Design for Polynomial Formal Verification: How Design Choices Impact Formal Verification

Luca Müller
DFKI GmbH
Bremen, Germany
luca.mueller@dfki.de

Martha Schnieber
University of Bremen
Bremen, Germany
schnieber@uni-bremen.de

Rolf Drechsler
University of Bremen
DFKI GmbH
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—The increasing complexity of *Integrated Circuits (ICs)* is felt throughout the whole *Electronic Design Automation (EDA)* process. Verification is especially affected, as complete formal coverage is desired, but hard to achieve due to the NP-completeness of underlying problems. *Polynomial Formal Verification (PFV)* aims to address this limitation, offering efficient formal verification approaches that provide provable polynomial complexity bounds. Previous results in this direction have shown that design choices impact formal verification, as different implementations of the same functionality lead to different levels of verifiability.**

In this work, we elaborate on this fact and introduce the concept of *Design for Polynomial Formal Verification (DfPFV)*, which considers efficient verifiability as an additional design parameter. As a case-study, we investigate symmetric functions, evaluating three different circuit implementations on their gate count, depth, and verifiability. Our evaluation shows that our proposed DfPFV approach not only ensures the verifiability of the design but is also able to improve the other two metrics for larger circuit sizes. Finally, we provide further insight on what these results imply and how DfPFV may be practically applied in the future.

I. Introduction

The ever-rising complexity of *Integrated Circuits (ICs)* in accordance with Moore's law poses challenges throughout the whole *Electronic Design Automation (EDA)* process [1]. While for many steps during synthesis, heuristic solutions are sufficient, complete coverage is desired for verification, which consumes a large part of the total design effort. This requires formal methods, but the NP-completeness of employed techniques results in a race between efficient verifiability and problem size [2]. This is where *Polynomial Formal Verification (PFV)* comes into play, contributing formal verification approaches that ensure efficient verifiability by providing provable polynomial upper complexity bounds [3]. Previous work in this field of research has shown that these bounds vary for different implementations of the same functionality, indicating that design choices impact verifiability [4].

This paper elaborates on these findings and introduces the concept of *Design for Polynomial Formal Verification (DfPFV)*, inspired by the widely adopted notion of *Design for Test (DfT)* [5]. The goal of DfPFV is to achieve an efficient, high-coverage verification process combined with near-optimal circuit properties on metrics like gate count and depth by considering efficient verifiability early on in the design process. Compared to previous work on *Design for Verification (DfV)* [6][7], DfPFV puts special emphasis on the efficiency of the verification process. To this end, PFV approaches are utilized for DfPFV to ensure that both efficient verifiability and full coverage are achieved. DfPFV is showcased on the case-study of symmetric functions, where three different implementations under consideration impact circuit metrics and verifiability. We demonstrate that a good trade-off between these two aspects can be achieved by considering verifiability as a design parameter early on in the EDA process.

The remainder of this paper is structured as follows. Section II introduces PFV in greater detail, reviewing previous works and findings. Section III showcases how DfPFV can be employed to design a circuit realizing symmetric functions with both desirable circuit properties and efficient, full-coverage verification. Finally, Section IV gives an outlook on how DfPFV may be applied in the future and which research questions this work opens up.

II. Polynomial Formal Verification

The goal of PFV is the development of formal verification methodologies for which polynomial time and space complexity bounds can be ensured on specific classes of circuits [3]. Most works on PFV use *Binary Decision Diagrams (BDDs)* [8] as canonical Boolean function representations to conduct the verification [9] and prove these bounds [10]. Here, the complete construction process of a BDD for a given circuit is important, as final BDD sizes depend only on the realized function, not the circuit implementation itself. To measure the complexity of the construction, metrics like the number of *If-Then-Else (ITE)* [11] calls and peak sizes during construction are usually considered with the help of BDD packages [12] [13].

Many different circuits have been studied in the context of PFV so far. The deepest investigations have been done into adder circuits [14], where, among others, polynomial bounds have been proven for prefix adders [15], floating point adders [16] and approximate adders [17]. In addition, more complex circuits like a RISC-V processor [18], multipliers [19] and sequential circuits [20] have been studied. Besides BDDs, polynomial bounds have also been established for different circuits using *Answer-Set Programming (ASP)* [21] [22] and *Boolean Satisfiability (SAT)* [23] [24].

Especially for integer adders, reviewing the different bounds and runtimes established on the verification process shows that they in part differ greatly depending on the implementation, even though all circuits realize the same function. For instance, prefix adders like the *Kogge-Stone Adder (KSA)* took over $300s$ to verify for $n = 10240$ inputs [15], while for simpler circuits like the *Ripple-Carry Adder (RCA)*, the verification took around $200s$ for the same size [25]. This indicates that design choices do indeed impact verifiability and a deeper investigation is worthwhile.

III. Considering Polynomial Formal Verification in the Design Process

To understand the impact of design choices on PFV better, we consider totally symmetric functions which can be implemented as digital circuits in various ways. First, we briefly introduce these functions, as well as previous works on their circuit realization. Afterward, we consider how three different implementations affect not only the circuit metrics gate count and depth but also the verifiability. To compare the verifiability, we construct the BDD of each implementation for different sizes with the CUDD package [12] to investigate how the verification complexity metrics runtime and BDD peak size are affected.

A. Symmetric Circuits

A function over n variables is called totally symmetric if it is invariant under all permutations of its inputs. For digital circuits implementing these symmetric functions, this means that their outputs depend only on the number of inputs set to 1, not on their order. These symmetric circuits find applications in different domains such as cryptography [26].

In the past, several different implementations of symmetric circuits have been proposed, trying to optimize for different metrics. The authors of [27] propose an implementation which can ensure complete path-delay fault testability. The authors of [28] optimize this design further for depth, trading off a higher gate count for reduced circuit depth.

Both these works divide their implementation into multiple parts, where the so-called Module(n) is the central component. This Module(n) has n inputs and n outputs, where the outputs represent all unate functions u_1 through u_n over n variables. A unate function u_i evaluates to 1 if at least i inputs of Module(n) are set. From these unate functions, all symmetric functions can be constructed trivially, which is why in the following we only consider this Module(n) of all unate functions and refer to its implementations as symmetric circuits.

B. Verifiability of Depth-Optimized Implementation

First, we investigate the verifiability of a generalization of Module(n) based on the method previously proposed in [27] and optimized for depth in [28]. The method implements Module(n) within a three-stage approach that is based on sorting networks that are recursively called. The first stage consists of AND-OR cells that are arranged within a shuffle-exchange network, followed by recursive calls to Module(n) in the second stage, and a final third stage resulting in the unate symmetric functions.

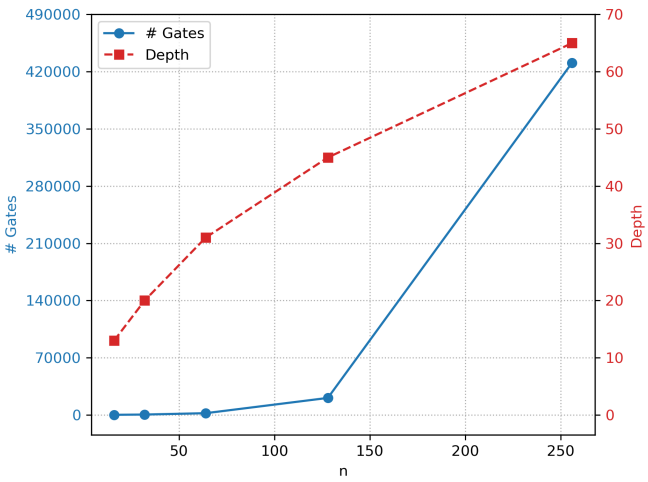

Fig. 1: Circuit metrics for depth-optimized implementation

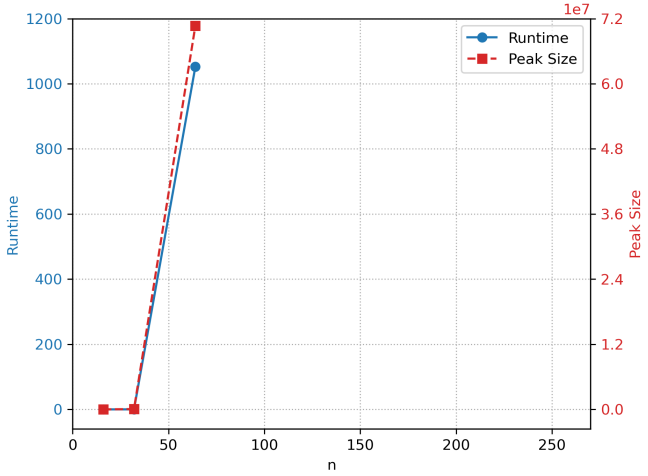

Fig. 2: Verification complexity for depth-optimized implementation

Figure 1 shows the gate count and depth for a re-implementation and generalization of $Module(n)$ presented in [28]. The number of gates starts relatively low but explodes to over 400,000 for $n = 256$, showing that the implementation is not optimized for gate count. Regarding the depth, we see the asymptotically logarithmic behavior.

The verification complexity of this implementation is displayed in Figure 2. For up to $n = 32$, verification is very efficient, until the runtime and peak size suddenly explode for $n = 64$. A timeout was reached starting from $n = 128$ inputs, showing that formal verification was infeasible for this depth-optimized implementation.

C. Taking a Step Back

These numbers show that optimizing for circuit metrics without considering verifiability potentially impedes polyno-

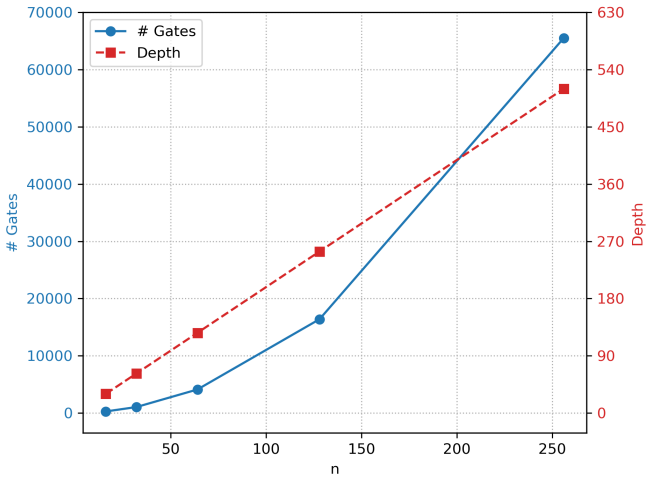

Fig. 3: Circuit metrics for DSTL implementation

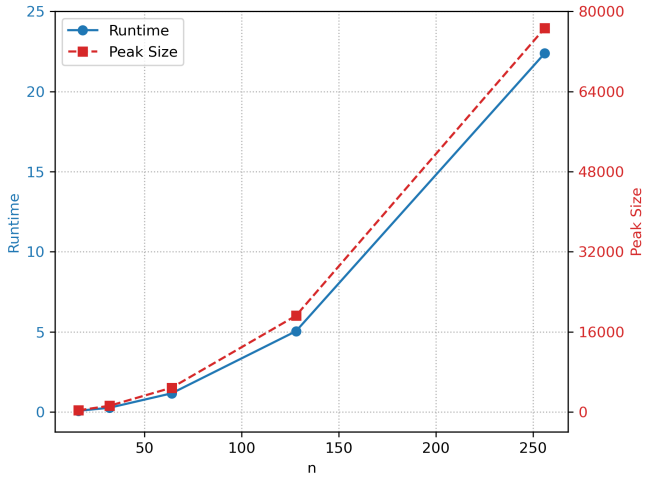

Fig. 4: Verification complexity for DSTL implementation

mial verification. In order to combat this, we implement a simpler circuit design, as previous work on PFV has shown that structurally less complex designs have a better verifiability. To this end, we consider a *Digital Summation Threshold Logic (DSTL)* array [29]. This array consists of DSTL cells, which include an AND gate and an OR gate. The cells are arranged within interconnected rows, where each row computes a unate symmetric function. The array structure of the simple DSTL cells results in a regular and simple circuit architecture.

The circuit metrics for the DSTL symmetric circuit implementation are displayed in Figure 3. The gate count scales more slowly than for the depth-optimized version, peaking below 70,000, while depth behaves linearly.

Figure 4 shows the resulting verification complexity. In contrast to the depth-optimized implementation, both the verification runtime and the BDD peak size during the verification of the simpler DSTL array scale well with the number of inputs, thereby enabling an efficient formal verification.

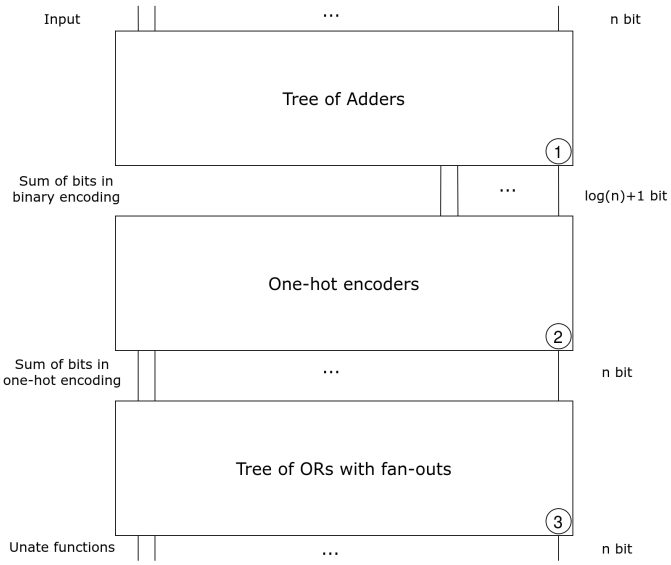

Fig. 5: Circuit schematic for DfPFV approach

D. Design for Polynomial Formal Verification

While the simpler implementation of a symmetric circuit led to improved verifiability, having linear depth is not a desired property of our symmetric circuit, as it increases the delay of the function computation. Thus, we now take a DfPFV approach and consider efficient verifiability as one additional design parameter, continuing to optimize for gate count and especially depth. From our previous knowledge on PFV, we know that adder circuits can be verified efficiently, giving us an idea of how to optimize for the verifiability design parameter. Symmetric functions can also be viewed as the addition of all input bits, i.e., unate function u_i is 1 if the sum over all input bits is at least i, due to the commutative property of addition.

Figure 5 shows the schematic for the symmetric circuit obtained by a DfPFV approach. It is divided into three stages: Stage ① consists of a tree of adder circuits to sum up all inputs. It starts with a layer of $n/2$ 1-bit adders (simple half adders) and ends with a layer consisting of a single $log(n)$-bit adder. To optimize for depth, we use the KSA for all addition operations, resulting in a logarithmic number of layers with a depth of at most $log(log(n))$. The output of this stage is the sum of input bits in binary representation, ranging from 0 to n. Stage ② consists of n one-hot encoders [30] of logarithmic depth, which change the representation of this sum to one-hot encoding. This has the effect that at the output of this stage, exactly one bit is set, i.e., only bit i is set if exactly i inputs are 1. Then, we can compute the unate functions u_i using all previously generated functions that set exactly j inputs to 1 with $j >= i$. For this, stage ③ consists of a logarithmic tree of OR gates with fan-outs for different unate function outputs. The entire tree computes u_1, while the remaining unate functions are correctly constructed via fan-outs, e.g., u_{n-1} is fanned out from the OR gate of stage inputs n and $n-1$. Fan-outs are added to avoid requiring an entire OR tree

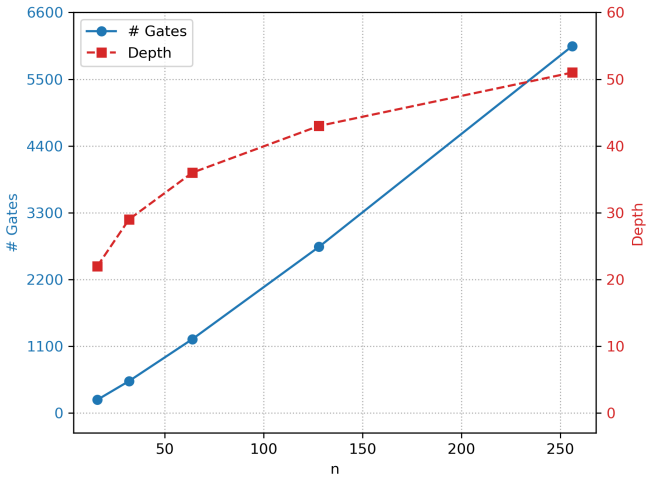

Fig. 6: Circuit metrics for DfPFV approach

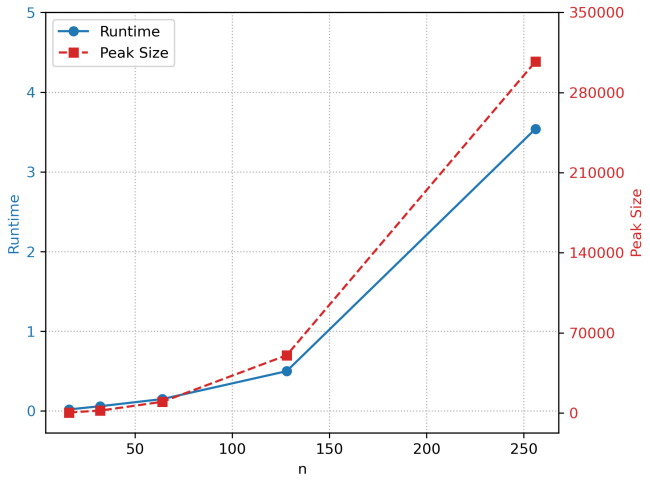

Fig. 7: Verification complexity for DfPFV approach

for each unate output, which has an overall positive effect on the gate count.

The metrics achieved by the circuit implementation of the DfPFV approach are shown in Figure 6. Asymptotically, the depth behaves the same as for the depth-optimized implementation shown in Figure 1, exhibiting a logarithmic behavior in n. At the same time, the gate count also scales better, showing a near linear behavior. This confirms that considering different optimization targets in a DfPFV approach still enables us to reach desired properties of certain circuit metrics, even providing a better scalability of the gate count.

Unlike the depth-optimized implementation based on [28], where verification fails for larger values of n, as displayed in Figure 2, the circuit implementation produced by the DfPFV approach is verifiable even for large values of n, as can be seen in Figure 7. For all considered values of n, the verification runtime remains below four seconds, with a peak BDD size of around 300,000 BDD nodes. Asymptotically, both of these verifiability indicators seem to behave in a polynomial fashion. This demonstrates that the DfPFV approach is able to account for different optimization targets in the design process, achieving a near linear gate count and logarithmic depth while at the same time ensuring efficient verifiability.

Table I compares the three implementations on all considered design parameters. In terms of gate count, it can be seen that our proposed DfPFV implementation is the most efficient starting from $n = 64$, whereas the depth-optimized implementation has a lower gate count for $n = 16$ and $n = 32$. Regarding depth, the implementation based on [28] once again achieves better numbers for lower values of n, which aligns with its optimization goal. Starting from $n = 128$ however, the depth of the DfPFV implementation even exhibits the lowest depth, scaling better than the depth-optimized implementation, despite being primarily optimized for verifiability. Looking at the verifiability of the three implementations, the two complexity metrics need to be differentiated. The DSTL implementation has the lowest BDD peak size across all sizes. However, this does not translate to the most efficient verification regarding runtime, indicating that the high gate count has an impact on verification complexity. In terms of runtime, as shown in Figure 2, the verification complexity of the depth-optimized implementation explodes quickly, reaching a timeout from $n = 128$. While it still shares the best runtime for $n = 16$, the proposed DfPFV implementation exhibits the most efficient verifiability w.r.t. runtime in our evaluation. Overall, these results show the upside of our proposed DfPFV implementation, which performs best in terms of verifiability, while at the same time optimizing for gate count and depth, especially for higher values of n.

IV. Looking Forward

In this work, we showed how verifiability can be considered as an additional parameter in the design process. On the case-study of symmetric functions, we demonstrated how design choices impact formal verification, as previous implementations optimized for depth made verification for input sizes starting from $n = 128$ infeasible. Our proposed DfPFV approach led to a circuit implementation that is not only easier to verify but also improves other design parameters, such as gate count and depth, for larger sizes.

While our evaluation constitutes the first time that the effect of considering verifiability early on in the EDA process has been studied, previous work on PFV suggests that design choices do indeed impact formal verification on several circuit classes. Accordingly, we recommend a deeper look at the concept of DfPFV going forward. As a first step, evaluations like ours could be extended to further circuit classes to establish the wider impact that DfPFV can have. Next, the concrete workflow of DfPFV could be discussed, building on decades of experience in the EDA domain. Finally, the integration into existing optimization algorithms and tooling may be explored to ensure a wider adoption of the DfPFV approach. Together, these future research directions may not only improve the formal verifiability of designs, but lead to the discovery of further optimization in other dimensions.

TABLE I: Comparison of the three implementations

n	Depth-Optimized				Simple				DfPFV			
	Metrics		Verifiability		Metrics		Verifiability		Metrics		Verifiability	
	# Gates	Depth	Runtime [s]	Peak Size	# Gates	Depth	Runtime [s]	Peak Size	# Gates	Depth	Runtime [s]	Peak Size
16	**126**	**13**	**0.02**	881	256	30	0.07	**313**	217	22	**0.02**	411
32	**448**	**20**	0.09	55899	1024	62	0.26	**1222**	525	29	**0.06**	2010
64	2104	**31**	1052.27	70681575	4096	126	1.16	**4833**	**1216**	36	**0.15**	9710
128	20792	45	T.O.	T.O.	16384	254	5.04	**19222**	**2737**	**43**	**0.50**	50392
256	430932	65	T.O.	T.O.	65536	510	22.39	**76673**	**6046**	**51**	**3.54**	307043

Ideally, this leaves us in a future with robust designs that are developed in an efficient manner, creating a positive impact on both the process and the end product.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project *PolyVer* (DR 287/36-1).

REFERENCES

[1] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, p. 82–85, Jan. 1998.

[2] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and trends in modern SoC design verification," *IEEE Design & Test*, vol. 34, p. 7–22, Oct. 2017.

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

[4] A. Mahzoon and R. Drechsler, "Polynomial formal verification of area-efficient and fast adders," in *2021 Reed Mulller Workshop (RM2021)*, 2021.

[5] T. W. Williams and K. P. Parker, "Design for testability—a survey," *Proceedings of the IEEE*, vol. 71, no. 1, pp. 98–112, 2005.

[6] A. Francis, P. Maropoulos, G. Mullineux, and P. Keogh, "Design for verification," *Procedia CIRP*, vol. 56, pp. 61–66, 2016. The 9th International Conference on Digital Enterprise Technology – Intelligent Manufacturing in the Knowledge Economy Era.

[7] C.-N. J. Liu, I.-L. Chen, and J.-Y. Jou, "An efficient design-for-verification technique for hdls," in *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, ASP-DAC '01, (New York, NY, USA), p. 103–108, Association for Computing Machinery, 2001.

[8] Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[9] A. Hu, "Formal hardware verification with BDDs: an introduction," in *1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997*, vol. 2, pp. 677–682 vol.2, 1997.

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

[11] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Conference proceedings on 27th ACM/IEEE design automation conference - DAC '90*, (Orlando, Florida, United States), p. 40–45, ACM Press, 1990.

[12] "Cudd decision diagram package." https://github.com/cuddorg/cudd. accessed 25-19-11.

[13] R. Krauss, J. Zielasko, and R. Drechsler, "FrEDDY: Modular and efficient framework to engineer decision diagrams yourself," in *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–2, 2025.

[14] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, (Vienna, Austria), p. 99–104, IEEE, Apr. 2021.

[15] A. Mahzoon and R. Drechsler, "Polynomial formal verification of prefix adders," in *2021 IEEE 30th Asian Test Symposium (ATS)*, (Matsuyama, Ehime, Japan), p. 85–90, IEEE, Nov. 2021.

[16] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Polynomial formal verification of floating point adders," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (Antwerp, Belgium), p. 1–2, IEEE, Apr. 2023.

[17] M. Schnieber, S. Froehlich, and R. Drechsler, "Polynomial formal verification of approximate adders," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, (Maspalomas, Spain), p. 761–768, IEEE, Aug. 2022.

[18] L. Weingarten, K. Datta, and R. Drechsler, "Polynomial formal verification of a RISC-V processor," *IEEE Transactions on Nanotechnology*, vol. 24, p. 140–151, 2025.

[19] R. Drechsler, A. Mahzoon, and M. Goli, "Towards polynomial formal verification of complex arithmetic circuits," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, (Prague, Czech Republic), p. 1–6, IEEE, Apr. 2022.

[20] C. Dominik and R. Drechsler, "Polynomial formal verification of sequential circuits," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (Valencia, Spain), p. 1–6, IEEE, Mar. 2024.

[21] M. Nadeem, J. Kleinekathofer, and R. Drechsler, "Polynomial formal verification exploiting constant cutwidth," *34th International Workshop on Rapid System Prototyping (RSP)*, 2023.

[22] M. Nadeem, C. K. Jha, and R. Drechsler, "Polynomial formal verification of sequential circuits using weighted-AIGs," in *2025 Design, Automation & Test in Europe Conference (DATE)*, (Lyon, France), p. 1–7, IEEE, Mar. 2025.

[23] L. Müller and R. Drechsler, "SAT can ensure polynomial bounds for the verification of circuits with limited cutwidth," in *2024 27th Euromicro Conference on Digital System Design (DSD)*, (Paris, France), p. 57–64, IEEE, Aug. 2024.

[24] L. Müller and R. Drechsler, "Polynomial formal verification parameterized by cutwidth properties of a circuit using Boolean satisfiability," *Microprocessors and Microsystems*, vol. 118, p. 105199, Nov. 2025.

[25] A. Mahzoon and R. Drechsler, "Polynomial formal verification of area-efficient and fast adders," 2021.

[26] S. Basu, M. Kule, and H. Rahaman, "Implementation of symmetric functions using memristive nanocrossbar arrays and their application in cryptography," *J. Circuits Syst. Comput.*, vol. 30, pp. 2150223:1–2150223:13, 2021.

[27] H. Rahaman and D. K. Das, *A Simple Delay Testable Synthesis of Symmetric Functions*, vol. 3285 of *Lecture Notes in Computer Science*, p. 263–270. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[28] M. Schnieber, S. Froehlich, and R. Drechsler, "Depth optimized synthesis of symmetric Boolean functions," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, (Tampa, FL, USA), p. 61–66, IEEE, 2021.

[29] S. Hurst, "Digital-summation threshold-logic gates: a new circuit element," *Proceedings of the Institution of Electrical Engineers*, vol. 120, pp. 1301–1307, 1973.

[30] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*. Elsevier, 2013.