Time Step Control and Threshold Crossing Detection in SystemC AMS 2.0

Liliana Lilibeth Andrade Porras, Torsten Maehne, Marie-Minerve Louërat, François Pêcheux

Time Step Control and Threshold Crossing Detection in SystemC AMS 2.0

Liliana Andrade, Torsten Maehne, Marie-Minerve Louërat et François Pêcheux
Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie (UPMC), Paris, France
WWW : http://www-soc.lip6.fr/, E-mail : liliana.andrade@lip6.fr

***Abstract*—The SystemC AMS 2.0 standard [1] published in March 2013 by the AMS Working Group (AMSWG), considerably extends the TDF execution semantics and proposes new language constructs to support dynamic and reactive behaviors. This paper presents two examples detailing the usefulness of Dynamic TDF and showing the dedicated constructs that allow time step variations during simulation. The goal of this paper is to demonstrate how these new capabilities improve simulation accuracy while maintaining a high execution performance by avoiding unnecessary computations.**

***Index Terms*—SystemC AMS 2.0, Dynamic Timed Data Flow, variable time step control, threshold detection, AMS system modeling.**

I. Introduction

The AMS extensions for SystemC [1] provide a uniform and standardized methodology for modeling and simulation of heterogeneous and embedded Analog/Mixed-Signal (AMS) systems at higher levels of abstraction. They were created in response to needs from telecommunication, automotive, and semiconductor industries [2]. The extensions are built on top of the SystemC Language Standard [3] and support the different Models of Computation (MoC) shown in Figure 1. The Timed Data Flow (TDF) MoC allows the discrete-time modeling and efficient simulation of signal processing algorithms and communication systems at functional and architectural level, the Linear Signal Flow (LSF) MoC supports the modeling of continuous-time behaviors and the Electrical Linear Network (ELN) MoC enables the modeling of electrical networks.

In the SystemC AMS 1.0 standard [4], the TDF MoC, which is based on the Synchronous Data Flow (SDF) formalism, is described as a discrete-time modeling style that considers data as a signal, which values are sampled with a constant time step [2]. A TDF model contains a set of modules, which read and write a fixed number of samples (corresponding to rates) from each of its input ports and to each of its output ports, respectively. Fixed time steps and rates during simulation cause difficulties when modeling complex multi-disciplinary systems because it is seldom necessary to change parameters such as the sampling time step and the rate of samples consumed or produced by a particular module during simulation. Another difficulty is that the time management in TDF models is entirely

This work is supported by the European project CATRENE CA701 – H-INCEPTION.

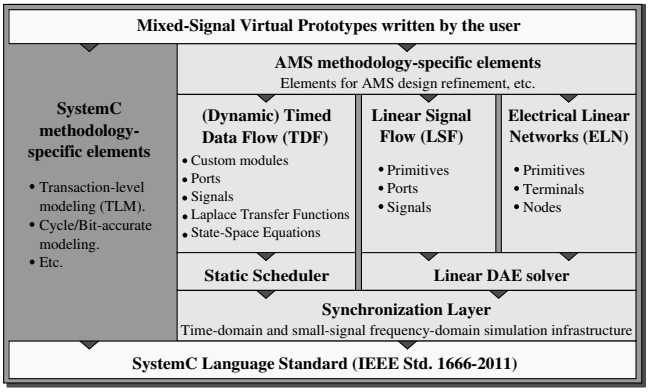

Figure 1. Architecture of SystemC AMS 2.0

under the responsibility of the user, who has to ensure that each signal in the model is finely sampled to obtain accurate simulation results and occasionally this can result in a significant simulation overhead.

To address this problem, Barnasconi et al. [5] introduced dynamic capabilities to change key properties in the TDF models such as the time step, rate, or delay attributes of the TDF ports and modules during execution. They presented use cases, requirements, basic execution semantics, and language constructs of a Dynamic Timed Data Flow (DTDF) MoC extension. This idea was recently formalized in the SystemC AMS 2.0 Standard [1]. The user is provided with new member functions that allow him to dynamically change the TDF attributes at specific times during simulation; new classes, which enable continuous-time and discrete-time decoupling of TDF clusters; a new template class to define the default interpolation mechanism for the continuous-time decoupling port; new TDF port classes and member functions to enable event-driven TDF module activation; new members functions to set and get the maximum timestep used in modules and ports; new member functions to return the last timestep value used and to return the maximum value of the simulation time.

Some of the new member functions described in the standard [1], which allow TDF models to dynamically manage the change of its attributes are presented below:

– **accept_attribute_changes()**: marks a TDF module to accept attribute changes caused by others TDF modules, which are part of the same TDF cluster.

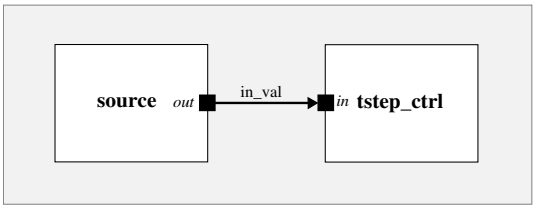

Figure 2. Structural composition of the time step controller

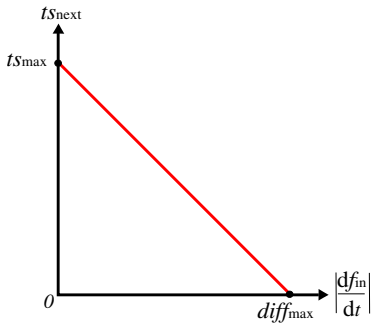

Figure 3. Representation of the linear equation used to calcule ts_{next}

- **does_attributes_changes()**: marks a TDF module to allow it to make itself attribute changes after all change_attributes() callbacks of the current cluster have been executed.
- **change_attributes()**: provides a context to change attributes in the TDF module and its ports.
- **request_next_activation()**: overrides the propagated time step defined by the function set_timestep() of the TDF modules and ports for the next module activation.
- **set_max_timestep()**: defines the maximum time step between two consecutive samples.

II. Time Step Controller

The structural composition shown in Figure 2 is described in this section. It is presented a mechanism that determines the timestep *ts* required to correctly sample a continuous and differentiable input signal *in_val*. We establish that the required *ts* to sample the input signal at the time point t_{i+1}, will depend on the slope value of the input signal calculated between two samples taken at the time points t_i and t_{i-1}. This behavior, corresponding to the linear equation $y = mx + b$ plotted in Figure 3, can be expressed mathematically by Eq. (1), where the x coordinate corresponds to the difference value of the input signal with respect to the time $\frac{df_{in}}{dt}$, the y coordinate corresponds to the time step for the time point t_{i+1}, and the cut-off points of the straight line with the x and y coordinates correspond with the maximum difference in amplitude $diff_{max}$ and the maximum time step ts_{max} allowed in the model, respectively.

$$ts_{next} = -\left(\frac{ts_{max}}{diff_{max}}\right)\left|\frac{df_{in}}{dt}\right| + ts_{max} \quad (1)$$

The equation indicates that if the change of the input signal with respect to time is increasing, then the next time step will be decreasing to ensure that all the variations of the signal will be sufficiently sampled. An implementation in SystemC AMS for the time step controller module is shown in Listing 1.

```
1  class tstep_ctrl : public sca_tdf::sca_module {
2
3      /* ... Port declarations ... */
4      /* ... Parameter constructor ... */
5      /* ... Initializations ... */
6
7      void set_attributes() {
8        does_attribute_changes();
9        accept_attribute_changes();
10     }
11
12     void change_attributes() {
13       double dy = in_val − in_prev;
14       double dx = t_val − t_prev;
15       double ts_next;
16       double diff_max = amp_max / ts_max;
17
18       if ( dx == 0.0 ) ts_next = ts_min;
19       else ts_next = ts_max * ( 1 − (fabs(dy/dx)/diff_max) );
20
21       if (ts_next > ts_max) ts_next = ts_max;
22       if (ts_next < ts_min) ts_next = ts_min;
23       set_max_timestep(ts_next, sc_core::SC_SEC);
24     }
25
26     void processing() {
27       /* ... reading and writting port values ... */ }
28 };
```

Listing 1. Time step controller implementation.

In the set_attributes() context (Listing 1, lines 7-10), the module is enabled to change its attributes and in the change_attributes() context (Listing 1, lines 12-24), the next time step (Listing 1, lines 18/19) that will be used as an attribute of the set_max_timestep() function (Listing 1, line 23) can be calculated. In general, the controller module sets the maximum time step that can be used during the next cluster activation.

III. Threshold Crossing Detection

To establish the exact time stamp at which an input signal crosses a threshold value called reference, it is necessary to ensure that if the input signal approaches this reference, the time step is made smaller and thereby it becomes possible to determine much more exactly the crossing instant. As in the SystemC AMS 1.0 standard it is not possible to implement such variation of the time step, the designer is forced to select a time step small enough to ensure the accuracy of the results. For example, if an input signal with variable frequency is considered, the time step selected will have to be chosen based on the maximum frequency value of the signal to fulfill the Nyquist-Shannon sampling theorem. Consequently, there will be oversampling in signal parts, where the frequency value is lower.

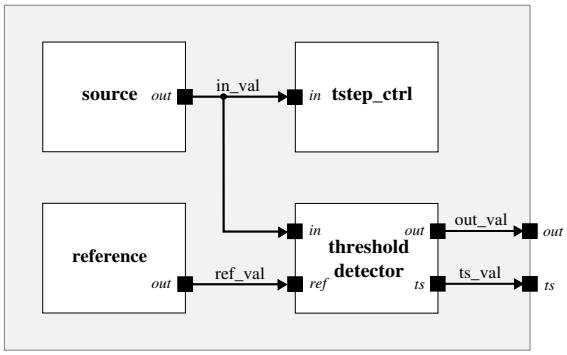

Figure 4. Structural composition of the threshold crossing detector

With the advent of the SystemC AMS 2.0 standard, it has become possible to define all the necessary conditions for adapting the time step value in the measure in which the input signal approaches from below or above the reference. To describe this behavior, two additional references are defined for setup a window in which the time will be decreased or increased in terms of the proximity of the input signal to the threshold. Outside of this window, the time step used can be either a value originally given by the designer or it can be a value set by another module such as the controller module presented in the Section II.

Figure 4 shows the structural composition used to implement this behavior. Inside the change_attributes() context (Listing 2, lines 12-23) of the threshold detector module implementation, it is necessary to define a function to calculate the next time step (Listing 2, lines 18-20) which will be used as argument in the request_next_activation() function (Listing 2, line 21).

```
1  class threshold_detector : public sca_tdf::sca_module {
2
3      /* ... Port declarations ... */
4      /* ... Parameter constructor ... */
5      /* ... Initializations ... */
6
7      void set_attributes() {
8        does_attribute_changes();
9        accept_attribute_changes();
10     }
11
12     void change_attributes() {
13       ...
14       if ( get_time() == sc_core::SC_ZERO_TIME ) ts_init = ts_val;
15       else ts_init = get_max_timestep().to_seconds();
16
17       if ( !((in_val < ref_min) || (in_val > ref_max)) ) {
18         /* ... Calculating ts_init variation based on the proximity of the
19          input signal with respect to the reference ... */
20         ts_next = /* ... ts_init variation ... */
21         request_next_activation(ts_next, sc_core::SC_SEC); }
22     }
23
24     void processing() {
25       /* ... reading and writting port values ... */}
26 };
```

Listing 2. Threshold crossing detector implementation.

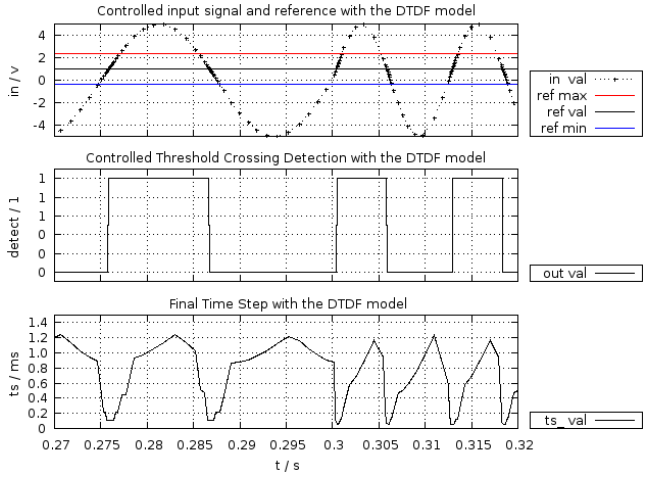

Figure 5. Threshold crossing detection results

The simulation results are shown in Figure 5, where the reference value is a constant signal set to 1.0[v] and the input is a sinusoidal signal with variable frequency and an amplitude set to 4.0[v]. The first trace allows to verify that the number of samples increases when the input signal is close to the reference; the second trace shows the detection result that is zero only when the input signal is below the reference; and the third trace shows the time step variation with respect to the simulation time.

IV. Conclusions

The paper presents some of the new features introduced by the SystemC AMS 2.0 standard. The described examples allow to verify how it is possible to change one important parameter associated to TDF models. It has been shown that time step variations in real models can be controlled dynamically during simulation to achieve accurate results. The implementations presented were developed using an early alpha version of Fraunhofer SystemC-AMS 2.0 [6], which has not yet been released for public use, but promises to expand the capabilities of SystemC AMS to model and simulate multi-disciplinary systems.

References

[1] ASI SystemC AMSWG, *Standard SystemC AMS extensions language reference manual*, version 2.0, Accellera Systems Initiative, Mar. 19, 2013. [Online]. Available: http://www.accellera.org/.

[2] M. Barnasconi, C. Grimm, M. Damm, K. Einwich, M.-M. Louërat, T. Maehne, F. Pêcheux, and A. Vachoux, *SystemC AMS extensions user's guide*, Open SystemC Initiative (OSCI), Mar. 8, 2010. [Online]. Available: http://www.systemc.org/.

[3] IEEE Computer Society, *1666-2005 IEEE standard SystemC language reference manual*, IEEE, Mar. 31, 2006, ISBN: 0-7381-4871-7.

[4] OSCI AMS Working Group, *Standard SystemC AMS extensions language reference manual*, version 1.0, Open SystemC Initiative, Mar. 8, 2010. [Online]. Available: http://www.systemc.org/.

[5] 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, 10 pp. [Online]. Available: http://www.accellera.org/resources/articles/amsdynamictdf (visited on 04/26/2013).

[6] *Fraunhofer SystemC-AMS*, Fraunhofer IIS, Design Automation Division EAS. [Online]. Available: http://systemc-ams.eas.iis.fraunhofer.de/ (visited on 04/29/2013).