2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD) | 979-8-3503-3265-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/SMACD58065.2023.10192137

Formal Verification of Divider Circuits by Hardware Reduction

Atif Yasin*, Tiankai Su*, Sébastien Pillement†, Maciej Ciesielski*
*University of Massachusetts, ECE Department, Amherst, MA, USA
†Nantes Université, CNRS, IETR UMR 6164, Nantes, France
atifyasin2@gmail.com, tiankaisu@gmail.com, sebastien.pillement@univ-nantes.fr, ciesiel@umass.edu

***Abstract*—The paper introduces a novel verification method of gate-level hardware implementation of divider circuits. The method, called *hardware reduction*, accomplishes the verification by appending the divider circuit with another circuit, which implements its arithmetic inverse, followed by logic synthesis. If the circuit under verification is correct, the resulting re-synthesized circuit becomes trivially redundant (composed of wires or buffers only). This method outperforms the established Boolean satisfiability, SAT-based and equivalence checking techniques and does not require a reference design.**

I. Introduction

Considerable progress has been made in recent years in verification of arithmetic circuits, such as multipliers, multiply-accumulate circuits, and other components of arithmetic datapaths [1][2][3]. However, verification of gate-level divider circuits has received only a limited attention [4][5][6]. Division plays a major role in many domains, including computer arithmetic, computational geometry, cryptography, and many other applications, and verification of hardware implementation of divider circuits is of prime importance.

This work addresses this need and offers a new technique called *hardware reduction*. The technique is based on appending the arithmetic circuit under verification with a circuit that computes its inverse; the resulting circuit is then synthesized and checked if it becomes redundant, i.e., is reduced to identity. The results show that despite the increase in gate count of such constructed circuit, this verification method offers a good alternative to the standard Boolean satisfiability (SAT)-based and combinational equivalence checking (CEC) techniques.

The rest of the paper is organized as follows. Section II provides the necessary background and a brief review of the related work in this field. Section III describes the details of hardware reduction applied to dividers. Finally, Sections IV and V present the results and conclusions.

II. Background and Related Work

An established verification technique often employed in industry is Theorem Proving. Theorem provers are inductive reasoning systems that use mathematical models to verify functional correctness of the design. They rely on a carefully constructed set of rewriting rules and complex formulas to represent the circuit and require an in-depth, domain-specific user knowledge of the design and the system [7][8]. The success of the proof relies on the choice of the rules and on the order in which they are applied to the system, with no guarantee of a successful conclusion. They are typically used to prove the correctness of larger systems and CPUs and their architecture, rather than of the low-level hardware implementation.

Some approaches rely on reverse-engineering by extracting logic gates of the circuit (such as XORs and carry chains) and comparing them against the expected elements of the divider using structural matching [4]. These methods however are customized for the particular hardware structure which may not always be available.

A more general approach, offered by formal verification, employs various canonical diagrams, such as BDDs [9] and *BMDs [10]. These techniques however need to transform the circuit into bit-level netlists, known as "bit-blasting", and as such are ineffective for arithmetic circuits with large bit-width operands. The state-of-the-art formal methods resort to SAT-based approach and equivalence checking. Some of those methods have been used in proving correctness of the SRT divider [11], but could handle only a single gate-level stage of the divider.

An approach to formal verification of arithmetic circuits that emerged in recent years has been based on symbolic computer algebra (SCA). In this approach, an arithmetic circuit is represented in an algebraic, rather than Boolean, domain. The specification of an arithmetic circuit and its implementation are represented as polynomials. The verification problem is then to check if the implementation satisfies the specification using a canonical polynomial representation, called Gröbner basis [12]. This method is known in computer algebra lexicon as *ideal membership testing* [13]. Several modifications have been reported in the literature to improve the efficiency of the technique for the verification of integer multipliers [2][3]. However, those techniques have not been successful in verifying gate-level divider circuits due to their high demand for memory.

An alternative approach to arithmetic verification, derived from SCA and originally proposed in [14], uses a technique of *algebraic rewriting*. With this approach, the polynomial representing encoding of the primary outputs (called output signature) is transformed by backward rewriting into a polynomial expressed in terms of the primary inputs (the input signature) using algebraic models of internal logic gates. The resulting signature is then checked against the specification polynomial of the circuit. The method, in fact, performs *function extraction* as it derives an arithmetic function of the circuit from its gate-level implementation. This method has been successfully used to verify complex adders and large multipliers [1][2][3].

However algebraic rewriting has its disadvantages, mostly due to a large number of the so called *vanishing monomials* accumulating during the rewriting, while eventually reducing to zero. This phenomenon is particularly severe in dividers, where a straightforward algebraic rewriting produces a pro-

hibitive number of such monomials. This problem has been recognized in [6]; it employs SAT to gain information about vanishing monomials by propagating logic information in the direction opposite to standard rewriting, i.e., from inputs to outputs. This helps determine the invariants, combinations of internal signal values that do not occur in the circuit due to the imposed constraint on the size of dividend X vs divisor D: $X < 2^{n-1}D$. These signals appear at the inputs to the atomic blocks (adders), extracted using reverse engineering. The invariants are then modeled as satisfiability don't cares and an integer linear programming (ILP) method is used to optimize the polynomials modulo those don't cares. While this approach makes it possible to handle large designs, it is quite complex and require extended CPU times to complete.

III. Divider Verification

Figure 1 shows a typical architecture of a restoring integer divider. The inputs are unsigned integers: X is the dividend, D the divisor, Q the quotient, and R the remainder. To reduce the circuit size and for practical considerations (the fact that $X = Q \cdot D + R$), the operands D, Q, R are n-bit wide and the dividend has $2n$-1 bits. Furthermore, to guarantee that the resulting quotient Q will not overflow, a condition $X < 2^{n-1}D$ is imposed on the inputs [15][6]. With these constraints, the optimized divider is implemented in an $n \times (2n-1)$ array structure, composed of n layers, each being n-bit wide, as shown in the Figure.

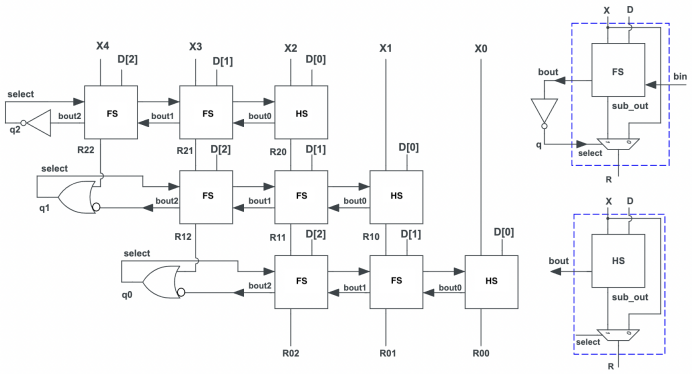

Fig. 1: Restoring integer divider for $n = 3$.

The division is performed by a repeated subtraction of the dividend by a divisor D, shifted after each subtraction by one bit to the right. If the expected subtraction result (partial remainder) is positive, subtraction takes place; otherwise it is not performed at that level and the unchanged partial remainder is passed on to the next layer.

In the following, each layer i corresponding to the quotient bit q_i has inputs R_{i+1}, D and outputs q_i, R_i. The input to the top layer is R_n, part of the dividend X. The output of the bottom layer is the final remainder R. The quotient bit q_i produced as the MSB of each layer serves as a select signal, which determines whether the input vector R_{i+1} or the difference $R_{i+1} - D$ is passed to the output R_i. Hence, the equation that characterizes layer i of the divider is: $R_i = R_{i+1} - q_i D$. The verification goal is to prove that the circuit correctly performs the division under the required input constraint $X < 2^{n-1}D$.

Such a conceptually simple array structure is difficult to handle by the SCA or algebraic rewriting approach. This is dictated mostly by the dependence of the quotient q_i on the n-bit subtractor with inputs R_{i+1} and D, which causes exponential increase in the size of polynomials. This has already been noticed by authors of [6] who solve this problem using satisfiability don't cares and ILP, as explained earlier. However, their approach is computationally expensive and unnecessarily complex. In the next section we propose an alternative way to perform the verification, without the use of algebraic rewriting, which produces even better results.

A. General Model

Figure 2(a) shows an abstract model of the divider verification employed in this work. It is governed by the following expression:

$$X = Q \cdot D + R \quad \text{and} \quad 0 \leq R < D \tag{1}$$

The middle part of the diagram is the divider circuit under verification. The lower box "reverses" the division X/D by computing $Q \cdot D + R$ from the quotient Q and the remainder R, produced by the divider. The goal is to prove that the computed result matches the original dividend X, i.e. that $X = QD + R$, and the condition $0 \leq R < D$ is satisfied.

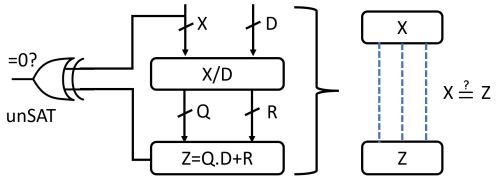

Fig. 2: Divider verification approaches: a) Divider verification model; b) solution with SAT, c) hardware reduction method.

In principle, this problem can be solved by creating a circuit $Z = Q \cdot D + R$ and checking the equivalence between its output Z and the dividend X. This can be done using standard SAT technique: create a "miter" (generalized XOR) between the dividend input X and output Z and check if the CNF[1] formula of the resulting miter circuit is unsatisfiable (unSAT). This is shown symbolically in Figure 2(b).

We tested this method on both restoring and non-restoring array dividers using the ABC system [16], with $MiniSAT$ [17] as the underlying SAT engine. While it required only a few seconds to prove the divider with 16-bit dividend X, the computation for larger instances timed out after one hour. In general, divider circuits with bit-widths greater than 16 could not be verified using this method. To address this problem, we investigated another approach, originally proposed for SQRT circuits [18], where algebraic rewriting has been replaced by logic synthesis.

B. Hardware Reduction

The main idea of the Hardware Reduction (HR) method is as follows: first, the divider circuit is appended with a circuit

[1]Conjunctive Normal Form, standard form used to encode SAT problems.

that computes $Z = Q \cdot D + R$, Figure 2(a); then, such a constructed circuit, with input X and output Z, is subjected to logic synthesis. If the divider correctly implements the division operation, the resulting circuit is trivially redundant, composed of wires/buffers connecting directly the bits of X and Z, as shown symbolically in Figure 2(c). The goal is to prove that output Z matches bit-by-bit the input X, i.e., $Z_i = X_i$ for all bits i of the dividend. However, this technique, when applied to the entire divider, is not scalable; we were only able to verify the dividers with a maximum bit-width of 20 due to excessive CPU time (but without a memory overload).

C. Layered Approach

To solve the problem, we investigated a *layered* approach, originally proposed in [5] for algebraic rewriting. In this work however, algebraic rewriting is replaced by logic synthesis.

Layered verification is a technique in which verification is applied to each row of the divider array, associated with a single output bit q_i. The layered approach can be justified by realizing that logic between two adjacent rows is typically not optimized during synthesis and the partial remainder signals at their boundaries are preserved during synthesis. This has been elaborated on in [4], with Theorem 2 stating that "In an array divider there is no way to optimize carry logic of adjacent rows". That is, the circuit can be synthesized horizontally along each layer, but not optimized vertically across their boundaries. Independently, synthesis tools, such as Synopsys DC, tend to maintain the layered structure to simplify physical synthesis (place & route) after logic optimization. Even if the design is given without explicit location of the layer boundaries, there are extraction techniques to determine those boundaries, as done in [6].

The basic idea of layered hardware reduction is similar to that shown in Figure 2(c), but applied to a single layer, rather than to the entire circuit. In this case an add-on circuit $Z_i = R_{i+1} - q_i D$ is appended to layer i, as shown in Figure 3. The goal is to check if the resulting circuit is redundant; i.e., if output Z_i of the synthesized circuit matches bit-by-bit the partial remainder input R_{i+1}. This redundancy, if not directly exposed by synthesis, can be proved using bit-by-bit SAT, or by a simple XOR comparison.

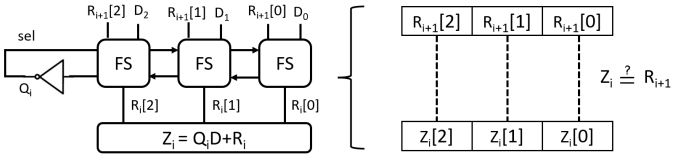

Fig. 3: Hardware reduction for a single layer.

In addition to verifying that each layer satisfies the relation $R_i = R_{i+1} - q_i D$, an additional constraint on the range of partial product computed at that layer must be verified. To correctly account for such a constraint, we need to view thr partial products as if they were computed across the entire width of the divider, including the corresponding bits of X. For this, we introduce a generalized partial remainder, labeled PR_i, each being $2n-1$ bit long, where $PR_n = X$ (dividend) and $PR_0 = R$ (remainder). A modified layer equation associated with layer i is then: $PR_i = PR_{i+1} - q_i 2^{i-1} D$. It correctly captures the fact that the value of such partial remainder reduces by half at each division iteration (layer). With this, it can be shown that $PR_i < 2^i D$, the constraint that needs to be verified. Similar constraint was also developed in [6] for non-restoring divider.

To check if such a constraint is satisfied by each layer, a comparator circuit that implements the complement of the above constraint, i.e., $PR_i \geq 2^i D$ is connected to the layer. Note again that each PR_i contains some bits of the divider X. A SAT procedure is then performed to check if it is unsatisfied (unSAT). Solving this SAT problem turns out to be very effective in terms of memory and time complexity, as shown in Table I. This also brings in an interesting observation that synthesizing a circuit appended with some additional function (such as the one that performs inversion, or a comparator that implements the desired constraint) will actually simplify the verification process.

IV. Results

The presented verification method was implemented as a prototype program with ABC environment as back-end. The experiments were conducted on a 64-bit Intel Core i7-7600 CPU, 2.80 GHz, with 30 GB of memory. The circuits were generated by an in-house divider generator tool and synthesized onto standard cells by the Synopsys Design Compiler (DC), using a *set don't touch* directive to maintain the layer boundaries. The gate-level mapped netlists produced by Synopsys DC were converted into an ABC-compatible format and synthesized using ABC commands: *strash, fraig, dch*.

The experimental results[2] are shown in Table I. The set of columns labeled Other Methods include verification results obtained with several methodologies: * *Simulation*: exhaustive simulation using Modelsim 10.5b; * *miniSAT*: SAT-based equivalence checking [19]; * *CEC-ABC*: combinational equivalence checking with ABC [16]; and * *SCA-DC*: symbolic computer algebra method of [6] (data given for non-restoring dividers, on Xeon CPU with 3.3 GHz clock and 64GB main memory). The entry " $-$ " in some fields indicates that the data was not available from the reference source.

The next set of columns, labeled This Work, reports the CPU times for different stages of our layered HR verification method: * *Resynthesis*: single-layer hardware reduction using ABC. * *HR-SAT*: verifying results of HR synthesis, followed by simple SAT, if needed; * $R < D$: verification of data range constraints for each layer, $PR_i < 2^i D$, including final $R < D$. * *Total time*: CPU time for all stages of the procedure for all layers. As one can see from the table, neither the simulation nor the equivalence checking CEC or SAT results could compete with the layered verification in terms of the CPU time.

In our experiments, we performed verification of all layers in series and reported the total time for the verification of the complete circuit. In practical implementation this can be done by running verification of all layer concurrently, therefore reducing the verification time by a factor of n.

[2]The software and the benchmarks used for the experiments are available at: https://drive.google.com/drive/folders/1D3m6yPUTQoesCOzb6U_ulgb5IhKkEqPK?usp=sharing

TABLE I: Hardware Reduction (HR) verification of restoring dividers. TO = Time-out 3600 sec.

Dividend # bits	# Gates	Verification time (sec) Other Methods				Verification time (sec) This Work: Hardware Reduction (HR)			
		Simulation	miniSAT [19]	CEC-ABC [16]	SCA-DC [6]	Resynthesis	HR-SAT	R < D	Total time
13	570	8.30	19.16	0.15	-	0.01	0.01	0.01	0.21
17	970	552.50	1584.32	0.34	1.91	0.01	0.01	0.01	0.27
19	1207	TO	TO	0.36	-	0.01	0.02	0.01	0.40
21	1470	TO	TO	0.38	-	0.01	0.02	0.01	0.44
23	1750	TO	TO	0.39	3.82	0.01	0.02	0.01	0.48
33	3700	TO	TO	0.58	6.79	0.02	0.03	0.01	0.68
63	13446	TO	TO	5.86	28.86	0.05	0.08	0.01	4.48
95	28200	TO	TO	18.85	70.72	0.10	0.15	0.02	12.96
127	51200	TO	TO	43.35	148.18	0.11	0.16	0.02	18.56
255	538579	TO	TO	1,073.03	989.91	0.25	0.21	0.10	123.46

V. Conclusions

This paper presents a novel approach to verify gate-level implementation of arithmetic dividers using an original *hardware reduction* (HR) technique. It seems counter-intuitive that adding more hardware to the design can simplify the verification problem at hand. While this obviously increases the gate-level complexity of the design, the resulting circuit (as long as it is bug-free) becomes *redundant* or trivial to verify. If the circuit is buggy, the complexity of the verification task is still lower than using a SAT-based approach. As demonstrated by the experiments, even if the synthesis is unable to reduce the resulting circuit to wires/buffers, standard SAT-based verification has a much easier task to prove equivalence.

The main limitation of the presented hardware reduction method is that it relies on the power of the synthesis tool used in the process: the synthesis may not always reduce the resulting circuit to bare wires. However, if the follow-up application of SAT solver cannot solve this significantly simplified problem, then no other SAT-based method can solve the *original* problem. In general, the method does not suffer from memory overload problem, experienced by rewriting methods, but for large circuits may require extended amount of time. The area penalty of the synthesized circuit imposed by the layered structure could be as high as 17%, but it is more important to have the circuit reliably verified than to have it aggressively optimized at the cost of making the functional verification ineffective. The main advantage is that the layered approach can significantly speed up the verification process and facilitate debugging. It can also facilitate debugging, since each layer is verified separately and could be done in parallel.

The idea of using synthesis to accomplish verification presented here can be applied to other circuits as long as a circuit implementing the inverse function can be efficiently generated. To the best of our knowledge, this is the first work that applied resynthesis to simplify verification with such good results.

ACKNOWLEDGMENT:
This work has been supported by a grant from the National Science Foundation, award No. CCF-2006465.

References

[1] M. Ciesielski, T. Su, A. Yasin, and C. Yu, "Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model," *IEEE Transactions on CAD*, pp. 1–1, 2019.
[2] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining sat and computer algebra," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 28–36.
[3] A. Mahzoon, D. Große, and R. Drechsler, "REVSCA-2.0: SCA-based formal verification of non-trivial multipliers using reverse engineering and local vanishing removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.
[4] M. H. Haghbayan and B. Alizadeh, "A dynamic specification to automatically debug and correct various divider circuits," *INTEGRATION, the VLSI journal*, vol. 53, pp. 100–114, 2016.
[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)*, Oct 2019, pp. 257–262.
[6] C. Scholl, A. Konrad, A. Mahzoon, D. Große, and R. Drechsler, "Verifying dividers using symbolic computer algebra and don't care optimization," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1110–1115.
[7] R. Kaivola and K. R. Kohatsu, "Proof engineering in the large: Formal verification of pentium®4 floating-point divider," in *Proc. 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, 2011*.
[8] D. M. Russinoff, *Formal Verification of floating-point hardware design: a mathematical approach*. Springer, 2018.
[9] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Tran. on Comp.*, vol. 100, no. 8, pp. 677–691, 1986.
[10] R. E. Bryant and Y. Chen, "Verification of arithmetic functions with binary moment diagrams," Pittsburgh, PA, USA, Tech. Rep., 1994.
[11] R. E. Bryant, "Bit-level analysis of an SRT divider circuit," in *33rd (DAC)*. ACM, 1996, pp. 661–665.
[12] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*. Springer, 1997.
[13] E. Pavlenko, M. Wedler, D. Stoffel, and W. Kunz, "Stable: A new QF-BV smt solver for hard verification problems combining Boolean reasoning with computer algebra," in *DATE*, 2011, pp. 155–160.
[14] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits using function extraction," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 35/12, pp. 2131–2142, 2016.
[15] I. Koren, *Computer Arithmetic Algorithms*. Universities Press, second edition, 2002.
[16] A. Mishchenko, "ABC: A System for Sequential Synthesis and Verification," *URL http://www.eecs.berkeley.edu/˜ alanmi/abc*, 2007.
[17] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, p. 53, 2005.
[18] A. Yasin, T. Su, S. Pillement, and M. Ciesielski, "SPEAR: hardware-based implicit rewriting for square-root circuit verification," *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 532–537, 2020.
[19] N. Sörensson and N. Eén, "MiniSat 2.1 and MiniSat++ 1.0 - SAT race 2008 editions," *SAT*, p. 31, 2009.