SEMINAR REPORT
Seminar: Embedded Systems in Summer term 2025

Rheinland-Pfälzische
Technische Universität
Kaiserslautern
Landau

Recent Developments in Formal Verification of Restoring and non-Restoring Dividers

David Mozek
Rheinland-Pfälzische Technische Universität Kaiserslautern, Department of Computer Science

Note: *This report is a compilation of publications related to some topic as a result of a student seminar. It does not claim to introduce original work and all sources should be properly cited.*

Divider circuits are among the most challenging arithmetic components to verify due to their structural complexity, sequential nature, and aggressive gate-level optimizations. Ensuring their correctness is crucial, as the well-known Intel Pentium FDIV bug illustrated in 1994. Formal verification offers a systematic solution, but classical SAT- and BDD-based approaches have struggled to scale for restoring and non-restoring dividers. This report surveys recent advances in the formal verification of divider circuits, with a focus on two different approaches. The first, proposed by Scholl and Konrad et al., develops symbolic computer algebra (SCA) methods that extend ideal membership checking with satisfiability don't-cares, extended atomic block representations, and delayed don't-care optimization to achieve scalable, fully automatic verification of large, optimized divider designs. The second, introduced by Yasin et al., proposes a hardware-reduction technique that composes the divider with an arithmetic inverse and applies synthesis-driven reduction. We present the theoretical foundations and algorithmic innovations of both approaches and compare their strengths and limitations.

1 Introduction

Division is a fundamental arithmetic operation used pervasively in processors, signal processing units, cryptographic hardware, and embedded controllers. As semiconductor designs become more diverse and specialized (from general-purpose CPUs to domain-specific accelerators), the cost of a missed error grows. An error in the correctness of an arithmetic circuit discovered after tape-out can force expensive recalls, reliability losses, or catastrophic downstream failures. A canonical and cautionary example is the Intel Pentium FDIV bug of 1994, where a subtle error in the FPU's division lookup table produced incorrect floating-point quotients for certain inputs. This error escaped validation, entered shipping silicon, and led to a public recall and large financial and reputational costs.

With an increased reliance on microprocessors of many kinds in our everyday lives, there are also many different vendors for such microprocessors. However, many of the smaller vendors may be unable to afford the expertise required for the manual verification of the provided hardware, leading to the need for fully automated verification methods. While there has much less progress in the automated verification of divider circuits compared to other arithmetic circuits, some promising methods were recently proposed for the verification of restoring and non-restoring dividers.

Symbolic computer algebra(SCA) has already been used extensively in the verification of other arithmetic circuits, such as integer multipliers [3]. More recently, an SCA based method for the verification of non-restoring dividers was proposed by [12] and further improved in [8, 9, 13]. Further, a novel approach using hardware synthesis for verification has been proposed by [14] and applied to restoring dividers. Due to most recent developments for the formal verification of dividers being done for restoring and non-restoring dividers, we will focus on these.

In this paper we aim to illustrate these methods and give an understanding of their advantages and disadvantages. We paint the current state of verification and related papers in Section 2. In Section 3 we give an introduction to restoring and non-restoring dividers, including possible hardware architectures, and a method called *backward rewriting*, which is used by all SCA based methods we discuss. Section 4 focuses on iteratively building up to the most recent symbolic computer algebra (SCA) based method from its earlier revisions. This is followed by Section 5 in which we discuss verification by hardware reduction. In Section 6 we discuss each method's weaknesses and advantages, as well as present a summary of the benchmarks reported by the authors of those methods. Finally, we conclude in Section 7.

2 Related Works

The formal verification of arithmetic circuits has been a longstanding research area. [2] provide a broad historical overview over these, presenting theorem provers, canonical diagrams, SAT and SMT based methods, SCA methods and finally hardware reduction in brief.

Recently there have been several approaches for the automatic verification of dividers. [15] focuses on the division by constants and [16] assumes the presence of hierarchical information about the divider, which is usually lost in the process of hardware synthesis. Following these, [12] proposed SAT based information forwarding (SBIF) to reconstruct some hierarchical information for a symbolic computer algebra based approach. This approach is based on *backward rewriting*, which is itself based on the backward construction of multiplicative binary moment diagrams(*BMDs) [1] proposed by Hamaguchi et al. [7] for the verification of multiplier circuits. While the notion of using backward construction for dividers, using the specification polynomial $Quotient \cdot Divisor + Remainder$ as the starting *BMD, was already proposed by Hamaguchi et al., results showed exponential blow-ups of *BMDs in the backward construction [7]. [12] remedies this issue by applying this approach to symbolic computer algebra (SCA) and proposing SBIF for the propagation of information from inputs to outputs. This method is then further improved by [13] and [8, 9], which we will focus on in Section 4.

In a similar timeframe, an approach using logic synthesis for verification was proposed by [14]. This approach aims to reduce the verification of an arithmetic circuit to a redundancy check by appending a circuit computing the inverse arithmetic operation to the outputs of the circuit being verified. Another novel approach was recently proposed for the verification of restoring dividers by [5], in which the verification is accomplished in three phases. First layer boundaries are extracted by setting signals derived from quotient bits. Next each layer is verified using combinational equivalence checking. Finally, a global proof that the entire circuit implements a divider is performed in a word-level space.

The variety of approaches to divider verification shows that the verification of divider circuits is still an active area of research and an optimal solution has yet to be found. In the following section, we introduce the concept of restoring and non-restoring dividers, as well as algebraic rewriting and hardware reduction-based methods for verifying them.

3 Fundamentals

Before we examine the verification methods in detail, it is necessary to understand the architectures of the dividers which we intend to verify. For this, we follow the restoring and non-restoring division algorithms from [8, 9, 12–14].

Divider circuits can be implemented using a variety of algorithms and architectural principles, each offering different trade-offs in terms of latency, area, and verification complexity. In this report, our focus is on restoring and non-restoring dividers, which belong to the class of array dividers. Array dividers are structurally similar to array multipliers, with a highly regular arrangement of cells. They perform the division iteratively, layer by layer, and produce the quotient bits in sequence. This regularity makes them attractive for hardware design, but also introduces verification challenges because each layer involves conditional subtraction and updating of the partial remainder.

Besides restoring and non-restoring dividers, several other divider architectures are also used in practice, such as SRT dividers, Goldschmidt dividers or Newton-Raphson dividers.

In the following we will focus on unsigned integer dividers.

3.1 Notation

For the notation used in this work we will follow [8, 9, 12, 13]. We define the input dividend as $R^{(0)} = (r^0_{2n-2}, ..., r^0_0)$, the output remainder as R, the quotient as Q represented by the bit-vector $(q_{n-1}, ..., q_0)$ with q_{n-1} and r^0_{2n-2} being the most significant bit respectively. Similarly, we define the divisor D as the bit vector $(d_{n-1}, ..., d_0)$. Thus, a correct divider should fulfil verification conditions $vc1$ and $vc2$ under input constraint IC [8, 9].

Definition 1 $vc1$: *A divider computes the quotient* Q *and remainder* R *such that* $Q \cdot D + R = R^{(0)}$*. This can be rephrased as* $Q \cdot D + R - R^{(0)} = 0$*.*

Definition 2 $vc2$: *The remainder* R *should be smaller than the divisor* $0 \leq R < D$*. In combination with* $vc1$ *this guarantees that the quotient is as large as it can be while fulfilling* $vc1$*.*

Definition 3 $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$*, to prevent an overflow in* Q*.*

We will thus be using the notion that a divider of bit-width n performs division using a divisor of size n and a dividend of size $2n - 1$.

3.2 Restoring Dividers

Restoring division is one of the simplest division algorithms. For every bit in the resulting quotient, the divisor D is shifted by the position of that bit and subtracted from the partial remainder $R^{(j)}$ of the previous bit position. The initial partial remainder $R^{(0)}$ is the dividend. If the new partial remainder is negative, the quotient bit is 0 and the divisor added back. Otherwise, the quotient bit is 1. The final partial remainder is then the remainder R.

Algorithm 1 Restoring division from [8]

```
for j = 1 to n do
    $R^{(j)} := R^{(j-1)} - D \cdot 2^{n-j}$;
    if $R^{(j)} < 0$ then
        $q_{n-j} := 0$;
        $R^{(j)} := R^{(j)} + D \cdot 2^{n-j}$;
    else
        $q_{n-j} := 1$;
$R := R^{(n)}$;
```

$$
\begin{aligned}
&D \cdot 2^{n-(j-1)} - D \cdot 2^{n-j} && (1)\\
&= D \cdot 2^{n-j} + D \cdot 2^{n-j} - D \cdot 2^{n-j} && (2)\\
&= D \cdot 2^{n-j} && (3)
\end{aligned}
$$

Figure 1: Optimization in non-restoring dividers

3.3 Non-restoring Dividers

To improve on restoring dividers, non-restoring dividers combine the subtraction and potential back addition into one step. This is achieved by adding the divisor shifted by one position less, instead of directly adding the shifted divisor back. As seen in Figure 1, if in the previous step $D \cdot 2^{n-(j-1)}$ had been added back and then, in the current step, $D \cdot 2^{n-(j)}$ subtracted, this is equal to just adding $D \cdot 2^{n-j}$ in the current step instead of subtracting. In the case that the previous partial remainder was positive after subtraction, non-restoring division behaves like restoring division. Algorithm 2 illustrates the algorithm in detail.

Algorithm 2 Non-restoring division from [8]

```
$R^{(1)} := R^{(0)} - D \cdot 2^{n-1}$;
if $R^{(1)} < 0$ then $q_{n-1} := 0$ else $q_{n-1} := 1$;
for j = 2 to n do
    if $R^{(j-1)} \geq 0$ then
        $R^{(j)} := R^{(j-1)} - D \cdot 2^{n-j}$;
    else
        $R^{(j)} := R^{(j-1)} + D \cdot 2^{n-j}$;
    if $R^{(j)} < 0$ then
        $q_{n-j} := 0$;
    else
        $q_{n-j} := 1$;

$R := R^{(n)} + (1 - q_0) \cdot D$;
```

Since the addition of two's complement numbers can be implemented by the addition of two unsigned numbers, a simple circuit for a non-restoring can be implemented by multiple layers of additions, where the sign of the two's complement number added is controlled by the carry bit from the previous addition. The carry bit of the i-th addition is then also the i-th bit of the quotient. This results in a layered architecture with n layers of additions and an additional layer performing the calculation $R := R^{(n)} + (1 - q_0) \cdot D$. Figure 2 depicts such a divider architecture for $n = 3$. On a side note, a restoring divider can be implemented similarly, but with two additions in each layer.

This design uses addition layers with a full adder for each bit of the dividend, resulting in $2n - 1$ full adders in each layer. It is possible to further optimize this design by only using n full adders in each layer [9, 13]. Such a design is depicted by Figure 3, including the gray full adder in the bottom most layer. As indicated by the gray full adder, it is further possible to omit the computation of the most significant bit in the bottom most row of the divider. Both of these optimizations are proven and explained in detail by [9].

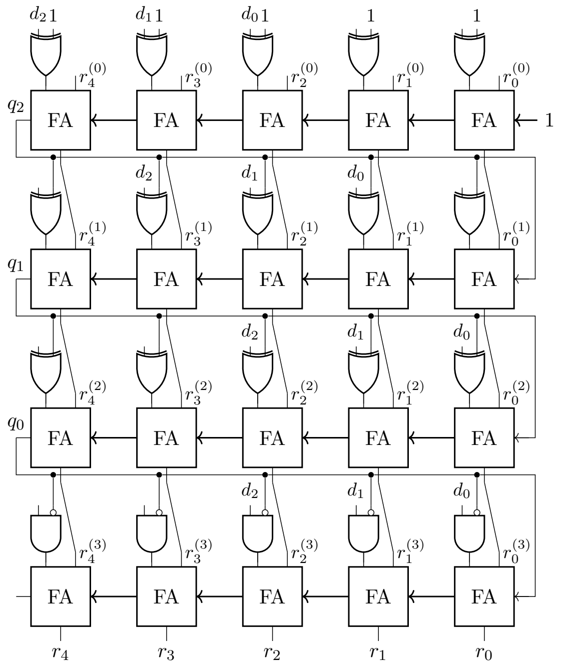
Figure 2: Unoptimized non-restoring divider [13], undefined inputs are assumed to be 0

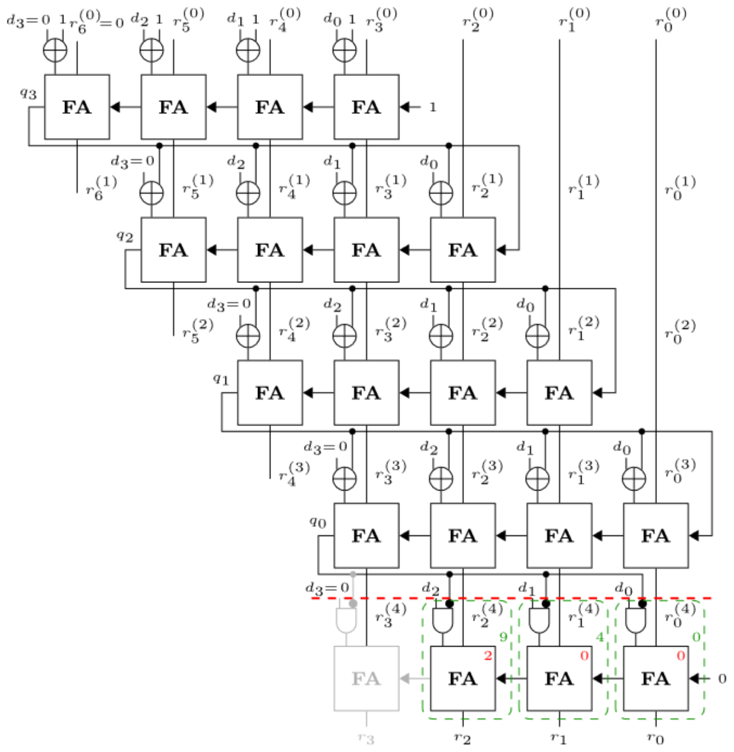
Figure 3: Optimized non-restoring divider from [9]

3.4 Backward Rewriting

In this section we will explain the basic backward rewriting technique for symbolic computer algebra used by [12] and then further improved in [8,9,13]. The goal of backward rewriting for the methods discussed in this report is to rewrite the output polynomial of the circuit as the input polynomial via substitution, using the gate polynomials of the circuit's logic gates. This substitution is performed in reverse order. For divider circuits, this order is topological, from inputs to outputs.

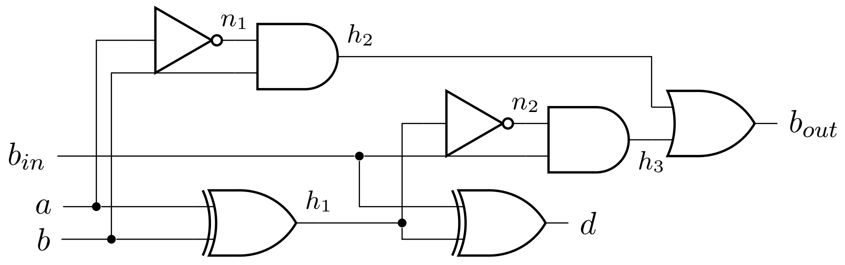

Figure 4: Full subtractor circuit implementing $d = a - b - b_{in} + 2b_{out}$

Logic Gate	Algebraic Model
NOT a	$1 - a$
a AND b	$a \cdot b$
a NAND b	$1 - (a \cdot b)$
a OR b	$a + b - a \cdot b$
a XOR b	$a + b - 2 \cdot a \cdot b$

Table 1: Algebraic models of common logic gates [16] (with added NAND gate)

To illustrate this technique, let us examine the full subtractor circuit shown in Figure 4. Following Table 1 we can translate the gate outputs into polynomials to arrive at the following set of equations:

$$\begin{aligned}
d &= h_1 + b_{in} - 2 \cdot h_1 \cdot b_{in} \\
b_{out} &= h_3 + h_2 - h_3 h_2 \\
h_3 &= b_{in} \cdot n_2 \\
h_2 &= b \cdot n_1 \\
h_1 &= a + b - 2 \cdot a \cdot b \\
n_2 &= 1 - h_1 \\
n_1 &= 1 - a
\end{aligned}$$

We can now compute the polynomial expression this circuit implements by substitution variables in reverse topological order, starting with the output signature $d - 2b_{out}$. If the full subtractor is implemented correctly, we should arrive at

$$d - 2b_{out} = a - b - b_{in}$$

First we substitute b_{out}:

$$d - 2(h_3 + h_2 - h_3 h_2)$$

and expand it

$$d - 2h_3 - 2h_2 + 2h_3 h_2$$

We continue to substitute h_3, d, n_2, h_2, n_1 and h_1 in the order they are listed in. While doing so, the polynomial is simplified and expanded as much as possible by leaving out terms with a factor of zero and combining terms with the same monomials, arriving at

$$a - b - b_{in} - 4a^2b^2b_{in} + 2a^2bb_{in} + 6ab^2b_{in} - 4abb_{in} - 2b^2b_{in} + 2bb_{in}$$

A crucial step for backward rewriting, which we have left out until now, is that we can reduce any power x^i to x for $i > 1$. This is due to the fact that input and output signals are either 0 or 1. Alternatively, x^i could be seen as the algebraic representation of $x \wedge ... \wedge x$, which obviously reduced to x in boolean logic. By applying this we arrive at

$$a - b - b_{in} - 4abb_{in} + 2abb_{in} + 6abb_{in} - 4abb_{in} - 2bb_{in} + 2bb_{in}$$

We can then further reduce this to

$$a - b - b_{in}$$

Thus we can conclude, that $d - 2b_{out} = a - b - b_{in}$ holds, and the full subtractor is correct.

4 Computer Algebra Methods

For *symbolic computer algebra* (SCA) methods the logic gates of the circuit to verify are translated into polynomials in an algebraic model. Table 1 shows how simple logic gates can be translated into an algebraic domain. SCA methods use the Gröbner basis for an ideal membership test to solve the verification task. In a general case, computing such a Gröbner Basis can be expensive, as they require exponential space in respect to the number of variables used [10]. However, restricted to integer arithmetic, the computation of a Gröbner basis can be achieved by substitution of variables if an appropriate term order is used.

This restriction is used in [8, 9, 12, 13] for the verification of (non-)restoring dividers using backward rewriting, as shown in Section 3.4, in reverse topological order. In the context of dividers, the polynomial $Q \cdot D + R$ is used [12]. For a correct divider, this specification polynomial reduces to the initial input dividend $R^{(0)}$. In SCA methods it is more common to represent this as $Q \cdot D + R - R^{(0)}$ having to reduce to 0. We will call this $vc1$. A correct divider further has to fulfil the condition $0 \leq R < D$ which we call $vc2$. This condition is harder to verify using SCA. We will discuss methods of verifying $vc2$ in Section 4.4.

Using SCA straightforward has some flaws however. Firstly, it is possible that backward rewriting is not applied on the boundaries between stages of the divider. Secondly, as reported by [12], there can be exponential blow-ups of the polynomial size in between stages of the divider. To address these issues, the authors of [12] propose *SAT based information forwarding.*

4.1 SAT-Based Information Forwarding

SAT based information forwarding (SBIF) [12] aims to simplify the polynomials as early as possible by computing don't cares using SAT solvers. For SBIF the SAT based information propagation is restricted to equivalences and antivalences between pairs of signals a and b. For the circuit CUV, which is being verified, a constraint C is used. In the case of a non-restoring divider, the authors of [12] use $C = (0 \leq R^{(0)} < D \cdot 2^{n-1})$.

The proposed algorithm (see Algorithm 3) first simulates the circuit using input vectors satisfying C, considering both $sim(a)$ and the negation $sim(\bar{a})$ to take antivalences into account. To verify the found equivalences and antivalences, the set of input signals is first divided into a set of equivalence classes. Following this, the signals are processed in topological order and checked for equivalent/antivalent predecessors using the simulation vectors. To limit the scope of the SAT solver in larger circuits, the algorithm computes W_a and W_b up to the given maximal depth d_{max}, instead of considering the whole circuit. Then, using a SAT-Solver, the candidate is checked, and if it is a true equivalence or antivalence, the equivalence classes are updated.

The equivalence classes computed using SAT based information propagation are then given to the modified backward rewriting algorithm (see Algorithm 4), where signals in the specification polynomial are replaced by topologically minimal representatives in their equivalence classes. Further, before a signal is replaced with its gate polynomial, the signals in the gate polynomial are also replaced by the topologically minimal representatives in their equivalence classes.

This method can derive that in each stage of the divider the most significant bits are antivalent

Algorithm 3 SAT based information propagation [12]

```
Input: Input constraint $C$.
       Circuit $CUV$ with set of signals $S = a_1, ..., a_n$.
       Maximal window depth $d_{max}$.
Output: Partition $E$ of signals (or their negations) into equivalence classes.
 1: Choose set $V$ of input vectors satisfying $C$;
 2: Simulate $CUV$ with vectors from $V$ leading to simulation vectors $sim(a)$ and $sim(\bar{a})$ for
    each $a \in S$;
 3: Choose topological order $\prec_{top}$ for $S$;
 4: $E = \{\{a_1\}, ..., \{a_n\}\}$;
 5: for each $a \in S$ in topological order do
 6:     for each $b \prec_{top} a$ with $sim(a) = sim(b)$ or $sim(a) = sim(\bar{b})$ do
 7:         if $a \notin [b]$ then
 8:             for $s \in \{a, b\}$ do
 9:                 $W_s = \{g \mid g \in S$ is an $i-$step predecessor of $s$ with $i \leq d_{max}\}$;
10:             if $UNSAT(CNF(a \oplus b^{\varepsilon}, W_a, W_b, C))$ then
11:                 $E = (E \setminus \{[a], [b]\}) \cup \{[a] \cup [b^{\varepsilon}]\}$;
    return $E$;
```

and the stage implements the addition/subtraction under the constraint C [13]. By using this information, exponential blow-ups can be avoided. If however an optimized divider, as discussed in Section 3.3 is to be verified, the most significant bits are neither antivalent nor equivalent [13]. The authors of [13] show that this leads to exponential sizes in the canonical polynomials occurring between stages.

4.2 SCA with Don't Care Optimization

To address the exponential memory blow-ups encountered with an optimized non-restoring divider when using SBIF, the authors of [13] propose to extend SBIF with *don't care optimization*(DCO). In this method, atomic blocks like the original gates or non-trivial atomic blocks like full adders and half adders are detected from a gate level netlist.

Given all atomic blocks of the circuit, don't cares, resulting from the input constraint, for the inputs of all atomic blocks are computed. The authors of [13] found that using a SAT to compute this is infeasible, as unrestricted SAT is too computationally expensive and restricting the window size is also unsuccessful due to the inability of confirming don't cares using local reasoning. Instead, a series of BDD-based image computations, based on [4], is used.

Following this, equality classes are computed using SBIF. Since SBIF fails for the optimized non-restoring divider shown in Figure 3, the information that signal combinations of the computed satisfiability don't cares cannot occur is added to the SAT problems. Similarly to rewriting only using SBIF, the atomic blocks are processed in reverse topological order. The computed don't cares and representatives are then used to optimize the polynomial if its size grew faster than a given threshold.

For this, backtracking points are created and pushed onto a stack, if there is no significant

Algorithm 4 (Simplified) backward rewriting using SBIF [12]

```
Input: Circuit $CUV$ with topological order $\prec_{top}$ on signals.
       Let $E = \{e_1, ..., e_m\}$ be returned by Algorithm 3.
       $\forall 1 \leq i \leq m$ let $r_i^{\varepsilon_{r_i}} \in e_i$ with $r_i$ minimal wrt. $\prec_{top}$.
       Let $r_1 \prec_{top} ... \prec_{top} r_m, p_{r_i}$ is the gate polynomial of $r_i$.
Output: 1 iff specification holds;
 1: $SP_m := SP$;
 2: $i = m$;
 3: while $SP_m$ depends on $a$ with $a^{\varepsilon_a} \in e_j, a \neq r_j$ do
 4:     if $\varepsilon_a = \varepsilon_{r_j}$ then
 5:         $SP_m = SP_m|_{a \leftarrow r_j}$;
 6:     else
 7:         $SP_m = SP_m|_{a \leftarrow (1-r_j)}$
 8: while $i \geq 0$ do
 9:     while $p_{r_i}$ depends on $a$ with $a^{\varepsilon_a} \in e_j, a \neq r_j$ do
10:         if $\varepsilon_a = \varepsilon_{r_j}$ then
11:             $p_{r_i} = p_{r_i}|_{a \leftarrow r_j}$;
12:         else
13:             $p_{r_i} = p_{r_i}|_{a \leftarrow (1-r_j)}$
14:     $SP_{i-1} = SP_i|_{r_i \leftarrow p_{r_i}}$;
15:     $i = i - 1$;
    return $SP_0 = 0$;
```

Algorithm 5 Backward rewriting with don't care optimization [13]

```
Input: specification polynomial $SP^{init}$.
       Input constraint $IC$.
       Circuit $CUV$ with atomic blocks $a_1 \prec_{top} ... \prec_{top} a_m$ in topological order $\prec_{top}$ on signals.
       $\forall 1 \leq i \leq m$ let $r_i^{\varepsilon_{r_i}} \in e_i$ with $r_i$ minimal wrt. $\prec_{top}$ and signals $s_1, ..., s_n$.
Output: 1 iff specification holds for all inputs satisfying $IC$.
 1: $SP_m := SP^{init}$; oldsize $:= size(SP_m)$; $i := m$; $ST := \emptyset$, an empty stack;
 2: $(dc(a_1), ..., dc(a_m)) :=$ Compute_DC$(CUV, IC)$;
 3: $(rp(s_1), ..., rp(s_n)) :=$ SBIF$(CUV, IC, (dc(a_1), ..., dc(a_m)))$, see Algorithm 3;
 4: while $i > 0$ do
 5:     $SP_{i-1} :=$ Rewrite$(SP_i, a_i)$;
 6:     if size$(SP_{i-1}) > threshold \cdot oldsize$ and $ST \neq \emptyset$ then
 7:         $(SP, j, type) := pop(ST)$;
 8:         $i := j$;
 9:         $SP_{i-1} := SP$;
10:         if $type = dc$ then $SP_{i-1} :=$ Opt_DC$(SP_{i-1}, dc(a_i))$;
11:         if $type = eq$ then $\forall s \in$ inputs of $a_i$ : $SP_{i-1} :=$ Replace$(SP_{i-1}, s, rp(s))$;
12:     else
13:         if $dc(a_i) \neq \emptyset$ then
14:             push$(ST, (SP_{i-1}, i, dc))$;
15:             $oldsize :=$ size$(SP_{i-1})$;
16:         if $\exists s \in$ inputs of $a_i$ with $rp(s) \neq s$ then
17:             push$(ST, (SP_{i-1}, i, eq))$;
18:             $oldsize :=$ size$(SP_{i-1})$;
    return evaluate$(SP_0)$, see Algorithm 6;
```

Algorithm 6 evaluate(SP_0) [13]

```
 1: for $i = n - 2$ to 0 do
 2:     if $SP_0|_{d_i=1, r_{i+n-1}^{(0)}=0} \neq 0$ then return 0;
 3:     $SP_0 := SP_0|_{r_{i+n-1}^{(0)}=d_i}$;
    return 1;
```

increase in the size of the polynomial and either the don't care set for the processed atomic block is not empty or there is a topologically smaller representative for an input signal of the atomic block. If the size of the polynomial grows by a factor larger than the threshold, the last backtracking point is popped from the stack and applied. In the case of n equality for an input signal of the atomic block, it is replaced by its topologically minimal representative, as described in Section 4.1. Alternatively, if the backtrack point was created due to a non-empty don't care set, the polynomial is optimized as follows for a polynomial $P(x_1, ..., x_n)$ with don't care cubes $dc_1, ..., dc_n$ [13]:

- Introduce a new integer variable v_i for each don't care cube dc_i
- Add for all $1 \leq i \leq n$"$v_i \cdot dc_i$" to p
- Multiply out and combine terms with the same monomial etc.
- Use Integer Linear Programming to minimize the size of P

This intends to find an assignment of don't cares which minimizes the size of P.

Finally, the authors note that for optimized dividers, such as shown in Figure 3, the polynomial does not reduce to 0. Although this is contradictory to the initial goal of the circuit being correct iff the polynomial reduces to 0, it is not an error, as the circuit only has to be correct for the input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$. Thus, the circuit is correct iff the polynomial reduces to 0 for all inputs satisfying this constraint [13]. To evaluate these cases, $R^{(0)}$ and $D \cdot 2^{n-1}$ are compared bitwise starting with the most significant bit. Algorithm 5 describes the overarching algorithm in detail.

Although this method drastically improves on SBIF [12], as demonstrated by the benchmarks shown in Section 6, its success is partially due to the following reasons, as stated by the authors of [8, 9]:

1. The dividers used have a number of atomic blocks that have any satisfiability don't cares that grows linearly with the bit width
2. Only a linear amount of backtracking is needed
3. If backtracking has to be used, don't care assignments have an essential effect in keeping polynomials small

The authors of [8, 9] however manage to construct a divider circuit which does not fulfil these characteristics and propose *delayed don't care optimization.*

4.3 Delayed Don't Care Optimization

Delayed don't care optimization (DDCO) aims to increase the robustness of don't care optimization by implementing two additions:

- Instead of optimizing don't cares over atomic blocks, fanout-free extended atomic blocks (EABs) are used.
- Backtracking is no longer used, but don't care information about an EAB is only used after a set delay of rewriting steps.

Further, this method no longer uses SBIF to propagate information regarding equivalences and antivalences.

The notion of using EABs is based on the work of [6, 11], where gates and atomic blocks are combined into fanout-free cones, meaning that every output from an EAB is connected to the input of at most one logic gate. For these, polynomials are computed and used in backward rewriting. The authors of [8, 9] then use these fanout-free cones to find better don't cares. Extended atomic blocks are computed using a directed graph based approach based on the atomic blocks found using the method presented by [13]. The nodes of the graph represent atomic blocks, outputs or gates which are not part of any atomic block. Iff there is an output from the block or gate represented by node n connected to an input of a block, gate or output represented by the node m, an edge is added from n to m. On this graph, the coarsest partition $\{P_1, ..., P_l\}$ fulfilling that, for all $P_i : \forall n \in P_i$ with more than one successor it holds that all successors of n are not in P_i [8, 9], is computed. To generate the extended atomic block ea_i, all gates and atomic blocks of set P_i are combined. Meaning that for every node in P_i (and thus ea_i) that has any output with a fanout larger than one, the node with the input to which this output is connected is not included in the extended atomic block ea_i. Using these EABs, satisfiability don't cares are computed using BDD based image computations similarly to [13].

The authors of [8, 9] further reason that don't care optimization only optimizes the size of the polynomial locally, but future sizes depend on future substitutions and local don't care optimization may lead to worse substitutions later. Instead of local don't care optimization, they propose delayed don't care optimization, for which they show that

> *Delayed don't care optimization* ***can*** *be exponentially better than local don't care optimization(even for a delay by only one rewriting step).*

As already mentioned, DDCO does not use backtracking like [13], but delays the don't care optimization by d rewriting steps. Algorithm 7 shows the detailed algorithm.

4.4 Verifying $0 \leq R < D$

Until now, we have only discussed how each method can be used to verify $vc1$ (see Definition 1), but a correct divider also needs to fulfil $vc2 = 0 \leq R < D$ [12]. Due to the exponential size of the polynomial representing $vc2$, backward rewriting is impractical. However, since there is a BDD of linear size representing this constraint, the authors of SBIF [12] propose to perform backward substitution on it. This approach can further be integrated into the BDD based image computation for (delayed) don't care optimization [13].

Algorithm 7 Backward rewriting with delayed don't care optimization [8, 9]

```
Input: specification polynomial $SP^{init}$.
       Input constraint $IC$.
       Circuit $CUV$ with EABs $ea_1 \prec_{top} ... \prec_{top} ea_m$ in topological order $\prec_{top}$ on signals.
       EABs $ea_i$ with input signals $x_i^{(i)}, ..., x_{n_i}^{(i)}$.
       Don't cares $dc(ea_i) = \{(\varepsilon_{1,1}^{(i)}, ..., \varepsilon_{1,n_i}^{(i)}), ..., (\varepsilon_{l_i,1}^{(i)}, ..., \varepsilon_{l_i,n_i}^{(i)})\}$.
       Delay $d$.
Output: 1 iff specification holds for all inputs satisfying $IC$.
 1: $SP_m := SP^{init}; i := m;$
 2: while $i - 1 > 0$ do
 3:     $i := i - 1;$
 4:     $SP_{i-1} :=$ Rewrite($SP_i, ea_i$);
 5:     for $j = 1$ to $l_i$ do
 6:         $SP_{i-1} := SP_{i-1} + v_j^{(i)} \cdot \prod_{e_{j,k}^{(i)}=1} x_k^{(i)} \cdot \prod_{e_{j,k}^{(i)}=0} (1 - x_k^{(i)});$
 7:     if $i + d > m$ then
 8:         continue;
 9:     $SP_{i-1}^{tmp} :=$ assign_dc($SP_{i-1}, v_1^{i+d-1} = 0, ..., v_{l_i}^{i} = 0$);
10:     dc0_size := size(assign_dc($SP_{i-1}^{tmp}, v_1^{i+d} = 0, ..., v_{l_{i+d}}^{i+d} = 0$));
11:     if dc0_size $\leq$ increase(size($SP_{i+d}$)) then
12:         for $j = i - 1$ to $i + d - 1$ do
13:             $SP_j :=$ assign_dc($SP_j, v_1^{i+d} = 0, ..., v_{l_{i+d}}^{i+d} = 0$);
14:     else
15:         $(z_1^{i+d}, ..., z_{l_{i+d}}^{i+d}) :=$ DC_opt($SP_{i-1}^{tmp}$);
16:         for $j = i - 1$ to $i + d - 1$ do
17:             $SP_j :=$ assign_dc($SP_j, v_1^{i+d} = z_1^{i+d}, ..., v_{l_{i+d}}^{i+d} = z_{l_{i+d}}^{i+d}$);
18: $SP_0 :=$ assign_dc($SP_0, v_1^{(d)} = 0, ..., v_{l_1}^{(d)} = 0$);
19: return evaluate($SP_0$), see Algorithm 6;
```

5 Formal Verification Using Hardware Reduction

All methods examined thus far, except for Delayed Don't-Care Optimization (DDCO) [8,9], fail when applied to restoring divider circuits [9]. To address this challenge, Yasin et al. [14] propose a fundamentally different approach based on hardware reduction. This method leverages logic synthesis as a verification tool, rather than relying solely on algebraic or SAT-based reasoning.

The central idea is to append the divider under verification with an auxiliary circuit that computes the inverse arithmetic operation, namely

$$Z = Q \cdot D + R$$

for dividers, and to check whether this reconstructed value matches the dividend $R^{(0)}$. Then $Z = R^{(0)}$ holds for a correct divider under the constraints $0 \leq R < D$. A naive equivalence check can be formulated by constructing a circuit that appends the computation of Z to the outputs of the divider and solving it using SAT. However, experiments by [14] have shown that SAT quickly becomes impractical for dividers with dividend widths beyond 16 bits, due to excessive computation times.

Instead, [14] propose *hardware reduction*, subjecting the composed circuit to logic synthesis. A correct divider circuit then reduces to wires and buffers connecting the inputs of the divider to the outputs of the inverse circuit. After this, it is sufficient, that each bit $r_i^{(0)}$ of the dividend $R^{(o)}$ is equal to the corresponding bit z_i of Z.

When applied to the entire divider at once, this method still suffers from excessive runtimes for dividends of a bit-width larger than 20 bits [14]. To overcome this, [14] propose a layered approach, in which verification is performed independently on each layer j of the restoring divider array. For the following, we will still use the notation introduced in Section 3.1. I.e. At layer j the inputs are the partial remainder bits $R_{j-1}^{(j-1)}$ output from the previous layer, divisor D and the outputs consist of quotient bit q_{n-j} and partial remainder bits $R_j^{(j)}$, where $R_i^{(i)} = (r_{2n-1-i}^{(i)}, ..., r_{n-i}^{(i)})$. To verify that $R^{(j)} = R_{j-1}^{(j-1)} - q_{n-j} \cdot D$, an additional circuit computing

$$Z_j = R_j^{(j)} + q_{n-j} \cdot D$$

is attached to the outputs of layer j and synthesis is applied to the combined circuit. If the divider layer is correct, the synthesized circuit reduces to wires or buffers connecting Z_j and $R^{(j)}$. Thus, the verification of the divider reduces to proving redundancy for each layer separately.

However, it is further necessary to ensure that the divider still fulfils $vc2$ (see Definition 2). This can be done on each layer by checking the constraint

$$R^{(j)} = R^{(j-1)} - D \cdot 2^{n-j}$$

This leads to the constraint

$$R^{(j)} < 2^{n-j-1}D$$

which implies for the bottom most layer with $j = n - 1$ that

$$R < D$$

To verify this, each layer is further appended by a comparator circuit that implements

$$R^{(j)} \geq 2^{n-j-1}D$$

Then, SAT can be used to check this comparator for unsatisfiability.

An advantage provided by the layer based approach is that it can be parallelized since the verification of each layer is independent of the other layers. Further, by separately verifying each layer, a bug in a layer can be isolated for debugging.

6 Discussion

In this section we compare the methods for verifying restoring and non-restoring dividers, which we introduced to the reader. For this, we will be using the benchmark results from [9, 12] and [14]. The results from [13] are left out, as the same methods was evaluated by [8, 9] again with similar results.

The experiments for all symbolic computer algebra methods where run on an Intel Xeon E5-2643 running at 3.3 GHz and 62 GiB of main memory [8,9,12]. For [13] and [9], the solver Gurobi was used for solving ILP problems and CUDD 3.0.0 for BDD based image computations. For the experiments of [14], an Intel Core i7-6700U running at 2.80 GHz with 30 GiB of main memory was used. A combination of Synopsis Design Compiler and ABC were used for synthesis. Although the experimental setups are not equal, we expect both systems to perform roughly within the same order of magnitude in single threaded performance, based on the date of release, used architecture and vendor specifications. Because of this, we conclude, that the results from [9,12] and [14] can be compared qualitatively. All tables additionally show the time needed by MiniSat 2.2.0 to solve the corresponding satisfiability problems, as reported by [8], in order to represent previous methods.

As [14] only present results for the verification of restoring dividers in respect to the size of the dividend, we have transformed the benchmark results in respect to the size n of the divisor. If, for a given divisor size, no runtime was given by [14] for an equivalent dividend size, the next larger dividend size provided by [14] is given and marked by $\leq$. If no such data for a larger dividend was available, the entry is marked "NA". Similarly, "MO" is used when the main memory of 62 GiB was exceeded by [9, 12, 14] and "TO" for runtimes exceeding 24 CPU hours.

Table 2 shows the runtimes of [9, 12, 13] for the verification of an optimized non-restoring divider, which still includes the computation of the MSB in the bottom most row of the divider. As can be seen, SCA with SBIF already fails at verifying such divider with a width of 16 bits due to excessive memory usage. This issue is remedied by the addition of *don't care optimization*, as was intended by the authors of [13]. However, it might be surprising that the most recent method using *extended atomic blocks* and *delayed don't care optimization* is slower than just using SBIF with DCO. This is due to there being more blocks where don't cares are applicable when using EABs, while the necessary amount of don't cares to apply does not change [9]. Still, SCA with DCO and EABs with DDCO vastly outperform MiniSat.

n	MiniSat	SCA+SBIF [12]	SCA+SBIF+DCO [13]	SCA+EABs+DDCO [8, 9]
4	0.22	0.16	0.15	0.23
8	68.58	904.30	0.39	0.94
16	TO	MO	1.59	1.87
32	TO	MO	5.06	6.78
64	TO	MO	21.88	28.24
128	TO	MO	114.73	153.71
256	TO	MO	825.11	1985.05
512	TO	MO	9183.28	27370.60

Table 2: SCA based method verification time in seconds for a non-restoring divider with partial optimization, as depicted in Figure 3, including the gray adder

n	MiniSat	SCA+SBIF+DCO [13]	SCA+EABs+DDCO [8, 9]
4	0.23	0.17	0.23
8	31.83	2486.89	0.95
16	TO	MO	2.17
32	TO	MO	7.25
64	TO	MO	26.87
128	TO	MO	149.75
256	TO	MO	1691.72
512	TO	MO	27351.10

Table 3: SCA based method verification time in seconds for a non-restoring divider with full optimization depicted in Figure 3, without the gray adder

The improvement of [8, 9] is however clearly shown in Table 3, where the verification times for a divider excluding the gray full adder in Figure 3 are presented. While SCA with SBIF and DCO fails and is outperformed by MiniSat for this further optimized divider at $n = 8$, SCA with EABs and DDCO performs similarly to the verification of the less optimized divider. Additionally, as shown in Table 4, the method from [8, 9] is able to verify a restoring divider with a divisor of 256 bits without a memory overflow and timing out at 512 bits.

For the verification of restoring dividers, verification by hardware reduction shows even more promising results, outperforming SCA based methods by an order of magnitude. However, for the correct verification using hardware reduction, the inverse arithmetic circuit Z_i and the comparator circuit for $R^{(j)} \geq 2^{n-j-1}D$ are required. In order to guarantee correctness of the divider both of these circuits have to be correct. To our knowledge, the verification of both of these circuits is not considered by the authors of [14]. We however also believe that the verification of these circuits should be faster and easier than the verification of a divider circuit, as they only consist of a conditional addition and a comparison respectively. Additionally, the effectiveness of hardware reduction is strongly dependent on the logic synthesis tool used and the authors of [14] observed an area penalty to the synthesized circuit as high as 17%.

n	MiniSat	SCA+SBIF+DCO [13]	SCA+EABs+DDCO [8, 9]	Hardware Reduction [14]
4	0.27	2.59	0.38	≤ 0.21
8	14.88	MO	1.42	≤ 0.40
16	TO	MO	6.63	≤ 0.68
32	TO	MO	29.02	4.48
64	TO	MO	193.40	18.56
128	TO	MO	2244.24	123.46
256	TO	MO	33593.30	NA
512	TO	MO	TO	NA

Table 4: Verification times in seconds for a restoring divider

7 Conclusion

The verification of restoring and non-restoring dividers still proves to be a difficult problem to solve. We have introduced the reader to two different approaches for the automated verification of hardware dividers.

First, we provided an overview over restoring and non-restoring division and introduced the reader to backward rewriting. We further summarized three symbolic computer algebra methods and conclude that, based on the results presented by their authors, we believe that the SCA based methods are a powerful tool for the verification of non-restoring dividers. However, there is a possibility that not yet known or untested optimizations in the design of the divider may lead to exponentially worse memory usage or computation times for SCA based methods, as was already shown for SBIF [12] and don't care optimization [14] by [8, 9, 12]. This issue might further be exasperated if the available main memory is limited. While SCA with extended atomic blocks and delayed don't care optimization is even capable of verifying restoring dividers without excessive memory usage, it suffers from long computation times. Even though using extended atomic blocks and delayed don't care optimization can lead to longer verification times than using SAT based information forwarding with don't care optimization, we believe that the increased robustness makes it a better choice for the verification of non-restoring dividers.

Additionally, we introduced the reader to a novel approach for verification, called *hardware reduction*. This method proved powerful for the verification of a restoring divider, if a few conditions are fulfilled:

- The divider circuit is easy to modify
- A powerful logic synthesis program is available
- Correct circuits for the comparator and inverse arithmetic operation are available

Although this method is promising, it is unproven for non-restoring dividers and the verification time of the inverse circuit also needs to be considered. More research into verification by hardware reduction is required to judge its potential.

These recent developments however mark an important step towards the automated formal verification of divider circuits, improving on existing methods by an order of magnitude. Looking ahead, SCA based methods are promising for a method enabling the verification of both restoring and non-restoring dividers. At the same time, hardware reduction may also be very effective, but requires further researches to be proven truly effective for both divider architectures.

References

[1] Randal E. Bryant & Yirng-An Chen (1995): *Verification of arithmetic circuits with binary moment diagrams.* In: *Proceedings of the 32nd ACM/IEEE conference on Design automation conference - DAC '95*, ACM Press, San Francisco, California, United States, pp. 535–541, https://doi.org/10.1145/217474.217583. Available at http://portal.acm.org/citation.cfm?doid=217474.217583.

[2] Maciej Ciesielski (2023): *Formal Methods in Arithmetic Circuit Verification: a Brief History and Challenges.*

[3] Maciej Ciesielski, Cunxi Yu, Walter Brown, Duo Liu & André Rossi (2015): *Verification of gate-level arithmetic circuits by function extraction.* In: *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, Association for Computing Machinery, New York, NY, USA, pp. 1–6, https://doi.org/10.1145/2744769.2744925. Available at https://dl.acm.org/doi/10.1145/2744769.2744925.

[4] Olivier Coudert & Jean Christophe Madre (2003): *A Unified Framework for the Formal Verification of Sequential Circuits.* In Andreas Kuehlmann, editor: *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*, Springer US, Boston, MA, pp. 39–50, https://doi.org/10.1007/978-1-4615-0292-0_4. Available at https://doi.org/10.1007/978-1-4615-0292-0_4.

[5] Jiteshri Dasari & Maciej Ciesielski (2023): *Formal Verification of Restoring Dividers made Fast and Simple.* In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, https://doi.org/10.1109/DAC56929.2023.10247789. Available at https://ieeexplore.ieee.org/document/10247789/.

[6] Farimah Farahmandi & Bijan Alizadeh (2015): *Groebner basis based formal verification of large arithmetic circuits using Gaussian elimination and cone-based polynomial extraction.* *Microprocessors and Microsystems* 39(2), pp. 83–96, https://doi.org/10.1016/j.micpro.2015.01.007. Available at https://www.sciencedirect.com/science/article/pii/S0141933115000083.

[7] K. Hamaguchi, A. Morita & S. Yajima (1995): *Efficient construction of binary moment diagrams for verifying arithmetic circuits.* In: *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 78–82, https://doi.org/10.1109/ICCAD.1995.479995. Available at https://ieeexplore.ieee.org/abstract/document/479995. ISSN: 1092-3152.

[8] Alexander Konrad, Christoph Scholl, Alireza Mahzoon, Daniel Große & Rolf Drechsler (2022): *Divider Verification Using Symbolic Computer Algebra and Delayed Don't Care Optimization.* In: *2022 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–10, https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_17. Available at https://ieeexplore.ieee.org/document/10026567/. ISSN: 2708-7824.

[9] Alexander Konrad, Christoph Scholl, Alireza Mahzoon, Daniel Große & Rolf Drechsler (2024): *Divider verification using symbolic computer algebra and delayed don't care optimization: theory and practical implementation.* *Formal Methods in System Design*, https://doi.org/10.1007/s10703-024-00452-3. Available at https://doi.org/10.1007/s10703-024-00452-3.

[10] Ernst W. Mayr (1997): *Some Complexity Results for Polynomial Ideals.* *Journal of Complexity* 13(3), pp. 303–325, https://doi.org/10.1006/jcom.1997.0447. Available at https://www.sciencedirect.com/science/article/pii/S0885064X97904477.

[11] Amr Sayed-Ahmed, Daniel Große, Ulrich Kühne, Mathias Soeken & Rolf Drechsler (2016): *Formal verification of integer multipliers by combining Gröbner basis with logic reduction.* In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1048–1053. Available at https://ieeexplore.ieee.org/abstract/document/7459464. ISSN: 1558-1101.

[12] Christoph Scholl & Alexander Konrad (2020): *Symbolic Computer Algebra and SAT Based Information Forwarding for Fully Automatic Divider Verification.* In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, `https://doi.org/10.1109/DAC18072.2020.9218721`. Available at `https://ieeexplore.ieee.org/document/9218721/`. ISSN: 0738-100X.

[13] Christoph Scholl, Alexander Konrad, Alireza Mahzoon, Daniel Große & Rolf Drechsler (2021): *Verifying Dividers Using Symbolic Computer Algebra and Don't Care Optimization.* In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1110–1115, `https://doi.org/10.23919/DATE51398.2021.9474019`. Available at `https://ieeexplore.ieee.org/abstract/document/9474019`. ISSN: 1558-1101.

[14] Atif Yasin, Tiankai Su, Sebastien Pillement & Maciej Ciesielski (2023): *Formal Verification of Divider Circuits by Hardware Reduction.* In: *2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 1–4, `https://doi.org/10.1109/SMACD58065.2023.10192137`. Available at `https://ieeexplore.ieee.org/document/10192137/`. ISSN: 2575-4890.

[15] Atif Yasin, Tiankai Su, Sébastien Pillement & Maciej Ciesielski (2019): *Formal Verification of Integer Dividers:Division by a Constant.* In: *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 76–81, `https://doi.org/10.1109/ISVLSI.2019.00022`. Available at `https://ieeexplore.ieee.org/document/8839474/`. ISSN: 2159-3477.

[16] Atif Yasin, Tiankai Su, Sébastien Pillement & Maciej Ciesielski (2019): *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, `https://doi.org/10.1109/VLSI-SoC.2019.8920335`. Available at `https://ieeexplore.ieee.org/document/8920335/`. ISSN: 2324-8440.