Analog Circuit Modeling in SystemC

Massimo Conti, Marco Caldari, Simone Orcioni, Giorgio Biagetti
D.E.I.T., Università Politecnica delle Marche
via Brecce Bianche, I-60131, Ancona, Italy

Abstract
This paper proposes a methodology for the extension of SystemC to mixed signal systems. An oscillator made up of an inverter chain has been used to test the accuracy and stability of the algorithm proposed. Finally, a simulation of a complex mixed-signal fuzzy controller is used to show the speed up achievable with a high level description of the analog block with respect to SPICE simulation.

1 Introduction
The rapid progress of VLSI process technology enables the implementation of complex systems in a single chip with several millions of gates. To deal with such complexity, parametric and reusable IP (Intellectual Property) cores, system level verification/validation techniques and a seamless design flow that integrates existing and emerging tools will be the key factors for a successful System on Chip (SoC) design.

In order to bridge the gap between technology capabilities for the implementation of SoCs and Electronic Design Automation (EDA) limitations, an environment integrating new concepts within new languages and methodologies is necessary. SystemC [OCSI,PRP,Gro02], based on C++ language, is an emerging library and simulation environment used to provide a virtual model required to verify in short time the design specifications.

CAD vendors, designers and researchers are making a great effort in the development and standardization of SystemC, that is becoming a de facto standard.

The implementation of digital and analog parts, such as DAC and ADC or RF front-end, in the same chip requires a mixed-signal simulation. Research and standardization activities for the extension of SystemC to mixed signal systems are therefore necessary [Vac97,Ein01,SC_AMS], as it has been done for VHDL-AMS and Verilog-A.

This work presents a possible solution for the extension of mixed-signal modeling in SystemC. The objective of this work is the definition of a methodology for the description of analog blocks using instruments and libraries provided by SystemC. In this way the analog and digital modules can be easily simulated together.

The methodology allows the designer to describe the analog system either at a low or a high level using analog macromodels. The analog simulator uses a variable time step Euler algorithm, with a compromise between the accuracy of the simulation and the CPU time required.

A simple CMOS inverter has been used to show in detail the methodology. Finally, the methodology has been applied to an oscillator and to the modeling and simulation of a complex mixed-signal fuzzy controller.

2 Description of Analog Modules in SystemC
SystemC is a library of C++ classes developed to build, simulate and debug a System on Chip described at system level. SystemC provides an event driven simulation kernel and it describes the functionality of a system
using processes. Three kinds of processes are available: methods, threads and clocked threads. The signals in the sensitivity list activate a thread, its execution may be suspended by calling the library function wait(event), at the occurrence of the event the thread is resumed. The methodology proposed in this work uses thread processes to model analog blocks in SystemC, in order to simulate analog blocks using the standard libraries and the simulation kernel of SystemC.

The SystemC extension to mixed-signal systems, proposed in this work, enables the creation of a high level executable model of both digital and analog part of the system in the same simulation environment. This allows a fast evaluation of the performances of the complete system.

We will face the problem of the implementation in SystemC of continuous time lumped systems described by a system of non-linear ordinary differential equations of the type

\[
\begin{align*}
x'(t) &= f(x, u) \\
y(t) &= g(x, u)
\end{align*}
\]

(1)

where \( f \) and \( g \) are vectors of expressions, \( x, y \) and \( u \) are state, output and input vectors, respectively. Eq. (1) can describe in detail the analog circuit under consideration or it may represent a high level macromodel describing its functionality. We think that a high level macromodel should be used in a SystemC system level simulation, using a SPICE-like simulator only when a detailed analysis is required.

The following solution is obtained using the explicit Euler formula

\[
\begin{align*}
x(t + dt) &= x(t) + dt \cdot f(x(t), u(t)) \\
y(t) &= g(x(t), u(t))
\end{align*}
\]

(2)

The explicit Euler formula is not the most efficient solution, but it is the simplest, for this reason it has been used to verify the efficiency in a SystemC implementation. An extension to other kind of solutions is possible using the methodology proposed in this work. The authors are now working to compare the results of different numerical solutions of eq.(1), e.g. explicit Euler and 4-th order Runge Kutta algorithms.

Different solutions are possible in the SystemC modeling of processes implementing eqs.(2) and in the choice of the time step \( dt \):

1) A global constant time step for all the analog processes. This “analog simulation clock” activates all the analog blocks. This solution is the simplest, but very inefficient since it requires a very small time step to reach an accurate solution.

2) A global adaptive time step for all the analog processes. All the analog blocks are activated by this adaptive “analog simulation clock”. A new global thread is necessary to calculate at each time instant the smallest time step required to guarantee the required accuracy. This thread acts as an analog simulation kernel. This solution is inefficient when analog blocks with different time constants are present in the same system.

3) The solution proposed in this paper is the following: each analog process calculates its adaptive time step, and it passes this time step to the blocks to which its outputs are connected. The analog inputs activate the process, that calculates the new state and output values. Then it sends them to the connected analog blocks after the minimum between its time step and the time steps received from the input blocks.

The SystemC implementation of an analog block is reported in Fig.1. The module is composed of two kind of threads: the calculus thread and the activation threads, one for each input module.

Each activation thread starts when a change occurs in the signals coming from the corresponding input module, as shown in Fig.1. The calculus thread, that updates the state and output vectors, is activated only by signals coming from the activation thread according to the algorithm reported in Fig.2. The activation thread calculates the difference between the actual value of each analog input signal and the value at the previous step. If the maximum variation is bigger than a predefined reference threshold the calculus thread is activated: this control can activate promptly the calculus thread when a sudden input variation occurs, avoiding delays or loss of impulse signals when the simulation step is too big.
The calculus thread is also activated if the minimum value between the time steps of the input modules is lower than the internal time step divided by a factor "K".

Fig. 3 reports the algorithm implemented in the calculus thread. After the acquisition of the analog inputs, the time step \( dt \) and the new value of the state are calculated. Then the time step is adapted on the basis of the

\[
\Delta > \text{threshold} \quad \text{flag} = 1 \\
\Delta < \frac{dt}{K} \quad \text{flag} = 1 \\
\text{Wait (analog inputs)}
\]
variation of the state value: the time step is decreased by a factor $F$, if the variation is bigger than a threshold parameter $\vartheta_2$ and the time step is bigger than the parameter $DTMIN$.

Conversely, the time step is increased by the same factor, if the variation of the state vector is lower than a threshold parameter $\vartheta_1$ and the time step is lower than the parameter $DTMAX$.

In this way the internal time step increases when the state vector is reaching a steady state, and decreases when the state vector is rapidly changing. $DTMAX$ and $DTMIN$ fix the maximum and minimum values of the time step, respectively.

Then the algorithm waits for an activation event coming from the activation threads by using the SystemC `wait(timeout, e)` function. The thread execution is suspended waiting for the activation event $e$ for a maximum time equal to the internal time step $dt$, after this time the thread is activated in any case.

The state value is updated, if the actual time $t_2$ less than the $t_1+dt$. The values of the parameters $F, \vartheta_1, \vartheta_2, DTMAX, DTMIN, threshold$ and $K$ strongly influence the accuracy of the simulation and the CPU time required. An appropriate tuning has been performed and the following examples will show some results.

## 3 Application Examples

Two examples will show the methodology used to model and simulate mixed-signal blocks in SystemC. An oscillator made up of an inverter chain has been used to test the accuracy and stability of the proposed algorithm. A complex mixed-signal fuzzy controller is used to show the speed up achievable with a high level description of the analog blocks with respect to SPICE simulations.

### 3.1 CMOS inverter chain oscillator

The first simple example is based on the CMOS inverter reported in Fig. 4a. As a first step the input, state and output vectors must be defined and a differential equation of the type (1) must be written. This phase corresponds to the creation of the macromodel of the block. Up to now this is not made automatically, but it must be performed by the user, who establishes the detail level of the description. In this example the description is at very low level, in the second example a very high level description is used.

The circuit of Fig.4a has been modelled by the circuit of Fig.4b, described by the following non-linear differential equations:

\[
\begin{align*}
C_i (V_i, V_{out}) &= C_p (V_{spp}, V_{adp}) + C_e (V_{gm}, V_{dm}) = C_{gs} + C_{gs} + C_{gm} \\
C_{gd} (V_i, V_{out}) &= C_{gd} (V_{spp}, V_{adp}) + C_{gs} (V_{gm}, V_{dm}) \\
dV_i &= \frac{I_{in} - V_i - V_o - V_{out}}{C_i} \\
dV_o &= \frac{V_i - V_o - V_{out}}{R_f \cdot C_{gd}} \\
I_{out} &= I_{mos} (V_i, V_{out}) - I_{mos} (V_i, V_{out}) + I_{cgd} (V_i, V_o, V_{out})
\end{align*}
\]

The variables $I_{in}$ and $V_{out}$, are the inputs, $I_{out}$ and $V_i$ are the outputs and $V_1$ and $V_2$ the states, as shown in Fig.5 reporting the SystemC module. Two activation threads are defined in the module, one for each input module.

A resistance $R_f$ must be introduced to allow a representation of the system in a form of the type of (1). A very low value of the resistance allows the convergence of the algorithm without affecting the accuracy of the results.
Fig. 4a,b Schematic of the inverter and model used in the SystemC description of a CMOS inverter.

Fig. 5 CMOS inverter in SystemC

Fig. 6 SystemC block description of the oscillator

The SystemC header file of the module is the following:

```c
//inverter.h
const float VCC=10.0f;  //supply voltage
const float FACTOR=1.1f;
const float K=1.0f;

SC_MODULE(inverter)
|
\ sc_in<float> i1_in, v2_in, dt1, dt2;
\ sc_out<float> i2_out, v1_out, dt;
\ char *NAME;
\ float WL,WLP,st0;
\ // parameters to be assigned at object instantiation
\ int flag;
\ float dt;
\ float Ipmos (float vi, float vout);
\ float Inmos (float vi, float vout);
\ float Cgs (float wl, float cox, float vgs, float vds, float vth);
\ float Cgd (float wl, float cox, float vgs, float vds, float vth);
\ float Cgb (float wl, float cox, float vgs, float vt);

sc_event activation_event;
void calculus();
void activation1();
void activation2();

SC_CTOR (inverter)
| { SC_THREAD(calculus); //constructor
| sensitive << activation_event;
| //sensitivity list
| SC_THREAD(activation1);
| sensitive << i1_in;
| SC_THREAD(activation2);
| sensitive << v2_in; }
```

The structure represented in Fig.1 and Fig.5 can be recognized. The WLN and WLP (the n_ch and p_ch MOSFET width) and st0 (the initial value of the state) are parameters whose values are assigned at object instantiation. Fig.6 shows the SystemC description of an oscillator consisting of three inverters, modeled as previously reported. Note that each module receives inputs from two modules and its output signals are
connected to two modules. The methodology proposed does not impose limitations on the number of input or output modules.

The accuracy of the transient numerical simulation is critical for oscillating circuits, in spite of their simplicity.

The main file of this analog module is the following:

```c
#include "systemc.h"
#include "inverter.h"
#include "time.h"

int sc_main (int ac, char *av[]) {
    double start, finish, duration;
    sc_set_time_resolution(1, SC_FS);
    sc_set_default_time_unit(1, SC_SEC);
    // signals
    sc_signal<float> in,in2,I1,I2,I3,
    sc_signal<float> V1,V2,V3,dt1,dt2,dt3;
    inverter INV1("INV1"); // instantiation
    inverter INV2("INV2");
    inverter INV3("INV3");
    INV1.WLP=1;  // micron
    INV2.WLN=1;
    INV1.ST0=0;  // initial state
    INV1.i2_out(I1);
    INV1.v1_out(V1);
    INV1.dt2(dt2);
    INV1.dt1(dt3);
    INV1.dt(dt);
    INV1.DTMAX=3.0e-12f;

    INV2.WLN=1;
    ...
    INV3.DTMAX=3.0e-12f;
    start = clock();
    sc_start(40,SC_NS);
    finish = clock();
    exc = (double)(finish - start);
    return 0;
}

void inverter::calculus() // SC_THREAD
{
    float i_in;
    float vout,cl,icgd;
    float state,state2,next_state,next_state2
    float delta,delta1,delta2;
    // init
    i2_out=0;
    v_out=ST0;
    V1=ST0;
    V2=0;
    flag=0;
    dt_aux=DTMIN;
    icgd=0;

    while(true)
    {

    }
}

void inverter::activation1()
{
    float in1=0;
    float in2=0;
    while(true)
    {
        dt_ext=deltat1_in.read();
        in2=il_in.read();
        ...
        do
            (dt=dt_aux;
             next_V1=V1+((i_in/cl)-(V1-V2-
             vout)/(R*cl))*dt;
             delta=fabs(next_V1-V1);
             delta2=fabs(next_V2-V2);
             if (delta>delta2) delta=delta1;
             else delta=delta2;
             if (delta>ϑ)
                 dt_aux/=F;
             if (delta<ϑ)
                 dt_aux*=F;
             if (dt_aux<DTMIN) dt_aux=DTMIN;
         } while ((delta>ϑ)&&(dt>DTMIN));
        t1=sc_simulation_time;
        wait(dt,SC_SEC,activation_event);
        t2=sc_simulation_time;
        if (t2<t1+dt) {
            next_V1=V1+(next_V1-V1)*(t2-t1)/dt;
            next_V2=V2+(next_V2-V2)*(t2-t1)/dt;
            if (flag=1) dt_aux=DTMIN;
        }
        flag=0;
        V1=next_V1;
        V2=next_V2;
        ...
    }
    i2_out=ipmos-inmos+icgd; // output current
    v1_out=V1; // output voltage
}
```
Fig. 7 reports the waveforms of the output voltage of the last inverter, obtained with 5 SystemC simulations:

a) with a constant time step \( dt = 10^{-11} \) s,
b) with a constant time step \( dt = 10^{-12} \) s,
c) with a constant time step \( dt = 10^{-13} \) s,
d) with variable time step with \( DTMAX = 10^{-11} \) s and \( K = 1 \),
e) with variable time step with \( DTMAX = 10^{-11} \) s and \( K = 10 \).

Simulations d) and e) uses the algorithm shown in Fig. 2 and 3. The difference between the oscillation period obtained with \( dt = 10^{-12} \) s and \( 10^{-13} \) s is less than 0.1%, conversely, the effect of the numerical error in the transient simulation is evident in the simulation with a constant time step of \( dt = 10^{-11} \) s. The maximum difference between the oscillation periods obtained with the b) d) and e) simulations is less than 0.3%.

The CPU time required for a transient simulation of 80ns is

a) constant \( dt = 10^{-11} \) s \( \rightarrow \) \( T_{CPU} = 0.15 \) s
b) constant \( dt = 10^{-12} \) s \( \rightarrow \) \( T_{CPU} = 1.4 \) s
c) constant \( dt = 10^{-13} \) s \( \rightarrow \) \( T_{CPU} = 13.8 \) s
d) variable \( dt \): \( K = 1 \) \( \rightarrow \) \( T_{CPU} = 0.49 \) s
e) variable \( dt \): \( K = 10 \) \( \rightarrow \) \( T_{CPU} = 0.35 \) s

The results show that the accuracy of the SystemC simulations with the proposed algorithm (variable time step) is acceptable, with a substantial reduction of the CPU time.

A comparison with WinSpice simulations is reported in Fig. 8, showing the waveforms of the output voltages of the three inverters of the oscillator as obtained from WinSpice simulations, on the left, and from SystemC simulations, on the right.

The maximum time step is fixed to \( 10^{-11} \) s for both, WinSpice and SystemC, in order to make the results comparable. The waveforms are similar and the CPU time required for a 800ns transient simulation are 4.9s for SystemC and 15.8s for WinSpice in the same PC.
This example shows that the proposed algorithm makes it possible the modeling and simulation of analog blocks using the SystemC kernel with an accuracy comparable with SPICE and with a reduction of the CPU time. The gain in terms of CPU time is not relevant, but this is due to the low level description of the inverter used in this example. An higher level view of the block will allow a strong reduction in CPU time, as it will be shown in the next example.

### 3.2 Mixed-Signal Fuzzy Controller

The second example is a complex mixed signal fuzzy controller chip implemented in a 0.35 µm CMOS technology [Orc03, Con00]. The circuit implementing the 3-dimensional fuzzy partition membership function is the core of a mixed-signal Fuzzy Controlled Integrated Circuit. This controller has been designed in a CNR project in collaboration with Merloni Elettrodomestici and Wr@p. The aim of this project is the design and implementation of a mixed signal programmable device for the fuzzy control of “white” household appliances.

The analogue implementation holds a key advantage over digital implementation in applications where computational speed is needed. The chip is designed to be extremely flexible. In stand alone mode it is a fuzzy controller able to elaborate analog and digital signals and to communicate through an I²C bus with an EEPROM, necessary to fix the weights of the fuzzy system. In slave mode, the device acts as a peripheral of a microcontroller.

The digital part, a microcontroller, an I²C driver and a small RAM, is composed of 2864 standard cells, and the analog fuzzy engine is composed of 29673 MOSFETs.

Fig.9 shows the architecture of the fuzzy controller. The analog part consists of 31 membership function blocks, one I/V converter, 3 V/I converters, 125 current multipliers, and 140 blocks of 5-bit D/A converters, used to store the weights of the fuzzy engine.

The fundamental block of the complete system, the membership function circuit (MFx-y in Fig.9) is reported in Fig.10. It consists of current mirrors and 4 differential pairs, biased by the voltages VLi and VRi. The complete system of 31 blocks, of the type reported in Fig.10, implements 125 three-dimensional membership functions [Orc03].

The following formulae describe the dependence of the 5 output currents with the input voltages.
These relationships have been implemented in the SystemC high level macromodel of the membership function block, reported in Fig.11.

In a similar way, all the analog and digital blocks, reported in Fig.9, have been described in SystemC at very high level, using nonlinear analytical models for each block and modeling the delay of analog blocks with an input and an output capacitance. Nevertheless, the functionality of the analog fuzzy engine is accurately described, as can be seen from Spectre simulations (dotted lines) and SystemC simulations (continuous lines) reported in Fig.12. They are practically coincident.

The CPU time required for the Spectre transient simulation is 4h 38’, only 3.3 seconds are necessary for a SystemC simulation. This strong CPU reduction factor (about 5000) is due to the very high level description used in SystemC. In the first phase of the design, in which the chip architecture is not well defined and the correct functionality of the complete System on Chip must be verified, this speed up is very important.

Furthermore, if necessary, a more detailed description on the analog circuit in SystemC is possible using the methodology presented in this work.

The implementation in the same simulation environment, SystemC, of both digital and analog parts enables a fast evaluation of the performances of the complete system. Furthermore, it is possible to model in SystemC the analog process to be controlled by the mixed-signal fuzzy chip. This allows the verification of the design in a real application.

4 Conclusions

The extension of SystemC to analog circuits is mandatory for the design of complex Systems on Chip including DAC or RF front end. Many researches are starting in this field. This work presents a suggestion for this extension. The first results are comforting in term of accuracy of the simulation and CPU time required.
5 Acknowledgments

This research has been sponsored in part by the European Medea+ project.

6 References


[SC_AMS] SystemC-AMS study group: http://mixsig.eas.iis.fhg.de


---

Fig. 9 Architecture of the mixed-signal fuzzy controller
MR4
VL4
ML2
Iout5
VDD
IoML1
VDD
VR1
ML3
VR4
Iout1
Iout2
Iout3
Iout4
Iout5
MR1
MR2
MR3
VL1
VL2
VL3
VL4
VR1
VR2
VR3
VR4

Fig. 10 Membership Function circuit: MFx-y block in Fig.9

Fig. 11 SystemC implementation of the Membership Function circuit of Fig.10

Fig. 12 Five output currents of the first level membership function: DC spectre simulations (dotted lines) and SystemC simulations (continuous lines)