# Extending SystemC to Support Mixed Discrete-Continuous System Modeling and Simulation

Alain Vachoux
Microelectronic Systems Laboratory
Swiss Federal Institute of Technology
Lausanne, Switzerland
alain.vachoux@epfl.ch

Christoph Grimm
Professur Technische Informatik
Universität Frankfurt
Frankfurt, Germany
grimm@ti.informatik.uni-frankfurt.de

Karsten Einwich
Design Automation Dept
Fraunhofer IIS/EAS
Dresden, Germany
karsten.einwich@eas.iis.fhg.de

***Abstract*—Systems on chip are more and more heterogeneous and include software, analog/RF and digital hardware, and non-electronic components such as sensors or actuators. The design and the verification of such systems require appropriate modeling means to deal with the increasing complexity and to achieve efficient simulation. SystemC is providing a modeling and simulation framework that supports digital (discrete) hardware and software systems from abstract specifications to register transfer level models. In the paper, we are proposing a way to extend the capabilities of SystemC to support mixed discrete-continuous systems by implementing a synchronous dataflow (SDF) model of computation (MoC). The SDF MoC is used to embed continuous-time behavior in SDF modules and to support the synchronization with the existing SystemC kernel. The paper presents an overview of the architecture and the syntax of the proposed extensions and gives modeling examples with simulation results.**

## I. INTRODUCTION

Systems on chip are more and more heterogeneous and include software, analog/RF and digital hardware, and non-electronic components such as sensors or actuators. The design of such systems requires dealing with several design formalisms or models of computation (MoC). A model of computation defines a set of rules to mimic (model) a particular behavior. Examples of MoCs are: the discrete-event MoC, the finite state machine MoC, the dataflow MoC, and the continuous-time MoC. The Ptolemy project [1] developed a multi-MoC framework based on the Java language that became a research reference when discussing issues related to the design of heterogeneous systems. SystemC is a set of C++ classes and methods that provides a means to describe the structure and the behavior of hardware/software systems from abstract specifications to register transfer level (RTL) models [2]. SystemC currently only supports the discrete-event MoC, but it is easily extendable to support other MoCs. This paper reports on an effort to extend SystemC to support the modeling and simulation of continuous-time systems and mixed discrete-event/continuous-time systems at different levels of abstraction [6]. The paper is organized as follows. Section II presents the context in which the presented extensions are being developed. Section III details the architecture of the extensions and the new syntax that has been added to SystemC. Sections IV and V give two modeling examples using the new syntax and show some simulation results. Finally, conclusions are drawn in Section VI.

## II. CONTEXT OF THE EXTENSIONS

The first set of targeted applications are signal processing dominated applications that have a number of specific characteristics. First, application behavior may be described as signal-flow models, e.g. they may be represented as directed graphs where nodes define signal processing functions as input-output signal mappings and arcs define the order in which the functions have to be executed. Second, signals have either discrete or real values sampled at equidistant discrete times. Sampling rates are often order of magnitude larger than operating frequencies in the system (over-sampled systems). Third, signal processing functions may be described as input-output functions (e.g. transfer functions), static non-linear or dynamic linear behaviors.

The Synchronous Dataflow (SDF) model of computation [3] is a natural candidate to model and simulate signal processing applications. A dataflow model is a network of concurrent processes communicating through unidirectional unbounded FIFOs. Processes implement the functions (computations) and produce/consume data tokens in FIFOs. Synchronous dataflow models restrict the number of consumed/produced tokens to be constant, so it is possible to statically define an execution order for the processes with bounded FIFO sizes. Single-rate SDF models consume and produce one token at a time, while multi-rate SDF models consume/produce tokens at various rates. For modeling signal processing applications, tokens represent data samples and sampling rate(s) are related to some global clock. It is possible to implement the SDF MoC in SystemC using the existing FIFO primitive channel [4]. However, the

SystemC kernel uses a dynamic scheduling and signal handshaking between processes and so cannot exploit the capability of SDF models to be statically scheduled. This approach therefore leads to significant simulation overhead as excessive context switching is generated. To overcome this limitation, a recent work [5] developed extensions to SystemC that implements the SDF MoC without requiring the use of signal ports. Communication between SDF modules is done through queue data structures and a number of modeling guidelines have to be followed to properly define the SDF model. The approach requires some modifications of the SystemC simulation kernel. Execution comparisons with the approach using SystemC FIFOs showed up to 75% improvement in simulation time.

The approach presented in this paper uses specialized SDF ports, signals and modules whose definitions are inherited from existing SystemC declarations. Furthermore, our approach does not require any change to the SystemC simulation kernel and permits a static scheduling of SDF modules before simulation. It should be stressed that the work described in this paper is only one step towards supporting continuous-time (CT) MoC and providing a framework to seamlessly plug-in specialized CT solvers [6].

## III. Extension Architecture & Syntax

The proposed extensions are developed using a layered approach (Fig. 1).

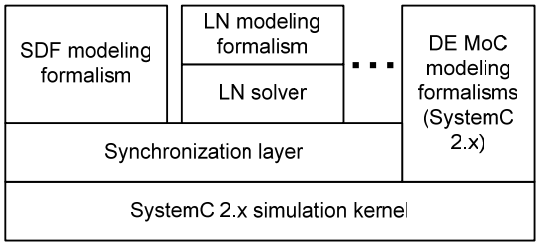


Figure 1. Mixed SDF-DE MoC implementation.

The base layer is the existing SystemC 2.x simulation kernel. The discrete-event (DE) MoCs available in SystemC 2.x are directly built on top of it. Continuous-time MoCs can be added on the top of a synchronization layer which is responsible of the coordination between the SystemC kernel and specific continuous-time solvers, for example a linear network (LN) solver (not discussed in this paper).

The SDF modeling formalism is based on a new kind of module: the SDF module, which is defined as follows (the sca_ prefix is used to denote new language elements):

```
SCA_SDF_MODULE(module-name) {
  // ports
  sca_sdf_in<type> port-name(s);
  sca_sdf_out<type> port-name(s);
  // SDF member functions
  virtual void attributes() { ... }
  virtual void init() { ... }
  virtual void sig_proc() { ... }
  virtual void post_proc() { ... }
  // constructor
  SCA_CTOR(module-name) { ... }
};
```

The SDF ports are bound to a new specific kind of channel that supports SDF communication and a sca_sdf_signal<*type*> declaration is available for defining compositions of SDF modules. The attributes() function allows for defining port attributes such as delay, sampling rate, or time step. The init() function includes initialization code that is executed just before simulation starts. The sig_proc() function is the only mandatory function as it defines the module's behavior or function. The post_proc() function allows for defining code that is executed once the module's function has been evaluated. Finally, the SCA_CTOR constructor does not need to register the above functions, but it may include some initialization code.

A SCA_SDF module is a primitive of the SDF MoC and therefore cannot include submodules. The structural composition of SCA_SDF modules can be done in a regular SystemC SC_MODULE. Currently, a SCA_SDF module cannot declare regular SystemC ports. A special port type has to be used for communication with DE MoCs.

During elaboration, SDF modules are grouped into dataflow clusters according to their connections. Modules in dataflow clusters are then scheduled according to the signal flow direction. In case of cyclic dependencies, delays of one at least one sampling period must be added to "break" the loops. It is assumed that the sampling period is at least two times smaller than the smallest not negligible time constant in the modelled system, so the effect of the delay should not affect the simulation results.

## IV. Example 1: PLL System

This example is used to illustrate the use of the proposed SystemC extensions. A phase-locked loop (PLL) system basically includes three components: a phase comparator, a loop filter, and a voltage controlled oscillator (Fig. 2).

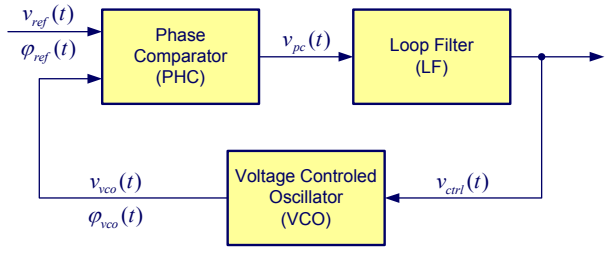


Figure 2. Basic PLL system.

The PLL component is described in a regular SC_MODULE and instantiates three SDF modules as follows:

```
SC_MODULE(pll) {
  sca_sdf_signal<double> ref, pco, lpo, vcoo;
```

```
    phc i_phc("phase comparator");
      i_phc.in1(ref);
      i_phc.in2(vcoo);
      i_phc.out(pco);
      i.phc.kpc = 3.72;   // gain
    lp1 i_lp1("lowpass filter");
      i_lp1.in(pco);
      i_lp1.out(lpo);
      i_lp1.fp = 112e3;   // pole frequency
    vco i_vco("voltage controlled oscillator");
      i_vco.in(lpo);
      i_vco.out(vcoo);
      i_vco.out.set_delay(1);   // feedback loop
      i_vco.kvco = 3e4;   // sensitivity
      i_vco.fc = 7e6;   // central frequency
};
```

Note the specification of a delay at the output port of the VCO to allow a proper scheduling of the SDF cluster. The model of the VCO is given next to illustrate a typical SDF module:

```
SCA_SDF_MODULE(vco) {
  sca_sdf_in<double> in;
  sca_sdf_out<double> out;
  // parameters
  double gain, kvco, fc, vfc;
  // derived parameters
  double wc, kvcor;
  // SDF functions
  void init() {
    wc = 2.0*M_PI*fc; kvcor = 2.0*M_PI*kvco; }
  void sig_proc() {
    double tn = sc_time_stamp().to_seconds();
    double wvco = (wc + kvcor*(in.read() - vfc));
    out.write(gain*sin(wvco*tn)); }
  SCA_CTOR(vco) {}
};
```

One testbench (Fig. 3) for the PLL model uses a sinusoidal source at the same frequency as the central frequency of the VCO. The sinusoidal source is described as a SDF module and its output port time step attribute gets the simulation time step value as:

```
src_sin.out.set_T(sc_time(1.0, SC_NS));
```

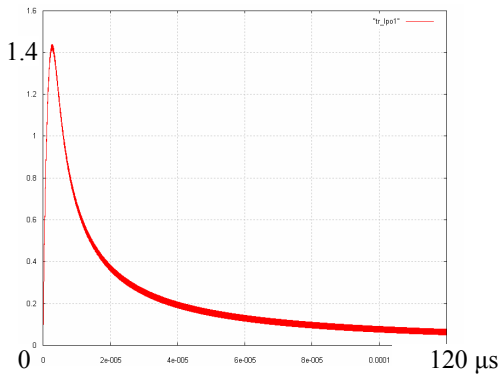


Figure 3. PLL simulation result: output of the lowpass filter. This shows the initial transient of the VCO control signal and its return back to the value generating the central frequency of the VCO.

## V. EXAMPLE 2: PWM CONTROLLER

Compared with existing tools such as Simulink or SPW which are applied at similar levels of abstraction, SystemC-AMS supports as well a through-going refinement process from an executable specification to different mixed-signal architectures. Fig. 4 shows the block diagram of a PWM control loop. Digital pulses of programmable width are generated by a pulse former. The pulses result to an average voltage *UC* in the power driver. The average voltage is controlled by a PI controller which corrects the pulse width depending on the desired voltage *Uprog*.

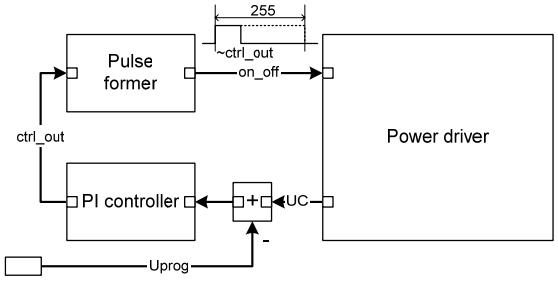


Figure 4. PWM Controller.

### A. Executable Specification

In SystemC-AMS, the executable specification can be given by a few lines of code that instantiate the blocks, and connect them in the SDF model of computation. We assume that the Voltage *Uprog* comes from a digital controller via a bus interface. In order to validate the specification we use very small time steps, and double signals:

```
// code extract
sca_sdf_signal<double> uc, dev, Uprog, corr, on_off;

busif bif1("bif1");
  bif1.out(Uprog);
  bif1.out.set_T(sc_time(0.005,SC_MS));
sca_s_pi_ctrl ctrl("ctrl");
  ctrl.x(deviation);
  ctrl.y(correction);
  ctrl.k=10.0; ctrl.T=10.0;
sca_spartial load("load");   // transfer function
  load.x(on_off);
  load.y(uc);
  load.add_pole(255, -1/0.05);
pulse_gen_de pulsegen1("pulsegen1");
  pulsegen1.in(correction);
  pulsegen1.out(on_off);
```

The behavior of each block is specified as abstract as possible. For simulation of the transfer function sca_partial, the component load uses explicitly solved equations. As an example the code of the PI controller is given below.

```
// PI controller
SCA_SDF_MODULE(sca_s_pi_ctrl) {
  sca_sdf_in<double> x;
  sca_sdf_out<double> y;
```

```
    void sig_proc() {
       sc_time now=simcontext()->time_stamp();
       sc_time t = now-last_change;
       last_change = now;
       state += x.read()*t.to_seconds();
       y.write( k * (T*state + x.read() ) ); }
    SCA_CTOR(sca_s_pi_ctrl) {
       last_change = sc_time(0,SC_SEC);
       state = 0.0; }
    double k, T;
    protected:
       double state; sc_time last_change;
};
```

### B. *Computation accurate model*

In order to evaluate different system architectures, one can easily modify the MoC, signal types or parameters of the model to mimic the behavior of the intended system. In order to analyze a digital implementation of adder, PI controller and pulse former, we use the signals of type sc_int<*bitwidth*>, and time steps that are similar to the sample frequency of an assumed digital implementation.

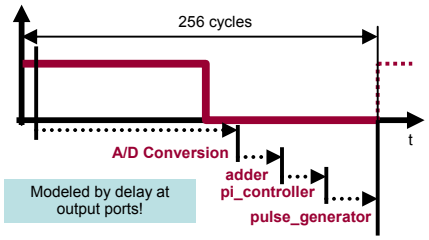


Figure 5. Time budgeting.

In order to evaluate timing of digital blocks, we add delays to the output ports of each module (Fig. 5):

```
// code extract
sca_sdf_in< sc_uint<BW> > in;
void attributes() {
      out.set_delay( clocks ); }
```

The resulting model describes important properties of an architecture and permits a comparison of different variants. Fig. 6 shows simulation results for different bit widths.

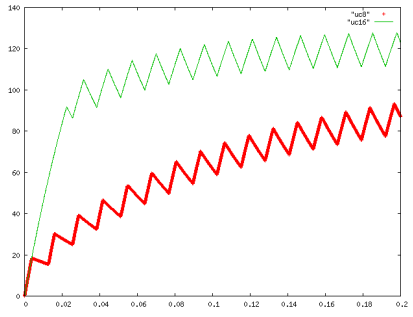

Figure 6. Comparison of different bit widths.

### C. *Interface accurate model and integration with HW and SW design*

For an implementation of an evaluated architecture *interface accurate* models are required. An interface accurate model provides all signals of the implementation. If the PWM model is used for development of a SW system that integrates a PWM, a transaction level interface to the software is required. For this issue, we replaced the simple model of the bus interface by a more complex one that uses Master/Slave Library in order to model the bus protocol:

```
// code extract
// internal registers, symbolic names and address
sc_inslave<sc_uint<BW> > inline;
sc_outslave<sc_uint<BW> > outline;
#define RESET 0 // registers[0] resets controller
#define CONFIG_REG 1 // registers[1] stores config.
#define COEFF_k 2 // registers[2] stores k of PI ctrl.
sc_signal< sc_uint<BW> > registers[128];
```

The interface accurate model as well includes a definition of a relation between the abstract size and a physical size (*UC*) for the transition to a netlist in the analog power driver.

## VI. Conclusions

This paper presented an extension of SystemC to support the modeling and the simulation of mixed discrete-continuous systems that is based on the synchronous dataflow model of computation. The approach is well suited for signal processing dominated applications and integrates seamlessly with the existing discrete-event simulation kernel of SystemC. Efficient simulation is achieved thanks to the clustering of dataflow modules and static scheduling. The approach also supports the refinement process from executable specifications to different mixed-signal architectures. Work is under way to add another model of computation supporting linear networks and linear dynamic behaviors. Such models would be embedded in synchronous dataflow clusters, hence extending the approach presented in this paper.

## References


[1] J. Eker, et al., Taming heterogeneity - the Ptolemy approach, Proc. of the IEEE , Volume: 91 , Issue: 1 , Jan. 2003, pp.127 - 144.

[2] T. Grötker, S. Liao, G. Martin, S. Swan, System Design with SystemC, Kluwer Academic Publishers, 2002.

[3] E.A. Lee, D.G. Messerschmidt, Synchronous data flow, Proceedings of the IEEE, vol. 75, no. 9, 1987.

[4] B. Niemann, F. Mayer, F. Javier, R. Rubio, M. Speitel, Refining a High Level SystemC Model, in SystemC: Methodologies and Applications, W. Müller and W. Rosenstiel Ed., Kluwer Academic Publishers, 2003.

[5] H.D. Patel, S.K. Shukla, Towards A Heterogeneous Simulation Kernel for System Level Models: A SystemC Kernel for Synchronous Data Flow Models, Proc. of International Symposium of VLSI Systems, IEEE Computer Society Press, 2004.

[6] A. Vachoux, C. Grimm, K. Einwich, SystemC-AMS Requirements, Design Objectives and Rationale, Proc. 2003 Design Automation and Test in Europe (DATE 2003), Munich, Germany, 2003.