Late Breaking Results: PolyRAD - Polynomial Formal Verification of Restoring Array Dividers

Mohamed Nadeem[Ш], Chandan Kumar Jha[Ш], Rolf Drechsler[Ш,†]
University of Bremen, Bremen, Germany[Ш]
DFKI GmbH, Bremen, Germany[†]
mnadeem@uni-bremen.de, chajha@uni-bremen.de, drechsler@uni-bremen.de

***Abstract*—Formally verifying divider circuits is complex, and multiple effective methods have been developed. However, none of these methods provides an upper bound on the verification time, which limits their scalability for large divider circuits. Recently, *Polynomial Formal Verification (PFV)* based approaches have been investigated to ensure circuit correctness in polynomial time and space. However, there is no PFV based approach for the formal verification of dividers. In this paper, we introduce for the first time a two-level partitioning strategy and present PolyRAD, a novel PFV approach for verifying *Restoring Array Divider (RAD)*. Finally, we prove that verification of RAD can be achieved in polynomial time, and conduct experimental evaluation on RAD of different sizes to validate our theoretical findings.**

***Index Terms*—Polynomial Formal Verification, Circuit Partitioning, Restoring Array Dividers.**

I. Introduction

As circuit designs become increasingly complex, ensuring correctness is crucial, underscoring the need for formal verification of arithmetic circuits, such as division. Several methods have been introduced for the verification of *Restoring Array Divider (RAD)* [1]–[6]. However, they fail to provide an upper bound on verification time, which can make verification unpredictable and resource-intensive, especially for large RAD sizes. Similar challenges occur in verifying other arithmetic circuits. Therefore, *Polynomial Formal Verification (PFV)* [7], [8] is a promising direction to ensure polynomial upper bounds on time and space for verification. PFV has been employed with *Circuit Partitioning* [9], [10] (i.e., *Cutwidth Partitioning*) to provide polynomial-time verification approaches for several types of simple arithmetic circuits (e.g., adders). However, for complex circuits such as dividers, we show that PFV can be applied to achieve formal verification in polynomial time.

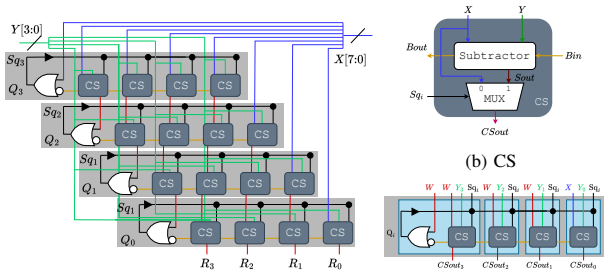
- We introduce, for the first time, a divide-and-conquer partitioning strategy for RAD circuits and propose *PolyRAD*, a novel PFV approach for RAD based on this strategy.
- We prove that RAD can be verified in polynomial time and conduct an experimental evaluation to confirm our findings and demonstrate the scalability of our approach.

II. PolyRAD Approach

A. Restoring Array Divider

RAD takes a $2N$-bit dividend X and an N-bit divisor Y and produces an N-bit quotient Q and remainder R. The division is performed using a sequence of *Controlled Subtractor (CS)*. In the first stage, Y is subtracted from the N most significant bits of X; in subsequent stages, it is subtracted from the N most significant bits of the partial remainder, with Y shifted right by one bit after each step. The sign of each subtraction determines the quotient bit Q_i and is forwarded to the next stage as the control signal Sq_i. If the partial remainder becomes negative, the subtractor restores it by adding Y back, and this sign is propagated as the control signal to the next stage. The overview of $8/4$ RAD can be seen in Fig. 1(a), where the dividend is 8-bits, and the divisor is 4-bits.

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project *PolyVer* (DR 287/36-1) and by the Data Science Center of the University of Bremen (DSC@UB) funded by the State of Bremen.



(a) RAD Horizontal Partitioning (c) Vertical Partitioning

Fig. 1: Overview of the RAD and its CS, together with the partitioning strategy.

B. Circuit Partitioning

To ensure that our partitioning can be applied to any RAD architecture (i.e., simple modular design (*Clean*) or highly optimized and merged design (*Dirty*)) without requiring knowledge of its internal structure, we propose two partitioning strategies that use only the RAD 's inputs and outputs, where the circuit is represented as an *And-Inverter Graph (AIG)* [11].

1) ***Horizontal Partitioning****:* In this step, the RAD is partitioned horizontally by iteratively selecting each Q_i (from Q_N to Q_0) and traversing its fan-in until reaching either the primary inputs Sq_i, X, and Y, or an intermediate gate (called *Intermediate Input*) that was already visited in a previous subcircuit (i.e., previous stage). The intermediate input has to represent one of the following cases:

(Clean): It is the previous stage's remainder and thus serves as an intermediate input to the CS in the current stage.

(Dirty): It is a sub-function of the previous stage's remainder, which is combined with the current stage's gates to compute the remainder.

We refer to the intermediate gate passed to the next quotient subcircuits as *Intermediate Output*. To ensure that the Intermediate Output preserves only one of the previous cases, it must correspond to a *Cone Gate*—i.e., the sub-function (fan-in) that can be traversed from the subcircuit inputs and passed to the next quotient subcircuits. Given the $8/4$ RAD, 4 quotient subcircuits (highlighted in gray) are obtained as shown in Fig. 1(a).

2) ***Vertical Partitioning****:* In this step, each quotient subcircuit Q_i is partitioned vertically into cell subcircuits, where each cell subcircuit starts from its primary inputs Sq_i, X (if it exists), and Y, and traverses its output gates until reaching the cone nodes. At this stage, the cell subcircuit may contain intermediate inputs coming from either the previous quotient

subcircuit or the previous cell subcircuit. The intermediate gate can be added if it is a fan-in of a gate whose other fan-in is reachable from the cell primary inputs. In this case, the intermediate input represents one of the following:

(Quotient intermediate): It corresponds to a remainder from the previous quotient.

(Cell intermediate): It corresponds to the input borrow or a sub-function of the input borrow used by the current CS.

Given a quotient subcircuit Q_i, 4 cell subcircuits (highlighted in light blue) are obtained as shown in Fig. 1(c). These two partitioning steps allow partitioning any RAD circuit without having information on its internal structure.

C. PolyRAD Verification Approach

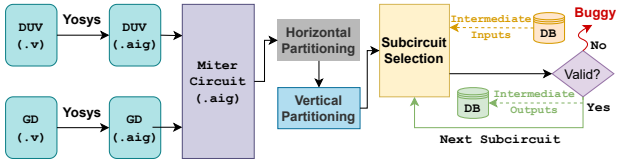


Fig. 2: PolyRAD Verification Approach.

The overview of PolyRAD is shown in Fig. 2. The approach takes two designs, the *Design Under Verification (DUV)* and the *Golden Design (GD)*, and constructs a miter circuit in AIG format. Horizontal partitioning is then applied, followed by vertical partitioning. Next, the cell subcircuits are verified iteratively. If a cell is correct, its intermediate outputs are stored in a database and used as intermediate inputs for subsequent cell subcircuits. Otherwise, it is buggy (for details, see [9]).

Theorem II.1 (PolyRAD Complexity)**.** *PolyRAD has time complexity* $\mathcal{O}(G + (\#C \times 2^{CW}))$*, where* G *is the gate count,* $\#C$ *is the number of cell subcircuits, and* CW *is the maximum number of inputs among them.*

Proof: The horizontal partitioning (recall Section II-B1) is employed such that each subcircuit starts from a quotient Q_i and traverses the input gates towards the primary inputs. To ensure that the intermediate gates (i.e., intermediate inputs and intermediate outputs) satisfy one of the cases (i.e., *Clean* and *Dirty*), the subcircuit is traversed w.r.t. the current subcircuit observed inputs to obtain the least number of cone gates (recall Section II-B1) required to be passed to the next subcircuit. In this step, it is required to check the edges (i.e., input wires) of each gate. Since in the AIG, the *AND* gate is restricted to have only two inputs, the number of edges can be, in the worst case, equal to $2 \times G$, where G is the number of gates appearing in the circuit. Therefore, the complexity of horizontal partitioning is $Complexity(HP) = \mathcal{O}(G+2G) = \mathcal{O}(G)$. Let $C_N, \ldots, C_0$ be the resulting subcircuits. Each C_i is then partitioned using vertical partitioning (recall Section II-B2) w.r.t. the inputs Sq_i, X_i, D_i such that each cell subcircuit starts from the selected inputs toward the outputs until reaching either a cone gate or the quotient Q_i. This requires traversing the edges within the subcircuit C_i (the number of edges is bounded by $2 \times G$). Hence, vertical partitioning has complexity of $\mathcal{O}(G)$ to partition all subcircuits $C_N, \ldots, C_0$. Consequently, the overall complexity of the two-level partitioning is characterized such that $\mathcal{O}(DT) = \mathcal{O}(G + G) = \mathcal{O}(G)$. Let $CS_M, \ldots, CS_0$ be the resulting cell subcircuits across all subcircuits $C_N, \ldots, C_0$. Since each CS_j represents a CS and its functional dependency, the overall number of subcircuits $\#C$ is bounded by the number of CS in the RAD (i.e., $\#C = M = N \times N = N^2$, where N is the size of the divisor Y). The verification is performed iteratively over $CS_M, \ldots, CS_0$ and starting from CS_0 of the first CS appearing in the quotient Q_N toward the last subcircuit $CS_{\#C}$ of Q_0. We refer by IN_j to the inputs (i.e., primary or intermediate) appearing in the subcircuit CS_j. The complexity of PolyRAD approach can be characterized such that $\mathcal{O}(PolyRAD) = \mathcal{O}(DT + \sum_{\#C}^{j}(2^{IN_j})) = \mathcal{O}(G + \#C \times 2^{CW})$, where $CW = \max(|IN_0|, \ldots, |IN_{\#C}|)$. ■

TABLE I: Comparison Between PolyRAD and Cadical approaches

Set	Circuit	PolyRAD						Cadical	
		#C	G	CW	DT	Time	Memory	Time	Memory
S1	8/4	16	147	6	0.02	1.45	1.99	0.0920129	0.0
	16/8	64	571	7	0.09	2.89	2.01	0.6416	0.07
	32/16	256	2291	8	0.63	9.42	2.04	4.36156	0.09
	64/32	1024	9187	8	5.38	38.52	2.2	389.82	0.18
	128/64	4096	36803	8	57.23	204.11	2.83	T.O.	T.O.
	256/128	16384	147331	8	742.55	1544.35	5.58	T.O.	T.O.
S2	8/4	16	142	7	0.01	1.33	1.99	0.0814059	0.0
	16/8	64	564	8	0.08	2.99	2.01	0.754663	0.07
	32/16	256	2276	8	0.63	9.04	2.04	5.69457	0.09
	64/32	1024	9156	8	5.5	38.66	2.19	336.655	0.18
	128/64	4096	36740	8	63.36	228.52	2.84	T.O.	T.O.
	256/128	16384	147332	8	740.94	1545.88	5.6	T.O.	T.O.
S3	8/4	16	148	7	0.02	1.35	1.99	0.089882	0.0
	16/8	64	572	8	0.08	2.75	2.01	0.611331	0.07
	32/16	256	2292	8	0.68	10.39	2.04	10.9393	0.09
	64/32	1024	9188	8	6.04	43.43	2.19	342.306	0.18
	128/64	4096	36804	8	56.6	204.58	2.84	T.O.	T.O.
	256/128	16384	147332	8	738.81	1544.81	5.61	T.O.	T.O.

III. Experimental Results

We have implemented *PolyRAD* in Python. We compare our approach with the modern-SAT approach *Cadical* [12] in terms of time and memory consumption. All experiments were performed on an Intel(R) Core(TM) i7-11370 with 3.30 GHz. We set a timeout of 3600 seconds and limited available RAM to 16 GB per instance.

We verify RAD circuits of different sizes, up to $256/128$. To demonstrate that our approach can be applied to any RAD architecture, we generated three sets of DUV instances, each produced using a different sequence of *ABC* [13] optimization commands. A clean design is used as the GD. Both the DUV and GD are synthesized into AIG format using *Yosys* [14]. The optimization command sets are summarized as follows:

(S1): "`strash;refactor;resyn2`".
(S2): "`strash;refactor;resyn;resyn2;resyn3`".
(S2): "`strash;refactor;resyn;resyn2`".

Table I shows the results of PolyRAD and *Cadical* in terms of wall time (seconds) and memory consumption (GB). Our approach successfully verifies RAD circuits of different sizes and under different optimization commands within the time limit. In contrast, *Cadical* exceeds the time limit starting from the $128/64$ RAD. We can see that CW remains constant across all circuit sizes. Also, G scales quadratically w.r.t. the size N (i.e., $G \leq N^2$). Therefore, the overall complexity of RAD can be simplified to $\mathcal{O}(N^2 + \#C \times 2^{CW}) = \mathcal{O}(N^2 + N^2) = \mathcal{O}(N^2)$. Hence, the verification time of the PolyRAD scales quadratically w.r.t. the size N, aligning with the results in Table I. Hence, RAD circuits belong to the PFV class of circuits that can be verified in polynomial time (Theorem II.1).

IV. Conclusion

We introduced PolyRAD, a novel PFV of RAD based on a two-level partitioning strategy that requires no internal structural knowledge. We showed that RAD can be verified in polynomial time and demonstrated PolyRAD's scalability on circuits up to $256/128$. As future work, we plan to extend our verification algorithm to other arithmetic circuits.

REFERENCES

[1] C. Scholl, A. Konrad, A. Mahzoon, D. Große, and R. Drechsler, "Verifying dividers using symbolic computer algebra and don't care optimization," in *Design, Automation and Test in Europe*, pp. 1110–1115, 2021.

[2] A. Konrad, C. Scholl, A. Mahzoon, D. Große, and R. Drechsler, "Divider verification using symbolic computer algebra and delayed don't care optimization: theory and practical implementation," *Formal Methods in System Design*, pp. 1–37, 2024.

[3] M. H. Haghbayan and B. Alizadeh, "A dynamic specification to automatically debug and correct various divider circuits," *Integration*, vol. 53, pp. 100–114, 2016.

[4] J. Dasari and M. Ciesielski, "Efficient formal verification and debugging of arithmetic divider circuits," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023.

[5] A. Yasin, T. Su, S. Pillement, and M. Ciesielski, "Functional verification of hardware dividers using algebraic model," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 257–262, 2019.

[6] M. H. Haghbayan and B. Alizadeh, "A dynamic specification to automatically debug and correct various divider circuits," *Integr. VLSI J.*, vol. 53, p. 100–114, Mar. 2016.

[7] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *International Conference on Computer-Aided Design*, pp. 70:1–70:9, 2022.

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

[9] M. Nadeem, J. Kleinekathofer, and R. Drechsler, "Polynomial formal verification exploiting constant cutwidth," in *IEEE International Workshop on Rapid System Prototyping*, Association for Computing Machinery, 2024.

[10] M. Nadeem, C. K. Jha, and R. Drechsler, "Polynomial formal verification of sequential circuits using weighted-AIGs," in *Design, Automation and Test in Europe*, 2025.

[11] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *Design Automation Conference*, pp. 532–535, 2006.

[12] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "Cadical 2.0," in *Computer Aided Verification: 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24–27, 2024, Proceedings, Part I*, (Berlin, Heidelberg), p. 133–152, Springer-Verlag, 2024.

[13] "Abc: A system for sequential synthesis and verification." available at https://people.eecs.berkeley.edu/~alanmi/abc/, 2018.

[14] C. Wolf, "Yosys open synthesis suite." https://yosyshq.net/yosys/.