Modeling embedded systems using SystemC extensions

Open SystemC Initiative

The Open SystemC Initiative (OSCI) is an independent, not-for-profit association composed of a broad range of organizations dedicated to supporting and advancing SystemC as an open industry standard for system-level modeling, design and verification.

There is a trend toward tighter interaction between embedded hardware/software (HW/SW) systems and their analog physical environment. This leads to systems in which digital HW/SW is functionally interwoven with analog and mixed-signal (AMS) blocks, as shown the communication system in *Figure 1*. Examples of these embedded analog/mixed signal (E-AMS) systems include cognitive radios, sensor networks and systems for image sensing. A challenge to their development involves understanding the interaction between HW/SW and AMS subsystems at the architectural level. This requires some means of modeling and simulating these interacting systems.

SystemC supports the refinement of HW/SW systems to RTL by providing a discrete-event (DE) simulation framework. A methodology for the generalized modeling of communication and synchronization that builds on this framework is available using transaction level modeling (TLM) techniques. However, the SystemC simulation kernel has not been designed for the modeling and simulation of analog, continuous-time systems and lacks a refinement methodology that describes analog behavior from the functional level down to the implementation level.

System-level tools are often used for functional modeling and simulation. They may capture continuous-time behavior, but they do not target the design of E-AMS systems at the architectural level. Hardware description languages (HDLs) such as VHDL-AMS and Verilog-AMS target the design of mixed-signal subsystems close to the implementation level, but have limited capabilities regarding HW/SW co-design at high levels of abstraction. Existing co-simulation solutions mixing SystemC and Verilog/VHDL-AMS do not provide high enough simulation performance and lack a seamless design refinement flow for modeling mixed discrete-event/continuous-time systems and HW/SW systems at the architectural level.

In response to demand from the telecoms, automotive, and semiconductor industries, SystemC AMS extensions are being introduced, providing a uniform and standardized methodology for modeling E-AMS systems. This paper gives an overview of those extensions.

Source: OSCI

```
01  SC_MODULE(lp_filter_lsf)
      // Hierarchical models use SC_MODULE class
02  {
03    sca_tdf::sca_in<double> in;
        // Input/outputs in data flow MoC
04    sca_tdf::sca_out<double> out;
05    sca_lsf::sca_sub* sub1;
        // Subtractor
06    sca_lsf::sca_dot* dot1;
        // Differentiator
07    sca_lsf::sca_tdf_source* tdf2lsf1;
        // TDF -> LSF converter
08    sca_lsf::sca_tdf_sink* lsf2tdf1;
        // LSF -> TDF converter
09    sca_lsf::sca_signal in_lsf, sig, out_lsf;
        // LSF signals
10    lp_filter_lsf(sc_module_name, double fc=1.0e3)
        // Constructor with parameters
11    {
12      tdf2lsf1 = new sca_lsf::sca_source("tdf2lsf1");
          // TDF->LSF converter
13      tdf2lsf1->x(in);
          // Port/signal binding like SystemC
14      tdf2lsf1->y(in_lsf);
15      sub1 = new sca_lsf::sca_sub("sub1");
16      sub1->x1(in_lsf);
17      sub1->x2(sig);
18      sub1->y(out_lsf);
19      dot1 = new sca_lsf::sca_dot("dot1",1.0/(2.0*M_PI*fc));
          // M_PI=3.1415…
20        dot1->x(out_lsf);
21        dot1->y(sig);
22      lsf2tdf1 = new sca_lsf::sca_sink("lsf2tdf1");
          // LSF->TDF converter
23        lsf2tdf1->x(out_lsf);
24        lsf2tdf1->y(out);
25    }
26  };
```

EXAMPLE 1 LSF model of a low pass filter structure with TDF converter modules

Use cases and requirements

The SystemC AMS extensions are designed to enhance the HW/SW-oriented SystemC class library by providing a framework for the functional modeling, architectural exploration, integration validation, and virtual prototyping of E-AMS systems. Such use cases require a means for modeling and simulation that is more abstract than is available from existing HDLs. At the same time, designers should be able to model AMS components and their interactions with HW/SW systems. The extensions provide a standard-

SystemC AMS extensions introduce new language constructs for the design of embedded analog/mixed-signal systems. This paper presents the novel modeling language for analog and mixed-signal functions that supports design and modeling of telecommunications, automotive and imaging sensor applications at various levels of abstraction. A simple example illustrates how new features facilitate a design refinement methodology for functional modeling, architecture exploration and virtual prototyping of embedded analog and mixed-signal systems.

Source: OSCI

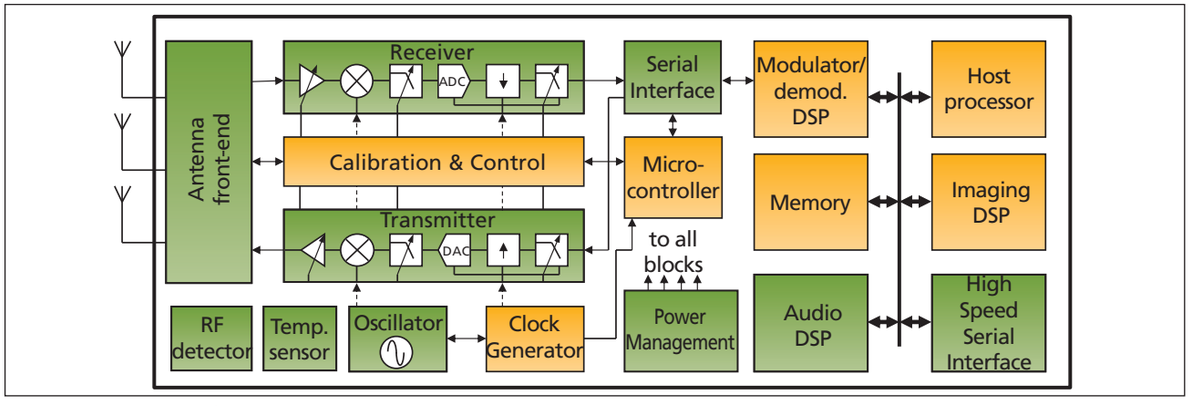


FIGURE 1 Example of an embedded analog/mixed-signal architecture: communication system

ized formalism for the modeling of AMS architectures that bridges the gap between functional modeling frameworks and the HDLs used for implementation.

To that end, we consider the following modeling formalisms:

Signal flow models describe AMS systems (such as control systems or filter structures) at the functional and architecture levels using block diagrams, assuming continuous time and directed real-valued signals. For simulation, differential and algebraic equations are solved numerically at appropriate time steps. Therefore, simulation of signal flow models is often a time-consuming task. Data flow models are common for modeling DSP algorithms and communication systems at the functional and architecture levels by processes that communicate via buffers. For simulation, the processes are activated in the data flow's direction, considering different data rates. Note that, in general, data flow models are untimed and can have different semantics. When modeling AMS systems, timed semantics are introduced by assuming discrete time steps between data tokens. Although not as accurate as a continuous-time signal flow, a discrete-time data flow provides a higher simulation performance with reasonable accuracy.

Electrical networks are essential for modeling E-AMS systems. They are required to model loads, protection circuits,

Source: OSCI

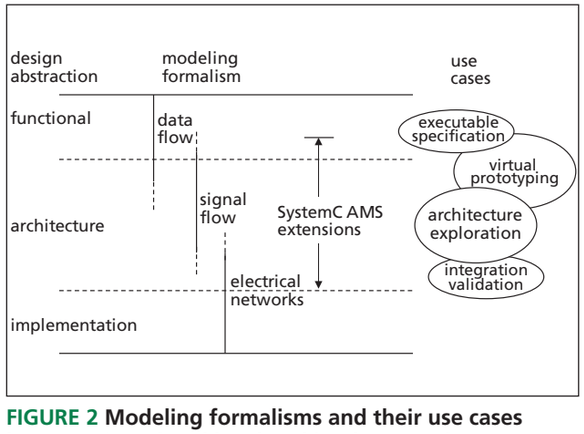


FIGURE 2 Modeling formalisms and their use cases between functional level and implementation

and buses at high frequencies using macro models for describing continuous-time relations between voltages and currents. For simulation, differential and algebraic equations are solved, but efficiency can be maintained by using only linear primitives and switches. *Figure 2* gives an overview of modeling formalisms that are required for design-

Continued on next page

Source: OSCI

```
01  SCA_TDF_MODULE(mixer)
      // TDF primitive module definition
02  {
03    sca_tdf::sca_in<double> rf_in, lo_in;
      // TDF in ports
04    sca_tdf::sca_out<double> if_out;
      // TDF out ports
05    void set_attributes()
06    {
07      set_timestep(1.0, SC_US);
          // time between activations
08    }
09    void processing()
       // executed at each activation
10    {
11      if_out.write( rf_in.read() * lo_in.read() );
12    }
13    SCA_CTOR(mixer) {}
14  };
```

EXAMPLE 2 TDF model of a mixer

ing AMS systems and their use cases between the functional level and implementation.

A major requirement is to maintain an acceptable simulation performance while modeling the architecture's behavior with sufficient accuracy. Therefore, the AMS extensions must enable the use of dedicated simulation kernels synchronized with the standard SystemC kernel.

Electrical networks require specific simulation capabilities. The simulation of such models needs structural analysis, the setup of differential and algebraic equations (DAE), and numerical methods for solving them.

Furthermore, the AMS language must enable the use of dedicated simulation kernels for special cases such as linear networks, which permit a significantly higher simulation performance compared with nonlinear networks. For models, the special case of synchronous data flow allows the implementation of a dedicated simulation kernel that

Source: OSCI

```
01  SCA_TDF_MODULE(lp_filter_tdf)
02  {
03    sca_tdf::sca_in<double> in;
04    sca_tdf::sca_out<double> out;
05    sca_tdf::sc_in<double> gain;
          // converter port for SystemC input
06    sca_tdf::sca_ltf_nd ltf;
          // computes transfer function
07    sca_util::sca_vector<double> num, den;
          // coefficients
08    void initialize()
09    {
10      num(0) = 1.0;
11      den(0) = 1.0;
12      den(1) = 1.0/(2.0*M_PI*1.0e4); // M_PI=3.1415…
13    }
14    void processing()
15    {
16      out.write( ltf(num, den, in.read() ) * gain.read() );
17    }
18    SCA_CTOR(lp_filter_tdf) {}
19  };
```

EXAMPLE 3 Continuous-time Laplace transfer function in a TDF module

Source: OSCI

```
01  SC_MODULE(frontend)
     // SC_MODULES used for hierarchical structure
02  {
03    sca_tdf::sca_in<double>     rf, loc_osc;
        // use TDF ports to connect with
04    sca_tdf::sca_out<double>    if_out;
        // TDF ports/signals in hierarchy
05    sc_core::sc_in<sc_dt::sc_bv<3> > ctrl_config;
        // SystemC input agc_ctrl configur.
06    sca_tdf::sca_signal<double> if_sig;
        // TDF internal signal
07    sc_core::sc_signal<double> ctrl_gain;
        // SystemC internal signal
08    mixer*         mixer1;
09    lp_filter_tdf* lpf1;
10    agc_ctrl*      ctrl1;
11    SC_CTOR(frontend)
12    {
13      mixer1 = new mixer("mixer1");
14        mixer1->rf_in(rf);
15        mixer1->lo_in(loc_osc);
16        mixer1->if_out(if_sig);
17      lpf1 = new lp_filter_tdf("lpf1");
18        lpf1->in(if_sig);
19        lpf1->out(if_out);
20      ctrl1 = new agc_ctrl("ctrl1");
          // SystemC module
21        ctrl1->out(ctrl_gain);
22        ctrl1->config(ctrl_config);
23    }
24  };
```

EXAMPLE 4 Structural description, including TDF and DE modules

provides considerable speed-up compared with the SystemC kernel.

Another important requirement is extensibility. Industrial design flows for E-AMS systems use SPICE-like circuit simulators with high accuracy, special support for RF, nonlinear systems or other application-specific extensions. Therefore, the SystemC AMS extensions should allow the industry or EDA vendors to integrate 'user-defined extensions' to support different domains.

Language structure and use cases

The SystemC AMS extensions meet the requirements and use cases discussed in the previous section. They provide support for signal flow, data flow, and electrical networks. The extensions are fully compatible with the SystemC standard as shown in *Figure 3*.

Electrical networks and signal flow models use a linear DAE solver that solves the equation system and is synchronized with the SystemC kernel. The use of this solver restricts networks and signal flow components to linear models to provide high simulation performance.

Data flow simulation is accelerated using static scheduling that is computed before simulation starts. This schedule is activated in discrete time steps, where synchronization with the SystemC kernel introduces timed semantics. We therefore call it a 'timed data flow' (TDF).

The SystemC AMS extensions define new language constructs identified by the prefix 'sca_'. They are de-

Source: OSCI

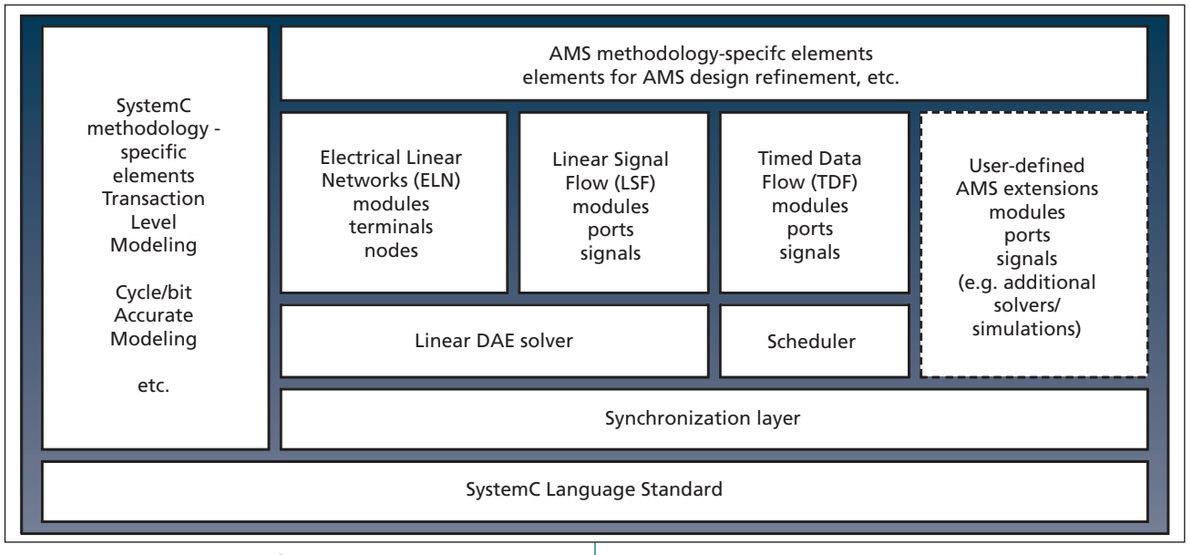


FIGURE 3 AMS extensions for SystemC

Source: OSCI

```
01  SCA_TDF_MODULE(ad_converter)
      // Very simple AD converter
02  {
03    sca_tdf::sca_in<double>  in_tdf;
        // TDF port
04    sca_tdf::sc_out<sc_dt::sc_int<12> > out_de;
        // converter port to DE domain
05    void processing()
06    {
07      out_de.write( static_cast<sc_dt::sc_int<12> >(in_tdf.read() ) );
08    }
09    SCA_CTOR(ad_converter) { }
10  };
```

EXAMPLE 5 TDF model of a simple A/D converter

clared in dedicated namespaces 'sca_tdf' (timed data flow), 'sca_eln' (electrical linear networks), and 'sca_lsf' (linear signal flow) according to the underlying semantics. By using namespaces, similar primitives as in SystemC are defined to denote ports, interfaces, signals, and modules. For example, a TDF input port is an object of class 'sca_tdf::sca_in<type>'. Linear signal flow (LSF), models are specified by instantiating a structure of signal flow primitives such as adders, integrators, differentiators, or transfer functions. These primitives are part of the language definition. The primitives are connected via signals of the class 'sca_lsf::sca_signal'. To access LSF signals from TDF and DE, converter modules must be instantiated. The instantiation of the primitives can be done in a regular 'SC_MODULE' using standard SystemC rules. *Example 1* (p. 21) gives a simple low pass filter structure with TDF interface, enabling its use in a TDF model.

Such models consist of TDF modules connected via TDF signals using TDF ports. The connected modules form a contiguous structure called a TDF cluster. Clusters must not have cycles without delays, and each TDF signal must have one source. A cluster is activated in discrete time steps. The behavior of a TDF module is specified by overloading the

Source: OSCI

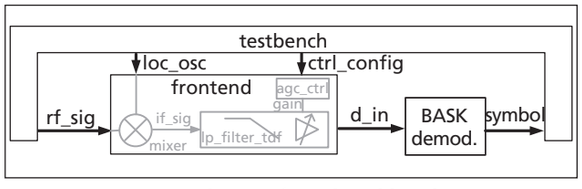


FIGURE 4 BASK receiver at functional level

predefined methods – 'set_attributes()', 'initialize()', and 'processing()'.

The method 'set_attributes()' is used to specify attributes such as rates, delays or time steps of TDF ports and modules.

The method 'initialize()' is used to specify initial conditions. It is executed once when the simulation starts.

The method 'processing()' describes time-domain behavior of the module. It is executed at each activation of the TDF module.

At least one definition of the time step value is expected and, in the case of cycles, one definition of a delay value per

Continued on next page

Source: OSCI

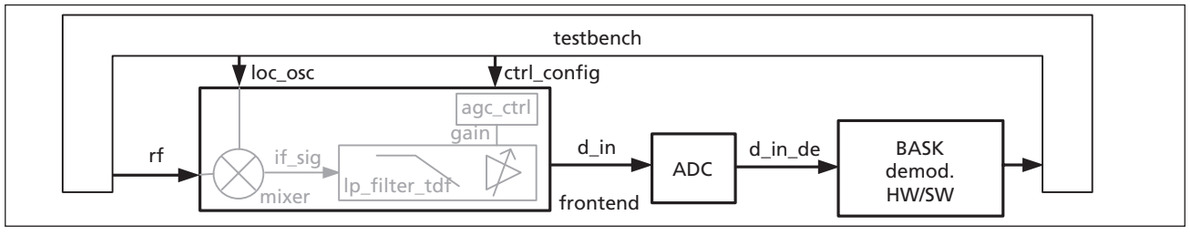


FIGURE 5 Architecture level model of BASK receiver with ADC and a BASK HW/SW system

Source: OSCI

```
01  SC_MODULE(lp_filter_eln)
02  {
03    sca_tdf::sca_in<double>  in;
04    sca_tdf::sca_out<double> out;
05    sca_eln::sca_node in_node, out_node;
        // node declarations
06    sca_eln::sca_node_ref gnd;
        // reference node
07    sca_eln::sca_r  *r1;
        // resistor
08    sca_eln::sca_c  *c1;
        // capacitor
09    sca_eln::sca_tdf_vsource *v_in;
        // converter TDF -> voltage
10    sca_eln::sca_tdf_vsink   *v_out;
        // converter voltage -> TDF
11    SC_CTOR(lp_filter_eln)
12    {
13      v_in = new sca_eln::sca_tdf_vsource("v_in", 1.0);
          // scale factor 1.0
14        v_in->ctrl(in);
              // TDF input
15        v_in->p(in_node);
              // is converted to voltage
16        v_in->n(gnd);
17      r1 = new sca_eln::sca_r("r1", 10e3);
          // 10kOhm resistor
18        r1->p(in_node);
19        r1->n(out_node);
20      c1 = new sca_eln::sca_c("c1", 100e-6);
              // 100uF capacitor
21        c1->p(out_node);
22        c1->n(gnd);
23      v_out = new sca_eln::sca_tdf_vsink("v_out", 1.0);
          // scale factor 1.0
24        v_out->p(out_node);
              // filter output as voltage
25        v_out->n(gnd);
26        v_out->tdf_voltage(out);
              // here converted to TDF signal
27    }
28  };
```

EXAMPLE 6 Continuous-time transfer function in a TDF module

cycle. TDF ports are single-rate by default. It is the task of the elaboration phase to compute and propagate consistent values for the time steps to all TDF ports and modules. Before simulation, a schedule is determined that defines the order of activation of the TDF modules, taking into account the rates, delays, and time steps. During simulation, the 'processing()' methods are executed at discrete time steps. *Example 2* shows the TDF model of a mixer. The 'processing()' method will be executed with a time step of 1μs.

In addition to the pure algorithmic or procedural description of 'processing()', different kinds of transfer function can be embedded in TDF modules. *Example 3* gives the TDF model of a gain-controlled low pass filter by instantiating a class that computes a continuous-time Laplace transfer function (LTF). The coefficients are stored in a vector of the class 'sca_util::sca_vector' and are set in the 'initialize()' method. The transfer function is computed in the 'processing()' method by the 'ltf' object at discrete time points using fixed-size time steps.

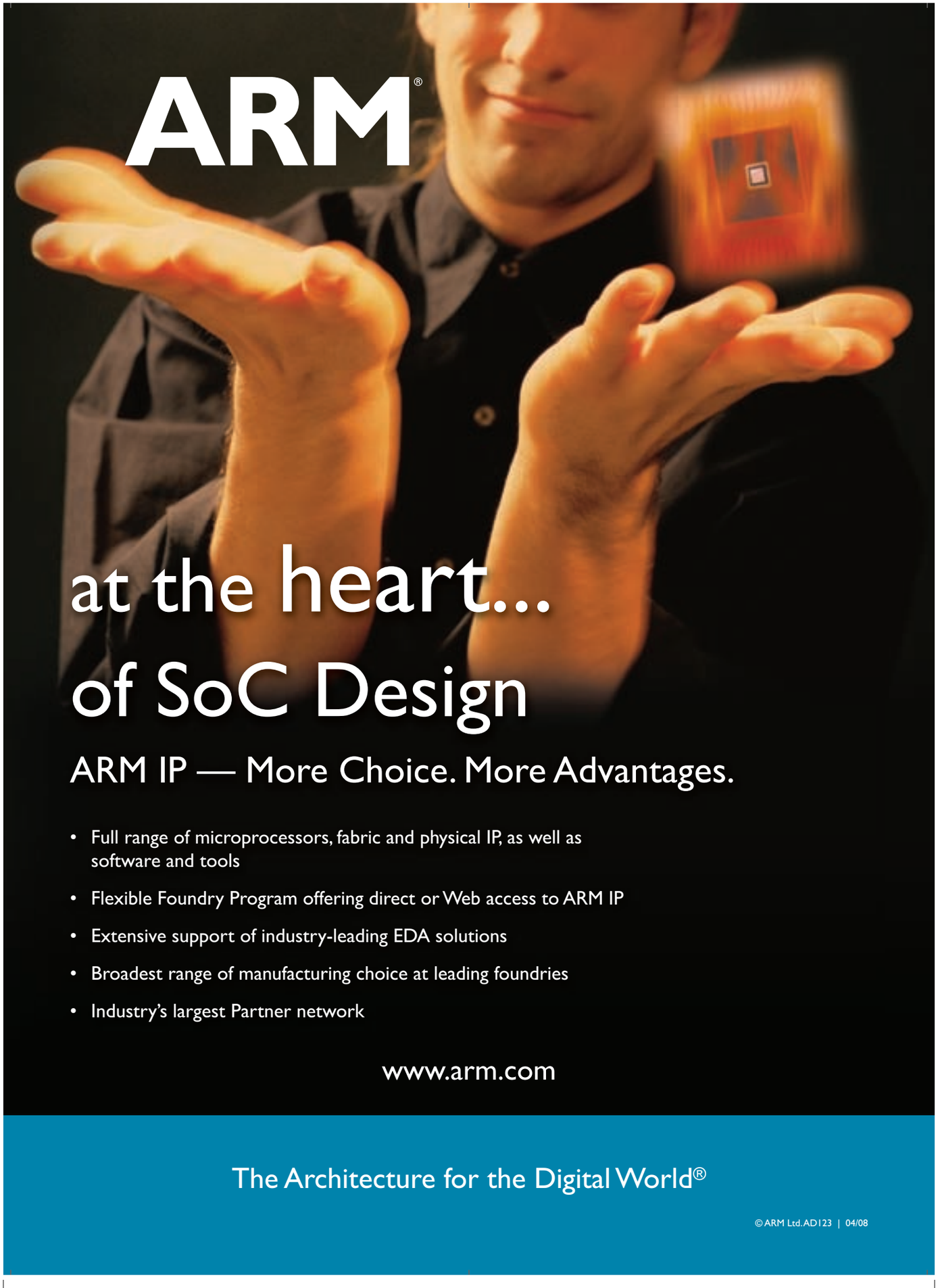
Source: OSCI

```
01  SCA_TDF_MODULE(bask_demodulator)
02  {
03    sca_tdf::sca_in<double> in;
        // values …
04    sca_tdf::sca_out<bool>  out;
        // symbol
05    void set_attributes()
06    {
07      in.set_rate(20000);
          // port in has 20000 samples/timestep
08      out.set_timestep(0.1, SC_MS);
09    }
10    void processing()
        // maps 20000 samples to 1 symbol
11    {
12      double val = 0.0;
13      for (unsigned long i = 0; i < in.get_rate(); i++)
14        val += abs( in.read(i) );
15      if ( val>THRESHOLD ) out.write( true );
        // THRESHOLD defined in header
16      else out.write( false );
17    }
18    SCA_CTOR(bask_demodulator) {}
19  };
```

EXAMPLE 7 Executable specification of BASK demodulator

The TDF modules in *Examples 2* and *3* can be instantiated and connected to form a hierarchical structure with other SystemC modules. The modules are connected as in SystemC with TDF signals ('sca_tdf::sca_signal<type>') and SystemC signals (*Example 4, p. 23*).

Predefined converter ports ('sca_tdf::sc_out' or 'sca_tdf::sc_in') can establish a connection to a SystemC DE channel (e.g., 'sc_signal<T>'), reading or writing values during the first delta cycle of the current SystemC time step. *Ex-*



Source: OSCI

```
01 #include "systemc-ams"
  // top level
02 #include "bask-demodulator.h"
  // other definitions
03 int sc_main(int argc, char* argv[]) {
04   sca_tdf::sca_signal<double> rf, loc_osc, d_in;
05   sca_tdf::sca_signal<bool>   symbol;
06   sc_core::sc_signal<sc_dt::sc_bv<3> > ctrl_config;
07   frontend fe1("fe1");
08     fe1.loc_osc(loc_osc); fe1.ctrl_config(ctrl_config);
09     fe1.rf(rf); fe1.if_out(d_in);
10   bask_demodulator bask1("bask1");
11     bask1.in(d_in);
12     bask1.out(symbol);
13   testbench tb1("tb1");
14     tb1.rf(rf); tb1.loc_osc(loc_osc); tb1.ctrl(ctrl_config);
15     tb1.symbol(symbol);
  // tracing, ...
16   sc_core::sc_start(51.2, SC_MS);
17 };
```

EXAMPLE 8 Executable specification of BASK receiver (top level)

ample 5 illustrates the use of such a converter port in a TDF module modeling a simple A/D converter with an output port to which a SystemC DE channel can be bound.

Electrical linear networks (ELN) are specified by instantiating predefined network primitives, such as resistors or capacitors. As with LSF, the primitives can be instantiated in a regular 'SC_MODULE'. To access the voltages or currents in the network, converters must be used. The SystemC AMS extensions provide converters that translate these voltages and currents to double-precision floating-point values in TDF and discrete-event models – and vice versa. *Example 6* gives the ELN model of a low-pass filter implemented with one resistor primitive and one capacitor primitive. Since the intention is to use the model in a TDF context, additional converter primitives are used to convert TDF data sample values to voltages – and vice-versa.

As well as modeling LSF, TDF, and ELNs, the SystemC AMS extensions provide a means of tracing signals.

Modeling methodology example

To demonstrate the capabilities of the AMS extensions, a simplified example is presented. The use of the AMS extensions is explained for the use cases shown in *Figure 2 (p. 22)*:

- executable specification at the functional level;
- architecture exploration at the functional, architecture, and implementation levels;
- integration validation; and
- virtual prototyping.

An executable specification is made to verify the correct understanding of the system specification by simulation. For this use case, data flow and signal flow are appropriate modeling styles. Note, that the executable specification introduces structures and algorithms that are hard to change or modify later. Therefore, structures and algorithms that match the architecture must be carefully chosen. The specification of the simple example is shown in *Figure 4*. It is a basic binary amplitude shift keying (BASK) receiver consisting of a mixer, a low pass filter, and a BASK demodulator.

To verify the receiver specification, it is modeled using TDF. To achieve good simulation performance we use TDF for the whole receiver model. We obtain data samples by oversampling the assumed continuous-time signals and represent them by double-precision values in the front end. We also assume that the BASK demodulator generates one symbol for every 20,000 input samples. *Example 7* gives the TDF model of the BASK demodulator, assuming a DSP-based implementation.

Using the front-end module in *Example 4*, the TDF module of the BASK demodulator given in *Example 7* and another module for the testbench, the structure of a model that can be used to evaluate the functional specification in *Figure 3*, is given in *Example 8*.

The architecture exploration stage evaluates and determines the key properties and comprises two phases. In the first, the executable specification is refined by adding non-ideal properties of an implementation to better understand the impact on overall system behavior. In the second, the architecture's structure and interfaces are adapted to match the final implementation to get a more accurate model. Note, that this also implies the use of different models of computation. For example, HW/SW architectures can be analyzed more accurately using cycle accurate or TLM modeling.

In this example, the impact of mixer noise and distortion is evaluated in the first phase. The 'processing()' method of the mixer in the executable specification is refined by adding a simple behavioral model of noise ('function my_noise()')

Source: OSCI

```
01  void processing()  // Mixer refined with distortions and noise
02  {
03    double rf = rf_in.read();
04    double lo = lo_in.read();
05    double rf_dist = ( alpha - gamma * rf * rf ) * rf; //alpha and gamma
06    double mix_dist = rf_dist * lo;                    //user-defined
07    if_out.write( mix_dist + my_noise() );
08  }
```

EXAMPLE 9 Refinement of the ideal mixer (Example 2) by adding non-linear distortion and noise

and by C++-code that models third-order non-linear distortion as illustrated in Example 9.

Other parameters such as bit widths, delays, time steps and rates can also be easily analyzed by refinement of the executable specification. For the evaluation of different sample rates, delays and time steps, the properties speci-

Continued on next page

TAPE-OUT COMES A LOT FASTER
WITH CALIBRE nmDRC.

INTEGRATED SYSTEM DESIGN + DESIGN FOR MANUFACTURING + ELECTRONIC SYSTEM LEVEL DESIGN + FUNCTIONAL VERIFICATION

Calibre® nmDRC | There's nothing like having an advantage when you're racing to market. That's exactly what you get with Calibre nmDRC. Widely recognized as the world's most popular physical verification solution, Calibre's hyperscaling architecture produces the fastest run times available. To accelerate things even more, Calibre nmDRC adds incremental verification and a dynamic results-viewing and debugging environment. Designers can check, fix and re-verify DRC violations in parallel, dramatically reducing total cycle time. Start out and stay ahead of the pack. Go to **mentor.com/go/calibre_nmdrc** or call us at **800.547.3000.**

Mentor Graphics®

THE EDA TECHNOLOGY LEADER

Source: OSCI

```
01 void processing()
  // Demodulator refined with quantization
02 {
03   sc_dt::sc_int<16> val;
  // evaluate 16 bit accuracy
04   for (unsigned long i = 0; i < in.get_rate(); i++)
05   val += abs(static_cast<sc_dt::sc_int<16> > ( in.read(i) ) );
  // THRESHOLD defined in header
06   if ( val > THRESHOLD ) out.write(true);
07   else out.write(false);
08 }
```

EXAMPLE 10 Refinement of simple BASK demodulator to evaluate quantization

fied in the attributes section of TDF modules can be modified. The impact of bit widths and quantization errors can be analyzed by replacing the floating-point data types in the 'processing()' methods with 'sc_int<bw>' finite-length integer types. In *Example 10*, different bit widths are evaluated by replacing the double-precision data type with a 16bit integer representation.

In the second phase of architectural exploration, the intended architecture is modeled more accurately (e.g., the low pass filter could be specified by an electrical network). Given knowledge of the required sample rates and bit widths, an A/D converter between the low pass filter and the BASK demodulator is added. This determines the A/D partitioning as shown in *Figure 5* and can be seen as a starting point for HW/SW co-design using SystemC. Instead of a full TDF model, a model that uses an A/D converter to translate the TDF signal 'd_in' to a cycle accurate or TLM signal 'd_in_de' can be developed. The interface to and modeling of the demodulator HW/SW system using pure SystemC is not covered here.

To prepare for the design of subsystems and integration validation, the interfaces of all subsystems must be modeled accurately. The interfaces and data types used in the models should match the implementation. For analog circuits, this relates to electrical nodes. For digital circuits, this relates to pin accurate buses. For HW/SW systems TLM interfaces might be appropriate. After the definition and design of the analog, digital HW and SW components, these components are integrated and their correctness is verified within the overall system. The layered architecture of the SystemC AMS extensions encourages its use as a simulation backplane in which other simulators can be integrated via their C-language interfaces. In the BASK receiver example, the low pass filter and the mixer could be replaced by circuit-level models to validate the connectivity and integrity of the subsystems.

The AMS extensions are not intended to replace circuit simulators. The use of more accurate circuit simulators to verify circuit implementations can be applied, once an integration of a circuit simulator is available.

Virtual prototyping provides software developers with a high-level untimed or timed model that represents the hardware architecture.

Especially for E-AMS systems, where software is interacting with AMS hardware, interoperability with SystemC TLM extensions is important. In the BASK receiver example, the refined TDF model combines high simulation speed with appropriate accuracy to act as a virtual prototype for further development of the HW/SW subsystem. Then, the AMS subsystem becomes part of the virtual prototype. For systems that are more complex and realistic than the simple BASK example, SystemC AMS extensions support a similar refinement methodology to SystemC that can be applied to the overall system by introducing hierarchical converter channels or adapters.

Summary

The SystemC AMS extensions enhance the available SystemC standard with support for linear electrical networks, linear signal flow, and timed data flow models to efficiently model and simulate AMS architectures. New language constructs support the creation of AMS models at higher levels of abstraction, introducing functional and architecture-level modeling. This enables a design refinement methodology for executable specification, architecture exploration, integration validation, and virtual prototyping of E-AMS systems in a seamless, C-based environment. The extensibility supports application specific requirements (e.g., to model nonlinear or radio frequency behavior). Extending the unique capabilities of SystemC with new AMS features offers a powerful modeling and simulation framework, enabling design and verification for a wide range of applications and systems.

Authors

This article was written by Christoph Grimm, Vienna University of Technology, Martin Barnasconi, NXP Semiconductors, Alain Vachoux, Ecole Polytechnique Fédérale de Lausanne, & Karsten Einwich, Fraunhofer IIS/EAS Dresden.