Orthogonal Signal Modeling and Operational Computation of AMS Circuits for Fast and Accurate System Simulation

Leandro Gil, Martin Radetzki
Embedded Systems Department
University Stuttgart
Pfaffenwaldring 5b
D-70569 Stuttgart, Germany
{leandro.gil, martin.radetzki}@informatik-uni-stuttgart.de

***Abstract*—We present a general mathematical model of signals for efficient and accurate simulation of analog and mixed signal (AMS) systems. It relies on signal coding and parameterization and allows heterogeneous system specification at different abstraction levels, as well as, the operational computation of continuous time systems' dynamical behavior. In particular, we derive a matrix for operational subdivision of continuous signals and use it to capture accurately the interaction between continuous and discrete time systems. A key advantage of this signal representation is that continuous signal monitoring and analysis can be performed more efficiently, speeding up system verification. We implemented the proposed modeling approach in SystemC AMS 2.0 to exploit the dynamic reactive behavior of TDF MoC for accurate synchronization between the digital and analog system parts. With the example of a PLL system we evaluate the capabilities of our implementation to cope with heterogeneous designs at different design abstraction levels. The experimental results show a significant simulation speedup for high accurate models.**

I. Introduction and Related Work

Electronic System Level (ESL) emerged to cope with the complexity of embedded system designs. Today's System Level Design Languages and Tools must fulfill widely spread requirements. They should allow specification of *different aspects* of *heterogeneous systems* based on *formal descriptions* for automated analysis and synthesis and they should support *fast execution* of the resulting models for early design verification and validation. SystemC is one of the most commonly used languages in the industry for Electronic System-Level (ESL) design and verification. The modeling and simulation capabilities of SystemC enable the abstract architectural description and simulation based analysis of large digital HW/SW systems [1]. SystemC AMS provides features for modeling and simulation of heterogeneous and Analog and Mixed-Signal (AMS) Systems at several levels of abstraction. Three models of computations (MoCs) support the system design flow at functional, architectural and implementation level. A data flow simulation formalism called *Timed Data Flow (TDF)* allows the discrete time modeling at functional and architectural level. It uses a static scheduling algorithm that computes the module activation order before simulation starts and synchronizes the discrete time execution with the SystemC simulation kernel. The continuous-time behavior simulation relies on a linear differential algebraic equation (DAE) solver, that allows the modeling of linear signal flow components and electrical networks. In order to maintain an acceptable simulation performance while modeling the system's behavior with enough accuracy (major requirement [2]), all MoCs of the SystemC AMS 1.0 standard utilized a fixed time step for computation. The designer was thus forced to choose a time step small enough to ensure the accuracy of the results.

Focused on heterogeneous MoCs, Zhu et al. presented a novel model of continuous-time signals that is defined with time continuum [3]. To simulate dynamical systems, SystemC ForSyDe uses a DAE solver with error estimation for time step control, similar to Ptolemy [4]. The simulation speed and efficiency was not significantly improved compared to SystemC AMS [5]. In the SystemC AMS 2.0 standard, new member functions allowing the dynamic change of TDF attributes during simulation were defined to improve the simulation performance. Barnasconi et al. presented in [1] the new dynamic capabilities of the TDF MoC, enabling event-driven module activation and showing promising results. Exploiting these features, Andrade et al. [6] presented a time step control mechanism that determines the required time step to sample a continuous signal. It is based on the slope between the current and the previous signal value and assumes that there are no discontinuities on the signal. This simple approach doesn't provide a good sampling if the signal has inflexion points. To establish the exact time stamp at which an input signal crosses a threshold value, they also propose a threshold crossing algorithm that decreases or increases the time step in terms of the proximity of the input signal to the threshold reference. The time step control is active in a range window, defined by two additional reference values. This approach is very time demanding.

To cope with the challenge of accurate behavior modeling and high simulation performance, we propose a vectorial signal model for analog and digital signals. It

978-3-9815370-6-2/DATE16/

is presented in section III after some preliminaries in section II. In section IV we present an operational approach to subdivide continuous time signals and explain how to use it for the fast computation of threshold crossing events and the reliable control of continuous time signal steps. We derive in section V operational approaches to compute linear continuous time systems described by functional and non-functional processes. In section VI we present an iterative computational approach for discrete time systems that benefits from the vectorial signal representation. The implementation details are presented in section VII. The results of a PLL simulation are shown in section VIII. Finally, we conclude the paper with a summary of our work in section IX.

II. Preliminaries

In embedded system design, a system is usually represented by a process network. The behavior of a process $P(s(\tau))$ and the interaction between processes is described through signals. A signal $s(\tau)$ is represented as a set of events $\{e_i\}$. Each event e_i maps a tag τ_i to a value v_i.

$$s(\tau) = \{e_i : \tau_i \rightarrow v_i\} \quad \forall \quad \tau_i \in \mathbb{T},\, v_i \in \mathbb{V} \tag{1}$$

The set of tags $\{\tau_i\}$ are partially or totally ordered. This functional representation of signals is known as the "tagged signal model" [7]. Tags are used to capture the main properties of models of computation, such as, time abstraction, precedence relationships and event synchronization. Since signals are the operands and the results of computation processes, an appropriate signal model is essential to allow an efficient implementation of computational processes.

III. Mixed Orthogonal Basis Vectorial Signal Model

In order to achieve a good simulation performance at different levels of abstraction we propose a signal model that is able to represent mixed signals in an efficient way and computes linear processes using simple algebraic mathematical operations.

The function $s(\tau)$ describing a signal can be expressed as an ordered set of values v_i in an infinite dimensional space $\mathbb{V}^\infty$.

$$s(\tau) = [v_i]_\infty \quad , \quad [v_i] \in \mathbb{V}^\infty \tag{2}$$

In this vectorial signal representation, a signal is considered as a point in an infinite dimensional space. Each dimension of the signal corresponds to a tag ($\tau_i \rightarrow i$). The domain $\mathbb{V}$ may be any data type. The geometric properties of vector signals allow to define systematically the mathematical operations that can be applied to signals. For example, it is possible to define closeness of signals as the distance between two points.

In order to optimize the geometrical representation of signals, we describe a signal $s(\tau)$ by a linear combination of elementary signals $s_e(\tau)$, defined in a finite interval $[0, T_e]$ and mapped to a finite interval $[\tau_k, \tau_k + T_k]$, as follows.

$$[v_i]_\infty = \sum_{k=0}^{\infty} a_k \cdot [v_{ki}]_\infty = \sum_{k=0}^{\infty} s_k(\tau) \tag{3}$$

where $s_k(\tau) = s_e(\rho_k \cdot \tau - \tau_k)$ and $\rho_k = T_e/T_k$. The set of elementary signals $\{s_e(\tau)\}$ is know as the *signal space*. Representing signals in terms of elementary signals is useful to model signal sources efficiently. For simulation purposes, we consider real valued[1] elementary signals ($\mathbb{V} = \mathbb{R}$) with finite energy E_{s_k}.

In order to approximate each elementary signal with a finite set of vector elements, we expand each elementary signal $s_e(\tau)$ in terms of a set of *linearly independent signals* $\{f_j(\sigma)\}$, defined in an interval $[\sigma_a, \sigma_b]$. The *signal space* $\{s_e(\tau)\}$ is thus parameterized with respect to a set of orthogonal basis signals by finding the set of coefficients $\{b_{er}\}$ so that:

$$s_e(\sigma) = [b_{er}]_\infty \bullet [f_r(\sigma)]_\infty = \sum_{r=0}^{\infty} b_{er} \cdot f_r(\sigma), \quad b_{er} \in \mathbb{R} \tag{4}$$

A continuous time or analog elementary signal $\tilde{s}_e(\tau)$ can be expanded over a series (infinite linear combination) of continuous orthogonal basis signals $\tilde{f_n}(\sigma)$ by a simple vector-matrix multiplication.

$$\tilde{s}_e(\sigma) = \sum_{n=0}^{+\infty} c_{en} \cdot \tilde{f_n}(\sigma) = [c_{en}]_\infty^T \cdot [\tilde{f_n}(\sigma)]_\infty \tag{5}$$

We consider that each analog elementary signal $\tilde{s}_e(\tau)$ and its first derivative are continuous and mapped to an interval $[\sigma_a, \sigma_b]$. Therefore, they can be approximated with good accuracy (least squares approximation) using a reduced number of orthogonal polynomials Q_n as basis signals.

$$\tilde{s}_e(\sigma) \approx \sum_{n=0}^{N-1} c_{en} \cdot Q_n(\sigma) = [c_{en}]_N^T \cdot [Q_n(\sigma)]_N \tag{6}$$

There is a set of recursion relations common to many classical orthogonal polynomials that are useful to compute numerical operations. The most suitable orthogonal polynomials for analog system simulation are Legendre $P_n(\sigma)$ and Chebyshev $T_n(\sigma)$ polynomials [8] [9]. In both cases, the least square error is distributed nearly uniformly in the approximation interval. The Chebyshev polynomials are defined in the interval $[-1, 1]$ by the following expression:

$$T_n(\sigma) = \frac{n}{2} \sum_{j=0}^{\lfloor n/2 \rfloor} (-1)^j \cdot \frac{(n-j-1)!}{(j)! \cdot (n-2j)!} \cdot (2\sigma)^{n-2j} \tag{7}$$

A discrete event elementary vector signal $\hat{s}_e(\tau)$ can be expanded using a finite linear combination of flat (piecewise constant) orthogonal basis signals $\hat{f_m}(\sigma)$, such as, block pulse, Walsh or Haar-Wavelets.

$$\hat{s}_e(\sigma) \cong \sum_{m=0}^{M-1} b_{em} \cdot \hat{f_m}(\sigma) = [b_{em}]_M^T \cdot [\hat{f_m}(\sigma)]_M \tag{8}$$

For simplicity, we consider discrete orthogonal signals represented by the generalized block pulse function:

[1] Signals may be also complex functions (frequency domain analysis) .

$$B_m(\sigma) = \begin{cases} 1 & \text{if } \sigma_m \le \sigma \le \sigma_m + T_m \\ 0 & \text{otherwise} \end{cases}, \quad \sigma_m = \sum_{k=0}^{m-1} T_k \qquad (9)$$

Equations (6) and (8) allow to define analog and discrete event elementary vector signals in a finite dimensional vector space $\mathbb{V}^N$ and $\mathbb{V}^M$ respectively. Finally, we combine the previous equations to obtain a *general expression for signal representation*. Any elementary signal $s_e(\tau)$ can be expanded using a set of orthogonal piece-wise continuous signals $\{H_{mn}(\sigma)\}$ as follows:

$$s_e(\sigma) \approx \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h_{emn} \cdot H_{mn}(\sigma) = [h_{emn}]_{MN} \cdot [H_{mn}(\sigma)]_{MN} \quad (10)$$

where $H_{mm}(\sigma) = \begin{cases} g_{mn} \cdot Q_n(\sigma) & \text{if } \sigma_{mn} \le \sigma \le \sigma_{mn} + T_{mn} \\ 0 & \text{otherwise} \end{cases}$

The constant function g_{mn} depends on the discrete orthogonal basis set. For the block pulse function $g_{mn} = 1$.

IV. Operational Subdivision of Continuous Time Signals

Any orthogonal polynomial $Q_n(\sigma)$ can be expressed in the following matrix form:

$$Q_n(\sigma) = [x^n]_N^T \cdot [B_Q]_{N^2} \cdot [c_n]_N \qquad (11)$$

where the matrix $[B_Q]_{N^2}$ represents the basis of the polynomial in terms of the power vector $[x^n]_N = [x^N \ \dots \ x^2 \ x \ 1]$. The basis matrix for Chebyshev polynomial can be computed using equation (7). For a 3^{rd} order polynomial the Chebyshev basis matrix is:

$$[B_T]_{4^2} = \begin{pmatrix} 0 & 0 & 0 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & -3 \\ 1 & 0 & -1 & 0 \end{pmatrix} \qquad (12)$$

In order to find an operational approach to subdivide a polynomial function defined in the interval $[-1, 1]$, we first map the function evaluation points to the subinterval $[\sigma_a, \sigma_b]$ using the expression:

$$\sigma_{ab} = \frac{1}{2} \cdot (\sigma_b - \sigma_a) \cdot \sigma + \frac{1}{2} \cdot (\sigma_a + \sigma_b) \qquad (13)$$

and then we find $[B_T(a,b)]_{N^2}$ using the recurrence property of Chebyshev polynomials:

$$T_n(\sigma_{ab}) = \sigma_{ab} \cdot T_{n-1}(\sigma_{ab}) - T_{n-2}(\sigma_{ab}) \qquad (14)$$

Finally, using equation (11) the subdivision matrix $[S_{a,b}]$ can be computed as:

$$[S_Q(a,b)] = [B_Q]_{N^2}^{-1} \cdot [B_Q(a,b)]_{N^2} \qquad (15)$$

Thus, the coefficients $[c_n(a,b)]_N$ of the polynomial expansion for any signal in a given sub-interval $[\sigma_a, \sigma_b]$, can be computed by a simple matrix multiplication.

$$[c_n(a,b)]_N = [S_Q(a,b)] \cdot [c_n]_N \qquad (16)$$

For Chebyshev polynomials of 3^{rd} order, the subdivision matrix is given by:

$$[S_T(a,b)] = \begin{pmatrix} 1 & \frac{a+b}{2} & \frac{3\cdot(a^2+b^2)+2\cdot a\cdot b-4}{4} & \frac{5\cdot(a^3+b^3)+3\cdot(a^2\cdot b+a\cdot b^2)-6\cdot(a\cdot b)}{4} \\ 0 & \frac{b-a}{2} & \frac{b^2-a^2}{2} & \frac{15\cdot(b^3-a^3)+3\cdot(a\cdot b^2-a^2\cdot b)+12\cdot(a-b)}{8} \\ 0 & 0 & \frac{a^2+b^2-2\cdot a\cdot b}{4} & \frac{3\cdot(a^3+b^3-a^2\cdot b-a\cdot b^2)}{4} \\ 0 & 0 & 0 & \frac{b^3-a^3+3\cdot(a^2\cdot b-a\cdot b^2)}{8} \end{pmatrix} \qquad (17)$$

A. Computing Threshold Crossing Events

The polynomial signal representation introduced in section III can be exploited for accurately threshold crossing detection by using polynomial root finding methods. As explained in [10], eigenvalue methods can be applied to find the roots of arbitrary polynomials, but they are computationally intensive and not appropriate for simulation purposes. To overcome this limitation, we profit from the properties of Chebyshev polynomials for the identification of crossing event free intervals. Chebyshev polynomials are ranged between ±1, therefore, the value of a signal represented with Chebyshev polynomials in a given interval $[\sigma_a, \sigma_b]$ is limited to the range:

$$\left[c_0 - \sum_{n=1}^{N-1} |c_n|, \ c_0 + \sum_{n=1}^{N-1} |c_n| \right] \qquad (18)$$

Using precomputed subdivision matrices to divide the signal interval into smaller intervals, we find quickly the intervals containing crossing events. Approximating the polynomial by a 3^{rd} order polynomial, the crossing point is computed analytically.

B. Time Step Control for Continuous Time Signals

To achieve a reliable time step control for continuous time signals, we subdivide successively a signal using the presented operational approach. The recursive subdivision is stopped, when the error that would be introduced by approximating each sub-signal with a line is smaller than a given tolerance value ϵ. We profit from the convergence properties of Chebyshev polynomials to approximate the truncation error with good accuracy. The simplest approach for interval partitioning is to apply a binary subdivision (the interval is partitioned into two equidistant subintervals). In order to distribute the sampling error more uniformly, we apply a non-symmetric signal subdivision. To this end, we compute the points, where the distance between the line connecting the start and end value of the signal and the signal curve is maximal. To determine such points, we profit again from Chebyshev polynomials properties. The subdivision point where the distance between the line and the signal curve achieve the maximal value can be computed solving the following equation:

$$\frac{d}{d\sigma}(12\cdot c_3\cdot\sigma^2+4\cdot c_2\cdot\sigma+c_1-3\cdot c_3-\bar{c}_1) = 0, \quad \bar{c}_1 = \sum_{n=0}^{N/2} c_{2\cdot n+1} \qquad (19)$$

As shown in Fig. 1 for a sine wave form ($\epsilon = 0.005$), this approach leads to a very good time step control.

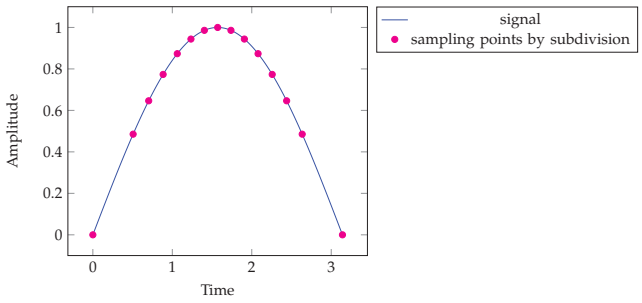

Fig. 1: Time step control using successive subdivisions

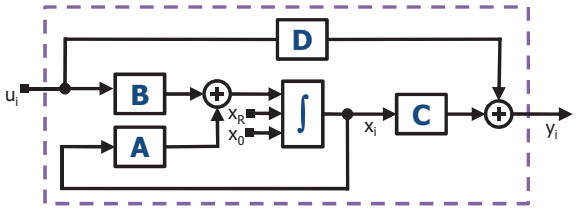

Fig. 2: Linear functional system with reset

V. Operational Computation of Continuous Time Processes

Processes specify relations among signals and are domain dependent (or polymorphic). Based on the presented signal model, we derive now the set of equations enabling the operational computation of linear process behavior for functional and non-functional processes.

If the processes of a continuous time domain system are linear, time invariant and described by functional signal relations, the whole system can be represented by a signal flow model as shown in Fig. 2. The system behavior is defined by the following equation system:

$$\begin{aligned} \tfrac{d}{dt}[x_i(\tau)]_P &= [A]_{PP} \cdot [x_i(\tau)]_P + [B]_{PQ} \cdot [u_i(\tau)]_Q \\ [y_i(\tau)]_R &= [C]_{RP} \cdot [x_i(\tau)]_P + [D]_{RQ} \cdot [u_i(\tau)]_Q \end{aligned} \tag{20}$$

The continuous dynamic behavior of the system is characterized by the vector of P continuous time signals $[x_i(\tau)]_P$, which is known as the *continuous state vector* of the system. $[u_i(\tau)]_Q$ and $[y_i(\tau)]_R$ are the *input signal vector* and the *output signal vector* respectively. Replacing the system signals $[x_i(\tau)]_P$ and $[u_i(\tau)]_Q$ by the mixed signal approximation (eq. 10) and considering the following *operational properties* of the *basis signal matrix* $[H_{mn}(\sigma)]_{MN}$,

$$\begin{aligned} \tfrac{d}{dt}[H_{mn}(\sigma)]_{MN} &\approx [D_h]_{(MN)(MN)} \cdot [H_{mn}(\sigma)]_{MN} \\ \int_{\sigma_a}^{\sigma_b}[H_{mn}(\sigma)]_{MN} &\approx [P_h]_{(MN)(MN)} \cdot [H_{mn}(\sigma)]_{MN} \end{aligned} \tag{21}$$

we convert the explicit ordinary differential equation (ODE) in (20) into the algebraic equation

$$\begin{aligned} &[X_{mn}]_{MNP} - [\bar{A}]_{PP} \cdot [X_{mn}]_{MNP} \cdot [P_h]_{(MNP)(MNP)} = \\ &[\bar{B}]_{PQ} \cdot [U_{mn}]_{MNQ} \cdot [P_h]_{(MNQ)(MNQ)} + [X_{0mn}]_{MN} \end{aligned} \tag{22}$$

The system matrices $[A]_{PP}$ and $[B]_{PQ}$ are modified due to the variable mapping from τ to σ. The operational matrices of differentiation $[D_h]_{(MNP)(MNP)}$ and integration $[P_h]_{(MNP)(MNP)}$ are approximated inverses and both can be used in the previous equation. Using the Kronecker product properties, we obtain a vectorial representation of equation (22) which can be solved by direct methods.

$$\begin{aligned} &([I] - [\bar{A}]_{PP} \otimes [P_h]^T_{(MNP)(MNP)}) \cdot vec([X_{mn}]_{MNP}) = \\ &vec([\bar{B}]_{PQ} \cdot [U_{mn}]_{MNQ} \cdot [P_h]_{(MNQ)(MNQ)} + [X_{0mn}]_{MNP}) \end{aligned} \tag{23}$$

If the processes of a linear system are not functional, such as the component equations of an electrical network, the resulting equation system is described by the following differential algebraic equation (DAE):

$$\begin{aligned} [E]_{TT} \cdot \tfrac{d}{dt}[z_i(\tau)]_T &= \\ [F]_{TT} \cdot [z_i(\tau)]_T + [G]_{TS} \cdot [w_i(\tau)]_S & \end{aligned} \tag{24}$$

The continuous state and output signals are contained in the signal $[z_i(\tau)]_T$. The input signal vector is denoted by $[w_i(\tau)]_S$. Applying the same steps as in the functional case, the system equation (24) can be also transformed into a linear algebraic equation:

$$\begin{aligned} &([I] \otimes [E]_{TT} - [\bar{G}]_{PP} \otimes [P_h]^T_{(MNT)(MNT)}) \cdot vec([Z_{mn}]_{MNT}) = \\ &vec([\bar{F}]_{TT} \cdot [W_{mn}]_{MNS} \cdot [P_h]_{(MNS)(MNS)} + [E]_{TT} \cdot [Z_{0mn}]_{MNT}) \end{aligned} \tag{25}$$

If the system processes are functional but not linear, the operational computation becomes very complex. However, the properties of Chebyshev polynomials allow a simple and efficient approximation and evaluation of continuous time signals.

VI. Computing AMS Processes

We execute digital and analog circuits using a data flow formalism. Each process reads and/or writes signals represented in vectorial form. It may request a *next process execution time*. In each *evaluation cycle*, the *execution time* is computed by the cluster manager and the processes are called sequentially. When a process is executed, it reads all input signals from the ports and determines the next *local event time*. After updating the *local process time*, the *event processing function* is executed. This step is repeated until all input signal events are read. Signals are defined in the same time interval (according to the *execution time*) but the number of events present on signals is arbitrary (not limited to a constant rate). Thereby, the *event processing function* may be executed several times during process execution. If the process doesn't have input ports, the *event processing function* is executed once per cycle. The resulting events may be written to the output signals with any arbitrary time delay value. For systems containing feedback loops, the evaluation cycle is repeated until signal convergence is achieved. Because they normally contain sequential processes, the *logic state* of the processes is stored in a special process variable. If an evaluation cycle is repeated to achieve signal convergence, the state value is restored.

VII. Implementation in SystemC AMS

We implemented our *orthogonal signal flow (OSF)* MoC as extension of the *synchronous data flow model TDF*, provided by SystemC AMS. All main components are defined in the namespace **sca_osf**. Module, port and signal classes are derived from respective TDF classes, supporting hierarchical model construction. Input and output port converters are provided for the interaction with SystemC and SystemC AMS model parts. The dynamic features of the TDF MoC allow the time-accurate interaction with discrete event parts. For modeling of *logic states*, we have implemented an additional class enabling solver synchronization. Port functions **read**, **initialize** and **write** are overloaded to provide signal read and writing operations at the *current process time* for a single value or an array of values (continuous time signals). The *current process time* can be advanced using the *osf function* **set_event_time** for synchronous read and write operations on module ports. The process time advance is limited by the *cluster next time step*. The *osf functions* **initialize_event** and **write_event** allow event writing on port signals at arbitrary times, supporting therefore asynchronous signals. During process execution, the *current process time* is iteratively updated to the next input signal event time and then, the *virtual function* **event_processing** is automatically called. Event times are defined relative to the signal start. The start and the end time points of module output signals can be obtained with the *osf functions* **get_initial_time** and **get_final_time** respectively. The delay properties of the process are specified in the module output ports using the functions **set_delay**, for a deadlock-free schedule, and **set_event_delay** for an arbitrary signal delay time. The simulation accuracy is controlled module-wise using the port functions **set_time_accuracy** and **set_signal_accuracy** to define the maximal admissible time and value deviation of discrete and continuous time signals respectively. The default attribute values for signal accuracy control are 1 *ns* and 1×10^{-6}. Listing 1 and 2 show the implementation of a signal comparator and a RS flip-flop with the OSF MoC. They are utilized in the PLLs' digital phase detector.

Listing 1: Signal threshold comparator using OSF MoC

```
1  class sca_cmp: public sca_osf_tdf_module
2  { public:
3   ::sca_osf::sca_tdf::sca_in<double> in;
4   ::sca_osf::sca_tdf::sca_out<bool>  out;
5
6   sca_cmp(sc_core::sc_module_name _name):
7   sca_osf_tdf_module(_name), in("in"), out("out")
8   {threshold = 0.0}
9
10  void event_processing(void){  double time;
11   in.compute_threshold_events(events, threshold);
12   for(it = events.begin(); it!= events.end(); ++it){
13   time = *it+0.5*out.get_tag_accuracy();
14   out.write_event(*it, in.read_event(time)>
15                   threshold);}}
16   private:
17   std::vector<double> events;
18   std::vector<double>::iterator it;
19   double  threshold;
20 };
```

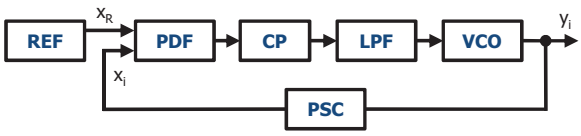

Fig. 3: PLL diagram

Listing 2: RS Flip-flop implementation using OSF MoC

```
1  class sca_ff_rs: public sca_osf_tdf_module
2  { public:
3   ::sca_osf::sca_tdf::sca_in<bool>  r, s;
4   ::sca_osf::sca_tdf::sca_out<bool> q, qb;
5   ::sca_osf::sca_state<bool> qin, qbin;
6
7   sca_ff_rs(sc_core::sc_module_name _name):
8   sca_osf_tdf_module(_name),
9   r("r"), s("s"), q("q"), qb("qb"){}
10
11  void set_attributes(void) {
12      q.set_timestep(1,SC_NS);  q.set_delay(1);
13      qb.set_timestep(1,SC_NS); qb.set_delay(1);
14      q.set_event_delay(1.0e-9);
15      qb.set_event_delay(1.0e-9); }
16
17  void initialize(void)
18  { q.initialize(false); qb.initialize(false); }
19
20  void event_processing(void) {
21      if((s.read()==1)&&(r.read()==0))
22          { qin  = 1; qbin = 0; }
23      if((s.read()==0)&&(r.read()==1))
24          { qin  = 0; qbin = 1; }
25      if((s.read()==1)&&(r.read()==1))
26          { qin  = 0; qbin = 0; }
27      q.write(qin); qb.write(qbin);}
28 };
```

The implementation of linear continuous processes at architecture and implementation (electrical) level is similar to LSF and ELN MoCs in SystemC AMS. A *spawn process* computes in each evaluation cycle the solution of the linear equation systems presented in section V. We apply equation (25) to compute switched electrical linear networks as proposed in [11].

VIII. Experimental Results

For the evaluation of our computational model to cope with heterogeneous designs at different abstraction levels, we implemented a PLL system similar to the SystemC AMS PLL model presented in [12]. As shown in Fig. 3, the PLL consists of a reference signal generator (REF), a phase-frequency detector (PFD), a charge pump (CP), a low-pass filter (LPF), a voltage controlled oscillator (VCO) and a prescaler (PSC). The VCO is modeled at functional and architecture level (signal flow), using the operational approach presented in section V. The charge pump and the analog filter are modeled at electrical level using the operational approach for non-functional systems, presented in the same chapter. The digital parts (PFD and PSC) are modeled at register transfer level applying the computational model described in section VI. The interaction between the analog and digital parts is computed using the subdivision approach presented in section IV. We chose $N = 8$ coefficients for representing

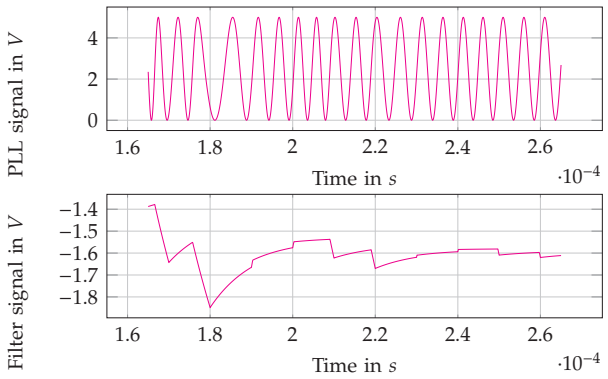

Fig. 4: PLL simulation results

sine waveforms, leading to a maximal absolute error of 6.79×10^{-6}. For the computation of the digital and linear system parts we used the default accuracy values. The cycle execution time is determined by the reference signal period (10 μs). Fig. 4 shows the results of system simulation. After a transient of 250 μs, the desired output frequency is reached. To evaluate the impact of our implementation (OSF MoC) on simulation performance, we also implemented the PLL system using discrete event (DE MoC) and (timed) synchronous data flow modeling (TDF MoC) for the digital system parts. The analog parts of the additional models were implemented using ELN, LSF and TDF MoCs. A time step of 1 *ns* is required to model processes delays. As shown in table I, the calculation of a static schedule during model initialization leads to an improvement in the simulation performance for TDF based models. Because our implementation keeps the process modeling and communication more abstract and efficient, a higher simulation performance can be obtained. In particular, a very efficient analog signal tracing can be achieved for validation purposes. Only few coefficients are necessary to represent continuous signals over a large time interval. To analyze the accuracy of the linear process computation, we compared the open loop analog filter simulation results using the trapezoidal rule (such as in ELN) and the operational computation (with N=5). As reference a transition matrix based computation was utilized. The resulting mean square errors were 17.48×10^{-3} and 2.63×10^{-8} respectively.

TABLE I: PLL simulation execution time in *ms*

Signal tracing	DE MoC	TDF MoC	OSF MoC
off	1023.0	801.4	502.1
on	22819.8	21728.2	2048.5

IX. Conclusion

We presented a mathematical signal model for abstract modeling of analog and discrete process communication and derived efficient computational approaches for continuous time signal threshold crossing detection and time step control. We also explained how to compute the behavior of linear systems using our signal model and extended SystemC AMS for the efficient modeling of digital parts involving feedback loops. The significant simulation performance improvement obtained in the evaluation, in particular, if analog signals are traced, reinforce the importance of our contribution for the system level design of AMS circuits.

Acknowledgment

The authors would like to thank the German Federal Ministry of Education and Research (BMBF) for financial support of this work within the project POWERBLOCK+ (grant number 16M3200F).

References

[1] M. Barnasconi, K. Einwich, C. Grimm, T. Maehne, and A. Vachoux, "Advancing the SystemC analog/mixed-signal (AMS) extensions, Introducing dynamic timed data flow," OSCI, Tech. Rep., Sep 2011. [Online]. Available: http://www.accellera.org/resources/articles/amsdynamictdf (visited on 12/08/2015)

[2] C. Grimm, A. Barnasconi, M. Vachoux, and K. Einwich, "An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions," Open SystemC Initiative, Tech. Rep., June 2008.

[3] J. Zhu, I. Sander, and A. Jantsch, "Hetmoc: Heterogeneous modelling in systemc," in *Specification Design Languages (FDL 2010), 2010 Forum on*, Sept 2010, pp. 1–6.

[4] J. Liu, X. Liu, T.-K. J.Koo, B. Sinopoli, S. Sastry, and E. Lee, "A hierarchical hybrid system model and its simulation," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 4, 1999, pp. 3508–3513 vol.4.

[5] S. Attarzadeh Niaki, M. Jakobsen, T. Sulonen, and I. Sander, "Formal heterogeneous system modeling with systemc," in *Specification and Design Languages (FDL), 2012 Forum on*, Sept 2012, pp. 160–167.

[6] L. Andrade, T. Maehne, M.-M. Louërat, and F. Pêcheux, "Time Step Control and Threshold Crossing Detection in SystemC AMS 2.0," in *Actes du huitième colloque du GDR SOC-SIP du CNRS*. CNRS, Jun. 2013, p. 3. [Online]. Available: http://www.lirmm.fr/ w3mic/SOCSIP/index.php/colloque/colloque-2013/

[7] E. Lee and A. Sangiovanni-Vincentelli, "Comparing models of computation," in *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*, Nov 1996, pp. 234–241.

[8] O. Palusinski, F. Szidarovsky, M. Abdennadher, C. Marcjan, and K. Reiss, "Accelerated simulation of integrated circuits using chebyshev series," in *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, vol. 1, May 1992, pp. 89–92 vol.1.

[9] I. Lazaro, G. Pineda, E. Espinosa, and S. Zavala, "Analysis of time varying power system loads via chebyshev polynomials," in *Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA '08*, Sept 2008, pp. 332–337.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[11] L. Gil and M. Radetzki, "Systemc ams power electronic modeling with ideal instantaneous switches," in *Specification and Design Languages (FDL), 2014 Forum on*, vol. 978-2-9530504-9-3, Oct 2014, pp. 1–8.

[12] T. Xu, H. Arriens, R. van Leuken, and A. de Graaf, "A precise systemc-ams model for charge pump phase lock loop with multi-phase outputs," in *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, Oct 2009, pp. 50–53.