Automatic Code Generation for a Seamless Low-cost Development Platform

Sven Jacobitz and Xiaobo Liu-Henke
Ostfalia University of Applied Sciences, Department of Mechanical Engineering, Institute for Mechatronics, Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Germany

Keywords: Automatic Code Generation, Model Transformation, Rapid Control Prototyping, Model Based Design, Open Source.

Abstract: The automatic generation of Electronic Control Units (ECU) software from functional models is becoming an increasingly important part of the development process. At first, this paper presents the demand towards a tool chain for functional development of embedded systems and then introduces the low-cost development platform LoRra on the basis of the seamless model-based rapid control prototyping development process. The core of the paper is the conception and realization of the automatic code generator of the LoRra platform. It generates modular, flexibly usable C code, suitable for real-time implementation on a microcontroller. The code is generated from models of the open source CAE software Scilab / Xcos, which can be used across domains. A simple case study is used to verify the function of the generated code under real-time conditions.

1 INTRODUCTION

A key challenge for innovative enterprises today is to develop ever more complex products to production maturity ever more quickly. To meet the growing requirements resulting from this, more and more hardware, software and technical systems are being integrated. The core of the resulting embedded mechatronic systems are Electronic Control Units (ECU) with the intelligent signal processing functions implemented on them. Due to the rapidly increasing scope of functionality and degree of networking, this results in increasingly complex software components (Akdur et al., 2021).

To enable small and medium-sized enterprises (SMEs) to offer mechatronic products competitively, a seamless, highly automated low-cost rapid control prototyping development platform (LoRra) was developed at Ostfalia as part of an EU-funded research project. This is being used in the Lower Saxony Future Laboratory for Mobility to design intelligent functions.

This paper introduces the automatic code generation approach of the LoRra platform, which contains a model-to-text transformation for functions modeled the open source CAE tool Scilab / Xcos. The rest of the paper will be structured as follows. In section 2 the LoRra platform for rapid control prototyping (RCP) is introduced, followed by the state of knowledge for automatic code generation in section 3. The concept of the code generator is introduced in section 4 and the realization is described in section 5. Finally, the results of the case study are presented in section 6. The paper closes with an conclusion and outlook to future work in section 7.

2 RCP USING THE LoRra PLATFORM

RCP is a seamless, verification-oriented methodology for the development of complex ECU functions (e.g. closed-loop control systems) (Liu-Henke et al., 2021). It is effectively used in the automotive and aerospace industries. Through the consistent use of models, RCP enables the testing of functions at an early stage of development (Hanselmann, 1996). The whole process is accompanied by Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) simulations. It can be subdivided into the process steps model building, analysis and synthesis, automated code generation, automated implementation and online experiment (Lückel et al., 2001). The basis are simulations of the functional behavior, usually described by a block diagram model, in a

Jacobitz, S. and Liu-Henke, X.
Automatic Code Generation for a Seamless Low-cost Development Platform.
DOI: 10.5220/0010894300003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 294-301
ISBN: 978-989-758-550-0; ISSN: 2184-4348
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

combination with a simulation of the controlled system.

As illustrated in Figure 1, a Computer Aided Engineering (CAE) tool chain is used throughout the entire RCP process. The combination of Matlab / Simulink and dSPACE is often used in the industry. It supports the process by several tools. Since high costs are associated with the acquisition and maintenance, the low-cost alternative LoRra was developed as part of several research projects at Ostfalia (Jacobitz and Liu-Henke, 2020). LoRra is based on low-cost software and hardware and is suitable for the development of functions in a wide variety of domains due to its modular, flexible structure.

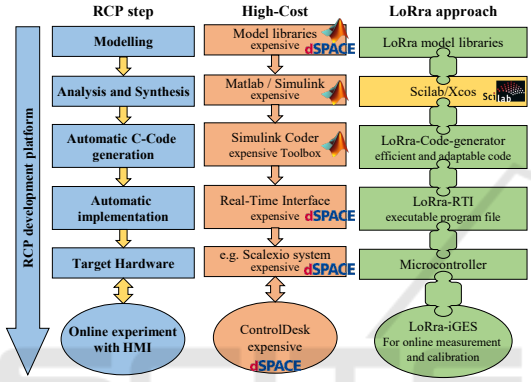


Figure 1: RCP development process with high-cost solution and LoRra approach (Jacobitz and Liu-Henke, 2020).

The open source tool Scilab / Xcos is used as the central CAE environment, which is suitable for simulating hybrid systems from a wide variety of domains (Nikoukhah, 2006). With the help of the model library, various continuous-time and discrete-time model components can be reused and parameterized. Through the LoRra code generator, efficient C code, executable on low-cost target hardware such as a microcontroller, is generated. Using the LoRra Real-Time Interface (RTI) (Jacobitz and Liu-Henke, 2019), this code is then automatically implemented into a run time environment and linked to the interfaces of the target hardware. A microcontroller is then used to run the real-time simulation and to carry out the online experiments. The measurement and calibration tasks during these experiments are supported by the human-machine interface iGES.

3 STATE OF KNOWLEDGE

Automatic code generation is an essential part of the RCP process. It leads to reproducible results as well as a fast implementation of the function designed and tested in the simulation model. Moreover, it avoids random errors and can generate highly efficient code due to special hardware-related optimizations (Toeppe et al., 1999). The basis for automatic code generation is the model-to-text transformation of the function model, which is usually given by a block diagram model.

In the automotive industry, automatic code generators have become standard development tools for ECU functions (Franco et al., 2016). Even in the aerospace industry, functions are frequently developed with the help of automatic code generators. The market offers a correspondingly large selection of different solutions and systems.

The CAE tool Matlab/Simulink is widely used in industry. With the extensions Simulink Coder and Embedded Coder, Simulink models can be transformed into universally usable or target hardware-specific optimized code (Lamberský et al., 2014). Many additional extensions are also available, e.g. for linking hardware peripherals. With TargetLink, dSPACE also offers a powerful production code generator that generates highly efficient code while taking target hardware-specific optimization criteria into account (Hanselmann et al., 1999).

The open-source CAE tool Scilab / Xcos, successor of Scicos developed within the Metalau project, has an integrated code generator. This generates C code for independent system simulation (so-called standalone simulation), which is intended for use with a functional mock-up interface, for example (Bucher and Balemi, 2005). For execution on a microcontroller under real-time conditions without manual adjustments, this is not suitable due to the program and memory structure.

Extensions such as X2C (Grabmair et al., 2014), developed at the Linz Center of Mechatronics, can also be used to generate efficient code from Xcos models that can be executed on microcontrollers. However, these extensions do not have the necessary interfaces for integration in the LoRra RCP process. To realize a seamless low-cost RCP development platform, the LoRra code generator, which is the core of this paper, was therefore designed and realized.

4 CONCEPTION OF THE LoRra CODE GENERATOR

The basis for consistent use in the cost-effective RCP development platform LoRra is the efficient and at the same time flexible transformation of Xcos models into microcontroller-suitable C code. Basics of modelling and simulation using Xcos, as well as the

mathematical background can be found in (Campbell et al., 2010). The generated code must correctly reproduce the model behavior, be reusable at the same time, and have interfaces for integration into a real-time environment (e.g. for the step from SiL to HiL simulation).

To achieve these goals, the LoRra code generator is designed modularly with open interfaces. Figure 2 illustrates the concept of code generation from Xcos models, using the LoRra code generator. First, a distinction is made between functional behavior of the individual blocks (section 4.1) and a topological description (section 4.2) - the interconnection of the blocks with each other. Transformation algorithms (section 4.3), which rely on external libraries, perform the model-to-text transformation and generate the code for the LoRra Code Interface (LCI - section 4.4).

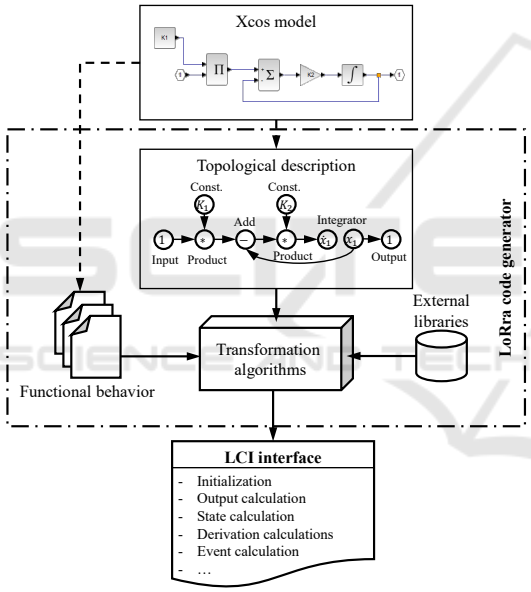


Figure 2: Concept of the LoRra code generator.

4.1 Functional Behavior

The functional behavior of a Xcos model is described by blocks, which establish a relationship between the inputs and outputs. A block in the model can be a basic block or a hierarchy block. Hierarchy blocks consisting of a subordinate model. For code generation, they are dissolved through their basic blocks and topology. Their structure is only kept in the LCI description (cf. section 4.4). Basic blocks establish an atomic functional relationship between their inputs and outputs and can thus be interpreted as operators of the model.

Xcos performs hybrid simulations of continuous-time and discrete-time systems. For this purpose, it distinguishes between regular (time continuous) and event (time discrete) signals. Figure 3 illustrates the functional relationship between inputs and outputs of a basic block. Using the general nonlinear state space representation of a dynamic system with the continuous states $\underline{x}$, the discrete-time states $\underline{z}$ and the parameter vector $\underline{r}$, the relationship between the l regular inputs $\underline{u}$ and outputs $\underline{y}$, where l is the number of regular inputs and k the number of regular outputs, is established (Nikoukhah and Steer, 1996). When the block is activated by an event input $\underline{d}$, the discrete-time states and the event delay vector $\underline{e}_{vz}$ are calculated according to Eq. (1). In addition, a non-continuous change of the continuous states can also occur here. An event d_i is described by an impulse function at time t_E (Campbell et al., 2010).

$$\begin{aligned} [\underline{x},\underline{z}]^T &= \underline{f}_2(\underline{x},\underline{z},\underline{u},\underline{d},\underline{r},t_E) \\ \underline{e}_{vz} &= \underline{f}_3(\underline{x},\underline{z},\underline{u},\underline{d},\underline{r},t_E) \end{aligned} \tag{1}$$

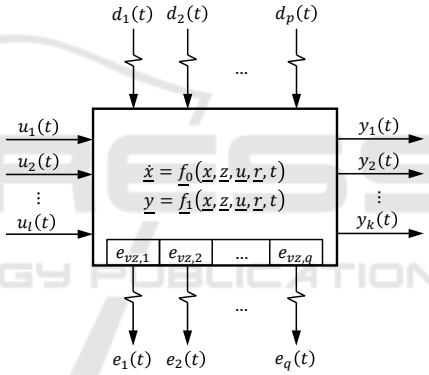


Figure 3: Functional relationship between inputs and outputs of a basic block.

4.2 Topological Description

The model topology is represented by a directed multi edge graph G according to Eq. (2). The basic blocks, represented by a set of nodes V, are thereby linked by regular edges R and event edges E. The definition of regular edges is represented by Eq. (3), where v is called the source node and w is called the target node. Event edges behave equivalently, where event inputs and outputs are linked.

$$G = \{V,R,E\} \tag{2}$$

$$R = \left\{ \begin{array}{c} \{v,w\}: v,w \in V \\ y \in w \\ u \in v \end{array} \right\} \tag{3}$$

4.3 Transformation Algorithms

The transformation algorithms perform the model-to-text transformation. They consist mainly of two parts: the function code generation and the linking of code components.To generate the functional related code of the basic blocks, detailed information about the block operations are required. This is given by the nonlinear state space representation, presented in section 4.1. The algorithms, describing the block behavior are stored in transformation rules. A distinction is made here between three basic elements (Deppe and Homburg, 1999):

- **Indirect Link Calculation (ND):** The calculated outputs do not depend on the inputs without delay (e.g. due to integration): $\left\{ y | y \in v \wedge \frac{\partial y}{\partial \underline{u}} = 0 \right\}$
- **Direct Link Calculation (D):** The calculated outputs depend on the inputs without delay: $\left\{ y | y \in v \wedge \frac{\partial y}{\partial \underline{u}} \neq 0 \right\}$
- **State Calculation (S):** Calculation of discrete states and derivatives of continuous states.

This is necessary for a correct calculation sequence especially for distributed simulation using multiple computing units. Additionally, there must be algorithms for the event calculations (E), performed during event activation, which contain the evaluation of Eq. (1).

The generated components of the functional code are linked to the LCI-based C code under consideration of the calculation sequence in such a way that the ND calculations are carried out first, since no information on the block inputs are necessary for this. Then, the signals to calculate the D outputs are available. Finally the S calculations could be done. The generated fragments of the ND, D, S and E calculation are merged and stored persistently.

4.4 LoRra Code Interface

The LCI aims at the reusability and universal applicability of the generated code. In addition to the code, information about the hierarchical model structure as well as the interfaces are stored in suitable data formats and integrated into an LCI package. Figure 4 illustrates the principle.

Standardized data structures and interfaces are provided so that a function can be used without knowledge of the inner structure of the code. Open interfaces for model initialization, calculation of block outputs (ND, D) and states (S) form the basis for the reusability of a generated function.

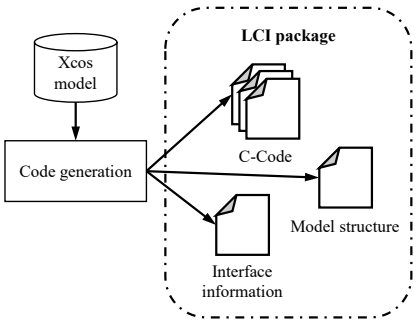


Figure 4: Concept of the LCI.

5 REALIZATION OF THE LoRra CODE GENERATOR

The code generation according to the concept from Figure 2 is realized by the process model shown in Figure 5. First, the distinction between functional behavior and topology as well as a preparation of the model structure is done by a pre-processing. Then, the transformation is done by the transformation algorithms. Last, a post-processing takes place, which outputs the LCI package.

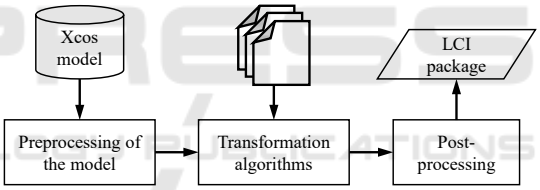


Figure 5: Process of automatic code generation.

5.1 Pre-processing of the Model

The pre-processing is used to prepare the model topology and to store it in a data structure, suitable for code generation. The results are a flat data flow graph G, defined by Eq. (2), and the LCI information for model structure and interfaces. The following steps are performed during pre-processing:

- Composition of the data flow graph
- Check for unsupported topology structures
- Optimization of topology
- Determination of the calculation sequence

A model-to-model transformation first transforms the Xcos model into a data flow graph. Here, all information relevant for code generation are transferred and superfluous properties (e.g. for graphical representation) are discarded. Hierarchy blocks are dissolved, resulting in a flat graph only consisting of basic blocks. For each basic block, the relevant data

for code generation (e.g. variable names, dimensions of inputs and outputs) are stored in a corresponding structure. In addition, not explicitly given sizes of regular signals are determined by backpropagation.

The structure of G is verified. Mainly this contains the search for algebraic loops, which can be detected as a cycle in the graph. Therefore, a depth-first search is performed, starting from the source nodes defined by Eq. (4). Sinks, defined by Eq. (5) are handled as leaf nodes. Going further, an optimization of the graph structure can be done at this point. Here, graph theory algorithms such as semantic matching or intelligent AI-based methods can be used to optimize various key performance indicators such as the number of nodes or the over all complexity.

$$V_r \subseteq V \Longleftrightarrow \forall v \in V_r : l = 0 \tag{4}$$

$$V_s \subseteq V \Longleftrightarrow \forall v \in V_s : l > 0 \wedge k = 0 \tag{5}$$

In a final step, the calculation sequence O is determined. It specifies the order in which outputs of the blocks must be calculated in order to avoid inconsistencies. Starting with the set of root nodes V_r according to Eq. (4), a breadth-first search based on algorithm 9 is performed therefore. The algorithm is performed, as long as the queue Q is not empty. The first element q is taken from the queue. If all predecessor nodes $N_G^-(q)$ were visited and q is not already within the calculation sequence, it is added. If q is not a signal sink according to Eq. (5), all successor nodes $N_G^+(q)$ of q are added to the queue.

Data: Set of root nodes V_r
Result: Calculation sequence O
$Q \leftarrow V_r$;
while $Q \neq \varnothing$ **do**
  $q \leftarrow$ first element of Q;
  $Q \leftarrow Q \cap \{q\}$;
  if $N_G^-(q) \in O \wedge \neg q \in O$ **then**
    $O \leftarrow O \cup \{q\}$;
    $Q \leftarrow Q \cup N_G^+(q)$;
  end
end

Algorithm 1: Breadth-first search for determining the calculation sequence.

5.2 Transformation Algorithms

A rule-based transformer is implemented to generate the function code. For each supported basic block, transformation rules that reflect the state space equation introduced in section 4.1 must be defined. The Scilab language is used as a descriptor for the transformation rules. The conceptual structures of an Xcos model defined by the provided basic blocks. Mainly, there is made a distinction between the calculation of state derivations (defined by the function f_0 in Figure 3), calculation of the outputs (defined by the function f_1 in Figure 3) as well as the event based calculations defined by (1). The LoRra code generator uses the individual given transformation rules to map these functions to the ND, D, S and E calculations.

The generated pieces of code are integrated into a LCI conform C-Code by using a template based approach. Thereby, the calculation sequence, determined by algorithm 1 is kept. Listing 1 contains the template for calculating the outputs as an example. The generated code components and other relevant information can be accessed via a predefined structure.

```
1  lci_status_t <lcg:functionName>
       _calcOutputs(uint8_t iCaller) {
2
3    // Indirect link calculations
4    <lcg:code.calcND>
5
6    // Direct link calculations
7    <lcg:code.calcD>
8
9    return LCI_OK;
10 }
```

Listing 1: Code template for the output calculation.

5.3 Post-processing

Within the post-processing, different tasks are executed depending on the configuration of the code generator. Here e.g. still another optimization on code level or the production of a separate documentation and the compilation of the program are possible. As a result, an LCI package is created and saved persistently. It can be embedded again in an Xcos model for SiL simulations, for example, or in a real-time environment for HiL simulations.

6 CASE STUDY

Various tests were successfully carried out to verify the LoRra code generator. In the context of this paper, the functionality is demonstrated by the case study of a speed controlled DC motor. Here, a PI controller according to Eq. (6) with anti-windup structure is used to control the speed of the traction motor for an autonomous guided vehicle. Figure 6 illustrates the model of this function in the Xcos environment.

$$G(s) = K_R \frac{T_R s + 1}{s} \tag{6}$$

For real-time testing, the interfaces of the model are linked to the interfaces of the target hardware via the LoRra-RTI. Hereby, the encoder, PWM output and digital output interfaces are operated. An STM32H7 microcontroller with a Cortex Microcontroller Software Interface Standard (CMSIS) compatible real-time operating system is used for the real-time simulation.

The result of automatic code generation and implementation into the real-time environment is demonstrated by listing 2. Scheduling is performed by the FreeRTOS operating system. First, the model is initialized, using the LCI-interface Function `PI_Controller_init`. Within an infinite loop, the reading / writing of hardware interfaces (`PI_Controller_readInputs` / `PI_Controller_writeOutputs`), the calculation of block outputs / state derivations (`PI_Controller_calcOutputs`) as well as the integration (`PI_Controller_callIntegrator`) are performed in predefined order. As integration method, the explicit Euler method is used here. Also, the measuring of the task turnaround time is performed by the RTI (`rti_ttTimer_start` and `rti_ttTimer_stop`).

```
// initialize the PI_Controller function
PI_Controller_init();
rti_tickCount = osKernelGetTickCount();

/* Infinite loop */
for(;;)
{
  // query memory status and protect memory
  if(rti_lock_mem(0U)==osOK) {
    rti_ttTimer_start();

    // read hardware inputs
    PI_Controller_readInputs();
    // calculate the outputs
    PI_Controller_calcOutputs(1U);
    // calculate the state derivations
    PI_Controller_calcDerivations(1U);
    // call the integration method
    PI_Controller_callIntegrator(rti_tickCount);
    // write hardware outputs
    PI_Controller_writeOutputs();

    rti_ttTimer_stop();
    // release the memory protection
    rti_unlock_mem();
    // if xcp is enabled
    #if (LORRA_RTI_Ena_XCP)
    // run the XCP Event for RCP
    xcp_evt(xcp_rcp_evt, xcp_get_daq_time());
    #endif
  }
  else
    rti_error(S3, "RTI Task: Memory locked!");

  /* turn Task to the blocking state until the next sample */
  rti_tickCount += TS_Ms;
  osDelayUntil(rti_tickCount);
}
```

Listing 2: Result of automatic code generation and implementation.

Figure 7 compares the results of the online simulation (measurement) with the results of the MiL simulation (simulation) for a step response to $n_{des} = 100\,1/s$. The curves shown in the upper part agree with high accuracy. This is illustrated by the deviation between both curves plotted in the lower part. The occurring deviations result from the measurement noise and the operating principle of the speed measurement by means of an encoder. The automatically generated code therefore implements the model behavior very well.

7 CONCLUSIONS AND OUTLOOK

This paper presented the LoRra code generator, based on open source software for model based design. Using the seamless model-based RCP development process for the functional development of embedded systems, the general structure of the low-cost development platform LoRra was presented, first. It uses the open-source CAE tool Scilab / Xcos as a central simulation tool. Through the state of knowledge and the analysis of available code generators, it was outlined that currently no model-to-text transformation exists, that is suitable for use in a low-cost RCP tool chain and that is not limited to predefined applications. As a solution, the concept of the LoRra code generator, which transforms Xcos models into flexibly usable C code with standardized interfaces, was presented and realized. A basic verification of the results was carried out using the speed control of a DC motor in a real-time simulation as a case study.

Future work will focus on further verification and analysis of the generated code. The evaluation of various metrics and the comparison with other code generators are the focus here. In addition, an extension

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

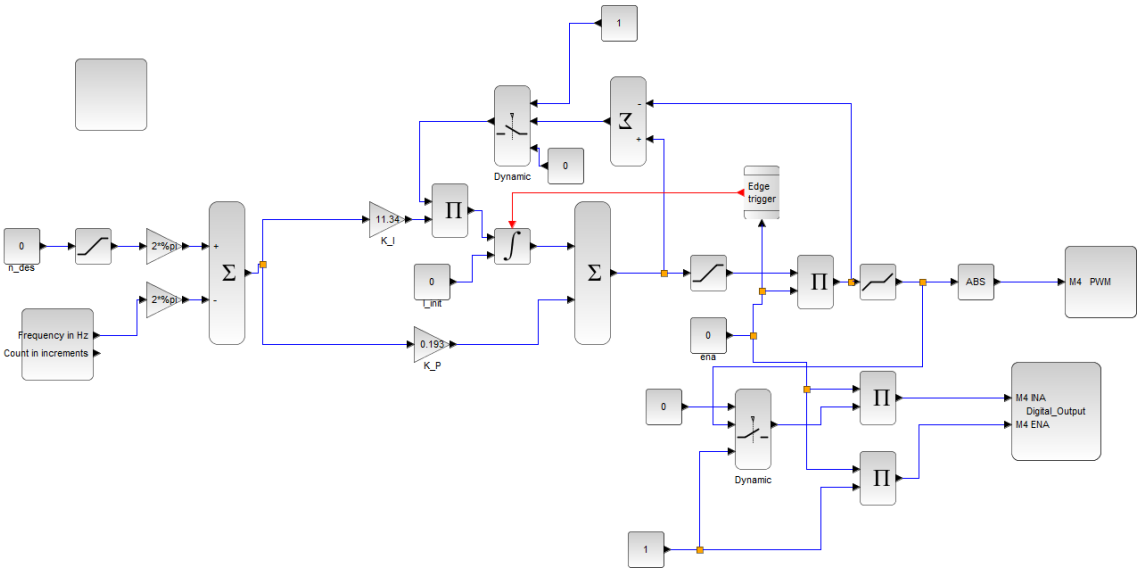


Figure 6: Functional model of the case study PI controller in Xcos.

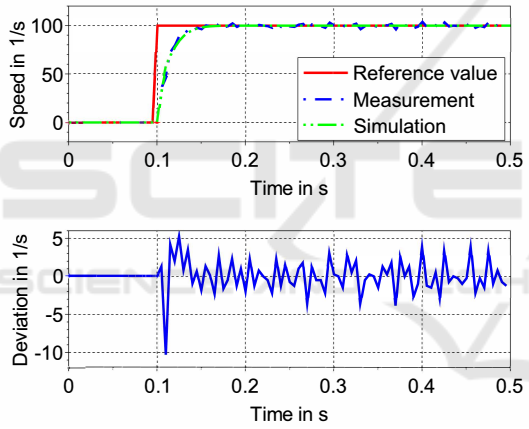
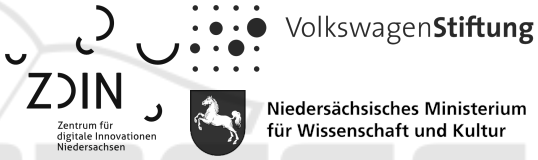


Figure 7: Measured and simulated step response of the PI controlled DC Motor.

of the functional scope is being pushed so that a high number of Xcos blocks is supported.

ACKNOWLEDGEMENTS

Funded by the Lower Saxony Ministry of Science and Culture under grant number ZN3495 within the Lower Saxony "Vorab" of the Volkswagen Foundation and supported by the Center for Digital Innovations (ZDIN).



REFERENCES

Akdur, D., Say, B., and Demirörs, O. (2021). Modeling cultures of the embedded software industry: feedback from the field. *Software & Systems Modeling*, 20(2):447–467.

Bucher, R. and Balemi, S. (2005). Scilab/Scicos and Linux RTAI - a unified approach. In *Proceedings of the 2005 IEEE Conference on Control Applications*, pages 1121–1126, Toronto, Canada.

Campbell, S. L., Chancelier, J.-P., and Nikoukhah, R. (2010). *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer Science+Business Media LLC, New York, NY, 2. ed. edition.

Deppe, M. and Homburg, C. (1999). Rapid Prototyping of Distributed Mechatronic Applications. In *Distributed and Parallel Embedded Systems*, IFIP - The International Federation for Information Processing, pages 203–212, Boston, MA, USA. Springer.

Franco, F. R., Neme, J. H., Santos, M. M., da Rosa, J. N. H., and Dal Fabbro, I. M. (2016). Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU. In *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, pages 869–875, Santa Clara, CA, USA.

Grabmair, G., Mayr, S., Hochwallner, M., and Aigner, M. (2014). Model based control design - A free toolchain. In *2014 European Control Conference (ECC)*, pages 826–831, Strasbourg, France.

Hanselmann, H. (1996). DSP in control: the Total Development Environment. In *Proceedings of 22nd IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 1647–1654, Piscataway, NJ.

Hanselmann, H., Kiffmeier, U., Köster, L., and Meyer, M. (1999). Automatic Generation of Production Quality Code for ECUs. In *Electronic Engine Controls 1999: Sensors, Actuators, and Development Tools*, SAE Technical Paper Series, Warrendale, USA. SAE International.

Jacobitz, S. and Liu-Henke, X. (2019). A Real-Time Interface for Xcos – Demonstrated on a Battery-management System. In *2nd Scilab Conference*, Berlin, Germany.

Jacobitz, S. and Liu-Henke, X. (2020). The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering. In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*, pages 57–64, Valletta, Malta. SCITEPRESS.

Lamberský, V., Križan, J., and Andreev, A. (2014). Generating Code Consistent with Simulink Simulation for Aperiodic Execution on a Target Hardware Powered by a Free RTOS. In *Mechatronics 2013*, pages 95–101, Cham. Springer International Publishing.

Liu-Henke, X., Jacobitz, S., Scherler, S., Göllner, M., Yarom, O., and Zhang, J. (2021). A Holistic Methodology for Model-based Design of Mechatronic Systems in Digitized and Connected System Environments. In *Proceedings of the 16th International Conference on Software Technologies*, pages 215–223. SCITEPRESS - Science and Technology Publications.

Lückel, J., Hestermeyer, T., and Liu-Henke, X. (2001). Generalization of the cascade principle in view of a structured form of mechatronic systems. In *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 123–128, Como, Italy.

Nikoukhah, R. (2006). A hybrid system formalism for modeling and simulation. In *2006 IEEE Conference on Computer Aided Control System Design*, pages 1568–1573, Dearborn, MI, USA.

Nikoukhah, R. and Steer, S. (1996). SCICOS - a dynamic system builder and simulator. In *1996 IEEE International Symposium on Computer-Aided Control System Design*, pages 430–435, Dearborn, MI, USA.

Toeppe, S., Bostic, D., Ranville, S., and Rzemien, K. (1999). Automatic code generation requirements for production automotive powertrain applications. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 200–206, Kohala Coast, HI, USA.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS