An Analogue and Mixed-Signal Extension to SystemC

Authors' names have been removed for blind review
Affiliation and address

***Abstract*— This paper presents a new methodology that enables extensions of SystemC to the analogue domain and allows modelling of mixed-signal and mixed energy-domain systems at arbitrary levels of abstraction. The new language constructs support analogue system variables, analogue components and user defined ordinary differential and algebraic equations. Support for digital-analogue interfaces has been provided for smooth integration of digital and analogue parts. Associated issues such as dealing with extremely small and zero time-step sizes have been addressed. A novel implementation of the lock-step mixed-signal synchronisation method to integrate the analogue kernel with the digital one has been proposed. Operation of the extended, mixed-signal simulation platform, named SystemC-A, is demonstrated using a suite of numerically difficult AMS examples including a practical, mixed-signal example of a PLL frequency multiplier with large-signal noise and jitter.**

***Index Terms*— System-Level Modelling, Mixed-Signal Systems, SystemC, Synchronisation, Lock-Step, Computer-Aided Design.**

I. Introduction

Design complexity and demanding time-to-market constraints have led to considerable challenges in the development of electronic design methodologies and Computer Aided Design (CAD) tools. The possibility to integrate a complete complex System on a single Chip (SoC) has started a new era [1]. SoC has created a need for powerful CAD tools and methodologies which would be capable of integrating information from multiple heterogenous sources (analogue parts, processors, RAM, ROM, etc.) and have the ability to work at high level of abstractions [2]. Advances in integrated circuit technology have been the driving force behind the extensive development of digital HDLs, whilst Analogue and Mixed-Signal (AMS) high level modelling is lagging behind and leaving the design community with immature design methodologies [3]. This has created a gap in the design of the two different parts which threaten the rate of production. Despite the success of digital systems, analogue circuitry is still needed in particular in modern ASICs (Application Specific Integrated Circuits) designed for telecommunication, wireless and computer network systems [3]. The design of analogue blocks in SoC and ASIC is still done to a large extent manually which requires time and effort together with specific skills [4]. All of these advances and challenges have put a pressure on CAD of AMS systems to keep up with the success of the pure digital CAD.

The Electronic Design Automation (EDA) industry and academia were trying extensively to meet these needs [5]. One common approach is to model and simulate digital and analogue systems with digital HDLs and analogue design tools respectively. Digital HDLs such as VHDL [6], Verilog [7] and SystemVerilog [8] are used to model digital systems while analogue design or general purpose equation solving tools such as SPICE [9] and MATLAB [10] dominate in the modelling of analogue systems at different abstraction levels. Another approach is to extend classical HDLs intended originally to model digital systems to the analogue domain such that both parts are modelled and simulated in a single environment, VHDL-AMS [11] and Verilog-AMS [12] being the flagship examples. The third approach is to readapt software programming languages such as JAVA [13], C/C++ [14] and UML [15] to model analogue and digital hardware. It can be done by adding special language constructs for hardware description and defining hardware description semantics.

The recent trend is toward C++ based modelling [16], [17] either through libraries or abstractions. C/C++ is already used by hardware engineers at system level to estimate system performances and verify functional correctness. There are various C/C++ based HDLs provided both by the EDA community, e.g. SystemC [18] and SystemVerilog [8], and the academia, e.g Handel-C [19] and SpecC [20].

SystemC [18] has enjoyed immense popularity since its introduction in September 1999 and gained a wide acceptance and support from the industry [21]. SystemC is a standardized modelling language intended to enable system level design and Intellectual Property (IP) exchange at multiple abstraction levels for systems containing both software and hardware components [22]. These features are required in SoC design but are not available in any existing HDLs. The latest SystemC version (V2.0.1) has been in use since 2003. Although it resembles existing HDLs and adds more features for digital modelling, it does not support AMS modelling as yet. A study group was established in 2003 [23] following a proposal submitted to the SystemC board of directors to form an Open SystemC Initiative OSCI working group tasked to develop AMS extensions to SystemC, named SystemC-AMS. This is proving to be a difficult task as many associated research issues still remain unresolved.

This paper addresses some of these issues and presents new methodologies that enable an extension of SystemC to the analogue domain. The extended language, named SystemC-A contains new language constructs and associated numerical implementations to allow modelling of general mixed-signal and mixed energy-domain systems at arbitrary levels of abstraction. SystemC-A supports analogue system variables, analogue components and user-defined Ordinary Differential and Algebraic Equations (ODAEs), as well as digital-analogue interfaces to assure smooth integration of digital and analogue parts.

II. Related Work

Popular HDLs, such as VHDL [6] and Verilog [7] have already been extended to provide AMS modelling [11], [12]. With regards to SystemC, a number of research publications have presented a variety of frameworks and methodologies related to analogue and mixed-signal applications. These frameworks are usually developed around additional C++ classes, methods and libraries. Some of the new frameworks provide, short of defining a fully-fledged AMS extension, limited analogue modelling capabilities. For instance, O'Nils et al [24] presented a methodology for quantification of noise coupling in AMS systems. Starting from a behavioural model of the system captured in SystemC, wrappers are added to each block. These wrappers add an estimated power consumption model for each block, which is triggered by events. The simulation results compared with circuit simulations in SPICE [9] showed that their approach is two orders of magnitude faster than SPICE.

An AMS simulation framework is presented by Bonnerud et al [25] for simulation of Analogue to Digital data Converters (ADC). The framework contains a C++ mixed-signal module library that includes a set of customizable primitive, compound modules and testbenches. They have implemented a clocking scheme for the scheduling of the AMS blocks to avoid multiple executions of these blocks due to the SystemC kernel. They have illustrated the usability of the framework by applying it to two case studies of pipelined ADCs with background calibration, achieving comparable accuracy to that of MATLAB [10]. Another AMS framework was presented by Conti et al [26] suggesting a new implementation of analogue blocks using analogue macromodels. It is a module composed of two kinds of threads, the calculus thread and activation threads, one for each input module. They have used two AMS examples to validate their framework, an oscillator made up of inverter chain and a complex mixed-signal fuzzy controller. The results were compared to other tools such as SPICE [9] and Spectre [27] achieving good results.

Grimm et al [28] presented a top-down modelling and simulation methodology based on a refinement process and implementing a library. The design methodology refines an executable specification to concrete AMS architecture through three levels of refinement, executable, computation accurate model and pin accurate model. The execution of the analogue processes is not controlled by the discrete event kernel; however the execution is controlled by a coordinator interface. For complex analogue processes, they used an external analogue simulator such as Saber [29] or SPICE [9]. The methodology was evaluated by designing a PWM controller. Another design methodology presented by Romberg and Grimm [30] based on refinement process started with a system specification captured by a new graphical design notations called HyCharts. The authors claim that Hycharts, which are based on formal semantics, precisely capture the mixed, continuous/discrete behaviour.

Einwich et al [31] presented a framework which supports signal processing dominated applications. The framework is based on an analogue extension for linear ODAEs and frequency domain modelling. The analogue extension includes a library of electrical circuit components and transfer functions. The synchronisation between the synchronous dataflow and linear continuous time is using a fixed time step. The concept was illustrated with the design of a telecommunication system including digital hardware and software and an analogue filter. Simulations were about 20 times faster than those of Saber. A framework called AnalogSL is presented by Grimm et al [32] for the creation of behavioural models of analogue power drivers. AnalogSL provides classes of components such as resistors, capacitors, coils and transistors which can be instantiated to form a netlist and then simulated by an algorithm for linear ODAE. The coupling of the analogue behavioural model with discrete-event simulators corresponds to the coupling of different processes in discrete simulators.

A mixed-signal SystemC design environment has recently been proposed [33] for behavioural modelling, simulation, and performance evaluation of microelectromechanical and microelectrofluidic SoCs. Continuous-flow systems such as microvalves, micropumps and channels are modelled by ODAEs in SystemC. The continuous equations are solved by using ODAEs solvers with SystemC, such as derivative and integral, and add them into a SystemC component behaviour model. They implemented a higher frequency clock to provide a series of time intervals for more accurate ODAEs solutions.

Most of these frameworks are application specific extensions [24], [25], [31], [28], [33], or abstraction level specific ones [24], [25], [26], [32]. What is still missing in the published research, is a general approach to AMS extensions in SystemC able to handle a wide class of non-linear dynamic systems.

III. Extension Outline

The general AMS extension proposed here is superset of SystemC. The extension covers a wide area of analogue and mixed-signal modelling as indicated on the comparative map shown in Fig. 1.

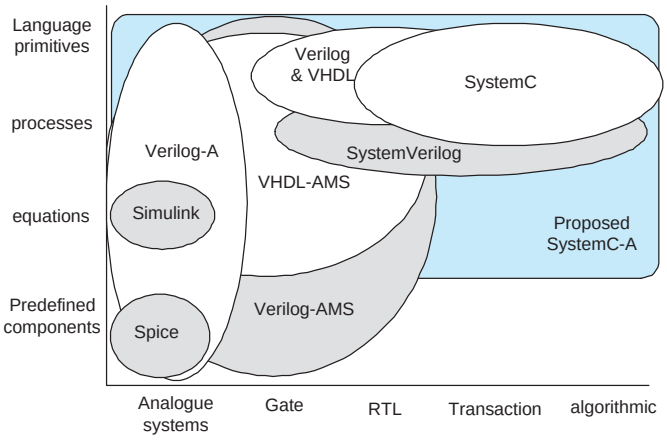

Fig. 1. The proposed SystemC-A on the map together with the current languages at different abstraction levels.

Support for various abstraction levels: In addition to modelling at various abstraction levels provided by the SystemC digital platform, the proposed extension provides extra

abstraction levels which are specific to analogue systems. Although our main focus is on the system level to tackle complexity, the extension is able of modelling at lower levels, specifically the netlist level and analogue behavioural level.

New syntax elements and classes extending SystemC to the analogue domain: Modelling of an analogue system requires a set of ordinary differential and algebraic equations (ODAEs). ODAEs should be easy to define, automatically built and updated, and then numerically solved. New language constructs are implemented for this purpose, such as systems variables, analogue components and user defined equations. Digital-analogue interfaces are defined for easy and smooth integration of the analogue and digital parts. Associated issues such as dealing with small time step sizes are addressed.

An implementation of continuous-time analysis for system level modelling: Necessary continuous-time analyses suitable for mixed-signal system level modelling are implemented and a flexible manner, which allows the user to define complex types of time-domain simulations such as large signal noise analysis. Noise analysis in a mixed-signal context is difficult to implement with traditional circuit simulators as it must be evaluated in the presence of large signal behaviour [34].

An analogue kernel implementation: An efficient analogue kernel is developed and integrated to synchronise with the existing SystemC digital kernel. The algorithms used in the development of the analogue kernel are highly tuned and optimized using a suite of numerically difficult analogue and mixed-signal examples. Examples range from small sets of ODAEs such as a Lorenz chaos model to non-trivial mixed-signal systems such as a switched-mode power supply and a phase-locked loop.

IV. AMS Extension Elements

The AMS syntax has been designed to facilitate modelling assuming minimal programming knowledge of a SystemC-A user. A number of modelling styles are possible, for example a SPICE-like net-list or VHDL-AMS-like simultaneous equations with a hierarchy of interconnected blocks as in any HDL. The SystemC-A nonlinear analogue system is modelled by a set of nonlinear ODAEs as shown in Eqn (1).

$$\mathbf{f}(\mathbf{v}(t), \dot{\mathbf{v}}(t), t) = \mathbf{0} \quad t \geq 0, \ \mathbf{v}(0) = \mathbf{v}_0 \tag{1}$$

where $\mathbf{f} : R^N \times R^N \times R^1 \rightarrow R^N$ is a vector function, $\mathbf{v}(t) \in R^N$ is a vector of unknowns, $\dot{\mathbf{v}}(t)$ is a vector of the unknown derivatives with respect to time, N is the number of unknowns and t is time.

A. Analogue system variables

In the set of ODAEs Eqn (1), the analogue system variables $\mathbf{v}(t)$ are the unknowns. The C++ concept of inheritance is used to define various types of analogue system variables, such as nodes, currents and free variables. In the proposed extension, they represent a hierarchy of system variables, all derived from an abstract base class. Currently only three types of variables derived from the base class have been defined, and this proved enough to model the application examples presented later. The hierarchy can be extended further to accommodate other types of applications.

B. Analogue components

Analogue circuit components provide equations to describe analogue behaviour. Similarly to the system variable hierarchy, components are derived from an abstract base class which contains a virtual *build* method invoked by the analogue kernel. A sample component hierarchy is illustrated in Fig. 2. It includes examples of SPICE-like circuit elements such as resistor, capacitor, inductor, diode and various types of autonomous sources. Arbitrary differential and algebraic equations can be included as user-defined components.

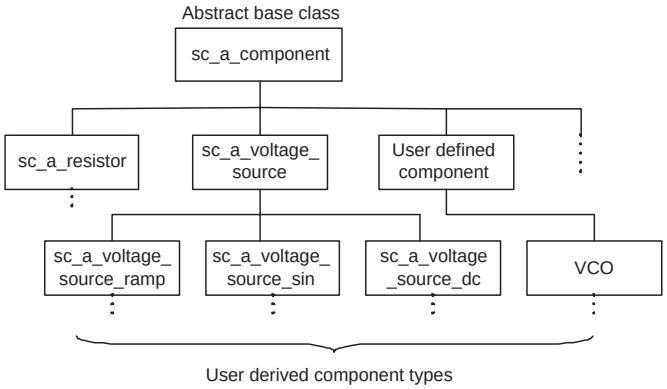

Fig. 2. Analogue components inheritance hierarchy.

A typical component class instantiation contains a pair of node pointers and a value. For example, a capacitor can be instantiated as follows:

```
sc_a_capacitor *c1= new sc_a_capacitor("c1", nodeA, nodeB, C);
```

where c1 is the component's instance name, nodeA and nodeB, are names of analogue system variable objects of type *node* and represent the two terminals to which the capacitor is connected, and C is the capacitance. The component base class constructor attaches each newly created instance to a global linked-list of system components to form a connected circuit. The list is used at the matrix build time. All the components are scanned to invoke their build functions. A netlist of an analogue circuit can be constructed by declaring system variables of type *node* and analogue components as shown in the following code of the loop filter in a phase-locked loop. Fig. 3 shows its corresponding schematic.

```
n2 = new sc_a_node("n2");
n0 = new sc_a_node("0");
n1 = new sc_a_node("n1");
sc_a_currentS *I1 = new sc_a_currentS("I1", n1,n0,&Iin);
sc_a_capacitor *c1 = new sc_a_capacitor("c1",n1,n2,3e-9);
sc_a_resistor *r1 = new sc_a_resistor("r1",n2,n0,1e3);
sc_a_capacitor *c2 = new sc_a_capacitor("c2",n2,n0,4e-9);
```

C. Virtual build method

The *build* method, which specifies the analogue behaviour of a component, is a virtual method with a default body in the abstract component base class. Its code defines one

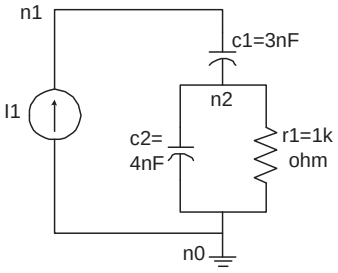

Fig. 3. The corresponding schematic of circuit description.

or more ODAEs. For example, Fig. 4 shows the capacitor representation as a SystemC-A component. The Figure shows the capacitor's differential equation, its representation after discretisation (S and X_n are discretisation operators), its Jacobian stamp and part of the corresponding build method in C++. The resulting Jacobian stamp conforms to the Modified Nodal Analysis formulation MNA. Calls to BuildRhs, build the differential equations for the capacitor (the right hand side RHS). Calls to BuildM, which build the corresponding Jacobian entries are optional. If these calls are not provided, the solver will build the Jacobian using a secant approach with finite difference approximation of the Jacobian entries. The entire equation set is formulated automatically at each Newton-Raphson iteration by scanning the components and invoking their build methods.

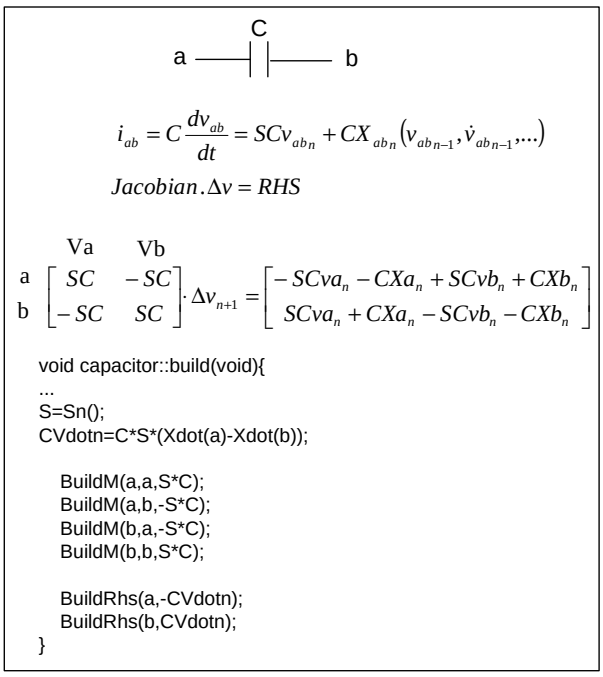

Fig. 4. Capacitor equation and build function.

V. Digital-Analogue Interactions

Connectivity between analogue and digital models requires special consideration since analogue and digital parts operate in two disparate mathematical domains. SystemC-A allows insertion of special interface objects directly between digital signals and analogue system variables. These interface models have no corresponding physical parts. This interfacing solution is similar to that adopted in Verilog-AMS.

interfaceDA is a SystemC-A module which contains an input port of type $bool$ and an output port of type $double$. A digital signal coming from the digital module is transformed into a smoothed analogue signal and directed towards the analogue module through the output port. The interface applies a smoothing function to prevent numerical difficulties in the analogue simulation due to abrupt transitions resulting from events on a digital node. The smoothing is done by Backward Euler integration with a very small time constant. This method is also capable of handling in a reliable manner the analogue solution in the presence of extremely small step sizes, including zero step sizes that occur in delta cycles.

interfaceAD is a SystemC-A module which takes analogue signal of type $double$ and produce a digital $bool$ signal. The criteria to generate a digital event are simple: if the specified analogue threshold E is exceeded, an event driving the output signal with the *true* value is generated. An event driving the output signal *false* is produced when the analogue input falls below the threshold. The digital part will react to this event if a concurrent statement reads this signal or if the sensitivity list of a process contains this signal.

VI. Analogue kernel

The analogue kernel consists of layers of several algorithms. Its main function is to build and solve the set of non-linear dynamic equations associated with the analogue part of the model. The flow chart in Fig. 5 shows the modelling and simulation flow details, illustrating how a general AMS system is solved by SystemC-A.

The component constructors in the user code will run first, initialising all the variables. Then, the simulation is started by executing the SystemC command (sc_start()) to formalise and solve the defined model. The Newton-Raphson NR nonlinear solver is first initialised (iteration $z = 0$) and then in every NR iteration, the component list is scanned to invoke the *build* functions which add the contribution of each component to the system matrix (J) and right-hand side vector (RHS). At each NR iteration the linear solver (based on the LU factorisation method) is called. Once the system equations are solved for incremental solution Δx, the solution x is updated ($x = x + \Delta x$) and tested for convergence. If the convergence criteria are not satisfied, the NR continues iterating. When convergence is reached, the analogue solver exits the NR algorithm and the kernel proceeds with the digital processes if required. Then, the next time point is calculated using the current Local Truncation Error (LTE) estimate. The solver then schedules an event at the next time point.

VII. Time Synchronisation Between Analogue and Digital Solvers

One of the most important problems in mixed-signal simulation is the time synchronisation between the event-driven digital simulation and the numerical integration in the analogue solver. Synchronisation is a key issue affecting the simulation

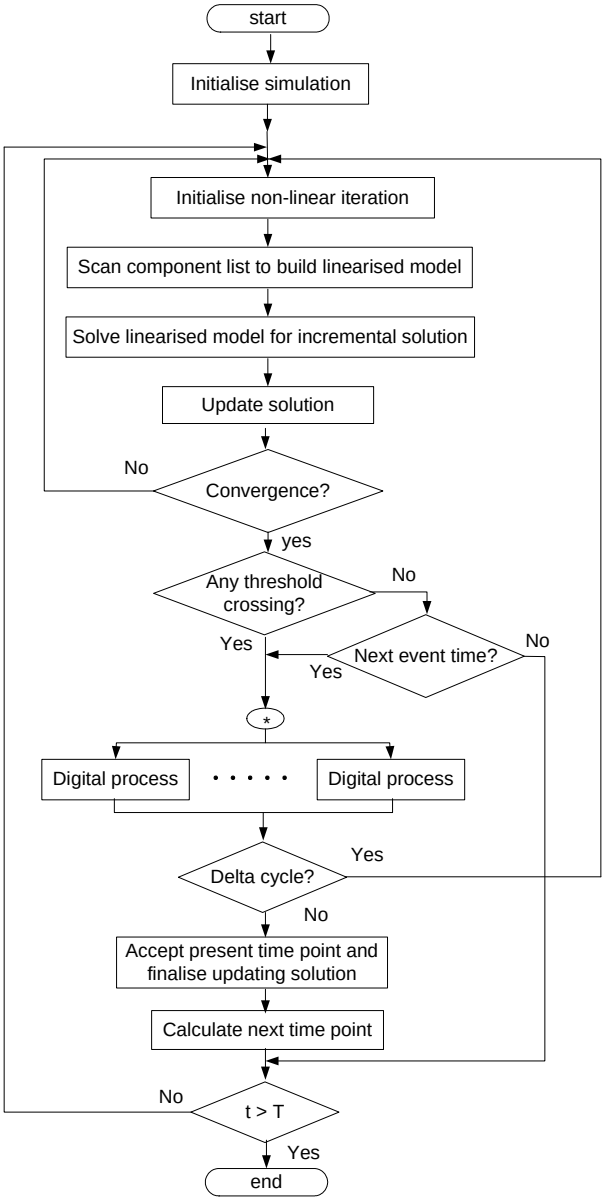

Fig. 5. Modelling and simulation flow chart of the SystemC-A solver.

speed and accuracy [35]. The idea of synchronisation is to modify the time stepping engine such that it fits into the event-driven paradigm [36]. The synchronisation is then accomplished by the event-driven simulator, which processes the events in the chronological order of their time stamps.

The major function of a mixed-signal simulator is synchronizing the two distinct algorithms so information can be exchanged without incurring errors or undue overhead. There are two fundamental approaches to time synchronisation at the analogue and digital interfaces, pessimistic and optimistic [35], [36], [37]. In the pessimistic approach, the simulators progress with the same time step. This approach ensures that there is no need for backtracking and no results are thrown away. One well-known example is the lock-step method. The optimistic approach allows each simulator to progress in time until it runs out of internal events. If an event from another simulator is generated before the end of this optimistic time interval, all results generated after that event are discarded. This means that simulators must be able to backtrack. Examples are Backplane [38], Ping-pong [35] and Calaveras [39]. The approach adopted here is the lock-step method.

The analogue simulator calculates the step sizes and the digital simulator uses these values. The analogue kernel advances until the current simulation time and, before suspending, schedules an event at the time equal to the current simulation time plus the next selected step size. This has been implemented by inserting a call in the SystemC kernel to the analogue kernel before the evaluation phase of the digital simulation cycle. This approach ensures that the SystemC kernel will make a step in time no larger than the analogue solver's step size. Since the analogue solver is controlled by the SystemC kernel, no synchronisation deadlock may happen.

Lock-step has been used by many commercial mixed-signal simulators, such as Lsim Power Analyst from Mentor Graphics and Pspice from Microsim Corporation. The lock-step pessimistic approach has been used in preference to the optimistic one because the prospect of wasting vast amounts of CPU time by the optimistic approach was considered too costly [35]. Also, the adopted method eliminates the need for backtracking and no results are thrown away. There were claims that the lock-step method produces ridiculously long runtimes [40]. However, this is true when the method is used to synchronise analogue and digital simulator from two or more different environments, because of the communication overhead. When two solvers are synchronised within the same environment, the lock-step approach is not expected to produce significant overheads.

VIII. Case Studies

This section presents three case studies, ranging from simple to complex, to verify functionality of the SystemC-A mixed-signal simulator. The Lorenz chaos is a system of simple ODAEs that demonstrate the modelling capabilities of the simulator at behavioural level. The Switched-Mode Power Supply (SMPS) and 2GHz Phase-Locked Loop(PLL) based frequency multiplier are non-trivial mixed-signal systems. Systems of this kind usually put standard SPICE-like simulators into difficulties because of the disparate time scales of their transients. In the case of the SMPS, the analogue transient in the output circuit is four to five orders of magnitude slower than that of the fast switching waveform in the digital controller. A typical simulation in a system of this kind might require few millions time points. Excessive CPU times often occur when the entire system is modelled on the circuit level. The capacity of SystemC-A to enable AMS modelling at behavioural level can vastly reduce simulation times when concepts need to be verified quickly and detailed circuit level modelling is not required.

A. Lorenz Chaos

The Lorenz chaos [41] system is most commonly expressed as three coupled non-linear ODAEs shown in Eqs. (2), (3) and (4)).

$$\dot{x} = \sigma(y - x) \quad (2)$$
$$\dot{y} = x(\rho - z) - y \quad (3)$$
$$\dot{z} = xy - \beta z \quad (4)$$

The SystemC-A model of the Lorenz chaos equations contains three free analog system variables $x,\ y$ and z with the following initial values: $x(0) = 0,\ y(0) = 5,\ z(0) = 25$. The *build* method which provides code for the equations and the corresponding Jacobian entries is shown in the code below. Fig. 6 shows the familiar Lorenz xz butterfly trajectory.

```
void LorenzChaos::build(void){
   sigma = 10.0, rho = 28.0, beta = 8.0/3.0;
   S = Sn();
   Xn = X(x), Yn = X(y), Zn = X(z);
   Xdotn = Xdot(x), Ydotn = Xdot(y), Zdotn = Xdot(z);

   // callbacks to build Jacobian
   BuildM(x,x,S + sigma);
   BuildM(x,y, -sigma);
   BuildM(x,z,0);
   // callback to build equation
   BuildRhs(x,-Xdotn + sigma*Yn - sigma*Xn );

   BuildM(y,x,-rho + Zn);
   BuildM(y,y,S+1);
   BuildM(y,z,Xn);
   BuildRhs(y,-Ydotn + rho*Xn - Yn - Xn*Zn );

   BuildM(z,x,-Yn);
   BuildM(z,y,-Xn);
   BuildM(z,z,S + beta);
   BuildRhs(z,-Zdotn + Xn*Yn - beta*Zn);
}
```

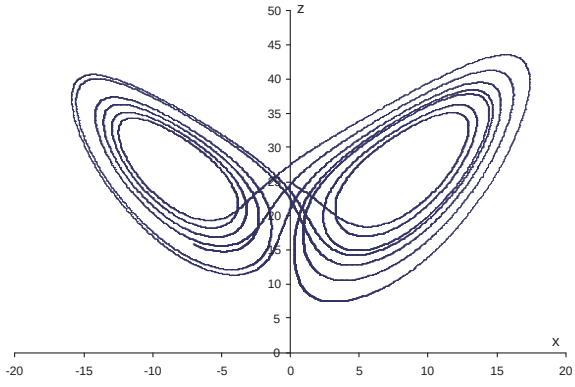

Fig. 6. Lorenz chaos xz butterfly trajectory.

B. Switched-Mode Power Supply

This example is a boost (step-up) [42] 3.3V regulator operating from a 1.5V source (Fig. 7). The analogue part of the model is an idealized boost SMPS with four basic components, namely a power semiconductor switch, a diode, an inductor, and a capacitor. The digital part is a pulse width modulator (PWM) controller. This SMPS uses a high frequency switch with varying duty cycle to maintain the output voltage.

The analogue part of the system was modelled at circuit level using analogue components from the simple analogue components library developed for SystemC-A as explained in Section IV-B, whereas the digital PWM is modelled as

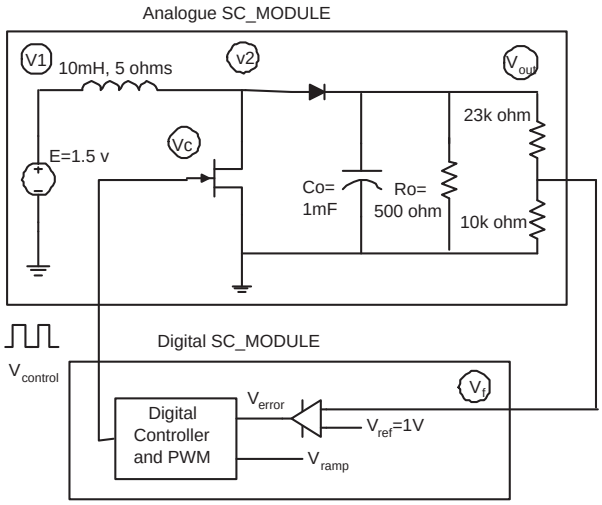

Fig. 7. Boost 1.5V/3.3V switched mode power supply with digital control.

a standard SC_MODULE at behavioural level. Simulations were carried on a Windows 2000 computer an AMD Athlon 1400 MHz processor and 512 MB RAM. The system was simulated for 0.2 seconds, when it reaches the steady state. Sample results at the steady state is shown in Fig. 8. The waveforms represent correspondingly the output ripple, error signal, inductor current, fast switching of $V_{control}$ and voltage at the transistor drain. Simulation statistics are shown in Table I.

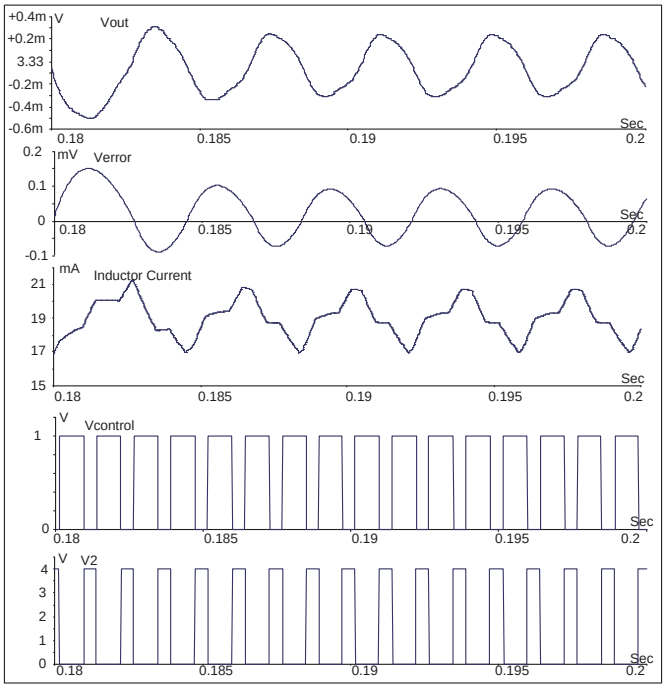

Fig. 8. SMPS simulation results for a 200ms time window in steady state of the boost SMPS working in continuous mode.

C. Phase-Locked Loop

Fig. 9 shows a block diagram of a digital PLL [43], [44] in a frequency synthesizer configuration. It consists of a reference

TABLE I
SMPS SIMULATION STATISTICS

Simulation time	200m Sec
Number of steps	2 Millions
CPU time	232.1 Sec

source, a phase/frequency detector, a charge pump, a loop filter, a voltage controlled oscillator (VCO), and a digital divider.

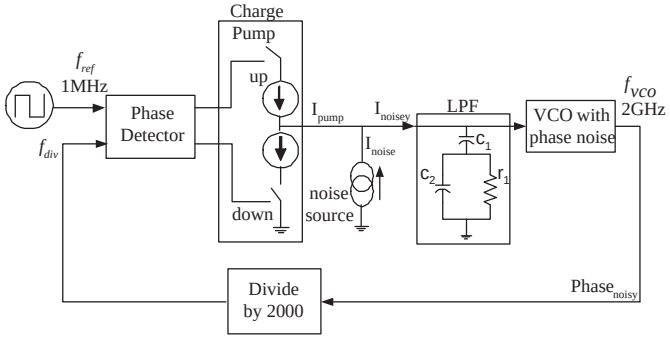

Fig. 9. 2GHz Phase Locked Loop with noise and jitter.

1) Noise Module: PLL noise behaviour is difficult to predict with traditional circuit simulators because of the repetitive large-signal switching events, which are an essential part of the PLL operation, hence noise performance must be evaluated in the time-domain [34]. Most classical simulators, SPICE being the best example, are not capable to simulate noise in PLLs as they can normally calculate small-signal noise around a quiescent operating point. Currently, the best suited simulator for PLL noise analysis is SpectreRF [34], which is capable of predicting the noise behaviour about a periodic operating point. In SystemC-A suitable large-signal noise modules can be constructed with no difficulty. For our PLL example a standard function was developed to generate a periodic process for a Gaussian white noise using the Box-Muller method to turn two uniformly distributed random sequences into two unity amplitude normal random X and Y ($mean = 0\ and\ variance = 1$) sequences which can be scaled to the required levels.

In this example two methods of modelling noise are implemented. The first method allows adding noise sources at any components and therefore provides a more accurate noise behaviour. Two different VCO models, presented in more detail in the following subsections, have been developed for both noise methods. In the first method the noise is injected by the controlled current source of the charge pump, although every PLL component is a potential noise source. The charge pump signal can be expressed as:

$$I_{noisy}(t) = I_{pump}(t) + I_{noise}(t) \tag{5}$$

Consequently, the phase of the VCO is subject to noise (referred to as the phase noise) which will manifest itself as jitter in the output waveform:

$$Phase_{noisy} = Phase(\theta + Jitter(\theta)) \tag{6}$$

In the second noise method, a noise source is added to perturb the VCO pulse directly. This perturbation represents the total effect of any noise source in the PLL. In this method [45] the total effect of noise is modelled in the VCO by scaling and adding the generated noise module output to the VCO phase where it is turned into jitter. The second method, although cruder than the first one, has the advantage of shorter CPU times since the system will be simulated with larger step sizes to cover variations in the VCO pulses every 0.5 ns.

2) VCO 1: Here the VCO frequency is the rate of change of the phase,

$$\dot{\theta}(t) = \frac{d\theta}{dt} = f(v) = f_c + df * V_{filter}$$

Where V_{filter} is the output voltage of the loop filter, f_c is the center frequency of the VCO, and $df = \frac{f_{max}-f_c}{V_{max}}$ is the VCO gain. The first VCO model is an analogue module in which the frequency is numerically integrated to compute the output phase which is used to generate the desired output signal. Output transitions are generated when the phase passes the value of 0.5 (the phase unit corresponds to a proportion of the duty cycle) in either direction. As SystemC-A allows different types of analogue descriptions to work together, the VCO was modelled here at behavioural level as an equation class rather than a netlist at circuit level. The VCO class is derived from the base component class and it contains its own methods to add the VCO contribution to the system Jacobian. Partial code of the VCO class is shown below.

```
vco::vco(char nameC[5],SystemVariable *node_a, sc_signal<bool>
*Vout): component(nameC,node_a, 0, value){
   Vco=Vout;
   theta = new sc_a_free_variable("theta");
}

void vco::build(void){
...
   phase = X(theta);
   phase = fmod(phase,1.0);
   Pnoise = SampleNoise();
   PhaseNoisy = phase + Pnoise;
   if (PhaseNoisy > 0.5)
      Vco->write(true);
   if (PhaseNoisy < 0.5)
      Vco->write(false);

   fmin = 0.5e9, fmax = 5e9, Vmax = 3.3, df, fc = 2e9;
   df= (fmax-fc) / Vmax;
   S=Sn();
   Qdotn = Xdot(theta);
   freq = fc + df * (a->readn());

   if (freq < fmin || freq> fmax){
      if (freq < fmin )
         freq = fmin;
      else
         freq = fmax;

      BuildM(theta,theta,S);
      BuildM(theta,a,0);
      BuildRhs(theta,-Qdotn + fc + (a->readn()) * df);
   }
   else{
      BuildM(theta,theta,S);
      BuildM(theta,a,-df);
      BuildRhs(theta,-Qdotn + fc + (a->readn()) * df);
   }
}
```

3) VCO 2: In the second noise method the VCO is a SystemC digital module, rather than an analogue component with a phase integrator, as shown in the following code:

```
void VCO2::Vf(){
   Tnow = sc_time_stamp().to_seconds();
   if (Tnow >= Tnext*0.99999){
      fmin=0.5e9, fmax=5e9, Vmax=3.3, df, fc=2e9;
      df= (fmax-fc) / Vmax;
      freq = fc + df * (Vfilter.read());
      if (freq < fmin )
         freq = fmin;
      else
         freq = fmax;

      period=1/freq;
      amp=25e-12;
      jitter = SampleNoise()*amp;
      Tafter = (period*0.5);
      Tnext=Tnow+Tafter+jitter;
      Vosc.write(!Vosc.read());
   }
      VCOphase.notify(Tnext-Tnow,SC_SEC);
      Vout.write(Vosc.read());
}
```

4) Simulation: With the first noise model, the system was simulated using extremely small analogue steps, much smaller that those calculated by the LTE control strategy. This was required to reflect accurately the effects of noise and jitter. The second noise method uses a simpler VCO model which does not require small step sizes as explained above.

The class *interfaceAD* developed as part of SystemC-A, could have been used in this example to convert signals between the analogue and digital worlds. Instead, an alternative approach based on direct connection between the modules was used. Here the two modules share the same digital signal or analogue node and conversion between the analogue and digital worlds is done implicitly within the modules. Interfaces between modules can be implemented in many different ways, for example directly through signal ports or analogue system variable ports, which is recommended especially at system level, or by nodes at analogue circuit level. In this example the connection between the LPF and VCO1 illustrates an analogue interface using node terminals.

The system response during the first eight micro seconds of the simulation, slow transients of the low pass filter voltage for both noise methods and histograms illustrating the VCO jitter are shown in Figures 10, 11 and 12 correspondingly. The histograms present the VCO jitter percentage occurrence for 5ps buckets and were calculated from the simulation results when the loop was in lock for both noise methods. Both sets of results illustrate similar behaviour.

5) Comparison with VHDL-AMS using both PLL models: Although a comparison of analogue simulators is not necessarily a fair process because simulators vary in their algorithms, methods, accuracy criteria and many details are kept hidden, we have used the PLL models to compare the speed of SystemC-A with that of the SystemVision VHDL-AMS simulator from Mentor Graphics [46]. Simulations were carried out on a Windows 2000 computer with an AMD Athlon 1400 MHz processor and 512 MB RAM. A fixed time step was used in both simulators to suppress effects of the analogue time stepping factor. In the first noise method, where the 200 μsec time interval was analyzed with the time step of 10ps, SystemC-A took 16 minutes and 55 seconds

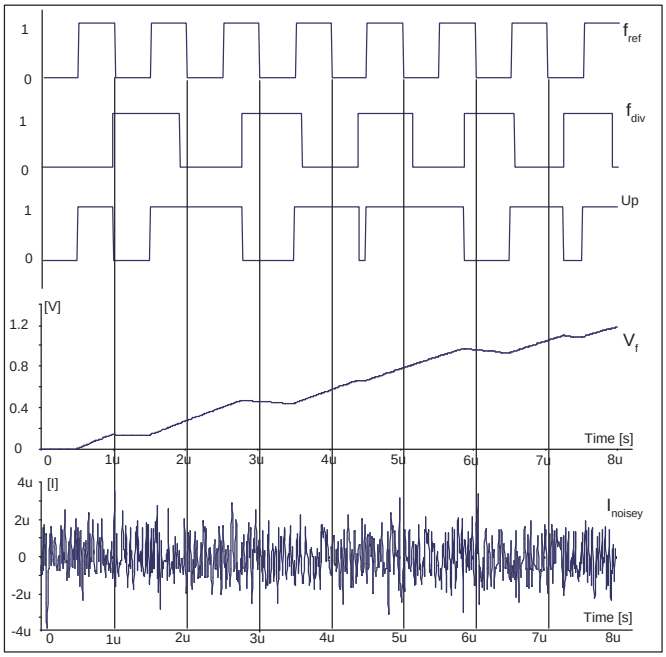

Fig. 10. 2GHz PLL frequency synthesizer simulation results.

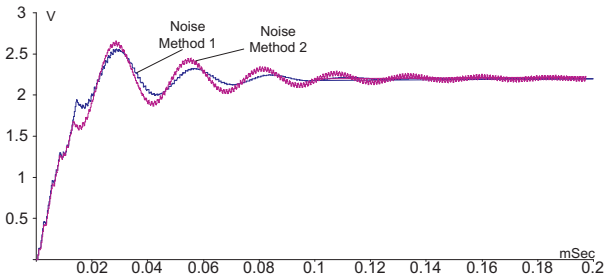

Fig. 11. Low pass filter simulated voltage for the two noise methods.

while SystemVision took 1 hour and 4 minutes on the same machine. This represents a factor of almost three times in favor of SystemC-A. Table II shows relevant statistics. For the simulation with the second noise method, the system was simulated again for an interval of 200 μsec and analyzed with time step of 0.2ns. Simulations took only 67 and 138 seconds in SystemC-A and SystemVision correspondingly.

TABLE II
PLL SIMULATION FIGURES

	Noise Method (1)	Noise Method (2)
Number of steps	20 Millions	10 Millions
Time step	10ps	0.2ns
(SystemC-A) CPU time	16m 55s	1m 7s
(SystemVision) CPU time	1h 4m 14s	2m 18s
Simulation time	200μs	200μs

IX. CONCLUSION

AMS extensions to SystemC, and a new superset of the language named SystemC-A, have been proposed. A future adoption of SystemC-A may provide significant advantages

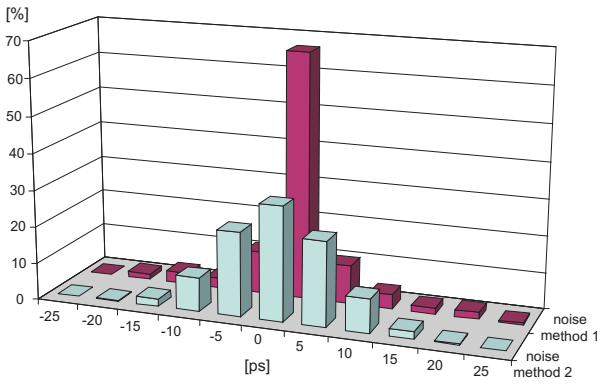

Fig. 12. VCO jitter histogram for the two noise methods.

in the modelling of modern heterogenous SoC. SystemC-A is an environment to model and simulate systems consisting of hardware/software, as well as digital and analogue parts at a variety of abstraction levels (from circuit to concept level). The proposed SystemC-A constructs can be extended easily to model many aspects of mixed energy-domain systems which may include electrical and non-electrical parts. Other advantages are the following. (1) High abstraction: SystemC-A provides the required high level of design abstraction required for the multi-million transistor era. In addition it also supports low abstraction levels required when modelling critical parts of any system. (2) Speed: SystemC-A can simulate complex systems at much higher simulation speeds than those offered by existing HDLs. (3) Extra modelling features: SystemC-A is based on the C/C++ language which is familiar to most hardware/software designers. Also it has all the properties of general programming languages. This gives a freedom in modelling and allows for description of very complex AMS systems in a user friendly manner. In many respects, SystemC-A resembles the semantics of standard HDLs. However, it has features which do not have their counterparts in existing HDLs, for example transient noise analysis. (4) Model reuse: In SystemC-A, a model can inherit properties of another, base model which gives an efficient way of model code reuse and helps significantly in the modelling process. Finally, it can be concluded from the examples that SystemC-A is a powerful and an easy-to-learn alternative to existing HDLs.

REFERENCES

[1] M. Hamour, R. Saleh, S. Mirabbasi, and A. Ivanov, "Analog IP Design Flow For SoC Applications," in *International Symposium on Circuits and Systems ISCAS*, vol. 4, (Bangkok, Thailand), pp. IV–676 – IV–679, 25-28 May 2003.

[2] J. Henkel, "Closing the SoC Design Gap," *IEEE Computer*, vol. 36, pp. 119–121, September 2003.

[3] F. Pichon, S. Blanc, and B. Candaele, "Mixed-Signal Modelling in VHDL For System-on-chip Applications," in *European Design and Test Conference*, (Paris, France), pp. 218–222, 6-9 March 1995.

[4] K. Oda, L. Prado, and A. Gadient, "A New Methodology for Analog/Mixed-Signal (AMS) SoC Design that Enables AMS Design Reuse and Achieves Full-Custom Performance, booktitle = ninth IEEE/DATC Electronic Design Processes Workshop (EDP)," (Monterey, USA), 21-23 April 2003.

[5] A. Habibi and S. Tahar, "A Survey on System-On-a-Chip Design Languages," in *IEEE 3rd International Workshop on System-on-Chip IWSOC*, (Alberta, Canada), June-July 2003.

[6] P. Ashenden, *The Designer's Guide to VHDL*. Morgan Kaufmann, 2000.

[7] IEEE Standard for Verilog Hardware Description Language, *IEEE Std. 1364-2001*, 13 ed., 2001.

[8] Accellera, *SystemVerilog Language Reference Manual 3.1*, 2004.

[9] W. Banzhaf, *Computer-Aided Circuit Analysis using SPICE*. Prentice Hall, 1989.

[10] D. Hanselman and B. Littlefield, *Mastering MATLAB 6*. Prentice Hall, 2003.

[11] IEEE Inc, *IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes),IEEE std 1076.1*, 1997.

[12] Open Verilog International, *Verilog-AMS Language Reference Manual 2.0*, January 2000.

[13] P. Bellows and B. Hutchings, "JHDL - An HDL for Reconfigurable Systems," in *IEEE Symposium on FPGAs for Custom Computing Machines*, (Los Alamitos, USA), March 1998.

[14] R. Roth and D. Ramanathan, "A High-level Hardware Design Methodology Using C/C++," in *High Level Design Validation and Test Workshop*, (San Diego, USA), pp. 73–80, 1999.

[15] B. Douglass, *Real-Time UML*. Addison-Wesley, 2000.

[16] G. Arnout, "C for System Level Design," in *Design, Automation and Test in Europe Conference and Exhibition*, (Messe Munich, Germany), 9-12 March 1999.

[17] S. liao, "Towards a New Standard for System-Level Design," in *International Symposium on Hardware/Software Codesign CODES*, (San Diego California USA), 3-5 May 2000.

[18] Open SystemC Initiative OSCI, www.systemc.org, *SystemC Language Reference Manual*, 2003.

[19] Computing Labortary, Oxford University, *Handel-C Manual*.

[20] D. Gajski and J. Zhu, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[21] J. Kunkel and K. Kranen, "SystemC Demonstrates Rapid Progress." EE Times, 26 September 2000.

[22] Open SystemC Initiative OSCI Documents, *SystemC 2.0.1 User's Guide*, 1996-2002.

[23] A. Vachoux, C. Grimm, and K. Einwich, "SystemC-AMS Requirements, Design Objectives and Rationale," in *Design, Automation and Test in Europe Conference and Exhibition*, (Messe Munich, Germany), 3-7 March 2003.

[24] M. O'Nils, J. Lundgren, and B. Oelmann, "A SystemC Extension for Behavioural Level Quantification of Noise Coupling in Mixed-Signal Systems," in *IEEE International Symposium on Circuits and Systems*, (Bangkok, Thailand), 25-28 May 2003.

[25] T. Bonnerud, B. Hernes, and T. Ytterdal, "A Mixed-Signal Functional Level Simulation Framework Based on SystemC," in *IEEE Custom Integrated Circuits Conference*, (San Diego California USA), 6-9 May 2001.

[26] M. Conti, M. Caldari, S. Orcioni, and G. Biagetti, "Analog Circuit Modelling in SystemC," in *Forum on Specification and Design Languages*, (Frankfurt, Germany), 23-26 September 2003.

[27] K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, and F. Sendig, "Design of Mixed-Signal Systems-on-a-Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1561 – 1571, December 2000.

[28] C. Grimm, C. Meise, W. Heupke, and K. Waldschmidt, "Refinement of Mixed-Signal Systems with SystemC," in *Design, Automation and Test in Europe Conference and Exhibition*, (Messe Munich, Germany), 3-7 March 2003.

[29] C. Chuang and C. Harrison, "Analogue behavioural modelling and simulation using VHDL and Saber-MAST," in *IEE Colloquium on Mixed Mode Modelling and Simulation*, pp. 1/1 – 1/5, November 1994.

[30] J. Romberg and C. Grimm, "Refinement of Hybrid Systems From Formal Models to Design Languages," in *Design, Automation and Test in Europe Conference and Exhibition*, (Messe Munich, Germany), 3-7 March 2003.

[31] K. Einwich, C. Clauss, G. Noessing, P. Schwarz, and H. Zojer, "SystemC Extensions For Mixed-Signal System Design," in *Forum on Specification and Design Languages*, (Lyon France), 3-7 September 2001.

[32] C. Grimm, P. Oehler, C. Meise, K. Waldschmidt, and W. Fey, "AnalogSL: A Library for Modelling Analog Power Drivers with C++," in *Forum on Specification and Design Languages*, (Lyon France), 3-7 September 2001.

[33] T. Zhang, K. Chakrabarty, and R. Fair, "Behavioral Modeling and Performance Evaluation of Microelectrofluidics-Based PCR Systems Using SystemC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 843 – 858, June 2004.

[34] M. Takahashi, K. Ogawa, and K. Kundert, "VCO Jitter Simulation and Its Comparison with Measurement," in *Asia and South Pacific Design Automation Conference ASP-DAC '99*, vol. 1, (Wanchai, Hong Kong), pp. 85 – 88, 18-21 January 1999.
[35] T. Kujanpaa, "Mixed Analog/Digital Circuit Simulation in APLAC," Tech. Rep. CT-37, Helsinki University of Technology, Department of Electrical and Communication Engineering, Circuit Theory laboratory, 1998.
[36] D. Lungeanu and C. R. Shi, "Distributed Event-Driven Simulation of VHDL-SPICE Mixed-Signal Circuits," in *International Conference on Computer Design ICCD*, (Texas, USA), pp. 302–307, 23-26 September 2001.
[37] D. Overhauser and R. Saleh, "Evaluating Mixed-Signal Simulators," in *IEEE Custom Integrated Circuits Conference*, (Santa Clara CA, USA), May 1995.
[38] M. Zwolinski, C. Garagate, Z. Mrcarica, T. Kazmierski, and A. Brown, "Anatomy of a Simulation Backplane," *IEE Proceedings Computers and Digital Techniques*, vol. 142, pp. 377 – 385, November 1995.
[39] Analogy Inc, *Guide to Mixed-Signal Simulation, Book one: VHDL-AMS*, 1996-1999.
[40] A. Brown and M. Zwolinski, "The Continuous-Discrete Interface - What does this really mean? Modelling and Simulation Issues ," in *Proceedings of the 2003 International Symposium on Circuits and Systems ISCAS '03*, vol. 3, pp. III–894 – III–897, 25-28 May 2003.
[41] J. Gleick, *Chaos: Making a New Science*. Minerva Publisher, November 1996.
[42] S. Ben-Yaakov, "SPICE Simulation of PWM DC-DC Convertor Systems: Voltage Feedback, Continuous Inductor Conduction Mode," *IEE Electronics Letters*, vol. 25, pp. 1061 – 1063, August 1989.
[43] B. Antao, F. El-Turky, and R. Leonowich, "Mixed-Mode Simulation of Phase-Locked Loops," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 8.4.1 – 8.4.4, 9-12 May 1993.
[44] N. Godambe and C. R. Shi, "Behavioral Level Noise Modeling and Jitter Simulation of Phase-Locked Loops with Faults Using VHDL-AMS," in *15th IEEE VLSI Test Symposium*, pp. 177 – 182, 27 April-1 May 1997.
[45] X. Mao, H. Yang, and H. Wang, "Behavioral Modeling and Simulation of Jitter and Phase Noise in Fractional-N PLL Frequency Synthesizer," in *IEEE International Behavioral Modeling and Simulation Workshop BMAS*, (San Jose, California, USA), 21-22 October 2004.
[46] Mentor Graphics Corporation, www.mentor.com, *System Vision Data sheet*.