC threads to transaction counts and frequencies. Transaction counts are updated each time a transaction is observed. We add wrapper functions of SystemC *wait on time* primitive in order to be notified each time a thread calls the *wait on time* primitive. Then each traffic model converts the transaction count of the concerned process to its transaction frequency, knowing the duration given by the wrapper function. Another wrapper function (of the sc_start() primitive) is required to notify the end of the simulation instant t_i, and to report the next instant t_{i+1}; components then update their traffic model according to the sum of the transaction frequencies of all threads during the interval $[t_i, t_{i+1}]$.

V. Case Study

To illustrate our approach we use a small, but representative SoC including a Microblaze soft core. It is based on a real FPGA system, designed with Xilinx EDK. We augmented it with power and temperature features (that the original system does not include because of the limitations of FPGAs).

Figure 3 is the hardware architecture of the chip as modeled with SystemC/TLM. It is made of an ISS (Instruction Set Simulator), two memories I_MEM and D_MEM, a TIMER, an interrupt controller INTC, and a VGA controller VGAC, which reads images from D_MEM. At the end of a transfer, the VGAC signals an interrupt. A temperature sensor T_SENS exposes the temperature to the software with a read-only register. It may also be programmed with two values T_{high} and T_{low} to send interrupts when the temperature measured equals or exceeds T_{high} (resp., equals or falls below T_{low}).

Components are grouped into three power domains. pd$_1$ comprises most of the components and is set to voltage 5 V. pd$_2$ comprises the VGAC which may be switched on and off (voltage values 5 V and 0 V). Finally pd$_3$ comprises the CPU to which we may apply Dynamic Voltage and Frequency scaling. The operating points are (5 V, 50 MHz) and (3 V, 20 MHz). A power controller PW_CTL manages power domains and frequencies. It has software-programmable registers to determine the power configurations to be applied.

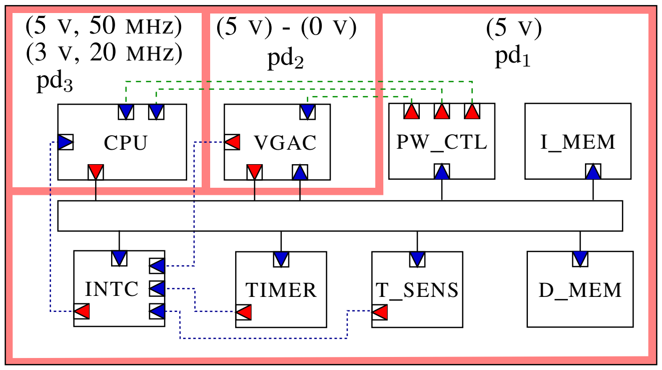

Fig. 3. TL-Model of the hardware architecture of the case study

A. The Embedded Software

Figure 4 is a sketch of the software. It computes a new image to be displayed by the VGAC, at each timer interrupt. The software implements a simple power management policy, sensitive to system temperature. The sensor is programmed with two temperature values. If the temperature reaches T$_{high}$ (resp., T$_{low}$) the software programs the power controller to switch off (resp. on) the VGAC, slow down (resp. speed up) the timer, and scale down (resp. up) processor voltage and frequency. The hysteresis mechanism implemented by these two thresholds is meant to avoid shaking.

```
int main () {
  init_all(); /* hardware components */
  while (1) {
    if (sens_it_signaled) // sensor interrupt?
       update_power_config_and_timer_period();
    if (must_compute_image) // timer interrupt?
       compute_image();
    wait_interrupt(); // let CPU sleep until next IRQ
  }
}
/////// Interrupt Service Routines ///////
/* request to compute a new image */
void timer_isr () { must_compute_image = 1; }
/* program vgac with new image address */
void vgac_isr () { program_image_add(); }
/* request to handle sensor interrupt */
void sens_isr () { sens_it_signaled = 1; }
```

Fig. 4. Sketch of the embedded software

B. SystemC/TLM and Thermal Solver Cosimulation

We use the temperature solver ATMI [4] to compute the temperature at the sensor place according to the power consumption of the components. Our method, based on a cosimulation technique, is able to set the sensor temperature at each instant, and to produce interrupts (if any) at the correct date. Cosimulation details are out of the scope of this paper.

C. Granularity of the Model: Example of the CPU

Our approach is meant to be applicable on models timed with a coarse granularity. To validate our approach (see Section VI), we need to be able to compare coarse granularity results with finer ones. We illustrate this on the CPU model (i.e., the ISS). The embedded software is run unchanged on the model whatever the granularity.

The ISS can be used at several granularities: (i) one instruction at a time (called *instruction-accurate* in the following), and then a synchronization point; this is the slowest, and can be done in a transparent way for the software; (ii) several instructions between two synchronization points. In the latter case, the ISS maintains a counter of instructions. After executing a sequence of n instructions in a row, the ISS produces a `wait` statement, with a duration computed as a function of n and the current frequency of the processor. Either the number n of instructions to be executed in a row is predefined and this is again transparent for the software, or the software is instrumented with `purge` statements at logical points. `purge` is interpreted by the ISS as a special instruction that produces the `wait` statement, with n the number of instructions executed since the last `purge`. Executing several instructions in a row in the ISS, with no synchronization point, speeds up the simulation. Placing `purge` statements is a way to control the faithfulness of the model.

Notice that the particular case of the ISS makes it possible to choose the time granularity freely, but an execution at a fine granularity is not always possible without rewriting the model for components other than CPUs.

VI. Experimental Results

All the experiments use variants of the case-study of Section V, and are meant to observe and validate the power model structure of Section III and the traffic analysis principles of Section IV. We assume that the physical data summarized by power-state models and traffic models have been validated beforehand, and we only discuss the influence of timing granularities on the simulation speed and the accuracy of the system-level model we built on top of these data. The instruction-accurate version is therefore considered as the reference. The experiments check that the coarse-grain models (that are available at earlier stages of the design flow) produce a good approximation of this reference.

A. Spreading power consumption over simulation intervals

Figure 5 illustrates the effect of a naive coarse-grain model (with temporal decoupling using `purge` statements as explained above), as mentioned in the introduction (Figure 1). The three curves represent the temperature evolution for our case-study where power-management has been turned off. The dashed curve (1) corresponds to a coarse-grain simulation, and is obtained when the power consumption is considered to happen at the beginning of the simulation intervals. The two other curves correspond to: (2) the instruction-accurate simulation (i.e. with a time step of 20 ns); and (3) a coarse-grain simulation with the principles of section IV implemented to spread power consumption due to traffic on appropriate simulation intervals. We observe that the curve (1) departs from the other ones significantly, showing unrealistic peaks.

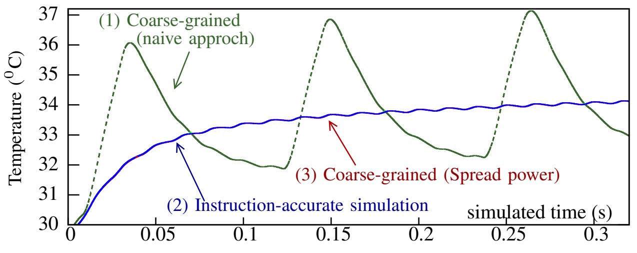

Fig. 5. Spreading power consumption over simulation intervals

B. Validating power management policies

Figure 6 illustrates the typical use of our system-level models: the validation of a power- and thermal-management policy. The top of the figure shows power consumption for the coarse-grain simulation only (the power consumption for instruction-accurate simulations is unreadable at this scale, because it changes far too often). The bottom of the figure shows the temperature evolution for the instruction-accurate and coarse-grain simulations, and the interrupts that trigger the power-management actions. We can see that the two curves almost coincide and that the interrupts occur at the same time.

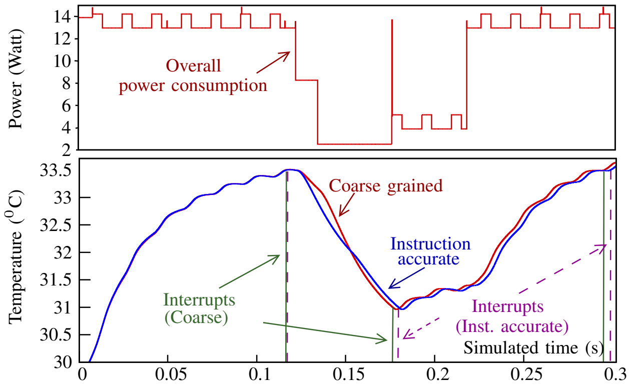

Fig. 6. Power and thermal simulation for the case study of section V

C. Granularity of the functional models

Figure 7 illustrates another experience with power consumption and temperature of the D_RAM component. 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. 7-(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. 7-(b) and Fig. 7-(c) are obtained with a coarse-grain 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 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.

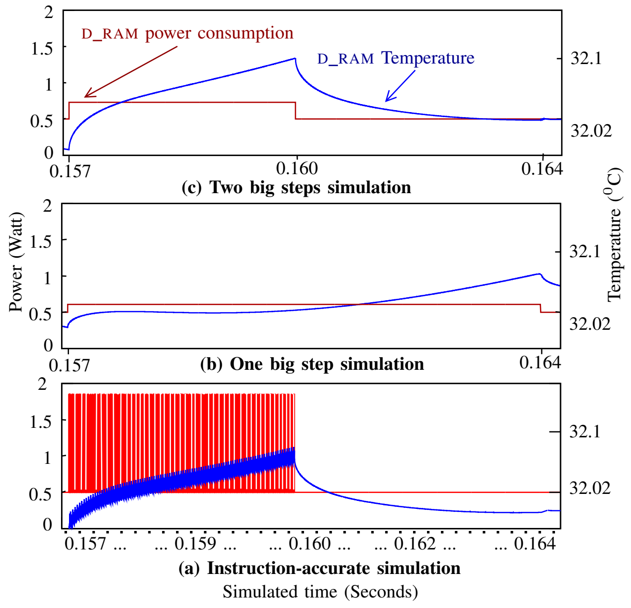

Fig. 7. The effect of coarse granularity on memory power consumption

D. Simulation speed

Figure 8 illustrates simulation speeds for the case study and the overhead of its power instrumentation. For coarse-grained models we are interested in (those based on logical synchronization points and explicit `purge` statements), we think that the instrumentation overhead is acceptable (25%). The fine-grained models (i.e., 1 instruction at a time, first line) are used only as a reference for validating the accuracy of the coarse-grain models. Unsurprisingly, at this level the instrumentation overhead would not be acceptable (1104%).

	SC/TLM	SC/TLM+Power	overhead
1-inst.	271.43s	3265.58s	+1103%
100-inst.	111.75s	169.74s	+52%
Coarse-grain	109.02s	136.17s	+25%

Fig. 8. Execution times and power instrumentation overheads for simulating $10s$ of the system at three distinct levels of granularity.

VII. Conclusion

We have shown how to start from a functional and timed system-level model of a system-on-chip, and augment it with information on the power-consumption of the individual components, which comes into two forms: power-state models, and traffic models. We showed how to compute traffic models carefully, so as to avoid bad effects of temporal decoupling on the faithfulness of the power consumption profile, without losing the simulation speed of temporal decoupling. Our models are coupled with a temperature simulator, which allows to observe the behavior of a power-management policy, including those that have a feedback effect on the functionality.

We implemented the described techniques as a set of libraries for power instrumentation, traffic analysis and model cosimulation, together with wrapper functions of SystemC primitives to notify end of instants and report *wait* durations. The implementation also includes non-trivial mechanisms to manage interrupts properly (considering the interrupt handler's non-functional effect at the right simulation time, aborting computation and canceling its non-functional behavior).

Further work will investigate other issues related to model granularity, e.g. when the functional model has a granularity finer than the thermal solver, or when the temporal decoupling annotations would allow a better precision than the raw SystemC time.

References

[1] Y. Liu, R. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 1526–1531.

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

[3] J. Cornet, F. Maraninchi, and L. Maillet-Contoz, "A method for the efficient development of timed and untimed transaction-level models of systems-on-chip," in *Proceedings of*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 9–14.

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

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

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

[7] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proceedings of the 1998 international symposium on Low power electronics and design*, ser. ISLPED '98. New York, NY, USA: ACM, 1998, pp. 173–178.

[8] R. A. Bergamaschi and Y. W. Jiang, "State-based power analysis for systems-on-chip," in *Proceedings of the 40th annual Design Automation Conference*, ser. DAC '03. New York, NY, USA: ACM, 2003, pp. 638–641. [Online]. Available: http://doi.acm.org/10.1145/775832.775992

[9] D. Das, P. P. Chakrabarti, and R. Kumar, "Thermal analysis of multiprocessor SoC applications by simulation and verification," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 15, pp. 15:1–15:52, March 2010.

[10] P. Kumar and L. Thiele, "System-level power and timing variability characterization to compute thermal guarantees," in *CODES+ISSS 2011*. Taipei, Taiwan: ACM, 2011, pp. 179–188.

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

[12] M. Yasin, C. Koch-Hofer, P. Vivet, and D. Greaves, "TLM power 3.0 (CBG) user manual," koo.corpus.cam.ac.uk/tlm-power3, 2012.

[13] A. Varma, E. Debes, I. Kozintsev, P. Klein, and B. L. Jacob, "Accurate and fast system-level power modeling: An xscale-based case study," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, 2008.

[14] P. Sotiriadis, A. Chandrakasan, and V. Tarokh, "Maximum achievable energy reduction using coding with applications to deep sub-micron buses," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 1. IEEE, 2002, pp. I–85.