Structural Generation of Virtual Prototypes for Smart Sensor Development in SystemC-AMS from Simulink Models

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*—We present a flow to reuse system-level analog/mixed-signal (AMS) models developed in MATLAB/Simulink for the extension of virtual prototypes in SystemC. To prevent time-consuming co-simulation, our flow translates the Simulink model into an equivalent SystemC-AMS model. Translation is supported either by wrapping code generated by MATLAB's Embedded Coder or by instantiating previously generated models. Thus, a one-to-one mapping of the model's hierarchy is possible which allows deep insights into the architecture and good traceability. The conducted case study on an accelerometer model shows the applicability of our approach. The generated hierarchical model is half as fast as a monolithic version but allows better observability and traceability of the system. It is tens of times faster than simulation in Simulink. The extended virtual prototype aims to support software engineers during development and validation of firmware in smart sensors.**

***Index Terms*—MATLAB/Simulink, SystemC-AMS, Virtual Prototype, Smart Sensor**

I. Introduction

Virtual prototypes (VPs) in SystemC are broadly used to handle the complexity of heterogeneous systems and allow the parallelization of hardware and software development. Smart sensors composed of mechanical, analog/mixed-signal (AMS), digital and software components show strong needs to integrate AMS features into the prototype [1]. We identified MATLAB/Simulink as an environment that is widely used for creating abstract component models for first exploration tests. We propose their reuse for the VP by automized translation into equivalent SystemC-AMS models. The advantage compared to the translation of RTL models as proposed in [2] is the early availability of these models. Our flow goes along well with existing translation options for digital parts (see [3], [4]). This supports the software development without creating additional modeling effort or co-simulation needs and ensures that designers and software developers have a common information and simulation base. Thereby, we achieve high modularity and traceability by preserving the structure and hierarchy of the model. It also allows good insights into the architecture.

II. Proposed Translation Flow

The proposed flow includes two major features as shown in Fig. 1. Monolithic modules can be generated for atomic blocks or complete models. They are written in the timed dataflow (TDF) model of computation of SystemC-AMS. Hierarchical modules instantiate components of the hierarchy level below, thus the generation can only be done as subsequent step. The process is repeated recursively with the newly generated instantiating modules to translate multiple hierarchy layers. Thus, the structure of the Simulink model is preserved and tracing is possible on each layer. Both steps are fully automized using MATLAB scripts. One major restriction applies. The flow is only supported for fixed-step models.

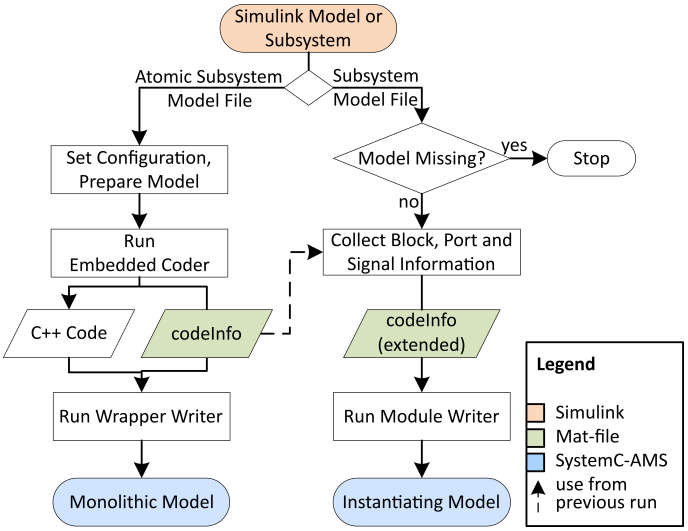

Fig. 1: Automized translation flow with its two options for the user. Some components are applicable to both branches.

MathWorks provides the Embedded Coder to generate C/C++ code from fixed-step Simulink models. The execution semantics of the generated code is similar to a TDF model and can thus be wrapped as SystemC-AMS module as shown by Kleen et al. [5]. The Embedded Coder also extracts a mat-file summarizing properties and interfaces of the generated code (see Fig. 1) that can be used as information source for the wrapper writing. Buses are set to non-virtual to avoid structural loss, otherwise the EC coder splits them into independent

978-3-9819263-7-8/DATE23/© 2023 EDAA

signals which can lead to ambiguous naming. Special data types as fixed-point formats are remapped with correct scaling.

The procedure for monolithic modules may be applied to all hierarchy levels of the Simulink model if the subsystems are set to atomic. Nevertheless, our flow allows using instantiation to prevent the lack in traceability of internal signals and readability. SystemC-AMS code must exist for all components. We developed a SystemC-AMS library with the most important primitives that are often used outside subsystems, i.e. to-file blocks, memory and delay blocks, type conversions, rate transitions and bus creators or selectors. All other primitives must be sorted into subsystems as functional groups. Port and signal information is collected to allow the correct connection of the components. Moreover, parameters are fed through to the higher level. Attention must be taken for multi-rate systems as monolithic modules always run on their minimum sampling time. If multiple multi-rate monolithic modules are connected, this can lead to inconsistent timesteps. Our script adds rate transitions where necessary. The modelers themselves must only take care for the avoidance of algebraic loops. MATLAB/Simulink can often resolve them itself but SystemC-AMS cannot and requires the insertion of a delay element. As delays in feedback loops can influence the simulation behavior significantly, we decided not to insert them automatically.

III. Experimental Results

We conducted performance tests to compare the efficiency of generated code to handwritten modules. We used the sigma-delta analog-to-digital (ADC) converter from ACCELERA's application examples [6] and created equivalent handwritten and generated models for comparison. Results are given in Table I. The generated models do not suffer from significant performance loss. The results also indicate that monolithic models are slightly faster than composed ones which leads to a trade-off between traceability and performance.

We investigated the applicability of the approach using a case study of an accelerometer. A block diagram of the model is given in Fig. 2. The model includes three levels of interest, i.e. the full sensor, its split into micro-electro-mechanical system (MEMS) and ASIC and the split into basic building blocks like the CIC filter. Monolithic modules are marked yellow. Higher levels are generated as instantiating models. For comparison, the full accelerometer is additionally generated as monolithic module. During the process, a delay has been added at the output of the stimulus block inside the ASIC to avoid an algebraic loop. We have validated that all three model versions (Simulink, monolithic and instantiating SystemC-AMS) behave similar. Their simulation performances are given in Table II. The monolithic model is twice as fast as the hierarchical one but it suffers from reduced observability as the structure and internal signals are unaccessible. Both models are tens of times faster than the Simulink version.

IV. Conclusion

We have presented a translation flow to generate SystemC-AMS models from Simulink models to allow the fast generation of heterogeneous virtual prototypes which are important for firmware development. The translation offers the possibility to preserve hierarchical structures. Experimental results show that the SystemC-AMS models run much faster than their Simulink twins and the simulation performance is not degraded compared to hand-written models in our case.

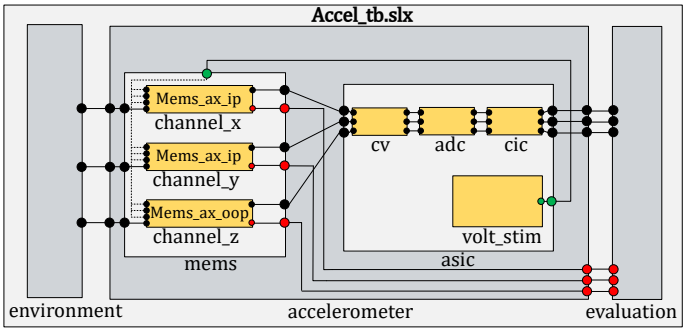

Fig. 2: Schematical view of the system under test. Yellow marked blocks are used for monolithic module generation. Red and green ports describe different bus types.

TABLE I: Simulation performance of a sigma-delta ADC. 1.0 s have been simulated with a step size of 1.0 µs.

	Language	Model Specification	Sim. Time
discrete	Simulink	basic blocks (discrete integrators)	4.8 s
	SystemC-AMS	loop filter, quantizer, DAC (all TDF)	0.90 s
	SystemC-AMS	single analytical TDF module	0.40 s
	SystemC-AMS	**generated** code with wrapper	0.31 s
cont.	Simulink	basic blocks, ode8 (Dormand-Prince)	19.79 s
	SystemC-AMS	ACCELERA's application example	0.78 s
	SystemC-AMS	**generated** code with wrapper	1.57 s

TABLE II: Simulation times of the accelerometer model versions. 0.1 s are simulated with t_s=1 × 10^{-6} s.

Model	Simulation Time
Simulink	78.11 s
SystemC-AMS monolithic module	1.28 s
SystemC-AMS hierarchical module	2.64 s

References

[1] A. Küster, R. Dorsch, C. Haubelt and K. Einwich, "Virtual Prototyping in SystemC AMS for Validation of Tight Sensor/Firmware Interaction in Smart Sensors", Proceedings of the 2022 Forum on specification and Design Languages (FDL), 2022

[2] 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, doi: 10.1109/TCAD.2017.2705129.

[3] L. Zhang, M. Gla, N. Ballmann and J. Teich, "Bridging algorithm and ESL design: Matlab/Simulink model transformation and validation," Proceedings of the 2013 Forum on specification and Design Languages (FDL), 2013, pp. 1-8.

[4] K. Hylla, J. Oetjens and W. Nebel, "Using SystemC for an extended MATLAB/Simulink verification flow," 2008 Forum on Specification, Verification and Design Languages, 2008, pp. 221-226, doi: 10.1109/FDL.2008.4641449.

[5] H. Kleen, S. Xiao, R. Görgen, N. Bannow and W. Nebel, "Automatische Übersetzung von MATLAB/Simulink-Modellen nach SystemC-AMS," ITG GMM GI, 2011

[6] Accellera Systems Initiative, "SystemC AMS Application Examples", Accellera Systems Initiative, 2022. [Online]. Available: https://accellera.org/downloads/standards/systemc [Accessed: Dec. 14, 2022].