# SIMULATION-BASED VERIFICATION FOR PARALLELIZATION OF MODEL-BASED APPLICATIONS

Claus B. Koch

Julius Maximilian University
Am Hubland
Würzburg, Germany
bertram.koch@stud-mail.uni-wuerzburg.de

Umut Durak

German Aerospace Center (DLR)
Lilienthalplatz 7
Braunschweig, Germany
umut.durak@dlr.de

David Müller

German Aerospace Center (DLR)
Lilienthalplatz 7
Braunschweig, Germany
david.mueller@dlr.de

## ABSTRACT

In order to keep up with the increasing volume and challenging requirements of Cyber-Physical Systems (CPS), model-based design and simulation-based verification are becoming essential development practices. This study extends the model-based design approach that is being proposed in the ARGO (WCET-Aware PaRallelization of Model-Based Applications for HeteroGeneOus Parallel Systems) project funded under the European Union's Horizon 2020 Research and Innovation Programme and proposes a complementary simulation-based verification approach. It not only introduces a workflow, but also presents a tooling support, the Scilab/Xcos X-in-the-Loop Toolbox Generator (XTG). Further, the workflow and the Scilab/Xcos XTG are demonstrated with an application use case from the aeronautics domain.

**Keywords:** simulation-based x-in-the-loop verification, cyber-physical multi-core systems.

## 1 INTRODUCTION

Recent Cyber-Physical Systems (CPS) research promotes modeling and simulation-based approaches for system development (Jensen, Chang, and Lee 2011) (Le Sergent, Dormoy, and Le Guennec 2016). The model-based design puts the system models in the center of the development process. Plant and controller modeling are the core activities. The productivity is boosted with the generation of system development artifacts including software code through transformations and stepwise refinement of system models (Atkinson and Kuhne 2003). This step is called code generation. Simulation-based verification advocates executing system models as the native mechanisms of testing (Murphy, Wakefield, and Friedman 2008) (Shokry and Hinchey 2009) (Kapinski, Deshmukh, Jin, Ito, and Butts 2016). Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL) and Hardware-in-the-Loop (HIL), commonly called X-in-the-Loop testing (XIL), are the main steps of simulation-based verification. Modeling and simulation-based

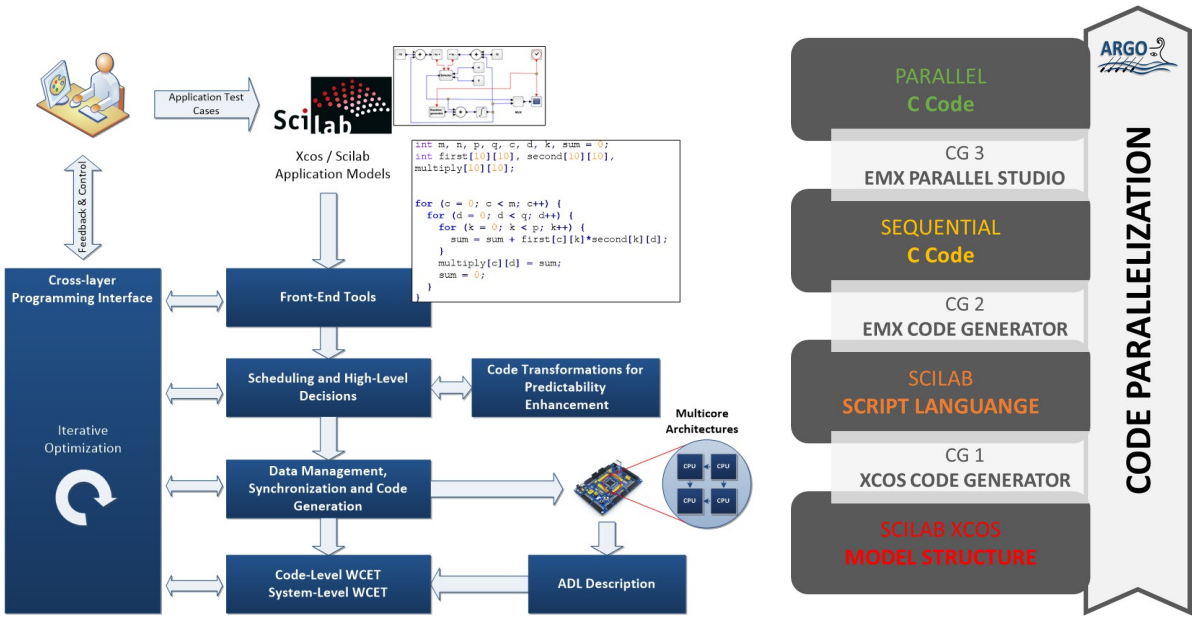


Figure 1: ARGO Tool-flow (left). ARGO Multi-step Code Generation Process (right).

development of parallel real-time software for multi-core processors require new approaches in the code generation and accordingly XIL testing steps.

ARGO (WCET-Aware PaRallelization of Model-Based Applications for HeteroGeneOus Parallel Systems) project that is funded under the European Union's Horizon 2020 Research and Innovation Programme is addressing the code generation and worst case execution time (WCET) aware parallelization of model-based applications for multi-core systems (Derrien et al. 2017) (Durak et al. 2016). In this study, we extend the ARGO model-based design workflow with a simulation-based verification approach. This paper presents a structured XIL testing approach and tooling support, namely Scilab/Xcos XTG (X-in-the-loop Toolbox Generator). It automates the multi-step code generation process, re-injects the code representations to the model instance and generates test scenario templates. Thereby, it supports the workflow by simulation-based verification.

CPS technologies are pronounced as vital for the future of aeronautics. Sampigethaya and Poovendran (2013) claim that the control of an aircraft's electrical, mechanical, structural, thermal and hydraulic components and processes, the entertainment of passengers, the coordination between aircraft and ground stations as well as the coordination between pilots and air traffic highly depend on advances in information and communication technologies. Durak (2018) further claims that the aeronautics domain is on the cusp of the fourth revolution. After realizing far reaching automation levels on aircraft, it is now looking at "smart" and "connected" flight. Therefore, in this endeavor, an application use case from the aeronautics domain is used to design, demonstrate and evaluate the proposed approach.

## 2 MODEL-BASED DESIGN

The Model-Based Design (MBD) is a flavor of model-based development and uses mathematical modeling to design, analyze and evaluate dynamic systems. It is characterized by the seamless use of executable and graphical causal block diagram models and state machines. Model-based design and simulation tools such as Scilab/Xcos (ESI Group 2017) or MATLAB/Simulink (Mathworks 2017) are employed for the system specification, the design and the implementation (Stürmer, Conrad, Fey, and Dörr 2006). Model-based

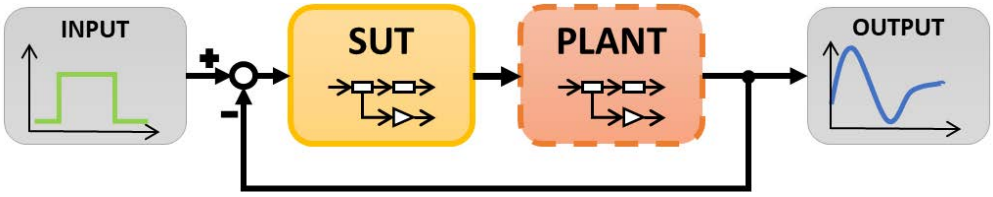


Figure 2: Closed-loop simulation configuration.

development promotes the construction of models and the generation of executable software entities through successive model-to-model and model-to-text transformations (Gaševic, Djuric, and Devedžic 2009). This study applies a novel multi-step model-to-text transformation and application parallelization process. The approach is being development in ARGO and commercialized by the project partner Emmtrix with the Emmtrix Code Generator (Emmtrix 2017a) and Emmtrix Parallel Studio (Emmtrix 2017b). It addresses the automatic worst-case execution time (WCET) aware parallelization of model-based real-time applications for multi- and many-core architectures (Oey et al. 2018). Real-time applications are implemented either in Scilab's high-level mathematical and scientific programming language (Scilab scipts) or using Scilab/Xcos causal block diagrams. The workflow, starting from the first design to the final parallel application, is depicted on the left part of Figure 1. The included multi-step code parallelization process, as shown on the right-hand side of Figure 1, consecutively transforms the model application to parallel C code. It is an automated process that comprises of four algorithm configurations and three code generators (CG). The incrementally numbered code generators transform the different algorithm representations stepwise into each other. The Xcos-Code-Generator (CG1) translates an application from the Scilab/Xcos model to the Scilab script configuration. In the next steps, the algorithm is analyzed, optimized, and transformed by the Emmtrix Code Generator (CG2) into sequential C-Code. In the final step, the algorithm is multi-core optimized and transformed into parallel C-Code by the Emmtrix Parallel Studio (CG3).

## 3 SIMULATION-BASED VERIFICATION APPROACH

### 3.1 Basic Concepts

Cyber-Physical Systems are composed networks of different electronic control units (ECU), sensors and actuators. The software is in close connection to the hardware and networked sensors, actuators, and other ECUs, so it cannot be tested without them. Therefore, the testing process must enclose interaction with other components and the physical environment. Simulation-based testing utilizes component and environment virtualization. This allows, compared to the real hardware system, for a straightforward and direct signal manipulation, and a behavior monitoring.

Control systems are classified as open-loop and closed-loop configurations. These configurations have reflections to simulation-based based software verification. In a closed-loop configuration, the System Under Test (SUT) is coupled to the plant model providing the feedback throughout the test execution. A closed-loop setup is sketched in Figure 2. The open-loop configuration, similar to a conventional unit test (UT), applies stimuli to the input of the system under test, and monitors the processed output. Because of its straightforward setup, the open-loop configuration is well applicable to single unit testing for verification purposes.

Kapinski et al. (2016) describe verification and falsification in the following way. Elements in an analysis configuration are the system under test $M$, a possible infinite set of parameters $P$, a possible infinite set of inputs $U$, and the property $\Psi$ that a system should hold for. Verification and falsification are defined in terms of behaviors $(M, p, u)$, where $p \in P, u \in U$, while $u$ is a function of time. The behavior of the system $M$ under the parameter $p$ and the input $u$ is denoted as $(M, p, u)$. $(M, P, U)$ describes the set of all possible

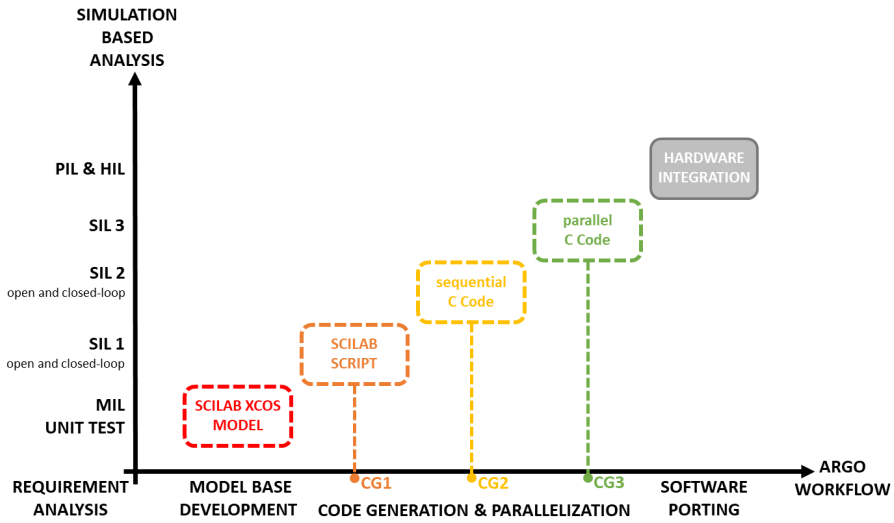


Figure 3: Code generation and simulation-based verification.

behaviors of $M$ under the parameters in $P$ and inputs in $U$. The evaluation of $M$ determines whether $\Psi$ holds for a particular $p$ and $u$. The notation $(M,p,u) \models \Psi$ describes that a system satisfies the conditions.

The aim of *verification* is to prove that a system holds for a defined set of parameters $P$ and inputs $U$ ($(M,p,u) \models \Psi$). This technique analyses the correctness of a system that has infinite value possibilities by a subset of parameters. It is an approach applicable to an automated and formal testing environment. In case a verification case fails, a diagnosis must be done. The error sources must be constrained to find the incorrectness.

*Falsification* on the other hand searches for $(M,p,u) \not\models \Psi$. It intends to find a set of parameters that a system processes in a wrong way. Falsification provides information to developers that are valuable for isolation and debugging.

Having defined the basic concepts, we promote closed-loop tests for the verification purposes and open-loop tests for further analysis, isolation and debugging.

### 3.2 X-in-the-Loop Testing

The XIL testing describes different configurations for a simulation-based verification. The XIL refers to MIL, SIL, PIL, and HIL configurations. This modular process allows to place the focus on the verification and the critical aspects of a specific configuration. The XIL setup provides a flexible system and supports a wide range of testing strategies. Over the software development life cycle, different configurations of an application are simulated and verified. XIL runtime environments provide interfaces to the system and configuration under test. The idea is to embed and simulate the SUT in a flexible and modifiable virtual environment representation. Input vectors simulate data signals to an algorithm or a system and the processed output values are monitored. The encapsulation of the application under development and the application analysis process come with a fluent testing process that reuses testing scenarios for different code configurations. A unique feature of XIL tests is that the same stimulus can be applied to different configurations and the behavior is directly comparable. XIL testing supports model-based design and automatic code generation. The multi-step approach of ARGO for the parallelization of model-based applications requires a reconsideration of the verification process. The automated code generation undergoes the process as dis-

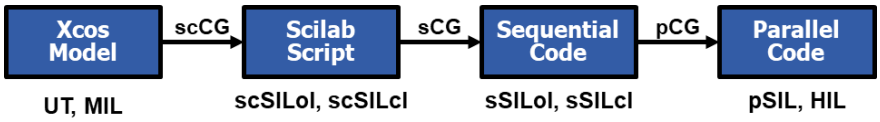


Figure 4: ARGO's code generation and verification steps.

cussed previously. As depicted on the x-axis of Figure 3, the Scilab/Xcos model is transformed into three distinct artifacts throughout the parallelization workflow. The development and the workflow is assigned to XIL testing. The consecutive steps of this process are plotted on the y-axis. Durak, Müller, Möcke, and Koch (2018) propose a series of XIL tests for each generation step that verifies the conformance of the generated code with the source model. The implementation of modules and the simulation-based testing in this development is done in Scilab scripts and using Scilab/Xcos.

It is recommended that the units of the overall model are verified during the Unit Tests (UT) in a white box open-loop fashion. At the MIL stage the overall SUT is verified against its requirements. Open-loop unit tests and closed-loop scenario simulations are conducted. The plausibility of the controller is tested by the execution of a number of operational scenarios. Ideally, applications are tested as a black box, independent of the software internals and without specific knowledge about the actual implementation. The results of unit testing and MIL simulations establish the reference for the subsequent XIL tests. SIL tests execute the generated code instance within a dedicated test loop. It verifies the functionalities of the target and is essentially done in non-real-time. Open-loop and closed-loop SIL tests are executed in a similar fashion as MIL tests. The significant difference is that the model implementations are substituted by the corresponding software instances. The generated code is reintegrated into the simulation environment and the schema of the test setup. There is a consecutive series of SIL tests. It is structured into the different software instances and the configurations simulation setup. Durak et al. (2018) proposes the following steps for the SIL tests which are corresponding to ARGO's multi-step parallelization workflow:

**SIL1:** At the first SIL testing step, Scilab script code is simulated. It is the outcome of the first code generation process and the model-to-text transformation. This SIL test verifies that the generated scripts perform computations in the same way as the Xcos model elements. Scilab script open-loop (scSILol) tests are performed on the subsystems. These tests are crucial for the isolation and localization of misinterpretations in the model-to-text transformation. Scilab script closed-loop (scSILcl) tests are done in a simulation setup of a plant representation and a controller in Scilab script configuration. Even though the controller model instance is substituted by a software instance, the closed-loop simulation setup of the SUT and the plant remains unchanged. It is expected that calculations are performed congruently to the one of the application in the model domain.

**SIL2:** The second SIL testing step focuses on the sequential C code. It is generated by a text-to-text transition of the Scilab script code. The open-loop and closed-loop simulations are done similarly to the first SIL step. The configuration of the system under test is substituted by a C code instance. The processed outputs are compared to the reference data of the MIL simulations. The submodules of the application are verified separately by unit testing (sSILcl). The sequential C code representation of the entire system is simulated and verified by closed-loop scenario tests.

**SIL3:** At the third SIL step, the application in parallel C code is verified. It is generated by the third code generator. The parallelized C code of the entire system is simulated and the results are again compared with the reference data of the MIL tests.

**HIL:** HIL simulations are performed in a real-time setup with an application deployed to the target platform. It enables the requirement verifications in a similar way as previous XIL simulation tests. Furthermore, it allows to test for non-functional requirements such as real-time performance. Closed-loop scenario tests are

proposed to verify that the processed outcome of the parallel application on the multi-core target platform complies with that of the overall Scilab/Xcos model.

### 3.3 Automated Generation of XIL Test Setups

This research supports the discussed XIL testing approach by a Scilab extension that generates a toolbox for XIL testing. The toolbox elements are scenario file templates, system interfaces and the model blocks in all code configurations as shown in Figure 4. The tool is called Scilab/Xcos XTG. It automatically generates a case specific XIL toolbox for the SUT. Such a toolbox is denoted XLT (X-in-the-Loop Toolbox). An XLT unifies the different configurations of the multi-step code generation process by re-injecting all source code instances into model blocks. These code model blocks become elements of the toolbox in the simulation environment, Scilab/Xcos. They are then used in the model-based workflow, analog to standard model blocks. Since the build process is automatically done by the Scilab/Xcos XTG, the source code does not need to be manually implemented and compiled. The code instances can also be executed like conventional model blocks. The Scilab/Xcos XTG also builds case specific input and output blocks. The input block applies test stimuli and the output block reads the processed values. The Scilab/Xcos XTG reduces the testing effort, unifies the process, and allows to apply identical test cases to all configurations of the multi-step development approach.

The automated build process done by the Scilab/Xcos XTG is depicted in Figure 5. It applies the code generation steps of the ARGO workflow. The application in the model domain is analyzed and based on the super-block (a model instance which encapsulates smaller units), the multi-step code generator is executed. In this first stage, the Scilab/Xcos XTG also compares the model algorithm to the function files of the different code instances. In the XIL approach, it is assumed that the code configuration is functionally identical. The interface data in the scenario and the algorithm files of the different software configurations are listed. The model interfaces and the input and output variables of the software domain are expected to be similar. In a comparison step, the parameters and variables are matched. The interface parameters of the model are used as the reference for the other domains. The Scilab/Xcos XTG indicates inconsistencies in the variables of the different configurations. The data of the interfaces respective variables gained in this analysis process are used in further steps. They are applied to define the XLT elements and are the foundation for further processing steps.

In the data matching and system building stage, the components of the new XLT are defined. Components are the model blocks, the interface blocks for system tests and the templates of the test scenarios. The components are adapted and implemented specifically for the SUT. The required information for the automatic implementation process is obtained in the previous stage. It is further matched, preprocessed, and used to build the components of an XLT. An XLT is structured as a folder that contains the elements of the blocks and the test scenario templates. Scilab builds and executes the algorithm files of the XLT. This build process generates and extends the Scilab/Xcos toolbox palette by the new+ elements of the XLT. These are the source-code-model elements and interface elements for the test scenario execution. An XLT enables the execution of verification and validation tests in all instances of the XIL testing process. The code blocks execute the different configuration instances of the SUT. The interface blocks interconnect the SUT and the rest.

An XLT provides the elements for an efficient and structured testing procedure. It is automatically generated for a specific application that is developed as a Scilab/Xcos model. A toolbox contains source code model blocks, simulation interface blocks and test scenario templates. These are the necessary elements to execute test scenarios.

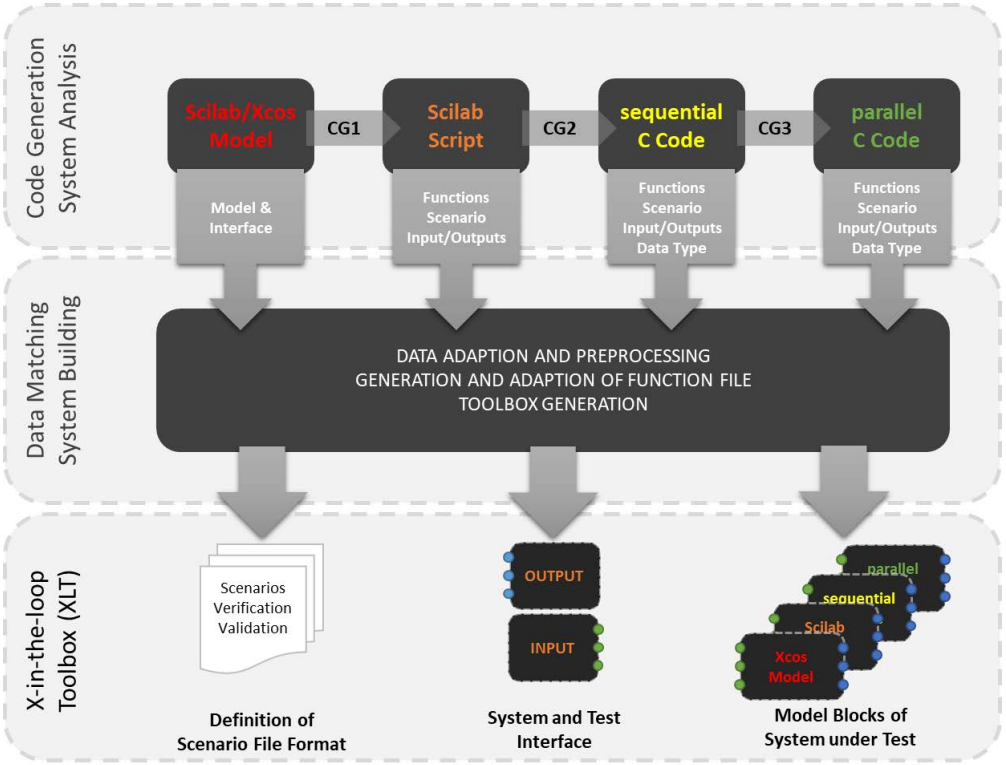


Figure 5: Test setup automation with Scilab/Xcos XTG.

A *test scenario template* is a structure file that consists of information about the test case, the system under test, the interface elements, the execution of simulation, and the simulation data. The structure is written in a general manner that can be adapted for a specific system under test. It is rather agile regarding the verification or validation strategy.

The *system and test interface* is the connection between the simulation environment Scilab/Xcos and the SUT. It is implemented as an input and output model block and applicable in the same way as a standard model block. The interface is the execution, simulation, and data analysis element in all x-in-the-loop configurations. An input block is the element that provides a test vector to the system under test. It loads a test scenario file, does the pre-configuration of the system under test and changes the input vector depending on the time. The output block is the part that reads the processed values of the system under test. The data is stored with a time stamp of the simulation. At the end of a simulation scenario, the output block processes the simulation data and the values are compared to the expected values of the scenario. A report of the simulated scenario is created by the output block. It is either done as a written text-based report (JSON), or a web-based table (html), or a plot of the simulated data.

## 4 TESTING AN ENHANCED GROUND PROXIMITY WARNING SYSTEM WITH SCILAB/XCOS XGT

Controlled Flight Into Terrain (CFIT) type accidents were responsible for many fatalities in civil aviation until the Federal Aviation Administration (FAA) made Terrain Awareness and Warning Systems (TAWS) mandatory for all turbine-powered passenger aircraft registered in the U.S. (Federal Aviation Administration 2000). There are various TAWS options available on the market that provide a set of features which aim to counter risks of CFIT. T3CAS from ACSS (2018), LANDMARK$^{TM}$ from L3 (2018) and TAWS from

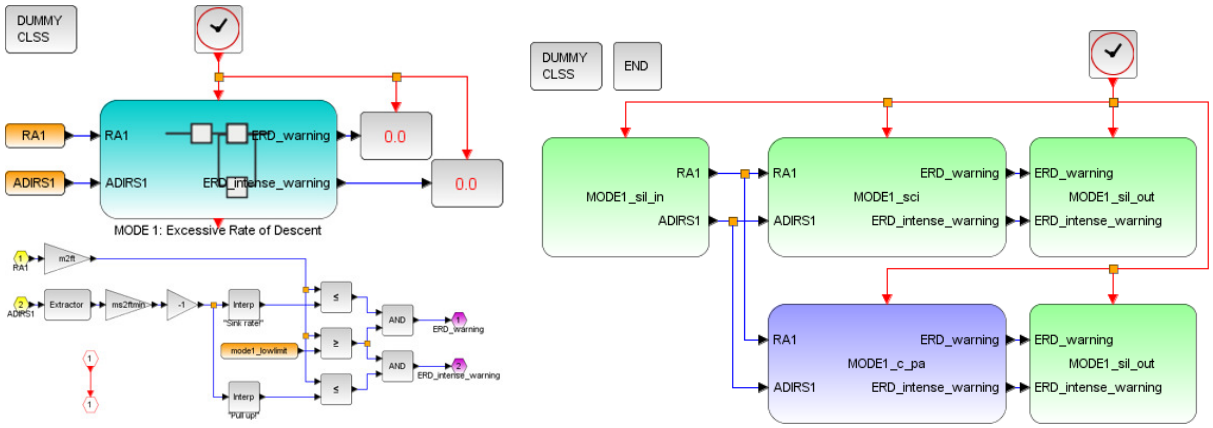


Figure 6: A model component from the ARGO EGPWS (bottom) (left) and a SIL setup of the model on the left side generated by the Scilab/Xcos XTG.

Universal Avionics (2018) are some examples. A brief comparison of these systems can be found in (Smith 2005).

The ARGO Enhanced Ground Proximity Warning (EGPWS) is a TAWS based on Honeywell's EGPWS Honeywell (2018) that is being used in the German Aerospace Center's (DLR) Airbus A320 Advanced Technology Research Aircraft (ATRA). The core feature set of an EGPWS is to create visual and aural warnings between 30 ft to 2450 ft Above Ground Level (AGL) in order to reduce the risk of CFIT. It has five different modes. In the case of multiple warnings, a data output management logic decides which warning to trigger in the cockpit based on a priority table. The ARGO EGPWS is implemented as a Scilab/Xcos model. We designed, applied and evaluated the proposed approach and the tool with this application use case.

ARGO's multi-step code generation process transforms the model application to parallel C code. As part of this study the autogenerated source code instances of ARGO's code generator are executed and analyzed in SIL tests. The process applies the proposed approach and the Scilab/Xcos XTG. Figure 6 shows an example of different configurations of Mode 1 "Excessive Rate of Descent", a subsystem of the ARGO EGPWS. The left hand side depicts a model implementation. At the top is the superblock interfacing the signals to other systems and the bottom part gives an impression about the inner functions of the super block. The right hand side shows an open-loop verification setup of elements automatically generated by the Scilab/Xcos XTG. It implements two code configurations of the ARGO EGPWS Mode 1 for comparison tests. The code configurations are simulated by an identical test scenario. The processed outputs of the two SIL configurations are compared to a previously generated database of the model configuration. Variations indicate misinterpretations in the code configuration process. This study applies this method to identify flaws in the model-to-code parallelization process.

## 5 CONCLUSION

There is an increasing interest in multi-core architectures as target platforms for Cyber-Physical Systems. Although these multi-core targets promise high computational performance, there are still many challenges to be tackled. The engineering community is looking for effective and efficient methodologies for multi-core application development. In this paper a simulation-based verification approach for multi-core platforms is presented. The parallelization workflow for model-based design that is being developed within the scope of the ARGO (WCET-Aware PaRallelization of Model-Based Applications for HeteroGeneOus Parallel Systems) project funded under the European Union's Horizon 2020 Research and Innovation Programme is

augmented by an X-in-the-Loop pipeline that produces verification evidences for every step of the parallelization process. The approach is further supported by a tool, the Scilab/Xcos X-in-the-Loop Toolbox Generator (XTG) for automated generation of the consecutive test setups. The proposed approach is exemplified with a use case from one of the major Cyber-Physical System domains, aeronautics. It is applied to the simulation-based verification of the ARGO Enhanced Ground Proximity Warning System. The future work includes creating further use cases and promoting the approach and tool in the Scilab/Xcos community.

## ACKNOWLEDGMENTS

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No 688131 - ARGO.Action. www.argo-project.eu

## REFERENCES

ACSS 2018. "T3CAS". www.l3aviationproducts.com/products/t3cas/. Accessed Jan. 02, 2018.

Atkinson, C., and T. Kuhne. 2003. "Model-driven development: a metamodeling foundation". *IEEE software* vol. 20 (5), pp. 36–41.

Derrien, S., I. Puaut, P. Alefragis, M. Bednara, H. Bucher, C. David, Y. Debray, U. Durak, I. Fassi, C. Ferdinand et al. 2017. "WCET-aware parallelization of model-based applications for multi-cores: The ARGO approach". In *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 286–289. European Design and Automation Association.

Durak, U. 2018. "Flight 4.0: The Changing Technology Landscape of Aeronautics". In *Advances in Aeronautical Informatics: Technologies Towards Flight 4.0*, edited by U. Durak, J. Becker, S. Hartmann, and N. S. Voros. Springer.

Durak, U., D. Müller, J. Becker, N. S. Voros, P. Alefragis, T. Stripf, P.-A. Agnel, G. Rauwerda, and K. Sunesen. 2016. "Model-based Development of Enhanced Ground Proximity Warning System for Heterogeneous Multi-Core Architectures". In *2016 AIAA Modeling and Simulation Technologies Conference*.

Durak, U., D. Müller, F. Möcke, and C. B. Koch. 2018. "Modeling and Simulation-based Development of an Enhanced Ground Proximity Warning System for Multicore Targets".

Emmtrix 2017a. "Emmtrix Code Generator (eCG)". Retrieved from https://www.emmtrix.com/tools/emmtrix-code-generator. (Accessed on November 10, 2017).

Emmtrix 2017b. "Emmtrix Parallel Studio (eCG)". Retrieved from https://www.emmtrix.com/tools/emmtrix-parallel-studio. (Accessed on November 10, 2017).

ESI Group 2017. "Scilab.io". Retrieved from https://www.scilab.io. (Accessed on November 10, 2017).

Federal Aviation Administration 2000. "Installation of Terrain Awareness and Warning Systems (TAWS) Approved for Part 23 Airplanes".

Gaševic, D., D. Djuric, and V. Devedžic. 2009. *Model driven engineering and ontology development*. Springer Science & Business Media.

Honeywell 2018. "MK VI and MK VIII Enhanced Ground Proximity Warning System (EGPWS) Pilot's Guide". www.aerocontent.honeywell.com/aero/common/documents/Mk_VI_VIII_EGPWS.pdf. Accessed Jan. 02, 2018.

Jensen, J. C., D. H. Chang, and E. A. Lee. 2011. "A model-based design methodology for cyber-physical systems". In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pp. 1666–1671. IEEE.

Kapinski, J., J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. 2016. "Simulation-based approaches for verification of embedded control systems: an overview of traditional and advanced modeling, testing, and verification techniques". *IEEE Control Systems* vol. 36 (6), pp. 45–64.

L3 2018. "LandMark Terrain Awareness & Warning Systems". www.l3aviationproducts.com/products/landmark. Accessed Jan. 02, 2018.

Le Sergent, T., F.-X. Dormoy, and A. Le Guennec. 2016. "Benefits of Model Based System Engineering for Avionics Systems". In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

Mathworks 2017. "Simulink". Retrieved from https://www.mathworks.com/products/simulink.html. (Accessed on November 10, 2017).

Murphy, B., A. Wakefield, and J. Friedman. 2008. "Best practices for verification, validation, and test in model-based design". Technical report, SAE Technical Paper.

Oey, O., M. Rückauer, T. Stripf, J. Becker, C. David, Y. Debray, D. Müller, U. Durak, E. K. Kasnakli, M. Bednara et al. 2018. "Interactive Parallelization of Embedded Real-Time Applications Starting from Open-Source Scilab & Xcos". In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*.

Sampigethaya, K., and R. Poovendran. 2013. "Aviation cyber–physical systems: Foundations for future aircraft and air transport". *Proceedings of the IEEE* vol. 101 (8), pp. 1834–1855.

Shokry, H., and M. Hinchey. 2009. "Model-based verification of embedded software". *IEEE Computer*.

Smith, D. 2005. "Traffic Alert Collision Avoidance Systems—TCAS Buyer's Guide". *Pilot's Guide to Avionics*, pp. 34–41.

Stürmer, I., M. Conrad, I. Fey, and H. Dörr. 2006. "Experiences with model and autocode reviews in model-based software development". In *Proceedings of the 2006 international workshop on Software engineering for automotive systems*, pp. 45–52. ACM.

Universal Avionics 2018. "TAWS | Terrain Awareness and Warning System". www.uasc.com/home/shop/avionics/taws. Accessed Jan. 02, 2018.

## AUTHOR BIOGRAPHIES

**CLAUS B. KOCH** is a student in the Joint European Master in Space Science and Technology and a Research Assistant at the Institute of Flight Systems of DLR. He holds a B.Sc. in Aerospace Informatics of the Julius-Maximilians-Universität Würzburg. His research interests lie in the software engineering field for robotics, avionics and astronautics. His email address is bertram.koch@stud-mail.uni-wuerzburg.de.

**UMUT DURAK** is a Research Scientist at the Institute of Flight Systems of the German Aerospace Center (DLR) and an adjunct faculty (Privatdozent) at the Department of Informatics of the Clausthal University of Technology. His research interests lie in modeling and simulation-based approaches in aeronautics. His email address is umut.durak@dlr.de.

**DAVID MÜLLER** is a Research Scientist at the Institute of Flight Systems of DLR. He received his M. Sc. in Aerospace Engineering from the Technical University Braunschweig. His research interest lies in the improvement of pilot-aircraft-interactions by way of assistant systems. His email address is david.mueller@dlr.de.