Divider Verification Using Symbolic Computer Algebra and Delayed Don't Care Optimization (extended abstract)

Alexander Konrad[a], Christoph Scholl[a], Alireza Mahzoon[b], Daniel Große[c], and Rolf Drechsler[b]
[a]University of Freiburg, Germany, {konrada, scholl}@informatik.uni-freiburg.de
[b]University of Bremen, Germany, {mahzoon, drechsle}@informatik.uni-bremen.de
[c]Johannes Kepler University Linz, Austria, daniel.grosse@jku.at

Abstract

Recent methods based on Symbolic Computer Algebra (SCA) have shown great success in formal verification of multipliers and – more recently – of dividers as well. Here we give an overview of our work published in [1–3] which enhances SCA-based verification by the computation of *satisfiability don't cares for so-called (Extended) Atomic Blocks (EABs)* and by *Delayed Don't Care Optimization (DDCO)* for optimizing polynomials during backward rewriting. The optimization is reduced to Integer Linear Programming (ILP). Whereas the basic methods using SCA failed for divider verification, using those novel methods we are able to verify (formally and fully automatically) large gate level implementations of several divider architectures (with bit widths up to 512).

1 Introduction and Background

Arithmetic circuits are important components in processor designs as well as in special-purpose hardware for computationally intensive applications like signal processing and cryptography. At the latest since the famous Pentium bug [4] in 1994, where a subtle design error in the divider had not been detected by Intel's design validation (leading to erroneous Pentium chips brought to the market), it has been widely recognized that incomplete simulation-based approaches are not sufficient for verification and formal methods should be used to verify the correctness of arithmetic circuits. Nowadays the design of circuits containing arithmetic is not only confined to the major processor vendors, but is also done by many different suppliers of special-purpose embedded hardware who cannot afford to employ large teams of specialized verification engineers being able to provide human-assisted theorem proofs. Therefore the interest in *fully automatic formal verification* of arithmetic circuits is growing more and more.

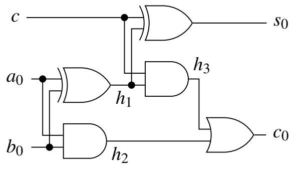
In particular the verification of multiplier and divider circuits formed a major problem for a long time. Both BDD-based methods [5, 6] and SAT-based methods [7] for multiplier and divider verification do not scale to large bit widths. Nevertheless, there has been great progress during the last few years for the automatic formal verification of gate-level multipliers. Methods based on *Symbolic Computer Algebra (SCA)* were able to verify large, structurally complex, and highly optimized multipliers. In this context, finite field multipliers [8], integer multipliers (e.g. [9–16]), and modular multipliers [17] have been considered. Here the verification task has been reduced to an ideal membership test for the specification polynomial based on so-called backward rewriting, proceeding from the outputs of the circuit in direction of the inputs.

Research approaches for divider verification were lagging behind for a long time. Attempts to use Decision Diagrams for proving the correctness of an SRT divider [18] were confined to a single stage of the divider (at the gate level) [19]. Methods based on word-level model checking [20] looked into SRT division as well, but considered only a special abstract and clean sequential (i.e., non-combinatorial) divider without gate-level optimizations. Other approaches like [21], [22], or [23] looked into fixed division algorithms and used semi-automatic theorem proving with ACL2, Analytica, or Forte to prove their correctness. Nevertheless, all those efforts did not lead to a fully automated verification method suitable for gate-level dividers.

A side remark in [24] (where actually multiplier verification with *BMDs was considered) seemed to provide an idea for a fully automated method to verify integer dividers as well. Hamaguchi et al. start with a *BMD representing $Q \times D + R$ (where Q is the quotient, D the divisor, and R the remainder of the division) and use a backward construction to replace the bits of Q and R step by step by *BMDs representing the gates of the divider. The goal is to finally obtain a *BMD representation for the dividend $R^{(0)}$ which proves the correctness of the divider circuit. Unfortunately, the approach has not been successful in practice: Experimental results showed exponential blow-ups of *BMDs during the backward construction.

Recently, there have been several approaches to fully automatic divider verification that had the goal to catch up with successful approaches to multiplier verification: Among those approaches, [25] is mainly confined to division by constants and cannot handle general dividers due to a memory explosion problem. [26] works at the gate level, but assumes that hierarchy information in a restoring divider is present. Using this hierarchy information it decomposes the proof obligation $R^{(0)} = Q \times D + R$ into separate proof obligations for each level of the restoring divider. Nevertheless, the approach scales only to medium-sized bit widths (up to 21 as shown in the experimental results of [26]).

In the following we give a brief review of SCA for verification (Sect. 2) and of (simple) divider circuits (Sect. 3). In



$$\begin{aligned} & 2c_0 + s_0 \\ \xrightarrow{c_0}\ & 2h_2 + 2h_3 - 2h_2h_3 + s_0 \\ \xrightarrow{h_3}\ & 2h_2 + 2ch_1 - 2ch_1h_2 + s_0 \\ \xrightarrow{s_0}\ & 2h_2 - 2ch_1h_2 + c + h_1 \\ \xrightarrow{h_2}\ & 2a_0b_0 - 2a_0b_0ch_1 + c + h_1 \\ \xrightarrow{h_1}\ & a_0 + b_0 + c \end{aligned}$$

Figure 1: Circuit with series of substitutions.

Sect. 4 we summarize the reasons why divider verification is not as easy as it might seem from a high-level point of view and in Sect. 5 we give an overview of our approach.

2 SCA for Verification

For the presentation of SCA we basically follow [1]. SCA based approaches work with polynomials and reduce the verification task to an ideal membership test using a Gröbner basis representation of the ideal. The ideal membership test is performed using polynomial division. While Gröbner basis theory is very general and, e.g., can be applied to finite field multipliers [8] and truncated multipliers [14] as well, for integer arithmetic it boils down to substitutions of variables for gate outputs by polynomials over the gate inputs (in reverse topological order), if we choose an appropriate "term order" (see [11], e.g.). Here we restrict ourselves to exactly this view.

For integer arithmetic we consider polynomials over binary variables (from a set $X = \{x_1, \dots, x_n\}$) with integer coefficients, i. e., a polynomial is a sum of terms, a term is a product of a monomial with an integer, and a monomial is a product of variables from X. Polynomials represent *pseudo-Boolean functions* $f : \{0,1\}^n \mapsto \mathbb{Z}$.

As a simple example consider the full adder from Fig. 1. The full adder defines a pseudo-Boolean function $f_{FA} : \{0,1\}^3 \mapsto \mathbb{Z}$ with $f_{FA}(a_0, b_0, c) = a_0 + b_0 + c$. We can compute a polynomial representation for f_{FA} by starting with a weighted sum $2c_0 + s_0$ (called the "output signature" in [10]) of the output variables. Step by step, we replace the variables in polynomials by the so–called "gate polynomials". This replacement is performed in reverse topological order of the circuit, see Fig. 1. We start by replacing c_0 in $2c_0 + s_0$ by its gate polynomial $h_2 + h_3 - h_2h_3$ (which is derived from the Boolean function $c_0 = h_2 \vee h_3$). Finally, we arrive at the polynomial $a_0 + b_0 + c$ (called the "input signature" in [10]) representing the pseudo-Boolean function defined by the circuit. During this procedure (which is called *backward rewriting*) the polynomials are simplified by reducing powers v^k of variables v with $k > 1$ to v (since the variables are binary), by combining terms with identical monomials into one term, and by omitting terms with leading factor 0. We can also consider $a_0 + b_0 + c = 2c_0 + s_0$ as the "specification" of the full adder. The circuit implements a full adder iff backward substitution, now starting with $2c_0 + s_0 - a_0 - b_0 - c$ instead of $2c_0 + s_0$, reduces the "specification polynomial" to 0 in the end. (This is the notion usually preferred in SCA-based verification.)

The correctness of the method relies on the fact that polynomials (with the above mentioned simplifications resp. normalizations) are canonical representations of pseudo-Boolean functions (up to reordering of the terms). (This is formulated as Lemma 1 in [2] and proven in [1], e.g..)

3 Divider Circuits

In the following we briefly review textbook knowledge on dividers. We use $\langle a_n, \dots, a_0 \rangle := \sum_{i=0}^{n} a_i 2^i$ and $[a_n, \dots, a_0]_2 := (\sum_{i=0}^{n-1} a_i 2^i) - a_n 2^n$ for interpretations of bit vectors $(a_n, \dots, a_0) \in \{0,1\}^{n+1}$ as unsigned binary numbers and two's complement numbers, respectively. The leading bit a_n is called the sign bit. An unsigned integer divider is a circuit with the following property:

Definition 1. *Let $(r^{(0)}_{2n-2} \dots r^{(0)}_0)$ be the dividend with sign bit $r^{(0)}_{2n-2} = 0$ and value $R^{(0)} := \langle r^{(0)}_{2n-2} \dots r^{(0)}_0 \rangle = [r^{(0)}_{2n-2} \dots r^{(0)}_0]_2$, $(d_{n-1} \dots d_0)$ be the divisor with sign bit $d_{n-1} = 0$ and value $D := \langle d_{n-1} \dots d_0 \rangle = [d_{n-1} \dots d_0]_2$, and let $0 \le R^{(0)} < D \cdot 2^{n-1}$. Then $(q_{n-1} \dots q_0)$ with value $Q = \langle q_{n-1} \dots q_0 \rangle$ is the quotient of the division and $(r_{n-1} \dots r_0)$ with value $R = [r_{n-1} \dots r_0]_2$ is the remainder of the division, if $R^{(0)} = Q \cdot D + R$ (verification condition 1 = "vc1") and $0 \le R < D$ (verification condition 2 = "vc2").*

The simplest algorithm to compute quotient and remainder is *restoring division* which is the "school method" to compute quotient bits and "partial remainders" $R^{(j)}$. In each step it subtracts a shifted version of D. If the result is less than 0, the corresponding quotient bit is 0 and the shifted version of D is "added back", i. e., "restored". Otherwise the quotient bit is 1 and the algorithm proceeds with the next smaller shifted version of D. *Non-restoring division* optimizes restoring division by combining two steps of restoring division in case of a negative partial remainder: adding the shifted D back and (tentatively) subtracting the next D shifted by one position less. These two steps are replaced by just adding D shifted by one position less (which obviously leads to the same result). More precisely, non-restoring division works according to Alg. 1.

Algorithm 1 Non-restoring division.

```
1: $R^{(1)} := R^{(0)} - D \cdot 2^{n-1}$;
2: if $R^{(1)} < 0$ then $q_{n-1} := 0$ else $q_{n-1} := 1$;
3: for $j = 2$ to $n$ do
4:     if $R^{(j-1)} \ge 0$ then
5:         $R^{(j)} := R^{(j-1)} - D \cdot 2^{n-j}$
6:     else
7:         $R^{(j)} := R^{(j-1)} + D \cdot 2^{n-j}$;
8:     if $R^{(j)} < 0$ then $q_{n-j} := 0$ else $q_{n-j} := 1$;
9: $R := R^{(n)} + (1 - q_0) \cdot D$;
```

For dividers it is near at hand to start backward rewriting not with polynomials for the binary representations of the output words (which is basically done for multiplier verification), but with a polynomial for $Q \cdot D + R$. For a correct divider one would expect to obtain a polynomial for $R^{(0)}$ after backward rewriting. As an alternative one could also start with $Q \cdot D + R - R^{(0)}$ and one would expect that for a correct divider the result after backward rewriting is 0. This would be a proof for verification condition (vc1). (Then it remains to show that $0 \le R < D$ (vc2) which we verified in [1–3] as well.)

4 Challenges of Divider Verification

We want to summarize the reasons why divider verification is not as easy as it might seem from a high-level point of view. Fig. 2 shows such a high-level view on the non-

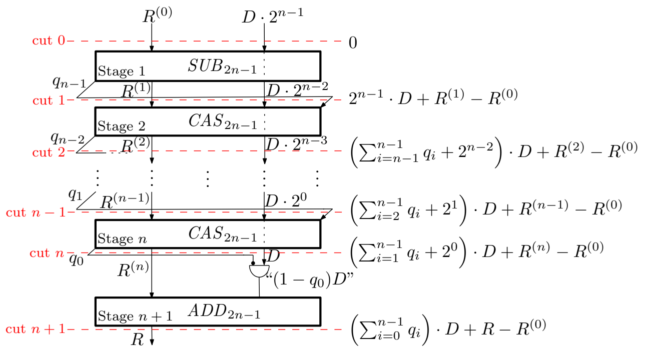


Figure 2: High-level view of a non-restoring divider.

restoring division algorithm. Stage 1 implements a subtractor, stages j with $j \in \{2,\dots,n\}$ implement conditional adders / subtractors depending on the value of q_{n-j+1}, and stage $n+1$ implements an adder. If we start backward rewriting with the polynomial $Q \cdot D + R - R^{(0)}$ (which is quadratic in n) and if backward rewriting processes the gates in the circuit in a way that the stages shown in Fig. 2 are processed one after the other, then we would expect the following polynomials on the corresponding cuts (see also Fig. 2): We would expect $(\sum_{i=1}^{n-1} q_i 2^i + 2^0) \cdot D + R^{(n)} - R^{(0)}$ for the polynomial at cut n which is obtained after processing stage $n+1$, since stage $n+1$ enforces $R = R^{(n)} + (1-q_0) \cdot D$. For $j = n$ to 2 we would (by induction) expect $(\sum_{i=n-j+2}^{n-1} q_i 2^i + 2^{n-j+1}) \cdot D + R^{(j-1)} - R^{(0)}$ for the polynomial at cut $j-1$ after processing stage j, since stage j enforces $R^{(j)} = R^{(j-1)} - q_{n-j+1}(D \cdot 2^{n-j}) + (1-q_{n-j+1})(D \cdot 2^{n-j}) = R^{(j-1)} + (1-2q_{n-j+1})(D \cdot 2^{n-j})$. Finally, the polynomial at cut 0 after processing stage 1 using the equation $R^{(1)} = R^{(0)} - D \cdot 2^{n-1}$ would reduce to 0.

Unfortunately, considering usual optimizations in implementations of non-restoring dividers the polynomials represented at the cuts between stages are different from this high-level derivation. The reason lies in the fact that the stages do not really implement signed addition / subtraction. In general, signed addition / subtraction of two $(2n-1)$-bit numbers leads to a $2n$-bit number. The leading bit of the result can only be omitted, if "no overflow occurs". The fact that no overflow occurs results from the input constraint $0 \le R^{(0)} < D \cdot 2^{n-1}$ of the divider and from the way the results of the different stages are computed [1]. Usual implementations even go one step further: By additional arguments using the input constraint and the circuit functionality it can be shown that it is not only possible to omit overflow bits of the adder / subtractor stages, but it is even possible to omit the computation of one further most significant bit. For a detailed analysis see [2, 3].

These considerations lead to an optimized implementation shown in Fig. 3 for $n = 4$, e.g.. (For simplicity, we present the circuit before propagation of constants which is done however in the real implemented circuit.) In summary, it is important to note that (1) the stages in Fig. 3 cannot be seen as real adder / subtractor stages as shown in the high-level view from Fig. 2, (2) backward rewriting leads to polynomials at the cuts which are different from the ones shown in Fig. 2, and (3) unfortunately those polynomials have (provably) exponential sizes.

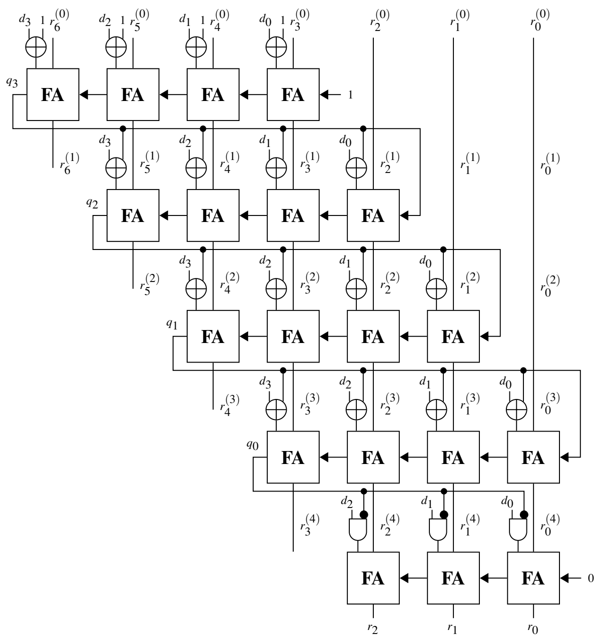


Figure 3: Optimized non-restoring divider, $n = 4$.

5 Overview of our Approach

Our approaches in [1, 2] work on the gate level, but they do not need any hierarchy information which may have been lost during logic optimization. They prove the correctness of non-restoring dividers by "backward rewriting" starting with the "specification polynomial" $Q \times D + R - R^{(0)}$ (similar to [24], with polynomials instead of *BMDs as internal data structure). Backward rewriting performs substitutions of gate output variables with the gates' specification polynomials in reverse topological order. They try to prove dividers to be correct by finally obtaining the 0-polynomial. The main insight of [1, 2] is the following: The backward rewriting method definitely needs "forward information propagation" to be successful, otherwise it provably fails due to exponential sizes of intermediate polynomials. Forward information propagation relies on the fact that the divider needs to work only within a range of allowed divider inputs (leading to input constraints like $0 \le R^{(0)} < D \cdot 2^{n-1}$). [1] uses SAT-based information propagation (SBIF) of the input constraint in order to derive information on equivalent and antivalent signals, whereas [2] uses BDDs to compute satisfiability don't cares which result from the structure of the divider circuit as well as from the input constraint. The don't cares are used to minimize the sizes of polynomials. In that way, exponential blowups in polynomial sizes which would occur without don't care optimization could be effectively avoided. Since polynomials are only changed for input values which do not occur in the circuit if only inputs from the allowed range are applied, the verification with don't care optimization remains correct. In [2] the computation of optimized polynomials is reduced to suitable *Integer Linear Programming* (ILP) problems.

In our most recent work [3] we made two additional contributions to improve [1] and [2]: First, we modified the computation of don't cares leading to increased degrees of flexibility for the optimization of polynomials. Instead of computing don't cares at the inputs of "atomic blocks" like full adders, half adders etc., which were detected in the gate

level netlist, we combined atomic blocks and surrounding gates into larger fanout-free cones, leading to so-called *Extended Atomic Blocks (EABs)*, prior to the don't care computation. Second, we replaced local don't care optimization by *Delayed Don't Care Optimization (DDCO)*. Whereas local don't care optimization immediately optimizes polynomials w.r.t. a don't care cube as soon as the polynomial contains the input variables of the cube, DDCO only adds don't care terms to the polynomial, but delays the optimization until a later time. This method has two advantages: First, by looking at the polynomial later on, we can decide whether exploitation of certain don't cares is needed *at all*, and secondly, the later (delayed) optimization will take the effect of following substitutions into account and thus uses a more global view for optimization.
Using those novel methods we were able to successfully verify different large gate-level dividers (with bit widths up to 512 bits).

Literature

[1] C. Scholl and A. Konrad, "Symbolic computer algebra and sat based information forwarding for fully automatic divider verification," in *DAC*, 2020.

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

[3] A. Konrad, C. Scholl, A. Mahzoon, D. Große, and R. Drechsler, "Divider verification using symbolic computer algebra and delayed don't care optimization," in *FMCAD*, 2022, pp. 108–117.

[4] T. Coe, "Inside the Pentium FDIV bug," *Dr. Dobbs J.*, vol. 20, no. 4, pp. 129—-135, 1995.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *TC*, vol. 35, no. 8, pp. 677–691, 1986.

[6] J. R. Burch, "Using BDDs to verify multipliers," in *DAC*, 1991, pp. 408–412.

[7] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using SAT for combinational equivalence checking," in *DATE*. IEEE Computer Society, 2001, pp. 114–121.

[8] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits," *TCAD*, vol. 32, no. 9, pp. 1409–1420, 2013.

[9] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Greuel, "An algebraic approach for proving data correctness in arithmetic data paths," in *CAV*, 2008, pp. 473–486.

[10] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *TCAD*, vol. 35, no. 12, pp. 2131–2142, 2016.

[11] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal verification of integer multipliers by combining Gröbner basis with logic reduction," in *DATE*, 2016, pp. 1048–1053.

[12] D. Ritirc, A. Biere, and M. Kauers, "Column-wise verification of multipliers using computer algebra," in *FMCAD*, 2017, pp. 23–30.

[13] A. Mahzoon, D. Große, and R. Drechsler, "PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers," in *ICCAD*, 2018, pp. 129:1–129:8.

[14] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining SAT and computer algebra," in *FMCAD*, 2019, pp. 28–36.

[15] A. Mahzoon, D. Große, and R. Drechsler, "RevSCA: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers," in *DAC*, 2019, pp. 185:1–185:6.

[16] A. Mahzoon, D. Große, C. Scholl, and R. Drechsler, "Towards formal verification of optimized and industrial multipliers," in *DATE*, 2020, pp. 544–549.

[17] A. Mahzoon, D. Große, C. Scholl, A. Konrad, and R. Drechsler, "Formal verification of modular multipliers using symbolic computer algebra and boolean satisfiability," in *DAC*, 2022, pp. 1183–1188.

[18] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electronic Computers*, vol. 7, no. 3, pp. 218–222, 1958.

[19] R. E. Bryant, "Bit-level analysis of an SRT divider circuit," in *DAC*, 1996, pp. 661–665.

[20] E. M. Clarke, M. Khaira, and X. Zhao, "Word level model checking - avoiding the Pentium FDIV error," in *DAC*, 1996, pp. 645–648.

[21] D. M. Russinoff, "A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor," *LMS Journal Comput. Math.*, vol. 1, pp. 148–200, 1998.

[22] E. M. Clarke, S. M. German, and X. Zhao, "Verifying the SRT division algorithm using theorem proving techniques," *Form Methods Syst. Des.*, vol. 14, no. 1, pp. 7–44, 1999.

[23] J. O'Leary, X. Zhao, R. Gerth, and C.-J. H. Seger, "Formally verifying IEEE compliance of floating point hardware," *Intel Technology Journal*, vol. Q1, pp. 1–10, 1999.

[24] K. Hamaguchi, A. Morita, and S. Yajima, "Efficient construction of binary moment diagrams for verifying arithmetic circuits," in *ICCAD*, 1995, pp. 78–82.

[25] A. Yasin, T. Su, S. Pillement, and M. J. Ciesielski, "Formal verification of integer dividers: Division by a constant," in *ISVLSI*, 2019, pp. 76–81.

[26] ——, "Functional verification of hardware dividers using algebraic model," in *VLSI-SoC*, 2019, pp. 257–262.