Analogue Mixed Signal Simulation Using Spice and SystemC

Tobias Kirchner, Nico Bannow

Robert Bosch GmbH
Corporate Sector Research and
Advance Engineering
P.O. Box 30 02 40, 70442 Stuttgart, Germany
Tobias.Kirchner@de.bosch.com

Christoph Grimm

Vienna University of Technology
Institute of Computer Technology

Grimm@ICT.TUWien.ac.at

***Abstract*—SystemC is a discrete event simulator that enables the programmer to model complex designs with varying levels of abstraction. In order to improve precision, it can be coupled to more specialized simulators.**

This article introduces the concept of loose simulator coupling between an analogue simulator and SystemC.

It explains the properties and advantages which include a higher simulation performance as well as a higher degree of flexibility.

A design example in which SystemC will be connected to SwitcherCad will demonstrate the benefits of loose coupling.

I. Introduction

Some progress has been made in simulation technology over the past couple of years. With more and more manufacturers supporting it, SystemC [1], [2] has grown in importance constantly.

Using SystemC it is possible to reach very high levels of abstraction. However, its ability to extend simulation coverage down to analogue level is limited. So far there is no way for SystemC to simulate designs with better precision than register transfer level (RTL). Still, there are circuits which just cannot be simulated inside the digital domain since they require feedback from the analogue world. This is where more specialized, mostly continuous time simulators need to be used. Because SystemC is essentially C++, it can do both, be a simulator and act as connecting and coordinating element between different other simulators. With a specialized simulator like MatLab Simulink, Spice or Saber connected to it, SystemC capabilities are extended towards analogue continuous time simulation.

Another attempt to extend SystemC simulation capabilities is *SystemC AMS Extensions*. It is an extension to the SystemC language that aims at analogue designs [3], [4].

II. Related Work

When coupling an analogue simulator to SystemC, the usual way of interfacing is by using the simulators built-in programming language interface and extending the SystemC kernel with co-simulation capability. In a tightly synchronized co-simulation all simulators start at simulation time $t_0 = 0s$. This paper refers to the simulator that runs first as the master. In a tight simulator coupling the simulator is usually not pre loaded with a bias point or an internal state. Master and slave simulator are synchronised at fixed intervals. In an improved approach the master determines when its next event occurs and commands the slave to advance to that point. The slave reports back to the master and indicates whether it has encountered an event of its own before the end of this time step. If it has, the master advances to that time instead [5]. As seen in Fig. 1 all simulators are instantiated at the beginning of the simulation and remain in memory until they are finished. Between simulations they are just halted, retaining their internal state. The advantage of this kind of coupling is a very tight and synchronous interaction of the simulators chosen. The outputs can be examined at any time during the simulation and they will be synchronized in regular intervals.

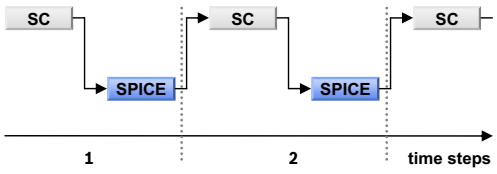

Fig. 1. Execution of tightly coupled simulators. Both of them start at t_0 and are executed at least once every simulation timestep.

Tight coupling also brings some problems with it. Most of all, only simulators which support a co-simulation interface can be used. There is also a lot of communication necessary in order to keep the simulators synchronized. Depending on the implementation of the coupling interface, this communication overhead may even use more computational power than the simulation itself. Especially simulations that are distributed across a network suffer from additional network latency delays.

III. Loose Simulator Coupling

In contrast to the tight coupling, with loose coupling there is a clear difference between the master and a slave simulator.

978-3-9810801-5-5/DATE09 © 2009 EDAA

The master gets executed upon start of simulation and remains active throughout the remainder of the simulation. A slave on the other hand gets called by the master, executes and shuts down when it has finished. When started for the first time, a slave calculates a bias point. When it exits, it will save the current state of the simulation to file so that it can resume the simulation when it is called again.

Not all slave simulators do have to start at t_0. The master may invoke them later during the simulation.

In order to be used as a loosely coupled simulator a slave needs to accept and execute the simulation model source files as a parameter and it needs to be able to load and save its internal state on request.

An example for such a simulator is the spice based SwitcherCad3 from Linear Technologies [6].

In order to connect SCad3 to SystemC the wrapper in Fig. 2 has been developed [7]. It provides access to the netlist, components and all other parameters of the analogue simulation as if they were an integral part of SystemC. In order to achieve this, it translates SystemC commands and integrates the resulting information into the netlist file prior to executing the slave.

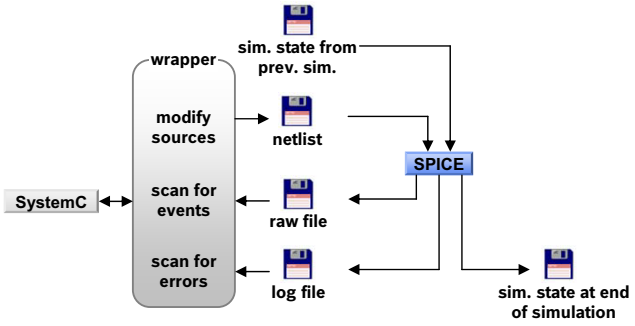

Fig. 2. All information between SystemC and SCad3 is exchanged via files.

It also scans the results for errors and events that need to be reported back to SystemC.

IV. Advantages of Loose Coupling

A. Selective Precision

The fact that the slave simulator does not have to start at t_0 is one of the most important advantages of loose coupling. This allows SystemC to enable SCad3 only when it is needed.

In a simulation where a SystemC microcontroller model is connected to an analogue circuit via I/O pins, the analogue simulator would not be invoked until the microcontroller has finished the reset and initialisation phase. During this phase there is no access to the I/O pins and thus no change in the analogue part of the circuit. Because the analogue simulator is not enabled at this point the simulation is accelerated considerably. This is shown in Fig. 3. SC_M2 is the first module that accesses the pins that are connected to the analogue parts of the design.

When the simulation of analogue circuits is not needed, SCad3 is not started. In contrast to tight simulator coupling, in this example analogue simulation can be resumed later during the simulation by making the appropriate simulator state file available to SCad3.

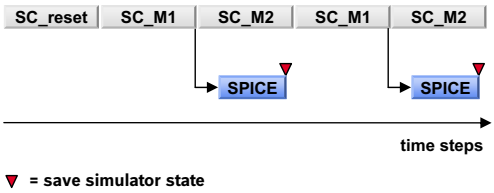

Fig. 3. Loose Coupling allows SystemC to simulate a microcontroller reset and invoke the analogue simulator afterwards.

This selective precision results in a considerable increase in simulation performance.

B. Re-use of Multiple Simulation Runs

Whenever the analogue slave simulator is invoked by the master, it is supplied with internal simulation state details. It then runs for a certain time and saves the updated simulation state information and the simulation results back to file. It so leaves a complete trace of previous activities which allow for a re-simulation at any time.

In order to re-simulate a section of time none of the preceding simulations has to be repeated since the results would not change but are already available. Only simulations after the point where the change did happen have to be updated with new results. See Fig. 4. After that the results are collected and arranged by the master.

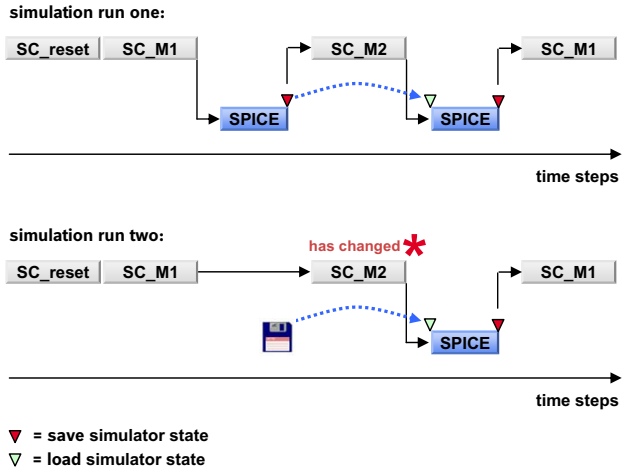

Fig. 4. If SC_M2 in simulation 1 changes, only Spice simulations after that point have to be re run.

C. Multi-Processor Computation Capability

Using the standard kernel SystemC simulations can not be designed in a way that they use multiple processors. Whether the analogue simulator makes use of more than one processor depends on the simulator and can not be influenced by the user.

In a normal, tight simulator coupling only one simulator runs at a time. They alternate between synchronization points.

In a loose coupling, however, simulators are synchronized when there is a change of inputs to or an event from the analogue simulator. Between these events the simulators can run asynchronously and in parallel. If a design contains analogue parts that are independent from each other, e.g. two output stages driven by different I/O pins, each part can be simulated with a dedicated simulator.

If an analogue event occurs only the simulators which are affected by it are stopped or started. This does not only decrease communications load but also distributes it, preventing bursts.

D. Universal Access to Analogue Components

In a tight simulator coupling access to components of an analogue simulation is limited by the interface. Depending on the implementation this access may be limited to a group of elements, e.g. voltage sources or just the interfaced component itself.

In a loose coupling, every component may be read or written between simulation steps. This is possible because each simulation step is an independent simulation with its own netlist. Each component in the netlist can be changed between simulation steps. It is also possible to add or remove components. In combination with the ability of a loosely coupled simulator to run ahead of the digital siulator, it is possible to make the simulation sensitive to events like extrema or turning points. These events can only be detected because they occurred in the past.

In a tight simulator coupling detection of an extremum is only possible if the simulation can be rolled back to that point.

E. Optimizing a Design for Loose Simulator Coupling

It has become obvious by now that loose coupling has its greatest advantages in designs, that need only very few synchronization points. It is therefore advantageous to use a mechanism like the one in Fig. 7 which translates requests to a loosely coupled simulator whenever necessary.

If the input to the analogue circuitry changes with every delta cycle, there is no advantage. But as soon as there are cycles that do not affect the analogue design, requests to the analogue simulator are cached and passed on to the simulator before a relevant change happens. The graph which is labeled *out* in Fig. 6 represents the input to the analogue part of the design in Fig. 5. As long as it remains stable, there is no need to interrupt the according analogue simulation run.

Fig. 5. SystemC is connected to the analogue simulator via the signal *out*.

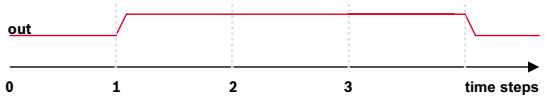

Fig. 6. The signal *out* in Fig. 5 changes only twice during five simulation time steps.

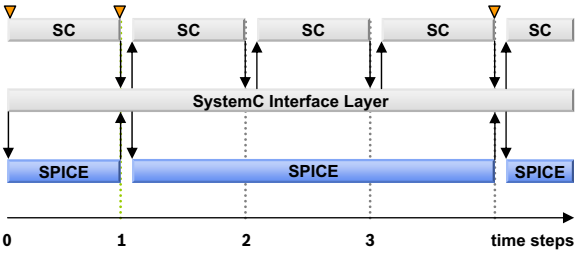

Fig. 7. An interface layer collects requests to the analogue simulator and passes them on if a signal changes.

Simulation results are only synchronized at points where the analogue simulator gets started. Introducing an interface layer like in Fig. 7 allows already existing simulations with tightly coupled simulators being quickly transformed into a loosely coupled equivalent.

V. Design Example

A. Controlling a Capacitive Load

The following example demonstrates the benefits of loose simulator coupling.

The example in Fig. 8 consists of an amplifier that is connected to a capacitive load. The amplifier has a digital push pull output stage which can be switched to either ± 20 V or tristate. It has a polarity input and a tristate input. An application for such a setup can be a piezo ceramic injector or if used with a inductive load a hydraulic valve.

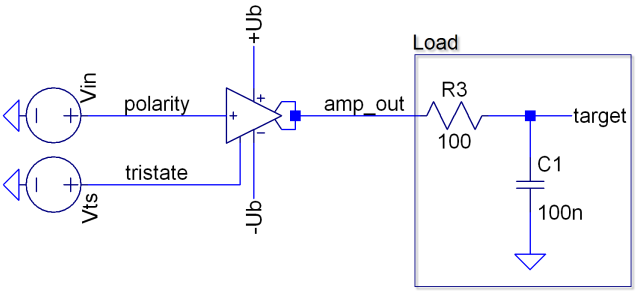

Fig. 8. Schematics of the design example. A push-pull amplifier drives a capacitive load.

B. Operation

The amplifier shall be controlled in a way that it charges the capacitor to a given voltage and keeps it at that level. Each time one of the inputs of the amplifier changes, a new analogue simulation gets started. It runs until either a new digital event that affects the analogue simulation occurs, or it terminates prematurely due to an event in the analogue domain.

A termination condition for the analogue domain needs to be set prior to simulation execution. Once the analogue simulation terminates, it hands back control to the digital simulator which then executes the next event.

Fig. 9 shows the output of the SCad3 simulation after the results from the different simulation steps have been merged.

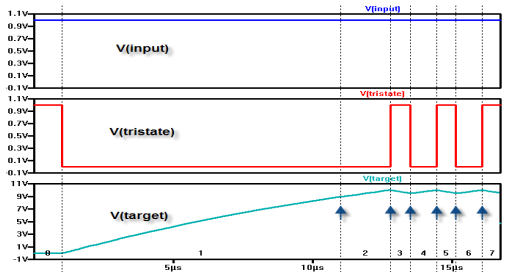

Fig. 9. The output of the simulation. The vertical lines mark the simulation boundaries.

The upper trace shows the input voltage to the amplifier. A value of 1 V causes the amplifier to output 20 V, 0 V at the input results in −20 V at the output. The trace in the middle is the tristate input of the circuit. The trace at the bottom shows the voltage across the capacitor. After a short delay it rises up to the desired level and stays there with a hysteresis of 0.5 V.

Each simulation is labeled with a number at the bottom. The vertical dashed lines mark the simulation boundaries. The arrows indicate where analogue events occurred. At the end of the first analogue simulation part, after approximately 1µs, no analogue event has occurred because simulation finished at the predefined point in time. All subsequent simulations are interrupted because the abort criterion was met, i.e. an analogue event occurred. The master then initializes a new simulation, sets the break condition and the values for the inputs of the amplifier. It is then started with a predefined time of 2 microseconds. The simulation will not run for the whole 2 microseconds but stop before that time because the abort condition will be met. The voltage sources inside the simulation are evaluated directly without any hardware interfaces as described in section IV-D. This means that signal conditioning is encapsulated within the wrapper.

If the simulation had to be extended or modified, previously calculated simulation steps that have not changed can be re-used. The operator just marks them as inactive to skip the actual calculation. The simulation in Fig. 9 has been extended so that the voltage across the capacitor drops to five volts. The result can be seen in Fig. 10. The *level* input of the amplifier is set low while the *tristate* input is switched low as well so that the amplifier will output −20 V. As a result the voltage drops very quickly. In order to mark the preceding simulations as inactive, the corresponding lines of code have to be changed manually. In future steps one could also implement some sort of automatism which tracks the inputs to the simulation and decides whether the data can be re-used or not. To proceed to the next simulation phase the code is simply extended with the new simulation steps.

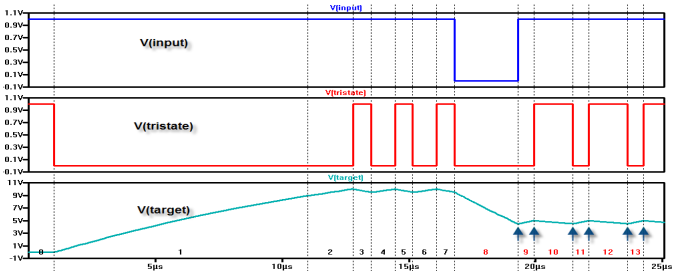

Fig. 10. The previous simulation has been extended so that the voltage is now stable at 5V.

Fig. 10 shows the simulation that has now been extended. The arrows in the graph indicate that an analogue event has occurred at this point. The arrows can only be found in the second part of the simulation because the first part has been loaded from files which were already available from the first simulation run. The total simulation time for this simulation run was only 6 seconds, compared to 7 seconds of the previous simulation. Without the re-use of simulation results, the total simulation time would have been 13 seconds.

VI. Conclusion

This paper has introduced the concept of loose simulator coupling and explained its advantages over conventional tight coupling.

Loose coupling can easily be established with many simulators because it just requires the slave simulator to be able to handle netlists as well as to load and store its internal state.

It can speed up the simulation by temporarily suspending the analogue simulation and by reducing synchronisation points. This also facilitates the distribution of multiple loosely coupled simulators across different processors and allows for a generic access to components between simulation steps.

By storing simulation results along with the simulators internal state and timestamps in a file, these results can be re-used in consecutive simulation runs.

References

[1] O. S. Initiative, *IEEE 1666-2005 Standard SystemC Language Reference Manual*, http://www.systemc.org, December 2005.

[2] T. Grötker, *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[3] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich, *An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions*, http://www.systemc-ams.org/ ed., June 2008.

[4] K. Einwich, A. Vachoux, C. Grimm, and M. Barnasconi, *SystemC AMS extensions Draft 1*, SystemC-AMS Working Group, December 2008. [Online]. Available: systemc-ams.org

[5] L. Gheorghe, F. Bouchhima, G. Nicolescu, and H. Boucheneb, "A formalization of global simulation models for continuous/discrete systems," in *SCSC: Proceedings of the 2007 summer computer simulation conference*. San Diego, CA, USA: Society for Computer Simulation International, 2007, pp. 559–566.

[6] M. Engelhardt, *SwitcherCad3 Manual*, 2nd ed., Linear Technonogy Inc., December 2007.

[7] T. Kazmierski and N. Clayton, "A two-tier distributed electronic design framework," in *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, vol. -. Washington, DC, USA: IEEE Computer Society, 2002, p. 227.