Preserving and Improving Verifiability of Circuits based on Local Transformations

Rolf Drechsler

Institute of Computer Science
University of Bremen/DFKI
28359 Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—The concept of testability preserving or even testability improving circuit transformations has been studied intensively. It has been demonstrated for various fault models, that circuits can be optimized with respect to area and/or delay, while considering testability at the same time. Recently, *Polynomial Formal Verification* (PFV) has been introduced, where upper bounds on run time and space complexity of the algorithms – ensuring 100% correctness – are given. While the testability aspects were properties of the underlying circuits, here we propose a similar approach in the context of verification algorithms: verification preserving transformations and verification improving transformations. This is discussed for PFV, while the concept can be considered for formal verification techniques in general and also for simulation-based approaches.**

***Index Terms*—circuit design, correctness, verification, test, formal methods, BDD**

I. Introduction

Test and verification are essential steps for ensuring the correct functional behavior of fabricated circuits and systems. In the testing field, approaches for *Design for Testability* (DfT) have been studied intensively and resulted in formulation of standards, like JTAG or IJTAG (see e.g. [1]). Furthermore, the research in the early 90s showed that testability can already be considered during the (logic) synthesis of a circuit. It can be shown that there exist *Testability Preserving Transformations* (TPT) that ensure that an optimization of the circuit will not decrease the testability. This has been investigated for various fault models, like stuck-at or path-delay (see [2]–[7]).

In the domain of verification, similar concepts are not established yet. While simulation and emulation are still widely used in practice, only *Formal Verification* can ensure 100% correctness [8]. But the underlying proof techniques are known to be very cost intensive with regards to runtime and space requirement.

Recently, *Polynomial Formal Verification* (PFV) [9], [10] has been introduced, focusing on providing efficient upper bounds on the verification algorithms. It has been shown for various types of circuits that algorithms with polynomial worst-case behavior can be developed. As suggested in [11], making use of the similarities of testing and verification, based on the idea of TPT, we look at the following research question:

> Can we design *Verification Preserving Transformations* (VPTs) or even *Verification Improving Transformations (VITs)*?

In the following, we discuss VPT and VIT in the context of PFV, but the approach can directly be generalized to simulation-based approaches where the improvement can be measured, e.g. for given testbenches.

The paper is structured as follows: In Section II, the core idea of PFV is discussed and examples are provided in Section III. In Section IV, we summarize the main findings and discuss directions for future work.

II. Polynomial Formal Verification

In *Polynomial Formal Verification* (PFV) [9], [10] a proof engine is provided for a given circuit or system that allows a complete verification in polynomial time and space. It is important to notice that this does not only hold for the final result, but for the complete verification run. E.g. it has been proven in [9] that based on BDDs, various adder architectures, like carry ripple or conditional sum, can be efficiently verified.

Once such a result is provided, the question arises whether the circuit can also be optimized with respect to area and delay while keeping the efficiency of the verification run.

Remark II.1. *While in the testing domain TPT are a property of the circuit alone, for VPT and VIP also the verification algorithm itself is considered.*

III. Application

In this section, we show two examples that demonstrate the effect of the circuit structure on the verification process.

A. SCA-based Verification of Multipliers

Multiplier circuits are known to be hard to verify based on bit-level solvers, like BDDs or SAT, while techniques based on *Symbolic Computer Algebra* (SCA) can handle these circuits efficiently (see [12]).

Parts of this work have been supported by DFG within the Reinhart Koselleck Project *PolyVer: Polynomial Verification of Electronic Circuits* (DR 287/36-1).

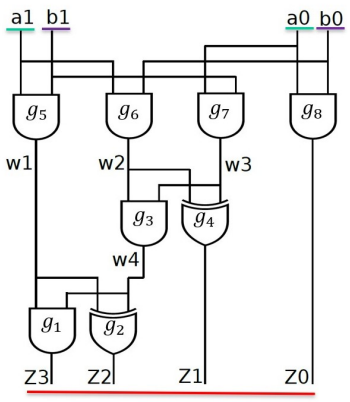

Fig. 1: Two-bit multiplier circuit

$$\text{SP} \coloneqq 8Z_3+4Z_2+2Z_1+Z_0 - (4a_1b_1+2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$\text{SP} \xrightarrow{g_1} \text{SP}_1 \coloneqq 8w_1w_4+4Z_2+2Z_1+Z_0 - 4a_1b_1 - 2a_1b_0 - 2a_0b_1 - a_0b_0$$

$$\text{SP}_1 \xrightarrow{g_2} \text{SP}_2 \coloneqq 8w_1w_4+4w_1+4w_4 - 8w_1w_4+2Z_1+Z_0 - 4a_1b_1 - 2a_1b_0 - 2a_0b_1 - a_0b_0$$

...

$$\text{SP}_7 \xrightarrow{g_8} r \coloneqq a_0b_0 - a_0b_0 = 0 \Longrightarrow \text{The circuit is bug-free!}$$

Fig. 2: Reduction of SP

In the following, a deeper insight on how the algorithms work is provided: In Figure 1, the gate-level netlist of a two-bit multiplier is given. The *Specification Polynomial* (SP) is then given by:

$$Z - A \cdot B = (8Z_3 + 4Z_2 + 2Z_1 + Z_0) - (2a_1 + a_0) \cdot (2b_1 + b_0)$$

The SP and the gate-level netlist are equivalent, iff the remainder becomes 0 when dividing SP by gate polynomials. The core SCA algorithm consists of backward rewriting and iterative substitution in reverse topological order. The process is sketched in Figure 2. After the substitution of gates g_1 and g_2 the term $8w_1w_4$ appears in positive and negative form and thus, can be removed. If all gates are substituted, in case of a correct circuit, the result is 0. Of course, the substitution order has a strong influence on the performance of the algorithm (see also [13]). In the example, if instead of g_2 an other gate had been chosen, like e.g. g_8, the immediate removal of the term $8w_1w_4$ would not have been possible. Thus, the algorithm would have to store the term resulting in larger memory requirements.

Once a polynomial substitution sequence has been proven for a circuit, all transformations of the circuit have to show that the early cancellation of terms remains.

B. BDD-Circuits

Since each Boolean function can be represented by a BDD, it is straightforward to map a BDD to a circuit by substituting each node by a multiplexer [14]. Furthermore, the depth can be reduced to derive circuits of logarithmic depth [15]. In these cases, the polynomial sizes are proven for internal signals, while some of the intermediate computations can only be estimated by the known complexity of the ite-operator [16]. In cases where the number of calls can be reduced by local transformations, this directly has a positive effect on the bounds for the verification process:

$$a \cdot b + a \cdot c = a \cdot (b + c)$$

In this case, instead of three synthesis operations, only two are needed, which will reduce the worst-case complexity.

IV. Conclusion and Future Directions

Following the technique of TPTs known from the testing domain, we introduced VPTs and VITs. This has been demonstrated on two FV scenarios. Future work is to study circuit transformations and their effect on verification techniques: formal approaches, but also simulation-based techniques.

References

[1] J. Rearick, B. Eklow, K. Posse, A. Crouch, and B. Bennetts, "IJTAG (internal JTAG): a step toward a DFT standard," in *IEEE International Conference on Test*, 2005, pp. 8 pp.–815.

[2] S. Kundu and A. Pramanick, "Testability preserving Boolean transforms for logic synthesis," in *Digest of Papers Eleventh Annual 1993 IEEE VLSI Test Symposium*, 1993, pp. 131–138.

[3] S. Devadas and K. Keutzer, "Synthesis and optimization procedures for robustly delay-fault testable combinational logic circuits," in *27th ACM/IEEE Design Automation Conference*, 1990, pp. 221–227.

[4] J. Rajski and J. Vasudevamurthy, "Testability preserving transformations in multi-level logic synthesis," in *Proceedings. International Test Conference 1990*, 1990, pp. 265–273.

[5] H. Hengster, R. Drechsler, and B. Becker, "Testability properties of local circuit transformations with respect to the robust path-delay-fault model," in *Proceedings of 7th International Conference on VLSI Design*, 1994, pp. 123–126.

[6] ——, "On the application of local circuit transformations with special emphasis on path delay fault testability," in *Proceedings 13th IEEE VLSI Test Symposium*, 1995, pp. 387–392.

[7] ——, "On local transformations and path delay fault testability," *J. Electron. Test.*, vol. 7, no. 3, p. 173–191, Dec. 1995. [Online]. Available: https://doi.org/10.1007/BF00995312

[8] R. Drechsler, *Formal System Verification*. Springer, 2018.

[9] ——, "PolyAdd: Polynomial formal verification of adder circuits," in *International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2021, pp. 99–104.

[10] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *Int'l Conf. on CAD*, 2022.

[11] R. Drechsler, "Fast and exact is doable: Polynomial algorithms in test and verification," in *2022 IEEE 23rd Latin American Test Symposium (LATS)*, 2022, pp. 1–2.

[12] A. Mahzoon, D. Große, and R. Drechsler, "PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers," in *Proceedings of the International Conference on Computer-Aided Design, ICCAD*, I. Bahar, Ed. ACM, 2018, pp. 1–8.

[13] A. Konrad and C. Scholl, "Symbolic computer algebra for multipliers revisited - it's all about orders and phases," in *Int'l Conf. on Formal Methods in CAD*, 2024.

[14] R. Drechsler, "Polynomial circuit verification using BDDs," in *2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, 2021, pp. 49–52.

[15] R. Drechsler and C. Dominik, "Edge verification: Ensuring correctness under resource constraints," in *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2021, pp. 1–6.

[16] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in *Design Automation Conf.*, 1990, pp. 40–45.