Demonstrator: Bridging Matlab/Simulink and ESL Design via Automatic Code Generation

Liyuan Zhang, Michael Glaß, and Jürgen Teich
Hardware/Software-Co-Design, Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
Email: {liyuan.zhang, glass, teich}@cs.fau.de

Motivation Matlab/Simulink is today's de-facto standard for model-based design in domains such as control engineering and signal processing. Commercial tools are available to generate embedded C or HDL code directly from a Simulink model. However, Simulink models are purely functional models and, hence, designers cannot seamlessly consider the architecture that a Simulink model is later implemented on. In particular, it is not possible to explore the different architectural alternatives and investigate the arising interactions and side-effects directly within Simulink. To benefit from Matlab/Simulink's algorithm exploration capabilities and overcome the outlined drawbacks, we introduce a model transformation framework that converts a Simulink model to an executable specification, written in an actor-oriented modeling language. This specification then serves as the input of an established Electronic System Level (ESL) design flow, enabling Design Space Exploration (DSE) and automatic code generation for both hardware and software.

Demonstrator This demonstration shows how to automatically transform Simulink models to an established ESL design flow by means of a code generator. Based on the generated code, we present a co-simulation approach [1] that combines complex environmental models from Matlab/Simulink with the auto-generated model of a controller. We use an Anti-lock Braking System (ABS) as an example where we investigate the impact of different controller implementations in the automotive E/E architecture.

Code Generation To bridge Simulink and ESL design flows, we developed an ESL *Code Generator* to perform model transformation, see [2]. The idea is that for any given Simulink models such as a controller in a control system, the designer can simply invoke our Code Generator to create the ESL model automatically. In our design flow, we use SystemC as a programming language with an extension of actors with a specific Model of Computation (MoC). We guarantee the preservation of the semantics of the generated model by (a) applying a specific 1-to-1 mapping from Simulink basic blocks to an actor library and (b) considering different transformations to capture single-rate and multi-rate Simulink models. After the model transformation is finished, this auto-generated SystemC model serves as the input of a well-established ESL design flow that enables DSE.

Model Validation Besides the Code Generator this demonstrate also shows a *validation technique* that considers the functional correctness by comparing the original Simulink model with the generated SystemC model. The main idea behind this technique is (1) to co-simulate the auto-generated model along with the the original model and (2) to reuse the environment model and the test bench that are originally created in Simulink also for the auto-generated model. Furthermore, the performance of the model can also be measured during co-simulation.

Case Study An ABS model is transformed from Simulink to SystemC by invoking our Code Generator. Then, by applying our validation technique, the correctness and the accuracy of the auto-generated model is examined. Lastly, to evaluate the performance of the model, application-depended quality of control is measured, such as the braking distance on an icy road.

References

[1] M. Glaß, J. Teich, and L. Zhang, "A Co-simulation Approach for System-Level Analysis of Embedded Control Systems," in *Proceedings Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS 2012)*, Samos, Greece, jul 2012.

[2] L. Zhang, M. Glaß, J. Teich, and N. Ballmann, "Bridging Algorithm and ESL Design: Matlab/Simulink Model Transformation and Validation," in *Proceedings of Forum on specification and Design Languages (FDL 2013)*, 2013.