Model-based Automation of Verification Development for automotive SOCs

Aljoscha Kirchner and Jan-Hendrik Oetjens
Engineering Integrated Circuits Methods, Tool & Technologies
Robert Bosch GmbH
Reutlingen, Tübinger Straße 123, Germany
{Aljoscha.Kirchner & Jan-Hendrik.Oetjens}@de.bosch.com

Oliver Bringmann
Wilhelm Schickard Institut für Informatik
Lehrstuhl Eingebettete Systeme
Eberhard Karls Universität Tübingen
Tübingen, Sand 13, Germany
Oliver.Bringmann@uni-tuebingen.de

Abstract— **Current technical trends in the automotive industry lead to a demand for more complex and at the same time secure systems, also in the area of SoC development. This is in contrast to the goal of achieving ever shorter and more efficient development cycles. These challenges are particularly evident in verification, which takes up a considerable part of system development due to increasing requirements. To address these challenges this paper presents a novel method for model-driven automation of verification. The method includes the formalization and modeling of the SoC specification using SysML as well as the generation of SystemVerilog Assertions based on the modeled specification. Thus the effort can be achieved by minimizing manual transformation as well as by reducing errors due to ambiguous specifications.**

Keywords—SysML, Model-Driven, Verification, Verilog Assertions

I. Introduction

Current trends in the automotive industry, such as autonomous driving and digitalization, are leading to an increasing demand for more powerful and reliable automotive System-on-Chips (SoCs). The resulting increase in the complexity of SoCs causes growing challenges in the field of verification.

There are two major issues:

- Up to 50% of project resources are currently required for functional design verification [1].
- More than 50% of the designs contain functional errors and thus lead to necessary rework [1]. This often results from an incomplete specification because unspecified parts of the design cannot be verified.

Verification is an integral part of SoC development; in some cases, the effort for the verification of complex systems exceeds the effort of the design [2]. Therefore, the above-mentioned deficits have a negative impact on productivity. The complexity of SoCs increases by about 58% per year, but productivity only by about 21%. The gap between productivity and growth rate is called the "productivity gap". To close this gap and increase the productivity of verification, new methods for specification and verification are needed [1] [3].

A. Related Work

The literature offers various solutions for formalizing specifications as well as approaches for automating verification. To the best of the author's knowledge, there is currently no existing solution for model-driven automation of verification based on a formalized and modeled specification in the area of automotive SoCs. Therefore, in the following a consideration of solutions from the literature is made, which pursue comparable approaches of the present work.

The aim of the method described in [4] is to increase the efficiency of verification and to formalize the specification. The Projection Temporal Logic (PTL) is used to describe the system in order to specify the behavior of the system as discrete state sequences. Based on this, the author uses the programming language Tempura to achieve an executability of the specification. The executability of the specification facilitates the verification and errors can be detected and corrected even before the verification. However, the specification of the system by means of temporal logic offers a unique description form but is difficult to read, especially without special knowledge.

In [5], [6], [7], [7] and [9] parts of the SysML are used for a formalization-approach of the specification as well as the support of the verification [10]. By formalizing and modeling the specification, the clarity and comprehensibility of the specification can be increased. However, this approach has only an indirect influence on the reduction of the verification effort without offering a solution for the automation of the verification process.

The contribution [11] enables a semi-formal description of the specification in tabular form, which on the one hand increases the unambiguity of the specification and on the other hand facilitates the validation of the specification. Based on this, the generation of SystemVerilog properties for verification based on the tabular specification is made possible. A comparable approach is offered in [12]. Here the generation of SystemVerilog Assertions based on processor architecture descriptions or models is enabled. In addition, the paper aim the formalizing of the specification and thus address the reduction of inconsistencies and implementation errors. The works from the literature shown here enable the generation of SystemVerilog Assertions. However, non-standardized formats are used for the description, which means that a general usability and comprehensibility of the description can only be given to a limited extent.

The contribution presented here is part of the method already presented in earlier papers [13], [14] and [15]. The overall method enables a model-based and formalized specification for the development of SoCs and based on this, implements various solutions for model-driven code generation.

B. *Contributions*

This paper presents a novel method for model-based automation and thus for reducing the verification effort. Through modeling and formalization a machine-readable format is created for the specification of the SoC. Thus, the transfer of information required in the current procedure for design and verification from the specification into a machine-readable format becomes unnecessary. The modeling of the system to be developed also increases the unambiguousness of the specification and consequently reduces the probability of misinterpretations during implementation as well as during the development of the verification. Therefore, the method presented here can not only achieve the effort of verification by automation it also enables a reduction of the effort for fixing errors due to an insufficient specification.

This paper is structured as follows. Section II gives a brief overview of the basic concepts. Section III explains the proposed method for modelling the architecture using SysML and based on this, the automation of verification. Section V discusses the application of the method to an industrial example of a pressure sensor SoC and section 5 finally contains the conclusion of this paper.

II. Basic Concepts

Verification is an indispensable part of the design of hardware/software systems, such as automotive SoCs. Therefore, both verification and validation are required in every part of the ISO 26262 industry standard and are essential to ensure the highest level of functional safety and quality for a product or process [16]. The verification proves that a system has been designed correctly with respect to its specification. Thus, it can be proven that the design of the system fulfills the functional as well as parts of the non-functional requirements of the specification.

During developing and designing a system, in addition to the usual typos and implementation errors, an ambiguous specification in particular poses a risk of errors. For example, a lack of clarity of system requirements, which can arise from the description in natural language, can lead to the developer being forced to interpret them during system design. As a result, a separate “picture of the system” is created in the developer’s mind, which no longer corresponds to the specification. However, if the developer implements on the basis of his own “picture of the system”, design errors occur because the design does not correspond to the specification. To minimize the risk of an incorrect implementation due to interpretation, it is important that another person than the developer involved in the design performs the verification of the system to fulfil the "four-eyes principle".

In addition to the lack of clarity in the specification, an incomplete specification poses particularly serious risks, because scenarios that may occur during the use of the system are not specified and therefore not verified. However, if there are errors in the specification or if the specification is incomplete, the design is still considered correct for the purposes of verification. There is thus a risk of an unpredictable state or behavior and in the worst case, the risk of endangering human life. Specification errors must therefore be minimized by a comprehensive validation of the specification.

Besides the already mentioned problem of interpretations during the design of a system due to an ambiguous specification, the same problem applies to verification. If the specification is ambiguous, the verification engineer is forced to plan and develop the verification on the basis of interpretations and even in this case, a separate picture of the system is created in the mind of the verification engineer [17]. In addition, there is a risk that the verification engineer will get the missing information of the specification by looking into the system design and thus the verification based on the implemented design is created. However, such a procedure would normally invalidate the verification, since the design is no longer checked against the specification but against itself, which makes it impossible to detect errors in the design using verification. To address the above mentioned challenges in SoC development and verification the method presented here combines the following two approaches:

A. *SysML-based Specification*

The quality and completeness of the specification is one of the decisive factors in the successful development of complex systems. It is difficult to check the completeness and correctness of the specifications, which are mostly written in natural language today. On the one hand, this is due to the enormous volume of 1000-2000 pages of such specifications, on the other hand, the description in natural language is prone to errors and complex relationships described therein are difficult to detect. To solve this problem, developers often use block diagrams and behavioral diagrams for visualization, which are created using a drawing tool or whiteboards. This offers a remedy in the first moment and serves as a basis for discussions, but has further deficiencies and disadvantages:

- Risk of inconsistencies due to insufficient maintenance after the creation
- Lack of standardization, developer draws at his own discretion
- Lack of machine readability, risk of double work
- Susceptibility to errors due to lack of unambiguousness of natural language

There is therefore still a risk that the specification will be incomplete and that reworks will be necessary. Nevertheless, as shown in [18], 79% of the requirements of today's specifications are specified using interpretable, difficult to validate and error-prone natural language. To further reduce these risks, the SysML-based modeling method for formalizing the specification was presented in the previous works [13], [14] and [15]. The use of SysML enables a machine-readable and standardized format for the specification of SoCs. This makes it possible to increase the completeness and unambiguity of the specification and thus minimize the risks of interpretability described in the previous section. In addition, the machine readability of the models offer opportunities for automation.

B. *Model-based Automation of Verification*

Due to the increasing complexity of today's SoCs, the planning, development and execution of verifications leads to considerable cost and time expenditures and limits the efficiency of SoC development according to current methods. Therefore, as part of the model-based method developed in this paper, possibilities of model-driven automation of verification shall be considered.

Besides a considerable reduction of effort, the automation of verification and the resulting increase in efficiency has further decisive advantages. For example, the probability of errors is considerably minimized by automating the transfer or transformation of the information from the specification to the verification. As described, a manual transfer always bears the risk of simple typing and implementation errors during the development of the verification, in addition to the risk of incorrect interpretations.

III. Model-based Automation of Verification

In order to achieve an automation in the area of verification, this paper describes an extension of the method to enable the model-based generation of connectivity checks. As with the manual creation of the verification, the "four-eyes principle" must also apply to the automation of the verification. If parts of the design as well as parts of the verification are to be generated based on the modelled specification, it is possible that the generated part of the design is verified by the generated part of the verification. This increases the risk of undetected so-called Common Cause Failure. However, the "four-eye principle" is fulfilled for automation if two completely independently developed and working generators or generation methods are used for design and verification. By using two independent generators, one obtains the highest probability that not the same failure that occurred during the generation of the design will also occur during the generation of the verification. Furthermore, it must be noted that a generator usually falls back on a predefined mapping between model and program code. This mapping should always be defined independently for the respective generator [19].

To ensure the "four-eyes principle" also in this work, an independent generation flow for the automation of the verification was developed. The architecture model from the specification serves as the basis for generation. The method shown here enables the generation of connectivity checks as part of the verification based on the architecture diagram. The architecture diagram is already included in the modelled specification and can be used for the generation without additional modeling effort. Connectivity checks verify the correct "wiring" of the SoC in the design according to the specification. To generate the connectivity checks, information about the connections between modules is read from the architecture diagram in a script-based manner and translated into SystemVerilog Assertions. Using the widely known description language SystemVerilog, parts of the verification can be described. An Assertion is a check of the design against the specification. If the specification is violated, an error is thrown [20] [21]. In the method presented here, the assertions are generated in two steps, as shown in Figure 1.

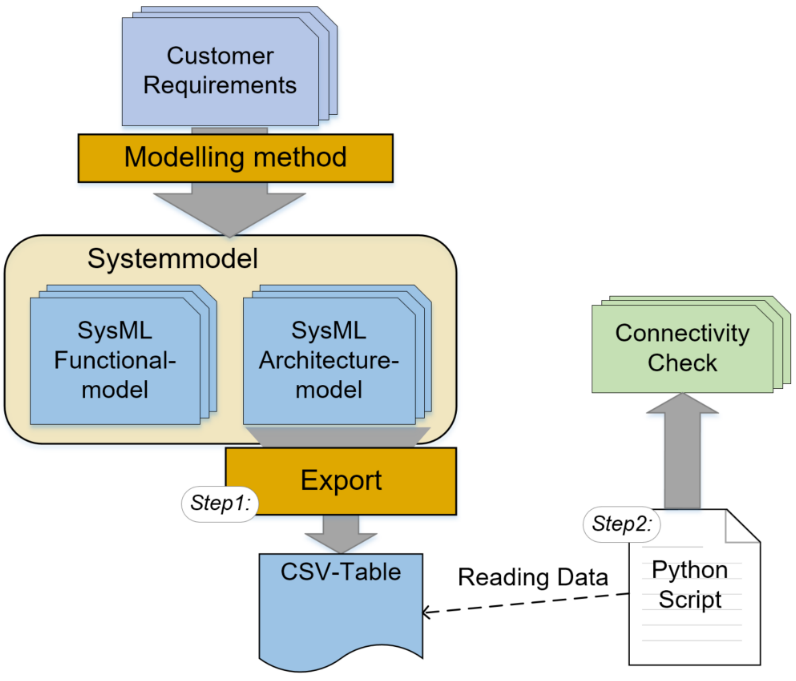


Figure 1: Method for model-based generation of Connectivity Check

In *Step1*, a script extracts the information from the architectural model and stores them into a table. In this process, an end port must be found for each start port. Ports located between these ports will be ignored. To make this possible, a script was implemented, which searches for the respective end port in the architecture diagram for each starting port. Another script is required to transfer this information to the table. In the model, each model element has its own ID. This ID of the end port is automatically cached in a *Tag*. Afterwards, the name of the corresponding part and its hierarchical order in the model is extracted based on the ID of the end port and entered into a table. In the next *Step2*, the information of the table will be read out by a Python script [22] and generated based on the port variables SystemVerilog Assertions. For this purpose, the special characters contained in the table are removed. Afterwards, the different entries of the table columns, both for the start and for the end port, are combined to string variables [23]. For this, the naming convention and notation rules of the names of the SysML model elements had to be converted to the naming convention and notation rules of VHDL.

IV. CASE STUDY ON AN EXAMPLE

This section demonstrates, on the basis of an industrial example, the application of the method with regard to modelling and generation in the field of connectivity checks. The example used here represents a subsystem of an inertial sensor SoC. The architecture of this example is shown hierarchically as a BDD in Figure 2 and as an *Architecture Diagram* in the IBD in Figure 3.

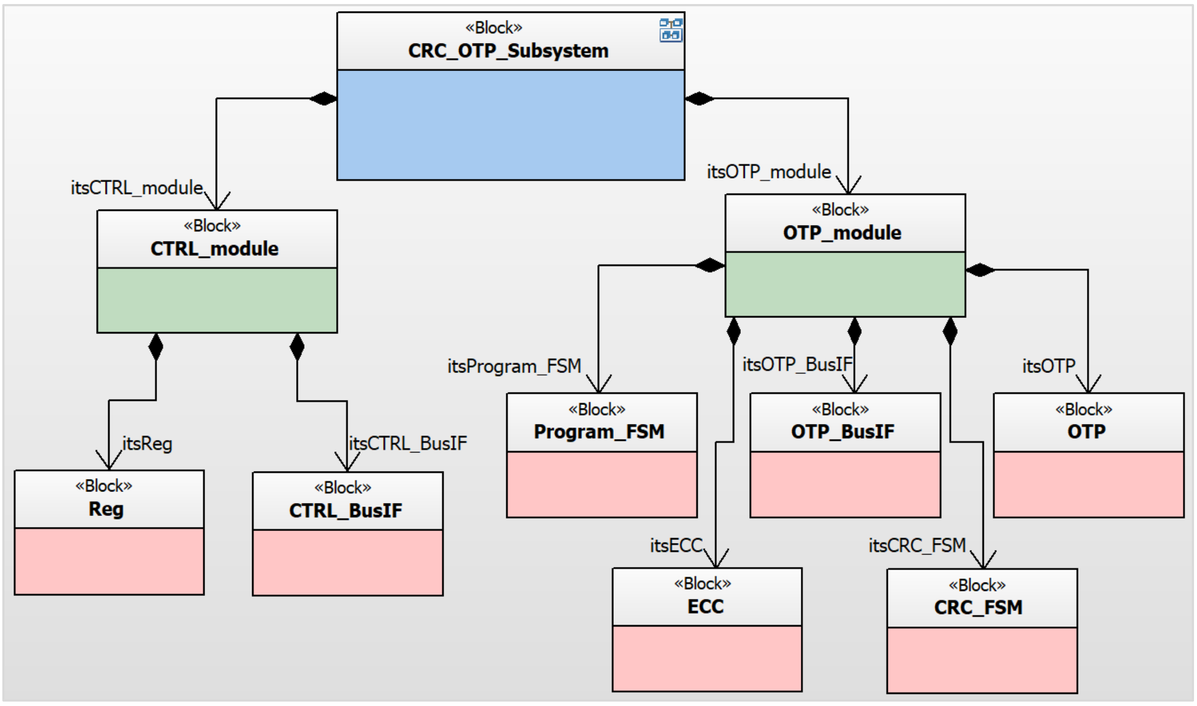


Figure 2: Example Architecture BDD Connectivity Check

The hierarchically decomposed system *CRC_OTP_Subsystem*, shown in Figure 2, consists of two components called *CTRL_module* and *OTP_module*. The component *CTRL_module* in turn consists of the sub-components *Reg* and *CTRL_BusIF*, the component *OTP_module* of the sub-components *Program_FSM*, *OTP_BusIF*, *ECC*, *CRC_FSM* and *OTP*. The key components of this subsystem are the *CTRL_module*, which is needed for the boot loading of the onChip software and the One-Time-Programmable Memory (*OTP*) where the initial values of the SoC are stored. The color scheme used here serves only to illustrate the hierarchies and is also applied to the architecture diagram in Figure 3.

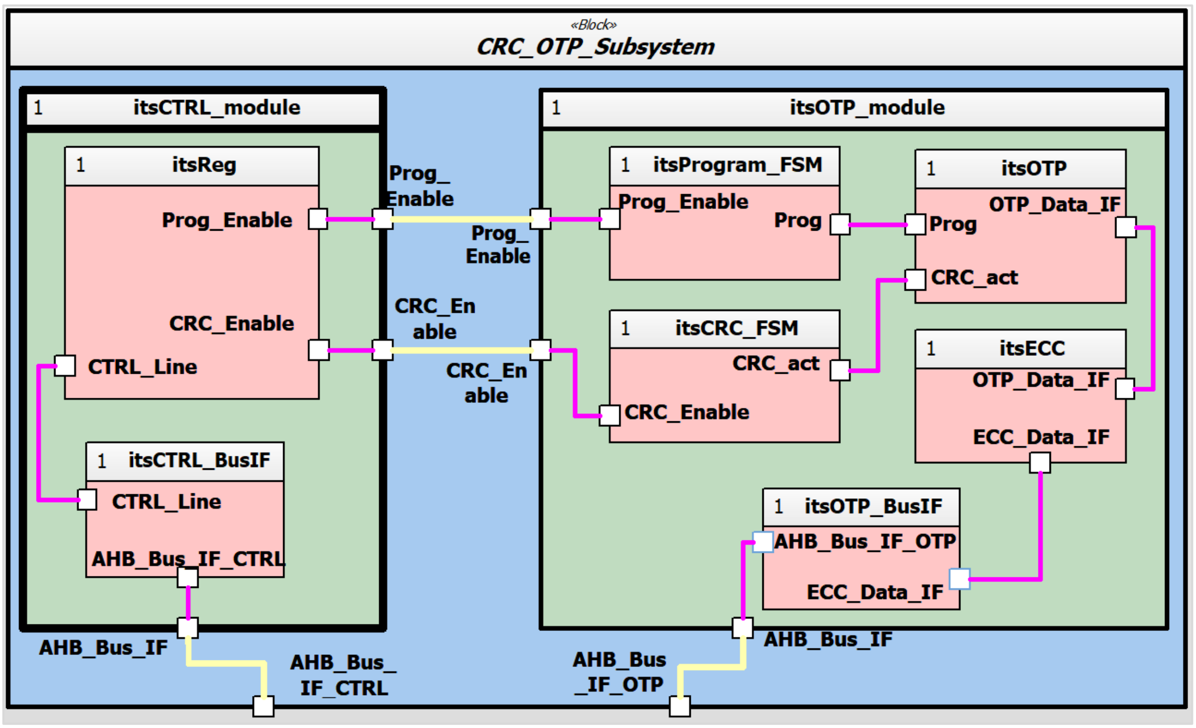


Figure 3: Example Architecture IBD Connectivity Check

For the architecture diagram in Figure 3, the hierarchy of the elements in relation to each other were also depicted in this representation, by modeling the part of *itsCTRL_module* within the *CRC_OTP_Subsystem* block. In the *Architecture Diagram* shown here, the ports of all hierarchy levels are mapped and connected using the SysML-element connectors. Individual ports are connected directly within a component and other connections extend across several hierarchy levels. The first step to generate the connectivity check as described is, to find the end port of each start port in the architecture diagram using a Java script and finally transfer the information into the table. Ports which, as with the connection, are located between *Prog_Enable* of the *itsReg* and *Prog_Enable* of the *itsProgram_FSM* must not be entered in the table. In the next step the Python script as described in the chapter above generates the SystemVerilog Assertions. Figure 4 shows an extract of the generated SystemVerilog Assertions based on the architecture diagram shown in Figure 3. The assertions check, Is the start port *portX* connected to the end port *portY* in the design.

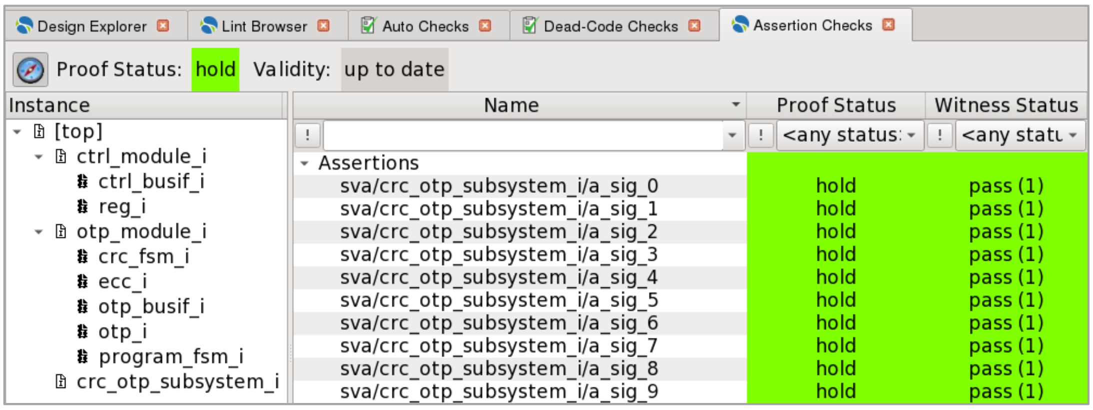
```
always_comb a_sig_2: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.crc_enable === crc_otp_subsystem.otp_module_i.crc_fsm_i.crc_enable)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg_i.crc_enable' to
  'crc_otp_subsystem.otp_module_i.crc_fsm_i.crc_enable' is broken");

always_comb a_sig_3: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.ctrl_line === crc_otp_subsystem.ctrl_module_i.ctrl_busif_i.ctrl_line)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg.ctrl_line' to
  'crc_otp_subsystem.ctrl_module_i.ctrl_ctrl_busif_i.ctrl_line' is broken");

always_comb a_sig_4: assert final (
  crc_otp_subsystem.ctrl_module_i.reg_i.prog_enable === crc_otp_subsystem.otp_module_i.program_fsm_i.prog_enable)
  else $error("Connection from 'crc_otp_subsystem.ctrl_module_i.reg_i.prog_enable' to
  'crc_otp_subsystem.otp_module_i.program_fsm_i.prog_enable' is broken");
```

Figure 4: Extract Result Generation Assertions

The figure shows among other SystemVerilog Assertions the result of the generation for the discussed example of the connection between *Prog_Enable* of the *itsReg* and *Prog_Enable* of the *itsProgram_FSM*. The figure also illustrates the change in naming convention between the model elements in Figure 3 and the VHDL as well as assertions in Figure 4 during generation. For example, the prefix "its" is deleted from the names of the parts in the model, the upper case is adjusted and the suffix "_i" is added for the VHDL instances. Afterwards the resulting assertions were loaded into the verification tool OneSpin 360 together with the VHDL architecture of the *CRC_OTP_Subsystem* and the connectivity of the architecture was checked. Figure 5 shows an extract of the OneSpin 360 [24].



Design Explorer | Lint Browser | Auto Checks | Dead-Code Checks | Assertion Checks

Proof Status: hold Validity: up to date

Instance
- [top]
  - ctrl_module_i
    - ctrl_busif_i
    - reg_i
  - otp_module_i
    - crc_fsm_i
    - ecc_i
    - otp_busif_i
    - otp_i
    - program_fsm_i
  - crc_otp_subsystem_i

Name	Proof Status	Witness Status
Assertions		
sva/crc_otp_subsystem_i/a_sig_0	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_1	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_2	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_3	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_4	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_5	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_6	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_7	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_8	hold	pass (1)
sva/crc_otp_subsystem_i/a_sig_9	hold	pass (1)

Figure 5: Successfully proven assertions in OneSpin 360

On the left side of the figure is the VHDL architecture of the *CRC_OTP_Subsystem* as a structure and on the right side an extract of the successfully executed assertions of the connectivity check. The automation of the development of connectivity checks based on the SysML model shows the possibilities of increasing efficiency in the verification area. The model-driven approach not only increases the effort in this area by improving the quality of the specification but also reduces the effort in developing the verification.

V. Conclusion

Based on the specification described in SysML, a method for model-driven generation of SystemVerilog Assertions for connectivity checks was presented. The realization of a modeled and formalized specification increases the explicitness of the specification and thus prevents errors in the implementation as well as in the verification. In addition, the modeling of the specification using SysML enables the description in a machine-readable format and thus enables the automation of the verification. With the method presented here, the effort can be achieved through the model-driven automation of the verification as well as through the reduction of errors by the formalization and modeling the specifications. As part of the paper the application of the method to an industrial example of an inertial sensor SoC was demonstrated and thus the generation of SystemVerilog Assertions directly from the modelled specification was shown.

Acknowledgment

This work has been developed in the ITEA3 project COMPACT. COMPACT is funded by the Finnish funding agency for innovation Tekes (Diary Number 3089/31/2017) and the German ministry of education and research (BMBF) (reference number: 01/S17028). The authors are responsible for the content of this publication.

References

[1] A. M. Mehta, ASIC/SoC Functional Design Verification, 1st ed., Springer, 2018, pp. 2 -8.

[2] A. Evans, A. Silburt, G. Vrckovnik, T. Brown and M. Dufresne, „Functional verification of large ASICs,“ *Proceedings 1998 Design and Automation Conference. 35th DAC*, San Francisco, CA, USA.

[3] O. Bringmann; W. Lange and M. Bogdan, Eingebettete Systeme, 3st ed., De Gruyter Oldenburg, 2018, pp. 47 - 48.

[4] P. Zhang, „Specification and verification of SOC using PTL,“ *in 2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)*, Wuhan, China.

[5] L. Musat, S. Kandl, P. Puschner, M. Hübl, A. Buzo and G. Pelz, *„Requirement semi-formalization methodology for SoC design,“ in 2015 International SoC Design Conference (ISOCC)*, Gyungju, South Korea.

[6] S. Lee, S. Park and Y. B. Park, „ Self-Adaptive System Verification based on SysML,“ *in 2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Auckland, New Zealand, 2019, pp. 1-3.

[7] M. Rahim, A. Hammad and M. Boukala-Ioualalen, "Towards the Formal Verification of SysML Specifications: Translation of Activity Diagrams into Modular Petri Nets," in *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, Okayama, 2015, pp. 509-516.

[8] M. d. S. Soares und J. Vrancken , „Requirements specification and modeling through SysML,“ *in 2007 IEEE International Conference on Systems, Man and Cybernetics, Montreal*, Que., Canada.

[9] M. R. S. Marques, E. Siegert und L. Brisolara, „Integrating UML, MARTE and sysml to improve requirements specification and traceability in the embedded domain,“ *in 2014 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre, Brazil*.

[10] *OMG Systems Modeling Language Version 1.6*, 12 2019, [online] https://www.omg.org/spec/SysML/1.6/PDF.

[11] R. Baranowski and M. Trunzer, „Complete Formal Verification of a Family of Automotive DSPs,“ *in 2016 Design and Verifikation Conference and Exhibition (DVCON)*, Munich, Germany.

[12] U. Kühne, S. Beyer, J. Bormann and J. Barstow, „*Automated formal verification of processors based on architectural models,“ in 2010 Formal Methods in Computer Aided Design*, Lugano, Switzerland.

[13] A. Kirchner, J.H. Oetjens and O. Bringmann, "Using SysML for Modeling and Code Generation for Smart Sensor ASICs," *in 2018 Forum on specification & Design Languages (FDL)*, pp. 5-16.

[14] A. Kirchner, J.H. Oetjens and O. Bringmann, "Automation of Embedded Software Development for Smart Sensor ASICs," *in 2019 2nd International Workshop on Embedded Software for Industrial IoT (ESIIT)*, Munich, Germany, pp. 3-9.

[15] A. Kirchner, J.H. Oetjens and O. Bringmann, "Using SysML for Modeling and Generation of Virtual Platforms," *in 2019 SNUG Europe 2019*, Munich, Germany.

[16] V. Gebhardt, G. M. Rieger, J. Mottok and C. Gießelbach, „Verifikations- und Validationsplanung,“ in Funktionale Sicherheit nach ISO 26262: Ein Praxisleitfaden zur Umsetzung, Heidelberg, dpunkt.verlag, 2013, pp. 135-166.

[17] C. Haubelt and J. Teich, „Einleitung,“ in *Digitale Hardware/Software-Systeme: Spezifikation und Verifikation*, Heidelberg, Springer-Verlag, 2010, pp. 11-22.

[18] M. Luisa, F. Mariangela and N. I. Pierluigi , „Market research for requirements analysis using linguistic tools,“ in Requirements Engineering volume 9, Springer Link, 2004, p. 40–56.

[19] ISO 26262-8:2011, „Road vehicles: Functional safety,“ International Organization for Standardization, 2011.

[20] "IEEE Std 1800TM-2012", *IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800-2012*.

[21] A. M. Mehta, "ASIC/SoC Functional Design Verification," 1st ed., Springer, 2018, pp. 75 -128.

[22] Python Software Foundation, „28.1. sys — System-specific parameters and functions,“ 2019. [Online]. Available: https://docs.python.org/2/library/sys.html. [Zugriff am Juni 2020].

[23] J. Ruf, R. J. Weiss, T. Kropf and W. Rosenstiel, „Modeling and Formal Verification of Production Automation Systems,“ in Integration of Software Specification Techniques for Applications in Engineering, Heidelberg, Springer Verlag, 2004, pp. 541-566.

[24] OneSpin Solutions, „OneSpin® 360 Design Verification Solutions,“ 2020. [Online]. Available: https://www.onespin.com/resources/flyers/.