Modeling Heterogeneous SoCs with SystemC: A Digital/MEMS Case Study

Ankush Varma†‡, M. Yaqub Afridi‡, Akin Akturk†, Paul Klein*, Allen R. Hefner‡ and Bruce Jacob†

†University of Maryland, College Park, MD 20740
‡National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899
*Intel Corporation, Chandler, AZ 85248

{ankush, blj}@eng.umd.edu

ABSTRACT

Designers of SoCs with non-digital components, such as analog or MEMS devices, can currently use high-level system design languages, such as SystemC, to model only the digital parts of a system. This is a significant limitation, making it difficult to perform key system design tasks — design space exploration, hardware-software co-design and system verification — at an early stage. This paper describes lumped analytical models of a class of complex non-digital devices — MEMS microhotplates — and presents techniques to integrate them into a SystemC simulation of a heterogeneous System-on-a-Chip (SoC). This approach makes the MEMS component behavior visible to a full-system simulation at higher levels, enabling realistic system design and testing. The contributions made in this work include the first SystemC models of a MEMS-based SoC, the first modeling of MEMS thermal behavior in SystemC, and a detailed case study of the application of these techniques to a real system. In addition, this work provides insights into how MEMS device-level design decisions can significantly impact system-level behavior; it also describes how full-system modeling can help detect such phenomena and help to address detected problems early in the design flow.

Categories and Subject Descriptors

B.7.2 **[Integrated Circuits]** Design Aids — *Simulation.* C.4 [**Performance of Systems]** Modeling Techniques. I.6 **[Simulation and Modeling]** Simulation Languages, Model Development, Applications.

General Terms

Design, Experimentation, Performance

Keywords

SystemC, MEMS, Modeling, Microhotplate, Gas Sensor, Power

CASES'06, October 23–25, 2006, Seoul, Korea.

1. INTRODUCTION

Embedded systems are simultaneously growing more complex in design and shrinking in physical size, resulting in an increasing number and diversity of components being fabricated on a single chip. The System-on-Chip (SoC) approach exploits increases in transistor count to deliver simultaneous benefits in performance, power dissipation, reliability, footprint and cost [13]. The emergence of the SoC design paradigm has led to the development of system-level design, modeling, and verification tools, such as SystemC, that focus on digital systems at high levels of abstraction. These tools enable system designers to perform detailed exploration of a wide range of configurations early in the design flow. In addition, high-level modeling also provides an “executable specification” that software developers may use as a target, and that verification engineers may use to generate test vectors. As a result, the software stack can be developed in parallel with the later stages of hardware design, significantly reducing time-to-market.

Modern SoCs can incorporate not only digital but also analog and MEMS components on the same silicon substrate. Extensive research has been done on analog and MEMS fabrication techniques, with the result that many such components can now be fabricated using processes compatible with standard digital CMOS process technologies [4]. This gives designers a new capability but raises a number of important questions. How are these non-digital components to be modeled in system simulation? How is the software driving heterogeneous components to be written, tested, debugged and optimized? To exploit the wide range of components and perform hardware-software co-design and validation, the high-level models used must accurately represent all SoC components.

In practice, the requirement to model *all* SoC components faithfully can be relaxed under certain circumstances — for example, if the communication between a non-digital and a digital component is predominantly unidirectional or deterministic. During high-level modeling, components such as pad drivers or clock generators can be abstracted away conveniently and without significant loss of accuracy because they do not usually impact high-level system behavior in complex ways.

However, this approach — abstracting away non-digital behavior entirely — becomes invalid when there is *feedback* in the system, such as in the case of microprocessors running control programs that interact with analog or MEMS sensors and actuators. Components with complex time-dependent behavior cannot be abstracted away because the behavior of the digital system can depend on both time and the state of the non-digital component. Unfortunately, current high-level SoC design tools, such as SystemC, only allow digital components to be modeled.

There is thus a gap between the high-level event-driven simulation methodology used by the SoC designer and the FEM, SPICE or MATLAB-based differential-equation-solving approach used for design and analysis of non-digital components. Accurate modeling of feedback systems containing heterogeneous components requires bridging this gap. The alternative — waiting for a hardware prototype before performing software development and verification — is undesirable for reasons of cost, complexity and time-to-market. Current design flows demand that the complete system be modeled, tested, debugged and verified well before the expensive fabrication stage, where design modification costs become prohibitive.

This paper presents an approach for modeling the functionality, performance, power, and thermal behavior of a complex class of non-digital components — MEMS microhotplate-based gas sensors — within a SystemC design framework. The components modeled include both the digital components (such as microprocessors, busses and memory) and the MEMS devices comprising a gas sensor SoC.

The contributions made in this work include the first SystemC models of a MEMS-based SoC and the first SystemC models of MEMS thermal behavior, as well as techniques for significantly improving simulation speed. Towards demonstrating the effectiveness of these techniques, a detailed case study of the application of the proposed approach to a real heterogeneous SoC is also presented, providing some insights on how device-level design decisions can have system-level impact, and how such issues can be studied and addressed through integrated full-system modeling.

The rest of this paper is organized as follows: section 2 provides an overview of related work in literature, section 3 describes the operation and architecture of the MEMS Gas Sensor SoC, section 4 discusses the methodology used for the characterization and modeling of system components, section 5 illustrates some of the results and insights that can be obtained using integrated SoC simulation, and section 6 presents conclusions and directions for future work.

2. RELATED WORK

There has been relatively little work so far on modeling the behavior of non-digital SoC components within standard SystemC frameworks. Bjornsen et. al. [8] describe using SystemC to model the transient behavior of high-speed analog-to-digital converters. They found SystemC to be an effective modeling tool, with simulation speeds significantly faster than HDL. Zhang et. al. [18] compared Verilog, VHDL, C/C++ and SystemC as candidates for modeling liquid flow in a microfluidic chemical handler, and found SystemC to be the most suitable, since SystemC processes, events and modules are suitable building blocks for expressing fluid flow in a manner analogous to dataflow. This paper presents the first SystemC models of a MEMS-based SoC, the first SystemC models of MEMS thermal behavior, techniques for improving simulation efficiency, and a detailed case study of the application of this approach to a real heterogeneous SoC. The rest of this section provides background information on related work in literature.

Attempts at generalized modeling of mixed-signal elements for large-scale hardware design include VHDL-AMS [10] and Verilog-AMS [11], aimed at extending the VHDL and Verilog language definitions to include analog and mixed-signal regimes. These have been moderately successful for mixed-domain component modeling; however, they are designed for implementation and end-of-design verification late in the design flow, not for system-level design and verification. Effective system-level design involves representing entire systems at high levels of abstraction and modeling them at high simulation speeds. These requirements are not adequately met by HDL frameworks that primarily target component-level design, creating the need for higher-level techniques and tools that are more efficient at system-level design.

A recent key advance in system design has been the development of higher-level languages and tools for expressing hardware constructs. In particular, SystemC [12] — a freely available C++-based library that provides a variety of hardware-oriented constructs and an event-based simulation kernel — has gained rapid acceptance. It is now supported by a variety of EDA tools and IP vendors and has been widely adopted as a standard modeling platform[1] that enables the development and exchange of fast system-level models and supports system-level software development, top-down design, IP core integration, hardware-software co-design and system verification. Designs can be expressed at a variety of levels of abstraction in SystemC [9, 12]. In particular, accurate and high-speed simulation at high levels of abstraction is a key tool, enabling designers to model the behavior of large workloads on complex systems.

The SystemC 2.0 standard [14] addresses purely digital simulation. However, increasing on-chip heterogeneity has led to the demand for modeling both digital and non-digital components within an integrated framework. Ongoing efforts such as SystemC-AMS [17] and SEAMS [7] propose extensions to the SystemC language definition and additions to the SystemC kernel to incorporate analog and mixed-signal devices into the simulation framework. In contrast, the techniques and models presented in this paper use a standard, unmodified SystemC kernel and library to model non-digital components, and represent the first application of SystemC design to a MEMS SoC.

3. THE MEMS GAS SENSOR SOC

A microhotplate-based gas sensor exploits temperature-dependent conductivity variations in certain thin films to facilitate the detection of trace gases in the ambient atmosphere. The MEMS gas sensor SoC presented here integrates an array of such sensors with on-chip digital circuitry to enable programmable control and data gathering. This SoC incorporates a wide range of components: a MEMS microhotplate-based gas sensor array, an 8051 microcontroller, and on-chip interconnect and peripherals. In such a system, one of the design challenges is posed by

[1] SystemC was approved as IEEE standard 1666 in December 2005.

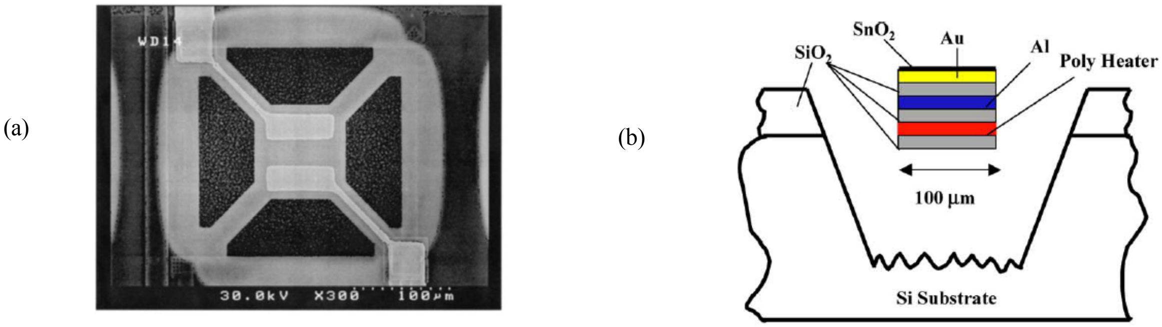

Figure 1. (a) Scanning Electron Microscope (SEM) micrograph of a microhotplate, showing it suspended above the underlying substrate. Cantilever supports at the corners provide structural support and electrical pathways. The gold electrodes, between which the thin sensor film is deposited, are also visible. The microhotplate is fabricated with a standard digital CMOS foundry process, followed by an etching step to suspend the microstructure and chemical vapor deposition of the metal oxide thin film.
(b) Cross-section of the suspended microhotplate. The figure shows the polysilicon heater, the Al temperature sensor, the metal oxide sensing film and the insulating SiO$_2$ layers. Cantilever supports are not shown.

the heterogeneity of the components involved: issues regarding analog, digital and MEMS design all need to be understood and taken into account. The following sections describe SoC design and operation, with section 3.1 presenting the structure and operation of microhotplate-based gas sensors, and section 3.2 describing overall SoC topology and system architecture.

3.1 The MEMS Microhotplate-Based Gas Sensor

The conductance of certain metal oxide films varies with the temperature, concentration, and type of gas molecules adsorbed into the film. Conductance-type gas microsensors use a MEMS microhotplate device to vary the temperature of the thin film to facilitate the detection of trace gases in the environment. Monolithic integrated gas sensors have numerous possible applications such as detecting food product freshness, detecting toxin leakage in chemical facilities, or identifying hazardous chemical agents in public places.

A microhotplate is a MEMS device used to obtain high temperatures over a localized area on a silicon chip. Bulk micromachining techniques [4] can physically and thermally isolate the heating elements from the underlying silicon substrate, allowing surface temperatures as high as 450ºC to be reached. Such structures feature low power dissipation, low fabrication cost, and scalability to different process technologies, making them suitable for use in chemical microsensors [3] or as microscopic infrared sources [15].

Recent advances in MEMS fabrication have allowed these to be scalably implemented with standard CMOS-compatible foundry processes, enabling designers to integrate MEMS gas sensors, analog components, and digital components into a single SoC [3, 4]. The microhotplate's small size facilitates building the on-chip sensor arrays needed for gas classification in complex sensing environments.

Structural Components A microhotplate-based gas sensor consists of a central platform supported at each corner by a cantilever joining it to the substrate, as illustrated in Figure 1(a). The material immediately below and around the platform is etched away in a single postprocessing step, which physically and thermally isolates it from the substrate. The central structure of the microhotplate is physically suspended over empty space, with only the cantilevers at the corners providing mechanical support.

Electrical Components Electrically, a microhotplate-based gas sensor comprises three major components, shown in Figure 2(a): a polysilicon heater, a temperature sensor, and a thin film gas sensor. The cross-section of the microhotplate in Figure 1(b) illustrates their physical implementation as conductive layers separated by insulating silicon oxide layers. A description of each component follows:

- **Polysilicon Heater**: Implemented as a serpentine resistor, this generates heat to raise microhotplate temperature. The heater current or voltage may be controlled. Note that the electrical resistance of a polysilicon heater is not constant and changes linearly with temperature within the range of operation.
- **Temperature Sensor**: Implemented in an Aluminum or Polysilicon layer with a known temperature coefficient of resistance (TCR). A small constant current is passed through this, and the voltage drop across it is used to measure microhotplate surface temperature.
- **Gas Sensor Film:** A thin film of tin or titanium oxide (SnO$_2$ or TiO$_2$) is deposited between two gold electrodes onto the top surface of the microhotplate, exposed to the external atmosphere. The thin film conductivity changes when specific molecules are adsorbed into it. The observed conductivity patterns depend on the temperature, concentration and type of adsorbed molecules, giving molecules a signature pattern that facilitates chemical detection. Since different thin films interact differently with gas molecules [4], individual elements in a microhotplate array may differ in the type of sensor film used to improve sensing ability.

A microsensor array can be encapsulated behind a digital-only interface as illustrated in Figure 2(b), facilitating integration into high-level digital SoC designs. A digital-to-analog converter (DAC) drives the polysilicon heater current and an ADC

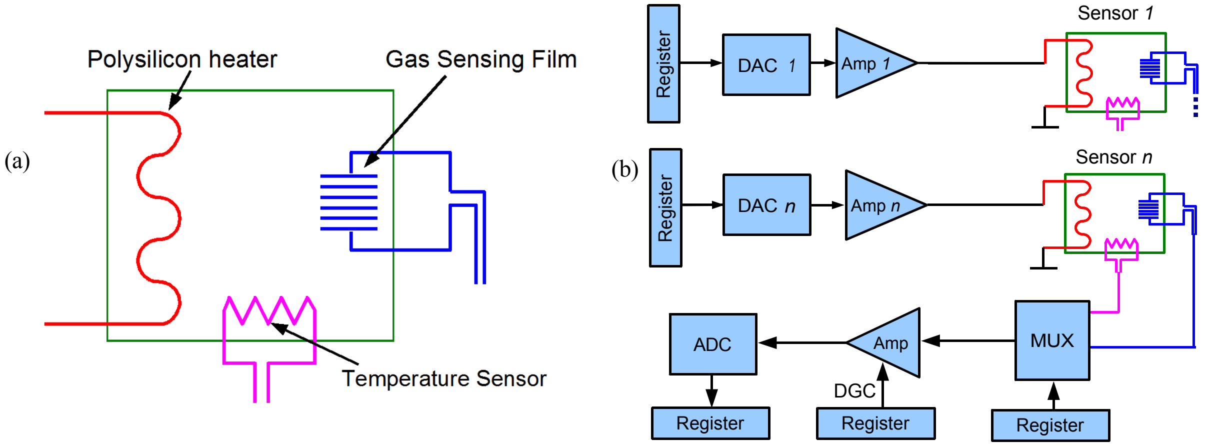

Figure 2. (a) Schematic showing the electrical components of the microhotplate-based gas sensor. (b) Schematic illustrating digital encapsulation of a sensor array using an ADC/DAC array and multiplexing. A Digital Gain Control (DGC) register may be used to improve accuracy and dynamic range.

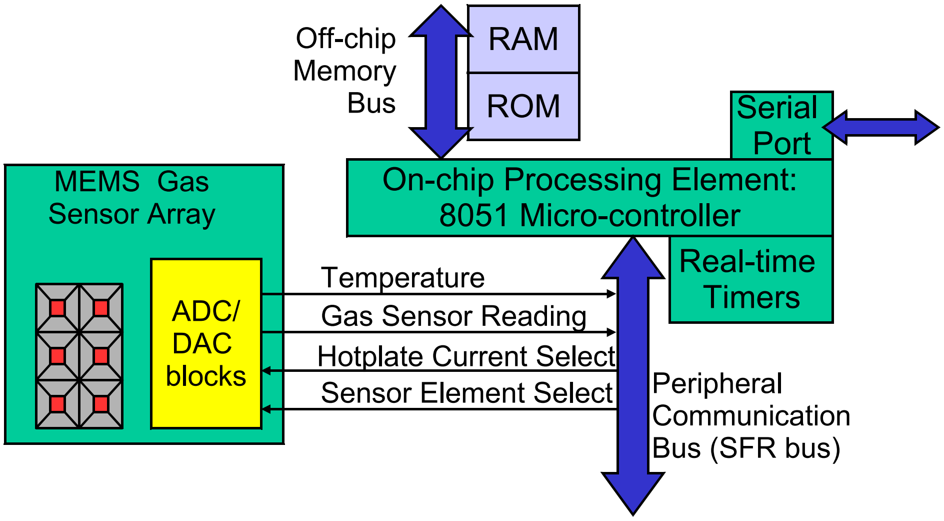

Figure 3. System topology for the integrated gas sensor SoC. A gas sensor array is connected to ADC/DAC and multiplexing circuitry, which communicates with the microcontroller over an on-chip bus.

senses the voltage drop across the temperature sensor. Multiplexing circuitry enables the use of a single ADC, thus helping to reduce chip area. The ADC and DAC are connected to registers that can be memory-mapped to a system bus via control circuitry.

3.2 System Architecture

The system topology for the integrated MEMS gas sensor SoC is illustrated in Figure 3. It consists of a microhotplate array, an 8051 microcontroller, and on-chip interconnect. The 8051 supports a single-master on-chip Special Function Register (SFR) bus, to which the gas sensor array is connected, allowing programs to access the microhotplate array via memory-mapped I/O.

A high-speed cycle-accurate SystemC model of the microcontroller was created to facilitate hardware-software development and testing. The HDL implementation of the microcontroller was synthesized from a commercially available 8051 IP core. The primary functions of the microcontroller software include controlling each microhotplate, maintaining the appropriate surface temperature, and collecting gas sensor data. A control algorithm monitors the temperature sensor reading and varies the heater current to converge rapidly and stabilize at the required surface temperature. Gas sensor conductivity readings are quickly taken at each temperature. This last activity is simple timed data-gathering, with no feedback loop involved. The gathered data may be processed on-chip or transmitted by the SoC to a central location for remote analysis.

4. METHODOLOGY

There are many challenges inherent in the integrated modeling of a heterogeneous SoC. First, microhotplate behavior is dependent not just on electrical parameters but also on the heating and cooling of the microstructure. This is addressed by setting up a lumped parameter model that correctly models the coupling between power dissipation, heating, and the electrical resistance of the heater. Even when this is done, a problem is posed by the fact that the behavior of analog and MEMS components is best represented by differential equations, not by the discrete-time event-based state machines used for digital simulation in SystemC. This is solved by expressing microhotplate behavior in discrete time, so that numerical methods can be applied, and then integrating this efficiently into SystemC's network-of-communicating-processes model of computation. In addition, the values of the various simulation parameters must be known to enable accurate system modeling.

There are thus four major issues that need to be addressed: modeling the MEMS microhotplates, integrating these models with SystemC, improving simulation efficiency, and obtaining the values of various component parameters. The remainder of this section discusses each of these in detail.

4.1 Electrical And Thermal Modeling Of MEMS Microhotplates

The work presented in this paper focuses on modeling the electrothermal aspects of the microhotplate, not the electrochemical gas-sensing aspects of the metal oxide thin film. A MEMS microhotplate can be modeled using a lumped analytical model incorporating the following state variables:

- Polysilicon heater power dissipation (*P*).
- Microhotplate surface temperature (*T*), measured using temperature sensor resistance.
- Ambient temperature (T_0).
- Microhotplate thermal resistance (R_{th}).
- Microhotplate thermal capacitance (C_{th})
- Polysilicon heater current (*I*), controlled by writing to a DAC register.
- Polysilicon heater electrical resistance (R_e).
- Polysilicon heater temperature coefficient of resistance (TCR or α).

Of these R_{th}, C_{th} and α are treated as constant for a given microhotplate structure. System behavior can be expressed as a set of differential equations in these variables. Second-order effects in microhotplate behavior, such as the slight (less than 5%) variation of R_{th} with temperature, are not currently modeled.

The thermal equation governing heat flow is:

$$P = \frac{(T - T_0)}{R_{th}} + C_{th}\frac{d(T - T_0)}{dt} \qquad \text{... (1)}$$

Where *t* represents time. The heater electrical power dissipation can be written simply as:

$$P = I^2 R_e \qquad \text{... (2)}$$

And the heater electrical resistance varies with temperature as:

$$R_e = R_{e0}(1 + \alpha(T - T_0)) \qquad \text{... (3)}$$

Taking $T' = T - T_0$, we use the above equations to obtain:

$$\frac{dT'}{dt} = \frac{I^2 R_{e0}(1 + \alpha T') - (T'/R_{th})}{C_{th}} \qquad \text{... (4)}$$

which is a first-order Ordinary Differential Equation (ODE).

Systems of differential equations are most commonly solved using numerical methods, which have a wide range of applicability. However, the above equation is simple enough to have an exact analytical solution. More complex systems, such as a collection of distributed heat sources on a chip [5, 6], typically require numerical analysis. For this study, we used the exact solution but, for purposes of completeness, also ran on the model with the numerical solution to measure the effect on runtime. The two mechanisms produce equivalent results, with the exact solution requiring less computation. Their impact on simulation speed is discussed in 4.3.

The Euler Forward Method for numerically solving such ODEs involves using a discrete-time representation of Equation 4 being used to derive microhotplate surface temperature at time-step *n+1* from the state variables at time-step *n*.

$$T'_{n+1} = T'_n + \left(\frac{I^2 R_{e0}(1 + \alpha T'_n) - T'_n/R_{th}}{C_{th}}\right)\delta t \qquad \text{... (5)}$$

This computation can be performed at runtime with the microhotplate implemented as a SystemC module with the parameters defined at instantiation. A SystemC process calculates and updates the state variables at each time-step. Since a microhotplate has a separate SystemC process, its time-step size can be varied independently of the time-step size used for other system components. In this case, the microcontroller runs on a 10ns time-step (a 100 MHz core clock frequency), while microhotplate simulation reaches convergence at a 100μs or smaller time-step. This is because the thermal time constant of the microhotplate ($\tau = R_{th}C_{th}$) is typically of the order of milliseconds, and time-steps of $\tau/10$ or smaller tend to converge. Note that the time-step chosen must be sufficiently small to ensure that the numerical solutions obtained are stable and convergent (the error increases with the square of the time-step in Euler Forward Iteration), yet not so small that too much simulation time is spent modeling the MEMS component, impeding system simulation.

An exact analytical solution to Equation 4 (in terms of T_n and t_n) is given by:

$$T'_{n+1} = T'_n e^{a(t_{n+1} - t_n)} + \frac{b(e^{a(t_{n+1} - t_n)} - 1)}{a} \qquad \text{... (6)}$$

$$a = \frac{\alpha I_n{}^2 R_{e0} - 1/R_{th}}{C_{th}}, \quad b = \frac{I_n{}^2 R_{e0}}{C_{th}}$$

This computation is performed in a similar manner at runtime. However, since this is an exact solution, each time-step may be arbitrarily large without significant loss of accuracy. The rest of this paper uses the exact solution unless otherwise specified.

4.2 Integration with SystemC

A SystemC simulation consists of a hierarchical network of parallel processes that exchange messages and concurrently update signal and variable values under the control of a simulation kernel. Signal assignment statements do not affect the target signals immediately, and the new values become effective only in the next simulation cycle. As shown in Figure 4, the kernel resumes when all the processes become suspended, either by executing a *wait* statement or upon reaching the last process statement. On resuming, the kernel updates the signals and variables and suspends itself again while scheduled processes resume execution. If the time of the next scheduled event is the current simulation time, the processes execute a *delta cycle*, in which signal and variable values are updated without incrementing the current time [12].

The microhotplate is modeled as a standard SystemC module. It does not require any changes to the SystemC kernel or library, and it obeys standard SystemC simulation semantics, running as a user-defined process. Each time it is invoked, the microhotplate simulation process calculates the amount of time elapsed since the last update, solves the system state equations accordingly, updates the state variables to reflect the new device state and finally suspends execution until it is invoked again by the kernel.

Each microhotplate has standard SystemC port/channel connections to the rest of the system. It communicates with actual microcontroller C programs compiled and loaded into the SystemC model of the microcontroller, rather than with mathematical idealizations of program behavior. In particular, system interrupts, computation time, microcontroller CPU states, and the busses are all cycle-accurate representations of the hardware being designed, validated against HDL simulations.

4.3 Simulation Efficiency

Effective design-space exploration depends on high simulation speeds, making simulation efficiency a key design issue. This section explores three avenues for improving simulation efficiency: using more efficient SystemC processes, reducing SystemC kernel synchronization overheads, and using exact solutions to reduce the computational overheads involved in MEMS modeling. These provide a combined speedup of over 70x compared to simulation done without these techniques.

SystemC provides two kinds of processes: SC_METHODS and SC_THREADS [12]. The main difference in terms of simulation semantics is that an SC_THREAD's state is stored each time it is suspended and is restored each time it resumes, allowing local variable values to be preserved. A process resumes from the exact state it left on suspension. Storing and restoring state across invocations has obvious simulation overheads. SC_METHODs, on the other hand, are similar to function calls and restart from the beginning each time they are invoked. No local state is preserved across invocations. We found that storing required state as class data fields to allow the use of SC_METHODs instead of SC_THREADs raised simulation speed from 56 KIPS (thousand instructions per second) to 281 KIPS[1].

Code profiling indicated that synchronization of the main CPU process with the SystemC kernel at each suspend-resume was the performance bottleneck, since the processor module incremented the elapsed time after each executed instruction in order to be cycle-accurate. To eliminate this bottleneck, we

[1] All measurements of simulation speed were performed with a 1.6GHz Pentium M processor with 2MB of L2 cache and 768MB of PC2700 DDR SDRAM. Compilation was done using gcc 3.4.4 with -O2 compile-time flags.

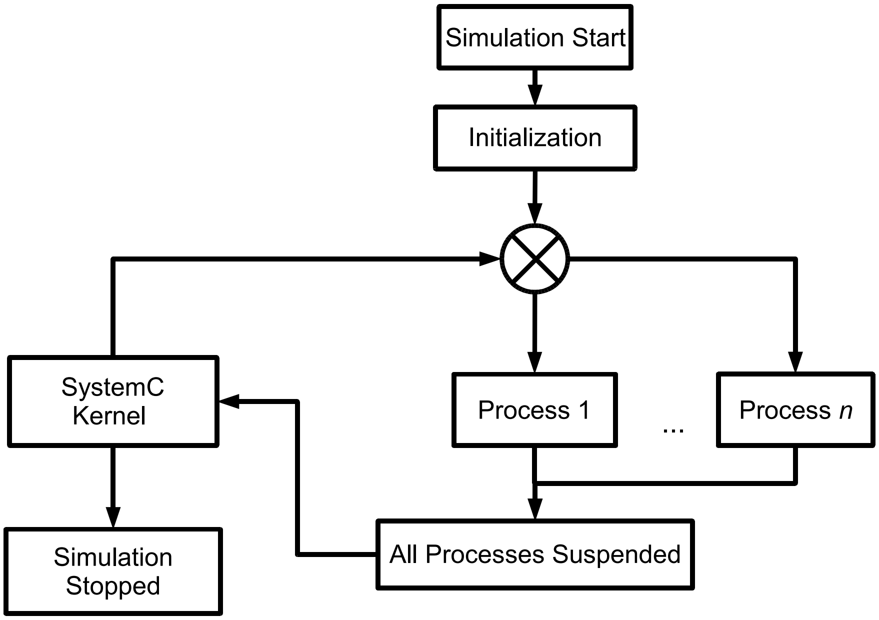

Figure 4. The execution semantics of SystemC. A number of interacting processes run until they end or execute a *wait* statement. Once all processes have run, the kernel updates all signals and variables before running ready processes again. The user can define specific conditions under which simulation should stop.

used *free-running simulation*, where the CPU continuously fetches and executes instructions while using an internal counter to keep track of elapsed cycles. This continues until an event that requires synchronization with the system occurs; events that trigger synchronization include interrupts, communication with system components that have separate processes, and reaching a user-defined limit on the maximum number of free-running cycles. When a synchronization event occurs, the processor informs the SystemC kernel of the time elapsed since the last synchronization (based on the internal counter that tracks elapsed cycles), updates any state required to ensure complete synchronization, resets the internal counter, and continues execution.

A processor usually spends much of its time fetching and executing instructions rather than communicating with other system components, so free-running simulation provides an elegant method for reducing overheads while maintaining cycle-accuracy. An upper bound can be put on the number of consecutive free-running cycles, causing regular synchronization regardless of other activity. We found that allowing up to 100 free-running cycles further sped up simulation from 281 KIPS to 2.89 MIPS. Allowing up to 4000 free-running cycles further boosted simulation speed to 4.17 MIPS, after which further increases led to no additional speedup. Profiling indicated that, after this optimization, the simulator was spending time in instruction processing and microhotplate modeling, not in synchronization.

Lastly, solving the differential equations governing microhotplate behavior also has a computational overhead. For a microhotplate with nominal time constant of 1ms, accurate modeling requires a time-step size smaller than 100μs while using the Euler Forward Method. Other, more sophisticated, numerical methods may be used that allow larger time-steps. Simulation efficiency is significantly higher when the exact analytic solution to the system of equations is used, since it allows the use of arbitrarily large time-steps without significant loss of accuracy. In practical terms, the microhotplate state only needs to be updated when the processor writes to it to change the DAC input or reads from it to find out the temperature, leading to lowered computational overheads. In the simulation framework presented here, system modeling proceeds at 4.17 MIPS using the exact solution and 3.71MIPS using the numerical solution (See section 4.1 for details on the two approaches)

4.4 Component Characterization

For characterization, the 8051 microcontroller IP core was synthesized to TSMC 0.25μm CMOS technology using Synopsys Design Compiler. Gate-level power simulation, with SAIF back-annotation [16] of activity was performed using Synopsys Power Compiler. Layout and back-annotation of delays and capacitance were performed using Tanner L-Edit. The microcontroller has a simple two-state power model, consuming 4.4mW when active (at 100MHz) and 0.25mW when idle. This state-machine based power model was observed to be accurate within 5% of gate-level power simulation for all programs run.

The values of the critical thermal and electrical parameters for the microhotplate — electrical resistance, temperature coefficient of resistance, thermal resistance and thermal capacitance — were the nominal design parameters and were verified experimentally on standalone hotplates fabricated through MOSIS, using the standard techniques described by Afridi et. al. [2, 3, 4].

5. RESULTS

The ability to model the complete system in detail enables designers to find answers easily and quickly to questions about overall system behavior. Such questions can range from how a microhotplate responds to a given input to finding out whether a given piece of code running on a microcontroller can meet desired performance parameters while controlling one or more MEMS devices. This section first presents a validation of the SystemC microhotplate model by comparing expected and observed behavior to a simple input. Further, this section discusses the observed results when a given temperature controller program is used to run a microhotplate and illustrates the kind of detailed observations that can be drawn from this. Lastly, it provides an example of how full-system simulation can help detect undesirable effects caused by valid low-level decisions that are suboptimal at the system level.

5.1 Model Validation

Validation of the microhotplate model was performed by using a function generator to apply a step voltage across a standalone microhotplate (implemented through MOSIS) and comparing the experimental data obtained against the SystemC model of such a device. Figure 5 shows such a comparison, and the high degree of correlation between simulation and experimental behavior is clearly seen. The simulated peak temperature is about 3% lower due to a small difference (caused by the slight temperature-dependence of thermal resis-

TABLE 1. Techniques for enhancing simulation efficiency, and their impact on performance. The exact analytical model for the microhotplates is used unless otherwise specified.

Technique	Simulation speed (MIPS)
SC_THREAD only	0.056
SC_METHOD only	0.281
SC_METHOD with up to 100 free-running cycles	2.89
SC_METHOD with up to 4000 free-running cycles	4.17
SC_METHOD with up to 4000 free-running cycles (Numerical model)	3.71

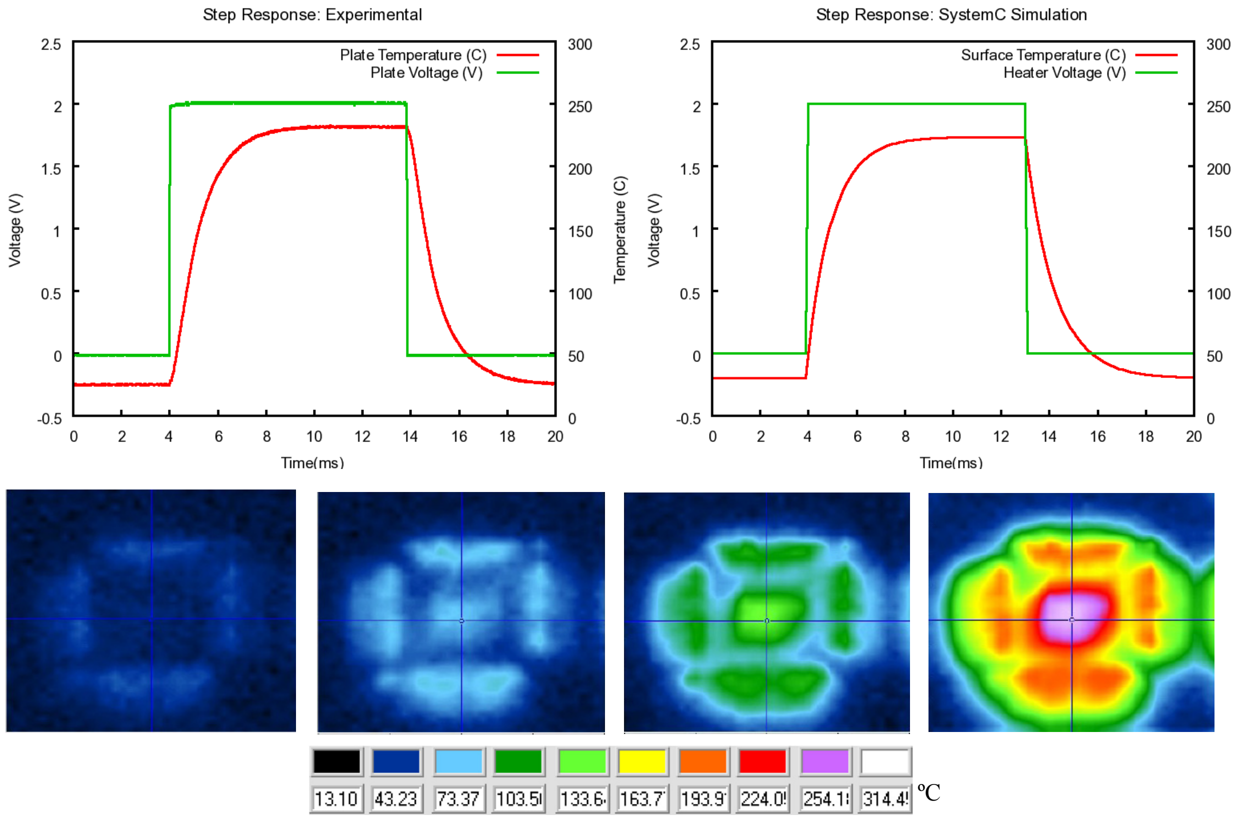

Figure 5. A comparison of experimental and simulated microhotplate behavior. A 2V voltage pulse is applied between 4 and 14ms. The observed changes in surface temperature are compared against those predicted by simulation. The plot on the left is “noisier” and less sharp simply because of the small, but unavoidable, experimental noise. The bottom strip shows a thermomicrograph sequence of a microhotplate structure heating up [1].

tance) between the simulated and observed values of R_{th}. Figure 5 also shows a thermomicrograph sequence of a MEMS microhotplate heating up, illustrating the high surface temperatures that can be attained over a localized area.

5.2 Simulation With a Controller Program

The above test provides crucial experimental validation for the microhotplate models used; however, system designers need to know how the system as a whole behaves when configured with a given topology and loaded with specific software. The results from SystemC simulation enable total SoC power dissipation and microhotplate behavior to be modeled in an integrated environment. This enables designers to observe the time-domain behavior of the entire system when running specific software.

To illustrate this, a test C program implementing a simple proportional controller was implemented, to control surface temperature in a single-microhotplate system. It was given a setpoint of 380ºC for 20ms followed by a setpoint of 200ºC for a further 20ms, after which the program turned the microhotplate off. This simplified program was chosen for illustration here because it is representative of the control aspects of the software stack used for microhotplate-based gas sensor applications.

Figure 6 illustrates the output of the simulation. The X axis represents system time in milliseconds, while microhotplate temperature, power, and current, as well as microcontroller power dissipation, are suitably scaled to be shown on the Y axis. The results shown here are based on a SystemC simulation incorporating both the cycle-accurate behavior of the microcontroller and the electrothermal behavior of the microhotplate. A discussion of the behavior of the four variables plotted follows.

The microhotplate heater current, directly controlled by the microcontroller, changes step-wise, since it is incremented in discrete steps through a DAC. The microhotplate power dissipation changes step-wise when current changes and smoothly at other times. It does not change in fixed size steps, since a) It is proportional to the *square* of the current and b) It depends on the electrical resistance of the polysilicon heater, which increases linearly with surface temperature. For example, between 3ms and 5ms, heater current is *constant*, yet microhotplate power dissipation rises smoothly in a classic asymptotic exponential curve. This is because the steadily increasing temperature raises the electrical resistance of the polysilicon heater (Equation 3), leading to an increase in power dissipation at a constant current. Note that the large change in microhotplate

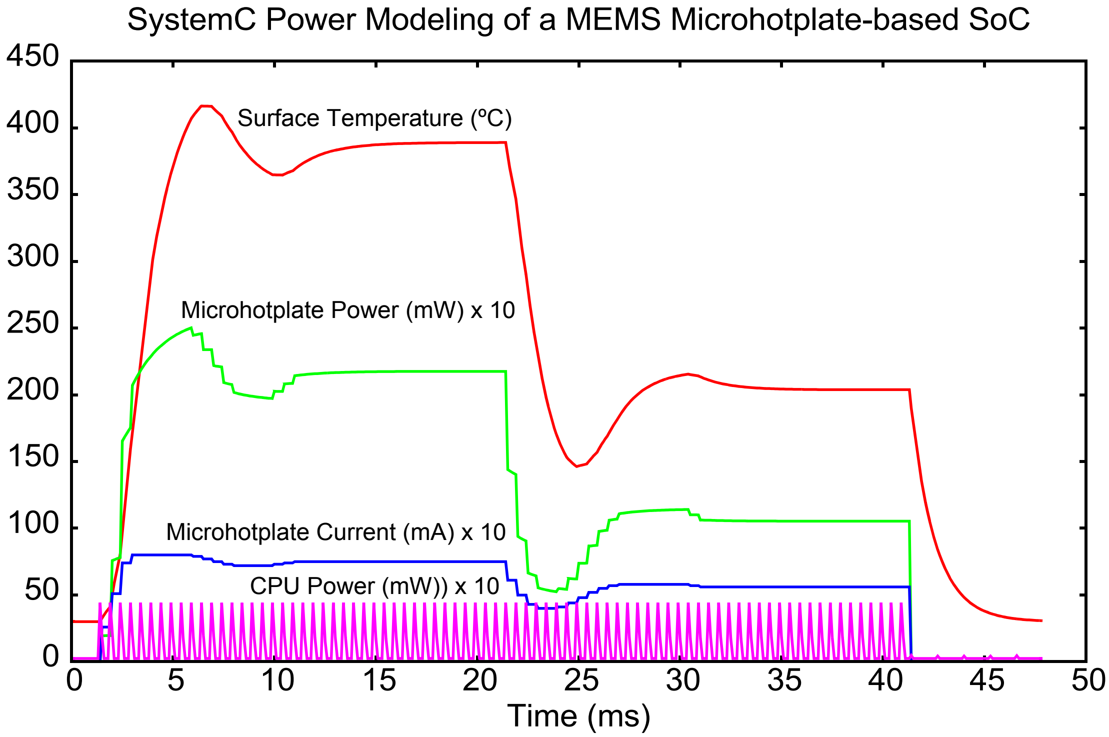

Figure 6. An example illustrating the use of integrated functional, power and thermal modeling in a heterogeneous system. The X axis represents system time in milliseconds, while other variables are suitably scaled to be shown on the Y axis. A feedback loop, in the form of a proportional controller program, is loaded into the SystemC simulator and given a surface temperature setpoint of 380ºC for 20ms, followed by a setpoint of 200ºC for another 20ms, and finally turns the microhotplate off at t=40ms.

power dissipation around 22ms corresponds to only a small variation in heater current, since they are quadratically related.

The microhotplate surface temperature changes smoothly, since the thermal capacitance of the microhotplate causes the temperature to be continuous in time, always varying smoothly. Around t=5 ms, the surface temperature first overshoots and then undershoots the setpoint of 380ºC before settling at it. This overshoot-and-stabilize behavior is typical of the proportional controller algorithm used. The same is true of the undershoot at t=25 ms. At t=40ms, the controller sets the heater current to 0, immediately dropping microhotplate power to 0. However, surface temperature follows a decaying exponential as it cools off, finally stabilizing at 30ºC, since that was set as the ambient room temperature in the simulation.

The “jagged” nature of the CPU power plot is due to the CPU waking up periodically in response to a timer interrupt, performing the computation required to run the controller, sending control signals to the microhotplate DACs, and then going into a low-power mode. The tiny “blips” in CPU power dissipation after t=40ms are due to interrupts being processed, but in these instances no feedback control computations are performed, leading to a much shorter active CPU duty cycle.

5.3 System-Level Effects of Low-Level Design Decisions

At the microhotplate design level, using a controlled-current or a controlled-voltage source to drive the heater is an implementation detail, with circuit-level concerns typically deciding the choice of one over the other. However, we found that such decisions could significantly impact system-level behavior, with integrated SystemC modeling of the MEMS device helping both to detect such behavior and to ensure optimal design.

In the previous example, a controlled current source was used to drive the microhotplate heater. However, exploring the design space using SystemC indicated that the behavior would be very different, exhibiting much less overshoot-undershoot behavior, if the hotplates heaters were driven by a controlled *voltage* source rather than a controlled *current* source. At first glance, this seems counter-intuitive, but it is borne out by the SystemC simulation (see Figure 7).

The reason that this seemingly minor device-level design decision has broader impact is that heater resistance increases with temperature, so power dissipation increases with temperature at constant current; but at constant voltage, microhotplate power dissipation *falls* with increasing temperature (since $P = I^2R = V^2/R$). A current-driven microhotplate thus has a small implicit *positive* feedback effect: higher power dissipation drives temperature up, which tends to cause a rise in power dissipation. A voltage-driven microhotplate, on the other hand, has a small implicit *negative* feedback effect: higher temperature causes higher heater resistance, which tends to reduce power dissipation. These loops interact with the overriding feedback loop implemented in software.

Figure 7 shows system behavior for the same control program when heater voltage, and not current, is directly con-

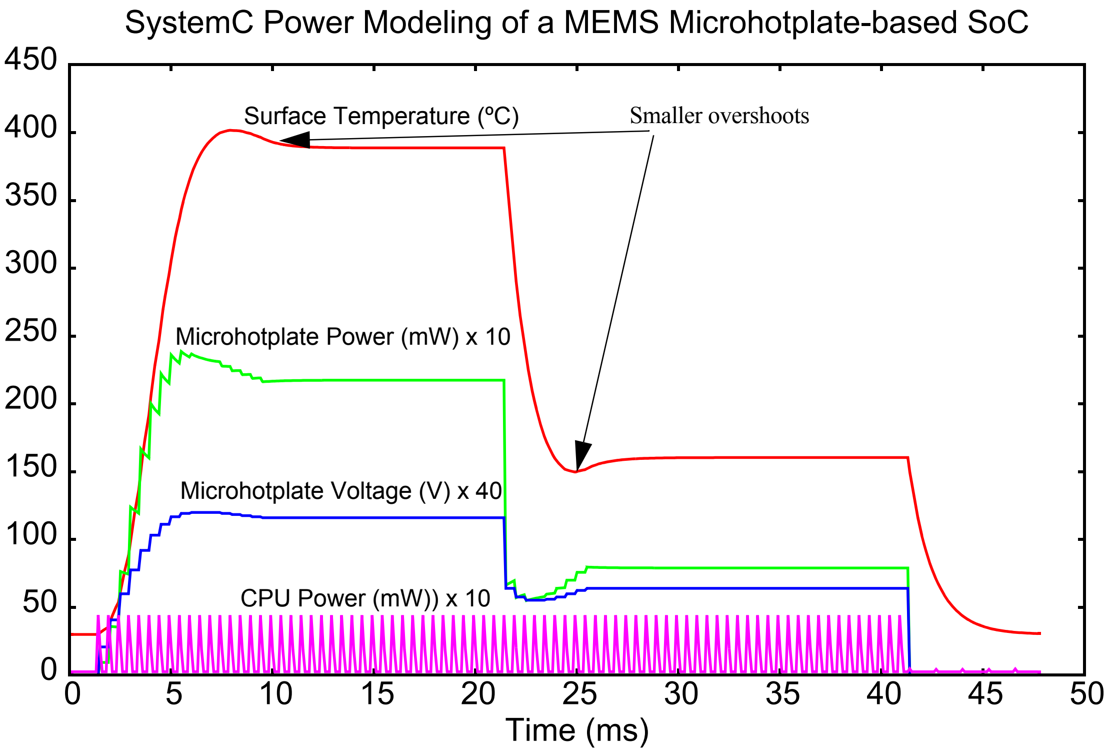

Figure 7. SystemC Power and Thermal modeling of a microhotplate driven by controlled-voltage source rather than a controlled-current source. A small inherent negative feedback loop introduced leads to much more stable behavior, with much smaller overshoots and a faster settling time (compare with the overshoot-undershoot behavior in Figure 6).

trolled. The negative feedback loop leads to significantly more stable behavior, with considerably smaller and fewer overshoots. Also note that power *decreases* when voltage is constant and temperature is rising (around 7ms). This is because the rising temperature raises microhotplate resistance, and the power dissipated is inversely proportional to this resistance. The increased feedback stability was an easily-overlooked factor that can now be used to guide system-level, component-level, and software-level decisions for the SoC presented here. Unanticipated feedback behavior is a serious issue, since, depending upon severity, it can lead to suboptimal performance or oscillatory behavior and may neccessitate software fixes or even require the system to be modified and re-fabricated.

Integrated simulation of both digital and MEMS components proved to be an extremely useful tool in the hardware-software co-design for this SoC:

- Full-system simulation results were among the inputs in the decision to use voltage-driven, rather than current-driven, microhotplates.
- Integrated simulations were used to assess system robustness while facing process variations in device parameters.
- Running the software stack under realistic conditions enables more thorough testing, leading to better defect detection *before* the system is fabricated.
- Interrupt routines, timer settings, operating frequency, I/O and control algorithm parameters can be better optimized when realistic simulation results are available. In the absence of these, designers need to allow larger margins of error to account for the uncertainty in the final performance of the system.

Complex system-level interactions, such as those illustrated above, need to be taken into account by system, software, and component designers, and integrated modeling of both microcontroller and MEMS device behavior in SystemC enabled precisely that.

6. CONCLUSION

This paper describes an approach for modeling the functionality, power, performance and thermal behavior of a complex class of MEMS components — MEMS microhotplate-based gas sensors — within a standard SystemC design framework. The system components modeled include both standard digital components (microprocessors, busses and memory) and MEMS devices.

The contributions made in this work include the first SystemC models of a MEMS-based SoC, the first modeling of MEMS thermal behavior in SystemC, techniques for attaining significant (over 70x) improvement in simulation speed and a detailed case study of the application of the proposed models and techniques to a real system. It also provides insights on how device-level design decisions can have system-level impact, which can be captured and addressed through accurate modeling of the entire system, including non-digital components.

Future work will include more detailed hotplate models that include second-order effects, analytical studies of microhotplate feedback behavior and application of the presented techniques to other components of heterogeneous SoCs.

ACKNOWLEDGMENTS

The work of Bruce Jacob is supported in part by MURI award AFOSRF496200110374, the Laboratory of Physical Sciences in College Park, MD, Cray, NIST, IBM, and the Department of Defense. The authors would also like to thank Neil Goldsman and George Metze for their contributions to making some of the work presented here possible.

REFERENCES

[1] AFRIDI, M., BERNING, D., HEFNER, A., SUEHLE, J., ZAGHLOUL, M., KELLEY, E., PARRILLA, Z., AND ELLENWOOD, C. Transient heating study of microhotplates using a high-speed thermal imaging system. In *Semiconductor Thermal Measurement, Modeling, and Management Symposium (SEMI-THERM)* (San Jose, CA, March 2002).

[2] AFRIDI, M., HEFNER, A., BERNING, D., ELLENWOOD, C., VARMA, A., JACOB, B., AND SEMANCIK, S. MEMS-based embedded sensor virtual components for SoC. In *Proceedings of the International Semiconductor Device Research Symposium* (2003).

[3] AFRIDI, M., HEFNER, A., BERNING, D., ELLENWOOD, C., VARMA, A., JACOB, B., AND SEMANCIK, S. MEMS-based embedded sensor virtual components for System-on-a-Chip (SoC). *Journal of Solid-State Electronics 48*, 10-11 (October-November 2004), 1777–1781.

[4] AFRIDI, M. Y., SUEHLE, J. S., ZAGHLOUL, M. E., BERNING, D. W., HEFNER, A. R., CAVICCHI, R. E., SEMANCIK, S., MONTGOMERY, C. B., AND TAYLOR, C. J. A monolithic CMOS microhotplate-based gas sensor system. *IEEE Sensors Journal 2*, 6 (2002), 644–655.

[5] AKTURK, A., GOLDSMAN, N., AND METZE, G. Self-consistent modeling of heating and MOSFET performance in 3-d integrated circuits. *IEEE TRANSACTIONS ON ELECTRON DEVICES 52*, 11 (November 2005), 2395–2403.

[6] AKTURK, A., GOLDSMAN, N., PARKER, L., AND METZE, G. Mixed-mode temperature modeling of full-chip based on individual non-isothermal device operations. *Solid-State Electronics 49*, 7 (2005), 1127 – 1134.

[7] ALJUNAID, H., AND KAZMIERSKI, T. J. SEAMS - a SystemC environment with analog and mixed-signal extensions. In *International Symposium on Circuits and Systems* (2004).

[8] BJORNSEN, J., AND YTTERDAL, T. Behavioral modeling and simulation of high-speed analog-to-digital converters using SystemC. In *International Symposium on Circuits and Systems* (2003).

[9] CAI, L., AND GAJSKI, D. Transaction Level Modeling: An overview. In *Intl. Conf. of Hardware-Software Codesign and System Synthesis (CODES+ISSS)* (2003).

[10] DOBOLI, A., AND VEMURI, R. Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22*, 11 (November 2003), 1504 – 1520.

[11] FREY, P., AND O'RIORDAN, D. Verilog-AMS: Mixed-signal simulation and cross domain connect modules. In *IEEE/ACM International Workshop on Behavioral Modeling and Simulation* (2000).

[12] GRÖTKER, T., LIAO, S., MARTIN, G., AND SWAN, S. *System Design With SystemC*. Kluwer Academic Publishers, 2002.

[13] INTERNATIONAL ROADMAP COMMITTEE AND ITWGS. International technology roadmap for semiconductors (ITRS): Design. Tech. rep., SIA, 2003.

[14] OPEN SYSTEMC INITIATIVE. *SystemC 2.0.1 Language Reference Manual*. Open SystemC Initiative, 2003.

[15] PARAMESWARAN, M., ROBINSON, A. M., BLACKBURN, D. L., GAITAN, M., , AND GEIST, J. Micromachined thermal radiation emitter from a commercial CMOS process. *IEEE Electron Device Letters 12*, 2 (1991), 57–59.

[16] SYNOPSYS. *Power Compiler User Guide*. Synopsys Inc., January 2005.

[17] VACHOUX, A., GRIMM, C., AND EINWICH, K. SystemC-AMS requirements, design objectives and rationale. In *Design Automation and Test in Europe (DATE)* (2003).

[18] ZHANG, T., CHAKRABARTY, K., AND FAIR, R. Integrated hierarchical design of microelectrofluidic systems using SystemC. In *International Conference on Modeling and Simulation of Microsystems* (2002).