Adaptive ODE Solvers for Timed Data Flow Models in SystemC-AMS

1st Alexandra Küster
Bosch Sensortec GmbH
Reutlingen, Germany
alexandra.kuester@bosch-sensortec.com

2nd Rainer Dorsch
Bosch Sensortec GmbH
Reutlingen, Germany

3rd Christian Haubelt
University of Rostock
Rostock, Germany

***Abstract*—The analog/mixed signal extensions to SystemC effectively tackle the needs for heterogeneous system integration using virtual prototyping. However, they introduce the inherent trade-off between accuracy and performance due to the discrete timestep. Besides the discrete-time scheduler, analog solvers are used within SystemC-AMS to solve linear ordinary differential equations (ODEs). In this paper, we derive two methodologies to integrate adaptive ODE solvers into SystemC-AMS that estimate the optimal timestep based on error control. The main advantage of the approaches is the avoidance of time-consuming global backtracking. Instead, they fit well into the execution semantics and scheduling approach of SystemC-AMS. A detailed comparison of both integration schemes is given and they are evaluated using a MEMS accelerometer as classical example of a heterogeneous system.**

***Index Terms*—Virtual Prototype, SystemC-AMS, ODE, Solver**

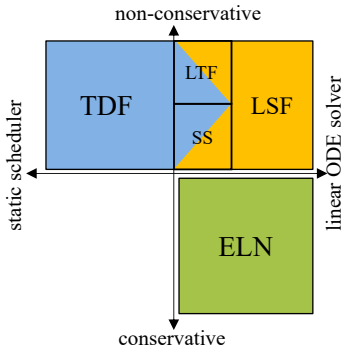


Fig. 1: Models of computation in the SystemC-AMS proof-of-concept implementation. Linear transfer functions and state-space equations are available in TDF and LSF MoC.

I. Introduction

Virtual prototyping in SystemC has evolved as an effective method to tackle early hardware/software integration [1]. In modern heterogeneous systems, software features are additionally interwoven with the analog/mixed-signal (AMS) parts of the design. Thus, heterogeneous full system virtual prototypes are necessary [2]. AMS extensions for SystemC [3] were introduced to serve these needs. Three different models of computation (MoCs) are supported for SystemC-AMS as shown in Fig. 1. The Timed Data Flow (TDF) MoC enables discrete-time simulation using a static schedule. Its modules implement a standardized interface including a *processing()* method that is executed repeatedly at each discrete time point. The Linear Signal Flow (LSF) MoC provides primitives to model non-conservative linear systems. The electrical linear network (ELN) MoC works as counterpart for conservative systems. Both MoCs are solved using linear ordinary differential equation (ODE) solvers. These solvers are additionally used if the pre-defined instances for linear transfer functions (LTFs) and state-space (SS) equations are used within a TDF module. In this case, the analog solver is (at least) executed for the discrete time points of the TDF grid to allow proper synchronization. The fixed timestep can be a major bottleneck for the simulation performance. SystemC-AMS 2.0 introduced dynamic features that allow to change timesteps during simulation. Although these features theoretically offer broad possibilities to optimize performance, the timestep is still subject to the designer's choice. In numerical analysis, variable-step solvers are well established in the context of ordinary differential equations. They adapt the timestep based on error estimation [4]. As ODE solvers cover major parts of a SystemC-AMS simulation, it is desirable to fit these approaches into its simulation semantics to allow more efficient and convenient analog simulation. The main contribution of the paper at hand is that we present two methodologies to integrate variable-step ODE solvers into SystemC-AMS. The first one pre-calculates the sample for the next step and delays it one sample before it gets visible at the output. Thus, local iterations are possible trying different step sizes. This approach is well-suited for feedback loops. The second method calculates each sample in-time. If the timestep estimation has failed and the error is too large, the timestep is split into multiple segments to meet the required level of accuracy. This approach is advantageous for systems with multiple timing sensitivities as multiple timesteps can be overlayed without disturbing the calculations. Both approaches preserve the SystemC-AMS simulation semantics. Backtracking is done locally for the ODE and not globally to avoid time-consuming recalculation of the whole cluster. This also avoids global checkpointing, i.e. to store all internal states which might be numerous for large models.

II. Related Work

To cope with the needs of heterogeneous and smart systems where analog components play a major role, several work has

979-8-3503-4859-0/DATE24/© 2024 EDAA

been done to enable effective virtual prototyping in SystemC-AMS. M. Lora et al. [5] focus on the model development proposing automatic translation and abstraction schemes to generate SystemC-AMS models from hardware description languages. SystemC-AMS also gained interest for verification purposes. M. Hassan et al. [6] e.g. apply data flow testing to TDF models for this purpose. A performance comparison of analog modeling languages can be found in [7]. The dynamic features of SystemC-AMS have been addressed by L. Porras et al. [8] who present two fundamental concepts for timestep adaption in discrete-time modeling, i.e. gradient-based timestep control and threshold crossing detection. To the best of our knowledge, neither control mechanisms for ODE solvers nor control mechanisms with inherent error estimation have been proposed so far. For the sake of brevity, we skip a detailed listing of publications on ODE solvers here and limit ourselves to common methodologies from numerical analysis.

III. Variable-Step ODE Solvers

ODE solvers are divided up into explicit and implicit methods. An explicit solver calculates the solution y_{k+1} of the next timestep based on the states of the current timestep t_k and the step size h:

$$\dot{y} = f(t, y, x) \tag{1}$$

$$y_{k+1} = y_k + hf(t_k, y_k, x_k) \tag{2}$$

Opposing to a classical initial value problem, an additional input x is used to actuate the system in our case. The easiest explicit solver is the first order forward Euler method [9]:

$$y_{k+1} = y_k + h\dot{y}_k \tag{3}$$

Higher order schemes calculate multiple supporting points to decrease the approximation error which scales with h^{p+1} if p is the order. As the solution for each step is computationally cheap, explicit solvers can be very fast. However, for stiff systems, they show stability issues and extremely small timesteps might be necessary to converge to a proper solution. In this case, implicit solvers are advantageous. They have improved stability properties but each step is computationally more expensive as they need to solve an equation of the form

$$y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}, x_{k+1}) \tag{4}$$

in each step. Again, the easiest example is an Euler step, called backward or implicit Euler:

$$y_{k+1} = y_k + h\dot{y}_{k+1} \tag{5}$$

Independent of the above mentioned types, variable-step solvers may use the same fundamental concept for the timestep control. To estimate the error of the solution, a second solver scheme is applied and the results are compared. Usually these schemes share intermediate solutions to limit the computational overhead for the second solving. We exemplarily extend our implicit Euler scheme by the second-order implicit trapezoidal method

$$y_{k+1} = y_k + \frac{1}{2}h(\dot{y}_k + \dot{y}_{k+1}). \tag{6}$$

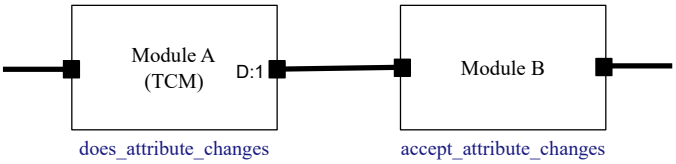


Fig. 2: Proposed structure within a TDF cluster. Module A implements the time controlling module and includes the ODE.

Applying both schemes for step k enables us to estimate the normalized error to

$$\epsilon = \frac{\|y_{k,trapez} - y_{k,euler}\|}{\epsilon_{abs} + \epsilon_{rel}(a_x|y_k| + a_{\dot{x}}h|\dot{y}_k|)} \tag{7}$$

with ϵ_{abs} and ϵ_{rel}, a_x and $a_{\dot{x}}$ being user defined thresholds and coefficients to fit the control scheme to the application's accuracy constraints, respectively. To cover the case of multi-dimensional outputs $\|\cdot\|$ denotes the maximum norm. The error is necessary to estimate the optimal timestep. A classical formula used for this purpose is

$$h_{new} = \begin{cases} h_{old} \cdot \max(0.9\epsilon^{\frac{-1}{O_E-1}}, 0.2) & \text{if } \epsilon > 1 \\ h_{old} \cdot \min(0.9\epsilon^{\frac{-1}{O_S}}, 5.0) & \text{if } \epsilon \leq 0.5 \\ h_{old} & \text{else} \end{cases} . \tag{8}$$

Thereby, O_E denotes the order of the error, and O_S denotes the highest solver order. The formulas for ϵ and h are implementation specific. We refer to these as they are available within the C++ library Boost [10] which we have used for our experiments. In order to handle arbitrary networks and systems, an effective implicit method must be given to ensure stability. However, especially the TDF instances are often used to quickly model small ODE systems which are not necessarily stiff. Here, explicit solvers can be an effective enhancement.

IV. Embedding Variable-Step Solving into SystemC-AMS

The proof-of-concept implementation of SystemC-AMS provided by Accelera [11] provides the two implicit methods explained above, i.e backward Euler and trapezoidal method. As the library does not include any explicit ODE solvers, we used adaptive Runge-Kutta solvers from the open-source library Boost for our experiments. For details on the family of Runge-Kutta solvers we refer to J. Butcher [12].

A. Method 1: Adaptive Pre-Calculation

Let us assume an arbitrary TDF cluster consisting of multiple TDF modules for our investigation. Each module can either set a timestep explicitly or inherit it from its neighbours. To enable dynamic timesteps, all modules must set the *accept_attribute_changes* flag. The TDF module that embeds an ODE is called the **time controlling module (TCM)**, see Fig. 2. It sets the *does_attribute_changes* flag and implements the *change_attributes()* function. Fig. 3 illustrates the first proposed synchronization scheme. Fig. 3a shows the synchronization between the TCM and the simulation kernel whereas Fig. 3b illustrates the exemplary internal processing

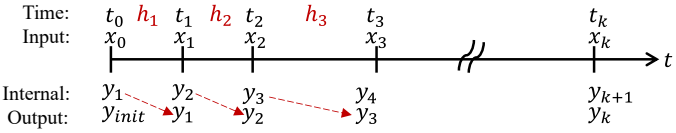
of the TCM for the k^{th} execution step. We derive the methodology using an explicit solver scheme. As the solution y_{k+1} hence only depends on the states of the previous step, it is possible to calculate one step in advance, i.e. the solution y_1 that corresponds to solver time t_1 is calculated at the kernel's simulation time $t_0 = 0$ using initial conditions for y_0, $\dot{y}_0$ and a step size $h_1 = t_1 - t_0$. Equivalently, y_{k+1} is calculated at simulation time t_k. The shift between solver and simulation time is compensated by delaying the output signal by one sample. To control the timestep, two different solvers are used to obtain values for y_{k+1}. In Fig. 3, the low order methods Euler and Heun are chosen but they can be replaced by arbitrary solver schemes in general. The results are used to estimate the error. If the error is too large, the state vector is backtracked, and the calculation is repeated using a smaller step size. Otherwise, the state vector is updated to the new value and a step size h_{k+2} for the next step is estimated which is either equal to h_{k+1} or increased if the error was below 0.5. Calculating one step in advance allows us to conduct multiple tries for h_{k+1} without the need of global checkpointing but requires a commitment to the upcoming step. As we iterate locally within the module, we must only checkpoint the local ODE states until a valid solution is found. Then, the *processing()* call ends and the module's timestep is set to the targeted h_{k+1} within the *change_attributes()* method. Considering the output delay, the calculated solution y_{k+1} will thus appear at the output at $t_{k+1} = t_k + h_{k+1}$.

We conducted our experiments using the open-source library BOOST. It provides adaptive explicit Runge-Kutta solvers. The error and step size estimation are based on Eqs. [7, 8]. The library provides controlled steppers that implement the function *try_step()* with the functionality as marked in Fig. 3b. Our SystemC-AMS module contains a loop that executes this function until the return value signals success. In this case, the state vector is updated and the step size is updated to an estimation for the consecuting step. The step size used for the final calculation is saved to apply it within the *set_attributes()* function of the TDF module.

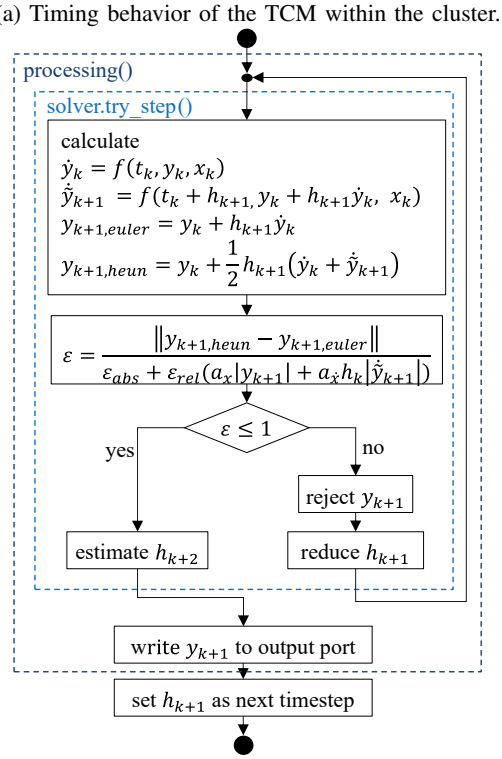
As mentioned, calculating one step in advance is easily feasible for explicit solvers. In implicit methods, solving y_{k+1} is coupled with solving $\dot{y}_{k+1}$ which in principle requires the knowledge of x_{k+1}. One possible solution is to estimate the next input, e.g. via linear extrapolation and follow the same scheme as described before. This might cause significant errors if the input signal is not smooth. In the following, a second synchronization scheme is presented that does not require the sample shift. We will discuss the advantages and drawbacks of the different approaches later based on the example of a MEMS accelerometer.

B. Method 2: Adaptive In-Time Calculation

The subsequent synchronization scheme is elaborated for implicit solvers but also works well with explicit solvers. One processing step of a TCM is illustrated in Fig. 4. The solution y_k for the current step is solved in-time and the resulting error ϵ is estimated. If the error is within its limits, the module



(a) Timing behavior of the TCM within the cluster.



(b) Processing sequence of a single TCM execution. The last step is performed in the *change_attributes()* method.

Fig. 3: Proposed synchronization scheme using adaptive precalculation.

calculates the next step size h_{k+1}, writes the current solution to the output port and ends its processing step. Again, the new step size h_{k+1} is applied within the *change_attributes()* method. However, if the error exceeds its limits ($\epsilon > 1$), we cannot arbitrarily decrease the step size anymore as the current module time is already fixed. Instead, we split h_k that has been applied to reach the current time t_k into multiple segments. The number of segments N dependends on the error. Here, it is set to $\lceil \epsilon \rceil$ as we work with low-order methods. For higher order methods, it might be sufficient to reduce it to $\lceil \sqrt[O_E]{\epsilon} \rceil$ as the error scales with h^{O_E}. However, as long as the signals are smooth, the majority of splits will result in $N = 2$ either way. The solution y_k is reverted and recalculated in N iterative steps of size h_k/N using linear interpolation for input x. Consequently, the full interval remains unchanged and the synchronization between solver and simulation time is ensured. The solutions calculated for $t_k + nh_k/N, n < N$ are solely used as intermediate solutions for the ODE solver, i.e. they never get accessible for connected modules. The step sizes for all other modules in the cluster keep unaffected. Fig. 5 shows an extraction of a simulation with and without interval split. The step size estimation of the implicit solver

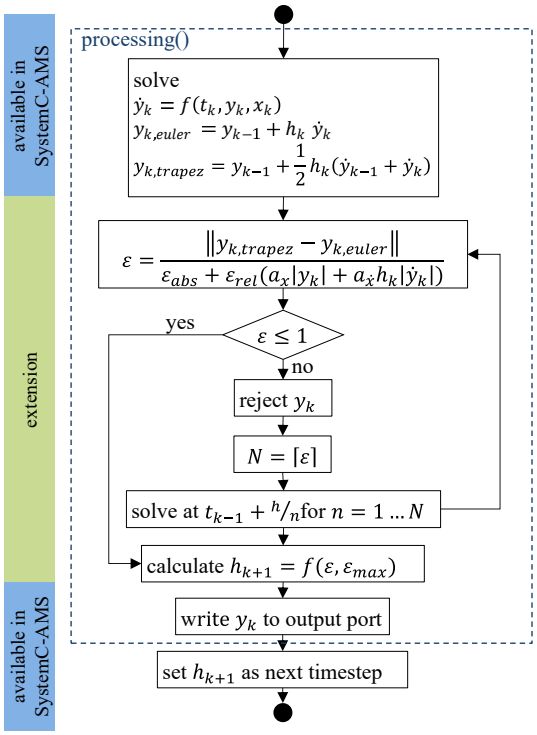


Fig. 4: Processing sequence of a single TCM execution for the in-time calculation scheme.

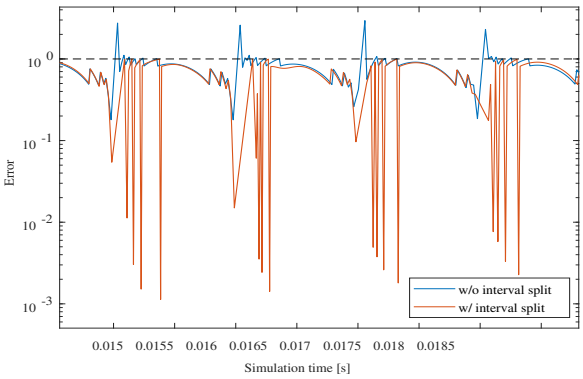


Fig. 5: Implicit ODE solver error with and without interval split for $\epsilon > 1$.

fails for 177 of 4123 steps (4.3%). When we enable the interval split, the number of steps with $\epsilon > 1$ reduces to 6 out of 4144 (0.14%). We limited the maximum number of segments to prevent numerical issues at discontinuities of the input signal. The majority of splitted intervals thereby only needs $N = 2$ segments. As the initial error is used for the timestep estimation, the total number of steps merely changes and the interval split mainly shifts the data points with $\epsilon > 1$ to lower errors. Still, additional 308 calculations (+7.4%) were executed for the splitted recalculations. We have extended the TDF MoC by the following functionalities with *ltf* being a TDF instance for a linear transfer function:

- *ltf*.**init_solver**(ϵ_{abs}, ϵ_{rel}, [h_{min}, h_{max}])
- y = *ltf*.**calculate_adaptive**(*num*, *den*, x, &s, &h, [&ϵ])

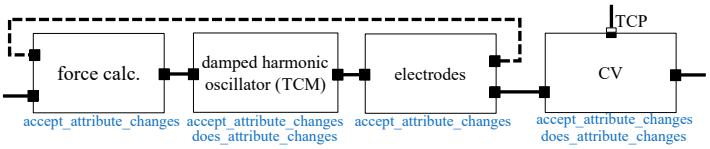


Fig. 6: Block diagram of a MEMS accelerometer model in SystemC-AMS. Each block refers to one TDF module.

All symbols are used as above and s denotes the state vector. The former function is called during initialization and the latter replaces the classical calculation method for these instances. It requires a pointer to the step size variable and adapts it based on the scheme. The user must take care to apply the timestep within the *change_attributes* method. Optionally, the resulting error can be read out. Similar functions exist for SS instances.

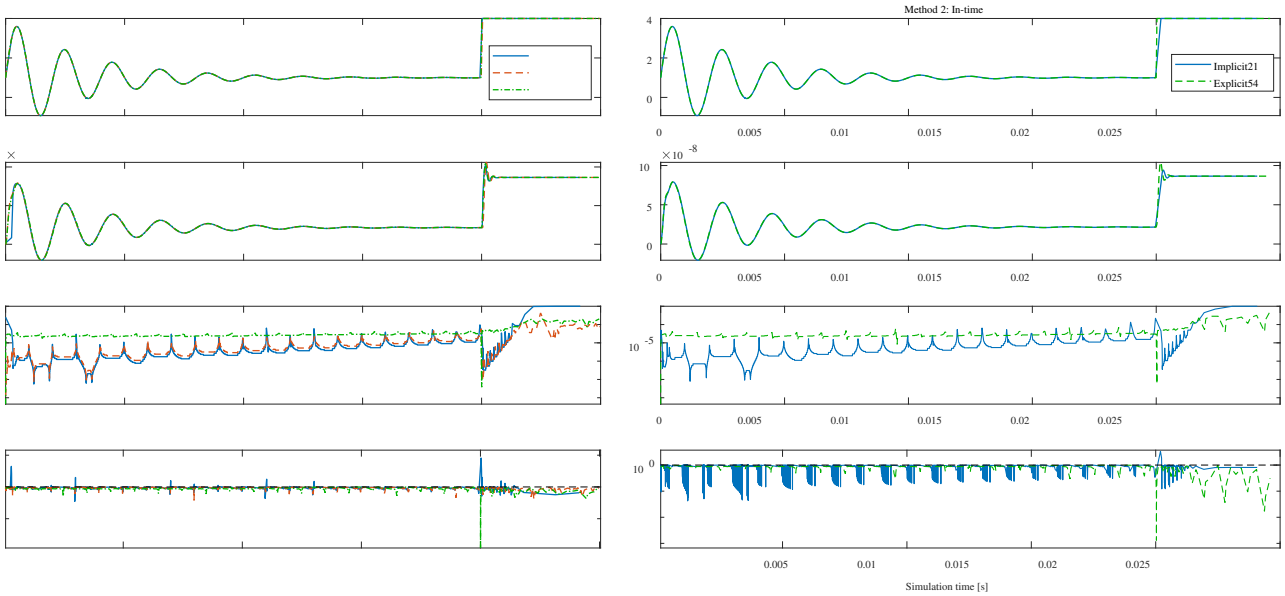
V. Application to a MEMS Accelerometer

The resulting step size using the proposed schemes is evaluated for the example of the MEMS accelerometer model shown in Fig. 6. We ignore the dotted line and the port denoted with TCP for now. The moving spring-mass system is modeled as damped harmonic oscillator. For many system-level investigations, it is sufficient to consider the fundamental mode. It can be easily described as LTF in the form

$$H(s) = \frac{X(s)}{F(s)} = \frac{1}{m}\frac{1.0}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{9}$$

with $X(s)$ being the deflection, $F(s)$ the force acting on the mass m, ω_0 the resonance frequency, and Q the quality factor. As this ODE reflects the main functionality of the system and defines the dynamics, it is well suited to control the cluster step size. It is actuated by an inertial force which is proportional to the input acceleration. The electrodes are abstracted to its sensitivities to the deflection of the moving mass. The charge-to-voltage (CV) converter derives the analog output voltage of the system based on the MEMS capacitance. The accelerometer is actuated with a decaying oscillation as an example for a continuously differentiable signal and afterwards disturbed by a sudden input change to check its robustness to discontinuities.

Both methodologies have been applied to the model, each of them using implicit and explicit solvers. The simulation results are given in Fig. 7. The MEMS deflection follows the acceleration input till the input step appears. An overshoot occurs for the undercritically damped MEMS. The trapezoidal method and backward Euler method (orders 2 and 1) are used for the implicit solver (Implicit21). To allow fair comparison, the pre-calculation approach implements an explicit solver using Heun's method and a forward Euler step to get the same orders (Explicit21). Additionally, a Dormand-Prince Runge-Kutta solver (orders 5 and 4) is applied. Fig. 7a shows the results for the pre-calculation method. The timestep behaves similar for both low order versions, i.e. our system doesn't show stiff properties. It is further noticeable that the maximum timestep occurs at the inflection points where the second



(a) Adaptive pre-calculation method results.

(b) Adaptive in-time method results.

Fig. 7: Simulation results of the accelerometer model shown in Fig. 6 using the proposed methodologies with both implicit and explicit solvers. The numbering accounts for the orders of the solver schemes.

derivative is zero. As an Euler step reflects a linear extrapolation, the quality of the approximation is anti-proportional to the curvature. Similarly, the minimum step size occurs at the extreme points where the second derivative reaches its maximum. As the timestep is smaller than one, the higher error order of the Dormand-Prince solver increases the accuracy for similar step sizes. As the error is controlled to be less than but close to one, we cannot see a smaller error but a larger timestep in this simulation. At the discontinuity, all schemes react with a sudden decrease of the step size. However, the difference between two step sizes is limited and the number of iterations is too. This shall avoid that the solver gets stuck in endless iterations at discontinuities. Consequently, the low order implicit method does not achieve $\epsilon > 1$ for this sample. Fig. 7b shows similar step sizes for the in-time method. However, the error estimation looks differently as the error is decreased more than necessary when the interval split is applied. As this effects single samples, we get these negative peaks at the bottom. The advantages and drawbacks of the presented methods are mainly related to their interferences with other SystemC-AMS features. They are discussed below.

VI. Interference with Other Language Features

The calculation of one step in advance requires a commitment to the upcoming step size. In principle, it prohibits to incorporate additional dynamic timestep adaptions. Besides *set_timestep()*, SystemC-AMS also implements the methods *set_max_timestep(<sc_time>)* and *request_next_activation(<sc_event>)*. The latter registers an event to trigger an immediate cluster activation. The former limits the maximum timestep between two cluster activations. Both methods can be combined to activate a cluster either at an incoming event or latest after the maximum timestep.

We extend our example by a range selection for the CV converter. The sensor shall support different full-scale ranges (4g, 8g, 16g) which can be configured via a register. Our register interface is implemented within the DE part of the virtual system prototype. A two bit DE-TDF converter port is added to the CV converter that encodes the range selection. A range switch shall trigger the cluster. We call the corresponding port **time controlling port (TCP)**, see Fig. 6. Please note that this behavior only works for single-rate clusters. The issue is inherent to the SystemC-AMS scheduler and has been addressed by B. Fernandez-Mesa et al. [13] who propose a different synchronization between the analog and DE domain. As the pre-calculated solution y_{k+1} depends on the choice of h_{k+1}, the solution becomes invalid if the cluster is activated earlier than expected. Nevertheless, it becomes visible at the output port as soon as the cluster is triggered. Fig. 8 illustrates the issue. The explicit solver deals well with the activation caused by the incoming event at 0.04 s. However, the implicit solver shows a signal distortion. Global backtracking would be required to prevent the errorneous output. Another option is to correct the solver behavior from the next sample onwards. The solver checks whether the applied timestep h_{app} corresponds to the targeted step h_{tar}. If not, it sets the next step size to $h_{next} = h_{tar} - h_{app}$, skips all calculations for this step and reapplys the same value again (yellow curve), i.e. the same value occurs twice, first at the disturbance event and secondly at its nominal point in time. As a cluster activation cannot be delayed by an event, $h_{next} > 0$ is ensured. If the error is not corrected (red curve), the solution converges back slowly to its nominal values. Correcting sparse disturbances due to events may not be a major issue but the effect prohibits to have

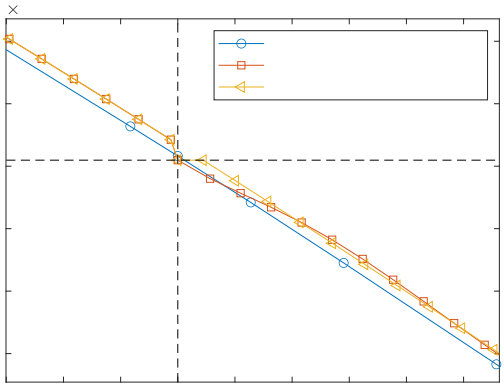


Fig. 8: Impact of a cluster activation caused by an sc_event at 0.04 s. The pre-calculated sample is based on an incorrect timestep assumption distorting the signal. The step recovery reapplies the calculated value at the correct point in time.

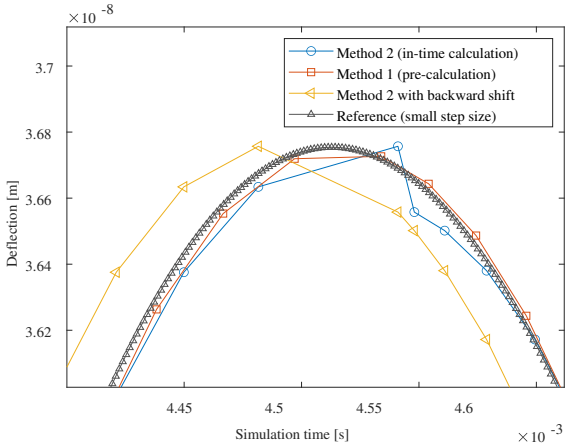


Fig. 9: Simulation of a feedback loop. The in-time method suffers from distortions due to the mandatory delay. To clarify the effect, a post-processing is shown that realigns the samples via backshift. The distortion vanishes.

TABLE I: Feature overview of the proposed methodologies.

	Pre-Calculating Approach	**In-Time Approach**
Explicit Solver	applicable	applicable
Implicit Solver	applicable with input estimation	applicable
Output delay	required	forbidden
Feedback loops	well-suited	incompatible
Sensitivity to events	creates punctual distortion	well-suited
Multiple TCMs	incompatible	well-suited

multiple concurrent TCMs for the pre-calculating approach (method 1). Here, the in-time approach is advantageous as it natively adapts to deviating timesteps. Using multiple TCMs is thus only supported for this method.

On the other hand, the pre-calculating approach is advantageous for feedback loops. As SystemC-AMS does not allow to iteratively converge to a solution, it cannot solve algebraic loops. As a consequence, TDF clusters require a delay element inside each loop. As the first approach includes a delay anyway, it solves the question where to put the delay element. Contrarily, no delay should be added to the loop in the second approach. If y_k is calculated using h_k but gets delayed by one sample, it occurs with a distance of h_{k+1} to the previous sample which distorts the signal. Similar effects occur if the delay is shifted within the loop. Fig. 9 illustrates this effect. For this simulation, the MEMS model in Fig. 6 has been extended with the feedback loop (dotted line) to account for the electrostatic force acting on the MEMS. It is proportional to the electrical sensitivity that depends on the current deflection. Table I summarizes our investigation.

VII. Conclusion

Two methodologies to integrate variable-step ODE solvers into SystemC-AMS have been presented. Their major advantage is the avoidance of global checkpointing. Which method is favorable is case dependent as both have their own advantages regarding the interference with other language features.

All in all, variable-step ODE solvers offer high potential to ease the handling of the inherent trade-off between accuracy and performance in SystemC-AMS simulations.

References

[1] B. Kim, Y. Kashiba, S. Dai and S. Shiraishi, "Testing Autonomous Vehicle Software in the Virtual Prototyping Environment," in IEEE Embedded Systems Letters, vol. 9, no. 1, pp. 5-8, March 2017.

[2] A. Küster, R. Dorsch, C. Haubelt and K. Einwich, "Virtual Prototyping in SystemC-AMS for Validation of Tight Sensor/Firmware Interaction in Smart Sensors," 2022 Forum on Specification & Design Languages (FDL), Linz, Austria, 2022, pp. 1-8.

[3] "IEEE Standard for Standard SystemC(R) Analog/Mixed-Signal Extensions Language Reference Manual," in IEEE Std 1666.1-2016 , vol., no., pp.1-236, 6 April 2016.

[4] R. L. Burden and J. D. Faires, "Initial-Value Problems for Ordinary Differential Equations" in *Numerical Analysis, Ninth Edition*, Brooks/Cole, Cengage Learning, 2011, pp.260-320.

[5] M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia and F. Fummi, "Analog Models Manipulation for Effective Integration in Smart System Virtual Platforms," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 2, pp. 378-391, Feb. 2018.

[6] M. Hassan, D. Groe, H. M. Le and R. Drechsler, "Data Flow Testing for SystemC-AMS Timed Data Flow Models," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019, pp. 366-371.

[7] W. Scherr and K. Einwich, "Beyond real number modeling: Comparison of analog modeling approaches," 2020 Forum for Specification and Design Languages (FDL), Kiel, Germany, 2020, pp. 1-5.

[8] Liliana Lilibeth Andrade Porras, Torsten Maehne, Marie-Minerve Louërat, François Pêcheux. Time Step Control and Threshold Crossing Detection in SystemC-AMS 2.0. Huitième colloque du GDR SOC-SIP du CNRS, Jun 2013, Lyon, France. pp.3. hal-00879835

[9] A. Struthers and M. Potter, Differential Equations: For Scientists and Engineers, Springer Cham, 2019, pp. 373-48.

[10] "Steppers", *boost C++ libraries*. Accessed: 2023/08/15. [Online]. Available: https://www.boost.org/doc/libs/1_83_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/odeint_in_detail/steppers.html

[11] "SystemC", *Accelera Systems Initiative*. Accessed: 2023/08/15. [Online]. Available: https://accellera.org/downloads/standards/systemc

[12] Butcher, John C. (2000), "Numerical methods for ordinary differential equations in the 20th century", J. Comput. Appl. Math., 125 (12): 129.

[13] B. Fernández-Mesa, L. Andrade and F. Pétrot, "Accurate and Efficient Continuous Time and Discrete Events Simulation in SystemC," 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2020, pp. 370-375.