A Roundtrip: From System Requirements to Circuit Variations and Back

Sören Kwasigroch, Nicolas Theobald, Johannes Koch, Christoph Grimm
Chair for Cyber-Physical Systems
University of Kaiserslautern-Landau
Kaiserslautern, Germany
{soeren.kwasigroch|ntheobal|johannes.koch|cgrimm}@rptu.de

***Abstract*—This paper presents an approach to the hierarchical verification of analog/mixed-signal (AMS) systems. The approach provides means for monitoring constraints and how they are satisfied across the whole development process, from left (requirements, constraints) to right (system integration). A novelty is that we propagate constraints continuously downwards towards implementation, and characterization results upwards back into constraint models. This has the advantage that inconsistencies and over-specification can be recognized earlier, which permits a more suitable resource partitioning. The approach is demonstrated by a multi-stage operational amplifier and a tire pressure monitoring system.**

***Index Terms*—Analog Mixed-Signal Systems, Process variation, Affine Arithmetic (AA), SystemC-AMS, SysMLv2**

I. Introduction

In particular for AMS systems with many uncertainties due to process, voltage, and temperature variations and significant non-ideal behavior of circuits a comprehensive verification planning is needed. Chen et al. give a detailed overview of verification challenges that include the verification time/effort and the test bench generation with suitable coverage in [1], [2]. Barnasconi et al. [3] further highlight the need for faster system verification that, in particular, could be achieved using models at a higher level of abstraction.

This work aims to improve communication throughout the AMS system development process and across levels of abstraction. This includes the generation of verification infrastructures to increase verification coverage and accuracy of behavioral models, as well human aspects like the documentation of test benches. A particular target is to reduce over-specification, which means unnecessary strict constraints by showing where constraints are over-satisfied.

To explain the approach, we use the V-model. The V-model structures the development horizontally into levels of abstraction and vertically into top-down specification activities on the left side and bottom-up integration activities on the right side. For AMS systems, we consider the abstraction levels (1) system level at which requirements and application needs are described, (2) block level at which blocks interact via directed signal flow, and (3) circuit level. On the system- and block-Level, we use SystemC AMS [4] models due to our familiarity with the language framework and its infrastructure. Furthermore, Barnasconi [3] concluded that the simulation speed of it is competitive to other system and block-level simulation tools, e.g., System Verilog. For the results on circuit level SPICE based tools have been used.

Verification activities in the V-model check performance characterization results after integration (right) against specified constraints (left).

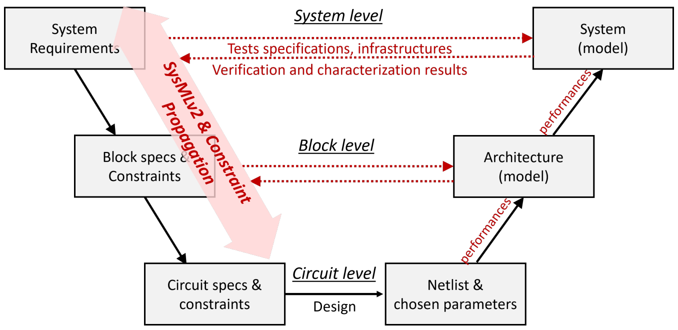

System Requirements
System level
Tests specifications, infrastructures
Verification and characterization results
System (model)
SysMLv2 & Constraint Propagation
performances
Block specs & Constraints
Block level
Architecture (model)
performances
Circuit specs & constraints
Circuit level
Design
Netlist & chosen parameters

Fig. 1. V-model with continuous verification across abstraction levels.

We extend the verification process within the V-model [5] as shown in Fig. 1 by linking the documentation, requirements, and specified constraints (left) with the integrated models and characterization results (right). We do this by generating SystemC test bench templates that include infrastructures that automatically feedback verification and characterization results into the system- and block-level specifications. Here, we check them against the specified constraints. To automate the methodology, we implemented a SysMLv2 based tool, called SysMD notebook [6], that includes techniques for constraint propagation and that interfaces with SystemC AMS [3] via the exchange of constraints and characterization results.

II. State of the Art

The verification of AMS systems is mostly based on modeling and simulation. Approaches to support verification can be structured into

1) Modeling and generation of verification infrastructures,
2) Means to support system-level modeling/simulation, particularly communication of verification and characterization results.

This work was partially funded by the BMBF project KI4BoardNet No. 16ME0782.

A. Modeling and generation of verification infrastructures

The generation of verification test cases and related infrastructures aims at supporting stakeholders at the system-level with application background; often they are not SystemC experts. In consequence, input languages are natural languages that are processed using natural language processing methods (NLP) [7], [8] or systems modeling languages like UML [9], [10] or SysML [11]–[14].

The NLP approaches specifically target the generation of assertions or verification test cases in languages like SystemC. The UML/SysML approaches more generally translate diagrams modeling structure or behavior into its equivalents in SystemC, often with support for round-trip (translation of SystemC back to UML/SysML). VeriSC [15] is more specifically related to verification; it generates a verification infrastructure that comprises a driver, a monitor and a checker module.

Compared with the above-mentioned approaches, we use SysMLv2 requirements and documentation as input and generate SystemC(-AMS) skeletons. We do not consider behavior and detailed generation. The reason is that SystemC experts might use their own verification packages and approaches, including but not limited to UVM(-AMS) [16]. However, the skeletons are intended to increase spec-coverage and to collect results of system verification and characterization back to the system specification model.

It is important to note that as of now we only deal with performance verification. For industrial applications, one must also consider the verification of connectivity (e.g., using SysML and generating IP-XACT [17]) and functional verification.

B. System-level modeling transport of characterization results

Means to support system-level modeling include methodologies for automated generation of behavioral models, e.g., [18], [19], or simulator coupling for cross-level simulation, e.g., [20]. Related to this work are, in particular, techniques for characterization. Such methods can be used to determine a block's performance through multi-run simulations at the circuit level and use these performances as parameters for block-and system-level models. Many characterization approaches, e.g. [21], [22], determine values or numerical probabilistic properties for system-level behavioral models. As dependencies are not modeled, any further propagation leads to over-approximations.

Approaches to include dependencies are described in [5], [23]–[26]. [26] uses a response surface modeling approach to model aging effects. [23], [25] use an affine kernel to model linear dependencies of performance properties from parameters similar to our approach. We use this approach towards a more comprehensive analysis that closes the gap to specified constraints. Compared with existing approaches for characterizing models, we link this information with the constraint propagation mechanism of a SysMLv2 based systems modeling tool and its constraint propagation mechanism.

C. Contribution

We show how to link simulation-based verification in SystemC and SPICE with a system specification in SysMLv2 in an overall verification approach:

1) We model structure, requirements, and constraints in SysMLv2 textual and use a constraint propagation mechanism that might reduce over-specification (Sec. III, Sec. V).
2) We bring characterization results across different levels of abstraction back into the above requirements model (Sec. IV)

We analyze the approach in two case studies: a multi-stage amplifier and a tire-pressure monitoring system in Sec. V.

III. Top-Down: Requirements and Tests

We document and model requirements and constraints in SysMLv2 textual. As a tool, we use a Jupyter notebook-like tool, called SysMD notebook [6]. It allows system and application experts to document tests using Markdown, including figures and equations in LaTeX, and to link this documentation with SysMLv2 textual models.

A. Modeling structure, requirements, constraints in SysMLv2

In SysMLv2 textual, we model intended architectures by block diagrams. In block diagrams, we connect parts via ports and interfaces. Note that in SysMLv2, textual interfaces (and subclasses thereof) connect elements like Signals in Systems. An example is given by Fig. 2.

```
package ams {
    import ScalarValues::*;
    import SI::*;
    import Signals::*;

    // Library elements
    part def Amplifier :> Base::Anything {
        port inp;
        port outp;
        attribute gain: Real [dB];
        // (...)
    }
}
// The system and block diagram level model
part multiStageAmplifier {
    port input;
    interface s1: Signal from input to lna.inp;
    part lna:       AmsLib::Amplifier;
    interface s2: Signal from lna.outp to stage2.inp;
    part stage2:    AmsLib::Amplifier;
    interface s3: Signal from lna.outp to driver.inp;
    part driver:    AmsLib::Amplifier;
    interface s4: Signal from driver.outp to output;
    port output;
    attribute gain: Real[db]=lna.gain*stage2.gain*driver.gain;
}
```

Fig. 2. SysMLv2 textual representation of a three-stage amplifier.

Requirements refer to a specific element (a part, a port, an interface) that is called the *subject*. Requirements can have a documentation and also constraints that shall be satisfied but are not necessarily satisfied, e.g. design artifacts. Also, requirements can have assumed requirements that must always be satisfied, e.g., related natural laws. An example is given by Fig. 3.

```
requirement gains {
    subject amp references multiStageAmplifier;

    //Requirements for the multiStageAmplifier
    require minGain { amp::gain >= 10.0 [dB] }
    require maxGain { amp::gain <= 20.0 [dB] }
}
```

Fig. 3. Requirement in SysMLv2 textual for the three-stage amplifier

B. Generation of SystemC verification infrastructures

The generated SystemC templates comprise source and header files for every SysMLv2 part, a main.cpp and a Makefile, as well as a CMakeLists.txt to enable compilation of the project. The templates inherit the structure and data of the original model, featuring the declaration of variables and constants with their respective values, the declaration of ports, including the correct wiring to their channels, and the instances of sub-modules. As only structural information is applied to the templates, actual behavior has to be added by the engineer.

To support the continuous verification approach, the generation process includes setting up a respective verification infrastructure. Test benches are generated for every SysMLv2 requirement in a separate folder. As depicted in Fig. 4, a test bench starts with comments that provide information about the requirements and constraints that shall be applied by block and circuit designers to cover the test scenario. Subsequently, the device under test (DUT) is instantiated and embraced by two placeholders that indicate the implementation of a driver (stimuli) which provides the input signal, a monitor, and the post-processing of the gathered simulation data, which prepares the simulation results for propagation to the SysMLv2 model. Therefore the SystemC designer using this template need to replace *'PASTE_RESULT_HERE'* with a function call or variable that returns the simulated *minGain* and *maxGain*.

A particular aspect of the test bench is its capability to feed results back into the SysMLv2 model. This is enabled via the ResultWriter class, which generates a JSON file that can be imported by the SysMD Notebook. At the end of the test bench, the necessary addResult() method calls are provided, to which the variables containing the results must be passed. These function calls also contain additional information that enables proper checking of the constraints and correct backpropagation of the result to the respective element of the SysMLv2 model.

C. Getting "Bottom Up" feedback; agile resource partitioning

After characterization, as described in the following section, a generated JSON file (see Fig. 5) contains the simulation results that can be propagated back into the SysMLv2 model. To ensure correct propagation, every entry features the full qualified name of the attribute. In our notebook, the values of SysMLv2 attributes can be either explicitly specified or calculated by functions (*min()*, *sin()*, ..). To use available results, an attribute's value must be specified via the *characterizedResult()* function, which searches for a matching entry in a given result file. To ensure that the constraint-net also

```
int sc_main(int argc, char* argv[]){
    // --- REQUIREMENTS ---
    //DUT: multiStageAmplifier_CLASS
    //Requirement minGain requires that
    //  multiStageAmplifier::gain >= 20.0 [dB]
    //Requirement maxGain requires that
    //  multiStageAmplifier::gain <= 30.0 [dB]

    // --- STIMULI ---

    // --- DUT ---
    //This is the device that is under test
    multiStageAmplifier_CLASS dut("dut");

    // --- MONITORING ---
    sc_start(1,SC_SEC);

    // --- RESULT WRITING and POST-PROCESSING ---
    ResultWriter rw;

    rw.addResult("minGain", PASTE_RESULT_HERE, "dB",
        Operator::GE, 20.000000000000025, "dB",
        "multiStageAmplifier::gain");
    rw.addResult("maxGain", PASTE_RESULT_HERE, "dB",
        Operator::LE, 30.000000000000032, "dB",
        "multiStageAmplifier::gain");

    rw.writeResultFile();

    return 0;
}
```

Fig. 4. Generated test bench for the gain requirements.

works if no results are available, the function can additionally be supplied with an auxiliary function executed otherwise.

```
[
  {
   "constraintName":"maxGain",
   "resultValue":34.239,
   "resultUnit":"dB",
   "referenceValue":30,
   "referenceUnit":"dB",
   "successful":0,
   "attributeQualifiedName":"multiStageAmplifier::gain"
  },
```

Fig. 5. JSON file with characterization results for the "maxGain" requirement.

IV. Bottom-Up: Characterization of Performances

In particular, AMS systems are often specified by their function and (performance) parameters at the left side of the V-model. In this section, we focus on the right side of the V-model. Here, the parameters of an implementation resp. model thereof are determined by characterization. Characterization results are propagated from the circuit-level models to system-level models, and as well back into the SysMLv2 model and its constraint propagation. For accuracy, it is of paramount importance to capture and maintain all correlations. The reason is that the variations are usually of significant size in AMS circuits. However, error cancellation techniques like differential designs or correlation techniques are used at all levels of abstraction to compensate for these variations.

A. Characterization of parameters

The underlying theory described in Zivkovic et al. [24] suggests characterizing the performances of circuits as linear

functions depending on voltage, temperature, and process parameters. These functions should be saved in Affine Forms [27], which will be introduced later in this section, for the insurance of save inclusion of each possible value of these performances.
In this paper, we focus on its integration into a design flow. We use worst case and sensitivity analysis (concretely: MunEDA's tool WiCkeD [28]) at the circuit level for relevant parameters with variations, e.g., voltage range, temperature range, and a selected number of process parameters. For these parameters, we determine:

- Worst case performances, e.g. slew rate, offset error for the chosen parameters with variations.
- Sensitivities of the performances to the chosen parameters.

The results are then represented as an Affine Form [27]. Affine forms are a mathematical approach to represent and compute uncertain values x that are chosen from an interval in a non-deterministic way. An (unknown) value x subject to n sources of variability is represented by an Affine Form $\tilde{x}$:

$$x \in \tilde{x} ::= x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \tag{1}$$

where

- $x_0 \in \mathbb{R}$ is the central value; we simply use the middle of the resulting property range.
- $\varepsilon_i \in [-1, 1]$ are noise symbols that represent normalized sources of uncertainty, and $x_i \in \mathbb{R}$ are factors like the uncertainty of process or voltage parameters that scale the contribution of different sources of uncertainty to the value of x resp. $\tilde{x}$. We determine these parameters by sensitivity analysis at the central value.

We furthermore use an additional noise symbol and factor for all other parameters, non-linearities, etc., to guarantee safe inclusion. Note that this allows us to deal with variations that are modeled as ranges.

It is important to understand that the indices of the noise symbols model correlation once we use many different performances. Then, a subtraction of two performances, e.g. for error cancellation in a differential circuit, will subtract correlated parameters as they share noise symbols with the same index. We refer the reader to [27] for details on Affine Arithmetic. To deal with probabilistic variations, we would have to use representations as described in [25]; this is, however, not yet implemented and is subject to future work.

B. Export of characterization results

In our example, sizing and characterization data are exported as a JSON file from MunEDA's WiCkeD tool [28]. It is structured in the sections *Sizing*, *Performance* and *Results*. The section *Sizing* section lists information for each parameter impacted by sizing in the design. The section *Performance* declares the propagated performances; in its section *Results*, the sensitivity from these performances to the parameters are listed as well, with the optimized extreme values of these performances. These are the input values to build the Affine Forms as described above.

C. SystemC AMS models with characterization results

To integrate the circuit level characterization results into SystemC AMS simulation at block or system level, e.g. for Corner-Case analysis, we

- select an assignment for the circuit-level set of parameters,
- generates the Affine forms to the respective sensitive values from the characterization results,
- run the SystemC AMS simulation with different parameter sets, depending on the analysis.

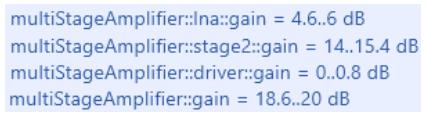
These steps are supported by the created C++ classes *JSONImport*, *AF_IndexStorage*, *Performance*. Remember that the indices of Affine Forms identify dependencies. The *AF_IndexStorage* therefore identifies globally (even across modules) shared indices and hence dependencies resp. correlations.

V. USE CASES: MULTI-STAGE AMPLIFIER AND TPMS

In this section, we describe two use cases: a multi-stage amplifier and a tire pressure monitoring system (TPMS) [29].

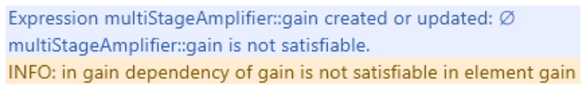
A. Multi-stage amplifier: agile ressource partitioning

Throughout Sec.III and IV, we have already given some details of the multi-stage amplifier example. In the following, we show how constraints can be refined in an agile way. For this purpose, we permit uncertain constraints at the block level that, however, must satisfy the overall system specification: The initial specifications of the example are lna: $[4.6, 12.0]$ [dB], s2: $[14, 16.8]$ [dB], and driver: $[0, 0.8]$ [dB]. So, the tool calculates the amplification from each stage and the total amplification as Fig. 6 shows. However, the SystemC simulation concluded an actual lower bound of the amplification of s2 of 15.8 [dB]. Therefore, the SysMD tool correctly shows us a violation of total gain $[10, 20]$ [dB] in Fig. 7.



```
multiStageAmplifier::lna::gain = 4.6..6 dB
multiStageAmplifier::stage2::gain = 14..15.4 dB
multiStageAmplifier::driver::gain = 0..0.8 dB
multiStageAmplifier::gain = 18.6..20 dB
```

Fig. 6. The individual values of the stages and the final gain after propagation with initial specifications.



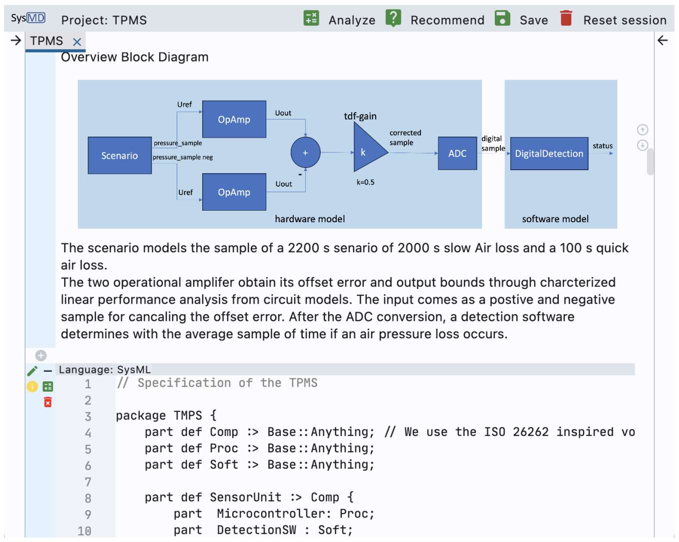
```
Expression multiStageAmplifier::gain created or updated: ∅
multiStageAmplifier::gain is not satisfiable.
INFO: in gain dependency of gain is not satisfiable in element gain
```

Fig. 7. Propagation after updated value of stage 2.

B. Tire pressure monitoring system (TPMS)

By the example of a TPMS system, we analyze the ability to avoid over-approximation due to the wrapping effect [27]. A TPMS monitors the tire for pressure losses. For analysis, we use a circuit-level model of operational amplifiers(OpAmps) for the analog pre-processing. The bottom-up characterization was applied on an OpAmp [30][p.12ff.]. As a simulation target, the reaction time to these pressure losses will be written out to a log file from the simulation.

1) Requirements model: The specification of the structure of these models is done in SysMLv2. The expected components and the attribute range are specified in this file (compare with Fig.8).



The scenario models the sample of a 2200 s senario of 2000 s slow Air loss and a 100 s quick air loss.
The two operational amplifer obtain its offset error and output bounds through charcterized linear performance analysis from circuit models. The input comes as a postive and negative sample for cancaling the offset error. After the ADC conversion, a detection software determines with the average sample of time if an air pressure loss occurs.

```
// Specification of the TPMS

package TMPS {
    part def Comp :> Base::Anything; // We use the ISO 26262 inspired vo
    part def Proc :> Base::Anything;
    part def Soft :> Base::Anything;

    part def SensorUnit :> Comp {
        part  Microcontroller: Proc;
        part  DetectionSW : Soft;
```

Fig. 8. TPMS System level model (excerpt).

The constraint net, evaluating the specification of SysMLv2 [6], can compute the upper and lower bound of dependent attributes between the components with this information.

2) SystemC AMS model at block level: The structure is partitioned as seen in the Fig. 8. All models are created using the Timed Data Flow model of computation using SystemC AMS. There are four models simulating hardware in this TPMS: the pressure sensor, two OpAmps, amplifying the sample in the right range, and an ADC. After that, a software detection block decides if a pressure loss occurs. Only the OpAmp model is written as a block-level electric model. The remaining blocks model simple ideal behavior.

In this system, the pressure sensor outputs a positive or negative mirrored measurement value from 0 to 0.1 mV, respectively, -0.1 to 0 mV, with a sample rate of 1000 Hz. In this model, both output ports of the scenario module are connected to two identical OpAmps. Therefore, the offset will be derived from the same value. The offset error will cancel itself by summing up the OpAmp output with different signs. For the analog to digital conversion, a 10-bit ideal model maps analog values from 0 to 1 mV to a range from 0 to 1023. Therefore, the requirement for the OpAmp model must have at least an amplification $V = 10$. The detection module computes the summed-up differences of the ADC output values over the simulation time and determines if a pressure loss occurs.

3) Circuit characterization for behavioral model: The OpAmp behavioral model at block- and system-level is modeled in SystemC AMS as a simple transfer function:

$$\begin{aligned} A(s) &= \frac{U_{out}(s)}{U_{in}^{+}(s) - U_{in}^{-}(s)} = \frac{U_{out}(s)}{U_{diff}(s)} \\ &= \frac{A_O}{s * (1 + A_O * A_g)}. \end{aligned} \tag{2}$$

Here, U_{in} is the input voltage with a plus and minus sign, U_{out} the output voltage, A the total amplification, A_O the open loop gain, and the wired gain. This transfunction is implemented with the *sca_tdf::sca_ltf_nd* class, which takes the transfer function in the nominator-denominator form. It considers the four performance properties: the open loop gain $A_O[dB]$, the maximum output voltage $U_{out,max}[V]$, the minimum output voltage $U_{out,min}[V]$ and the offset $U_{os}[V]$. The feedback network transfer function A_g is derived by the classical negating OpAmp setup. The time behavior, denoted with small letters, will then be refined as follows:

$$u_{out}(t) = \max\{u_{out,min}, \min\{u_{diff}(t) * A, u_{out,max}\}\} + u_{os}. \tag{3}$$

Values of the performance lie within the following ranges:

- A_O =[-208.19, 390.19] [dB]
- $u_{out,max}$ =[-9.57,10.62] [V]
- $u_{out,min}$ =[-0.86,1.84] [V]
- u_{os} =[-6.03,5.99] [V]

This is obtained by summing up the maximal and minimal range over all 69 used parameters: $x_0 + \sum_{i=1}^{69} |x_i|$.

4) Input: The input was chosen to be a 2200 s simulation time scenario for demonstration purposes, which is not very realistic for a car tire. It contained a 2000-second (33 min 20s) long and 100-second fast airdrop. In Fig. 9, the trace of this pressure sensor's positive and negative-signed samples are depicted for both scenarios in the first row.

5) Results: For the depicted simulation run in Fig. 9 the performances have the following configuration : $A_0 = 2.19e^{+6}$, $u_{out,min} = 0.99$, $u_{out,max} = 8.72$ and $u_{os} = -2.92$. The first row shows the inputs of the *scenario* module. Below are the outputs of the operational amplifier and the corrected sample. *In particular, the offset error of the circuits is partially canceled in the overall behavioral model – this is as expected as the affine forms that parameterize it include the correlation information*. The third row shows the digital value of the ADC module outputs. Furthermore, the status of the detection software is plotted. It starts with 8 for *no pressure loss* and changes to 12 for a pressure loss after 168s. After the tire is filled up, the status returns to status 8; however, it returns a pressure loss after 8s. The TPMS was simulated for seven runs with a total simulated time of 2200s. The wall clock time with an interfacing module *JSONImport* needed an average of 81.264s. The wall clock time for simulation without

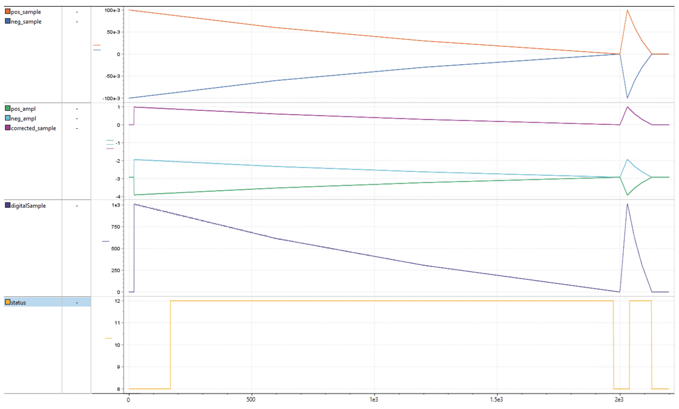


Fig. 9. Trace-file of an arbitrary parameter configuration

interfacing and hard-coded performances averaged 81.313s. This indicates that the overhead created by the interfacing module, which involves reading the properties in JSON and determining the values based on a configuration, is marginal.

VI. Conclusion

We have described a verification approach that links the requirements on the left side of the V-model with the characterization results from model-based verification on the right side of the V-model. We have shown how to use SysMLv2 textual in this context to model requirements and constraints and how to generate SystemC templates. An advantage of the approach is that characterization results are propagated back into the requirements model in SysMLv2. In the requirements model, a constraint propagation mechanism then checks the overall consistency and allows a re-partitioning of resources. This can help avoid over-specification.

The methodology is supported by a tool called *SysMD Notebook* that we demonstrated in two examples. The tool supports a subset of SysMLv2 textual: KerML and SysMLv2 with limited support for behavioral modeling. This subset is sufficient for the given use case, modeling structure, requirements, and constraints. Current work increases the supported subset of SysMLv2. The tool is being published as open source, and its integration of constraint propagation is being submitted as an RFC to the OMG standardization body.

The usefulness of the approach also depends on the availability of libraries. Future work deals with two modeling libraries: one for modeling the in-vehicle networks as described in [6]. The other library targets the specification and verification of AMS systems.

References

[1] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and Trends in Modern SoC Design Verification," *IEEE Design & Test*, vol. 34, no. 5, pp. 7–22, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8000621/

[2] A. Fürtig, G. Gläser, C. Grimm, L. Hedrich, S. Heinen, H.-S. L. Lee, G. Nitsche, M. Olbrich, C. Radojicic, and F. Speicher, "Novel metrics for Analog Mixed-Signal coverage," in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2017, pp. 97–102.

[3] M. Barnasconi, "SystemC AMS extensions: Solving the need for speed," *DAC Knowledge center*, vol. 6, 2010.

[4] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich, "An introduction to modeling embedded analog/mixed-signal systems using systemc ams extensions," in *DAC2008 International Conference*, vol. 23, 2008.

[5] B. Prautsch, H. Dornelas, R. Wittmann, F. Henkel, F. Schenkel, J. Koelsch, C. Grimm, and G. Strube, "AnastASICA – towards structured and automated analog/mixed-signal IC design for automotive electronics," in *ANALOG 2020; 17th ITG/GMM-Symposium*, 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9257337

[6] *Constructive Model Analysis of SysMLv2 models by Constraint Propagation*. IEEE, 2024.

[7] C. M. Kirchsteiger, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "Specification-based verification of embedded systems by automated test case generation," in *Distributed Embedded Systems: Design, Middleware and Resources*, B. Kleinjohann, W. Wolf, and L. Kleinjohann, Eds. Boston, MA: Springer US, 2008, pp. 35–44.

[8] C. B. Harris and I. G. Harris, "Glast: Learning formal grammars to translate natural language specifications into hardware assertions," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 966–971.

[9] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren, "Uml for esl design - basic principles, tools, and applications," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, 11 2006, pp. 73–80.

[10] C. Xi, L. JianHua, Z. ZuCheng, and S. YaoHui, "Modeling systemc design in uml and automatic code generation," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 932–935. [Online]. Available: https://doi.org/10.1145/1120725.1120760

[11] K. Nguyen, Z. Sun, P. Thiagarajan, and W.-F. Wong, "Model-driven soc design via executable uml to systemc," in *25th IEEE International Real-Time Systems Symposium*, 2004, pp. 459–468.

[12] W. Raslan and A. Sameh, "System-Level Modeling and Design using SysML and SystemC," in *2007 International Symposium on Integrated Circuits*, 2007, pp. 504–507.

[13] A. Abdulhameed, A. Hammad, H. Mountassir, and B. Tatibouet, "An approach based on SysML and SystemC to simulate complex systems," in *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2014, pp. 555–560.

[14] D. Genius and L. Apvrille, "A Tool for Investigating Cyber-Physical Systems via SystemC AMS Virtual Prototypes Derived from SysML Models," in *DVCon Europe 2023; Design and Verification Conference and Exhibition Europe*, 2023, pp. 27–33.

[15] K. R. G. da Silva, E. U. K. Melcher, G. Araujo, and V. A. Pimenta, "An automatic testbench generation tool for a SystemC functional verification methodology," in *Proceedings of the 17th Symposium on Integrated Circuits and System Design*, ser. SBCCI '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 66–70. [Online]. Available: https://doi.org/10.1145/1016568.1016592

[16] T. Maehne, Z. Wang, B. Vernay, L. Andrade, C. Ben Aoun, J.-P. Chaput, M.-M. Louërat, F. Pêcheux, A. Krust, G. Schröpfer, M. Barnasconi, K. Einwich, F. Cenni, and O. Guillaume, "UVM-SystemC-AMS based framework for the correct by construction design of mems in their real heterogeneous application context," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2014, pp. 862–865.

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

[18] M. U. Farooq, L. Xia, F. A. Hussin, and A. S. Malik, "High level fault modeling and fault propagation in analog circuits using NLARX automated model generation technique," in *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, vol. 2, 2012, pp. 846–850.

[19] C. Sanchez-Lopez and E. Tielo-Cuautle, "Behavioral model generation for symbolic analysis of analog integrated circuits," in *International Symposium on Signals, Circuits and Systems, 2005. ISSCS 2005.*, vol. 1, 2005, pp. 327–330 Vol. 1.

[20] Y. Zaidi, C. Grimm, and J. Haase, "Fast and unified SystemC AMS - HDL simulation," in *2009 Forum on Specification & Design*

Languages (FDL), 2009, pp. 1–6, ISSN: 1636-9874. [Online]. Available: https://ieeexplore.ieee.org/document/5404078

[21] D. De Jonghe and G. Gielen, "Characterization of Analog Circuits Using Transfer Function Trajectories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1796–1804, 2012.

[22] T. Koskinen and P. Cheung, "Hierarchical tolerance analysis using statistical behavioral models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 5, pp. 506–516, 1996.

[23] C. Zivkovic, C. Grimm, J. Kölsch, D. Short, M. Ferstl, D. Denger, D. Krems, and A. Barisic, "Bringing uncertainties into system simulation: A SystemC AMS case study," in *2020 Forum for Specification and Design Languages (FDL)*, 2020, pp. 1–6, ISSN: 1636-9874. [Online]. Available: https://ieeexplore.ieee.org/document/9233008

[24] C. Zivkovic, C. Grimm, M. Olbrich, O. Scharf, and E. Barke, "Hierarchical Verification of AMS Systems With Affine Arithmetic Decision Diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1785–1798, 2019.

[25] C. Zivkovic, J. Roedel, N. Chavan, F. Rethmeier, and C. Grimm, "Variation-Aware Performance Verification of Analog Mixed-Signal Systems," in *DVCon Europe 2023; Design and Verification Conference and Exhibition Europe*, 2023, pp. 7–13. [Online]. Available: https://ieeexplore.ieee.org/document/10461376

[26] N. Heidmann, N. Hellwege, M. Taddiken, D. Peters-Drolshagen, and S. Paul, "Analog behavioral modeling for age-dependent degradation of complex analog circuits," in *2014 Proceedings of the 21st International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2014, pp. 317–322. [Online]. Available: https://ieeexplore.ieee.org/document/6872209

[27] J. Stolfi and L. H. de Figueiredo, "An Introduction to Affine Arithmetic," *TEMA Trend. Mat. Apl. Comput.*, vol. 4, pp. 297–312, 2003. [Online]. Available: https://tema.sbmac.org.br/tema/article/view/352

[28] H. Dornelas, A. Schmidt, G. Strube, and E. Fabris, "New Technology Migration Methodology for Analog IC Design using MunEDA tools," Tech. Rep, Tech. Rep., 2015.

[29] N. N. Hasan, A. Arif, M. Hassam, S. S. Ul Husnain, and U. Pervez, "Implementation of tire Pressure Monitoring System with wireless communication," in *2011 International Conference on Communications, Computing and Control Applications (CCCA)*, 2011, pp. 1–4.

[30] G. B. Clayton and S. Winder, *Operational amplifiers*. Elsevier, 2003.