Lecture Notes in Electrical Engineering 454

Franco Fummi
Robert Wille
Editors

Languages, Design Methods, and Tools for Electronic System Design

Selected Contributions from FDL 2016

Springer

Lecture Notes in Electrical Engineering

Volume 454

About this Series

"Lecture Notes in Electrical Engineering (LNEE)" is a book series which reports the latest research and developments in Electrical Engineering, namely:

- Communication, Networks, and Information Theory
- Computer Engineering
- Signal, Image, Speech and Information Processing
- Circuits and Systems
- Bioengineering

LNEE publishes authored monographs and contributed volumes which present cutting edge research information as well as new perspectives on classical fields, while maintaining Springer's high standards of academic excellence. Also considered for publication are lecture materials, proceedings, and other related materials of exceptionally high quality and interest. The subject matter should be original and timely, reporting the latest research and developments in all areas of electrical engineering.

The audience for the books in LNEE consists of advanced level students, researchers, and industry professionals working at the forefront of their fields. Much like Springer's other Lecture Notes series, LNEE will be distributed through Springer's print and electronic publishing channels.

More information about this series at http://www.springer.com/series/7818

Franco Fummi • Robert Wille
Editors

Languages, Design Methods, and Tools for Electronic System Design

Selected Contributions from FDL 2016

Springer

Editors
Franco Fummi
University of Verona
Verona, Italy

Robert Wille
Johannes Kepler University Linz
Linz, Austria

ISSN 1876-1100 ISSN 1876-1119 (electronic)
Lecture Notes in Electrical Engineering
ISBN 978-3-319-62919-3 ISBN 978-3-319-62920-9 (eBook)
DOI 10.1007/978-3-319-62920-9

Library of Congress Control Number: 2017950291

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

The increasing integration and complexity of electronic system design requires constant evolution of the used languages as well as associated design methods and tools. The *Forum on specification and Design Languages* (FDL) is an established international forum devoted to the dissemination of research results, practical experiences, and new ideas in the application of specification, design, and verification languages. It considers descriptions means for the design, modeling, and verification of integrated circuits, complex hardware/software embedded systems, and mixed-technology systems.

FDL is the main platform to present and discuss new trends as well as recent work in this domain. The 2016 edition of FDL was an interesting and lively meeting thanks to the commitment of the authors and presenters.

This book is devoted to FDL 2016 and contains the papers that were evaluated best by both the members of the program committee as well as the participants of the forum which took place in September 2016 in Bremen, Germany. It reflects thereby the wide range of topics which have been covered at this event. The selected contributions particularly highlight the increasing role of AMS languages and verification tools—as essential, e.g., in the fields of smart systems and IoT, where devices are the result of a deep integration of heterogeneous analog and digital components. The papers propose state-of-the-art methodologies for their design, verification, and safety analysis. By this, the portfolio of papers in this book provides an in-depth view on the current developments in our domain which surely will have a significant impact in the future.

We would like to thank all authors for their contributions as well as the members of the program committee and the external reviewers for their hard work in evaluating the submissions. Special thanks go to Rolf Drechsler and his team from the University of Bremen, who were responsible for a splendid organization of FDL

2016, as well as Sophie Cerisier from the *Electronic Chips and Systems design Initiative* (ECSI). Finally, we would like to thank Springer for making this book possible.

Verona, Italy — Franco Fummi
Linz, Austria — Robert Wille
May 2017

Contents

Knowing Your AMS System's Limits: System Acceptance Region Exploration by Using Automated Model Refinement and Accelerated Simulation

Georg Gläser, Hyun-Sek Lukas Lee, Markus Olbrich, and Erich Barke

Abstract Virtual prototyping of Analog/Mixed-Signal (AMS) systems is a key concern in modern SoC verification. Achieving first-time right designs is a challenging task: Every relevant functional and non-functional property has to be examined throughout the complete design process. Many faulty designs have been verified carefully before tape out but are still missing at least one low-level effect which arises from interaction between one or more system components. Since these extra-functional effects are often neglected on system level, the design cannot be rectified in early design stages or verified before fabrication. We introduce a method to determine system acceptance regions tackling this challenge: We include extra-functional effects into the system models, and we investigate their behavior with parallel simulations in combination with an accelerated analog simulation scheme. The accelerated simulation approach is based on local linearizations of nonlinear circuits, which result in piecewise-linear systems. High-level simulation speed-up is achieved by avoiding numerical integration and using parallel computing. This approach is fully automated requiring only a circuit netlist. To reduce the overall number of simulations, we use an adaptive sampling algorithm for exploring systems acceptance regions which indicate feasible and critical operating conditions of the AMS system.

Keywords Parameter space • Acceptance region • Piece-wise linear • Simulation • Modeling • Bordersearch • Mixed-signal • Virtual prototyping • Automated model refinement • Design automation • Extra-functional properties • Accelerated simulation • System level • Verification

G. Gläser (✉)
IMMS Institut für Mikroelektronik- und Mechatronik-Systeme gemeinnützige GmbH, Ehrenbergstr. 27, D-98693 Ilmenau, Germany
e-mail: georg.glaeser@imms.de

H.-S.L. Lee • M. Olbrich • E. Barke
Institut für Mikroelektronische Systeme, Applestr. 4, D-30167 Hannover, Germany
e-mail: lukas.lee@ims.uni-hannover.de; markus.olbrich@ims.uni-hannover.de; erich.barke@ims.uni-hannover.de

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_1

1 Introduction

Many carefully verified designs fail due to flaws which neither the design nor the verification engineer has identified. As shown in Fig. 1 the design flaw might even be located outside the functional behavior of the system's components: Distorted supplies or parasitic couplings can raise severe problems that are only visible in certain conditions but are crucial to the overall functionality. Such design flaws are usually not covered by abstract system-level models since they neglect low-level effects. Within this contribution, we consider a common DC–DC converter circuit as shown in Fig. 2 for demonstration. This hysteretic current-mode buck converter will always be stable in simulation assuming idealized models and reference voltages [1]. However, its stability can be influenced by distortions not visible on system level. For example, distortions due to ground-bounce or crosstalk from the supply to the reference voltages may cause malfunction. Uncovering these interactions by simulation on transistor or layout level is virtually impossible for such a demonstrator or even more complex systems due to enormous computing times.

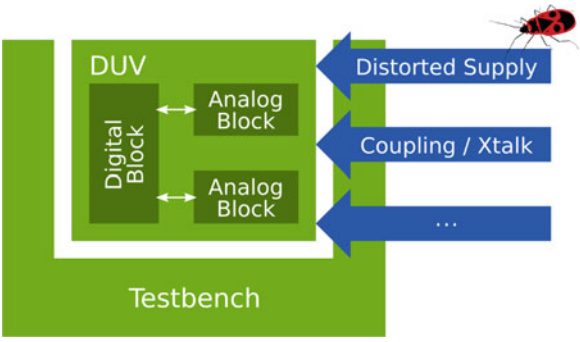

Fig. 1 System-level verification targets at verifying all functional properties using abstract models

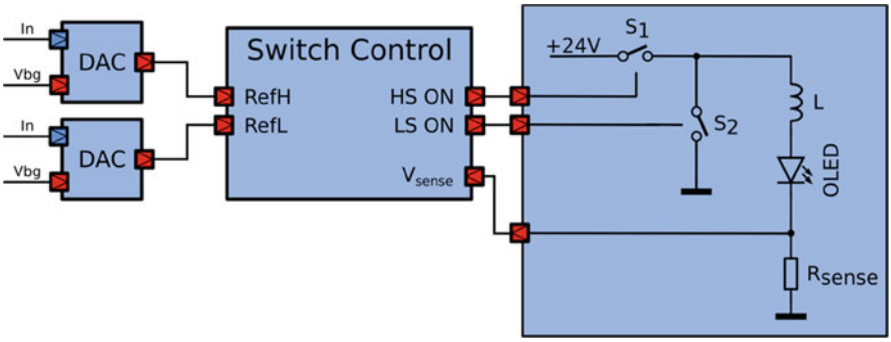

Fig. 2 Hysteretic current-mode buck converter as application scenario. The output current is determined via references generated by digital-to-analog converters (DAC)

In this contribution, we introduce a methodology to determine the system acceptance regions (SAR), i.e., the safe operating regions in the system-level parameter space of the distorting effects. In contrast to process and design parameters, they are not usually included in predefined models since they emerge from parasitic effects not visible on the regarded level of abstraction. We make their parameter space accessible by refining the existing models with parasitic effects using an automated procedure. Its exploration requires a large number of simulations in order to identify the system acceptance regions. This number can be reduced by using a systematic sampling methodology based on the *Bordersearch* algorithm [2]. However, further optimization of the simulation performance is needed due to the analog components in the simulation. We avoid time-consuming numerical integration algorithms by utilizing a piecewise-linear (PWL) modeling scheme that results in a piecewise-linear system. Considering nonlinear system behavior, the switching from one linearized circuit state to another state is a key issue. We have implemented an algorithm for determining the switching points along with multi-core processing for further performance improvement.

Following the discussion of the state of the art in Sect. 2, we introduce a new methodology to refine models with extra-functional properties in Sect. 3. Aiming at a high simulation performance, we use the accelerated analog simulation presented in Sect. 4. Based on the introduced methods, we explain the concept and exploration of system acceptance regions in Sect. 5. In Sect. 6 we demonstrate the method for a designed and fabricated DC–DC converter.

2 Related Work

Modern embedded AMS systems have been subject to research for a long time [3]. Still, the challenges arising in today's complex system-on-chips demand new methods for design and verification. Effects like crosstalk causing *signal integrity* issues have been studied [4, 5] especially in the digital domain. Addressing these challenges in analog systems is subject to ongoing research [6] since they are only visible on system level by considering effects from lower levels of abstraction. If the parameters of these effects are not specified at the beginning of the design phase they impose a significant design risk.

The evaluation of the system performance influenced by additional low-level effects requires appropriate models to be implemented. These methods have been studied, e.g., by Alassir et al. [4] and Eo et al. [7]. Introducing these effects automatically into system-level simulation is still uncommon: Fault injection by, e.g., distorting signals with saboteur modules was proposed by Leveugle and Ammari [8] but those approaches lack a general framework for AMS model refinement that could be used for exploring the newly introduced parameter space.

Procedures for extracting fault and acceptance regions were developed, e.g., by Dobler et al. [2] who also discussed the use of *Design of Experiments* based methods [9] in this context. Similarly, methods for extracting feasible regions in parameter

spaces have been developed by Stehr et al. for sizing and optimizing purely analog circuits [10]. However, the use of these methods for extracting acceptance regions in combination with model refinement to gain knowledge about the given system has not yet been published.

Each parameter space exploration algorithm demands for many simulations to be executed—especially for high-dimensional problems. Hence, the simulation performance is crucial to efficiently explore these regions. To speed-up our simulations, we use an accelerated analog simulation approach [11, 12]. This approach uses piecewise-linear models to avoid numerical integration and nonlinear equation solving.

This combination of accelerated analog simulation and automated refinement of component models with extra-functional properties is used to identify the critical scenarios in a AMS system at a very early design stage.

3 Automated Model Refinement

A large number of simulations is necessary to reach a sufficiently high verification coverage. Executing these simulations using low-level models on, e.g., transistor or layout level is clearly not feasible due to extreme high computing times. More abstract models try to solve this problem by reducing the simulation complexity by only implementing purely functional properties. However, additional effects as for instance power-supply rejection are usually neglected. The decision which effects can be neglected is crucial: If only a single relevant effect is neglected, the verification might falsely accept the system behavior—whereas the critical effect is not visible in simulation causes the design to fail. Since this set of relevant effects is unique for each design, a flexible modeling approach is needed to adapt to the actual use case.

The implementation and maintenance effort significantly increases if different combinations of effects have to be modeled and their impact on the system evaluated. Consider a system demanding for five effects and their interactions to be regarded. This raises the task of implementing $2^5 - 1 = 31$ variants of the model to be realized and maintained and urges for an automated approach.

Whereas our approach is not limited to a specific modeling language, we consider an existing system-level component model in SystemC-AMS [13] because of the availability of tools for code analysis. The system refined by an additional effect is shown in Fig. 3. Analyzing the model code automatically using libClang [14] yields structural information—e.g., ports, signals, internal structure, and locations of functions—about the targeted component model and its instances in the overall system. Based on this, a predefined generic text-template [15] is rendered to generate the model code for a wrapper realizing the actual effect. This procedure is repeated for all refinements to be applied to the model. Note that refinements might not be commutative: Consider, e.g., a multiplicative and an additive noise source at a given port of a model or one effect depending on another. In such cases,

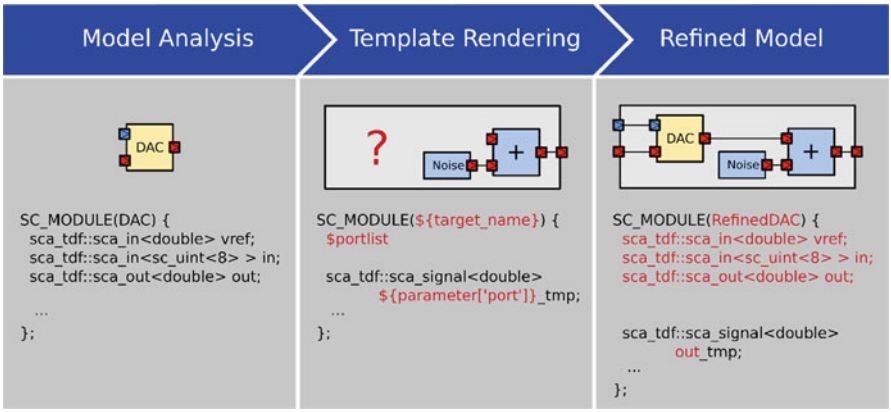

Fig. 3 Refinement flow of a component model

a predefined order of applying refinements has to be ensured. Since the template rendering engine has no knowledge about the nature of the effect, the order has to be defined by the user.

Effects arising in the interaction of several system components cannot be modeled by refining a single component. These effects demand for additional connections to be introduced to the system model. Therefore, it is necessary to modify the given modules by inserting and connecting new ports and signals. In the context of SystemC this results in adding member variables to the classes representing the component's models and connecting them in the constructor code of the top-level module. We realize this direct modification of the model code using information about the code structure obtained by libClang [14].

We limit the approach to refine SystemC-AMS models in this contribution. However, the presented method is also applicable to other modeling languages such as Verilog-AMS as long as the required structural information can be extracted.

4 Accelerated Analog Simulation Using Piecewise Linearization

An accelerated simulation of AMS circuits based on piecewise-linear models has been presented in a previous work [12]. Our simulation environment focuses on analog subcircuits as shown in Sect. 1. It provides an accelerated simulation kernel for transient simulations of analog circuits. Speed-up is achieved by avoiding numerical integration and directly using the linear time-domain solution of the system. The time-domain solution is described by sums of exponential terms of the form (1), which can be efficiently evaluated during simulation.

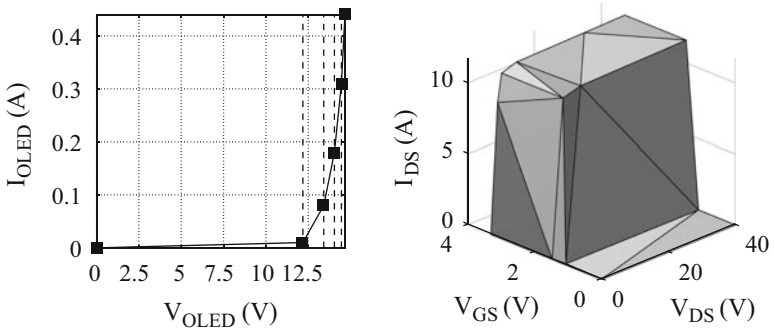

Fig. 4 Abstract piecewise-linear behavioral model of OLED and low-side switch. (**a**) OLED behavioral model with five linear sections (*straight lines*). (**b**) MOSFET behavioral model with 15 linear sections (*triangles*)

$$y_k(t) = y_{k,0} + \sum_i a_{k,i} \mathrm{e}^{\lambda_i t} \tag{1}$$

To succeed, this approach requires piecewise-constant inputs (implicitly given by the digital part of AMS circuits) and a linear or at least linearized circuit. We solve the latter problem by replacing all nonlinear devices by piecewise-linear models [16]. Exemplary PWL models for diodes (one-dimensional input: V_{OLED}) and MOS field-effect transistors (two-dimensional inputs: V_{GS} and V_{DS}) are shown in Fig. 4. The models are generated by taking advantage of geometric methods [17, 18].

The use of PWL device models results in the generation of multiple linear state-space circuit models describing the analog circuit behavior. The possible combinations of these piecewise-linear models result in a switched-linear systems [19] with different circuit states, which can also be described as a hybrid automaton [20]. It is natural that exactly one state of the automaton is valid at the same time. The modeling approach of static components (i.e., no dynamic nonlinearities) can be applied to nonlinear semiconductor devices as well as to nonlinear macro models. For example operational amplifiers or entire analog driver stages can be treated as a macro model.

Approximating a circuit by a hybrid system with linear continuous dynamics has been used before, see e.g. [21–23]. It is proven and applicable method to control the complexity of system-level modeling. Each circuit state v corresponds to a discretized state-space representation of the form (2) and (3).

$$\dot{\mathbf{x}}(t) = \mathbf{A}_v \mathbf{x}(t) + \mathbf{B}_v \mathbf{u}(t) \tag{2}$$

$$\mathbf{y}(t) = \mathbf{C}_v \mathbf{x}(t) + \mathbf{D}_v \mathbf{u}(t). \tag{3}$$

4.1 Switching Between State-Space Models

To generate simulation results it is necessary to retransform the prepared circuit state to the time domain. The time domain solution yields the monitored output functions. The computed output functions are only valid until the input excitation changes to a new (constant) value or an event was triggered by switching the valid circuit state. It should be noted that all continuity conditions (capacitor voltages and inductor currents) must be fulfilled.

Switching between different PWL models is a significant step for our simulation methodology. The switch-over time depends on the threshold voltages and currents of nonlinear components (e.g., diodes, MOSFETs, and operational amplifiers) and can be determined by finding the first root of the function

$$f(t) = V_{\mathrm{PWLdevice}}(t) - V_{\mathrm{PWLdevice_{limit}}} \tag{4}$$

in a given interval, where

- $V_{\mathrm{PWLdevice}}$ is the node voltage of a PWL device and
- $V_{\mathrm{PWLdevice_{limit}}}$ is a limit of the validity range defined by the linear section of each PWL model.

Several root-finding algorithms for such functions are known. We found that the Newton–Raphson method and the bisection method yield unsatisfactory results, as they often do not converge towards the first root and exhibit long runtimes. A specialized root-finding algorithm for this task has been presented in [11]. It guarantees to find the first root in a given interval. This means that for each valid circuit state, which is composed by the combined PWL component model states, the root-finding algorithm must be executed

$$R = 2K + 3L \tag{5}$$

times, where

- K is the number of one-dimensional models and
- L is the number of two-dimensional models.

The factors of Eq. (5) yield from the number of PWL section limits. In case of straight lines of a one-dimensional model there are two limits: one upper limit and one lower limit of each linear section. For two-dimensional models exist three limits: all edges of the triangle. After the determination of all first roots in a given interval the earliest switch-over time indicates the next valid circuit model selection.

Figure 5 shows a comparison between our simulation approach with existing analog circuit simulators. Instead of performing numerical integration, linearization and solving the system of equations during each time step, our approach is only sensitive to input changes and internal model switching. As mentioned before, a circuit model switch is triggered by a transition from one linear section of a PWL

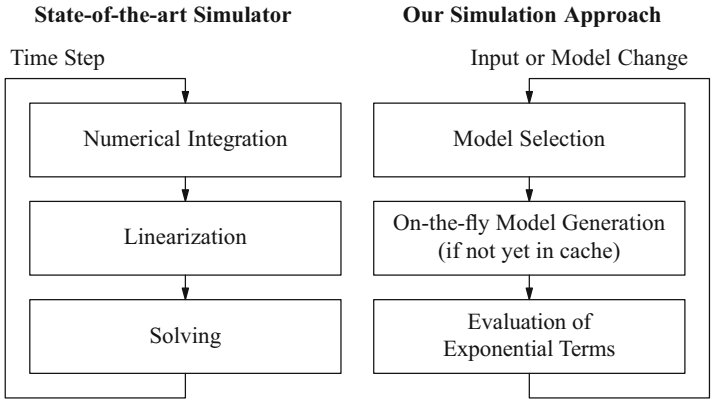

Fig. 5 Comparison between the state of the art and our analog simulation flow

component to another. The active linear section, selected by the simulator kernel, remains valid as long as no change at any input occurs and the circuit does not exceed the current section due to its dynamics. In case of an input change, a new valid section must be calculated along with its new initial values to satisfy the continuity of inductor currents and capacitor voltages. In contrast to an input change, the dynamics of the circuit make switching to an adjacent linear section necessary.

4.2 Parallelizing the Specialized Root-Finding Algorithm

Most of today's advancements in the computing power of processors result from higher levels of parallelization. The processing power available to sequential working threads grows comparably slowly. A sequential program can only use a fraction of the theoretical processing power of most state-of-the-art processors. Therefore, parallelization is of growing importance for performance-critical applications like simulations.

Simulating a large analog circuit including a large number of nonlinear devices causes a long runtime. The root-finding algorithm must be executed very often which can be seen from Eq. (5). The determination of all first roots can be processed independently within a simulation run. For such cases the computations can be run in parallel to speed-up the simulation.

We demonstrate our simulation approach for a scalable nonlinear transmission line (NLTL) with N stage, see Fig. 6.

In case of a NLTL_{128} for each circuit model change the root-finding algorithm must be executed 256 times. Table 1 shows different simulation speed-ups caused by different degrees of parallelization within the simulation kernel.

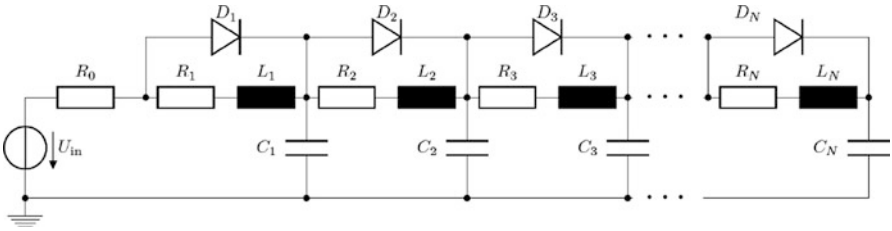

Fig. 6 Example of nonlinear transmission line (NLTL_N)

Table 1 Simulation runtimes of circuit NLTL_{128}

	Tend	Speed-up
Our approach (single threaded)	74.87 s	1.00×
Our approach (multi threaded: 2 threads)	43.01 s	1.74×
Our approach (multi threaded: 4 threads)	26.43 s	2.83×
Our approach (multi threaded: 8 threads)	17.26 s	4.34×
Our approach (multi threaded: 16 threads)	14.73 s	5.08×

In case of the demonstrated buck converter circuit merely including three nonlinear element the parallelization approach is unfortunate. We found that due to the additional parallelization overhead multi-threaded simulations are profitable only for larger circuits.

5 System Acceptance Regions

Acceptance and fault regions are well known in Integrated Circuit (IC) testing, e.g., in *Shmoo plotting* [24]. The concept is shown in Fig. 7: For each point in the parameter space, the corresponding system's behavior is classified into *correct* or *incorrect* using simulations or measurements. The *system acceptance region*, i.e., the typically continuous region with correct behavior, represents all parameter combinations ensuring the system's function. Note that the border between acceptance and fault region might be *fuzzy* due to stochastic influences on the system (e.g., noise sources) [2].

In this contribution, we assume the system model's parameters to live in an acceptance region. Still, additional parameters not included in this model might cause the final system to fail. Extracting these cases in a conventional verification is hardly possible due to a missing specification value to test for and a missing model to test with.

These critical scenarios emerge, if an additional effect severely interacts with the parameters included in the simulation. This situation is shown in Fig. 7: The system with parameters p_{1s}, p_{2s}, and p_{3s} is in the acceptance region taking only the first parameters into account as shown in Fig. 7a. The additional effect with parameter p_3 results in incorrect system behavior due to its interaction shown in Fig. 7b. Hence,

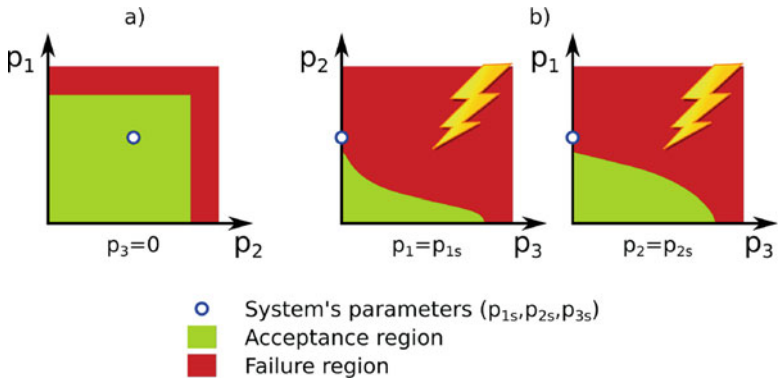

Fig. 7 System acceptance regions example: a system may live in an acceptance region for two parameters p_1 and p_2 but may be distorted by an additional effect with parameter p_3. (**a**) Without p_3. (**b**) Acceptance region with p_3

it is important to know the shape of the SAR: Based on the knowledge of the shape, an additional test can be implemented to reduce the design risk. Additionally, the shape unveils the interactions between different parameters. In the figure, the region in the first plot could be parameterized independently for each p_1 and p_2. The other plots show interactions between parameters, i.e., the description of the regions must include this relationship.

Our goal is to extract the acceptance regions for effects not present in the system-level model by using automated model refinement as introduced in Sect. 3. Based on this, we propose an extraction flow as shown in Fig. 8. After analyzing the given model of the system, the components are refined with the targeted properties. For performance reasons, the analog portions are substituted by PWL models as described in Sect. 4. The extraction of the SAR is done by sampling of the newly introduced and possibly multidimensional parameter space. Since naive sampling strategies are clearly not feasible in higher dimensions, an adaptive strategy is needed. We utilize the *Bordersearch* algorithm described by Dobler et al. [2] for reducing the number of parameter combinations to be simulated.

This algorithm aims to model the border between acceptance and fail regions. Based on this, it automatically selects the parameter combinations to be evaluated close to this border. This adaptive sampling significantly improves exploration quality and runtime especially for high-dimensional problems.

6 Application Scenario

For demonstration, we examine the design of a hysteretic buck converter [1] for driving organic LEDs (OLED, organic light emitting diode) shown in Fig. 2. As a first step, the system-level model is created to verify properties of the functional

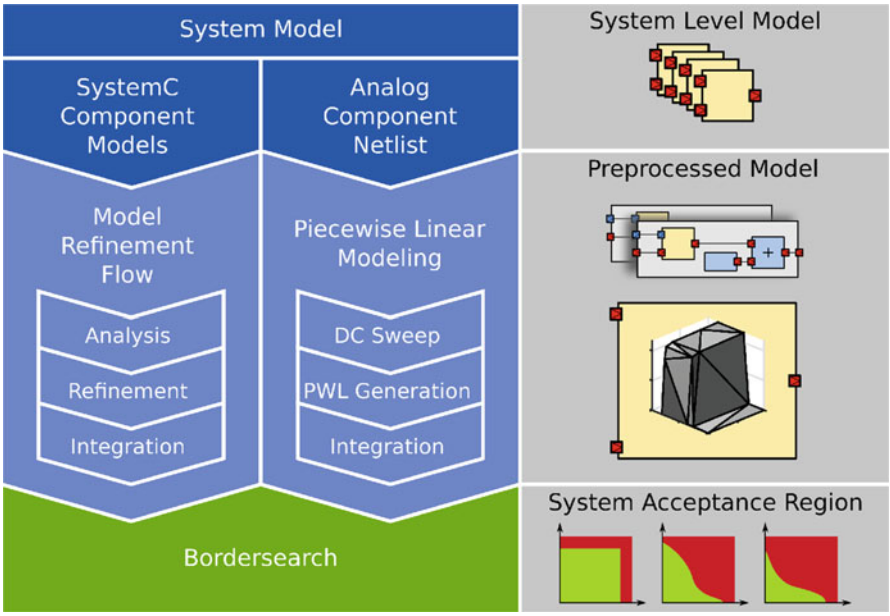

Fig. 8 System acceptance region extraction flow

behavior—in this case, the stability of the overall system. In this context, short circuit between supply and ground through S_1 and S_2 must not occur and the switching frequency must be limited to a certain value. This has been done, e.g., by Dietrich et al. [1] with idealized components. This autonomously switching, externally controlled system has not yet been analyzed formally under influence of low-level parasitic effects. We want to evaluate this question by simulation, since a fabricated test chip for finding out the most critical effects is clearly unwanted for economical reasons (Fig. 9).

We create the simulation setup as shown in Fig. 10 using the methodology presented before. The reference voltage generators (reference and DAC circuits) are refined by additional effects. Additive white noise is used here to model various distortions. A testbench block generates stimulus signal exercising the system in usual use-cases while observing internal signals. The observed waveforms are checked by the testbench for instable behavior.

To assure the stability of the final system, we explore the acceptance regions for all additional parameters. The simulation runtimes for different exploration strategies are shown in Table 2. Both *Bordersearch* and PWL simulation make this exploration feasible with rather high accuracy. For rough estimates, the simulation runs can be reduced even further. The estimated computing time for normal equidistant sampling of the parameter space is clearly not applicable due to the extremely high number of points to be simulated.

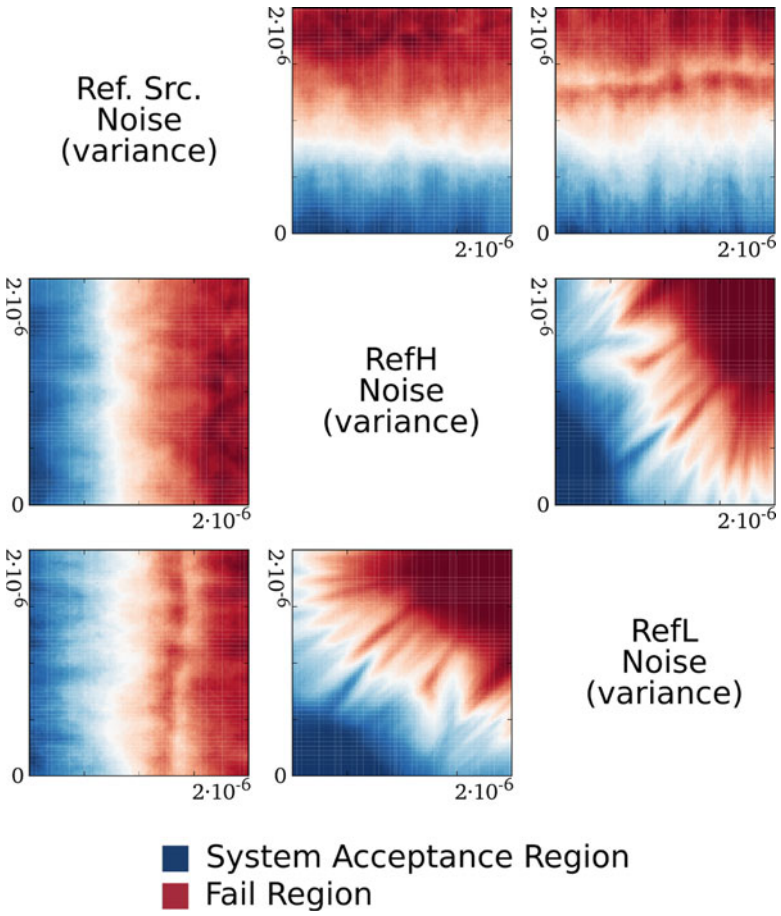

Fig. 9 Projection of 3D system acceptance region of noise-variances of distortion models

In Fig. 9, we show 2D-projections of the estimated three-dimensional acceptance regions for illustration. Here, we examined the variance of the annotated noise-effects as parameters between 0 and 2×10^{-6}. The axis labels are given by the horizontal or vertical captions. For example, the upper-right plot axis are given by *RefL Noise (variance)* as *x*-axis and *Ref. Src. Noise (variance)* as *y*-axis. The region exhibits typical behavior for noise-effects: The border between acceptance and fail regions are not sharp and the projection of the 3D SAR to 2D also contributes to this effect. In the design process, these regions give the designer a deep insight into the impact of effects to the system, as for instance the interaction distortions of RefH and RefL. This provides the designer with the knowledge for extending the specification and verification plan with checks for the position in the parameter space. Moreover, possibly occurring trade-offs in the design can be evaluated at a very early point in the design process for enhancing the overall system performance.

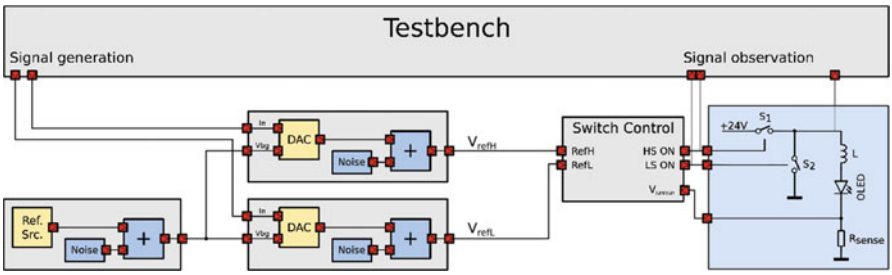

Fig. 10 Preprocessed model of hysteretic buck converter circuit for acceptance region exploration

Table 2 cpu time of acceptance region exploration based on different simulation approaches and sampling strategies

	Reference	Sampling	Bordersearch
Number of points	1	1×10^6	1×10^4
CPU time (our approach)	2.09 s	580 h	5.8 h
CPU time (Saber)	54.3 s	628 d (est.)	6.28 d (est.)

7 Conclusion

In this contribution, we addressed the challenges in virtual prototyping of AMS systems achieving first-time right designs. We proposed a method for automated modeling of extra-functional effects for efficient system-level simulations. It provides system acceptance regions for AMS systems using automated refinement of component models. Since the simulation runtime needed for this process is very high even for small AMS systems, we integrated an accelerated and parallelized simulation approach. Applying the proposed method in later design phases is also possible but challenges the method by even higher complexity due to more signals and possible effects.

For demonstrating our methodology, we examined a DC–DC converter circuit. In this circuit, we regarded exemplarily distortions to generated reference voltages to evaluate their impact on the system's stability. The extracted acceptance regions show interactions between the effects introduced in the refinement process. This provides the design and the verification engineers with information about critical scenarios and crucial points to avoid or test for.

In future research, this information could be extracted in a more automated way by examining the shape of these regions. This can also be used to reduce the dimensionality of the parameter space to be explored: If the interactions are known, they could be possibly treated in several groups separately. Even a ranking of the criticality of extra-functional effects can be realized.

References

1. Dietrich, S., Sandner, H., Vanselow, F., Wunderlich, R., & Heinen, S. (2012). In *2012 IEEE 10th International New Circuits and Systems Conference (NEWCAS)* (pp. 369–372). doi:10.1109/NEWCAS.2012.6329033.
2. Dobler, M., Harrant, M., Rafaila, M., Pelz, G., Rosenstiel, W., & Bogdan, M. (2015). *2015 Design, Automation Test in Europe Conference Exhibition (DATE)* (pp. 1036–1041).
3. Kundert, K., Chang, H., Jefferies, D., Lamant, G., Malavasi, E., & Sendig, F. (2000) *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(12), 1561. doi:10.1109/43.898832.
4. Alassir, M., Denoulet, J., Romain, O., & Garda, P. (2013). *IEEE Transactions on Components, Packaging and Manufacturing Technology, 3*(12), 2081. doi:10.1109/TCPMT.2013.2262151. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6531653.
5. Bai, X., & Dey, S. (2004). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23*(9), 1355. doi:10.1109/TCAD.2004.833612. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1327675.
6. Barke, E., Fürtig, A., Gläser, G., Grimm, C., Hedrich, L., Heinen, S., et al. (2016). *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*.
7. Eo, Y., Shin, S., Eisenstadt, W. R., & Shim, J. (2002). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21*(12), 1489. doi:10.1109/TCAD.2002.804381.
8. Leveugle, R., & Ammari, A. (2004). *Proceedings Design, Automation and Test in Europe Conference and Exhibition, 2004* (Vol. 1, pp. 590–595). doi:10.1109/DATE.2004.1268909.
9. Rafaila, M., Decker, C., Grimm, C., Kirscher, J., & Pelz, G. (2010). *2010 Forum on Specification and Design Languages (FDL 2010)* (pp. 1–6). London: IET.
10. Stehr, G., Graeb, H. E., & Antreich, K. J. (2007). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26*(10), 1733. doi:10.1109/TCAD.2007.895756.
11. Zaum, D., Hoelldampf, S., Olbrich, M., Barke, E., & Neumann, I. (2010). *2010 Forum on Specification Design Languages (FDL 2010)* (pp. 1–6). doi:10.1049/ic.2010.0158.
12. Hoelldampf, S., Zaum, D., Neumann, I., Olbrich, M., & Barke, E. (2011). *2011 IEEE International Systems Conference (SysCon)* (pp. 527–530). doi:10.1109/SYSCON.2011.5929046.
13. Barnasconi, M., Einwich, K., Grimm, C., Maehne, T., Vachoux, A., et al. (2013). *Standard system ams extensions 2.0 language reference manual*. Accellera Systems Initiative (ASI).
14. clang: a C language family frontend for LLVM. http://clang.llvm.org/.
15. Mako Templates for Python. http://www.makotemplates.org/.
16. Hoelldampf, S., Lee, H. S. L., Zaum, D., Olbrich, M., & Barke, E. (2012). *Proceedings of IEEE International SOC Conference (SOCC)*.
17. Garland, M., & Heckbert, P. S. (1998). *Proceedings of IEEE Visualization (Vis)* (pp. 263–269)
18. Lindstrom, P., & Turk, G. (1999). *IEEE Transactions on Visualization and Computer Graphics, 5*(2), 98.
19. Bemporad, A., Ferrari-Trecate, G., Morari, M., et al. (2000). *IEEE Transactions on Automatic Control, 45*(10), 1864.
20. Lee, H. S. L., Althoff, M., Hoelldampf, S., Olbrich, M., & Barke, E. (2015). *2015 20th Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 725–730). Piscataway, NJ: IEEE.
21. Chua, L., & Deng, A. C. (1986). *IEEE Transactions on Circuits and Systems, 33*(5), 511. doi:10.1109/TCS.1986.1085952.
22. Chen, W. K. (2009). *Feedback, nonlinear, and distributed circuits* (3rd ed.). Boca Raton, FL: CRC Press/Taylor and Francis.
23. Zhang, Y., Sankaranarayanan, S., & Somenzi, F. (2012). *Formal methods in computer-aided design (FMCAD), 2012* (pp. 196–203).
24. Baker, K., & von Beers, J. (1996). *Proceedings of the International Test Conference, 1996* (pp. 932–933). doi:10.1109/TEST.1996.557162.

Designing Reliable Cyber-Physical Systems

Gadi Aleksandrowicz, Eli Arbel, Roderick Bloem, Timon D. ter Braak, Sergei Devadze, Goerschwin Fey, Maksim Jenihhin, Artur Jutman, Hans G. Kerkhoff, Robert Könighofer, Shlomit Koyfman, Jan Malburg, Shiri Moran, Jaan Raik, Gerard Rauwerda, Heinz Riener, Franz Röck, Konstantin Shibin, Kim Sunesen, Jinbo Wan, and Yong Zhao

Abstract Cyber-physical systems, that consist of a cyber part—a computing system—and a physical part—the system in the physical environment—as well as the respective interfaces between those parts, are omnipresent in our daily lives. The application in the physical environment drives the overall requirements that must be respected when designing the computing system. Here, reliability is a core aspect where some of the most pressing design challenges are:

- monitoring failures throughout the computing system,
- determining the impact of failures on the application constraints, and
- ensuring correctness of the computing system with respect to application-driven requirements rooted in the physical environment.

This chapter gives an overview of the state-of-the-art techniques developed within the Horizon 2020 project IMMORTAL that tackle these challenges throughout the stack of layers of the computing system while tightly coupling the design

G. Aleksandrowicz • E. Arbel • S. Koyfman • S. Moran
IBM Research Lab, Haifa, Israel

R. Bloem • R. Könighofer • F. Röck
Graz University of Technology, Graz, Austria

T.D. ter Braak • G. Rauwerda • K. Sunesen
Recore Systems, Enschede, The Netherlands

S. Devadze • A. Jutman • K. Shibin
Testonica Lab, Tallinn, Estonia

G. Fey • J. Malburg • H. Riener
German Aerospace Center, Bremen, Germany

M. Jenihhin • J. Raik (✉)
Tallinn University of Technology, Tallinn, Estonia
e-mail: jaan.raik@ttu.ee

H.G. Kerkhoff • J. Wan • Y. Zhao
University of Twente, Enschede, The Netherlands

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_2

methodology to the physical requirements. (The chapter is based on the contributions of the special session *Designing Reliable Cyber-Physical Systems* of the *Forum on Specification and Design Languages* (FDL) 2016.)

Keywords Adaptive test strategy generation • Automatic test case generation • Checker minimization • Checker qualification • Concurrent online checkers • Counterexample-guided inductive synthesis • CPS • Cross-layered fault management • Cyber-physical systems • Dependable CPSoC • Embedded systems • Fault classification • Fault management infrastructure • Fault tolerance • Gating-aware error injection • Gradual degradation • Health monitors • Heterogeneous • IDDQ • IEEE 1687 • Many-core • NBTI aging • Parameter synthesis • Reliability analysis • Resource management software • Run-time resource mapping • Satisfiability modulo theories • System-on-chip

1 Introduction

Cyber-physical systems (CPS) [30] are smart systems that integrate computing and communication capabilities with the monitoring and control of entities in the physical world reliably, safely, securely, efficiently, and in real-time. These systems involve a high degree of complexity on numerous scales and demand for methods to guarantee correct and reliable operation. Existing CPS modeling frameworks address several design aspects such as control, security, verification, or validation, but do not deal with reliability or automated debug aspects.

Techniques presented in this chapter are developed in the EU Horizon 2020 project IMMORTAL.[1] These techniques target reliability of CPS throughout several abstraction layers during design and operation, considering fault effects from different error sources ranging from design bugs via wear-outs and soft errors towards environmental uncertainties and measurement errors [3].

We consider the cyber part of CPS at four layers of abstraction shown in Fig. 1. The *analog/mixed-signal (AMS) layer* models the components, especially sensors and actuators, using Matlab/Simulink or VHDL-AMS. In this layer we focus on the aging behavior of the design. Thermal and electrical stress can degenerate sensor quality, actuator quality, or reduce the overall performance characteristics of the design. In Sect. 2 we present a health monitoring approach to warn the system early if functional parts of the system degenerate to such an extent that reliable operation can no longer be ensured and, e.g., redundant components must be activated.

At the *digital hardware layer* the CPS is either described at the *Register Transfer* (RT)-level, e.g., in a synthesizable subset of VHDL or Verilog, or as gate-level netlists. At this layer the analog signals of the analog/mixed-signal layer are abstracted to binary values. During operation of the CPS, even correctly designed

[1] Integrated Modelling, Fault Management, Verification and Reliable Design Environment for Cyber-Physical Systems, http://www.h2020-immortal.eu.

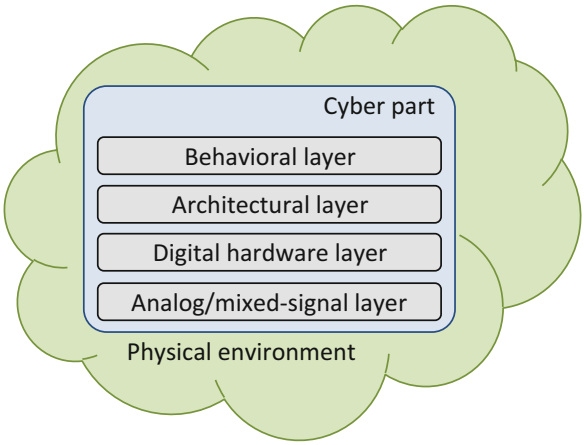

Fig. 1 The stack of layers of a CPS

systems may behave incorrectly, e.g., radiation may change values in latches or change the signal level in wires. Such effects are called soft errors and appear as bit-flips at the digital hardware layer. Error detection codes, e.g., parity bits, or *Error Correction Codes* (ECC), e.g., Hamming codes, are used to mitigate soft errors. In Sect. 3 we present approaches to automatically detect storage elements that are not protected by error detection or error correction codes or prove that storage elements are protected. Moreover, Sect. 4 provides advanced online-checker technology beyond traditional ECC schemes achieving full fault coverage.

At the *architectural layer*, we consider the CPS as a set of computational units with different capabilities, a communication network between those computational units, and a set of tasks that are described at a high level of abstraction, which should be executed on the CPS. Section 5 proposes an infrastructure for reading out the information about occurrences of faults in the lower layers and accumulating this information for preventing errors resulting from those faults. Section 6 explains how to use this infrastructure to (re)allocate and (re)schedule resources and tasks of the CPS if a computational unit can no longer provide reliable operation. As a result the CPS is enabled for fault-tolerant operation.

The *behavioral layer* considers the functional behavior and tasks of the CPS. The elements at this layer are modeled as behavioral descriptions of the system's functionality and can be realized either in software or in hardware. In Sect. 7 we consider the generation of test strategies from a system's specification given as temporal logic formulæ. Here, we focus on specifications which are agnostic of implementations and allow freedom for the implementation. Therefore the generated test cases must be able to adapt to different implementations. In Sect. 8 we present an approach to automatically synthesize parameters for behavioral descriptions of a CPS. The parameter synthesis approach can be used to assist a designer in finding suitable values for important design parameters such that given requirements are met, eliminating the need for manual error prone decisions.

2 Health Monitoring at the Analog/Mixed Signal Layer

CPS have to cope with analog input and provide analog output signals in the physical world, and be able to carry out computational tasks in the digital world. Practice has shown that major problems in terms of failures occur in the analog/mixed-signal part, which includes (on-chip) sensors and actuators. In contrast to the digital world, the (parametric) faults in the analog/mixed-signal parts of a CPS are much more complex to detect and repair.

In the case of wear-out, e.g., resulting from *Negative-Bias Temperature Instability* (NBTI) [51], it has been shown that analog stress signals cause different wear-out results as compared to digital ones, leading to more sophisticated NBTI models. The NBTI aging mechanism usually results in increased delay times (lower clock frequencies) in pure digital systems [54] while in analog/mixed-signal systems several key system-performance parameters will change, like for instance the offset voltage in OpAmps and data converters [50]. Experiments have also shown that drift of sensors [53] and actuators are often key parameters to cause faulty behavior in a CPS as a result of aging.

Stress voltages, stress temperatures, and duration of them (mission profile) are the principal factors of wear-out. Hence, in the case of a real CPS, these stress parameters must be measured during life-time and subsequently handled as mission profiles cannot be predicted accurately in advance. A combination of environmental *Health Monitors* (HMs) [4] and key performance parameters [50], nowadays implemented as embedded instruments, are required for this purpose. Temperature, voltage, and current health monitors, as well as gain, offset, and delay monitors have been developed for this purpose. It is obvious that these embedded instruments should be extremely robust against aging and variability.

In the new generation of CPS, these embedded instruments will be connected by the new IEEE 1687 standard [4]. The embedded instrument will consist in that case of the original *raw* instrument and the IJTAG *wrapper* part. An example of an IJTAG-compatible I_{DDT} health monitor [24] is shown in Fig. 2. It is related to the well-known reliability-sensitive quiescent power supply I_{DDQ} measurements. The embedded instrument consists of a current-to-voltage conversion, remaining as close to V_{DD} for the core under test as it is possible. As the resulting voltages are small, several amplification stages are required after this. The last step is the conversion to a 14-bits digital word, via the frequency. In addition several supporting circuits are required, like controller and samples memory.

In order to obtain highly dependable CPS, which includes reliability, availability, and maintainability [26], more than just health monitors and embedded instruments are required. It also includes software and computational capabilities to extract the correct information from the HMs, and calculate the remaining lifetime of *Intellectual Property* (IP) components being part of a CPS from that [54]. Useful HMs for the digital cores have shown here to be I_{DDQ} and I_{DDT} embedded instruments as well as delay monitors.

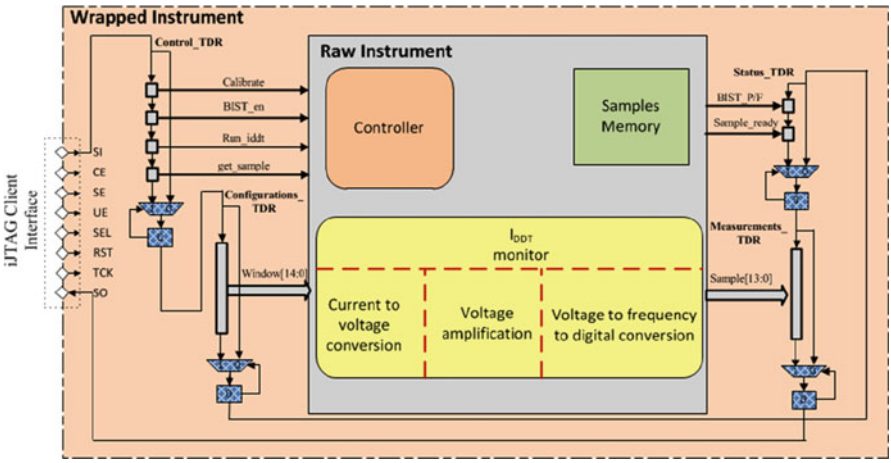

Fig. 2 An IJTAG-compatible I_{DDT} health monitor for lifetime prediction

For digital systems, like multi-core processor *System-on-Chips* (SoCs) this platform is already well on the way. To know the remaining life-time is essential in dependable CPS, as many applications are safety-critical and hence do not allow *any* down-time to ensure high availability. Existing digital systems have already been shown to be capable to react *after* a failure has occurred, mainly by the use of pseudo online *Build-In Self Test* (BIST) of processor cores. In addition, the (on-chip) repair in the case of multi-core processor SoCs has been successfully accomplished by shutting down the faulty core and replace it by a spare processor core, or increase the workload of a partly-idle processor core.

In the case of the analog/mixed-signal part of a *CPS-on-Chip* (CPSoC), the situation is much more difficult. Phenomena like NBTI aging result in this case in changing key system parameters of IPs (OpAmps, filters, ADCs, and DACs), like offset, gain, and changing frequency behavior. Using our new analog/mixed-signal NBTI model in our local designs of 65 and 40 nm TSMC OpAmps and SAR-ADCs, higher-level system key parameters were derived which were used subsequently in a Matlab environment. Figure 3 shows four possible degradation scenarios, as well as the application of our two-stage repair approach. First, key parameters are monitored and digitally tuned if changing; when the maximum tuning range is accomplished, a bypass and spare IP counter action is carried out.

One can see from the figure that the CPSoC remains within its green boundaries (of parameter P) of correct operation. The figure also shows that the different degradation mechanisms trigger tuning and replace counter measures at different times. The dependability improves by several factors at the cost of more sophisticated health monitors, software and embedded computational resources, all translating into more silicon area.

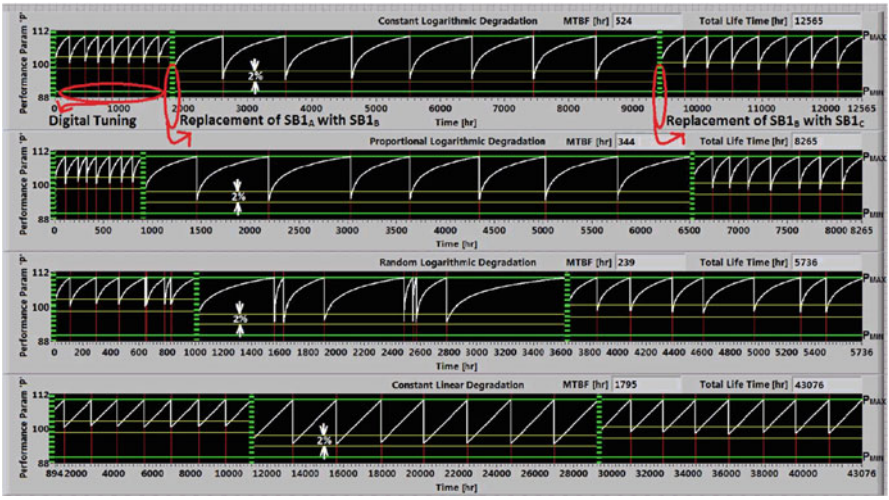

Fig. 3 Simulation of a redundant and digital IP tuning platform for highly dependable mixed-signal CPS system for four different degradation scenarios

3 Comprehensive and Scalable RT-Level Reliability Analysis

The dependability of CPS crucially depends on the reliability and availability of their digital hardware components. Even if all digital hardware components are free of design bugs, they may still fail at run-time due to, e.g., environmental influences such as radiation or aging and wear-out effects that result in occasional misbehavior of individual hardware components. In the following we subsume such transient errors under the term *soft error* [33].

A common approach to achieve resiliency against soft errors adds circuitry to automatically detect or even correct such errors [35]. This can be achieved by including redundancy, e.g., in the form of parity bits or more sophisticated error detection or correction codes [33]. Soft error reporting in the RT level can be done by using *error checkers*. Once a soft error is reported by a checker, it is up to the *Fault Management Infrastructure* (FMI) to decide how to react to this transient fault.

Therefore, the ability to understand the reliability of a given hardware component in a CPS becomes a key aspect during the component design phase. In order to cope with the ever shrinking design cycles it is highly desired that this analysis is performed in pre-silicon. Many methods for pre-silicon resiliency analysis have been proposed. These methods can be roughly classified into two categories: simulation-based methods, e.g., [22, 28, 31, 32], and formal methods, e.g., [16, 27, 43]. At the heart of the simulation-based methods lies the concept of error injection. In this approach the design is simulated and verified for robustness in the presence of transient faults injected deliberately during simulation. This approach is workload-dependent and achieves low state and fault coverage due to the enormous state space

size. In an attempt to alleviate the coverage issues of the simulation-based approach formal methods have been suggested. A common practice in this approach is to perform formal verification using a fault model which models single event upsets. Being applied monolithically, this approach suffers from capacity limits inherent to formal verification methods which makes it impractical in many real-life industrial cases.

Many hardware mechanisms used for soft error protection are local in their nature. For example, parity-based protection, Error Correction Code (ECC) logic, and residue checking mechanisms are all examples of design techniques aimed at protecting relatively small parts in the design, referred to as *protected structures*. An *error detection signal* is a Boolean expression that is assigned true when an error has occurred. A *protected structure* consists of an error checker fed by error detection signals, of *protected sequential elements* and of various *gating conditions* on the way to the checker. Gating conditions are required in high performance designs to turn off reliability checks when certain parts of the logic are not used.

Based on the locality of the protected structures, we propose a novel approach for reliability analysis and verification, a basic version of which was presented in [7]. We divide the reliability verification process into an *analysis stage* and a *verification stage*. In the *analysis stage* the local protection structures are identified, and in the *verification stage* it is verified that the protection structures work properly. There are aspects of the verification that can be proved with formal verification, e.g., it can be proved formally that a certain sequential element is protected by a certain checker under certain gating conditions [7]. Since each protection is local in its nature, applying formal techniques is scalable. Other aspects may require dynamic simulation; for example, proving that the gating conditions are not over-gating the protected structure [6]. In the following we provide an overview of our new approach, and give a glimpse at the technical "how."

3.1 Analysis Stage

The analysis stage identifies the protected structures and is divided into two substages. The *identification of error detection signals* stage and the *structural analysis* stage.

Identification of Error Detection Signals In this stage the building blocks of the error detection and correction logic are identified. For this purpose we use the error checkers as anchors and employ formal and dynamic methods to accurately and efficiently identify various error detection constructs. An example for parity checking identification is described in [7]. Other examples of error detection logic that can be identified accurately and locally in this stage are residue and one-hot checking. Error correction code, however, doesn't need to be connected to error checkers. To detect ECC computation we rely on the fact that we are looking for linear ECC, a computation of the form $v = Au$ for vectors v, u over $\mathbb{Z}_2$. To achieve

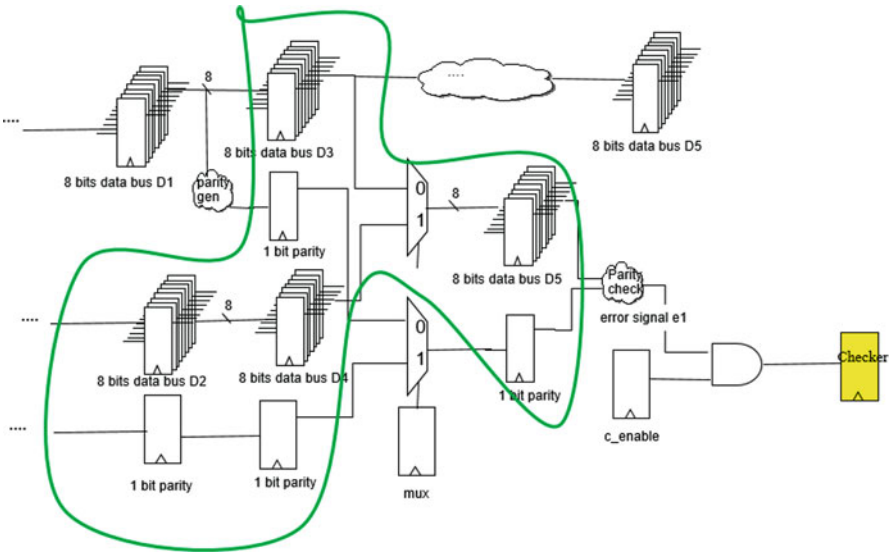

Fig. 4 A parity protected structure

this, we iterate over all the vectors in the design and identify the vectors with bits that are leaves of a XOR-computation tree. After discovering an ECC-like matrix we first purge non-ECC instances, such as the case of the identity matrix, or matrix with too many one-hot columns indicating that most input bits are used only once. We determine whether this is an ECC generation or an ECC check by searching for a unit submatrix with dimensions corresponding to the output size.

Structural Analysis In order to identify the protected structure of each error detection signal, we analyze the topology of the netlist representing the design. The objective is to identify the set of sequential elements protected by an error detection signal. The challenge here is twofold:

- Understating the boundaries of the protection, e.g., if the protection is parity-based, the parity generation logic and the parity checking logic form the boundary of the protected sequential elements.
- Proper identification of the corresponding gating logic.

For example, in Fig. 4 the protected sequential elements are the encircled ones, plus more sequential elements connected to data bus *D*2 and the corresponding parity bit which are out of scope. Specifically, the *c_enable* signal is the gating condition of the error detection signal—an erroneous parity check will make the error checker fire only if the value of the sequential element is 1, and this sequential element is not a part of the protected structure; similarly, the *mux* signal is not protected. Also, the data bus *D1* is located before the parity generation logic and thus is not a part of the protected structure either. Due to lack of space, the full algorithm for detecting

the protected structure, will not be provided here. However, we will give a glimpse of the way the algorithm copes with the above challenges.

Consider a parity protected structure. When the parity generation is in the scope of the given netlist, then the boundary of the parity protection can be easily identified by detecting the parity generation. Moreover, in this case the protected data bus and parity bits are the intersection of the input cone of the parity check and the output cone of the parity generation, whereas the gating conditions are not in that intersection. Hence, the boundaries of the protection and the gating condition logic can be identified quite easily.

However, when the parity generation is not in the scope of the given netlist, it is more difficult to distinguish between the protected structure and gating conditions. In industrial systems this situation is quite common. In order to distinguish between data and gating conditions in such cases, the analysis is performed on the *parse tree* which represents more clearly the designer intent than the corresponding Boolean logic representation. Consider the following assignment for a vector bus *data*:

$$\text{data}(0\ldots7) \Leftarrow \text{data}_1(0\ldots7) \text{ when } \text{cond}_1 \text{ else } 00000000$$

It is quite easy to understand from the parse tree that cond_1 is a gating condition, while $\text{data}_1(0\ldots7)$ is the data source, while it is more challenging to infer the same from a set of logical assignments of the form

$$\text{data}(i) \Leftarrow \text{data}_1(i) \wedge \text{cond}_1$$

Moreover, it is impossible to distinguish between data source and the gating condition when a statement

$$\text{bit}_1 \Leftarrow \text{bit}_2 \text{ when cond else } 0$$

is represented by a logical equivalent

$$\text{bit}_1 \Leftarrow \text{bit}_2 \wedge \text{cond}$$

Therefore, in order to cope with the above challenge we perform the analysis using the parse tree.

3.2 Verification Stage

At this stage we verify that the constructs found at the earlier stage indeed protect the relevant sequential elements. The verification that is required here has two aspects: (a) verifying that under the relevant gating conditions the sequential elements are indeed protected by the corresponding error detection signals or error correction

logic; (b) verifying that the gating conditions are not over-gating and will not prevent a checker from firing when it should, causing silent data corruption.

For the former, formal verification can be used, leveraging the locality of protection structures. In [7] we perform it for simple parity protection. More research is still required to expand the approach from [7] to include other protection types and more complex parity structures.

The challenge in the latter verification aspect is that in order to verify that the gating conditions are not over-gating a global scope is required, since the gating conditions can be dependent on various parts of the design. In [6] we present a novel and effective approach to verify that the gating conditions are not over-gating. We use the identification of the analysis stage to synthesize drivers that perform smart *gating aware* error injection. These drivers are then integrated in the standard functional verification environment existing for any industrial system.

4 Qualification and Minimization of Concurrent Online Checkers

Besides standard approaches for fault detection we also consider advanced error detection schemes on the digital hardware layer for CPS. Particularly, the proposed online checkers enable cost-efficient mechanisms for detecting faults during lifetime of the state-of-the-art many-core systems. These mechanisms must detect errors within resources and routers as well as enable reconfiguration of the routing network in order to isolate the problem and provide graceful degradation for the system.

Our approach [41, 42] exceeds the existing state of the art in concurrent online checking by proposing a tool flow for automated evaluation and minimization of the verification checkers. We show that starting from a realistic set of verification assertions a minimal set of checkers are synthesized that provide 100% fault coverage with respect to single stuck-at faults at a low area overhead and the minimum fault detection latency of a single clock-cycle. The latter is especially crucial for enabling rapid fault recovery in reliable real-time systems.

An additional feature of the proposed approach is that it allows formally proving the absence or presence of true misses over all possible valid inputs for a checker, whereas in the case of traditional fault injection only statistical probabilities can be calculated without providing the user with full confidence of fault detection capabilities. The formal proof as well as the minimal fault detection latency is guaranteed by reasoning on a pseudo-combinational version of the circuit and by the application of an exhaustive valid set of input stimuli as the verification environment.

The checker qualification and minimization flow starts with synthesizing the checkers from a set of combinational assertions. Thereafter, a pseudo-combinational circuit is extracted from the circuit of the design under checking. The pseudo-

combinational circuit is derived from the original circuit by breaking the flip-flops and converting them to pseudo primary inputs and pseudo primary outputs. Note that, at this point, additional checkers that also describe relations on the pseudo primary inputs/outputs may be added to the checker suite in order to increase the fault coverage.

Subsequently, the checker evaluation environment is created by generating exhaustive test stimuli for the extracted pseudo-combinational circuit. These stimuli are fed through a filtering tool that selects only the stimuli that correspond to functionally valid inputs of the circuit. As a result, the complete valid set of input stimuli that serve as the environment for checker evaluation is obtained. The obtained environment, pseudo-combinational circuit, and synthesized checkers are applied to fault-free simulation. The simulation calculates fault-free values for all the lines within the circuit. Additionally, if any of the checkers fires during fault-free simulation, it means a bug in the checker or an incorrect environment.

If none of the checkers is firing in the fault-free mode, then checker evaluation takes place. The tool injects faults to all the lines within the circuit one-by-one and this step is repeated for each input vector. As a result, the overall fault detection capabilities for the set of checkers in terms of fault coverage metrics are calculated. In addition, each individual checker is weighted by summing up the total number of true detections by the checker. Finally, the weighting information is exploited in minimizing the number of checkers, eventually allowing to outline a trade-off between fault coverage and the area overhead due to the introduction of checker logic.

Experiments carried out on the control part (routing and arbitration) of a *Network-on-Chip* (NoC) router showed on a realistic application the feasibility and efficiency of the framework and the underlying methodology. Experimental results showed that the approach allowed selecting the minimal set of 5 checkers out of 31 verification assertions with the fault coverage of 100% and area overhead of only 35% [41, 42].

5 Managing Faults at SoC Level During In-Field Operation of CPS

When a fault occurs during in-field operation in a complex SoC within a CPS, which is working under the control of the software, it is necessary that the latter becomes aware of the fault and reacts to it as quickly as possible. The SoC management software, e.g., *Operating System* (OS), must then take actions to isolate and mitigate the effects of the fault. These actions include fault localization, classification based on diagnostic information, and proper handling of affected resources and tasks by the OS. This implies a cross-layer *Fault Detection, Isolation, and Recovery* (FDIR) procedure, since the faults can be detected on the hardware layer, and recovery actions can be taken throughout the stack of layers.

5.1 Fault Management Infrastructure

In order to deliver the information from the instruments, store health and statistics information, and provide the required inputs to the OS, the SoC contains the FMI which consists of both hardware and software side.

We propose a hierarchical in situ FMI (see Fig. 5) with low resource overhead and high flexibility during operation. IEEE 1687 IJTAG is used as a backbone of FMI to implement a hierarchical instrumentation and monitoring network for efficient and flexible access to the instruments which are attached to the monitored resources. The main benefit of using IEEE 1687 IJTAG infrastructure for in situ fault management is based on considerable reuse of existing test and debug infrastructure and instrumentation later in the field for the new purpose of fault management. In our architecture, traditional IJTAG is extended with asynchronous fault detection signal propagation to significantly improve the fault detection latency.

Fault Manager (FM) is a part of OS (kernel) which is responsible for updating both health and resource maps. If a fault is detected in the system, FM must start a diagnostic procedure to find out the location of the fault as precisely as possible. This location information must be reflected in the *Health Map* (HM) by setting the fault flag for the appropriate resource and updating the fault statistics.

Instrument Manager (IM) is a hardware module which is responsible for the communication with the instruments through IJTAG network. It informs FM about fault detections and provides the read/write access to the instruments.

Health map (HM) is a data structure in a dedicated memory which holds the detailed information about the faults and the fault statistics. HM is the run-time model of CPS including fault monitors and implements a structural view of

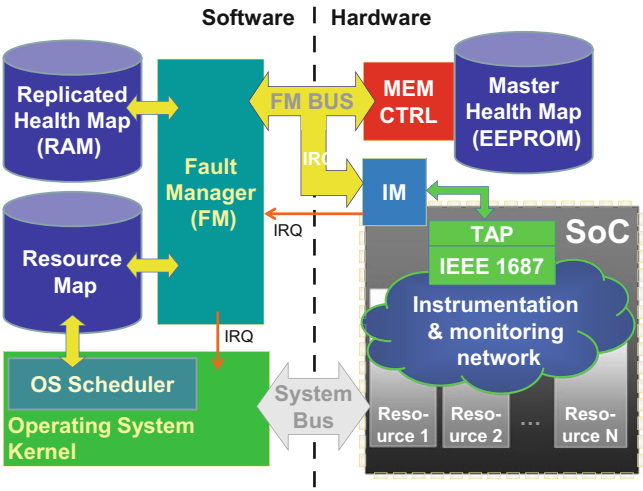

Fig. 5 Overview of the fault management infrastructure

the system's hardware resources and its important parts identified by the static (design time) analysis. To retain the information about the known faults across power cycles, HM should be stored in a reliable non-volatile memory.

Resource map (RM) is a data structure in the system memory which holds the information about the current status of the system's resources. It should be modified on the fly during system's normal operation, should a fault be detected by an instrument or a diagnostic routine.

5.2 Fault Classification and Handling

Ability to classify errors, malfunctions, and faults is an important basis for health map management, effective system recovery, and fault management. We propose to classify the faults according to their severity levels and their contribution to the permanent malfunction of system's components and modules. Such classification has a strong relation to fault management processes and the architecture of the Health Map.

The classification of faults to be used in FM procedures consists of the following categories:

- **Persistence**: This parameter shows what the nature of the fault occurrence is, i.e., whether it is transient, intermittent, or permanent.
- **Severity**: Faults can be different in their influence on the resource. While one fault can be benign (e.g., one of several similar execution units in a superscalar CPU fails), another can make the resource useless (e.g., program counter in a CPU core).
- **Criticality**: Depending on the resource where the fault has occurred, its consequences for operability and stability of the system as a whole can span from none to total system failure.
- **Diagnostic Granularity**: A fault is found by an instrument or deducted by diagnostic procedure. A fault entry in the data structure of fault management system should be assigned with the information about how it was found, e.g., by an instrument, diagnostic procedure, or an OS self-test.
- **Fault location**: The attributed location of the fault is the result of a fault detection or a fault diagnosis procedure.

When a fault occurs in the system and is detected with the help of FMI, the system must react and handle that fault in order to mitigate the current or future effects it can have on the system. The information which the proposed fault classification method offers is used in this process. The complete procedure which allows for quick and efficient fault handling should consist of the following steps: fault detection, fault localization, coarse-grained fault classification (before detailed diagnostic information becomes available), immediate system response (e.g., rescheduling a task affected by the fault), fault diagnosis, and, finally, a conclusive fine-grained fault classification.

6 Many-Core Resource Management for Fault Tolerance

On the architectural layer advanced CPS will rely on heterogeneous many-core SoCs to provide the demanded throughput computing performance within the allowed energy budget. Heterogeneous many-core architectures typically have many redundant and distributed resources for processing, communication, memory, and IO. This inherent redundancy can potentially be used to implement systems that are fault tolerant and degrade gradually. To realize this potential, we combine the FMI with run-time resource management software. First, the many-core architecture is instrumented with FMI and online checkers and health monitors. As we have explained in the previous sections, the online checkers and health monitors report faults and physical degradation at the lower hardware layers through the FMI that makes the information available for system and application software. The proposed instrumentation can thus provide a system wide HM showing the health and the functioning of the hardware resources of the running system. It reports on faulty components and also on health issues warning about fault expectancy. The former allows reacting on and recovering from faults whereas the latter allows anticipating and reconfiguring before faults occur. Second, the health information is lifted and abstracted to augment run-time resource management software [23, 47, 48] with information about hardware resources to be used less or entirely avoided by reconfiguring the way tasks and communications are mapped to resources.

The resource manager partitions computation, communication, and memory resources based on resource reservations of the application [1, 46]. The run-time mapping algorithms of the resource management software relies on abstract representations of task and platform graphs and are optimized for embedded systems [48].

In [2, 49] and [45] it was shown how reconfigurable multi/many-core architectures in combination with run-time resource management software can be used to implement fault-tolerance features. This work depended on ad hoc detection and reporting of faults and did not include health information about physical wear-out or accelerated aging. Here an important next step is taken to combine resource management with detailed health information systematically reported by a cross-layered fault management infrastructure at run-time.

The run-time resource management [23, 47] is made health-aware. Figure 6 illustrates the resource management with integrated HM information. Through the fault manager described in the previous section, measurements of health monitors and checkers provide domain-specific and/or hardware-specific information. For separation of concerns and extensibility, it is desired to hide this domain-specific knowledge from the upper software layers. At the lower layers, the domain-specific knowledge is required to map the sensor/checker data (the domain) onto a fixed range of values. So, the health data stored in the HM is modeled as a health function $\mathsf{health} : R \rightarrow [0, 1]$ that maps each hardware resource (provider) $r \in R$ to a health value $v \in [0, 1]$, where R is the finite set of resources in the target

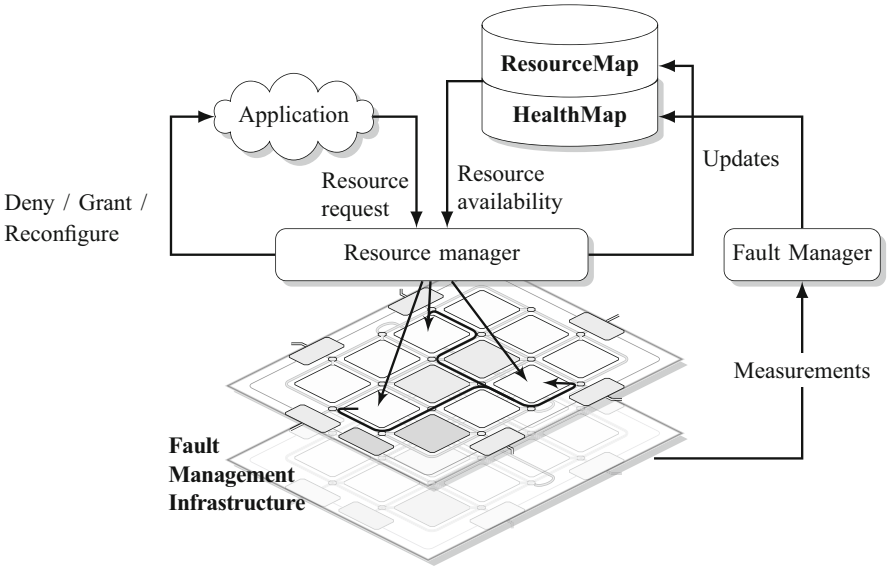

Fig. 6 Health information in resource management

architecture. A high health value $\mathsf{health}(r)$ indicates that a resource $r \in R$ is functioning correctly, whereas a low health value indicates the deterioration of the resource.

The advantages of using a health function with a range in the real numbers, as opposed to a function with a Boolean range, is that degradation can be modeled. The resource manager may circumvent the use of specific resources to reduce aging and hot spots. These resources are assumed to function correct when the health function is still positive, and can, therefore, still be activated when the system utilization increases.

The health function can be further extended to cover more details about the resource providers which could help the resource manager to choose best fitting resources for each task. As Fig. 6 illustrates, the fault manager reads the sensor/checker data out of the FMI, and processes the measurements by mapping the outcome to the range of the health function. Multiple sensors measuring the same hardware component should apply sensor fusion to conform to this HM function. In this fusion, again domain-specific knowledge is leveraged to weight the importance and possible relation between the sensors.

The health function is subsequently used in the selection process of the resource management, in which two use cases are identified:

1. New resource requests are handled according to the information contained in the HM.
2. For a resource in use, if the health indicator exceeds a configurable threshold, the resource manager will isolate the resource and attempt to reconfigure the applications currently using the corresponding resource.

For use case (1), a new request for resources is made by an application and the resource manager consults the RM to find the most suitable resources to fulfill the request. Both the assignment of tasks to processing elements and inter-task communication through the interconnect are taken into account. In this process, the resource manager uses a cost function to determine the best fit of the (partial) application onto the available resources of the platform. The configurable cost function takes the health map into account to define optimization objectives such as wear leveling. The cost function is designed to assign increasingly higher cost to a hardware resource $r \in R$, which should be used less or should not be used according to the HM, such that

$$\left(\lim_{\mathsf{health}(r) \to 0} \mathsf{cost}(r) \right) = \infty$$

For use case (2) whenever the HM is updated with new measurements, the new values are compared with a configurable threshold. When the threshold is exceeded, action needs to be taken to reduce the usage of that resource or completely stop using it. For resources currently in use possibly by several applications, this can require one or multiple granted resource requests to be reassigned to a different resource.

In a system including the proposed FMI and fault management approach, the FDIR procedure is facilitated by the results of the fault classification based on different fault categories determined by monitoring in lower layers, information from the instruments as well as the accumulated fault statistics.

7 Deriving Adaptive Test Strategies from LTL-Specifications

To obtain confidence in the correctness of a CPS system at the behavioral layer, model checking [13, 39] can prove that (a model of) the system satisfies desired properties. However, it cannot always be applied effectively.

This may be due to third-party IP components for which no source code or model is available, or due to high effort for building system models that are precise enough. Since our *System Under Test* (SUT) is safety critical, we desire high confidence in its adherence to specification φ. Nevertheless, even though φ may be simple, the implementation of the SUT can be too complex for model checking. Especially, if it considers further signals to synchronize with other systems. And finally, model checking can only verify an abstracted model and never the final and “live” system.

Testing is a natural approach to complement verification, and automatic test case generation allows to keep the effort at reasonable size. Deriving tests from a system specification instead of the implementation, called black-box testing, is particularly attractive as (1) tests can be generated way before the actual implementation work starts, (2) these tests can be reused on various realizations of the same specification, and (3) the specification is usually way simpler than the actual implementation. In addition, the specification focuses on the most important aspects that require intensive testing. Fault-based techniques [25], in which test cases are generated to detect certain fault classes, are particularly interesting to detect bugs.

Various methods focusing on coverage criteria exist to generate test sets from executable system models (e.g., finite state machines). Methods to derive tests from declarative requirements (see, e.g., [19]) are less common, as the properties still allow implementation freedom and, therefore, cannot be used to fully predict the system behavior under given inputs. Thus, test cases have to be *adaptive*, i.e., able to react to observed behavior at run-time. This is especially true for *reactive systems* that interact with their environment. Existing techniques often get around this by requiring a deterministic model of the system behavior as additional input [18].

In [10] we presented a new approach to synthesize test strategies from temporal logic specification. This approach is also applicable on a CPS if a temporal logic specification is given. The derived adaptive strategies can be used during the development process for system verification as well as after deployment for run-time verification to detect faults that occur only after a certain amount of time, for example due to aging. Figure 7 outlines our proposed testing setup. The user provides a specification φ, expressing requirements for the system under test in *Linear Temporal Logic* (LTL) [37]. The specification can be incomplete. The user also provides a fault model, for which the generated tests shall cause a specification violation, in form of an LTL formula that has to be covered.

Based on hypotheses from fault-based testing [36], we argue that tests that reveal faults as specified by our fault models are also sensitive to more complex bugs. We assume permanent and transient faults by distinguishing various fault occurrence frequencies and computing tests to reveal faults for the lowest frequency for which this is possible. Test strategies are generated using reactive synthesis [38] with partial information [29], providing strong guarantees about all uncertainties: If the synthesis is successful and if the computed tests are executed long enough, then they reveal all faults satisfying the fault model in every system that realizes the specification. Finally, existing techniques from run-time verification [9] can be used

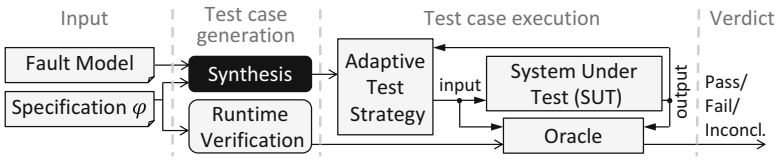

Fig. 7 Synthesis of adaptive test strategies from temporal logic specifications [10]

to construct an oracle that checks the system behavior against the specification while tests are executed.[2]

If the specification is incomplete, tests may have to react to observed behavior at run-time to achieve the desired goals. Such adaptive test cases have been studied by Hierons [21] from a theoretical perspective, however, relying on fairness (every non-deterministic behavior is exhibited when trying often enough) or probabilities.

Testing reactive systems can be seen as a game between two players: the tester providing inputs and trying to reveal faults, and the SUT providing outputs and trying to hide faults, as pointed out by Yannakakis [52]. The tester can only observe outputs and has, therefore, partial information about the SUT. The goal for the game is to find a strategy for the tester that wins against every SUT. The underlying complexities are studied by Alur et al. [5]. Our work builds upon reactive synthesis [38] (with partial information [29]). This can also be seen as a game, however, we go beyond the basic idea. We combine the concept of game theory with fault models defined by the user. Nachmanson et al. [34] synthesize game strategies as tests for non-deterministic software models. Their approach, however, is not fault-based and focuses only on simple reachability goals.

To mitigate scalability issues, we compute test cases directly from the provided specification φ. Our goal is to generate test strategies that *enforce* certain coverage objectives *independent* of any freedom due to incomplete specification. Some uncertainties about the behavior of the SUT may also be rooted in uncontrollable environment aspects like weather conditions. For our proposed testing approach, this makes no difference.

We follow a fault-centered approach. The definition of the fault class is a composition of the fault kind and the fault frequency. While the fault kind expresses the type of the fault, such as a bit flip or a stuck-at fault, the fault frequency describes the frequency of the fault being present in the system. This can be (1) a permanent fault that is present all the time, (2) a fault that occurs from some point onwards, (3) a fault that occurs again and again, or even (4) a fault that occurs only once in the future. A test strategy that is capable of detecting a fault that occurs only at a low frequency, for example only once in the future, is also capable of detecting a fault that occurs at a higher frequency, for example from some point in time onwards. Thus, the goal is to derive a strategy for the lowest fault-frequency possible.

Certain test goals may not be enforceable with a static input sequence. We thus synthesize *adaptive* test strategies that direct the tester based on previous inputs and outputs and, therefore, can take advantage of situational possibilities by exploiting previous system behavior. The derived strategies force the system to enter a state in which it has to violate the specification if the fault is present in the system.

[2]The semantics of LTL are defined over infinite execution traces; however, we can only run the tests for a finite amount of time. This can result in inconclusive verdicts [9]. To overcome this problem, we refer to existing research on interpreting LTL over finite traces [14, 15, 20].

Our generated test strategies reveal all instances of a user-defined fault class for every realization of a given specification and do not rely on any implementation details.

8 Parameter Synthesis for CPS

Many problems in the context of computer-aided design and verification of CPS can be reduced to deciding the satisfiability of logic formulæ modulo background theories [8]. In parameter synthesis, the logic formulæ describe how the CPS evolves over time from a set of initial states, where some parameters are kept open and have to be filled such that none of a given set of bad states is ever reached. Parameter synthesis can be effectively reduced to solving instances of $\exists\forall$-queries. An $\exists\forall$-query asks for the existence of parameter values such that for all possible state sequences, the CPS avoids reaching a bad state.

Solving such $\exists\forall$-queries is especially challenging in the context of CPS, where the variables are quantified over countably infinite or uncountably infinite domains. Different approaches for parameter synthesis for hybrid automata, e.g., [11, 12, 17], have been proposed. The approaches considered the problems of computing one value for the parameters as well as all possible parameter values, but are restricted to hybrid automata with linear and multiaffine dynamics. We propose a *Satisfiability Modulo Theories* (SMT)-based framework for synthesizing one value for open parameters of a CPS modeled as logic formulæ [40] using *Counterexample-Guided Inductive Synthesis* (CEGIS) [44] and introduce the notion of *n-step inductive invariants* to reason about unbounded CPS correctness.

CEGIS CEGIS is an attractive technique from software synthesis to infer parameters in a sketch of a program leveraging the information of a provided correctness specification. In software synthesis, CEGIS was able to infer those parameters in many cases, where existing techniques from quantifier elimination failed.

Suppose that Q, I, and K are the sets of all possible states, inputs, and parameter valuations, respectively. We use the correctness formula $\mathsf{correct} : I' \times K \to \mathbb{B}, (i', k) \mapsto \mathsf{correct}(\hat{i}', \hat{k})$ that evaluates to true if and only if the CPS with concrete parameter values $\hat{k} \in K$ is correct when executed on the concrete input sequence $\hat{i}' \in I'$, where $I' = Q \times I^n$. The basic idea of CEGIS is to iteratively refine candidate values for parameters based on counterexamples until a correct solution is obtained. The CEGIS loop is depicted in Fig. 8. The loop repeats two steps to compute parameter values $\hat{k} \in K$ such that $\forall i' \in I' : \mathsf{correct}(i', \hat{k})$ holds and maintains a database $D \subseteq I'$ of concrete parameter values, which is initially empty. The database is used to lazily approximate the domain of I' with a small set of values. In the first step, a candidate parameter $\hat{k}$ is computed such that $\bigwedge_{\hat{i}' \in D} \mathsf{correct}(\hat{i}', \hat{k})$ holds, i.e., the parameter values $\hat{k}$ guarantee correctness of the CPS for (at least) all input sequences stored in the database D. The candidate parameters are then verified by checking if a counterexample $\hat{i}'$ exists that refutes $\forall i' \in I' : \mathsf{correct}(i', \hat{k})$

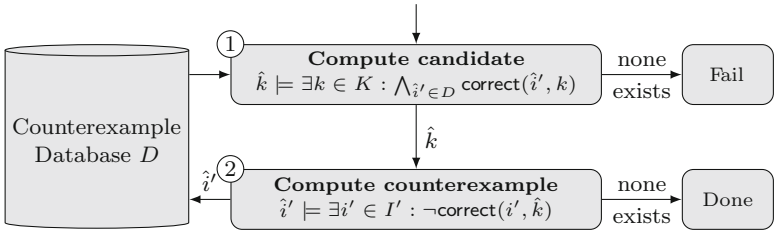

Fig. 8 Counterexample-guided inductive synthesis (CEGIS) [44]

considering the entire domain I' of input sequences. If so, the counterexample $\hat{i}'$ is added to the database D. Otherwise, if no counterexample exists, the approach terminates and returns the parameters $\hat{k}$. In the general case, the CEGIS loop has three possible outcome: (1) parameters $\hat{k} \in K$ can be found such that the formula $\forall i' \in I' : \mathsf{correct}(i', \hat{k})$ becomes true (Done), (2) the unsatisfiability of the formula $\exists k \in K : \forall i' \in I' : \mathsf{correct}(i', k)$ is proven because no new parameters can be computed (Fail), or (3) the CEGIS loop does not terminate but refines the candidate values for the parameters forever. To guarantee termination of the loop, at least one of the two involved domains, K or I', has to be finite. However, even if both domains are infinite, the approach is in many cases able to synthesize parameters.

***n*-Step Inductive Invariants** The correctness of a CPS is defined by using an invariant-based approach. A user symbolically defines the set of all possible initial states $\mathsf{init} : Q \times K \to \mathbb{B}$, the set of all safe states $\mathsf{safe} : Q \times K \to \mathbb{B}$, the sets of all states of an inductive invariant $\mathsf{inv} : Q \times K \to \mathbb{B}$, and a transition function $T : Q \times I \times K \to Q$ of the CPS in the form of logic formulæ modulo theories. By induction, a CPS cannot visit an unsafe state and is correct if:

1. all initial states satisfy the invariant, i.e.,

$$\mathsf{A}(q, k) :\Leftrightarrow \Big(\mathsf{init}(q, k) \to \mathsf{inv}(q, k)\Big),$$

2. all states that satisfy the invariant are also safe, i.e.,

$$\mathsf{B}(q, k) :\Leftrightarrow \Big(\mathsf{inv}(q, k) \to \mathsf{safe}(q, k)\Big),$$

3. from a state that satisfies the invariant, the invariant is again satisfied after at most n steps of the transition relation T and all states that can be reached in the meantime are safe, i.e.,

$$\mathsf{C}(q_0, i_1, \ldots, i_n, k) :\Leftrightarrow \Big(\mathsf{inv}(q_0, k) \to \bigvee_{j=1}^{n} \mathsf{inv}(q_j, k) \wedge \bigwedge_{l=1}^{j-1} \mathsf{safe}(q_l, k)\Big),$$

where q_j is an abbreviation for $T(q_{j-1}, i_j, k)$ for all $j > 0$.

The CPS is correct if concrete parameter values $\hat{k} \in K$ exist such that

$$\forall q_0 \in Q : \forall i_1, \ldots, i_n \in I : \mathsf{correct}(q_0, i_1, \ldots, i_n, \hat{k})$$

holds, where the correctness formula

$$\mathsf{correct}(q_0, i_1, \ldots, i_n, \hat{k}) :\Leftrightarrow \Big(A(q_0, \hat{k}) \wedge B(q_0, \hat{k}) \wedge C(q_0, i_1, \ldots, i_n, \hat{k})\Big)$$

is defined over sequences of inputs $(q_0, i_1, \ldots, i_n) \in I'$ of n steps.

8.1 Heuristics and Implementation

We implemented the CEGIS loop depicted in Fig. 8 as a proof-of-concept tool, ParSyn-CEGIS,[3] based on an SMT-solver. The SMT-solver is used to find concrete parameters and counterexamples. In case of CPS typically infinite domains are considered such that the CEGIS loop may not converge. To improve convergence in practice, we developed three simple heuristics:

1. *Counterexample randomization*: To avoid the generation of too similar counterexamples, the ParSyn-CEGIS attempts to randomize every second counterexample. In an iterative loop, for each value of the counterexample, a random value of the same type is generated and substituted. If the adapted counterexample still violates the correctness check, i.e., is still a counterexample, the randomized value is kept. Otherwise, it is rejected.
2. *Restart strategy*: Inspired by the implementation of today's solvers for Boolean satisfiability, we implemented a restart strategy. The restart strategy aids the SMT-solver to recover from learned information that does not help in deciding the overall $\exists\forall$-query. When a restart happens, all counterexamples are removed from the database and the CEGIS loop starts from the beginning without a priori knowledge. After each restart, the period of the restart is increased.
3. *Demand for progress*: Given two subsequent values $\hat{k}_a$ and $\hat{k}_b$ of the same parameter, we measure their progress by $\mathsf{progress}(\hat{k}_a, \hat{k}_b) = \|\hat{k}_a - \hat{k}_b\|$. This measure is used to restart the synthesis procedure when the CEGIS loop gets stuck by producing similar counterexamples, but counterexample randomization is not effective. In each iteration, for the last pair of parameter values $\hat{k}_{c-1}$ and $\hat{k}_c$, the progress value $\mathsf{progress}(\hat{k}_{c-1}, \hat{k}_c)$ is computed. If the progress value repeatedly falls below a fixed progress threshold δ, e.g., more than 10 times, a restart is initiated.

[3] ParSyn-CEGIS, https://github.com/hriener/parsyn-cegis.

Parameter synthesis automates the task of finding good values for important design parameters in CPS and eliminates the error prone design steps involved in determining those parameter values manually.

9 Conclusions

Within the IMMORTAL project, we have identified several challenging problems in the context of reliability and automated debug considering advanced CPS throughout the stack of layers and the design flow. For each of these problems we presented in this chapter a glimpse on how to solve the issues and how tool automation can improve the overall design process. For further details on each of the solutions we refer to the respective publications.

Overall, reliable CPS design and the corresponding design automation is a vivid and ongoing research topic. CPS design automation links traditional hardware-oriented aspects with software engineering and the large body of work in control theory.

Acknowledgements This work was supported in part by the European Union (Horizon 2020 IMMORTAL project, grant no. 644905).

References

1. Abeni, L., & Buttazzo, G. (2004). Resource reservation in dynamic real-time systems. *Real-Time System, 27*(2), 123–167.
2. Ahonen, T., ter Braak, T. D., Burgess, S. T., Geißler, R., Heysters, P. M., Hurskainen, H., et al. (2011). CRISP: Cutting edge reconfigurable ICs for stream processing. In *Reconfigurable computing: From FPGAs to hardware/software codesign* (pp. 211–237). New York: Springer.
3. Aleksandrowicz, G., Arbel, E., Bloem, R., ter Braak, T. D., Devadze, S., Fey, G., et al. (2016). Designing reliable cyber-physical systems – Overview associated to the special session at FDL'16. In *2016 Forum on Specification and Design Languages, FDL 2016, Bremen, Germany, September 14–16, 2016* (pp. 1–8).
4. Ali, G., Badawy, A., & Kerkhoff, H. G. (2016). On-line management of temperature health monitors using the IEEE 1687 standard. In *TESTA* (pp. 1–4).
5. Alur, R., Courcoubetis, C., & Yannakakis, M. (1995). Distinguishing tests for nondeterministic and probabilistic machines. In *STOC* (pp. 363–372).
6. Arbel, E., Barak, E., Hoppe, B., Koyfman, S., Krautz, U., & Moran, S. (2016). Gating aware error injection. In *HVC* (pp. 34–48).
7. Arbel, E., Koyfman, S., Kudva, P., & Moran, S. (2014). Automated detection and verification of parity-protected memory elements. In *ICCAD* (pp. 1–8).
8. Barrett, C. W., Sebastiani, R., Seshia, S. A., & Tinelli, C. (2009). Satisfiability modulo theories. In *Handbook of satisfiability* (pp. 825–885). Amsterdam: IOS Press.
9. Bauer, A., Leucker, M., & Schallhart, C. (2011). Runtime verification for LTL and TLTL. *TOSEM, 20*(4), 14.
10. Bloem, R., Könighofer, R., Pill, I., & Röck, F. (2016). Synthesizing adaptive test strategies from temporal logic specifications. In *FMCAD* (pp. 17–24).

11. Bogomolov, S., Schilling, C., Bartocci, E., Batt, G., Kong, H., & Grosu, R. (2015). Abstraction-based parameter synthesis for multiaffine systems. In *HVC* (pp. 19–35).
12. Cimatti, A., Griggio, A., Mover, S., & Tonetta, S. (2013). Parameter synthesis with IC3. In *FMCAD* (pp. 165–168).
13. Clarke, E. M., & Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic. In *LOP* (pp. 52–71).
14. De Giacomo, G., De Masellis, R., & Montali, M. (2014). Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI* (pp. 1027–1033).
15. De Giacomo, G., & Vardi, M. Y. (2013). Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.
16. Frehse, S., Fey, G., Arbel, E., Yorav, K., & Drechsler, R. (2012). Complete and effective robustness checking by means of interpolation. In *FMCAD* (pp. 82–90).
17. Frehse, G., Jha, S. K., & Krogh, B. H. (2008). A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC* (pp. 187–200).
18. Fraser, G., Wotawa, F., & Ammann, P. (2009). Issues in using model checkers for test case generation. *Journal of Systems and Software, 82*(9), 1403–1418.
19. Fraser, G., Wotawa, F., & Ammann, P. (2009). Testing with model checkers: A survey. *Software Testing, Verification and Reliability, 19*(3), 215–261.
20. Havelund, K., & Rosu, G. (2001). Monitoring programs using rewriting. In *ASE* (pp. 135–143).
21. Hierons, R. M. (2006). Applying adaptive test cases to nondeterministic implementations. *Information Processing Letters, 98*(2), 56–60.
22. Holcomb, D. E., Li, W., & Seshia, S. A. (2009). Design as you see FIT: System-level soft error analysis of sequential circuits. In *DATE* (pp. 785–790).
23. Hölzenspies, P. K. F., ter Braak, T. D., Kuper, J., Smit, G. J. M., & Hurink, J. (2010). Run-time spatial mapping of streaming applications to heterogeneous multi-processor systems. *International Journal of Parallel Programming, 38*(1), 68–83.
24. Ibrahim, A. M. Y., & Kerkhoff, H. G. (2016). An IJTAG-compatible I_{DDT} embedded instrument for health monitoring and prognostics. In *ITC, Poster Session*.
25. Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering, 37*(5), 649–678.
26. Khan, M. A. (2014). *On Improving Dependability of Analog and Mixed-Signal SoCs: A System-Level Approach* Ph.D. thesis, University of Twente.
27. Krautz, U., Pflanz, M., Jacobi, C., Tast, H.-W., Weber, K., & Vierhaus, H. T. (2006). Evaluating coverage of error detection logic for soft errors using formal methods. In *DATE* (pp. 176–181).
28. Krishnaswamy, S., Plaza, S., Markov, I. L., & Hayes, J. P. (2009). Signature-based SER analysis and design of logic circuits. *IEEE Transactions on Computer-Aided Design, 28*(1), 74–86.
29. Kupferman, O., & Vardi, M. Y. (1997). Synthesis with incomplete information. In *ICTL* (pp. 91–106).
30. Lee, E. A., & Seshia, S. A. (2015) *Introduction to embedded systems: A Cyber-physical systems approach* (2nd ed.). LeeSeshia.org
31. Leveugle, R., Calvez, A., Maistri, P., & Vanhauwaert, P. (2009). Statistical fault injection: Quantified error and confidence. In *DATE* (pp. 502–506).
32. Maniatakos, M., & Makris, Y. (2010). Workload-driven selective hardening of control state elements in modern microprocessors. In *VTS* (pp. 159–164).
33. Mukherjee, S. S., Weaver, C. T., Emer, J. S., Reinhardt, S. K., & Austin, T. M. (2003). A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO* (pp. 29–42).
34. Nachmanson, L., Veanes, M., Schulte, W., Tillmann, N., & Grieskamp, W. (2004). Optimal strategies for testing nondeterministic systems. In *ISSTA* (pp. 55–64).
35. Nicolaidis, M. (2010). Design techniques for soft-error mitigation. In *ICICDT* (pp. 208–214).
36. Offutt, A. J. (1992). Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology, 1*(1), 5–20.
37. Pnueli, A. (1977). The temporal logic of programs. In *FOCS* (pp. 46–57).

38. Pnueli, A., & Rosner, R. (1989). On the synthesis of a reactive module. In *POPL* (pp. 179–190).
39. Queille, J.-P., & Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In *PROGRAM* (pp. 337–351).
40. Riener, H., Könighofer, R., Fey, G., & Bloem, R. (2016). SMT-based CPS parameter synthesis. In *ARCH@CPSWeek* (pp. 126–133).
41. Saltarelli, P., Niazmand, B., Hariharan, R., Raik, J., Jervan, G., & Hollstein, T. (2015). Automated minimization of concurrent online checkers for network-on-chips. In *ReCoSoC* (pp. 1–8).
42. Saltarelli, P., Niazmand, B., Raik, J., Govind, V., Hollstein, T., Jervan, G., et al. (2015). A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in NoC routers. In *NOCS* (pp. 6:1–6:8).
43. Seshia, S. A., Li, W., & Mitra, S. (2007). Verification-guided soft error resilience. In *DATE* (pp. 1442–1447).
44. Solar-Lezama, A. (2013). Program sketching. *International Journal on Software Tools for Technology Transfer, 15*(5–6), 475–495.
45. Sourdis, I., Strydis, C., Armato, A., Bouganis, C.-S., Falsafi, B., Gaydadjiev, G. N., et al. (2013). DeSyRe: On-demand system reliability. *Microprocessors and Microsystems, 37*(8-C), 981–1001.
46. Steffens, L., Fohler, G., Lipari, G., & Buttazzo, G. (2003). Resource reservation in real-time operating systems – a joint industrial and academic position. In *ARTOSS* (pp. 25–30).
47. ter Braak, T. D. (2014). Using guided local search for adaptive resource re-servation in large-scale embedded systems. In *DATE* (pp. 1–4).
48. ter Braak, T. D. (2016). *Run-Time Mapping: Dynamic Resource Allocation in Embedded Systems*. Ph.D. thesis, University of Twente.
49. ter Braak, T. D., Toersche, H. A., Kokkeler, A. B. J., & Smit, G. J. M. (2011). Adaptive resource allocation for streaming applications. In *SAMOS* (pp. 388–395).
50. Wan, J., & Kerkhoff, H. G. (2015). Embedded instruments for enhancing dependability of analogue and mixed-signal IPs. In *NEWCAS* (pp. 1–4).
51. Wan, J., Kerkhoff, H. G., & Bisschop, J. (2016). Simulating NBTI degradation in arbitrary stressed analog/mixed-signal environments. *IEEE Transactions on Nanotechnology, 15*(2), 137–148.
52. Yannakakis, M. (2004). Testing, optimizaton, and games. In *LICS* (pp. 78–88).
53. Zambrano, A. C., & Kerkhoff, H. G. (2015). Fault-tolerant system for catastrophic faults in AMR sensors. In *IOLTS* (pp. 65–70).
54. Zhao, Y., & Kerkhoff, H. G. (2016). A genetic algorithm based remaining lifetime prediction for a VLIW processor employing path delay and IDDX testing. In *DTIS* (pp. 1–5).

On the Application of Formal Fault Localization to Automated RTL-to-TLM Fault Correspondence Analysis for Fast and Accurate VP-Based Error Effect Simulation: A Case Study

Vladimir Herdt, Hoang M. Le, Daniel Große, and Rolf Drechsler

Abstract Electronic systems integrate an increasingly large number of components on a single chip. This leads to increased risk of faults, e.g., due to radiation, aging, etc. Such a fault can lead to an observable error and failure of the system. Therefore, an error effect simulation is important to ensure the robustness and safety of these systems. Error effect simulation with Virtual Prototypes (VPs) is much faster than with RTL designs due to less modeling details at TLM. However, for the same reason, the simulation results with VP might be significantly less accurate compared to RTL. To improve the quality of a TLM error effect simulation, a fault correspondence analysis between both abstraction levels is required. This chapter presents a case study on applying fault localization methods based on symbolic simulation to identify corresponding TLM errors for transient bit flips at RTL. First results for the interrupt controller of the SoCRocket VP, which is being used by the European Space Agency, demonstrate the applicability of our approach.

Keywords Error effect simulation • Virtual prototype (VP) • SystemC • TLM • RTL • Fault localization • Formal analysis • SMT • Fault correspondence analysis • Symbolic simulation • SoCRocket • Intermediate verification language • IVL • IRQMP • Fault injection • Transient bit flip • ESL • High-level error model • Partial order reduction • Model checking

V. Herdt (✉) • H.M. Le
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
e-mail: vherdt@informatik.uni-bremen.de; hle@informatik.uni-bremen.de

D. Große • R. Drechsler
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany

Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
e-mail: grosse@informatik.uni-bremen.de; drechsle@informatik.uni-bremen.de

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_3

1 Introduction

Ensuring the functional safety of electronic systems becomes one of the most important issues nowadays, as these systems are being more and more deeply integrated into our lives. Even if a system can be proved to perform its intended functionality correctly, failures are still possible due to hardware (HW) faults caused, e.g., by radiation or aging. The risk of such faults is rapidly increasing with the raising complexity of design and technology scaling. To evaluate and facilitate the development of safety measures, fault injection is a widely accepted approach, which is also recommended in different functional safety norms such as IEC 61508 and ISO 26262.

Traditional HW fault injection approaches operate on low levels of abstraction such as gate level or *Register Transfer Level* (RTL). While physical faults can be quite accurately modeled at these abstraction levels, the slow simulation speed becomes a major bottleneck for modern systems. This used to be a problem for software (SW) development and system verification as well, until the emergence of *SystemC Virtual Prototypes* (VPs). VPs are basically full functional SW models of HW abstracting away micro-architectural details. This higher level of abstraction, often termed as *Transaction-Level Modeling* (TLM) [12], allows significantly faster simulation compared to RTL. Therefore, safety evaluation using VP-based fault injection, envisioned as error effect simulation in [20], is a very promising direction.

Fault injection techniques for SystemC have attracted a large number of work, see e.g. [16, 19, 21, 24]. They form a technical basis for error effect simulation, but do not focus on the core problem: the higher level of abstraction of TLM poses a big challenge in error modeling. Please note that we use the term "fault" for RTL and "error" for TLM due to the fact that TLM is just a modeling abstraction. A high-level error model, if not carefully designed, would yield significantly different simulation results compared to a low-level fault model [4, 17]. This would make the safety evaluation results using VPs to become misleading. Unfortunately, deriving an accurate high-level error model is very difficult [4, 17].

In this chapter, we examine the idea of a novel cross-level fault correspondence analysis to aid the design of such error model. The prerequisite of our analysis is the availability of an RTL model and its corresponding TLM model. Then, for a given RTL fault model, our analysis automatically identifies for an RTL fault a set of candidates for error injection in the TLM model. These candidates are potentially *equivalent* to the RTL fault, in the sense that the error-injected TLM model would produce the same failure as the fault-injected RTL model.

The core idea of this RTL-to-TLM fault correspondence analysis is inspired by the concept of *formal fault localization* (see [9] for C and [13] for SystemC TLM). These approaches automatically identify candidates for modifications in the model under verification, which would make a given set of failing testcases to become passing testcases. Such a candidate potentially points to the location of the bug that

causes a failure. Our analysis is dual to this: Given successful TLM simulations,[1] we search for locations to inject an error to produce the same failure observed in faulty RTL simulations. Hence, we can apply the same set of techniques: instrumenting the TLM model to include nondeterministic errors and leveraging existing TLM model checkers to compute the candidates.

The chapter also presents first results of a case study on the *Interrupt Controller for Multiple Processors* (IRQMP) of the SoCRocket VP, which is being used by the European Space Agency [22], to demonstrate the feasibility of the analysis. In particular, we implement the cross-level analysis on top of the recent symbolic simulation approach for SystemC [14, 15] and apply our analysis to find TLM error injection candidates for transient bit flips at RTL. To the best of our knowledge, it is the first time such results on fault correspondence between TLM and RTL are reported.

2 Related Work

As mentioned earlier, a large number of fault injection techniques for SystemC TLM models exists. These approaches assume some TLM error models without qualitative or quantitative assessment of their correspondence to RTL faults. Beltrame et al. [1] includes a comprehensive list of TLM errors that might result from RTL bit flips. The paper also provides a set of guidelines on how to (manually) derive corresponding TLM errors, whereas our analysis is automated.

Another line of work is mutation analysis for SystemC TLM models [3, 18, 23]. At the heart of any mutation analysis is also an error model. However, the purpose of such model is to mimic common design errors, but not HW faults caused by external impact.

The most close work to ours is the one in [2], which proposes an automatic RTL-to-TLM transformation to speed-up RTL fault simulation. The transformation produces an equivalent TLM model from each fault-injected RTL model. The obtained results can be mapped back to RTL. However, this approach relies on a particular RTL-to-TLM transformation. Such transformation might not provide the best possible speed-up compared to hand-crafted TLM models. In contrast to our analysis, this approach is not applicable when the corresponding TLM model already exists.

[1] A successful TLM simulation produces the same output as RTL simulation under the same inputs.

3 Preliminaries

In this section we first briefly describe the interrupt controller IRQMP of the SoCRocket VP used later in the case study. The second half of this section briefly describes the (extended) intermediate verification language to enable the formal representation of TLM models as well as leveraging advanced symbolic execution techniques.

3.1 Interrupt Controller for Multiple Processors (IRQMP)

The IRQMP is an interrupt controller from the SoCRocket VP supporting up to 16 processors [22]. It consists of a register file, several input and output wires, and an APB Slave bus interface for register access. The register file contains shared and processor-specific registers. Every register has a bit width of 32. Each bit naturally represents an interrupt line.

The IRQMP supports incoming interrupts (using the *irq_in* wire or *force* register) numbered from 1 to 31 (interrupt line 0 is reserved/unused). Lines 15:1 are regular interrupts and lines 31:16 are extended interrupts. In regular operation mode, IRQMP ignores all incoming extended interrupts. The *irq_req* and *irq_ack* wires are connected with every processor and allow to send interrupt requests and receive acknowledgements.

The functionality of the IRQMP is to process incoming interrupts by applying masking and prioritization of interrupts for every processor. Prioritization of multiple available interrupts is resolved using the *level* register. A high (low) bit in the *level* register defines a high (low) priority for the corresponding interrupt line. On the same priority level, interrupt with larger line number is higher prioritized. See the specification [10] of the IRQMP for more details.

3.2 Extended Intermediate Verification Language (XIVL)

The XIVL has been proposed in [15] as an extension to the SystemC *Intermediate Verification Language* (IVL) [14] to act as high-level intermediate representation with formal verification support for SystemC TLM. In essence, the XIVL provides a small core of instructions to capture the cooperative multi-threaded simulation semantics of SystemC and supports all arithmetic and logic operators of C++. For verification purposes it provides symbolic expressions as well as the assume and assert functions with their usual semantics. Control flow is modeled using high level control flow structures. A small set of object-oriented programming features—including virtual functions, inheritance, and dynamic dispatch—is supported for modeling TLM designs more naturally.

Symbolic simulation for SystemC as proposed in [5, 6, 11, 14] essentially combines symbolic execution with complete exploration of all process schedules. Partial Order Reduction techniques are employed to improve scalability by pruning redundant schedules [7, 8].

A formal verification approach based on symbolic simulation has been presented in [15] and demonstrated for the IRQMP TLM model. In this work, we instrument the available TLM model to include nondeterministic errors and leverage the existing TLM model checker based on symbolic simulation for fault localization.

4 RTL-to-TLM Fault Correspondence Analysis

Our proposed RTL-to-TLM fault correspondence analysis allows to find errors in a TLM model that corresponds to faults in the RTL model. We assume that the RTL and TLM model are functionally equivalent, i.e., they produce the same outputs when given the same inputs. Furthermore, we assume that a set of (representative) inputs, e.g., in the form of testcases, is available for the RTL or TLM (since both use the same inputs) model. These inputs should preferably cover a large set of functionality of the design. Similarly, we assume that a set of fault injection locations is available for the RTL model. Otherwise, the fault injection locations can be obtained by tracing the execution of the RTL model, e.g., using an observer class, based on the available testcases. Please note that a fault injection location consists of three pieces of information: (1) a source line, (2) an injection time, i.e., a number that denotes which execution of this source line should be fault injected, and (3) the bit position which shall be flipped. The reason for information (2) is that we consider transient faults in this work.

4.1 Correspondence Analysis Overview and Algorithm

Figure 1 shows an overview of our analysis approach. A corresponding algorithm is shown in Fig. 2. It computes a set of corresponding error candidates on TLM level for every injected fault on RTL. The algorithm considers every fault injection location L on the RTL model. First we construct the faulty RTL model with respect to L (Line 4). In our implementation, we attach a fault injection class as observer to our RTL model that can inject an error at runtime. Then we simulate the correct and faulty RTL model with the same input. There are two possible cases: (1) Both RTL models produce the same output. In this case the injected fault had no observable effect and simulation is repeated with a different input (Line 5). (2) Otherwise, both RTL models produce a different output. Then we apply a formal fault localization analysis based on symbolic simulation on the TLM model (Line 9) to produce a set of possible corresponding error locations C on the TLM model. Essentially, all these reported error locations C produce the same failure, i.e., same output, as the

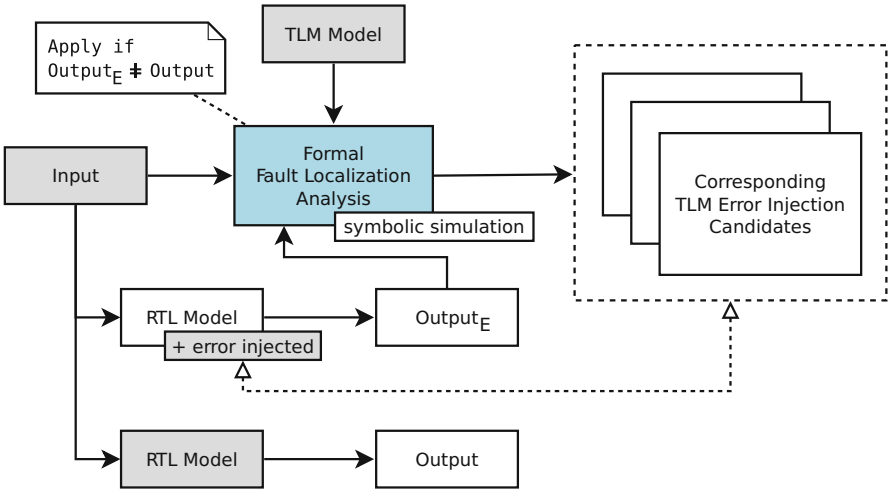

Fig. 1 Fault correspondence analysis overview

```
1  result ← ∅ /* Mapping from RTL fault injection location to set
       of corresponding TLM error injection candidates          */
2  for L ∈ RTL-fault-injection-locations do
       /* Start with empty result set for L                    */
3      result[L] ← ∅
4      RTL-Model_E ← RTL-Model with L injected
5      for input ∈ testcases do
           /* Check if RTL fault L has observable effect       */
6          output ← simulate(RTL-Model, input)
7          output_E ← simulate(RTL-Model_E, input)
8          if output ≠ output_E then
9              candidates ← fault-localization-analysis(TLM-Model, input, output_E)
10             if result[L] = ∅ then
                   /* First set of candidates                  */
11                 result[L] ← candidates
12             else
                   /* Combine with existing set                */
13                 result[L] ← result[L] ∩ candidates
14             if result[L] = ∅ then
                   /* No corresponding TLM error for L found, check
                      next RTL fault                           */
15                 break
```

Fig. 2 Fault correspondence analysis

faulty RTL model for the given input. The set C is integrated into the result set for L by computing a set intersection (Line 13)—or simple assignment in case C is the first result (Line 11). If the result set is empty, our analysis concludes that no corresponding TLM error can be found for the current RTL fault and we consider the next RTL fault. The reason is that a corresponding TLM error must produce the same output as the faulty RTL model for *all* inputs. Otherwise, the result set is not empty, we continue with the next input.

4.2 Example

Method Overview

As an example, consider a bit flip fault in the RTL model of the IRQMP (interrupt controller, see preliminaries Sect. 3.1) when initially configuring the mask register using a bus transfer operation as fault injection location L. Furthermore, consider three test scenarios with different inputs:

1. Send interrupt using the *irq_in* (incoming interrupt) wire
2. Send interrupt using the *force* register
3. Send the same interrupt twice using the *irq_in* wire

All of these inputs result in different outputs for the RTL model with and without injection of the fault L. In particular no interrupt is generated by the interrupt controller for the masked interrupt line. On the TLM model, our analysis identifies different candidates for corresponding errors. In particular, it identifies different transient bit flip errors during computations as well as wire/bus transfer that lead to the same observable behavior.

The results are summarized in Table 1. For the first and second input we obtain multiple possible error locations in the TLM model. By sending the same interrupt twice (third input) to the interrupt controller, the effects of a transient computation error and non-related transfer error are eliminated. By computing the intersection of all possible error locations for these inputs, the corresponding TLM error is obtained—bus transfer error during configuration of the mask register.

Fault Correspondence

To further illustrate this fault injection example, Figs. 3 and 4 show relevant code of the IRQMP for configuring the *mask* register at RTL and TLM, respectively. The source line where the fault has been injected at RTL and the corresponding error location at TLM are highlighted. The RTL code is available in VHDL, and the TLM code in SystemC.

The RTL code stores internal signals for registers separated for regular (lines 15:1, *reg* type) and extended interrupts (lines 31:16, *ereg* type). The processing

Table 1 Corresponding TLM error injection candidates (corresponding error highlighted in bold)

TLM error injection location	Inputs		
	1	2	3
Bus transfer			
Mask register configuration	**X**	**X**	**X**
Force register configuration		X	
Wire transfer			
Incoming interrupt wire	X		
Computation			
Prioritization logic 1	X	X	
Prioritization logic 2	X	X	

```
1  -- combinatorial VHDL process, which essentially contains the whole logic
       of the IRQMP - sensitive on the reset signal, update of internal
       signals as well as incoming bus and interrupt signals
2  comb : process(...)
3    -- send out current internal signal values and compute new values which
         will overwrite the current values in the next clock cycle
4    variable v  : reg_type;  -- registers to store regular interrupt lines
         for local computation
5    variable v2 : ereg_type; -- registers to store extended interrupt lines
         for local computation
6    begin
7      -- ... prioritize interrupts, register read ...
8
9      -- register write
10     if ((apbi.psel(pindex) and apbi.penable and apbi.pwrite) = '1' and
11         (irqmap = 0 or apbi.paddr(9) = '0')) then -- essentially, check
               that bus is enabled and used in write mode
12       case apbi.paddr(7 downto 6) is -- decode target register
13       -- ...
14       when "01" => -- write to processor specific mask register
15         for i in 0 to ncpu-1 loop -- iterate over all processors
16           if i = conv_integer( apbi.paddr(5 downto 2)) then -- decode and
                 check target processor
17             v.imask(i) := apbi.pwdata(15 downto 1); -- write to mask of
                   processor i, RTL fault injected here
18             if eirq /= 0 then -- check if extended interrupts are also
                   handled
19               v2.imask(i) := apbi.pwdata(31 downto 16); -- in this case
                     also update the extended interrupt lines of the mask
                     register
20             end if;
21           end if;
22         end loop;
23       -- ...
24       end case;
25     end if;
26
27     -- ... register new interrupts, interrupt acknowledge, reset ...
28 end process;
```

Fig. 3 Example IRQMP code excerpt in VHDL showing register configuration, to illustrate fault injection on RTL (*line with fault injection highlighted*)

```
1  // generic (using templates) and re-usable TLM register class, the
       inherited class provides the basic interface and some default
       implementation
2  template<typename DATA_TYPE>
3  class sr_register : public sc_register_b<DATA_TYPE> {
4    //...
5  public:
6    void bus_write(DATA_TYPE i) {
7      // callbacks notify observers about register access directly without
           context switches
8      raise_callback(SR_PRE_WRITE);
9      // corresponding TLM error location - update the internal register
           value, the write mask allows selective updates
10     this->write(i & m_write_mask);
11     // the IRQMP will trigger interrupt re-computation after the mask
           register is updated
12     raise_callback(SR_POST_WRITE);
13   }
14   //...
15 }
16
17 // a register bank groups multiple registers - similarly, this is a
       generic implementation
18 template<typename ADDR_TYPE, typename DATA_TYPE>
19 class sr_register_bank : public sc_register_bank<ADDR_TYPE, DATA_TYPE> {
20   typedef typename std::map<ADDR_TYPE, sr_register<DATA_TYPE> *>
         register_map_t;
21   register_map_t m_register; // use a mapping (address to register) to
         store registers
22   //...
23 public:
24   // bus read/write transactions matching a register address are
         automatically redirected to this class
25   bool bus_write(ADDR_TYPE offset, DATA_TYPE val) {
26     sr_register<DATA_TYPE> *reg = get_sr_register(offset); // retrieve
           register for the given (bus) address
27     if(reg) {
28       reg->bus_write(val); // update register value
29     }
30     return true;
31   }
32   //...
33 }
34
35 // register bank used as member variable in the IRQMP
36 sr_register_bank<unsigned int, unsigned int> r;
```

Fig. 4 TLM code for register configuration, showing a corresponding TLM error (*line highlighted*) for Fig. 3

logic of the IRQMP is available in the combinatorial process *comb*, shown in Fig. 3. Essentially, it contains the whole logic of the RTL model and is triggered whenever some input or internal signal changes. It is responsible for interrupt prioritization, processing of register read/write requests and interrupt acknowledgements, and writing output signals. Internal signal values are updated in a separate process at every clock cycle. In particular Fig. 3 shows processing code for a bus write request to the CPU specific *mask* register for normal (Line 17) and extended interrupts (Line 19). The target register and processor are encoded in the bus address signal,

they are decoded in Line 12 and Line 16, respectively. In this example a fault is injected in Line 17 during *mask* register configuration.

The TLM implementation, shown in Fig. 4, of the IRQMP keeps a register bank (Line 36), which essentially contains a mapping of an address value to a register (Line 21). A bus write transaction will call the *bus_write* function of the register bank (Line 25–31). The function will retrieve the target register directly based on the write address in Line 26 and dispatch the write access to the register class in Line 28. The register will finally update its internal value in Line 10, which is the corresponding TLM error. Before and after the update, callback functions are used to notify the IRQMP and update its internal state directly without any context switches. In this case the main processing thread of the IRQMP will be notified to recompute outgoing interrupts. Please note that the register bank implementation is not specific to the IRQMP but a generic and reusable implementation. Furthermore, the TLM model can update the whole register, while the RTL model only updates bits 15:1 when extended interrupts are ignored. This is not a problem, since the prioritization logic of the TLM model simply ignores the extended interrupt lines in this case and therefore produces the same failure (when the corresponding error is injected) as the faulty RTL model.

5 Formal Fault Localization Analysis

This section describes our formal fault localization analysis, to obtain candidates for corresponding TLM errors, in more detail. We reduce this problem to a verification problem of assertion violations by encoding error injection selection nondeterministically and adding appropriate constrains to prune invalid solutions. Then we employ symbolic simulation for an efficient exhaustive exploration to find all possible solutions. In general different formal verification techniques besides symbolic simulation could also be used to find solutions.

Figure 5 shows an overview of our approach. The analysis requires the TLM model as well as the input and output of the faulty RTL model (marked grey in Fig. 5). We assume that the TLM model is available in the XIVL format to apply formal analysis techniques. The TLM model contains annotations for a fine grained selection of instruction where error injection can take place (see 1 in Fig. 5).

First a testbench is generated by using the input and output to construct an input driver and result monitor, respectively (see 3 in Fig. 5). The testbench is also available in the XIVL format and contains the simulation entrypoint—the main function—which is responsible to setup all components. The result monitor contains the assertions which constrain valid solution.

Then the TLM model and testbench are automatically combined to a complete TLM model. During this process, the TLM model will be instrumented with symbolic error injection logic to nondeterministically select an error injection location for a transient one bit error (see 3 in Fig. 5). The annotations on the TLM model are used to guide the instrumentation.

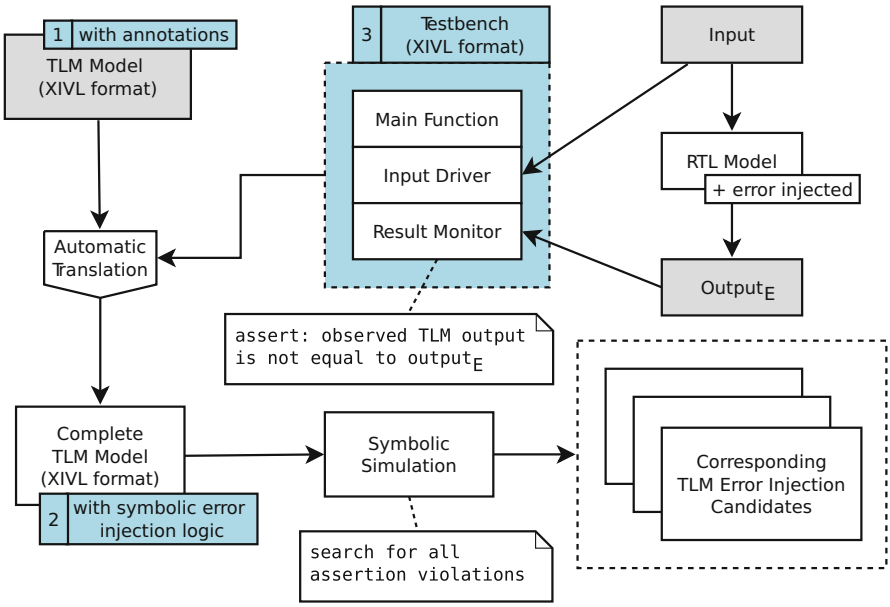

Fig. 5 Formal fault localization analysis overview

Finally, the complete TLM model is passed to our symbolic simulation engine for a formal analysis. Based on the symbolic error injection logic instrumented into the TLM model, the symbolic simulation will find and report all concrete error injection locations that will cause the TLM model to produce the same failure, i.e., same output, as the faulty RTL model for the given input.

In the following we will discuss (1) annotations, (2) symbolic error injection logic, and (3) testbench encoding in more detail.

5.1 Annotations

We use annotations in the TLM model for a fine grained control of error injection. Assignment instructions are annotated to denote that error injection can take place. During instrumentation, every annotated assignment will be modified to either stay unchanged or toggle a bit flip in the result—based on a nondeterministic choice during analysis (at runtime). For convenience, we also support function annotations. These will be propagated to all assignments in the function. Using annotations is a flexible approach to control error injection more precisely. These annotations can happen manually, or using a static analysis that modifies the code automatically, e.g., based on a specification provided by the user. This ensure that meaningful locations for error injection are reported—otherwise the initial state or the output

values would be modified. Furthermore, injection can be selectively activated and deactivated at runtime, based on a boolean global variable. This allows to run the same code blocks, e.g., functions, with and without error injection. We use it to deactivate error injection during initialization of the TLM model. The reason is that the same code is also used during testbench specific configuration, where error injection should be allowed.

We use a reusable modeling layer for registers and wires based on [15], which are used by many TLM peripheral models including the IRQMP. Error injection in bus and wire transfer operations, as well as many computations can be handled by injecting errors in the modeling layer itself. An example for the register class is shown in Fig. 6. The *injectable* annotation on the *set_bit* function is propagated to both assignments.

```
1  // XIVL implementation of the TLM register class
2  struct Register {
3    // write mask allows selective updates on bus access
4    uint32_t value;
5    uint32_t write_mask;
6
7    bool get_bit(Register *this, uint32_t index) {
8      return this->value & (1 << index);
9    }
10
11   @injectable
12   void set_bit(Register *this, uint32_t index, bool value) {
13     // injectable annotation is automatically propagated to both
           assignments
14     if (value) {
15       // set bit
16       this->value |= 1 << index;
17     } else {
18       // clear bit
19       this->value &= ~(1 << index);
20     }
21   }
22
23   uint32_t read(Register *this) {
24     return this->value;
25   }
26
27   void write(Register *this, uint32_t value) {
28     // errors can be injected at assignments marked with @injectable
29     @injectable this->value = value;
30   }
31
32   void bus_write(Register *this, uint32_t value) {
33     // ... PRE/POST write callback handling omitted in this example ...
34     @injectable this->value = value & write_mask;
35   }
36 }
```

Fig. 6 Excerpt of an annotated TLM register class in XIVL

5.2 Symbolic Error Injection Logic

We integrate a set of global variables and functions, shown in Fig. 7, into the complete TLM model for symbolic error injection. In particular the functions *inject_bitvector_error* and *inject_bool_error* are used from within the TLM model. Consider again the example in Fig. 6. For example, the assignment *@injectable this->value = value;* in the *write* function in Line 29 is transformed to *this->value = inject_bitvector_error(value, 32);* when annotations are resolved during instrumentation.

The *inject_bitvector_error* function (defined in Fig. 7) expects two arguments, an integer and its bitwidth. The bitwidth argument denotes the range of bits from which one is selected nondeterministically for flipping. The bitwidth argument is automatically generated based on static type informations during the instrumentation process, which rewrites the annotations. By using a 64 bit integer type as argument and return value, we also automatically support all integer types with smaller bitwidth. For this case study, 32 bit values are sufficient. The *inject_bool_error* function in principle works analogously.

For convenience we use short names in Fig. 7 for the global variables, in the TLM model they have a unique name prefix to avoid name clashes with existing code. The boolean *active* variable allows to selectively toggle error injection and thus control it more precisely. It is accessed by means of the *activate/deactivate_injection* functions defined in Line 8 and Line 4, respectively.

The *condition* variable is initialized with a symbolic integer value in Line 15 and together with the *id* variable allows nondeterministic selection of an injection location. This works as follows: Consider an invocation of *inject_bitvector_error* and assume that *active* is true. If no error has been injected yet, then both branch directions in Line 33 are feasible, i.e., *condition* = *id* can evaluate to *true* and *false*. Therefore, symbolic execution will split into two independent paths S_T and S_F, respectively, and explore both branch directions. This will update the path conditions of S_T and S_F with *condition* = *id* and *condition* $\neq$ *id*, respectively. Please note, that *id* is a concrete integer value in this case, e.g. the number 4. Since *id* is incremented on every call of *inject_bitvector_error*, the true branch of the *if* statement in Line 33 becomes infeasible for the S_T path and all its descendants. Therefore, at most a single error is injected on every execution path. The *location* variable stores a copy of the injection *id* for debugging purposes.

The *@track* instruction is specifically recognized by our symbolic simulation engine and records a snapshot of all instruction pointers of the callstack of the currently executed thread, i.e., essentially the currently executed instruction in the active thread and all of its called functions. This allows to pinpoint the error injection location for later inspection.

The function *flip_single_bitvector_bit* is used in Line 39 as a helper function to inject a single bit error into an integer variable. It creates and constrains a symbolic integer value to nondeterministically select a single bit in the range defined by the *bitwidth* argument (Line 20–21). The global *bit* variable records

```
1  // dynamically toggle error injection
2  bool active = false;
3
4  void deactivate_injection() {
5    active = false;
6  }
7
8  void activate_injection() {
9    active = true;
10 }
11
12 // variables store injection choices for inspection and ensure that only
       single error is injected
13 int id = 0;
14 int location = -1;
15 int condition = ?(int);
16 int bit = -1;
17
18 int64_t flip_single_bitvector_bit(int64_t val, uint8_t bitwidth) {
19   // non-deterministically choose a single bit in *bitwidth* to flip
20   uint8_t x = ?(uint8_t);
21   assume (x >= 0 && x < bitwidth);
22   // perform the actual bit flip
23   val = val ^ (1 << x);
24   // store choice for later inspection
25   bit = x;
26   // result is symbolic due to non-deterministic choice
27   return val;
28 }
29
30 int64_t inject_bitvector_error(int64_t val, uint8_t bitwidth) {
31   // unique id ensures only a single error is injected
32   id += 1;
33   if ((condition == id) && active) {
34     // record the injection choice for later inspection
35     @track "one bit error injection";
36     location = id;
37
38     // perform a non-deterministic bit flip
39     val = flip_single_bitvector_bit(val, bitwidth);
40   }
41   return val;
42 }
43
44 bool inject_bool_error(bool val) {
45   // essentially similar to injecting a bitvector error
46   id += 1;
47   if ((condition == id) && active) {
48     @track "boolean error injection";
49     location = id;
50
51     val = !val;
52   }
53   return val;
54 }
```

Fig. 7 Encoding details for transient one bit error injection

the nondeterministic choice in Line 25 for later inspection (eventually the SMT solver will provide concrete values for nondeterministic choices). Based on the nondeterministic choice, the function performs the bit flip in Line 23 and returns the result. Please note that the result itself becomes a symbolic expression.

5.3 Testbench

This section provides more details on the testbench focusing on assertion generation to guide the formal analysis. For illustration purpose, we discuss a (simplified) concrete example testbench for the IRQMP. Essentially, the input specifies incoming interrupts for the IRQMP and the output is a prioritized list of interrupt requests generated by the IRQMP.

When sending the interrupt mask $0b110$ as input and injecting a fault in the RTL model that results in wrong prioritization, the output $[2, 3]$ is observed instead of the expected output $[3, 2]$—since higher interrupt lines have higher priority. Based on the input and faulty output the testbench is constructed. The monitoring logic records the observed interrupts in an array *irq*. Furthermore, it keeps track of the number of received interrupts in the *num_irqs* variable. Finally, the monitor asserts that $((irq[0] \neq 2) \,||\, (irq[1] \neq 3) \,||\, (num\_irqs \neq 2))$ holds at the end of simulation. Essentially, it asserts that the observed output for the TLM model is not equal to the output of the faulty RTL model. Thus, the symbolic simulation engine will search for all possible error inject locations that violate the assertion, i.e., produce the same failure at TLM as the faulty RTL model.

As an optimization, to prune irrelevant search paths which cannot produce the output of the faulty RTL model, we place *assume* instructions in the monitor. For this example, we would assume that the first received interrupt is 2 and the second is 3. Furthermore, we would assume that $num\_irqs < 2$. Then a simple *assert (false);* can be placed at the end of simulation. Using stepwise assumptions during symbolic simulation, instead of a single assert in the end, can significantly reduce the considered search space, by pruning irrelevant search paths early.

6 Case Study

We have evaluated our proposed fault correspondence analysis on the IRQMP model from the SoCRocket VP [22] as a case study. Our formal fault localization analysis is based on symbolic simulation approach of [14, 15]. In the following we report first experimental results for our proposed approach.

6.1 Experiments

All experiments were performed on an Intel 2.6 GHz machine with 16 GB RAM running Linux. We employ Z3 v4.4.1 as our SMT solver.

For the experiments we use a set of representative test scenarios to cover different functionality of the IRQMP. Every scenario is using different inputs. Incoming interrupts are sent from the testbench to the IRQMP via register access (using a bus-transfer operation) or by writing to the *irq_in* wire. Furthermore, different priority levels are tested and resending of interrupt requests. Please note, all tests only use a single CPU and do not consider extended interrupts. The reason is that the RTL and TLM model are not functionally equivalent when using these features.

We use a set of representative fault injection locations for the RTL model. For every of these fault locations, a fault is injected in the RTL model for every test scenario. Table 2 shows a summary of our experimental results for injecting a bit flip during register configuration, interrupt prioritization computation as well as interrupt sending and acknowledgment. In particular, the following faults are injected:

1. a bit is flipped in the mask register, therefore the corresponding interrupt line is not processed or additional interrupt line activated;
2. a bit is flipped in the force register, therefore an additional interrupt needs to be processed or is omitted;
3. an incoming interrupt line is flipped, which has similar effect as the previous fault, but covers different functionality of the IRQMP;

Table 2 Combined results of experiments (runtimes in seconds)

RTL fault location	Type	Average runtime		Sym. error injections	SMT queries	TLM errors		
		TOTAL	SMT			Max	Min	Result
(1) Mask register configuration	BT	285.86	184.18	162	30,992	6	1	1
(2) Force register configuration	BT	317.88	210.14	196	34,690	7	4	4
(3) Incoming interrupt wire	WT	317.80	207.18	166	35,057	6	2	2
(4) Incoming interrupt wire reset	WT	690.74	454.03	311	74,595	0	0	0
(5) Missing acknowledgement	WT	395.10	262.00	211	42,839	7	6	6
(6) Wrong acknowledgement	WT	363.81	241.43	211	41,502	1	1	1
(7) Prioritization logic 1	Comp	209.41	154.01	127	21,856	12	4	4
(8) Prioritization logic 2	Comp	373.88	242.48	201	40,538	9	0	0

4. a bit is flipped when resetting the incoming interrupt lines to zero;
5. the activation signal of the active interrupt acknowledgment wire is flipped, therefore an acknowledgment is missed;
6. a bit is flipped when sending the acknowledgment, this results in a wrong interrupt being acknowledged.
7. a bit is flipped when computing the interrupt prioritization, therefore a wrong interrupt is sent or the order of two incoming interrupts is changed (e.g., flipping the second bit in 0b11 results in 0b01, thus interrupt line 1 is sent first, even though line 2 has higher priority—since line 2 is still in the pending register, it will eventually be sent out too);
8. similar to the previous one, but at different location, where interrupts are interpreted as number instead of lines, e.g., 0b11 is interpreted as number 3 instead of interrupt lines 1 and 2;

Essentially, Table 2 combines the analysis results for all test scenarios for every fault injection location and reports the average values over all analysis runs. The first column shows a description of the RTL fault. The second column shows the type of the fault: *BT=Bus Transfer*, *WT=Wire Transfer*, and *Comp=Computation*. The third and fourth columns show the average runtime (in seconds), which is further divided into total analysis and SMT time. The SMT time shows how much of the analysis time is spent with solver queries. The number of solver queries is reported in the column *SMT Queries*. The column *Sym. Error Injections* shows how many errors have been injected during analysis of the TLM model. Please note that every error injection represents a nondeterministic bit flip. Finally, the column *TLM Errors* shows the maximum, minimum, and result of error injection locations detected on the TLM model. The result denotes the number of TLM errors when computing the intersection of TLM errors for every test scenario. In other words the result column denotes the number of candidates for corresponding TLM errors found by our analysis.

For the RTL faults 4 and 8, no corresponding TLM errors are found (result column contains 0). The reason is that the TLM model is designed on a higher level of abstraction. For performance reasons callbacks are used that directly modify the internal model state without any delta cycles and context switches. Therefore, in case of RTL fault 4, a reset of the incoming interrupt lines is not necessary (and not available in the TLM model - setting the interrupt lines to a specific value is only applied once and not permanently until change). In the other case (RTL fault 8) it is due to a signal line where an error effect is delayed and propagated across a delta cycle. For both cases, it is possible that the RTL fault corresponds to multiple simultaneous TLM errors. For their localization, the symbolic error injection logic must be extended to support multiple errors. This extension is left for future work.

7 Conclusion

In this chapter we proposed an RTL-to-TLM fault correspondence analysis to improve the quality of error effect simulation using VPs. We employ formal methods to identify TLM error injection candidates for transient bit flips at RTL. First experiments on the IRQMP from the SoCRocket VP demonstrated the applicability and effectiveness of our approach in finding a small set of candidates for corresponding TLM errors.

Acknowledgements This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project EffektiV under contract no. 01IS13022E, by the German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1, and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

References

1. Beltrame, G., Bolchini, C., & Miele, A. (2009). Multi-level fault modeling for transaction-level specifications. In *GLSVLSI* (pp. 87–92).
2. Bombieri, N., Fummi, F., & Guarnieri, V. (2012). FAST: An RTL fault simulation framework based on RTL-to-TLM abstraction. *Journal of Electronic Testing, 28*(4), 495–510.
3. Bombieri, N., Fummi, F., & Pravadelli, G. (2008). A mutation model for the SystemC TLM 2.0 communication interfaces. In *DATE* (pp. 396–401).
4. Cho, H., Mirkhani, S., Cher, C. Y., Abraham, J. A., & Mitra, S. (2013). Quantitative evaluation of soft error injection techniques for robust system design. In *DAC* (pp. 1–10).
5. Chou, C. N., Chu, C. K., & Huang, C. Y. R. (2013). Conquering the scheduling alternative explosion problem of SystemC symbolic simulation. In *ICCAD* (pp. 685–690).
6. Chou, C. N., Ho, Y. S., Hsieh, C., & Huang, C. Y. (2012). Symbolic model checking on SystemC designs. In *DAC* (pp. 327–333).
7. Flanagan, C., & Godefroid, P. (2005). Dynamic Partial-Order Reduction for model checking software. In *POPL* (pp. 110–121).
8. Godefroid, P. (1996). *Partial-order methods for the verification of concurrent systems: An approach to the state-explosion problem*. Berlin/Heidelberg: Springer.
9. Griesmayer, A., Staber, S., & Bloem, R. (2007). Automated fault localization for C programs. *Electronic Notes in Theoretical Computer Science, 174*(4), 95–111.
10. GRLIB IP library. http://www.gaisler.com/index.php/products/ipcores/soclibrary
11. Herdt, V., Le, H. M., Große, D., & Drechsler, R. (2016). Compiled symbolic simulation for SystemC. In *ICCAD*.
12. IEEE (2011). *IEEE standard SystemC language reference manual* (p. 1666). Piscataway: IEEE Standards Association.
13. Le, H. M., Große, D., & Drechsler, R. (2012). Automatic TLM fault localization for SystemC. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 31*(8), 1249–1262.
14. Le, H. M., Große, D., Herdt, V., & Drechsler, R. (2013), Verifying SystemC using an intermediate verification language and symbolic simulation. In *DAC* (pp. 116:1–116:6).
15. Le, H. M., Herdt, V., Große, D., & Drechsler, R. (2016). Towards formal verification of real-world SystemC TLM peripheral models - a case study. In *DATE* (pp. 1160–1163).
16. Lee, D., & Na, J. (2009). A novel simulation fault injection method for dependability analysis. *IEEE Design Test of Computers, 26*(6), 50–61.

17. Li, M. L., Ramachandran, P., Karpuzcu, U. R., Hari, S. K. S., & Adve, S. V. (2009). Accurate microarchitecture-level fault modeling for studying hardware faults. In *HPCA* (pp. 105–116).
18. Lisherness, P., & (Tim) Cheng, K. T. (2010). SCEMIT: A SystemC error and mutation injection tool. In *DAC* (pp. 228–233).
19. Miele, A. (2014). A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems. *Microprocessors and Microsystems, 38*(6), 567–580.
20. Oetjens, J. H., Bannow, N., Becker, M., Bringmann, O., Burger, A., Chaari, M., et al. (2014). Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges. In *DAC* (pp. 1–6).
21. Perez, J., Azkarate-askasua, M., & Perez, A. (2010). Codesign and simulated fault injection of safety-critical embedded systems using SystemC. In *EDCC* (pp. 221–229).
22. Schuster, T., Meyer, R., Buchty, R., Fossati, L., & Berekovic, M. (2014). SoCRocket - A virtual platform for the European Space Agency's SoC development. In *ReCoSoC* (pp. 1–7).
23. Sen, A., & Abadir, M. S. (2010). Coverage metrics for verification of concurrent SystemC designs using mutation testing. In *HLDVT* (pp. 75–81).
24. Shafik, R. A., Rosinger, P., & Al-Hashimi, B. M. (2008). SystemC-based minimum intrusive fault injection technique with improved fault representation. In *IOLTS* (pp. 99–104).

Error-Based Metric for Cross-Layer Cut Determination

A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud, A. Romanovsky, and A. Yakovlev

Abstract With the increase of system complexity in both platforms and applications, power modelling of heterogeneous systems is facing grand challenges from the model scalability issue. To address these challenges, this chapter studies two systematic methods: selective abstraction and stochastic techniques. The concept of selective abstraction via black-boxing is realised using hierarchical modelling and cross-layer cuts, respecting the concepts of boxability and error contamination. The stochastic aspect is formally underpinned by Stochastic Activity Networks (SANs). The proposed method is validated with experimental results from Odroid XU3 heterogeneous 8-core platform and is demonstrated to maintain high accuracy while improving scalability.

Keywords Extra-functional properties • Power modelling • Task affinity • Hierarchical modelling • Selective abstraction • Model scalability • Error contamination • Order Graphs • Cross-layer cut • Stochastic Activity Networks • Möbius tool • Monte-Carlo simulation • Heterogeneous systems • ARM big.LITTLE • Odroid XU3

1 Introduction

Continued scaling of semiconductor technology has caused an increase of computing system capabilities, and with it, a seemingly unstoppable expansion of system application space. This is leading to a rapid increase of system complexity and diversity, exacerbating the scalability of system modelling. A typical example of such complex and diverse systems is multi-core heterogeneous platforms.

A. Rafiev (✉) • F. Xia • A. Iliasov • R. Gensh • A. Aalsaud • A. Romanovsky • A. Yakovlev
Newcastle University, Newcastle upon Tyne, UK
e-mail: ashur.rafiev@ncl.ac.uk; fei.xia@ncl.ac.uk; alexei.iliasov@ncl.ac.uk; r.gensh@ncl.ac.uk; a.m.m.aalsaud@ncl.ac.uk; alexander.romanovsky@ncl.ac.uk; alex.yakovlev@ncl.ac.uk

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_4

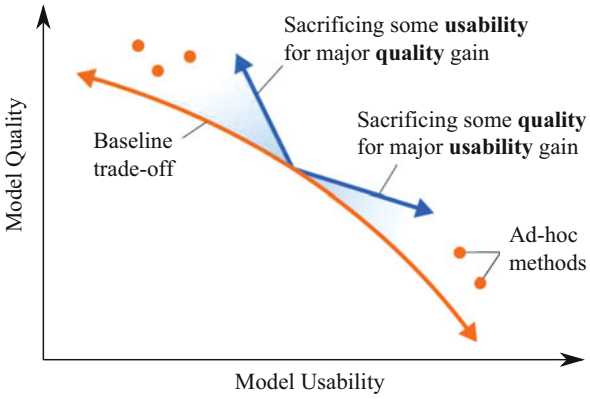

Fig. 1 The realm of complexity control

Addressing the model complexity is highly challenging due to the trade-off between *quality* (i.e. the accuracy, precision, and fidelity) of the model and its *usability* (defined by scalability, computation complexity, and design effort), as shown in Fig. 1.

Modelling *non-functional properties*, e.g. power dissipation, is as crucial as modelling *functional properties*, such as operational correctness [2]. Non-functional models typically include functional representations. A widely used method of modelling power uses Virtual Prototypes (VP) to generate states from a functional simulation for use in power simulators forming a co-simulation [8, 15].

Analogue hardware models such as SPICE models provide some of the highest representational quality, but they are not usable for studying entire computers with software running on hardware. For such studies, discrete event models, such as Instruction Set Architecture- (ISA-)accurate [3], cycle-accurate, and RTL models [16], are useful when studying functional properties. Instruction Set Simulators (ISS), however, commonly have simulation speeds of the order of a few Million Instructions Simulated per Second (MISPS) [15, 17], and this puts a limit on their usability when the system scales to many-cores, especially for statistical analysis [4].

There have been, as a result, significant efforts in model simplification for power studies. One way of simplifying away from ISA- or cycle-accuracy targeting multiple cores is extrapolation [5]. In this approach, a typical subsystem (e.g. a single core) is fully characterised and represented with a complete model, and other similar cores are represented by simplified models obtained through extrapolation. However, this method tends to be less effective for highly heterogeneous systems. It is also possible to depart from ISA-accuracy through abstraction. For instance, Transaction Level Modelling (TLM) concentrates on the functional properties of larger system blocks [1, 10].

Functional modelling can be highly abstracted by using stochastic techniques, shrinking models by regarding system behaviours as stochastic, rather than deterministic [4].

Non-functional parameters like power also open up further ways to systematic model simplifications based on what may be called the "significance factor" [12]. During any particular operation of a heterogeneous system consisting of multiple parts, some parts of the system may consume more power than other parts. If a quantitative power model is to be precise to a certain degree, it makes sense to make the model power-proportional by using a simpler model for a less power hungry part and a detailed model for a more power hungry part. This approach can be broadly described as *selective abstraction*, i.e. the level of abstraction (and therefore the cost and quality) of each part of the model depends on the part's contribution to the parameter under study.

This work aims towards the scalable modelling of multi-core heterogeneous systems by concentrating on stochastic modelling and selective abstraction. By doing so, we seek to support designers to systematically traverse the trade-off space between modelling quality and model scalability in "good" trajectories, shown in Fig. 1 as vectors, as opposed to *ad-hoc* techniques targeting specific points in this trade-off.

1.1 Research Methodology and Contributions

In order to validate the presented ideas, we created a model of a real hardware multi-core heterogeneous system using a mature stochastic modelling method, applied the proposed method of selective abstraction to optimise the model for scalability, and evaluated the cost and the accuracy against the actual measurements. Stochastic Activity Networks (SANs) [14] is a well-known modelling method with an extensive support by the Möbius tool [18] provides the capabilities to model power as a reward function, thus has been used to facilitate the stochastic modelling aspect of the method. Odroid XU3 development board [11] based on the ARM big.LITTLE architecture has been selected as the target system for modelling.

The following contributions have been made:

- We developed new structuring methods to tackle complexity and scalability in modelling by providing a power-proportionality metric for selective abstraction and methods to retain accuracy by avoiding error contamination.
- We validated these methods using power modelling in SANs and showed their effectiveness in improving the trade-offs between accuracy, scalability, and usability.

The chapter is organised as follows. Section 2 describes the workflow and the method. Section 6 presents the design of supporting experimental studies. Section 7 describes the developed model and discusses the results from the simulation results. Section 8 concludes the work.

2 The Proposed Method

The workflow of the proposed method is illustrated in Fig. 2. A hierarchical representation of the system resources in the form of an Order Graph (OG) [12] is derived from the system design knowledge. The behaviour of the system is captured in a detailed SAN model [14]. System power characteristics, usually obtained from initial experiments, are applied to the OG to compute the power-proportionality metric for selective abstraction, which determines the regions for black-boxing in the SAN model. The next step is to combine OG and SAN to capture dependencies within the model in the form of a graph, which will help identify any error contamination. If no contamination found, the SAN model can be used further in power studies, e.g. simulations. In case of error contamination, the designer has to redo the abstraction selection with the updated knowledge on the model's boxability, or even redesign the system.

2.1 Hierarchical Modelling and Selective Abstraction

Hierarchical representations have been used for modelling complex systems for a long time. The idea of separating the "vertical" relation between the layers of abstraction from the "horizontal" knowledge of the system at each particular layer of abstraction has been hinted in [21] and then formally defined in Zoom structures [6] as the concepts of *verticality* and *horizontality*. Zoom structures are based on partial

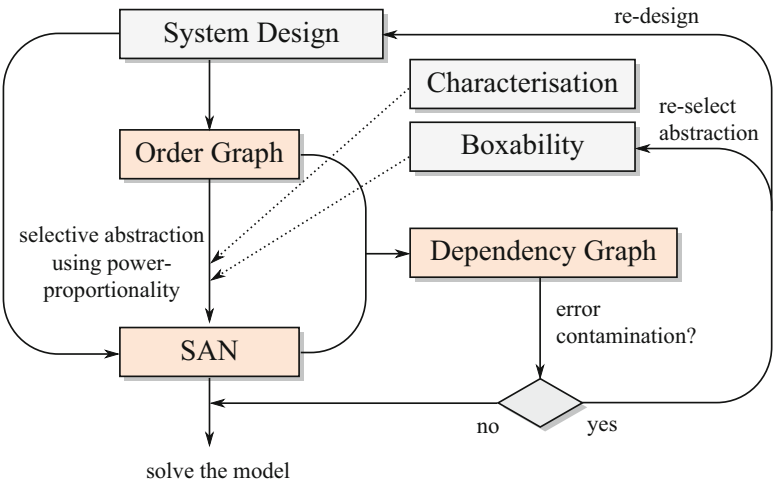

Fig. 2 The flowchart of the proposed method

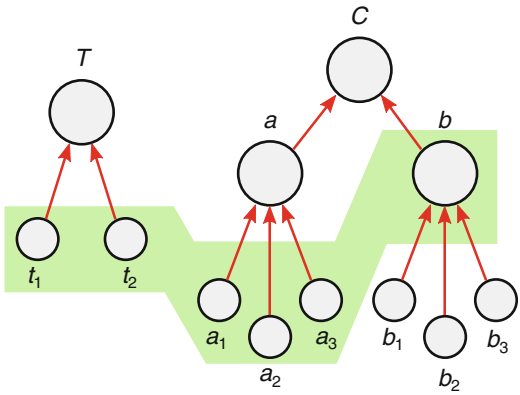

Fig. 3 Example vertical view of an Order Graph with a cross-layer cut

orders and are very permissive. In contrast, OGs put a number of constraints on the modelling, which guarantee consistency between the abstraction layers.

An OG is a graph with nodes representing various system resources arranged in tree hierarchies; different concepts can be represented by separate trees. For example, in Fig. 3, T can represent the hierarchy of tasks, and C—computational units. The hierarchies can be built from the knowledge of the system structure and by similarities of its constituents. The distance from the root relates to the level of abstraction. The formal definition and properties can be found in Sect. 3.

OG contains the static knowledge of the system and needs to be paired with a dynamic model to capture the system behaviour (in our case: SANs). Each branch of a tree must have a representative element in the dynamic model, but multiple nodes from a single branch cannot be included. The nodes in OG that are included in this model relation form a *cut*. If the cut goes through different depths in the hierarchy (layers of abstraction), it is called a *cross-layer cut*. The cut containing all leaves relates to the most concrete (detail) model of the system.

3 Hierarchical Modelling in Order Graphs

The following discussion revisits the definition of a hierarchy as a sequence of model transformations, which thereafter is applied to graph models leading to Order Graphs. The latter combines the notions of resource modelling with the hierarchical representation of system layers.

3.1 Introducing Hierarchies

An underlying approach for having adjustable fidelity in the models relies on different levels of abstraction. Naturally, these layers have to be consistent with each other; however, the very definition of consistency may vary from model to model and depend on the system properties that need to be preserved.

A common way to define a model of a system is to represent it as a set tuple $M = (E_1, E_2, \ldots E_n)$, where each set E_k contains system elements of a particular type, e.g. vertices, edges, labels, etc. We can also generalise these to a single type—"system elements", $\mathcal{E}$—so $E_1 \subset \mathcal{E}, E_2 \subset \mathcal{E}, \ldots$. Thus, we can have a type-agnostic representation of a model: $\mathcal{M} = E_0 \cup E_1 \cup \ldots \cup E_n$.

Let M_a and M_b be some system models with corresponding sets of system elements $\mathcal{M}_a, \mathcal{M}_b$, and some relation between these elements $\gamma \subseteq \mathcal{M}_a \times \mathcal{M}_b$. Given a boolean predicate Φ, such that:

$$\Phi : \mathbb{P}(M_a) \times \mathbb{P}(M_b) \times \mathbb{P}(\mathcal{M}_a \times \mathcal{M}_b) \to \{0, 1\}, \tag{1}$$

the relation γ is called a *consistency relation* between models M_a and M_b under the predicate Φ if $\Phi(M_a, M_b, \gamma) = 1$. Φ is called the *rule set*, and for convenience can be specified as a conjunction of smaller predicates of the same type (1).

The predicate Φ is called *strongly consistent* if it requires γ to be a total surjective relation, i.e. for every element in $\mathcal{M}_a$ there must be at least one related element in $\mathcal{M}_b$, and for every element in $\mathcal{M}_b$ there must be at least one related element in $\mathcal{M}_a$. In this case, γ is called a *transformation*; transformations are further denoted as $\gamma = \mathcal{M}_a \vdash \mathcal{M}_b$ (or $\gamma = M_a \vdash M_b$ since $\mathcal{M}_a, \mathcal{M}_b$ are derived from M_a and M_b).

Let $\{\ldots, M^{(k-1)}, M^{(k)}, M^{(k+1)}, \ldots\}$ be an infinite or finite set of models of the same system, where each $M^{(k)}$ models the system in a specific level of details. An *abstraction hierarchy* is a total order of models where any two adjacent models form a transformation $\gamma_k = M^{(k)} \vdash M^{(k+1)}$ under a given strongly consistent predicate Φ_k, and the size of models monotonically decreases (or increases) with k:

$$\mathcal{H} = \ldots \vdash M^{(k-1)} \vdash M^{(k)} \vdash M^{(k+1)} \vdash \ldots \tag{2}$$

Each $M^{(k)}$ is k-th level of abstraction, also called *order* k.

A hierarchy is called *homogeneous* if it uses the same rule set Φ for all its consistency relations; this implies that $\mathbb{P}\left(M^{(k)}\right) = \mathbb{P}\left(M^{(k+1)}\right)$ for all k.

Every hierarchy contains both horizontal and vertical knowledge: each abstraction layer $M^{(k)}$ is a horizontal view of the system, while the set of relations $\{\ldots, \gamma_k, \gamma_{k+1}, \ldots\}$ stores the information on how different layers interlink. Notions of horizontality and verticality can be found in [6].

Figure 4a shows the conventional approach to hierarchical graphs, which is based on clustering and uses tree structures [9]. Each node of a higher layer zooms into a subgraph in a lower layer. Consequently, an edge between two nodes becomes multiple edges between the corresponding subgraphs. The notation used in the

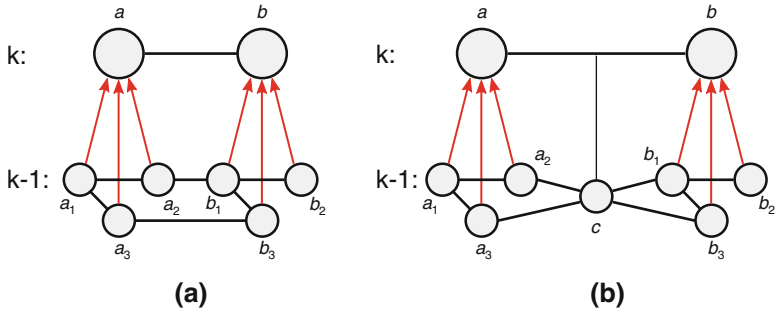

Fig. 4 Conventional hierarchy representation (**a**) compared to Order Graphs (**b**); k is the higher level of abstraction and $k-1$ is the lower level

diagram is based on Zoom Structures [6]. A convenient way to display graph hierarchies is zoom views, showing verticality and horizontality with vertical and horizontal arcs, respectively. The following is a redefinition of hierarchical graphs in the terms presented in Sect. 3.1.

A *Hierarchical graph* is a homogeneous hierarchy, such that, each k-th order is a graph $G^{(k)} = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges; and all consistency relations in this hierarchy are defined as follows:

Inclusion function represents vertex clustering by relating multiple vertices in the lower order to a single vertex in a higher order.

Supplementary inclusion function ensures that all edges within a cluster are also included, i.e. if two vertices in the lower order relate to the same vertex in the higher order, any edge connecting them is automatically related to the same high-level vertex.

Edge grouping function groups edges connecting vertex clusters: an edge in the lower order connects vertices iff there is an edge in the higher order connecting related high-order vertices.

A more formal definition of these rules can be found in [12]. The inclusion function can be chosen arbitrarily, and from it, the other two uniquely describe the edges in the hierarchical graph.

The most important property of the rule set defined above is that it preserves all paths in the graph during the mapping. In other words, for any vertices $v_1, v_2 \in V$ and related vertices $v_1', v_2' \in V'$, if there exists a path between v_1 and v_2 in $G^{(k)}$, there will be a path between v_1' and v_2' in $G^{(k+1)}$, and vice versa:

$$\forall v_1, v_2 \in V, v_1', v_2' \in V' : \gamma_v(v_1) = v_1' \wedge \gamma_v(v_2) = v_2' \Rightarrow \left(P(v_1, v_2) \Leftrightarrow P\left(v_1', v_2'\right)\right), \tag{3}$$

where $P(x, y)$ is a function that is true iff there is a path between x and y. This property ensures that the dependencies between resources are consistent throughout the hierarchy.

3.2 Order Graphs

The central subject of our method is the study of a computational platform comprising a number of diverse resources and the way resources may be handled in order to realise a computation. Can we say that the edges of a graph are also resources? It is actually true, and this contradiction is explained and solved by OGs. As an example, let's imagine that Fig. 4a models a network interaction, where a is a server and b is a client. On the very abstract level we do not care about the structure of the network, we just need to know that the client and the server are connected somehow, thus we model this entire system as the client and the server connected directly with a single dependency. However, in a more detailed model we can no longer ignore the network protocols and have to introduce it at least as a single resource node as shown in Fig. 4b.

A distinct property of the proposed OG modelling method is that a high-order edge relates to a node at the next lower order. In this case we say that the node *supports* an edge, while in fact this is the same entity viewed from the different abstraction levels. In real-life systems, any dependency is always supported by a resource of some kind, and this "fractal" structure goes down to the smallest details, including atoms and below. We may not want to include all these in the model, and this is pragmatically solved by saying that an edge is either supported by a resource at the lower layer or stays an edge like in conventional hierarchical graphs.

An *Order Graph* is a homogeneous hierarchy, such that, each k-th order is a graph $G^{(k)} = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges; and all consistency relations in this hierarchy are defined as follows:

Inclusion, **supplementary inclusion**, and **edge grouping** are defined as in Sect. 3.1.

Support function is a one-to-one mapping of some vertices onto some of the edges of a higher order graph. The first rule on this function tells that we can map a vertex in the lower order to some edge $\langle v_1, v_2 \rangle$ in the higher order iff this vertex is connected to at least one vertex related to v_1 and at least one vertex related to v_2. In addition, all vertices adjacent to v must be related either to v_1 or v_2. Finally, the same vertex cannot be used in a vertex-to-vertex and a vertex-to-edge relation; and the same higher order edge cannot be used in an edge-to-edge and a vertex-to-edge relation.

Supplementary support function groups all edges adjacent to v into the same higher order edge.

These rules are formally defined in [12]. OGs preserve paths in the same way as (3) shows for hierarchical graphs.

3.3 Cross-Layer Cuts

In the approach presented in this chapter, the analysis of the system is performed on a flat model, not the entire hierarchy. The actual benefit of using hierarchies in this case is in the possibility to obtain a flat model (or models) by cutting the hierarchy not horizontally but across multiple layers. The level of details is selected per element of the system, which gives high control on adjusting the precision of the obtained models, ultimately leading to the best sized models for the given fidelity requirement.

An *elementary transformation* is the minimum set of changes that may happen between two graphs without violating the rule set of OGs. Thus, OGs have the following types of elementary transformations, shown in Fig. 5:

Inclusion: Vertices and edges of the lower order are mapped into a single vertex in the higher order. Figure 5a shows vertices a_1, a_2, a_3, and edge e_1 being mapped into vertex a; relation $\langle e_1, a \rangle$ is implied and not drawn. This elementary transformation also appears in conventional hierarchical graphs.

Edge grouping: Edges of the lower order are mapped into a single edge in the higher order. Figure 5b shows edges e_1, e_2 being mapped into edge e. The relations are drawn as thin black lines to be differentiated from vertex-to-vertex relations. This elementary transformation also appears in conventional hierarchical graphs.

Support: One vertex is mapped into one edge in the higher order. Figure 5c shows vertex c being mapped into edge e; relations $\langle e_1, e \rangle, \langle e_2, e \rangle$ are implied and not drawn. This elementary transformation is unique to OGs.

Any transformation $\gamma = G^{(k)} \vdash G^{(k+1)}$ in OG can be represented with a sequence of elementary transformations $\gamma = \gamma_1 \circ \ldots \circ \gamma_n$, or:

$$G^{(k)} \vdash G^{(x_1)} \vdash \ldots \vdash G^{(x_n)} \vdash G^{(k+1)}. \tag{4}$$

For two consecutive orders $G^{(k)}, G^{(k+1)}$ of an OG, a *cross-layer cut* $G^{(x)}$ between order k and order $(k+1)$ is a graph, such that $G^{(k)} \vdash G^{(x)} \vdash G^{(k+1)}$ under the same rule set, and $G^{(x)}$ is partially equal to $G^{(k)}$ and $G^{(k+1)}$.

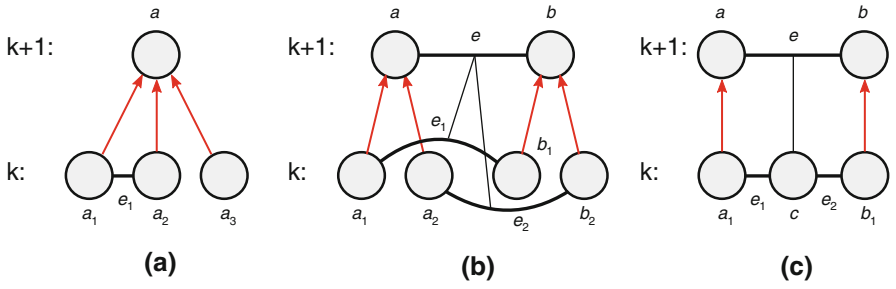

Fig. 5 Elementary transformations in Order Graphs and their notation: (**a**) inclusion, (**b**) edge grouping, (**c**) support

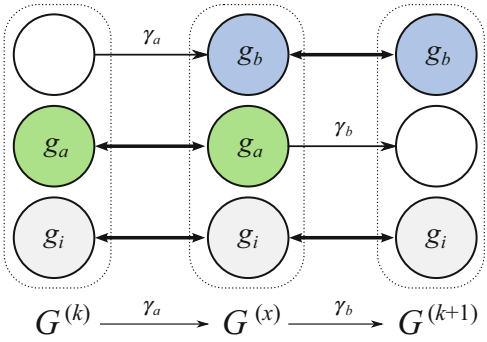

Fig. 6 Cross-layer cut $G^{(x)}$ explained

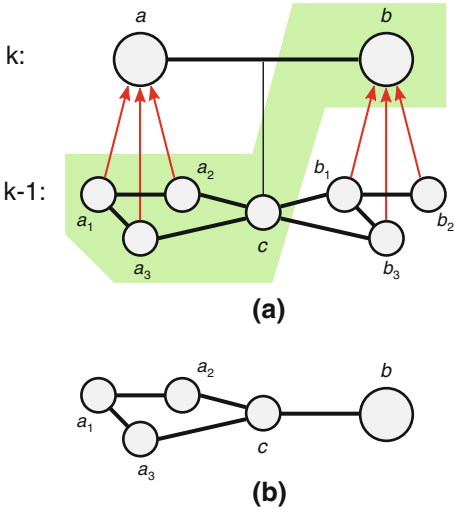

Fig. 7 Cross-layer cut example from Fig. 4b showing (**a**) the cut and (**b**) resulting flat graph

Figure 6 explains the above definition. Let $\gamma_a = G^{(k)} \vdash G^{(x)}$ and $\gamma_b = G^{(x)} \vdash G^{(k+1)}$. From $\gamma = G^{(k)} \vdash G^{(k+1)}$ and $G^{(k)} \vdash G^{(x)} \vdash G^{(k+1)}$, it follows that $\gamma = \gamma_a \circ \gamma_b$. Then, $G^{(x)}$ can be split in three parts: $G^{(x)} = g_a \cup g_b \cup g_i$, where g_i is the part that is not changed by γ, so $g_i \subseteq G^{(k)}$, $g_i \subseteq G^{(k+1)}$; $g_a \subseteq G^{(k)}$ is the part not changed by γ_a, and $g_b \subseteq G^{(k+1)}$ is the part not changed by γ_b. Thus, $G^{(x)}$ contains parts equal to subgraphs of $G^{(k)}$ (namely, g_a and g_i) and subgraphs of $G^{(k+1)}$ (g_b and g_i).

An example of a cross-layer cut can be found in Fig. 7.

Making a cut through more than two layers—from $G^{(k)}$ to some $G^{(k+b)}$—can be done iteratively. Firstly, obtain a cut between $G^{(k)}, G^{(k+1)}$, so $G^{(k)} \vdash G^{(x_1)} \vdash G^{(k+1)}$. Then, obtain a cut $G^{(x_2)}$ between newly created $G^{(x_1)}$ and $G^{(k+2)}$, which may now contain parts from $G^{(k)}, G^{(k+1)}$, and $G^{(k+2)}$. Repeat the process until the final cut $G^{(x_{b-1})} \vdash G^{(x_b)} \vdash G^{(k+b)}$ is found.

Cross-layer cuts are models of the same system and are consistent with the layers in the corresponding OG and preserve the connectivity property. The choice, which cut is the most appropriate, depends on the application. Section 4 presents the use case of parametric-proportional approach to optimise the model size for modelling system power.

4 Selective Abstraction

Moving up in the abstraction hierarchy, thus grouping multiple nodes into one, represents grouping the corresponding elements in SANs into a single entity by averaging/totalling their parameters (known as *black-boxing*). This reduces the size of a model for the price of added inaccuracy.

In this work we consider the simple system S that can be represented with a hierarchy consisting of two boxable parts a and b as shown in Fig. 8. The most detailed model of the system is k-th order of the graph. The result of this model's analysis is the estimated power (parameter) value

$$p = p_a + p_b, \tag{5}$$

where p_a is the combined power output from the part a, and p_b is the combined power output of b. It is reasonable to assume that some parameter characterisation is done prior to modelling, so the total baseline power of the system p^0 is known, as well as the powers of system parts p_a^0 and p_b^0. From this we can find the baseline error of the detail model:

$$E = \frac{p - p^0}{p^0}, \tag{6}$$

and the local errors of its parts:

$$E_a = \frac{p_a - p_a^0}{p_a^0}, E_b = \frac{p_b - p_b^0}{p_b^0}. \tag{7}$$

There are three possible models that can be obtained using black-boxing in this hierarchy: (a), (b), and (a, b). The latter is the fuzzy model of order $(k + 1)$, the others are cross-layer models. The goal is to choose the model that introduces the least possible error. Let's focus on cross-layer models for now, in particular, on doing one black-boxing at a time.

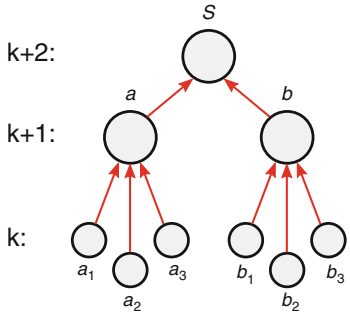

Fig. 8 Hierarchical system structure of the example

Let p' is the power estimate computed from the cross-layer model:

$$p' = p'_a + p'_b. \tag{8}$$

The model errors E', E'_a, E'_b are calculated similarly to (6) and (7). The added error in the model is

$$\Delta E = E' - E = \frac{p' - p^0}{p^o} - \frac{p - p^0}{p^o} = \frac{p' - p}{p^0}, \tag{9}$$

and similarly:

$$\Delta E_a = \frac{p'_a - p_a}{p^0_a}, \Delta E_b = \frac{p'_b - p_b}{p^0_b}. \tag{10}$$

We can substitute (5), (8) in (9) to find that:

$$\Delta E = \frac{p'_a - p_a}{p^0} + \frac{p'_b - p_b}{p^0}. \tag{11}$$

When one of the parts is black-boxed then the rest remains unchanged since we are focused on cross-layer models. We assume that, *in a well-designed system model, black-boxing of one part does not introduce errors in the other part of the system*, in other words, there is *no error contamination*. (In practice, this assumption puts restriction on the number of systems we can model. However, when applied to "designing for modelling", this restriction is reasonable.) As a consequence, the local parameter estimates outside of black-box are not changed. Thus, for example, if we black-box part a, then $p'_b = p_b$.

Let $x \in \{a, b\}$ is the part we chose to black-box, and $\Delta E^{(x)}$ is the corresponding added error, which from (11) is

$$\Delta E^{(x)} = \frac{p'_x - p_x}{p^0}. \tag{12}$$

We know that from (10):

$$\Delta E_x = \frac{p'_x - p_x}{p^0_x}, \tag{13}$$

hence we have a proportion, from which we find the added error in the system:

$$\Delta E^{(x)} = \Delta E_x \cdot \frac{p^0_x}{p^0} = \Delta E_x \cdot q^0_x, \tag{14}$$

where the ratio $q_x^0 = \frac{p_x^0}{p^0}$ is the portion of the local power dissipation in the total system power. Here, p_x^0 and p^0 are known from the initial characterisation; however, ΔE_x is not possible to find before computing the cross-layer model. And since the goal of the method is to choose the model without having to process all of them, we need to find an appropriate replacement for this variable. Fortunately, one can notice that, when q_x^0 is small, the Eq. (14) becomes insensitive to ΔE_x, so we can use an approximate estimate Δe – a *local potentially added error*, which is considered to be constant for the underlying modelling method throughout the system.

Ideally, the goal of selective abstraction is to obtain a cut that provides the minimal model while its added error satisfies the given threshold ε: $|\Delta E| < \varepsilon$. Our proposed power-proportional metric of selecting the cut is

$$|\Delta E| = |\Delta e_x q_x| \,, \tag{15}$$

where Δe_x is the local change of the percentage error, as a result of the black-boxing, in the part being black-boxed, and $q_x = \frac{p_x}{p}$ is the proportion of power consumed by this part in relation to the total power consumption. The values of p_x and p can be found from the model characterisation experiments, but Δe_x is typically not known before solving the model. Thus, instead of using the precise metric, one may rely on heuristic approximations. A number of methods are suggested in [13]. In this work, we use constant Δe_x under the assumption of only black-boxing similar component sub-models. Cross-layer cuts for deeper hierarchies can be found iteratively in polynomial time.

Equation (15) assumes that black-boxing one part of the system does not effect the accuracy of the other parts in the model. However, this is not the case if the behaviour of a detail part is dependent on the behaviour of an abstract part. It is important to remember that Δe_x is a percentage error, so the total deviation from the real value is amplified when the error leaks from a part with smaller q_x to a part with larger q_x. This concept of *error contamination* has been discovered in our work with selective abstraction.

The proposed method of detecting and localising contaminating errors is done by deriving a dependency graph from the dynamic model in relation to OG. The errors from the black-boxed parts propagate along the paths in this graph: the error of a node is maximum of its own error and errors of its preset nodes. The structure of the dependency graph puts restrictions on what resources can be used in selective abstraction (i.e. are *boxable*). Section 7 gives concrete examples of the models and obtained dependency graphs with and without error contamination.

It is also important to note that the method benefits from heterogeneity in the system: a bigger difference in the power consumption of the system parts provides better error tolerance in a cross-layer cut.

5 Background on Stochastic Modelling

SANs is an extension to General Stochastic Petri Nets (GSPNs) which is based on Petri Nets (PNs) [14]. It inherits the general attributes of PNs including a distributed representation of system states, making it easy to represent parts of a system directly as local subsystems, and more straightforward representations of such important issues as concurrency and synchronisation. A well-established method, it is supported by the mature software tool-kit: Möbius [18].

SANs are capable of representing both deterministic and stochastic events, and event durations in time. The elements used in this work include (a) transitions whose firing speeds (rates) are specified as stochastic, following given distributions, (b) transitions with multiple firing cases with specific probabilities for each case, and (c) input and output gates with predicates and implications specified through logic functions.

The Möbius tool, used in this work, incorporates a set of solvers including both Monte-Carlo simulation and state-space related solvers. Numerical Markovian solutions can be done for steady-state or time averaged interval rewards, but limited to models with exponentially distributed firing rates. The tool's concept of "rewards" can be easily extended to physical parameters, such as power. In our examples, we compute time-interval average power and use average error as an accuracy metric. The method is not limited to this and can give probabilistic estimation for transient power values.

6 Case Study

In this work, we aim to evaluate the impact of selective abstraction on the total error in the model. In order to do this, we want to build three models of the same system: a detailed model without any black-boxing, a cross-layer model with selective black-boxing, and an abstract model with maximum black-boxing. The result of analysing the power consumption using each of these models has to be compared with actual measurements from the platform.

6.1 Platform Description

One of the best off-the-shelf examples of a heterogeneous system for power analysis is the Odroid XU3 board [11]. The main component of Odroid XU3 is the 28 nm 8-core Application Processor Exynos 5422. This System-on-Chip is based on the ARM big.LITTLE architecture [7] and consists of a high performance Cortex-A15 quad-core processor block, a low-power Cortex-A7 quad-core block, a Mali-T628 GPU, and 2 GB LPDDR3 DRAM. The board contains four real-time current

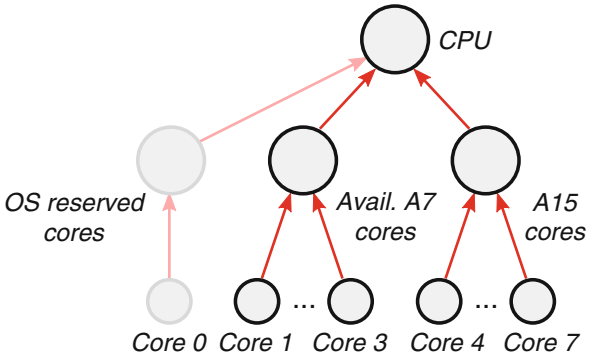

Fig. 9 Hierarchical structure of Odroid XU3 CPU cores. In the experiments, Core 0 is reserved for OS and tools

sensors allowing the measurement of power consumption on the four separate power domains: A7, A15, GPU, and DRAM.

For each domain, the supply voltage and clock frequency can be tuned through a number of preset pairs of values. The performance-oriented A15 quad core block can scale its frequencies from 200 to 2000 MHz, whilst the low-power A7 block has a frequency range from 200 to 1400 MHZ. Core 0 in the A7 domain has an additional speciality of running the OS kernel and drivers, and it cannot be switched off. We avoid using this core for stress tests and benchmarks to reduce the impact from the OS on the measurements. The CPU structure is represented as an OG hierarchy in Fig. 9.

6.2 Power Modelling

The average system power consumption can be found analytically as the function of the system workload and the system's power characteristics [20]. This work uses a simplified workload-based power model, which has been shown to provide sufficient accuracy [19]. The method of selective abstraction can be applied to advanced power models as well.

In our model, the power is a function of the type of executed task T, core type C, frequency F, voltage V, and the number of cores (of this type) running n. In the experiments, the frequencies and voltages of the cores remain constant per power domain, hence the values of F and V are tied to C and do not need to be considered separately. The system workload is modelled on a per-task basis as the ratio of the task's CPU time $t\,(T)$ to the total duration of the experiment t_{exp}.

Additionally, the cores are never put to sleep. Because of this, there is a constant background power P_{idle} consumed regardless of the workload, called *idle power*, which depends only on the core type C (considering constant F and V). The power spent to do actual computation P_{act} is called *active power*. The total power of a power domain C is found as a time-averaged active power added to the constant idle power:

$$P_{\text{total}}(C) = P_{\text{idle}}(C) + \sum_{T} \frac{t(T)}{t_{\text{exp}}} P_{\text{act}}(n, T, C). \tag{16}$$

The values for P_{act} and P_{idle} can be characterised offline in a form of a table function. However, the exact value of $t(T)$ is known only during run-time. In our work, we use stochastic modelling to predict this value. The parameters for workload prediction models include the spawn rates for the tasks and the average CPU time required to complete a task (completion rates). It is reasonable to assume these are known to the model designer.

The presented simplified power model does not take into account system temperature. In fact, it is suspected to be the main source of error in our experimental results, as the models were characterised on fully loaded cores, much hotter than during the actual experiments. This baseline error contributes to all layers of abstraction, including the detail model. The focus of the presented research is to investigate the additional error due to black-boxing.

6.3 Experiment Setup

Figure 10 shows the model evaluation framework. The fixed rates are used to generate random execution traces – list of task spawning events, affinities, and completion events. These traces are executed on the platform to produce power traces – sets of timestamped power measurements. At the same time, the rates and platform characteristics are used to parametrise the SAN models, which are analysed in the Möbius tool. The models do not know the actual execution traces. The mean power from the model analysis is then compared to the mean power obtained from

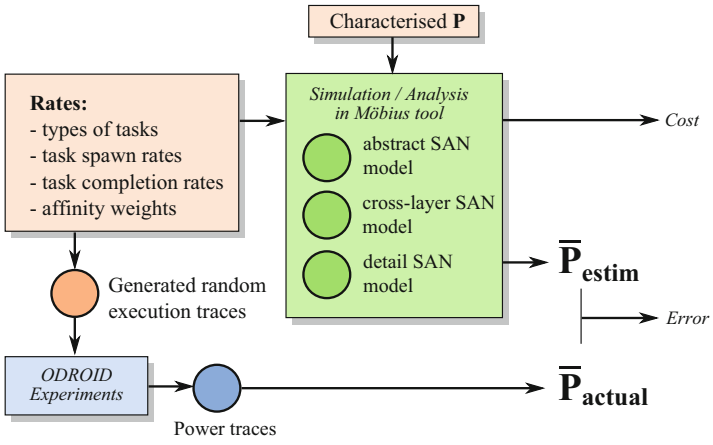

Fig. 10 Model evaluation framework

the power traces to estimate the modelling error. The computational cost (time) of analysing the models is also evaluated.

Execution traces reflect the following scenarios of the system operation. During the experiment, two types of tasks (T0 and T1) spawn continuously. The spawn intervals, in ms, are exponentially distributed with rates, respectively, $\lambda_{\text{spawn,T0}} = 0.003$ and $\lambda_{\text{spawn,T1}} = 0.002$. Both tasks are CPU-heavy: T0 performs a floating point square root computation in a loop; T1 performs integer multiplication and addition. The tasks are then scheduled on the available cores. In order to increase the heterogeneity of the system, we assigned different probabilities to scheduling onto different cores (affinity weights) providing three possible scenarios: Scenario EQ represents equal scheduling chances for all cores regardless of their type, Scenario CA differentiates between cores, while Scenario TCA has weights depending on the type of a task as well as the core.

Each tasks is executed on a core until it is finished and then removed. Completion times are random and exponentially distributed[1] with constant rates (the frequencies of the cores are kept unchanged during the experiments). In real-life situations, low-power A7 cores typically operate on lower frequencies than A15, so their performance is reduced. To mimic this behaviour in the experiments, A15 is set to work approximately twice as fast as A7, and the completion rates for A7 domain in the model are halved. The rate values are shown in Table 1 as "target completion rates". During the model characterisation, these values are re-adjusted for more precision, as discussed in Sect. 6.4.

Table 1 Power and time characteristics

Domain	A7		A15	
Number of cores N	3		4	
Idle power, W	0.0737		0.6211	
Task	T0	T1	T0	T1
1 core active power, W	0.1392	0.1427	1.4684	1.4281
N cores active power, W	0.2703	0.2771	4.0101	3.9066
Exec time, ms	8745	9730	9827	8184
Target completion rate	0.001	0.0015	0.002	0.003
Adjusted comp. rate	0.00114	0.00154	0.00204	0.00367

[1]We had to limit our examples to exponential distribution in order to test Markovian solvers. In simulation-only studies, any other random distribution can be programmed in.

6.4 Characterisation Experiments

Platform characteristics required to parametrise the models include per-core power consumption for each core type when idle and when running each type of a task. We set A7 cores to run at 1000 MHz, and A15 cores are set to run at 1800 MHz. The core frequencies are set below the highest mark to avoid throttling. Since the power can be measured only per domain, which consists of 4 cores, it requires additional experiments and calculations to be performed.

Idle power is measured directly for each power domain. Core 0 is reserved for running the OS, the scheduler, and the power trace logger, and is not used directly for running the experiments. Its impact on power consumption is viewed as a background noise included in the idle power of the A7 domain. To measure the active power we ran each task separately on each domain, i.e. providing characterisation data for each $\langle T, C \rangle$ pair. Also, since we didn't know if the power consumption scales linearly by adding more cores, we ran each set on $1 \leq n \leq N$ active cores. The active power of a single core is then related to the measured power as $P_{\text{act}}(1) = (P_{\text{meas}} - P_{\text{idle}})/n + P_{\text{idle}}$. Similarly, the power of running all cores in the domain is $P_{\text{act}}(N) = (P_{\text{meas}} - P_{\text{idle}}) \cdot N/n + P_{\text{idle}}$. All instances of measured or computed values are within 3% range from the respective mean values across all experiments with the exception of a single A7 core executions, which deviate by 5%. This is within the acceptable error range, so we can still assume linear power scaling: $P_{\text{act}}(n) = n \cdot P_{\text{act}}(1)$. The final values used in the model are shown in Table 1.

Since rates and characteristics are the only things connecting the models with the actual experiment, we take extra care when generating traces and executing them on the platform. A specialised scheduler has been designed to address this issue.

Each entry of the execution trace contains the timestamp of spawning a task, the task type, its affinity, and input data (a single integer value T), which affects the task completion time. To guarantee that the execution times follow the exponential distributions with the given rates and to simplify the trace generation, T is equal to the requested execution time in ms. The tasks T0 and T1 henceforth are required to match their execution time to T as close as possible. It is possible to achieve by reading the system timer, but calling kernel functions may cause unwanted interference. Instead, we achieve this by doing a task in a loop and calibrate the number of iterations to complete in the given time. The task calibration function for some core i and task j is $f(i, j, T) = x_i \cdot y_j \cdot T$; the constants x_i and y_j are found experimentally: $x_{A7} = 21, x_{A15} = 40$ (confirming that A15 running at 1800 MHz is roughly twice as fast as A7 at 1000 MHz), $y_{T0} = 95, y_{T1} = 1700$. Thus, for example, in order to run T1 on A7 for 100 ms, we need to do 3,570,000 loop iterations. However, this calibration is not perfect and requires further adjustment.

During the characterisation experiment, we request the tasks to run for 10s by specifying $T = 10,000$ and then measure the actual completion time. Considering that the target completion rates are used for generating the traces, but the actual times are skewed, as shown in Table 1, we apply a simple proportion to calculate the adjusted completion rates to be used in the model.

7 Models and Results

Following Sect. 2.1, by using black-boxing some x cores can be grouped into more abstract meta-cores combining the performance, power consumption, and scheduling probabilities of constituent cores. We sacrifice the accuracy by considering the task completion in the meta-core to be exponentially distributed with the rate $\lambda^{x}_{\text{exec}}$, while it is in fact the sum of exponential distributions.

The model template for scaling is a parametrised SAN where the elements are replicated to a given target number of system resources. In our case, the templates scale to n cores and m tasks. Figures 11 and 12 show example models scaled to 2 tasks and 2 (meta-)cores. Hence, some model elements are added per task (prefixed with T_i, $0 \leq i < m$), some appear per core (prefixed with C_j, $0 \leq j < n$); the others are instanced $n \times m$ times: per-core and per-task (interfaces). Different types are shown in different colours. Colour is not a part of an actual model and is used solely for visual aid.

All the scenarios in Sect. 6 use two types of tasks ($m = 2$), and the scaling is done only on the number of cores. The models representing different levels of abstraction are

3+4 model ($n = 7$) is the most detailed model considering each core separately.

1+4 model ($n = 5$) is the model obtained using the proposed method of selective abstraction. Here, three A7 cores are grouped into a single meta-core representing the entire domain.

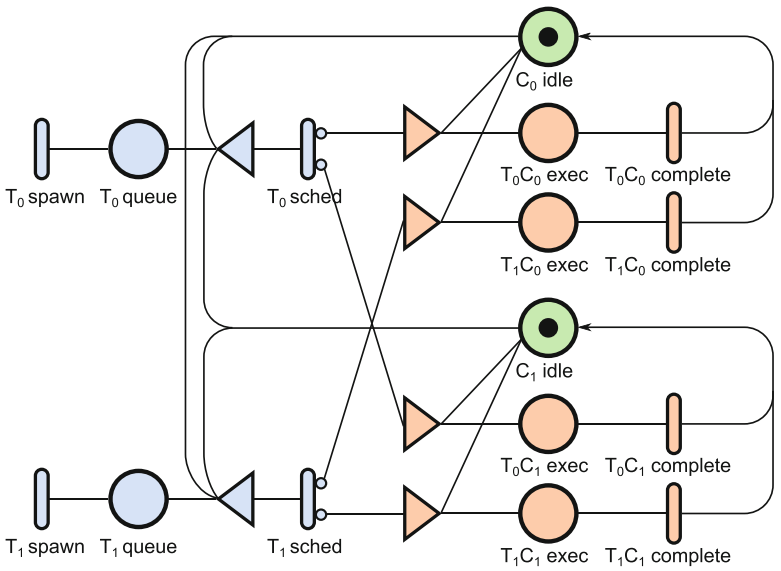

Fig. 11 Model for the naïve scenario leading to error contamination

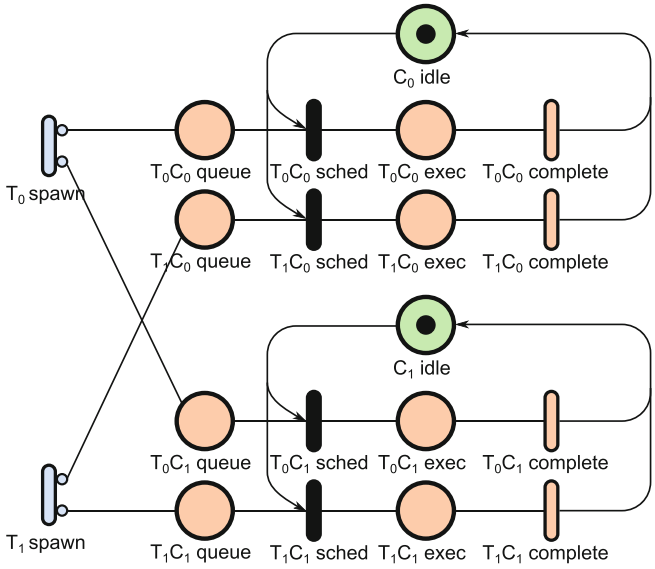

Fig. 12 Fixed affinities in the scenario produce a better model

1+1 model ($n = 2$) is the most abstract model with two meta-cores, one representing the A7 domain, the other representing A15.

Table 1 gives $q_{A7} = 0.065$, and from (15), under the assumption of constant Δe_x, we have $\Delta E = 0.065 \cdot \Delta e_x$ for the (1+4) model. This assumption can be justified by simulation results if

$$\left|E_{(1+4)} - E_{(3+4)}\right| \approx q_{A7} \cdot \left|E_{(1+1)} - E_{(3+4)}\right|, \tag{17}$$

where $E_{(1+1)}$, $E_{(1+4)}$, and $E_{(3+4)}$ are the total percentage errors in the respective models.

The initial experiment setup was not specific on how exactly task affinities are realised in the scheduler, so we implemented two variants of the system and produced two models, respectively. Figure 11 shows the model of a system with task-exclusive queues, where the scheduling weights are applied after the queueing, which means that the scheduler needs to check every core for availability. Figure 12 models tasks with fixed affinities, i.e. the task is paired with a core once it is spawned and then waits in the queue for this core to become available (idle).

Unfortunately, the first implementation proved to be of a poor quality because of error contamination due to a high inter-dependence of its elements. Figure 14a shows its dependency graph, obtained by the mapping rules from Fig. 13. Let's assume C_0 elements produce extra $|\Delta e|$ due to black-boxing, and C_1 must be protected from error contamination. From the graph, it is evident that there is a path connecting C_o to every other element in the model propagating the error. Figure 14b,

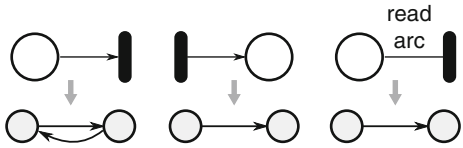

Fig. 13 Rules for SAN mapping to dependency graphs

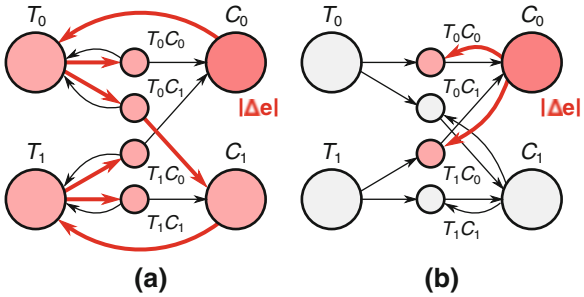

Fig. 14 Dependency graphs with highlighted *error contamination* paths: (**a**) for Fig. 11, (**b**) for Fig. 12

showing the second implementation, is able to contain the error locally: it pollutes only the adjacent nodes, but keeps C_1 unaffected.

We were unable to find a way to resolve the contamination by model transformation without changing the system's behaviour, as the problem has roots in the functional properties of a modelled system. This means that the algorithm modelled by Fig. 11 has no viable selective abstraction and system redesign is needed (Fig. 2). The main contribution of this work is to detect and identify error contamination in models by the structural analysis, as described above, before the model is simulated. The rest of the chapter uses the template shown in Fig. 12.

7.1 State-Space Analysis

Our investigation showed that state-space analysis has a number of limitations compared to simulations. The presented models have unbounded state spaces, and, unless we put a hard constraint on the total number of tasks in the system, the state-space analysis is not possible. From the task spawn rates we know that on average the number of tasks spawned during a 15 s experiment is 75. The lack of scalability in state-space analysis makes even this number infeasibly large. The highest number that doesn't fail to compute is 50 for the $(1 + 1)$ model, taking 5729 s of computation time on a 2-core Intel i7 5500U machine. For more detailed models, the feasible number of tasks goes down to 12 for $(1 + 4)$ and 9 for $(3 + 4)$, which is not enough for trustworthy results.

Consequently, with the added restriction to exponentially distributed rates, the state-space analysis methods appear rather impractical for the presented application. We recommend using simulations instead, reserving state-space analysis only for model verification purposes.

Table 2 Simulation results compared to measured values

Scen	Model	Pwr, W	Pwr var	Error (%)	Sim, s
EQ	1 + 1	1.7662	0.0470	6.53	2.535
	1 + 4	1.7287	0.0424	4.27	2.546
	3 + 4	1.7215	0.0423	3.84	2.764
	meas.	1.6579	0.0572		
CA	1 + 1	2.0205	0.0619	7.99	2.545
	1 + 4	1.9470	0.0468	4.06	2.610
	3 + 4	1.9421	0.0468	3.79	2.764
	Meas.	1.8711	0.0385		
TCA	1 + 1	2.0038	0.0608	10.41	2.547
	1 + 4	1.9274	0.0439	6.21	2.613
	3 + 4	1.9245	0.0440	6.05	2.771
	Meas.	1.8148	0.0279		

7.2 Simulation Results

Table 2 presents the power values obtained from simulations in Möbius tool and compares them against the actual power measurements. The simulation time is given for 1500 simulation batches, which are required for calculating 0.01 relative interval with 95% confidence. The variances have been calculated separately over 20,000 simulation batches. Experimental results are averaged over 50 random trace runs for each scenario, and the variance is also calculated.

The achieved 4–6% accuracy meets the typical requirement for system-wide power modelling and shows a good potential in using stochastic methods. The results justify (17), thus confirming the expectations of selective abstraction metric: the error added by moving from (3 + 4) to (1 + 4) model in comparison to going from (3 + 4) straight to (1 + 1) is proportional to the power output of A7 domain in relation to the total power. The difference between simulation times, however, is not large and appears to grow linearly with the model size in the presented sample.

Figure 15 plots the simulation results in a Quality/Usability trade-off space with the inverse of error e^{-1} representing Quality, and the inverse of simulation time t^{-1} being the metric for Usability. The results demonstrate that the method allows trading little accuracy for the steady increase in usability, and demonstrates the scalability of SAN models for simulation studies.

8 Conclusions

This work aims to produce a method towards scalable power models for multi-core heterogeneous systems. We concentrate on rationalising model sizes based on power-proportional representation and stochastic modelling. A systematic approach to selective abstraction using OGs is developed. The method includes ways of

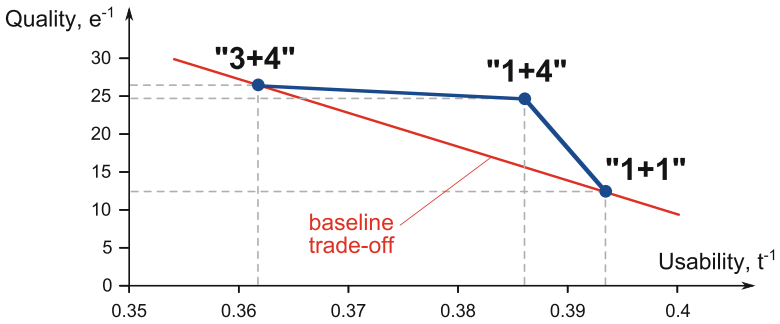

Fig. 15 Simulation results in the Quality/Usability trade-off diagram

identifying error contamination and determining boxability. Stochastic techniques are investigated with SAN and Möbius. Selective abstraction is shown to be effective for model size and designer effort reduction, and SAN models are demonstrated to have excellent scalability for simulations. In these ways our method supports the systematic discovery of good trade-offs between modelling quality and model scalability.

In addition to further improvements to the SAN model of the platform by adding memory and cache, the future work may include the application of cross-layer cuts to other modelling methods.

Acknowledgements The authors want to thank Rishad Shafik and Mohammed Al-Hayanni for useful discussions. This work is supported by EPSRC as a part of PRiME project EP/K034448/1. A. Aalsaud is supported by a postgraduate studentship from the Education Ministry of Iraq.

References

1. Accellera Systems Initiative. (2011). IEEE 1666 standard: SystemC language reference manual. http://www.accellera.org/
2. Bartolini, A., Cacciari, M., Tilli, A., & Benini, L. (2013). Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Transactions on Parallel and Distributed Systems, 24*(1), 170–183.
3. Caldari, M., Conti, M., Crippa, P., Nuzzo, G., Orcioni, S., & Turchetti, C. (2002). Instruction based power consumption estimation methodology. In *International conference on electronics, circuits and systems '02*, (Vol. 2, pp. 721–724).
4. Chen, M., Yue, D., Qin, X., Fu, X., & Mishra, P. (2015). Variation-aware evaluation of MPSoC task allocation and scheduling strategies using statistical model checking. In *Proceedings to DATE '15* (pp. 199–204).
5. Chen, X., Zhang, G., Wang, H., Wu, R., Wu, P., & Zhang, L. (2015). MRP: Mix real cores and pseudo cores for FPGA-based chip-multiprocessor simulation. In *Proceedings to DATE '15* (pp. 211–216).
6. Ehrenfeucht, A., & Rozenberg, G. (2014). Zoom structures and reaction systems yield exploration systems. *International Journal of Foundations of Computer Science, 25*(3), 275–306.

7. Greenhalgh, P. (2011). *big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7—Improving Energy Efficiency in High-Performance Mobile Platforms*. ARM, White Paper.
8. Kaiser, S., Materic, I., & Saade, R. (2010). ESL solutions for low power design. In *Proceedings of the ICCAD* (pp. 340–343).
9. Kumar, B., & Davidson, E. S. (1980). Computer system design using a hierarchical approach to performance evaluation. *Communications of ACM, 23*(9), 511–521.
10. Narayanan, V., Lin, I.-C., & Dhanwada, N. (2005). A power estimation methodology for SystemC transaction level models. In *Proceedings to CODES+ISSS '05* (pp. 142–147).
11. Odroid XU3. http://www.hardkernel.com/main/products
12. Rafiev, A., Xia, F., Iliasov, A., Gensh, R., Aalsaud, A., Romanovsky, A., et al. (2015). Order graphs and cross-layer parametric significance-driven modelling. In *Proceedings to ACSD*.
13. Rafiev, A., Xia, F., Romanovsky, A., & Yakovlev, A. (2016). *Error-based metric for cross-layer cut determination*. Technical Report NCL-EEE-MICRO-TR-2016-201, School of EEE, Newcastle University.
14. Sanders, W., & Meyer, J. (2001). Introduction to generalized stochastic Petri nets. In *Lectures on formal methods and performance analysis. Lecture notes in computer science* (Vol. 2090, pp. 315–343). Berlin: Springer.
15. Sassolas, T., Sandionigi, C., Guerre, A., Mottin, J., Vivet, P., Boussetta, H., et al. (2015). A simulation framework for rapid prototyping and evaluation of thermal mitigation techniques in many-core architectures. In *Proceedings in ISPLED '15*.
16. Synopsys. (2015) Primetime PX. https://www.synopsys.com/apps/support/training/primetime_fcd.html
17. The gem5 simulator system. http://www.m5sim.org
18. The Möbius modelling tool. https://www.mobius.illinois.edu
19. Walker, M. J., Das, A. K., Merrett, G. V., & Hashimi, B. (2015). Run-time power estimation for mobile and embedded asymmetric multi-core CPUs. In *HIPEAC workshop on energy efficiency with heterogeneous computing*. HiPEAC.
20. Yang, S., Shafik, R. A., Merrett, G. V., Stott, E., Levine, J. M., Davis, J., et al. (2015). Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *Proceedings to PATMOS '15*.
21. Zurcher, F. W., & Randell, B. (1968). Iterative multi-level modeling—A methodology for computer system design. In *Proceedings of IFIP Congress 68* (pp. 138–142).

Feature-Based State Space Coverage Metric for Analog Circuit Verification

Andreas Fürtig, Sebastian Steinhorst, and Lars Hedrich

Abstract This chapter proposes a systematic and fast analog coverage-driven verification methodology which could increase the confidence in verification of today's analog blocks. We define an appropriate coverage metric to score simulations and then minimize the simulation effort for achieving full state space coverage with an algorithm generating appropriate input stimuli. Our proposed method uses characteristic properties of a discretized representation of the state space such as the spatial distribution of eigenvalues, guiding the generation of short and purposeful stimuli. The experimental results show a significant speed-up with similar accuracy compared to the state of the art.

Keywords Coverage • Analog coverage • State space coverage • State space discretization • Analog circuit verification • State space analysis

1 Introduction and Related Work

Traditionally, analog circuit design and verification needs sophisticated designers and verification engineers to prevent faulty behavior and expensive redesigns. Nowadays, the pressure on them due to the significant analog part on common chips (automotive, consumer) and short design cycles is further increasing. Unfortunately there are not many systematic approaches to tackle the functional verification problem for analog circuits—the standard procedure to prevent hard to find bugs is to use expert knowledge from experienced designers.

A. Fürtig (✉) • L. Hedrich
Institute for Computer Science, Goethe Universität Frankfurt am Main, Frankfurt, Germany
e-mail: fuertig@em.cs.uni-frankfurt.de; hedrich@em.cs.uni-frankfurt.de

S. Steinhorst
Department of Electrical and Computer Engineering, Institute for Advanced Study, Technical University of Munich, München, Germany
e-mail: sebastian.steinhorst@tum.de

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_5

One direction to systematically check analog circuits may be the full automatic characterization [1] based on formalized specifications using machine readable specifications [2] or formal languages such as PSL [3]. However, the effort to setup these specifications is sometimes large and, even worse, they do not guarantee to find unknown bugs because they rely on predefined input stimuli for each performance test case. With simulation only, there still exist uncovered scenarios which may later arise as a bug in the field.

Formal verification for analog circuits [4, 5] will certainly help as it can guarantee to find problematic design flaws violating the specification. However it suffers from long runtimes, hard to interpret results, and the perennial "translate specification into a formal language" problem.

A compromise could be the use of coverage metrics and coverage-increasing measures. The digital world has developed a lot of coverage metrics [6–8] and uses them with success. Depending on the complexity of the Device Under Verification (DUV), the methods are more or less complete. The complete methods investigate for example Finite State Machines (FSM) [9] and have some means to try to restrict the simulation input stimuli to the relevant part of the state space (see SFSM in [9]). The less complete methods (code coverage, specification coverage) use measures to guide the verification to the most probable bug location for example by systematically visiting each conditional branch in an HDL-description.

For analog circuits, a very low number of coverage investigating approaches besides the above explained formal verification techniques exist. There are some approaches stemming from the test community measuring and increasing the analog fault coverage [10, 11]. However, they are not intended to find functional faults. Horowitz et al. [12] also tries to increase the confidence in the functional verification using a high-level functional model but without a systematic method to increase some underlying measure. Two other approaches are built for hybrid systems [13, 14], suffering from being able to handle strongly nonlinear analog circuits on transistor level. Steinhorst et al. [15] and Karthik et al. [16] concentrate on the analog state space to systematically implement formal verification, hence being accurate and complete. However, they also have no well-defined measure for the coverage and suffer from the large state space to investigate. As a remedy, in this chapter, we later propose a coverage optimization algorithm that takes into account the dynamics of the state space. Consequently, the algorithm can identify regions of critical nonlinear behavior requiring a very dense coverage, as well as regions with highly linear behavior which is not critical for the verification coverage. The latter will enable us to drastically reduce the volume of the visited state space.

Contributions This chapter introduces a complete methodology for optimization of analog verification coverage by analyzing the dynamics of the state space of the Design under Verification (DUV), providing four main contributions outlined in the following.

- We present a state space coverage metric in Sect. 3 which creates a relation between transient simulation waveforms and states of a discrete representation of the DUV which we introduce in Sect. 2.

- Based on the coverage metric, we introduce a coverage optimization algorithm that maximizes the defined coverage metric.
- The state space coverage metric and algorithm are further developed into the proposed λ state space coverage metric to identify interesting regions and neglect uniform parts of the state space in Sect. 4.
- Our metrics and algorithms are evaluated on several analog transistor level circuits in Sect. 5 and clearly show the advantages over a state-of-the-art approach.

2 State Space Model Generation

The state space coverage analysis we are proposing in this chapter requires a discrete model of the analog circuit. We use a trajectory-based state space discretization proposed in [17]. This method is based on discretizing the underlying DAE-System of the circuit in the state space. The discretization is performed using an electrical circuit simulator with full SPICE accuracy [18].

With this method we can construct a discrete state space model M_{ATS}:

$$\text{Electrical Circuit} \overset{\text{discrete modeling}}{\longrightarrow} M_{\text{ATS}} \tag{1}$$

2.1 Analog Transition System (ATS)

For the ATS we define a five-tuple $M_{\text{ATS}} = (\Sigma, R, L_V, T, L_\lambda)$ where

- Σ is a finite set of states of the system.
- $R \subseteq \Sigma \times \Sigma$ is a total transition relation, hence for every state $\sigma \in \Sigma$ there exists a state σ' such that $(\sigma, \sigma') \in R$.
- $L_V : \Sigma \to \mathbb{R}^{n_d}$ is a labeling function that labels each state with the vector of n_d variables containing the values of the state space variables and the inputs of the DAE system.
- $T : R \to \mathbb{R}_0^+$ is a labeling function that labels each transition from σ to σ' with a real valued positive or zero transition time that represents the time required for the trajectory in the state space between these states.
- $L_\lambda : \Sigma \to \mathbb{R}^{n_\lambda}$ is a labeling function that labels each state with a vector of the n_λ eigenvalues associated with the state.

Within the structure M_{ATS}, a path π beginning at state σ is a sequence of states $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_n$ with $\sigma_0 = \sigma$ and $(\sigma_i, \sigma_{i+1}) \in R$ for $0 \leq i < n$.

In an extension to the method of [17] we calculate and store the eigenvalues of each state during the discretization process. For this purpose, the system's dynamics are linearized in the specific state and then transformed into the frequency domain using Laplace transformation. The number of nonzero entries in Kronecker's

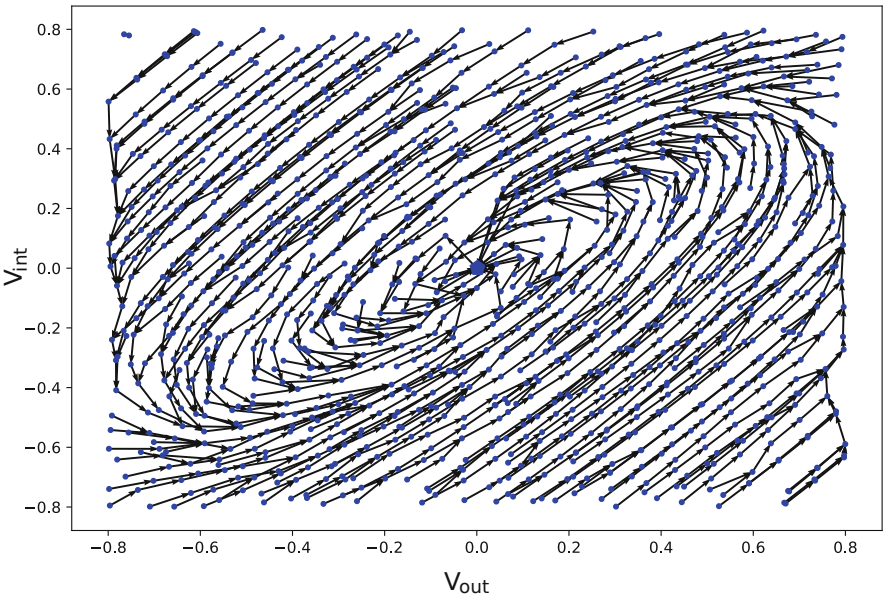

Fig. 1 Discretization of the state space of a bandpass filter. *Blue points* represent Σ with one DC operating point at $V_{\text{out}} = V_{\text{int}} = 0.0\,\text{V}$. *Black arrows* indicate transitions relation R of the system. The state space is plotted for an input voltage $V_{\text{in}} = 0.0\,\text{V}$

canonical form of the transformed capacitance matrix of the frequency domain representation corresponds to the number n_λ of eigenvalues in the generalized eigenvalue problem. For a detailed description of the eigenvalue decomposition, please refer to [19].

Figure 1 shows the discrete state space of a bandpass filter. For display purposes, the state space is plotted for a given input voltage. The complexity of the state space modeling process is exponential in the number of energy-storing elements inside, and inputs to a circuit. Relevant analog circuit blocks usually do not exceed a system order of 8, which can be handled well by this approach. Moreover, by application of an Eigenvalue-based model order reduction of the DAE system [20], circuits with more than 200 parasitic capacitances can be handled. This is achieved by reducing the state space to the dominant state variables of a system and discarding the parasitic ones which are mathematically proven not to affect the system behavior above a defined threshold.

The following section will describe how this discrete state space is used to compute an analog coverage for a given set of different circuits.

3 State Space Coverage

This section describes a method to match a transient simulation response to the previously described state space. Our goal is to automatically create an input stimulus for an analog simulator to maximize the presented analog state space coverage. A path finding algorithm is presented afterwards, as well as possible restrictions of this method.

The state space coverage ζ denotes the ratio between visited states and the sum of all reachable states Σ_R of a given circuit. The wanted coverage metric should assess a simulation response based on the following characteristics:

- A coverage value near 100% implies a high probability that all possible faults of the circuits could be detected.
- The measure has to be monotonic in the number of visited states: If more states are visited, the measure should increase.

The resulting number can be used to compare different input stimuli for an analog simulator leading the designer to much more useful test cases, reducing the possibility of missing possible design flaws.

The Analog Transition System M_{ATS} described in Sect. 2 creates a vast number of states which could possibly not be reachable at all. Using the number of states $|\Sigma|$ of the full system results in a metric not able to gain full coverage. Hence, a set of reachable states Σ_R is computed from all states Σ visited by the state space discretization using a simple set-based reachability algorithm. For our purpose, the number of reachable states is lower or equal to the number of all states.

3.1 State Space Coverage Calculation

A transient simulation response consists of a set of different data points, representing the state of an analog circuit at a given time step. To match each of these points to a set inside our M_{ATS}, we use an Euclidean distance to mark a state as covered by a simulation.

In a very first straightforward approach, one can compute a nearest neighbor for every data point. For that, we store the previously defined Analog Transition System M_{ATS} in a suitable space-partitioning data structure in the form of a k-d tree [21]. The number of nodes in this tree equals the number of states in the system. Hence, if a discretization only consists of very few states, each point of a simulation response will lead to a covered state, although the state is very far off. Obviously this simple approach does not calculate a smooth and adequate measure. Since every point has a nearest neighbor, the distance is not considered (cf. Fig. 2, upper left).

A much better approach is to select every state in a given distance around a data point of the transient simulation response. This allows to have a measure independent of the sampling distance in the state space as well as the sampling

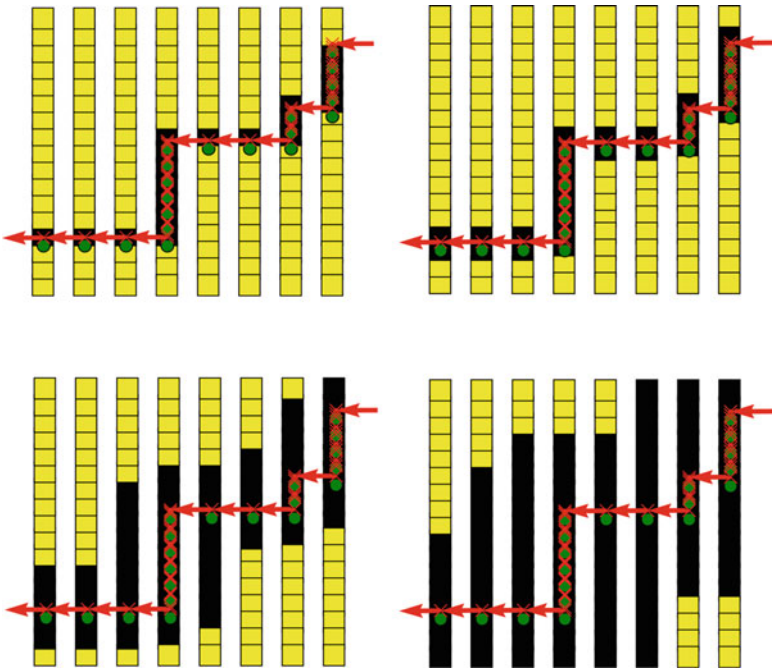

Fig. 2 Euclidian distance method for selecting the *covered* states of a simulation result. *Red crosses* indicate the trajectory corresponding to the transient simulation result. *Black boxes* are marked as covered, and *yellow boxes* are marked as uncovered

distance of the transient simulation result. Figure 2 shows different values for a distance to accept different states inside a M_{ATS} marked as covered. It can be seen that a maximum distance must be chosen adequately, since using a too large distance could mark states with different behavior compared to the transient trajectory under investigation, while a too small distance will underestimate the set of covered states C.

A good starting point for the distance is to select the *median* distance between two neighbor states in the discrete state space or to use a percentage of the diameter of the reachable state space. Here, we conservatively take the median length of all transitions R inside the M_{ATS}.

Consequently, the coverage of a given transient simulation response can now be computed using the cardinality of the elements in the set C and the number of states in the reachable discrete state space Σ_R:

$$\zeta = \frac{|C|}{|\Sigma_R|} \tag{2}$$

This gives us the possibility to rate a set of test by calculating a coverage for each test as long as a full coverage is reached. Full coverage in this context means every

reachable state inside a M_{ATS} was reached by simulation. Hence, no unexpected behavior can occur. To enhance the coverage ζ a designer could develop new tests as long as uncovered states exist. As we will see in Sect. 4 an alternative is to restrict the number of "to be reached" states from $|\Sigma_R|$ to $|\Sigma_\lambda|$.

3.2 Path Planning

As mentioned in the previous subsection, a full coverage of a discrete state space is a desirable goal. For automation purposes, a path planning method is introduced, as the creation of appropriate input stimuli is crucial for the usage of the coverage described before.

First of all, the M_{ATS} is enhanced by a labeling function $\omega_\sigma : \Sigma \rightarrow \mathbb{N}_0^+$ that labels each state with a weight, denoting the number of visits of this state by a simulation. Together with the relation R, this eases a path finding inside the discrete state space. Another helping aspect is an additional set Σ_{DC}, which holds a set of DC operation points of the M_{ATS}. As stated beforehand a path π through the discrete state space can be directly used to create an input stimulus for a simulation software, as the labeling function L_V also holds the inputs to the system. Timing information can be gathered from the transitions between two states in the M_{ATS}.

Using an A^* algorithm, it is easy to compute a path through the state space targeting an uncovered state. This method will lead to a vast number of very small simulations. As a result, the startup time of the simulation software will dominate the simulation time. To avoid this behavior, a path planning algorithm is needed to create simulation input stimuli which meet the following characteristics:

- The resulting path should avoid already visited states.
- It should consist of as many unvisited states in the M_{ATS} as possible.

An approach to satisfying these criteria exists in [15], but with larger circuit size a full input stimulus created using this method consists of significantly more data points than $|\Sigma_R|$ itself. More complex circuits lead to a very long runtime of the simulation, due to the increased state space dimensions and more state space points. As we will see in the results section, the constructed single stimulus by that method performs badly in terms of the achieved state space coverage.

To improve this method, we introduce a weight-based path planning: Let π be a set of states describing the path from a starting to a target state. The length $|\pi|$ is the number of states inside the path. Since every state has a weight ω_σ, the weight of a path is $\omega_\pi = \Sigma_i^n \omega_{\sigma_i}$. An unvisited, randomly chosen state σ_u (indicated by its weight $\omega_\sigma = 0$) is used to compute *all* possible paths from every operation point $\sigma_d \in \Sigma_{\mathrm{DC}}$. Additionally, we compute the *longest* path starting in state σ_u back to the operating points. This gives us the possibility to concatenate the resulting paths, producing a longer overall path. To avoid very long paths (like the ones created by the method in [15]), the length of the resulting path is limited by the number of unvisited states in the whole system.

3.3 Coverage Maximization Algorithm

With bigger analog circuits, the possibility to reach full coverage with one single input stimulus is very small. For that, we introduce an algorithm to cover all reachable states inside a M_{ATS}. It is very easy to see that one single stimulus created by the path finding algorithm described beforehand will not reach all states in the system.

The presented Coverage Maximization Algorithm is shown in Fig. 3. In every step, the algorithm selects an unvisited state and calculates a path targeting that state. Selecting the longest path with minimum cost maximizes the possibility to cover at most unvisited points at once. While traversing the graph, we are able to create an input stimulus for a path. Each data point of the resulting transient response is mapped to a state inside the M_{ATS}, increasing the weight of that state as well as the overall coverage of the whole system. The algorithm stops if every state in the state space is covered by a simulation.

With increasing complexity of the investigated circuits, it is furthermore not possible to reach high coverage measures ζ in a reasonable computation time and with short overall input stimuli length. Hence, the number of states to inspect must be reduced, which will be described in the next section.

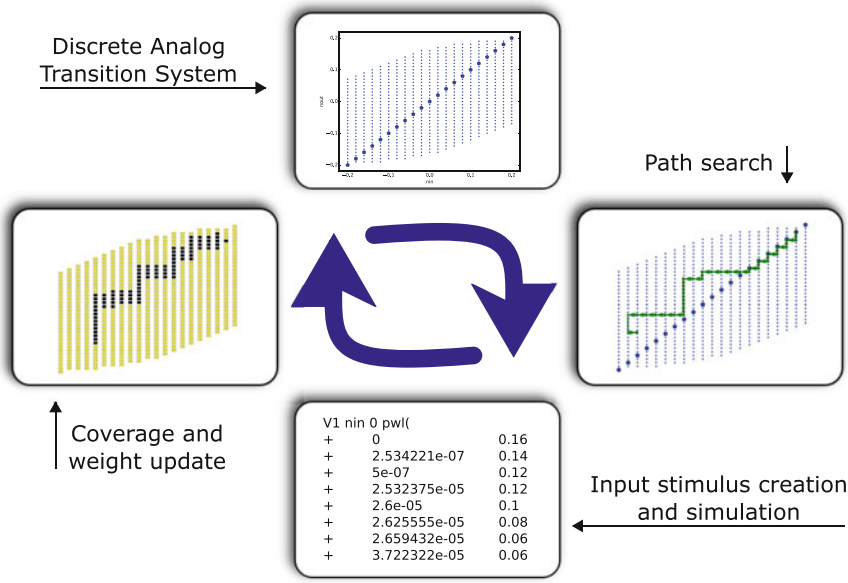

Fig. 3 Coverage maximization algorithm based on discrete state space modeling

4 λ State Space Coverage

As we will see later on in the results section, trying to reach a full coverage is a very time consuming procedure even for small devices like a Schmitt trigger or lowpass filter circuit. Visiting every single reachable state in an analog circuit often makes no sense, since many regions of the state space have a homogeneous behavior—in most cases linear behavior—and can be investigated by one trajectory through these regions. To reduce the number of states to cover without missing regions with a heterogeneous behavior and important states, we are segmenting the discrete state space into different regions. Namely, these are regions with uniform (linear) behavior, nonlinear parts with high dynamic (such as limited output voltage swings) or border regions of the discretization. Regions with nonlinear or static nonlinearities need much deeper investigation, too. In this section we will describe different classes of analog circuits and suggest some methods to detect them.

To differentiate the states in the M_{ATS} distance-based methods are used as well as eigenvalues, which are computed and stored during the discretization process in every state of the system. Static circuits like mixer or Low-dropout regulators can be compared using their linear or translinear behavior. In sum, this leads to five different coverage value vectors which will be described in the following. Each vector has the same length as the amount of states in the discrete state space and is either set to 0 or 1, depending on the response of the according method of inspect.

4.1 Local Linear Regions

Many analog circuits have a linear behavior and huge regions inside the discrete state space with similar behavior. To detect those regions, we are using the previously stored eigenvalues in every state of the system. Each state σ in the system has a list of ancestors and successor states. $\mathbf{L}_\sigma$ is set to 1 if one of the neighboring states has a significantly ($|\cdot| > 50\%$) different eigenvalue than σ, otherwise it is set to 0. Figure 4 shows a simple lowpass filter circuit and two large linear regions. As the circuit is ideal, no nonlinearities occur and we have a large linear region (blue).

On the other hand, in Fig. 5 the same analysis is conducted for an inverting active RC lowpass with an operational amplifier. This circuit has a large linear region and small nonlinear regions. The latter is due to shifted eigenvalues when the operational amplifier output reaches saturation at the supply rails and in this case also 0.7 V before reaching the 2.5 V positive supply rail.

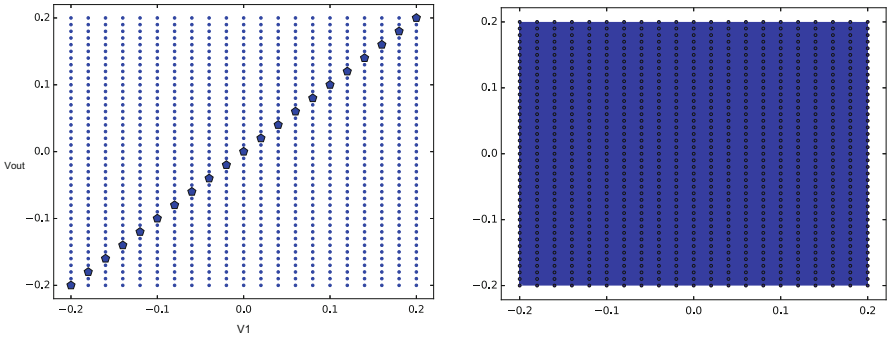

Fig. 4 Detection of linear regions: The discrete state space of a simple RC-lowpass filter circuit (*left*) and the values of $\mathbf{L}_\sigma$ (*right*). As the whole system is linear we got only one big linear region (*blue*)

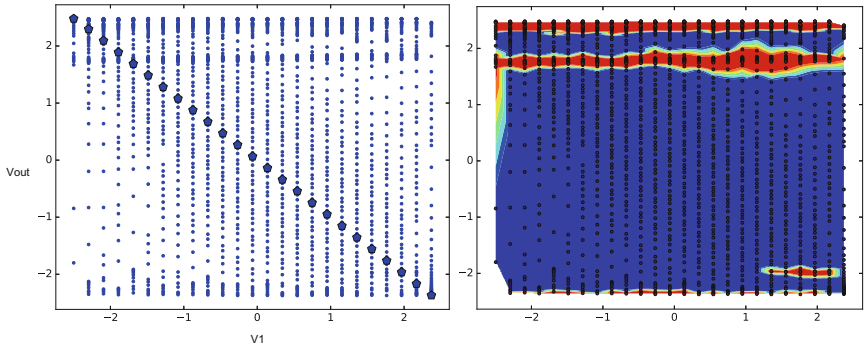

Fig. 5 Detection of linear regions: The discrete state space of an inverting active RC-lowpass filter circuit with limiting due to the used operational amplifier (*left*) and the values of $\mathbf{L}_\sigma$ (*right*)

4.2 Global Linear Regions

Similar to the previously described detection of local linear regions, global linear regions can be detected using the eigenvalues of the whole system. First of all, we compute the median of all eigenvalues of the whole M_{ATS}, as this indicates the basic dynamic level of the analog circuit. $\mathbf{D}_\sigma$ is then the absolute difference to the median value for each state $\sigma \in \Sigma_R$. All values are normalized to $[0, \ldots, 1]$ to ease the later summation process. Figure 6 shows the results of this detector for a basic Schmitt trigger circuit.

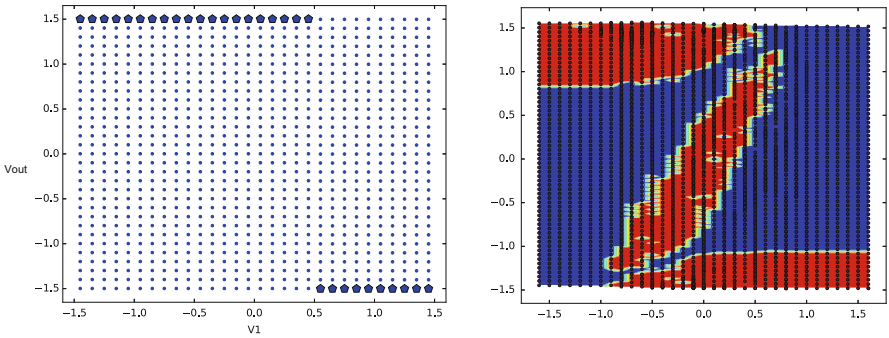

Fig. 6 Detection of global linear dynamic regions: The discrete state space of a Schmitt trigger circuit (*left*) and the resulting areas with nonlinear dynamics (*right, red points*)

4.3 Border Regions

As mentioned before, border regions are interesting and should be visited in any case by the path finding algorithm. To compute the states in the border region of the reachable set, the convex hull $conv(\Sigma_R)$ of all reachable states of the circuit is computed using the approach from [22]. $\mathbf{B}_\sigma$ is set to 1 if the state σ is located within a Euclidean distance on the edges of the resulting polytope, otherwise it is set to 0.

4.4 DC Operating Points

In the same manner as the border regions, the direct neighborhood of each DC operating point is computed. $\mathbf{O}_\sigma$ is set to 1 if the state σ lies in the neighborhood of the DC operating point or is the state itself. Figure 7 shows the result of the DC operating point detector as well as the border regions.

4.5 Static Circuits

Besides the so far described circuits, linear constant (Low-dropout regulators) or the so-called translinear circuits (mixer circuit) exist. For these class of analog circuits—which are easily discernible as they only consists of DC operating points—an optimal output function $\mathbf{f}_{\text{out}}$ exists. This function is currently guessed but could be automatically gathered by some sort of optimization process. $\mathbf{S}_\sigma = |\mathbf{f}_{\text{out}} - \mathbf{f}_{\text{meas}}|$ is the absolute error between the output function and measured output voltage of the analog circuit normalized to $[0, \ldots, 1]$. Figure 8 shows the result of the static circuit area detector.

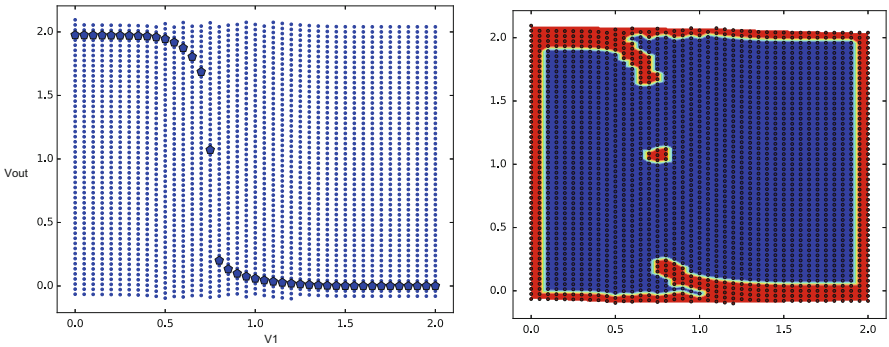

Fig. 7 Detection of regions at the border and around DC operating points: Discretization of an inverter example (*left*) and the resulting areas (*right*)

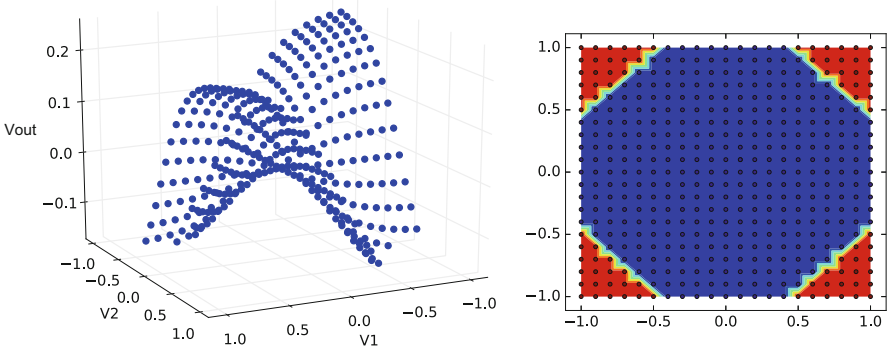

Fig. 8 Detection of static circuit regions: Discretization of a mixer circuit (*left*) and the resulting normalized error (*right, red points* indicate a high error)

As every step of the analysis described beforehand indicates possible interesting states of the full M_{ATS} system, the region of interest of the device under test is formed by the nonzero entries in $\mathbf{I}$ defined as:

$$\mathbf{I} = \mathbf{L} + \mathbf{D} + \mathbf{B} + \mathbf{O} + \mathbf{S}. \tag{3}$$

The importance of each state is now indicated by the according value in the vector $\mathbf{I}$. On the other hand, if its value is 0, none of the previously described detectors marked this state as important. This information can now be used and integrated in the path planning algorithm presented in Sect. 3. In this algorithm, each state was initialized with a node weight $\omega_\sigma = 0$ indicating that this state was never visited before by a transient simulation. According to this, we initialize the weight of a state by its interest factor I_σ:

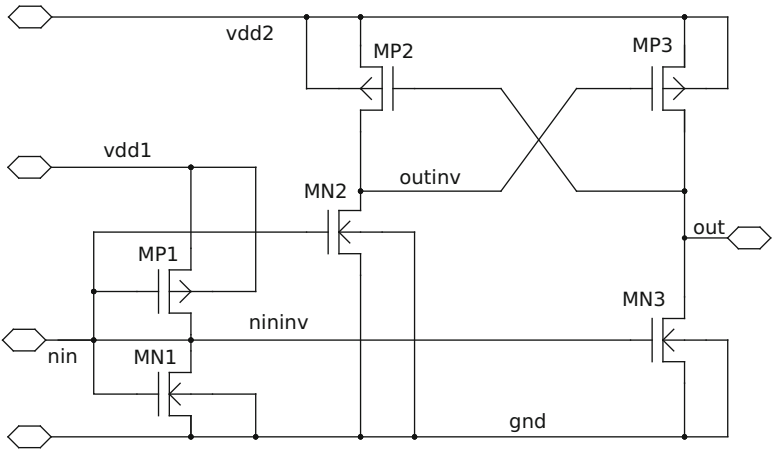

Fig. 9 Schematic of a levelshifter circuit

$$w_\sigma = \begin{cases} 0, & \text{if } I_\sigma \geq t, \\ 1, & \text{otherwise} \end{cases}, \tag{4}$$

where t is a given threshold. All states with a low weight ω_σ (and therefore a high interest value $I_\sigma \geq t$) are now preferred by the path finding algorithm. States with a higher weight are not removed from the path planning algorithm, so that there is still a small possibility that a simulation covers this state (Fig. 9).

With these information we are now able to create a reduced set of states $\Sigma_\lambda \subseteq \Sigma_R$ which consists of all interesting state space points with an interest factor $I_\sigma \geq t$:

$$\Sigma_\lambda = \{\sigma \in \Sigma_R | I_\sigma \geq t\}. \tag{5}$$

The λ *state space coverage* can now be defined as the number of visited states divided by the number of states in the reduced set Σ_λ, where in the numerator only states are counted which belong to that reduced set Σ_λ:

$$\zeta_\lambda = \frac{|C \cap \Sigma_\lambda|}{|\Sigma_\lambda|}. \tag{6}$$

The definition of a λ state space coverage and experimental results show clear evidence that the concept is still very pessimistic uncovering all design flaws with large possibility. Figure 10 shows a full λ state space analysis of a levelshifter circuit (Fig. 9). As a *normal* state space coverage calculation (as mentioned in Sect. 3) has to reach 1081 different states in the system, this number can be reduced to 641 by an analysis of the state space. Detailed results are listed in the next section and Table 2.

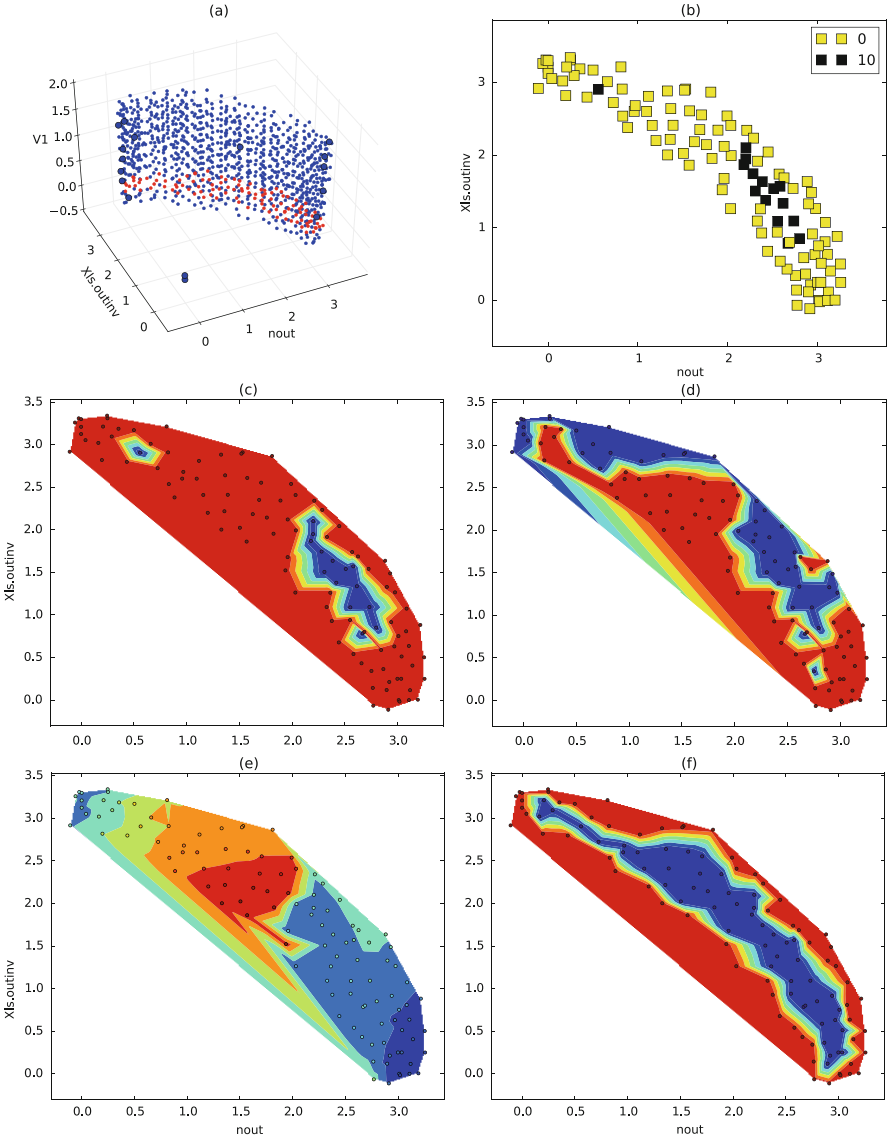

Fig. 10 Results of the λ state space analysis of the level-shifter circuit: (**a**) shows the reachable set in the state space. The *yellow points* in (**b**) show interesting points to be covered (i.e., the Σ_λ set), indicated by a high weight w_σ in (**c**). *Red* in (**d**) marks areas with high dynamic, based on the eigenvalue in (**e**). (**f**) shows border regions. For visualization purposes, (**b–f**) are plotted for the plane $V1 = 0.0\,\mathrm{V}$

5 Results

In this chapter we will demonstrate our proposed method on various analog circuits on transistor level as well as on some selected Verilog-A implementations (see Table 1). The examples try to cover many possible types of analog circuits: static nonlinear systems, dynamic linear systems, and dynamic nonlinear system to show the wide scope of our method.

In Table 2 three methods are presented. The *normal* method is taking every state of the discretization into account to calculate a coverage, while the *proposed* method only uses interesting states based on the criterion of Sect. 4. To show the overall speed-up, both methods are compared against the *single* method [15]. The experiments are carried out on a 3.4 GHz Dual-Core machine.

Starting from some very basic analog circuits, a lowpass filter has a high amount of states depending on the discretization accuracy. For this often used circuit, a lot of simulations seems to be needed to gain full coverage for a straightforward "normal" method. In comparison to that, the analysis of the state space reduces the number of interesting states by 74%. Full coverage can be reached by one simple simulation. This should be desired for circuits of that size. For another very basic example, the inverter circuit, the amount of states can be reduced by 77.6% as well (Fig. 11).

A bandpass filter [20] example with one input and two dimensions (Fig. 12) has more than 5000 states after the discretization process. Due to the heavy nonlinearities at the limiting region of the operational amplifier full coverage in this example is not possible, as not every as reachable marked state can really be reached by a simulation path. This is because there are always some discretization errors during the creation of the state space. Hence, a trajectory can be computed with the presented path finding algorithm from Sect. 3, but the created input stimulus for the simulation does not reach all wanted target states. After 173 simulations a coverage of 83.06% is obtained. With the presented state space analysis, only 1948

Table 1 Statistics of applied analog circuits on transistor level

Analog circuit	Number of inputs	Number of transistors	State space dimensions	States in reachable set $\lvert\Sigma_R\rvert$	States in λ reduced set $\lvert\Sigma_\lambda\rvert$	Schematic
RC lowpass filter	1	0	2	861	227	Basic
Inverter circuit	1	2	2	1510	338	Basic
Schmitt trigger	1	10	2	1903	356	[20]
Bandpass filter	1	8	3	5660	1948	[20]
Level shifter	1	6	3	1081	641	[23]
Log domain filter	1	13	2	2526	394	[24]
gmC filter	1	69	3	344	239	[24]
Mixer	2	–	2	442	120	Industrial
Low-dropout regulator	2	–	2	469	214	Industrial

The *Mixer* and *Low-dropout regulator* examples are implemented as a Verilog-A behavior model and therefore have no transistors as the other examples

Table 2 Results of the proposed coverage calculation algorithm

Analog circuit	Method	Coverage > 50%		Coverage > 75%		Overall coverage in %		
		# Sim.	Runtime	# Sim.	Runtime	Coverage	# Sim.	Runtime
RC lowpass filter	*Normal*	1	4.88	2	5.62	ζ100.00	5	7.28
	Proposed	1	1.60	1	1.60	ζ_λ100.00	1	1.60
	Single	1	7.11	1	7.11	ζ100.00	1	7.11
Inverter	*Normal*	1	0.15	2	0.23	ζ92.72	5	0.49
	Proposed	1	0.15	1	0.15	ζ_λ94.32	2	0.24
	Single	1	3.79	1	3.79	ζ89.93	1	3.79
Schmitt trigger	*Normal*	3	98.96	5	134.06	ζ87.06	17	215.83
	Proposed	1	5.62	2	8.16	ζ_λ91.40	3	10.79
	Single	1	19.29	1	19.29	ζ30.32	1	19.29
Bandpass filter	*Normal*	6	74.87	15	124.95	ζ83.06	173	556.86
	Proposed	1	31.08	6	97.37	ζ_λ88.99	18	164.75
	Single	1	388.96	–	–	ζ62.59	1	388.96
Level shifter	*Normal*	–	–	–	–	ζ43.91	9	81.46
	Proposed	1	12.81	–	–	ζ_λ72.21	6	27.97
	Single	1	266.24	–	–	ζ66.39	1	266.24
Log domain filter	*Normal*	1	1.91	2	2.15	ζ100.00	7	3.23
	Proposed	1	0.40	1	0.40	ζ_λ100.00	4	1.66
	Single	1	16.65	1	16.65	ζ100.00	1	16.65
gmC filter	*Normal*	2	0.58	7	2.98	ζ76.57	8	3.98
	Proposed	1	0.47	3	1.18	ζ_λ85.17	4	1.92
	Single	1	8.59	–	–	ζ65.71	1	8.59
Mixer	*Normal*	2	0.49	4	2.23	ζ100.00	8	5.23
	Proposed	1	0.43	3	1.48	ζ_λ100.00	6	3.92
	Single	–	–	–	–	–	–	–
Low-dropout regulator	*Normal*	1	1.37	3	2.40	ζ100.00	5	2.79
	Proposed	1	1.13	2	2.19	ζ_λ100.00	3	2.31
	Single	–	–	–	–	–	–	–

Normal is the presented path planning algorithm with underlying standard state space coverage metric. *Proposed* is the proposed path planning algorithm with underlying λ state space coverage metric. *Single* is the path planning algorithm from [15] (not available for *Low-dropout regulator* and *mixer* examples)

states are marked as interesting, so the overall sum of simulations can be reduced to only 18 simulations, reducing the overall runtime of the simulation by 70.4%. As the discretization error still exists, full coverage cannot be reached for that example, too.

Another unsophisticated, but also an example with strong nonlinearities is a Schmitt trigger circuit where the simulation runtime could be reduced by 95%. As this circuit has big areas with linear behavior and only a small region with nonlinear dynamics, the number of important states can be decreased dramatically.

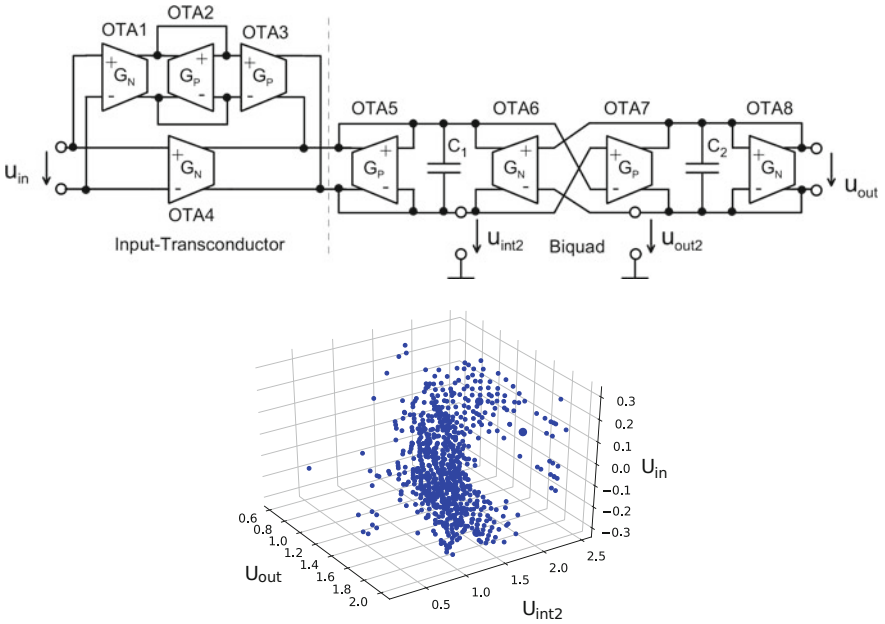

Fig. 11 Schematic of a real industrial gmC filter [24, 25] (*top*) and the result of the discretization process with 5660 states (*bottom*)

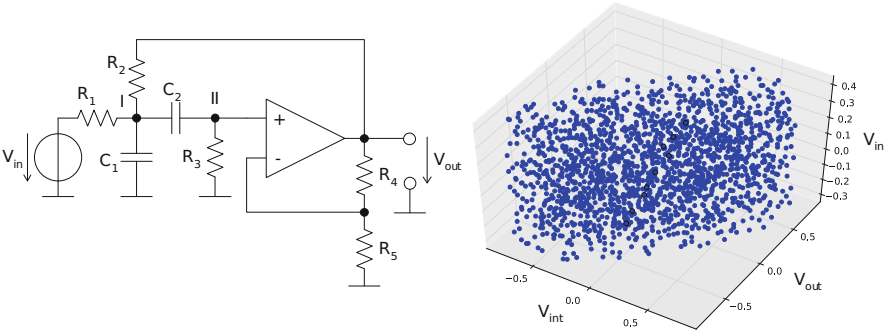

Fig. 12 Schematic of a bandpass filter [20] (*left*) and the result of the discretization process with 5660 states (*right*)

To demonstrate our approach on static nonlinear system, we calculate a coverage for two static examples: a mixer and a Low-dropout regulator circuit. Both implementations are used in an industrial environment. The resulting state space could be decreased to speed-up the simulation effort. Our presented method detects the interesting regions (e.g., high load for the LDO) automatically and calculates a high coverage by only running few simulations to that region.

To complete our result section, we are also able to create a discretization of a Verilog-A model description of an analog circuit (see mixer and low-drop regulator). This speeds up the simulation process on the one hand and also eases the implementation effort of sophisticated circuits. Using this approach, our presented method can also be used to create input stimuli to check the equivalence of two different implementations of the same system with large confidence.

6 Conclusion

In this chapter, a new coverage metric for analog circuits has been proposed. The λ state space coverage uses the eigenvalues and structural properties of the reachable state space of a nonlinear analog circuit on transistor level to extract a set of states in the state space which have to be visited by input stimuli. It keeps strongly nonlinear regions in that set while neglecting linear, uniform regions, resulting in 6.8 times speed-up of the simulation time of the generated stimuli. The quality of the input stimuli is still as high as with the presented standard state space-based coverage method and better than state-of-the-art methods. Experimental results show that hidden faults can be uncovered and real industrial circuits with up to 69 transistors (see Fig. 11) can be handled efficiently. We can conclude that the confidence in the input stimuli for a certain analog circuit can be measured by the proposed metric and that the verification coverage can be significantly increased with a small simulation overhead using the proposed λ state space coverage maximization algorithm.

Acknowledgements This work has partly been carried out in the project ANCONA, funded by the German Federal Ministry of Education and Research (BMBF) in the ICT2020 program under grant No 16ES021. The project is supported by the industry partners Infineon Technologies AG, Robert Bosch GmbH, Intel AG, and Mentor Graphics GmbH.

This work is partly supported by the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No 291763.

References

1. Eckmüller, J., Gröpl, M., & Gräb, H. (1998). Hierarchical characterization of analog integrated circuits. In *DATE '98: Design, Automation and Test in Europe*.
2. Ma, M., Hedrich, L., & Sporrer, C. (2014). ASDeX: A formal specification for analog circuit enabling a full automated design validation. *Design Automation for Embedded Systems, 18*(1), 99–118.
3. Dong, Z. J., Al Sammane, G., Zaki, M. H., & Tahar, S. (2007). Towards assertion based verification of analog and mixed signal designs using PSL. In *Forum on Design Languages (FDL)*.
4. Zaki, M. H., Tahar, S., & Bois, G. (2008). Formal verification of analog and mixed signal designs: A survey. *Microelectronics Journal, 39*(12), 1395–1404.
5. Steinhorst, S., & Hedrich, L. (2008, March 10–14).Model checking of analog systems using an analog specification language.In *Proceedings of the Design, Automation and Test in Europe DATE '08* (pp. 324–329).

6. Haedicke, F., Große, D., & Drechsler, R. (2012). A guiding coverage metric for formal verification. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012* (pp. 617–622). New York: IEEE.
7. Fine, S., & Ziv, A. (2003, June).Coverage directed test generation for functional verification using Bayesian networks. In *Design Automation Conference, 2003. Proceedings* (pp. 286–291).
8. Piziali, A. (2007). *Functional verification coverage neasurement and analysis* (1st ed.). New York: Springer Publishing Company, Incorporated.
9. Jou, J.-Y., & Liu, C. (1999). Coverage analysis techniques for hdl design validation. In *Proceedings of Asia Pacific CHip Design Languages* (pp. 48–55).
10. Arabi, K., & Kaminska, B. (1997, April). Parametric and catastrophic fault coverage of analog circuits in oscillation-test methodology. In *15th IEEE VLSI Test Symposium, 1997* (pp. 166–171).
11. Park, J., Madhavapeddiz, S., Paglieri A., Barrz, C., & Abraham, J. A. (2009, June). Defect-based analog fault coverage analysis using mixed-mode fault simulation. In *Mixed-Signals, Sensors, and Systems Test Workshop, 2009. IMS3TW '09. IEEE 15th International* (pp. 1–6).
12. Horowitz, M., Jeeradit, M., Lau, F., Liao, S., Lim, B. C., & Mao, J. (2010). Fortifying analog models with equivalence checking and coverage analysis. In *Proceedings of the 47th Design Automation Conference, DAC'10, New York, NY, USA* (pp. 425–430).
13. Julius, A., Fainekos, G. E., Anand, M., Lee, I., & Pappas, G. (2007). Robust test generation and coverage for hybrid systems. *Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control (HSCC)* (pp. 329–342).
14. Nahhal, T., & Dang, T. (2007). Test coverage for continuous and hybrid systems. In *Computer Aided Verification* (pp. 449–462). Heidelberg: Springer.
15. Steinhorst, S., & Hedrich, L. (2010, May). Improving verification coverage of analog circuit blocks by state space-guided transient simulation. In *IEEE International Symposium on Circuits and Systems*.
16. Karthik, A. V., Ray, S., Nuzzo, P., Mishchenko, A., Brayton, R. K., & Roychowdhury, J. (2014). ABCD-NL: Approximating continuous non-linear dynamical systems using purely Boolean models for analog/mixed-signal verification. In *ASP-DAC* (pp. 250–255).
17. Steinhorst, S., & Hedrich, L. (2012). Trajectory-directed discrete state space modeling for formal verification of nonlinear analog circuits. In *Proceedings of the International Conference on Computer-Aided Design* (pp. 202–209). New York: ACM.
18. Davis, A. (2003). An overview of algorithms in Gnucap. In *University/Government/Industry Microelectronics Symposium* (pp. 360–361).
19. Steinhorst, S., & Hedrich, L. (2010). Advanced methods for equivalence checking of analog circuits with strong nonlinearities. *Formal Methods in System Design, 36*(2), 131–147.
20. Hartong, W., Klausen, R., & Hedrich, L. (2004). Formal verification for nonlinear analog systems: Approaches to model and equivalence checking. In R. Drechsler (Ed.), *Advanced formal verification* (pp. 205–245). Boston: Kluwer Academic Publishers.
21. Bentley, J. L. (1975, September). Multidimensional binary search trees used for associative searching. *Communications of the ACM, 18*(9), 509–517.
22. Chazelle, B. (1993). An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry, 10*, 377–409.
23. Wang, W.-T., Ker, M.-D., Chiang, M.-C., & Chen, C.-H. (2001). Level shifters for high-speed 1 V to 3.3 V interfaces in a 0.13 μm Cu-interconnection/low-k CMOS technology. In *2001 International Symposium on VLSI Technology, Systems, and Applications. Proceedings of Technical Papers* (pp. 307–310). New York: IEEE.
24. Hedrich, L., & Hartong, W. (2001). Approaches to formal verification of analog circuits. In *Low-power design techniques and CAD tools for analog and rf integrated circuits* (pp. 155–192). Dordrecht: Kluwer Academic Publishers.
25. Python, D., & Enz, C. (1999). A 40 μW, 75 dB dynamic range, 70 kHz bandwidth biquad filter based on complementary MOS transsonductors. In *Solid-State Circuits Conference, 1999. ESSCIRC'99. Proceedings of the 25th European* (pp. 38–41). New York: IEEE.

Error-Free Near-Threshold Adiabatic CMOS Logic in the Presence of Process Variation

Yue Lu and Tom J. Kazmierski

Abstract This paper provides the first analysis of process variation effect on the adiabatic logic combined with near-threshold operation. One of the significant concerns is whether reliable performance is retained with voltage scaling. We find that typical variations of process parameters do not affect error-free operation at the minimum-energy frequency. Monte Carlo simulations of a 4-bit full adder using ECRL logic with 0.45 V supply voltage show that in the presence of typical process variations, energy consumption of the circuit operating at 25 MHz increases by 10.2% in the worst case while a 100% error-free operation is maintained. The maximum operating frequency (208 MHz) is reduced to nearly half of the nominal value (385 MHz). To further improve the robustness of the adder against process variation, a bit-serial adiabatic adder is considered with an even lower energy consumption per cycle.

Keywords Adiabatic logic • Efficient Charge Recovery Logic • Near threshold • Process variation • Low power

1 Introduction

Currently, with the emergence of energy-harvesting device [1] and biomedical applications requiring ultra-low energy consumption, new approaches should be explored to save more power. Voltage scaling is one of the most effective solutions to reduce power and has been practically applied in many digital designs. The reduction of the supply voltage causes a quadratic decrease in the dynamic power at the expense of increased delay. NTC (Near-Threshold Computing) [2, 3] is an important research approach, which reduces the supply voltage approximately to the threshold voltage, accordingly much of the energy savings typical of subthreshold operation can be retained with acceptable performance loss. Adiabatic logic [4, 5]

Y. Lu (✉) • T.J. Kazmierski (✉)
Faculty of Physical Sciences and Engineering, University of Southampton, Southampton SO17 1BJ, UK
e-mail: yl15g13@soton.ac.uk; tjk@ecs.soton.ac.uk

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9_6

is another attractive method for reducing the dynamic power, which uses ramping supply voltage sources to recover the energy which would otherwise dissipate as loss. Recently, some papers for near-threshold adiabatic circuits have been published [6, 7], where the nominal supply voltage is replaced by the near-threshold voltage to save even more power.

The impact of process variation on digital designs operating at near-threshold voltages cannot be ignored due to the reduced noise margin [2]. The variation sources can be divided into three major parts, process variation, supply voltage and operating temperature, which are known as PVT (process, voltage and temperature). This chapter considers effect of process variation on the reliability of near-threshold adiabatic logic. The transistor length and threshold voltage are the two major parameters affected by process variation because of sub-wavelength lithography, etching process steps and random dopant fluctuations. It has been demonstrated [8] that effective mobility also emerges as another leading effect of process variation due to the local variation of mechanical stress.

The sensitivity to process variation is significantly increased when the supply voltage is reduced to near-threshold region. As supply voltage gets close to the threshold voltage, both the dynamic and leakage powers decrease, but their slow-down rates are different. Leakage power shows a slower reduction, thus its ratio in total power dissipation is increased. Since the leakage power is exponentially dependent on the threshold voltage, the variation of the threshold voltage may lead to serious power fluctuations in a near-threshold design. It is well known that near-threshold adiabatic designs can save substantial energy; however, estimated values of energy consumption would be inaccurate if process variation is not taken into account [9]. Also the optimal frequency, where the minimum energy point is attained, is significantly affected. To the best of our knowledge, no paper has analysed effects of process variation on near-threshold adiabatic logic before. As a case study, two implementations of 4-bit near-threshold adiabatic full-adder are considered to demonstrate effects of process variation. The sensitivity of the energy consumption and operating frequency on process variation is discussed. We show that a bit-serial implementation reduces the energy consumption four times and maintain an error-free operation while 0.36% of the parallel adder design are erroneous.

Additionally, to further improve the robustness of adiabatic operation at near-threshold voltage, a adder based on bit-serial structure is considered.

2 ECRL Logic Operating at Near-Threshold Voltage

Over the last two decades, a large number of adiabatic logic families have been proposed [5, 10, 11]. Among them, Efficient Charge Recovery Logic(ECRL) is the most traditional one, firstly proposed in [10]. ECRL consists of a cross-coupled PMOS pair to store information and the pull-down network is composed with two NMOS transistors connecting the input signals. The structure of an ECRL buffer is shown in Fig. 1.

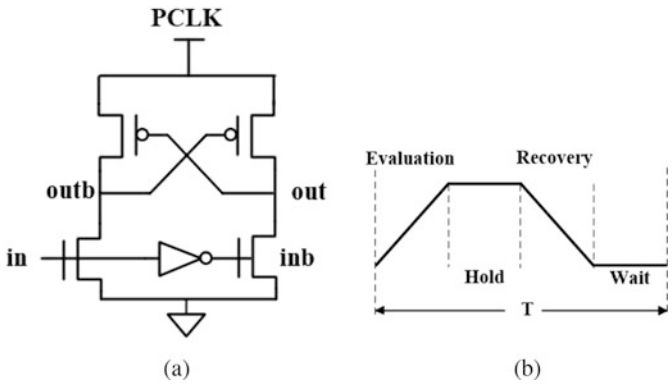

Fig. 1 ECRL buffer [10]. (**a**) Schematic. (**b**) Power clock

The energy loss mechanisms of the adiabatic logic is composed of these three energy dissipation, adiabatic, leakage and non-adiabatic losses [5].

$$E_{\text{total}} = E_{\text{adia}} + E_{\text{leak}} + E_{\text{non-adia}} \tag{1}$$

To be specific, adiabatic and leakage energy consumption can be expressed as,

$$E_{\text{adiabatic}} = \int_0^T R\frac{C^2V_{\text{DD}}^2}{T^2}dt = 2\frac{RC}{T}CV_{\text{DD}}^2 \tag{2}$$

$$E_{\text{leak}} = V_{\text{DD}}\overline{I_{\text{leak}}}\frac{1}{f} \tag{3}$$

where C represents output capacitance, T denotes the transition time of power clock, and f and V_{DD} are the operation frequency and peak-peak supply voltage, respectively.

When ECRL logic circuit operates in near-threshold regime, the supply voltage V_{DD} is set slightly higher than the threshold voltage, thereby causing reduction of both adiabatic and leakage energy consumption, accordingly a substantial portion of energy can be saved. Because of different decrease rate with reduction of supply voltage, the leakage energy consumption would take larger part in total energy dissipation. It is worth mentioning, although ECRL logic can work in subthreshold region, the operation frequency range is quite small and extremely sensitive to process variations. Once the switching probability is quite low, the optimal operating frequency would be very close to maximum frequency, easily affected by process variation. Comparatively, near-threshold computing of ECRL circuit can not only save large energy cost, but also improve the robustness against process variation.

3 Analysis of Process Variation Effect on Near-Threshold Adiabatic Operation

To demonstrate the performance of adiabatic logic and near-threshold operation, a 4-bit ripple carry adder is designed and simulated. Ripple carry adder consists of full-adder, which can be realized by XOR and NAND gates. The schematic of ripple carry adder is shown in Fig. 2. The sum and carry parts of full adder are implemented by Efficient Charge Recovery Logic (ECRL). The transistor-level circuits can be seen in Fig. 2.

The whole design is implemented using Hspice simulator and the functionality of circuit can be confirmed through the simulations. In the experiment, the supply voltage is set to 0.45 V, which is slightly larger than the absolute value of PMOS threshold voltage (0.423 V). To compare the energy consumption of circuit operating at different voltage region, we adjust the peak-peak voltage to 0.7 and

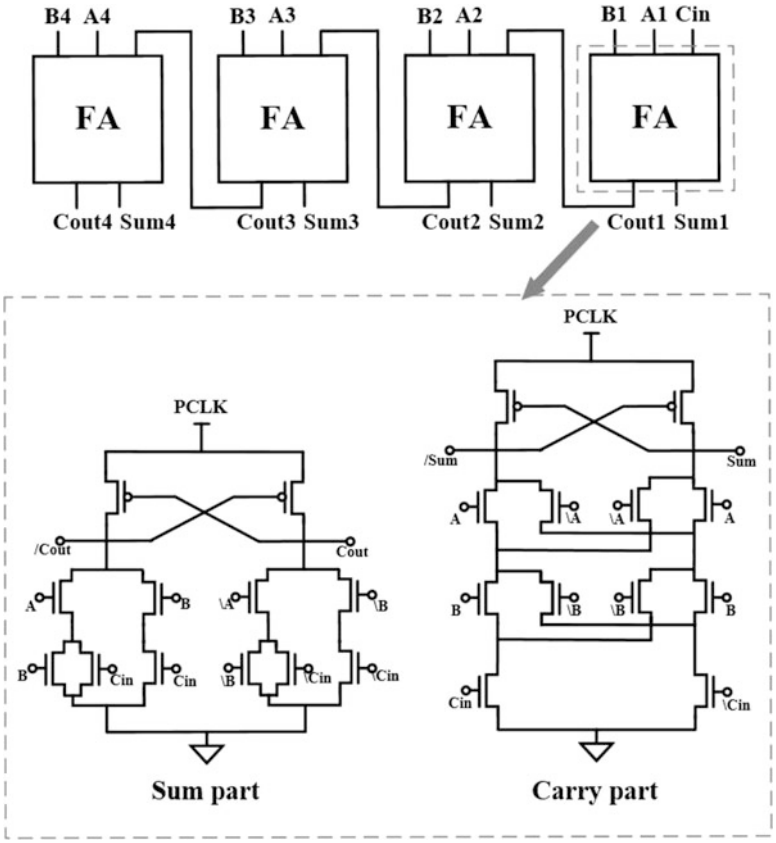

Fig. 2 Topology of ECRL 4-bit full adder [5]

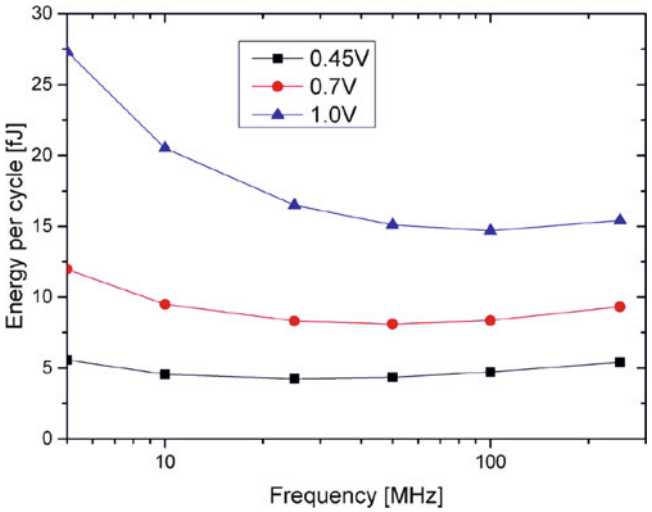

Fig. 3 Energy dissipation of adiabatic 4-bit full adder at 0.45, 0.7 and 1.0 V

Table 1 Optimal frequency, maximum frequency and minimum energy per cycle of adiabatic 4-bit full adder circuit operating in both super (1.0 V) and near-threshold (0.45 V) region

Vdd (V)	Optimal freq (MHz)	Max freq (MHz)	Minimum energy/cycle (fJ)
1.0	125	1500	14.20
0.45	25	384.6	4.25

1.0 V, respectively. Figure 3 shows the changes of energy dissipation with different supply voltage, it seems the near-threshold ECRL 4-bit full-adder operating at the peak-peak voltage of 0.45 V saves at least 60% energy consumption compared with nominal voltage design when the operating frequency is below 125 MHz. Besides, optimal frequency, maximum frequency and minimum energy per cycle of the design have been tested and shown in Table 1.

In order to verify the robustness of full-adder design operating in near-threshold regime, process variations are taken into consideration in the design. According to the experiment results in [12], for ECRL circuits, the effect of inter-die variations is much smaller than that of intra-die variations; therefore, only intra-die parameter variations are considered in the design. All the experiments are demonstrated based on 65 nm PTM technology [13], and the relevant process variations are described in [8]. The following three transistor parameters are recognized as the leading sources of process variation: gate length, threshold voltage and mobility. Their mean values and standard deviations for both NMOS/PMOS transistors are shown in Table 2. In [14], the author compares the calculated value of the standard deviation with another using the equation presented in [15]; it seems the difference is negligible (approximately 5 mV) for both PMOS and NMOS transistors.

Table 2 Varied process parameters [8]

Parameter	Mean	Standard deviation (%)
Length	70 nm	±4
V_{thn}	0.423 V	±5
V_{thp}	−0.365 V	±5
μ_{effn}	491 $cm^2/V\,s$	±21
μ_{effp}	57.4 $cm^2/V\,s$	±21

Table 3 Effect of process variation on energy consumption of 0.45 V design operating in nominal optimal frequency

Parameters	Avg energy/cycle	Max energy/cycle	Min energy/cycle
Length	4.29 fJ(+0.79%)	4.36 fJ(+2.49%)	4.21 fJ(−0.95%)
V_{th}	4.29 fJ(+0.99%)	4.70 fJ(+10.63%)	4.07 fJ(−4.187%)
Mobility	4.28 fJ(+0.76%)	4.43 fJ(+4.1%)	4.18 fJ(−1.66%)
Total	4.30 fJ(+1.11%)	4.69 fJ(+10.21%)	4.06 fJ(−4.55%)

The influence of the length, threshold voltage and mobility on the energy consumption can be seen in Table 3, where threshold voltage has stronger effect on the average, maximum and minimum energy consumption per cycle compared with the other two parameters. To be specific, variation for average energy per cycle is equal to 1.11%, but in worst case, the process variations would lead to 10.21% extra energy dissipation.

As seen in Fig. 4, the changes of maximum and minimum energy consumption in the presence of process variation are given, and the difference with the nominal value of energy cost can be clearly observed. In general, the variation of energy dissipation is becoming more stable with the increase of operation frequency. Besides, the optimal frequency of the whole design is largely affected by process variation. To demonstrate the reliability of near-threshold design in optimal frequency, the simulation waveforms of the fourth single-bit full-adder is shown in Fig. 5. The input signals are power clock, A4, B4 and the carry bit from the previous adder, while the results of sum and carry bits can be observed in the waveforms. Besides, the orange, red and blue simulation waveforms are represented by design with nominal parameters and varied parameters (maximum and minimum energy dissipation per cycle). Clearly, the design with varied parameters can also realize correct functionality; therefore, its reliability in nominal optimal frequency can be confirmed.

In the presence of process variation, error rate of circuit operating in nominal maximum frequency (385 MHz) achieves 52.55%, and it approximately becomes zero when the working frequency is reduced to 208.3 MHz in Fig. 6. Moreover, it is demonstrated that performance of the near-threshold adiabatic design is quite stable at optimal frequency; even when the circuit is affected by process variation, error rate in the optimal frequency remains zero.

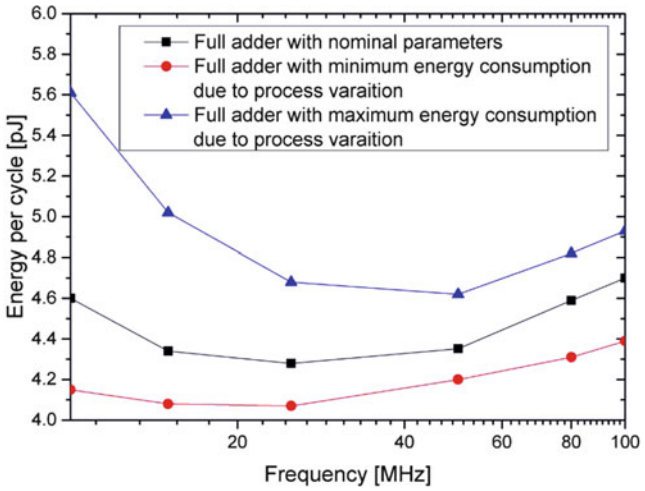

Fig. 4 Energy dissipation of 0.45 V adiabatic full-adder design with nominal and varied parameters

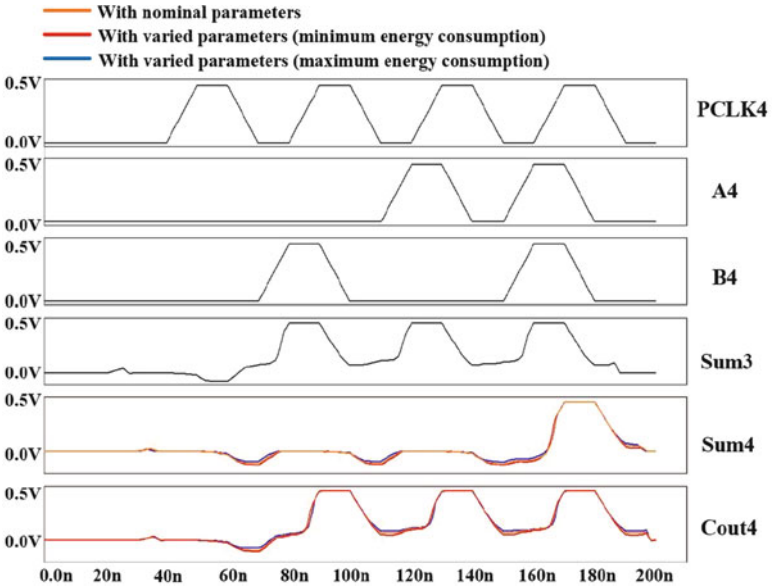

Fig. 5 Simulation waveforms of adiabatic 4-bit full adder with varied parameters at 0.45 V voltage

As discussed before, variation of the threshold voltage has the most significant influence on the performance of circuits. In order to further study the tolerance of near-threshold adiabatic logic against process variations, the standard deviation of Monte Carlo simulation for threshold voltage is increased. According to experiment

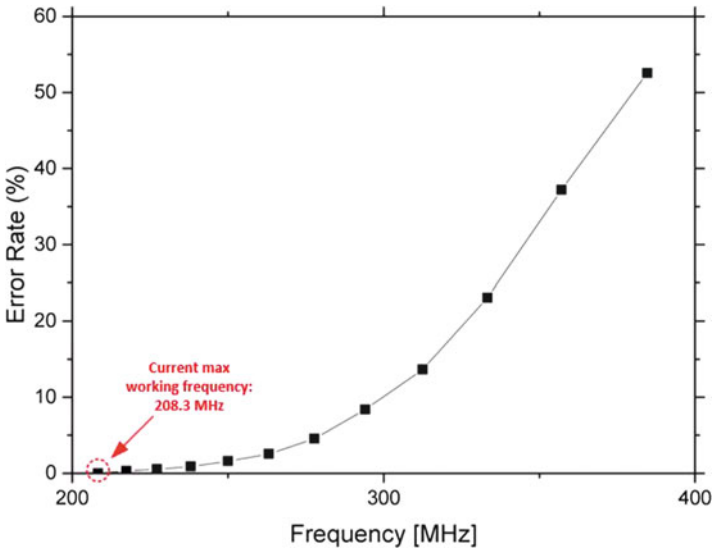

Fig. 6 Error rate of near-threshold adiabatic full-adder design with process variation at 0.45 V voltage

results, when the standard deviation is set to 14%, the error rate of 4-bit full-adder operating in 200 MHz would be non-zero, approximately 0.36%. For simplicity, the waveforms of the fourth single-bit full-adder are presented in Fig. 7 and failed waveforms are highlighted using red cycle. Obviously, with the increase of circuit complexity, reliability issues of near-threshold adiabatic logic would be more serious.

4 Near-Threshold 4-Bit Adiabatic Adder Based on Bit-Serial Structure

As shown in the previous experiment results, once the threshold voltage variation become more serious, up to 14%, the adiabatic operation at near-threshold region would not be error-free anymore. To further improve the robustness of the adiabatic logic circuit while ensuring its characteristics of low-power, a 4-bit adder is reconfigured based on bit-serial structure. The bit-serial adiabatic adder consists of a single-bit full-adder and a flip-flop, realized using the ECRL adiabatic technique. Since adiabatic logic implements computation sequentially, a flip-flop in an adiabatic system could be realized by three buffers.

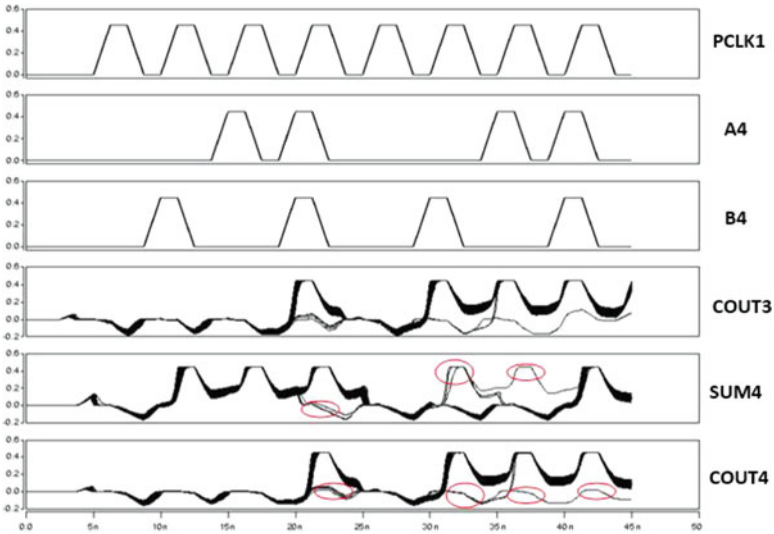

Fig. 7 Monte Carlo simulation waveforms of adiabatic 4-bit full adder with varied parameters at 0.45 V voltage

Simulation results shows that the energy can be reduced four times compared with the parallel design (see Table 4). Additionally, in the serial arithmetic, 4-bit addition can be realized using only one single-bit adder, therefore the transistor count is also significantly reduced. Only 36 transistors are required.

Bit-serial operation also improves the robustness against process variation. In the serial adiabatic adder, fewer transistors are connected in a stack, and accordingly the effects of variation on the threshold voltage are alleviated. To demonstrate the robustness of the bit-serial design, Monte Carlo simulations with varying threshold voltage have been carried out. The results can be observed in Fig. 9. Compared with the equivalent parallel design, when the variation of the threshold voltage reaches to 14%, the bit-serial adiabatic design can still maintain the error-free operation.

5 Conclusion

This chapter provides the first analysis of process variation effects on the energy consumption and reliability of near-threshold adiabatic logic. As a case of study, a 4-bit full adder is tested and results show that, in the worst case, process variation could cause 10.2% extra energy dissipation at the optimal frequency, while the performance is not significantly affected. The maximum frequency is reduced by

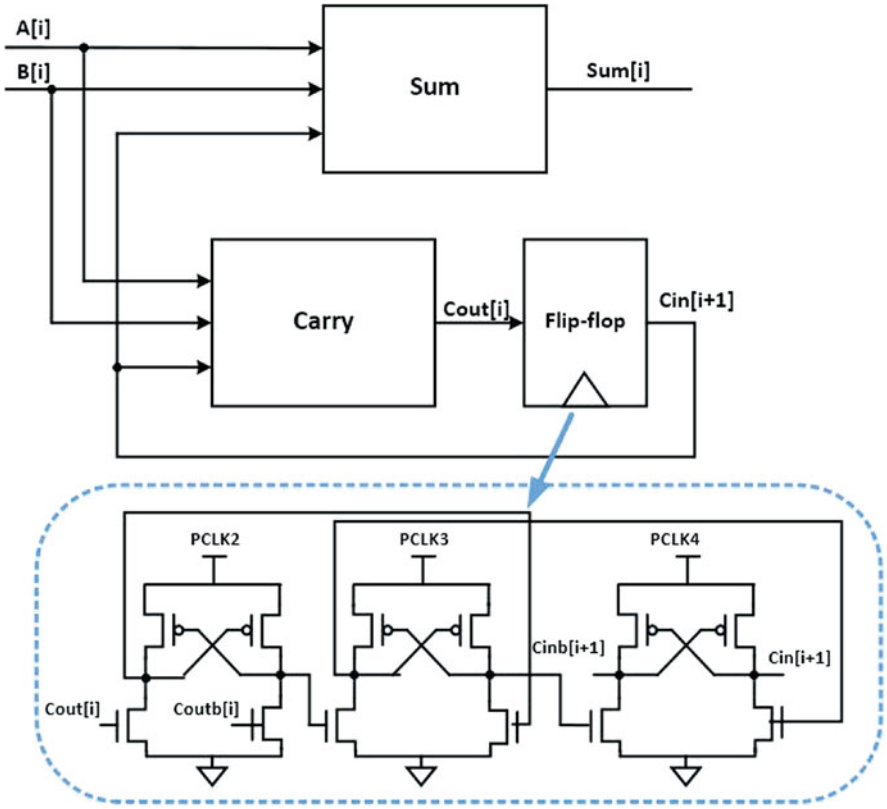

Fig. 8 A possible configuration of a 4-bit adiabatic adder based on bit-serial structure

Table 4 The comparison of the 4-bit adder based on normal and bit-serial architecture in terms of energy consumption (operating at 25 MHz with 0.45 V supply voltage) and transistor count

Architecture	Energy/cycle (fJ)	Transistor count
Normal	4.25	96
Bit-serial	1.10	36

45.8% and approximately equal to 208 MHz. In a bit-serial implementation, the energy per cycle can be further reduced while the robustness is also significantly improved. Robustness of near-threshold adiabatic logic in the presence of process variation makes it a promising candidate for future ultra-low-energy-consumption digital designs.

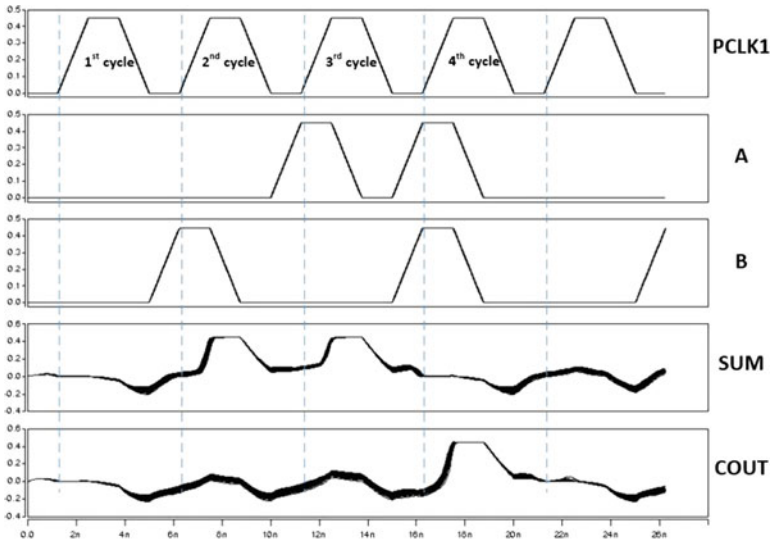

Fig. 9 Monte Carlo simulation waveforms of adiabatic 4-bit full adder with varied parameters at 0.45 V voltage based on bit-serial structure

References

1. Kazmierski, T. J., & Beeby, S. (2011). *Energy harvesting systems*. New York: Springer.
2. Hanson, S., Zhai, B., Bernstein, K., Blaauw, D., Bryant, A., Chang, L., et al. (2006) Ultralow-voltage, minimum-energy CMOS. *IBM Journal of Research and Development, 50*(4.5), 469–490.
3. Kaul, H., Anders, M., Hsu, S., Agarwal, A., Krishnamurthy, R., & Borkar, S. (2012). Near-threshold voltage (NTV) design: Opportunities and challenges. In *Proceedings of the 49th Annual Design Automation Conference* (pp. 1153–1158). New York: ACM.
4. Yemiscioglu, G., & Lee, P. (2012). 16-bit clocked adiabatic logic (CAL) logarithmic signal processor. In *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)* (pp. 113–116). New York: IEEE.
5. Teichmann, P. (2011). *Adiabatic logic: Future trend and system level perspective* (Vol. 34). Dordrecht: Springer Science & Business Media.
6. Sathe, V. S., Papaefthymiou, M. C., & Ziesler, C. H. (2005). Boost logic: A high speed energy recovery circuit family. In *IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI'05)* (pp. 22–27). New York: IEEE.
7. Hu, J., & Yu, X. (2010). Near-threshold adiabatic flip-flops based on pal-2n circuits in nanometer cmos processes. In *2010 Second Pacific-Asia Conference on Circuits, Communications and System (PACCS)* (Vol. 1, pp. 446–449). New York: IEEE.
8. Zhao, W., Liu, F., Agarwal, K., Acharyya, D., Nassif, S. R., Nowka, K. J., et al. (2009). Rigorous extraction of process variations for 65-nm CMOS design. *IEEE Transactions on Semiconductor Manufacturing, 22*(1), 196–203.
9. Karpuzcu, U. R., Kim, N. S., & Torrellas, J. (2013). Coping with parametric variation at near-threshold voltages. *IEEE Micro, 33*(4), 6–14.
10. Moon, Y., & Jeong, D.-K. (1996). An efficient charge recovery logic circuit. *IEEE Journal of Solid-State Circuits, 31*(4), 514–522.

11. Vetuli, A., Pascoli, S. D., & Reyneri, L. M. (1996). Positive feedback in adiabatic logic. *Electronics Letters, 32*(20), 1867–1869.
12. Amirante, E., Bargagli-Stoffi, A., Fischer, J., Iannaccone, G., & Schmitt-Landsiedel, D. (2001). Variations of the power dissipation in adiabatic logic gates. In *Proceedings of the 11th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS* (Vol. 1, pp. 9–1).
13. Arizona State Univ. PTM. (2006, February). *65nm BSIM4 model card for bulk CMOS.* Arizona State Uni. http://ptm.asu.edu/modelcard/2006/65nm_bulk.pm. Accessed 22 Aug 2017.
14. Khursheed, S., Zhong, S., Aitken, R., Al-Hashimi, B. M., & Kundu, S. (2010) Modeling the impact of process variation on resistive bridge defects. In *2010 IEEE International Test Conference (ITC)* (pp. 1–10). New York: IEEE.
15. Asenov, A., Brown, A. R., Davies, J. H., Kaya, S., & Slavcheva, G. (2003). Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale mosfets. *IEEE Transactions on Electron Devices, 50*(9), 1837–1852.

Index

F. Fummi, R. Wille (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 454,
DOI 10.1007/978-3-319-62920-9