SEAMS - A SYSTEMC ENVIRONMENT WITH ANALOG AND MIXED-SIGNAL EXTENSIONS

H Aljunaid and T J Kazmierski

School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK

ABSTRACT

We describe an efficient implementation of analog and mixed-signal extensions integrated with SystemC 2.0. SEAMS (SystemC Environment with Analog and Mixed-Signal extensions) uses a general-purpose analog solver to handle analog extensions and to provide modelling capabilities for general, mixed-mode systems with digital and non-linear analog behavior. We have extended the SystemC 2.0 kernel to invoke and synchronize our analog solver in each simulation cycle while maintaining compliance with the SystemC simulation cycle semantics. The operation of SEAMS is illustrated with the practical examples of a boost power converter and a 2GHz phase-lock loop frequency multiplier with noise and jitter models. Mixed-signal systems of this kind are known to be difficult to simulate as they exhibit disparate time scales which put most simulators in numerical difficulties. We hope that the practical experience of SEAMS might aid the recent efforts to standardize analog and mixed-signal extensions for SystemC.

1. INTRODUCTION

Several high level hardware description languages, such as VHDL and Verilog, have recently been extended to provide mixed-signal modelling capability and new standards for VHDL-AMS [1] and Verilog-AMS [2] are now widely available. In the light of the growing popularity of mixed, analog and digital ASICs, this is not surprising. Naturally, with the advent of SystemC, several research publications highlighted the possibility to extend the standard, digital modelling capabilities of SystemC [3, 4, 5, 6] to the analog domain. Bjornsen et al. [7] described an efficient software framework for rapid behavioral modelling and simulation of analog-to-digital converters based on SystemC. Their framework consists of three parts, the first is an analog extension representing analog signal classes, the second is a signal module library, which includes basic blocks required for A-D modelling, such as switched-capacitor integrators, operational amplifiers and a track-and-hold amplifier. The third part is a mixed-signal system test bench. Einwich *et al.* [6] presented a SystemC extension allowing an overall specification and verification of mixed-signal systems with linear analog models. They consider an ADSL line driver with analog linear filters as an example. This contribution outlines the concept of a general, mixed-signal SystemC simulator comprising an analog kernel with an underlying transient solver for non-linear, algebro-differential equations. We devote particular attention to the problem of synchronizing the analog kernel with the digital one. Our synchronization technique is compliant with the definition of SystemC simulation cycle semantics [4]. For this purpose, we link the analog kernel to the SystemC environment as a user module and synchronize it with the SystemC kernel via a lockstep synchronization algorithm. Operation of the extended, mixed-signal SystemC simulation platform is demonstrated using practical example of a boost power converter, in which analog behavior tightly interacts with a digital control loop. The results presented here might aid the current efforts to standardize analog extensions for SystemC [8].

2. ANALOG AND DIGITAL SOLVER SYNCHRONIZATION

Like in the case of most high-level hardware description languages, a SystemC model consists of a hierarchical network of parallel processes, which exchange messages under the control of the simulation kernel process and concurrently update the values of signals and variables. Signal assignment statements do not affect the target signals immediately, but the new values become effective in the next simulation cycle. The kernel process resumes when all the user defined processes become suspended either by executing a *wait* statement or upon reaching the last process statement. On resumption, the kernel updates the signals and variable and suspends again while the user processes resume. If the time of the next earliest event t_n is equal to the current simulation time t_c, the user processes execute a delta cycle.

To comply with the SystemC execution semantics, our

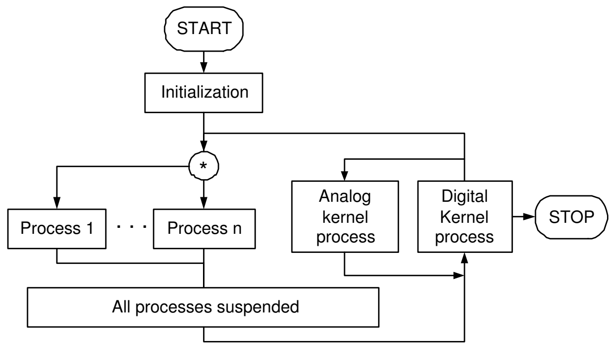

Fig. 1. Simulation cycle of a SystemC system with analog extensions.

mixed-signal SystemC simulator comprises an analog kernel, which runs as a SystemC process and drives the user defined analog modules. This scenario requires the analog solver to be able to handle delta cycles in a manner similar to that of digital processes, namely, the state of the analog solver may not be updated until after the SystemC kernel advances the simulation time ahead of the current simulation time t_c. In other words, the underlying analog solver not only must be able to execute delta cycles with the step size $h = 0$, but also must have a capability to backtrack to the state just before t_c (t_c^-), should new events at t_c change the analog stimili. The former requirement would be difficult to satisfy with most standard, SPICE-like simulators since they require a minimum step size greater than zero. Moreover, small step sizes can cause large round off errors and lead to inaccurate results. The analog model proposed in section 3 can be solved with arbitrary step size, including $h = 0$. Backtracking is achieved by retaining to the analog state (t_c^-) when required. The analog kernel repeatedly executes the simulation cycle shown in figure 1, which might involve delta cycles and backtracking. Analog simulators do not use events but instead employ an entirely different approach to time step control, namely, continuous step-size adjustment to minimize the errors caused by the numerical integration method used to solve differential equations in the circuit model. In order to minimize the error in the numerical integration, it may be necessary to repeatedly reject the circuit solution at a time point and to cut the time step. It is therefore necessary for the analog kernel in a SystemC environment not to advance past the current simulation time t_c unless a delta cycle occurs and re-evaluation of the current step is necessary. There are essentially two algorithms for the synchronization of two or more solvers running concurrently [9]. One, optimistic approach allows each simulator to progress in time until it runs out of internal events (internal activity has ceased) and then suspend. The alternative approach, adopted here, is the lock-step algorithm. The analog kernel advances until the current simulation time and, before suspending, schedules and event at the time equal to the current simulation time plus the next selected step size. This approach ensures that the SystemC kernel will make a step in time no larger that the analog kernel's step size. Since the analog kernel runs as a user process and is controlled by the SystemC kernel, no synchronization deadlock may happen. The only causes for deadlock can arise due to a failure to converge in the analog solver or due to unresolvable delta cycles. The signals that pass between the digital and analog domains are carried by global nets. The solution method adopted here allows each global net to be able to connect two partitions (the digital and analog simulator), one of which owns the net. This means that a general global net is divided into two smaller terminal nets, implemented as SystemC ports. The synchronization approach adopted in this paper is based on the lock-step principle to ensure that no results are thrown away and there is no need for backtracking. Most existing digital solvers cannot backtrack and therefore no fundamental changes are required if a mixed-signal system is integrated to the SystemC kernel. Our lock-step synchronization algorithm has been implemented as a modification to the digital kernel and can be described in the form of pseudo-code as follows:

```
time <- 0;
initialize both the analog and digital kernel;
while (time <= end_time) do

   while (immediate notifications are pending) do
      execute the analog kernel
      distribute notifications generated
        by the analog kernel on global nets

      while (there are active processes) do
         run a selected process
      end while (there are active processes)

      update signals

   // check if a delta cycle is necessary
   end while (immediate notifications are pending)

   advance time to the next timed notification
end while (time<= end_time)
```

In most cases, delta cycles caused by zero-delay paths are eventually resolved but, in general, sharing zero-delay paths between the two kernels should be avoided. If the maximum allowed number of delta cycles is exceeded, the algorithm treats this situations as deadlock and stops.

3. ANALOG MODEL

The description of the analog model relies on C++ classes that provide behavior for circuit nodes and primitive components. The component classes contain virtual methods that are invoked at matrix build time each time the analog solver requires to update the Jacobian according to modified nodal analysis stamps.

```
void analog::BuildCircuit(void){
 ...
  node v1("v1"),v2("v2"),vc("vc"),vout("vout"),vf("vf");
  node vramp("vramp");
  voltageS   E("E",v1,0,1.5);
  inductor   l1("l1",v1,v2,10e03,5.0);//10mH,5ohm
  resistor   ro("ro",vout,0,500.0);
  resistor   r1("r1",vout,vf,23e3);
  resistor   r2("r2",vf,0,10e3);
  MOS0       M1("M1",v2,vc,0);
  diode0     D1("D1",v2,vout);
  capacitor  Co("Co",Vout,0,1e-3);
  voltage_ramp vr("vr",vramp,0,1e-6);
 ...
} // analog::BuildCircuit() --------------------
```

The analog kernel invokes BuildCircuit()once prior to the simulation. Additional user-defined methods represent interfacing components that provide functionality for synchronization between SystemC ports and their corresponding values in the analog solver. The interfaces are bound at the build stage.

```
sc_signal<bool> ASig,S1;
sc_signal<double> S2;
sc_clock Clk("Clock", 0.1, SC_US,0.5);
  ...
    analog analog1("analog1");
    analog1.Vcontrol(S1);
    analog1.Vout(S2);
    analog1.clock1(ASig);

    digital digital1("digital1");
    digital1.Vout(S2);
    digital1.Vcontrol(S1);

    stim stim1("stim1");
    stim1.A(ASig);
    stim1.Clk(Clk);
  ...
```

Small step sizes including the case of a delta cycle where $h = 0$ are handled by smoothing the digital signal as illustrated in figure 2.

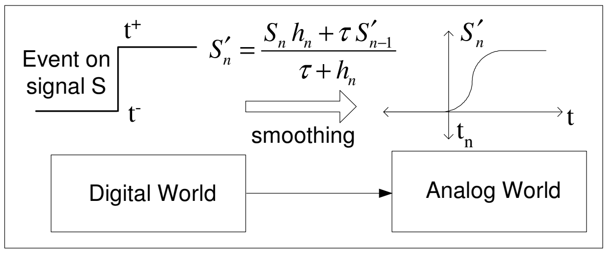

Fig. 2. Handling small step sizes.

4. PRACTICAL EXAMPLES

The analog extensions added to the language contain classes to handle electrical nodes and primitive analog components with user defined behavior necessary to build and solve the analog equation set. A boost 1.5V to 3.3V power converter and a 2GHz PLL-based frequency multiplier with noise and jitter have been used as examples of non-trivial analog and digital interaction. 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 a switched-mode power supply, the analog transient in the output circuit is four to five orders of magnitude slower than that of the fast switching waveforms in the digital controller. As a typical simulation in a system of this kind might require a hundred million time points, excessive CPU times often occur when the entire system is modelled on the circuit level The capacity of SystemC to enable system-level, mixed-signal modelling can vastly reduce simulation times where concepts need to be verified quickly and detailed, circuit-level modelling is not required. The power converter block structure is presented in figure 3. The controller's digital behavior in the example is modeled as a standard SystemC *SC_MODULE*. The testbench instantiates both the analog and digital module and provides global nets. Sample simulation results at steady-state are presented in figure 4.

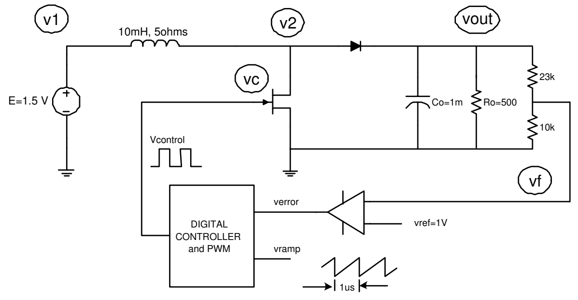

Fig. 3. Boost 1.5V to 3.3V switch-mode power supply with digital control.

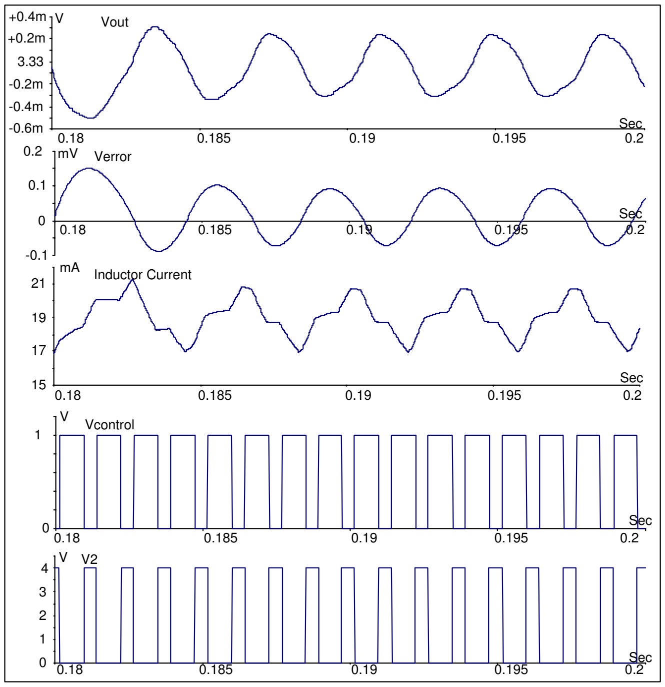

Fig. 4. SMPS simulation results for a 200ms time window in steady-state of the boost switch-mode power supply working in continuous mode.

The PLL multiplier simulation also requires millions of time steps to accurately reflect the effects of noise and jitter. The behavioral VCO model includes integration of the phase which converts the input voltage noise to a jitter in the output frequency. The system's block diagram is shown in figure 5 as well as the VCO jitter histogram calculated from the simulation results when the loop was in lock. Figure 6 shows different system values in the first micro seconds of the simulation.

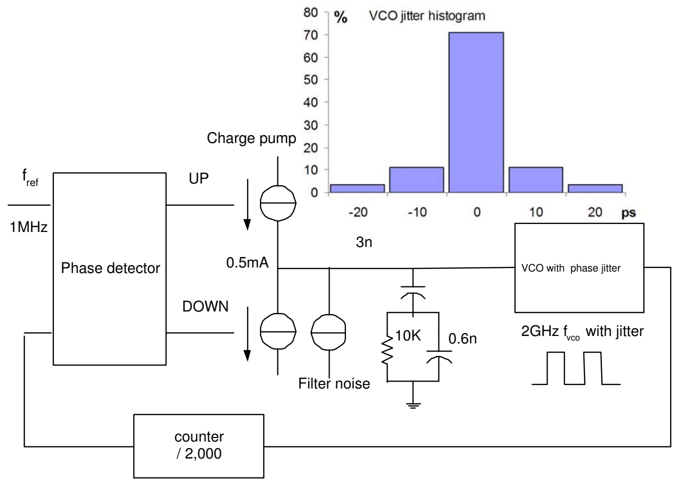

Fig. 5. 2GHz PLL multiplier with noise and jitter.

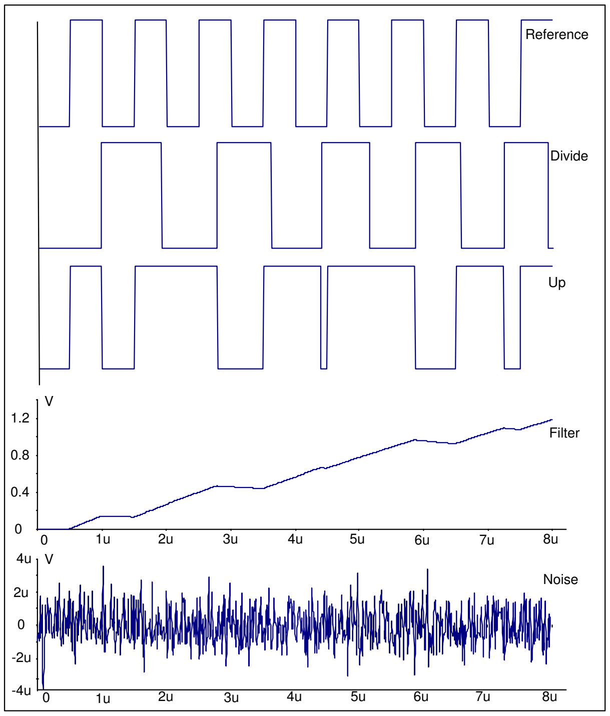

Fig. 6. 2GHz multiplier simulation results.

5. CONCLUSION

An extension to SystemC has been developed to simulate general, mixed-mode systems with digital and non-linear analog behavior. A novel analog solver has been developed coupled to the SystemC kernel via a lock-step synchronization algorithm. Its principle has been demonstrated using the practical examples of a mixed-signal boost power converter, in which analog behavior interacts with a digital control loop and a PLL frequency multiplier with noise and jitter.

6. REFERENCES

[1] E. Christen and K. Bakalar, "Vhdl-ams - a hardware description language for analog and mixed-signal application," *IEEE Trans CAS II*, vol. 46, no. 10, pp. 1263–1272, October 1999.

[2] *Verilog AMS Language Reference Manual, Accelera*, August 1998.

[3] J. Gerlach, "System level design using the systemc modeling platform," *Proc. SDL 2000*, 2000.

[4] W. Mueller, "The simulation semantics of systemC," *Proc. DATE 2001*, 2001.

[5] P. Panda, "Systemc - a modeling platform supporting multiple design abstractions," *Proc. ISSS 2001, Montreal, Canada*, 2001.

[6] K. Einwich, C. Clauss, G. Noessing, P. Schwarz, and H. Zojer, "SystemC extensions for mixed-signal system design," *Proc. FDL'2001, Lyon, France*, September 2001.

[7] T.E. Bonnerund, B. Hernes, and T. Ytterdal, "A mixed-signal, functional level simulation framework based on systemc for soc applications," *Proc. Custom IC Conf., CICC'2001, San Diego, California*, pp. 541–545, May 2001.

[8] A. Vachoux, C. Grimm, and K. Einwich, "Systemc-ams requirements, design objective and rationale," *Proc. DATE'2003, Munich, Germany*, pp. 388–393, 3-7 March 2003.

[9] M. Zwolinski, C. Garagate, Z. Mrcarica, T.J. Kazmierski, and A.D. Brown, "The anatomy of the simulation backplane," *IEE Proceedings on Computers and Digital Techniques*, vol. 142, no. 6, pp. 377–385, November 1995.