



Advancing the SystemC Analog/Mixed-Signal (AMS) Extensions

Introducing Dynamic Timed Data Flow

Authors

Martin Barnasconi, NXP Semiconductors

Karsten Einwich, Fraunhofer IIS/EAS Dresden

Christoph Grimm, Vienna University of Technology

Torsten Maehne, Université Pierre et Marie Curie

Alain Vachoux, École Polytechnique Fédérale de Lausanne

September 29, 2011

Copyright © 2011 Open SystemC Initiative (OSCI)

Abstract

SystemC™ [1] is an industry-standard language for electronic system-level (ESL) design and modeling, enabling the creation of abstract architectural descriptions of digital hardware/software (HW/SW) systems, also known as *virtual prototypes*. The introduction of the SystemC Analog/Mixed-Signal (AMS) Extensions [2] now facilitates modeling of heterogeneous embedded systems, which include abstract AMS behavior. This paper presents the continuing advancements of the SystemC AMS extensions to further expand the capabilities of SystemC AMS for the creation of these *mixed-signal virtual prototypes*.

To comply with demanding requirements and use cases (e.g., in automotive applications), new execution semantics and language constructs are being defined to facilitate a more reactive and dynamic behavior of the Timed Data Flow (TDF) model of computation as defined in the current SystemC AMS 1.0 standard. The proposed *Dynamic TDF* introduces fully complementary elements to enable a tighter time-accurate interaction between the AMS signal processing and control domain while keeping the modeling and communication abstract and efficient. The features of Dynamic TDF are presented in this paper by means of a typical example application from the automotive domain.

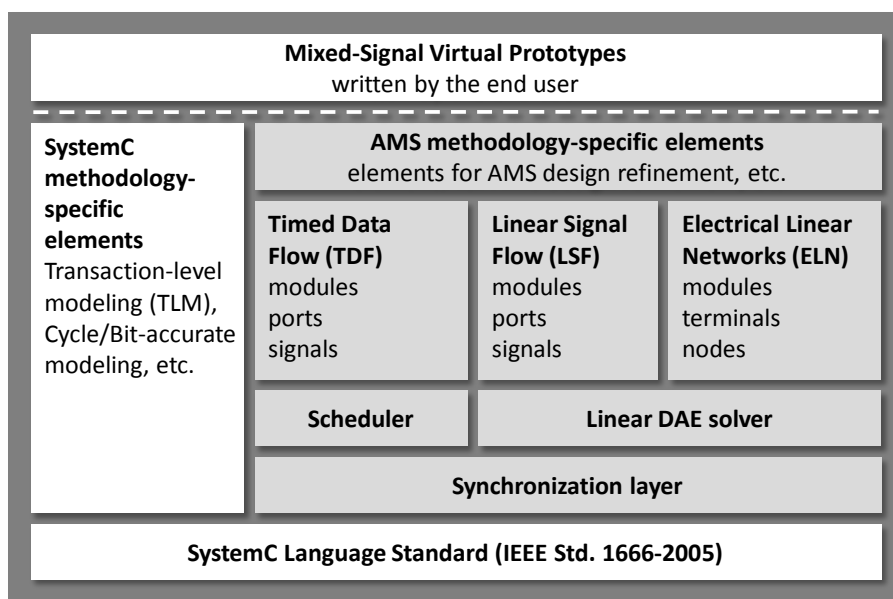


Figure 1: Architecture of SystemC, TLM, and AMS extensions for building a mixed-signal virtual prototype

Introduction

The SystemC AMS 1.0 language standard enables the description of analog/mixed-signal behavior as a natural extension to existing SystemC-based design methodologies (Figure 1). SystemC together with its AMS extensions allow the creation of an executable description of a mixed discrete- and continuous-time system. Digitally-oriented HW/SW architecture descriptions made in SystemC — often using transaction-level modeling [3] — can be augmented with abstract AMS behavior by using the SystemC AMS extensions. This approach supports use cases such as software development, architecture exploration, and system validation.

There is a strong need to include the interaction between the digital HW/SW and AMS or radio frequency (RF) parts in the system, especially to validate the entire system architecture in terms of functional correctness, partitioning, and dimensioning. The functionality of all of these parts is nowadays tightly interwoven with calibration and control loops that cross the analog-digital domain boundary. Traditionally, the signal processing and control-oriented paths were analyzed and designed independently from each other. However, the need for optimal system performance across multiple domains requires a more integral design and verification approach, which supports (AMS) signal processing and control functionality in a single modeling platform.

The Timed Data Flow (TDF) model of computation defined in the SystemC AMS 1.0 standard has already shown its value for signal-processing-oriented applications, such as RF communication and digital signal processing (DSP) systems, where the complex envelope of the modulated RF signal can be described as an equivalent baseband signal and where baseband algorithms are described naturally using data flow semantics. Because TDF is derived from the well-known Synchronous Data Flow (SDF) model of computation, high simulation performance can be obtained due to the calculation of a static schedule prior to simulation.

However, the use of the SystemC AMS extensions in other application domains has been limited due to the restrictions caused by the fixed time step mechanism in TDF. Modeling control systems (e.g., in automotive applications) becomes vastly inefficient by this limitation (Figure 2). The fixed and constant time step in TDF does not allow end users to easily model systems in which activation periods or frequencies are changed dynamically. A dynamic change of frequencies is required to efficiently model voltage controlled oscillators (VCO), clock recovery circuits, and phenomena such as jitter. A dynamic assignment of activations is needed to model power down of hardware (e.g., in wireless sensor networks) or to model pulse width modulation (PWM) in a functional way. These types of applications underline the need to introduce *Dynamic TDF* capabilities, which are the subject of this paper.

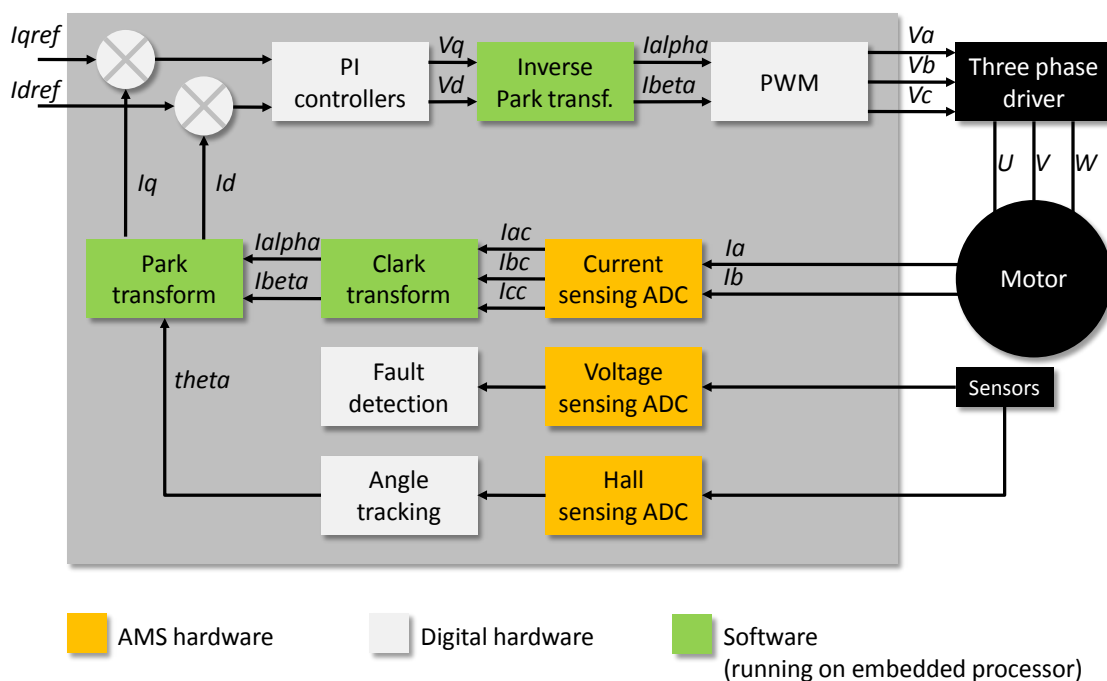


Figure 2: Embedded mixed-signal system for automotive: Motor power steering control module

Use cases and requirements for *Dynamic Timed Data Flow*

The objective of Dynamic TDF is to offer mechanisms to dynamically change key TDF properties such as time step, rate, or delay. This efficiently overcomes the previously mentioned limitations of the TDF model of computation. These new features should also support the successive transition from abstract, functional-level modeling to refined mixed-signal architecture modeling with ideal and ‘non-digital’ properties. In that respect, we consider the following relevant use cases:

- **Abstract modeling of sporadically changing signals** that are constant over long time periods and that cannot be modeled by sampled, discrete-time signals in an efficient way. A particular application is power management that switches on/off subsystems. In order to model power down, it is required to be able to specify a condition that enables/disables the execution of the AMS computations.

- **Abstract description of reactive behavior**, for which computations are executed for events or transactions, such as an analog threshold crossing or for which an immediate request or response is required (e.g., as part of a TLM model). Typical applications are sensor systems, in which crossing a threshold will cause an action. An event-triggered modeling approach for these systems would be more natural and also more efficient. A reactive synchronization mechanism would be beneficial, especially in cases where these AMS systems are integrated together with TLM-based virtual prototypes. This avoids the penalty of introducing fine-grained computations by using small time steps to detect the actual event.
- **Capture behavior where frequencies (and time steps) change dynamically**. This is the case for applications such as VCO, PLL, PWM, or clock recovery circuits, which are often controlled by analog or digital signals. Modeling oscillators with variable frequencies (e.g., clock recovery) or capturing jitter is not possible when using constant time steps. In order to allow modeling of such systems, it is required to be able to change the time step continuously during simulation.
- **Modeling systems with varying (data) rates**, which are changed during operation and thus during simulation. This is the case, for example, when communication systems are described at high levels of abstraction. To perform cross-layer optimization and to evaluate the correctness of a particular signal-processing algorithm, both the physical layer (PHY) as well as media access control (MAC) as part of the data link layer (DLL) need to be modeled. An example of such systems are cognitive radios, in which parameters such as data rates and modulation schemes are adapted to estimated parameters of the channel.

Table 1 gives the summary of the presented use cases, requirements, and application examples.

Use cases	Requirements	Application examples
Abstraction of sporadically changing signals	Switch on/off AMS computations	Power management unit
Abstract description of reactive behavior	Detect analog zero- or threshold-crossing	Sensor circuits; alarm mode of systems
	Request and response caused by digital event or transaction	AMS embedded in digital HW/SW virtual prototype
Capture behavior where frequencies (and time steps) change dynamically	Changeable time step of AMS computations	VCO, PLL, PWM, Clock recovery circuits
Modeling systems with varying (data) rates	Changeable time step and/or data rate	Communication systems, multi-standard radio interfaces (e.g., cognitive radios)

Table 1: Use cases, requirements, and applications for introducing Dynamic TDF capabilities

Application example: Motor control system

In order to illustrate the value of Dynamic TDF, we use a simple example from the automotive domain as shown in Figure 3, a DC motor control system with a proportional-integrator (PI) controller and a pulse width modulator (PWM). The formed control loop reduces the impact of aging, parameter deviations, and environmental conditions on the load, which is in this case a power driver and DC motor described with a single pole transfer equation.

The PI controller computes the required pulse width based on the reference value (i_{ref}) and measured current (i_{meas}) through the load. The PWM generates a pulse (v_{drv}) of variable width (resulting in a varying duty cycle) that increases the current through the load.

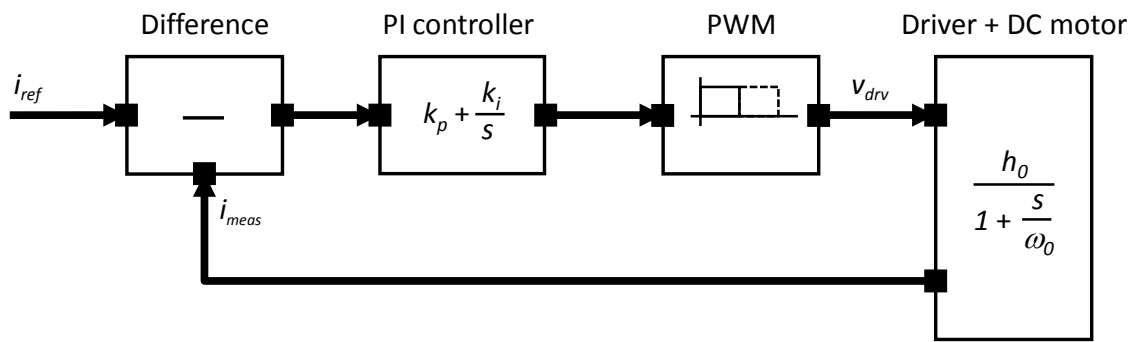


Figure 3: Functional model (in the Laplace domain) of a DC motor control system

When using the conventional TDF model of computation defined in the SystemC AMS 1.0 standard, the AMS computations are executed at fixed discrete time steps while considering the input samples as continuous-time signals. Since the PWM block has an almost discrete-event behavior, the need to have very steep ramps for the signal transitions at its output imposes the use of very small time steps. A fine-grained time grid is essential for a correct overall response of the system, as it needs to meet the accuracy constraint (time constants) of the PI controller and the Driver + DC motor. However, a too-fine-grained time grid will reduce the simulation performance, as the number of time steps and thus AMS computations will increase.

Alternatively, the PWM could be modeled as a pure SystemC discrete-event model. But this makes the simulation less efficient due to the use of the dynamic scheduling of SystemC (evaluate/update cycle) instead of the more efficient TDF static scheduling. Furthermore, it will introduce unnecessary synchronization between TDF and the SystemC discrete-event model of computation.

By introducing Dynamic TDF for this application, the computation of the motor control loop is only triggered four times per pulse period by changing the TDF time step (Figure 4): first at the start of the rising edge, second at the end of the rising edge, third at the end of the voltage pulse plateau, and fourth at the end of the falling edge. Each time, the PWM adjusts the scheduling of the next activation based on the duty cycle sampled at its input during the rising edge. With the pulse output active, energy is supplied to the power driver of the DC motor resulting in an increase of the current. This process repeats itself while the system reaches its steady state. Note that the PWM output signal (v_{drv}) represents a continuous-time waveform and thus has finite-slope edges.

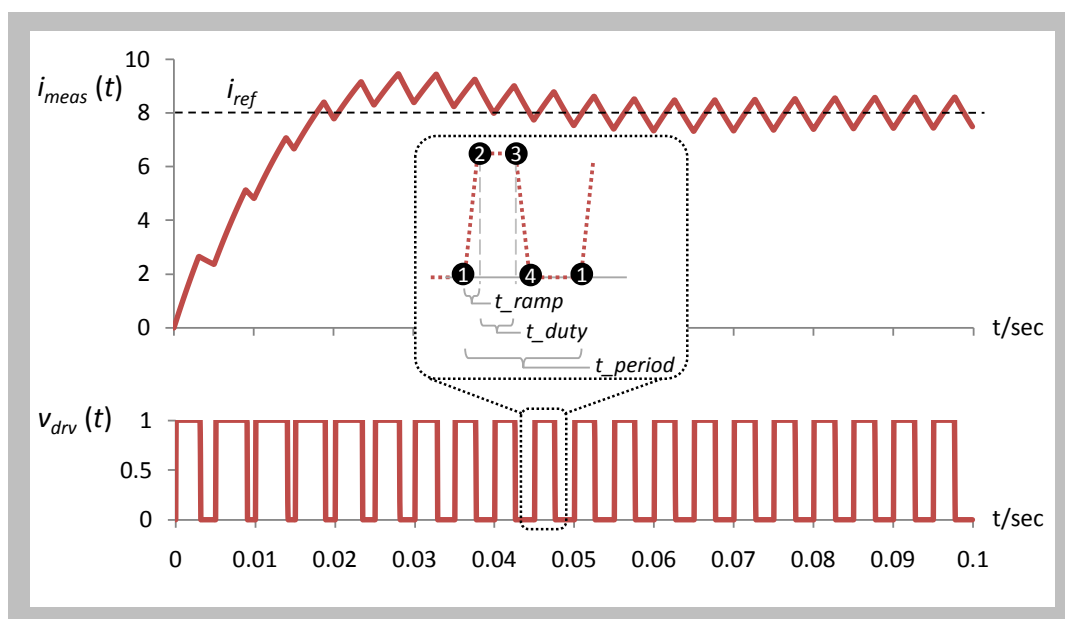


Figure 4: Step response of the motor control loop using Dynamic TDF with four activations per period

To efficiently model the PWM pulse with a varying pulse width, the time step attribute will be changed such that the PWM pulse's rising edge and falling edge are included resulting in only four activations of the PWM TDF module per period, as shown in [Figure 4](#).

Execution semantics and language constructs of Dynamic Timed Data Flow

The Dynamic TDF modeling paradigm aims at efficient modeling of functions requiring dynamic activation, such as PWM, while maintaining the principles of TDF modeling as introduced in the SystemC AMS extensions. To this end, new expressive ways are added to the TDF model of computation to dynamically change key TDF attributes such as time step, rate, or delay. Recall that TDF modules interconnected by TDF signals via TDF ports constitute so-called TDF clusters and that the order in which the modules in clusters have to be activated is statically computed before simulation starts. The new capabilities of dynamically changing the mentioned TDF attributes may then require new schedules to be dynamically recomputed.

Before illustrating these new capabilities, let us start first with a conventional TDF model for the PWM block ([Listing 1](#)). The `set_attributes()` callback is used to define the fixed time step of the module activation by means of a module parameter. The callback could also define the values of rates or delays. The `initialize()` callback (not needed in this case and therefore not shown) may be used to define initial conditions such as delay values or filter coefficients. The `processing()` callback defines the module's PWM behavior.

```
// pwm.h

#include <cmath>
#include <systemc-ams>

SCA_TDF_MODULE(pwm)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    pwm( sc_core::sc_module_name nm, double v0_ = 0.0, double v1_ = 1.0,
        const sca_core::sca_time& t_period_ = sca_core::sca_time(5.0, sc_core::SC_MS),
        const sca_core::sca_time& t_ramp_ = sca_core::sca_time(0.05, sc_core::SC_MS),
        const sca_core::sca_time& t_step_ = sca_core::sca_time(0.01, sc_core::SC_MS) )
: in("in"), out("out"), v0(v0_), v1(v1_), t_period( t_period_.to_seconds() ),
  t_ramp( t_ramp_.to_seconds() ), t_duty_max( t_period - 2.0 * t_ramp ),
  t_duty( t_duty_max ), t_step( t_step_ ) {}

    void set_attributes()
    {
        set_timestep( t_step ); // fixed time step for module activation
    }

    void processing()
    {
        double t = get_time().to_seconds(); // current time
        double t_pos = std::fmod( t, t_period ); // time position inside pulse period

        if ( t_pos < t_ramp ) { // calculate and clamp duty time
            t_duty = in.read() * t_duty_max;
            if ( t_duty < 0.0 ) t_duty = 0.0;
            if ( t_duty > t_duty_max ) t_duty = t_duty_max;
        }
    }
}
```

```

double val = v0; // initial value
if ( t_pos < t_ramp )           // rising edge
    val = ( (v1 - v0) / t_ramp ) * t_pos + v0;
else if ( t_pos < t_ramp + t_duty ) // plateau
    val = v1;
else if ( t_pos < t_ramp + t_duty + t_ramp ) // falling edge
    val = ( (v0 - v1) / t_ramp ) * ( t_pos - t_ramp - t_duty ) + v1;
// else return to initial value

out.write(val);
}

private:
double v0, v1;           // initial and plateau values
double t_period, t_ramp; // pulse period and ramp time
double t_duty_max;      // maximum duty time
double t_duty;          // current duty time
sca_core::sca_time t_step; // module time step
};

```

Listing 1: Conventional TDF module of the PWM function (no Dynamic TDF)

The following new callback and member functions are introduced as part of the TDF model of computation to support Dynamic TDF:

- The **change_attributes()** callback provides a context in which the time step, rate, or delay attributes of a TDF cluster may be changed. The callback is called as part of the recurring execution of the TDF schedule.
- The new **request_next_activation()** member function, as well as existing TDF member functions that set attributes (e.g., **set_timestep()**, **set_rate()**, **set_delay()**, etc.), can be called within the **change_attributes()** callback to redefine TDF properties. The **request_next_activation()** member function will request a next cluster activation at a given time step, event, or event list, which is specified as argument. Note, however, that if multiple TDF modules belonging to the same cluster redefine the next cluster activation by using **request_next_activation()**, the earliest point in time will be used and the other requests will be ignored.
- The **allow_dynamic_tdf()** and **disallow_dynamic_tdf()** member functions are used in the **set_attributes()** callback to check whether the functional behavior captured in the TDF module can cope or not with dynamic time steps. All TDF modules belonging to the same TDF cluster must explicitly state that they allow Dynamic TDF by using the **allow_dynamic_tdf()** member function. Otherwise, the conventional (static) TDF is considered for the cluster. The use of **disallow_dynamic_tdf()** is especially useful when TDF modules are used to model DSP functionality, in which variable time steps are not allowed. If neither of these two functions is called, the module is considered to not support Dynamic TDF.
- The **set_max_timestep()** member function is introduced to define a maximum time step of a TDF module to enforce a module activation if this time period is reached. Bounding the time step is essential to guarantee sufficient time points for the calculation of a continuous-time response, especially when embedding continuous-time descriptions (e.g., by using Laplace transfer functions).

With the help of these new capabilities, the Dynamic TDF module for the PWM block is given in [Listing 2](#). In this example, the module is registered as supporting Dynamic TDF. The time step is then redefined by calling the **request_next_activation()** member function in the **change_attributes()** callback.

Note that the behavior of the PWM module has not been changed; only the module activation is affected. Therefore, we can use the same implementation in the `processing()` callback as shown in [Listing 1](#).

```

// pwm_dynamic.h

#include <cmath>
#include <systemc-ams>

SCA_TDF_MODULE(pwm) // for dynamic TDF, we can use the same helper macro to define the module class
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    pwm( sc_core::sc_module_name nm, ... ) // same module arguments as in Listing 1
    : in("in"), out("out"), ...           // same constructor initialization as in Listing 1
    {}

    void set_attributes()
    {
        allow_dynamic_tdf();                // module supports dynamic TDF
    }

    void change_attributes()
    {
        double t = get_time().to_seconds(); // current time
        double t_pos = std::fmod( t, t_period ); // time position inside pulse period

        // Calculate time step till next activation
        double dt = 0.0;
        if ( t_pos < t_ramp )                // rising edge
            dt = t_ramp - t_pos;
        else if ( t_pos < t_ramp + t_duty ) // plateau
            dt = ( t_ramp + t_duty ) - t_pos;
        else if ( t_pos < t_ramp + t_duty + t_ramp ) // falling edge
            dt = ( t_ramp + t_duty + t_ramp ) - t_pos;
        else                                  // return to initial value
            dt = t_period - t_pos;

        t_step = sca_core::sca_time( dt, sc_core::SC_SEC );
        if ( t_step == sc_core::SC_ZERO_TIME ) // time step should advance
            t_step = sc_core::sc_get_time_resolution();
        request_next_activation( t_step );    // request the next activation
    }

    void processing()
    {
        ... // same PWM behavior as in Listing 1
    }

private:
    ... // same member variables as in Listing 1
};

```

Listing 2: Dynamic TDF module of the PWM function

Table 2 below shows an example with PWM parameters to compare the two variants of the TDF model of computation. The simulation of the conventional PWM TDF model uses a fixed time step that triggers too many unnecessary computations. When using Dynamic TDF, the PWM model is only activated if necessary.

TDF model of computation variant	t_step (ms)	t_ramp (ms)	t_period (ms)	Time accuracy (ms)	#activations per period
Conventional TDF	0.01 (fixed)	0.05	5.0	0.01 (= t_step)	500
Dynamic TDF	variable	0.05	5.0	defined by <code>sc_set_time_resolution()</code>	4

Table 2: Comparison between the conventional and Dynamic TDF model of the PWM

When introducing Dynamic TDF modules, where module activations may occur at user-defined time points or are driven by events, some care should be taken how to combine these new models with existing TDF descriptions, which rely on the fixed discrete-time activations. The SystemC AMS 1.0 standard defines dedicated converter ports or converter modules acting as interfaces between different models of computation. In a similar way, the interface between the conventional (static) TDF computations and Dynamic TDF will be realized by dedicated converter ports to decouple static TDF clusters from dynamically changing clusters. All of these features and their associated language constructs and execution semantics are currently being defined as part of a future update of the Language Reference Manual of the SystemC AMS standard.

Summary and outlook

This paper proposes new features for the SystemC AMS extensions to facilitate a more reactive and dynamic behavior of the Timed Data Flow modeling approach as defined in the SystemC AMS standard. The *Dynamic Timed Data Flow* capabilities have been demonstrated, which offer unique and fully complementary language constructs and execution semantics enriching the existing TDF model of computation. This means that the proven model abstraction strategy using data flow semantics is maintained, while user-defined TDF module activations are introduced to enable a tighter, yet efficient and time-accurate, synchronization for AMS signal processing and control systems. These Dynamic TDF features further expand the capabilities of SystemC AMS to support more demanding ESL design methodologies and modeling requirements for heterogeneous systems such as in automotive applications.

The Open SystemC Initiative AMS Working Group will continue detailing the use cases and requirements and will standardize the language constructs of Dynamic TDF as part of a planned update of the Language Reference Manual of the SystemC AMS standard. Interested in knowing more or getting involved in this standardization effort? Please sign up on the AMS discussion forum [4] or visit www.systemc.org for more details!

References

- [1] IEEE Standard 1666™-2005 - IEEE Standard SystemC Language Reference Manual, <http://standards.ieee.org/findstds/standard/1666-2005.html>
- [2] Open SystemC Initiative, SystemC AMS 1.0 Standard, <http://www.systemc.org/downloads/standards/ams10/>
- [3] Open SystemC Initiative, Transaction-Level Modeling 2.0 Standard, TLM-2.0, <http://www.systemc.org/downloads/standards/tlm20/>
- [4] SystemC AMS forum, http://www.systemc.org/Discussion_Forum/ams_forum/

Open SystemC Initiative (OSCI)

The Open SystemC Initiative grants permission to copy and distribute this document in its entirety. Each copy shall include all copyrights, trademarks, and service marks, if any. All products or service names mentioned herein are trademarks of their respective holders and should be treated as such. All rights reserved.

