Improving and Extending the Algebraic Approach for Verifying Gate-Level Multipliers

Daniela Ritirc Armin Biere Manuel Kauers
Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
daniela.ritirc@jku.at armin.biere@jku.at manuel.kauers@jku.at

***Abstract*—The currently most effective approach for verifying gate-level multipliers uses Computer Algebra. It reduces a word-level multiplier specification by a Gröbner basis derived from a gate-level implementation. This reduction produces zero if and only if the circuit is a multiplier. We improve this approach by extracting full- and half-adder constraints to reduce the Gröbner basis, which speeds up computation substantially. Refactoring the specification in terms of partial products instead of inputs yields further improvements. As a third contribution we extend these algebraic techniques to verify the equivalence of bit-level multipliers without using a word-level specification.**

I. Introduction

Arithmetic circuit verification is still considered a challenge, even after more than 20 years after the Pentium bug [15]. Currently the most effective approach for gate-level arithmetic circuit verification uses computer algebra [15], [16], [21].

The authors of [21] use a method called function extraction where the output signature is reduced by a Gröbner basis to an input signature and then compared to a specification. Independently [16] presents optimizations which rewrite and simplify the Gröbner basis. Based on these ideas we introduced in [15] a column-wise checking which cuts the circuit logic into slices and verifies correctness incrementally. Related work [11], [14] uses algebraic techniques over finite fields to verify different arithmetic circuits (Galois field multipliers). We focus on integer multiplier circuits, as in [15], [16], [21]. Our first contribution refines the approach of [15] by identifying full- and half-adders in the circuit in order to rewrite the Gröbner basis. As second contribution we observed that factoring out partial products is beneficial in this context too.

Alternatively it is also common to apply gate-level equivalence checking, which uses a reference circuit, instead of requiring a word-level specification. In [18] equivalence checking of multiplier circuits on the gate-level is achieved by first extracting half-adder circuits ("addition graphs") from the accumulation of partial products. Equivalence of these extracted half-adder circuits is checked by a dedicated procedure. Proofs of soundness and completeness are lacking.

More recently [17] proposes an algebraic variant of combinational equivalence checking, also based on Gröbner basis theory. It is similar to SAT sweeping [10], and compares circuits bit-wise, e.g., output by output, again without soundness nor completeness proof. As third contribution we present a new algebraic approach, an extension of [15], which incrementally shows equivalence of two arbitrary gate-level circuits in a column-wise fashion, and prove soundness and completeness.

This work is supported by Austrian Science Fund (FWF), first two authors by NFN S11408-N23 (RiSE), third author by Y464-N18 and SFB F5004.

II. Algebra

As in [11], [14], [15], [17], [21] we describe each gate in the circuit and its specification with multivariate polynomials. We prove correctness of a circuit by showing that the specification of the circuit is implied by the gate polynomials. We need facts of the theory of Gröbner basis [3], [4], [5], [15]:

- The ring $\mathbb{Q}[X]$ contains all polynomials in variables $X = x_1, \ldots, x_n$ with coefficients in $\mathbb{Q}$.
- A polynomial is a finite sum of monomials. A monomial is a constant multiple of a term, where a term is a power product $x_1^{u_1} \cdots x_n^{u_n}, u_i \in \mathbb{N}$, over the variables X.
- An order $\leq$ is fixed on the set of terms such that $1 \leq \tau$ and $\sigma_1 \leq \sigma_2 \Rightarrow \tau\sigma_1 \leq \tau\sigma_2$ for all terms τ, σ_1, σ_2.
- An order is a *lexicographic term order* if for all terms $\sigma_1 = x_1^{u_1} \cdots x_n^{u_n}$, $\sigma_2 = x_1^{v_1} \cdots x_n^{v_n}$ we have $\sigma_1 < \sigma_2$ iff there exists i with $u_j = v_j$ for all $j < i$, and $u_i < v_i$.
- The largest term (w.r.t. $\leq$) in a polynomial $p \in \mathbb{Q}[X] \setminus \{0\}$ is the leading term $lt(p)$. The leading coefficient $lc(p)$ and leading monomial $lm(p)$ are defined accordingly.
- A subset $I \subseteq \mathbb{Q}[X]$ is an ideal if (i) $0 \in I$, (ii) if $f, g \in I$, then $f + g \in I$, (iii) if $f \in I, h \in \mathbb{Q}[X]$, then $hf \in I$.
- A set $\{p_1, \ldots, p_m\} \subseteq \mathbb{Q}[X]$ is called a basis of an ideal I, if $I = \{q_1 p_1 + \cdots + q_m p_m \mid q_1, \ldots, q_m \in \mathbb{Q}[X]\}$. This is denoted by $I = \langle p_1, \ldots, p_m \rangle$.
- A basis $\{g_1, \ldots, g_m\}$ of an ideal I is said to be a Gröbner basis (w.r.t. $\leq$) if $\langle lt(g_1), \ldots, lt(g_m) \rangle = \langle lt(I) \rangle$.
- Every ideal of $\mathbb{Q}[X]$ has a Gröbner basis. Given an arbitrary basis of an ideal, Buchberger's algorithm computes a Gröbner basis for it in a finite number of steps.
- If for all polynomials in the basis $\{p_1, \ldots, p_m\} \subseteq \mathbb{Q}[X]$ of ideal I it holds that $lcm(lm(p_i), lm(p_j)) = lm(p_i) \cdot lm(p_j)$ (with lcm=least common multiple) then $\{p_1, \ldots, p_m\}$ is a Gröbner basis of I.

In combination with a multivariate version of polynomial division, the theory of Gröbner bases allows to answer the question whether a polynomial $q \in \mathbb{Q}[X]$ is an element of an ideal $I = \langle G \rangle = \langle g_1, \ldots, g_m \rangle \subseteq \mathbb{Q}[X]$:

- The remainder r of the division of q by G is a polynomial such that $q - r \in I$ and no term in r can be divided by any leading term of G. We say r *is reduced* w.r.t. G.
- If G is a Gröbner basis for I, then $q \in I$ iff r is zero.

III. Circuit Translation

Following [15] we consider multiplier circuits with $2n$ boolean inputs $a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}$ and $2n$ outputs $s_0, \dots, s_{2n-1}$. Each internal gate (output) is represented by a gate variable, denoted $g_0, \dots, g_k$. In this case we assume $X = a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}, g_1, \dots, g_k, s_0, \dots, s_{2n-1}$.

The structure of the circuit is not allowed to include any cycles. We relate the output (gate variable) u of a gate to its input(s) v, w by one of the following polynomials:

$$\begin{array}{llll} u = \neg v & \text{implies} & -u + 1 - v & = 0 \\ u = v \wedge w & \text{implies} & -u + vw & = 0 \\ u = v \vee w & \text{implies} & -u + v + w - vw & = 0 \\ u = v \oplus w & \text{implies} & -u + v + w - 2vw & = 0 \end{array} \qquad (1)$$

All variables are supposed to range over boolean values. Thus we have the relation $u(u-1) = 0$. The polynomials on the left hand side of this equation encoding this assumption are called *field polynomials F*.

Definition 1. [15] Let C be a circuit. Let $G \subseteq \mathbb{Q}[X]$ be the set containing for each gate of C the corresponding polynomial of (1) (with u, v, w replaced by the variables of the edges attached to the gate), as well as the polynomials $a_i(a_i - 1)$ and $b_i(b_i - 1)$ for $0 \le i < n$, called *input field polynomials*. Then the ideal generated by G in $\mathbb{Q}[X]$ is denoted by $J(C)$.

We use the last item in the list summarizing Gröbner basis theory to generate a Gröbner basis for $J(C)$:

Theorem 1. [19], [11], [15] Let C be a circuit, G as in Def. 1. Let $\le$ be a lexicographic term order for a variable order such that the variable attached to the output edge of a gate is always greater than all the variables attached to the input edges of that gate. Then G is a Gröbner basis with respect to $\le$.

Definition 2. [15] Let C be a circuit. A polynomial $p \in \mathbb{Q}[X]$ is called a *polynomial circuit constraint* (PCC) for C if for every choice of

$$(a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}) \in \{0,1\}^{2n}$$

and resulting values $g_1, \dots, g_k, s_0, \dots, s_{2n-1}$ implied by the gates of C the substitution of these values into p gives zero.

The following corollary derived from [15] shows that $J(C)$ contains all relations among the variables in the circuit C.

Corollary 1. Let $p \in \mathbb{Q}[X]$ then p is a PCC $\Leftrightarrow p \in J(C)$.

Thus a given word-level specification holds for C iff it is a member of the ideal $J(C)$. Since we know how to derive a Gröbner basis of $J(C)$ we can decide membership using multivariate division. Because of Cor. 1 such a verifier is sound and complete. We are particularly interested in relations between circuit outputs and inputs.

Definition 3. [15] A circuit C is called a *multiplier* if

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \quad \text{is a PCC.}$$

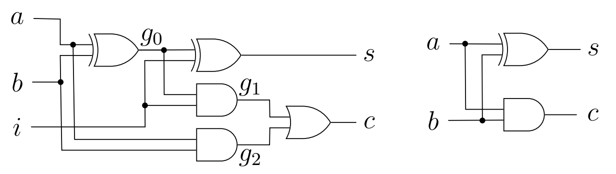

Fig. 1. Full-adder circuit (left) and half-adder circuit (right)

In this and in the following section we focus on multipliers. However, the theory presented in this section can easily be generalized to arbitrary acyclic circuits. This generalization is applied in Sect. VI to miters of multipliers.

IV. Adder Rewriting

Simply reducing the specification w.r.t. the generated Gröbner basis is not efficient. In general the number of monomials in the results of intermediate reduction increases drastically [8], [9], [15].

Since the Gröbner basis of an ideal is not unique, we can try to improve the performance of reduction by using an alternative Gröbner basis. A natural candidate is the (unique) reduced Gröbner basis [5]. However, some simple experiments show that already computing this Gröbner basis for a 4-bit multiplier takes longer than 20 minutes and thus this notion is not useful in practice. In recent work [16] more efficient optimizations have been developed which only partially reduce the Gröbner basis. The authors of [15] also make use of these optimization originally proposed in [16].

On top of these optimizations [15] further introduced a column-based incremental checking algorithm which divides the overall Gröbner basis G of the multiplier circuit C into $2n$ smaller sliced Gröbner bases. Then the specification in Def. 3 is split incrementally into so-called carry recurrence relations, and it is shown that these carry recurrence relations hold, i.e., the reduction of their corresponding polynomials by the sliced Gröbner bases reduces to zero.

We enhance the technique from [15] by further rewriting the sliced Gröbner bases. We search for full- and half-adders in the gate-level representation of a multiplier and consider them as isolated circuits. We then apply variable elimination to replace all polynomials of internal gates of full- and half-adders by the specification of the corresponding adder. This results in a smaller and more compact Gröbner basis representation of the whole multiplier which speeds up the reduction procedure.

Definition 4. A circuit C is called a *full-adder* if

$$-2c - s + a + b + i \quad \text{is a PCC}$$

for outputs c, s and inputs a, b, i and a *half-adder* if

$$-2c - s + a + b \quad \text{is a PCC.}$$

Example 1 (Full-adder)**.** From the full-adder circuit A depicted in Fig. 1 we derive the following polynomials:

$$\begin{aligned} H = \{ & -c + g_1 + g_2 - g_1 g_2, \quad -s + g_0 + i - 2g_0 i, \\ & -g_2 + ab, \quad -g_1 + g_0 i, \quad -g_0 + a + b - 2ab \quad \} \end{aligned}$$

According to Thm. 1, H is a Gröbner basis for $J(A) = \langle H \rangle$ w.r.t. ordering $a < b < i < g_0 < g_1 < g_2 < s < c$.

The goal is to eliminate the polynomials for the internal variables g_0, g_1, g_2 and express the outputs s, c directly in terms of the adder inputs a, b, i. The following definition and theorem are instances of a more general theory in [5] specialized to our specific situation where R denotes the ring $\mathbb{Q}[a, b, i, s, c, g_0, g_1, g_2]$ and R_{elim} the ring $\mathbb{Q}[a, b, i, s, c]$.

Definition 5. [5] Given $J \subset R$ the *elimination ideal* J_{elim} is an ideal of R_{elim} defined by

$$J_{elim} = J \cap R_{elim}.$$

Theorem 2. [5] Let $J \subset R$ be an ideal and let H' be a Gröbner basis of J with respect to the ordering

$$a < b < i < s < c < g_0 < g_1 < g_2.$$

Then the set

$$H_{elim} = H' \cap R_{elim}$$

is a Gröbner basis of the elimination ideal J_{elim}.

Example 1 (continued)**.** In order to eliminate g_0, g_1, g_2, it is not sufficient to remove the polynomials involving these variables from H. Instead, we need to compute a second Gröbner basis H' with respect to an ordering as given in Thm. 2. Discarding from H' the polynomials involving g_0, g_1, g_2 gives

$$\begin{aligned} H_{elim} = \{ & -2c - s + a + b + i, \\ & -s + a + b + i - 2ab - 2ai - 2bi + 4abi \}. \end{aligned}$$

By Thm. 2, H_{elim} is a Gröbner basis for $J_{elim}(A)$.

Note that since we use $\mathbb{Q}$ as coefficient domain, we could as well make the first polynomial of H_{elim} monic by dividing it by -2. Not doing so has the advantage that we do not get fractional coefficients.

The Gröbner basis H_{elim} is not autoreduced, because the polynomial $-2c-s+a+b+i$ can be reduced by the polynomial with leading term s. Autoreduction would further rewrite and simplify the Gröbner basis H_{elim}. In the incremental approach [15] carries of full- and half-adders connect two slices and are multiplied by two before remainder computation. Therefore we do not apply autoreduction, since reducing by the specification of a full- and half-adder turns out to be more efficient, as we try to explain next.

In the Gröbner basis G the gates of a multiplier are reverse topologically ordered. Figure 2 shows a column-wise order [15] where the multiplier is partitioned into totally ordered slices and the gates within a slice are ordered reverse topologically. Assuming that the carry output c of a full- or half-adder is always larger than the sum output s, the intermediate reduction polynomial includes the terms $2c + s$ before c is reduced. Therefore s is canceled in parallel during reducing c by the specification of a full- or half-adder, i.e., the first polynomial $-2c - s + a + b + i$ in H_{elim}.

In a multiplier with only of full- and half-adders (such as the multiplier in Fig. 2) we never reduce the specification by

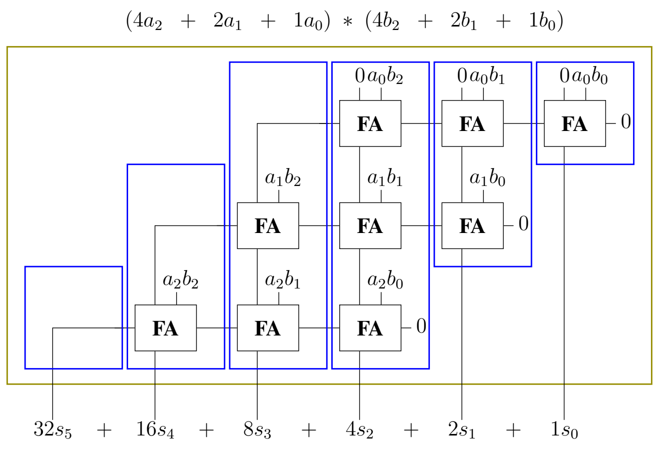

Fig. 2. Column-wise slicing for a 3-bit CSA multiplier as in [15].

the polynomials where the leading term is the sum output bit s of a full- or half-adder, as confirmed by our experiments.

Without sacrificing completeness, we can not delete these polynomials from the Gröbner basis, even though it further speeds up verification, again confirmed by the experiments.

We do not apply Thm. 2 globally on the Gröbner basis G of a multiplier, which would lead to the same computation issues as for deriving a reduced Gröbner basis of G. We apply Thm. 2 only locally on the extracted full-adders when the gate variables g_0, g_1, g_2 are used only internally in the full-adder, meaning no gate outside the full-adder uses one of these variables as input. We split the Gröbner basis G of the multiplier into disjoint sets G', G'' such that G' contains all polynomials including the internal variables and G'' contains all other polynomials. By splitting the Gröbner basis G we also split the ideal $J(C) = \langle G \rangle = \langle G' \cup G'' \rangle = \langle G' \rangle + \langle G'' \rangle$.

Since the internal variables do not occur in G'' it is not affected by variable elimination. We apply variable elimination on G' using Thm. 2 and the same ordering for G where the internal variables are moved to be the largest elements. This ensures that after variable elimination G' and G'' have the same ordering. Actually we will never execute the calculations for eliminating the internal variables of full-adders. We incrementally search for patterns as depicted in Fig. 1 and write down directly the polynomials derived for H_{elim} with corresponding input and output variables.

Deriving a Gröbner basis H'_{elim} for a half-adder circuit containing the specification $-2c - s + a + b$ can be obtained by exchanging the polynomial $-c + ab$ with the specification polynomial in the original Gröbner basis H'. Like for full-adders it can easily be checked that H'_{elim} is a Gröbner basis.

V. Partial Product Elimination

In multipliers with simple partial product generation, i.e. a partial product is the output of an AND-gate taking two circuit inputs as inputs, we find exactly n^2 polynomials in the Gröbner basis representing these (partial product) AND-gates.

To eliminate these polynomials from the Gröbner basis we cut off the partial product generation from the circuit and verify it separately. We do not include the corresponding polynomials $p_{i,j} = a_i b_j$ in the Gröbner basis. Instead we change the specification of the multiplier from Def. 3 to the specification stated in Cor. 2.

Corollary 2. A circuit C is a multiplier if

$$\sum_{i=0}^{2n-1} 2^i s_i - \sum_{i,j=0}^{n-1} 2^{i+j} p_{i,j} \in J(C) \quad \text{with} \quad p_{i,j} = a_i b_j.$$

It can easily be checked by expanding the sums in Def. 3 and replacing $p_{i,j}$ with $a_i b_j$ in Cor. 2, that they are equal.

In multipliers using Booth encoding [13] these polynomials can not be found and this technique is not applicable. It might be possible to find similar patterns in this situation too.

VI. Incremental Equivalence Checking

The goal of equivalence checking is to verify that two circuits C, C' produce the same output with regard to the same input. Equivalence checking can be used to verify circuits without requiring a word-level specification by showing that a circuit is equivalent to a "golden" reference circuit. We present an incremental equivalence checking technique based on [15], which divides the problem into smaller sub-problems.

This section is not limited to multiplier circuits. It also applies to all acyclic circuits under the condition that the inputs and the number of output bits of the circuits are the same. We generalize Sect. III as follows.

Consider C to be a circuit with l boolean inputs $a_0, \ldots, a_{l-1}$ and m outputs $s_0, \ldots, s_{m-1}$. Each internal gate (output) is represented by a gate variable $g_0, \ldots, g_j$. Further let C' be a circuit with m different outputs $s'_0, \ldots, s'_{m-1}$ but with the same l boolean inputs $a_0, \ldots, a_{l-1}$. Like for C each gate (output) in C' is represented by a gate variable $g'_0, \ldots, g'_k$. By $C \cup C'$ we denote the union of C and C' and as before Gröbner bases for C and C' can be derived.

Definition 6 (Equivalence Problem)**.** Let C, C' be two circuits. Then we say C and C' are equivalent written $C \equiv C'$ if

$$s_i - s'_i \qquad \text{is a PCC for all} \quad i = 0, \ldots, m-1.$$

Lemma 1. $\quad C \equiv C' \quad$ iff $\quad \sum_{i=0}^{m-1} 2^i (s_i - s'_i) \in J(C \cup C')$

Proof. Case "⇒": Follows from the definition of an ideal.
Case "⇐": Let $\varphi\colon X \to \mathbb{B} \subseteq \mathbb{Q}$ be an evaluation of all variables consistent with PCCs, i.e., circuit semantics, of C and C' as in Def. 2. Note that values of s_i, s'_i in $\mathbb{B}$ are uniquely determined for fixed values of inputs $a_0, \ldots, a_{l-1}$. The evaluation is extended to an evaluation of polynomials in the natural way (the unique homomorphic extension), i.e., $\varphi\colon \mathbb{Q}[X] \to \mathbb{Q}$. Since $\varphi(s_i), \varphi(s'_i) \in \mathbb{B}$ it is clear that $\varphi(s_i - s'_i) \in \{-1, 0, 1\}$.

Assume $C \not\equiv C'$, then there is a largest k with $0 \leq k < m$ and $\varphi(s_k - s'_k) \neq 0$, which gives the following contradiction

$$\begin{aligned} 0 &= \varphi(\sum_{i=0}^{m-1} 2^i (s_i - s'_i)) = \sum_{i=0}^{k} 2^i \varphi(s_i - s'_i) \\ &= \underbrace{2^k \varphi(s_k - s'_k)}_{\in\{-2^k, 2^k\}} + \underbrace{\sum_{i=0}^{k-1} 2^i \varphi(s_i - s'_i)}_{\in[-2^k+1, 2^k-1]} \neq 0 \end{aligned}$$

using $\sum_{i=0}^{m-1} 2^i (s_i - s'_i) \in J(C \cup C')$ for the first equation. □

This lemma gives us a word-level specification for gate-level equivalence checking. What remains is to derive a Gröbner basis for $J(C \cup C')$ as follows.

Lemma 2. Let C, C' be two circuits. Let G, G' be the Gröbner bases for the ideals $J(C), J(C')$ w.r.t. orderings $\leq, \leq'$, satisfying the conditions of Thm. 1. Let $\leq_\cup$ be a reverse topological order, such that $\leq, \leq'$ are contained in $\leq_\cup$. Then $G \cup G'$ is a Gröbner basis for $J(C \cup C')$ w.r.t. $\leq_\cup$.

Proof. $G \cup G'$ contains all gate polynomials and input field polynomials of C and C', but no further polynomials. Since C, C' only share the input variables, the intersection $G \cap G'$ only contains the input field polynomials. The input variables are the smallest elements in $\leq, \leq'$, thus by construction they are also the smallest elements in $\leq_\cup$. Furthermore the term orderings for the gate polynomials of C and C' are still valid in $\leq_\cup$. By the constraints on $\leq_\cup$ the leading term of each polynomial in $G \cup G'$ is either the output variable of a corresponding gate or the square of an input variable. Thus by the last item in the list about Gröbner bases (cf. Sec. II) $G \cup G'$ is a Gröbner basis for $J(C \cup C')$ w.r.t. $\leq_\cup$. □

For our incremental equivalence checking technique we use the definitions of *slices* and *sliced Gröbner basis* of [15].

Definition 7. Let C and C' be two circuits as above.
1) For each pair of output bits s_i and s'_i we determine its input cone, namely the gates which s_i and s'_i depends on:

$$I_i := \{\text{gate } g \mid g \text{ is in input cone of output } s_i \text{ or } s'_i\}$$

2) We define *slices* S_i as difference of consecutive cones I_i:

$$S_0 := I_0 \qquad\qquad S_{i+1} := I_{i+1} \setminus \bigcup_{j=0}^{i} S_j$$

Definition 8 (Sliced Gröbner Bases)**.** Let G_i be the set of polynomials of the gates of C and C' in slice S_i cf. Eqn. 1.

Corollary 3. G_i is a Gröbner basis for slice S_i.

Proof. Applying Lemma 2. □

Now we define a sequence of relations independent from the underlying Gröbner basis representation, which yields an abstract characterization of an incremental bit-level equivalence checking algorithm.

Definition 9. Let C, C' be two circuits as above. A sequence of m polynomials $\Delta_0, \ldots, \Delta_m$ over the variables of C, C' is called a sequence of *slice polynomials* if

$$-\Delta_i + 2\Delta_{i+1} + (s_i - s'_i) \in J(C \cup C') \quad \text{for all } 0 \leq i < m$$

Then the $E_i = -\Delta_i + 2\Delta_{i+1} + (s_i - s'_i)$ polynomials are called the *slice relations* for the sequence $\Delta_0, \ldots, \Delta_m$.

Theorem 3. Let C, C' be two circuits as above and $\Delta_0, \ldots, \Delta_m$ be a sequence of slice polynomials as defined in Def. 9. Then C, C' are equivalent ($C \equiv C'$) in the sense of Def. 6 iff $2^m\Delta_m - \Delta_0 \in J(C \cup C')$.

Proof. Using Def. 9 we obtain modulo $J(C \cup C')$

$$\sum_{i=0}^{m-1} 2^i(s_i - s'_i) = \sum_{i=0}^{m-1} 2^i(2\Delta_{i+1} - \Delta_i) = 2^m\Delta_m - \Delta_0.$$

$\square$

From this abstract theorem we derive our incremental equivalence checking algorithm. We fix the boundary $2^m\Delta_m = 0$ and derive Δ_i recursively by computing the remainder of $2\Delta_{i+1} + s_i - s'_i$ modulo the sliced Gröbner bases. This ensures that all E_i are contained in $J(C \cup C')$. Note that $J(C \cup C')$ contains all field polynomials F, thus we add them to $G \cup G'$. In the end we check if $\Delta_0 = 0$. By similar arguments as in the proof of Thm. 4 in [15] we show correctness of our algorithm.

Algorithm 1: Equivalence Checking Algorithm

```
Input  : Circuits $C, C'$ with sliced Gröbner bases $G_i$
Output: Decide if $C$ and $C'$ are equivalent ($C \equiv C'$)
1 $\Delta_m \leftarrow 0$;
2 for $i \leftarrow m-1$ to $0$ do
3     $\Delta_i \leftarrow$ Remainder ($2\Delta_{i+1} + s_i - s'_i$,  $G_i \cup F$)
4 end
5 return $\Delta_0 = 0$
```

Theorem 4. Algorithm 1 is correct.

Proof. By definition E_i is contained in $\langle G_i \cup F \rangle$ which is a subset of $\langle G \cup G' \cup F \rangle$ since $G_i \subseteq G \cup G'$. Therefore $E_i \in J(C \cup C')$. We show inductively that Δ_i is reduced w.r.t. $H_i := \bigcup_{j \geq i}(G_j \cup F)$. For the induction it is required that s_i and s'_i are reduced w.r.t. to H_{i+1}, which holds due to the construction of the sliced Gröbner bases (Def. 8). With $H_0 = G \cup G' \cup F$ we get Δ_0 is reduced w.r.t. $G \cup G' \cup F$ thus $\Delta_0 = 2^m\Delta_m - \Delta_0 \in J(C \cup C')$ iff $\Delta_0 = 0$, concluding the proof using Thm. 3. $\square$

VII. Experiments

In our experiments we focus on integer multipliers with $2n$ output bits and two input bit vectors of size n. We use the same multiplier types as [15]. The "btor"-benchmarks are generated by Boolector [12]. The "sp-ar-rc"-multipliers are part of the AOKI benchmark set [7] which includes several multiplier architectures. In both multipliers the partial products are generated as products of two input bits which are then accumulated by full- and half-adders, as in Fig. 3 for $n = 4$.

TABLE I
Improving the approach of [15] by Adder Rewriting and Partial Product Elimination

mult	n	Mathematica					Singular				
		[15]		+Adder Rew.			[15]		+Adder Rew.		
			+cs		+ppe	-s		+cs		+ppe	-s
btor	8	1	1	1	1	0	0	1	1	0	1
btor	16	4	4	1	1	1	1	1	0	0	1
btor	32	36	38	3	2	2	16	19	1	1	1
btor	64	417	443	11	6	5	MO	MO	14	9	4
btor	128	TO	TO	99	45	39	EE	EE	EE	EE	EE
sp-ar-rc	8	1	2	1	1	1	0	0	0	1	0
sp-ar-rc	16	7	7	1	1	1	1	2	0	1	0
sp-ar-rc	32	72	71	2	1	1	39	53	2	1	1
sp-ar-rc	64	874	897	10	6	5	MO	MO	15	10	5
sp-ar-rc	128	TO	TO	105	50	41	EE	EE	EE	EE	EE

In "btor" full- and half-adders are accumulated in a grid-like structure, whereas in "sp-ar-rc" full- and half-adders are accumulated diagonally (see again Fig. 3). We used the available tool AIGMULTOPOLY [15] and added our optimizations Adder Rewriting and Partial Product Elimination. We further included our incremental equivalence checking approach. The tool takes an AIG [10] representation of a circuit as input and returns a file containing computation instructions which can be passed on to Mathematica [20] or Singular [6].

In our experiments we used a standard Ubuntu 16.04 Desktop machine with Intel i7-2600 3.40GHz CPU and 16 GB of main memory. The (wall-clock) time limit was set to 1200 seconds and the main memory usage was limited to 14GB. All experimental data is available at http://fmv.jku.at/algeq.

The time in all experiments is listed in seconds (wall-clock time). We measure from starting AIGMULTOPOLY until Mathematica and Singular are finished including the time which the tool needs to generate the files for Mathematica and Singular (worst case 4 seconds for $n = 128$) and launching time of the computer algebra systems. We mark unfinished experiments by TO (reached time limit), MO (reached memory limit) or EE (error state). An error occurs in Singular when the maximum number of 32767 ring variables is exceeded.

In Table I Adder Rewriting (cf. Sect. IV) is applied on top of the incremental column-wise approach of [15]. Including the specification of each full- and half-adder to the Gröbner basis as constraint (+cs), without eliminating any variable, slows down computation slightly. Apparently, computer algebra systems can not make use of these redundant constraints.

Applying Adder Rewriting for the sliced Gröbner bases speeds up the computation substantially. Additionally applying Partial Product Elimination (+ppe) as described in Sect. V brings further improvement. Since the considered multipliers can fully be partionioned into full- and half-adders we never reduce by a polynomial where the leading term is the sum output of a full- or half-adder (cf. Sect. IV). Elimination of these polynomials from the Gröbner basis (-s) brings further improvements, but loses completeness.

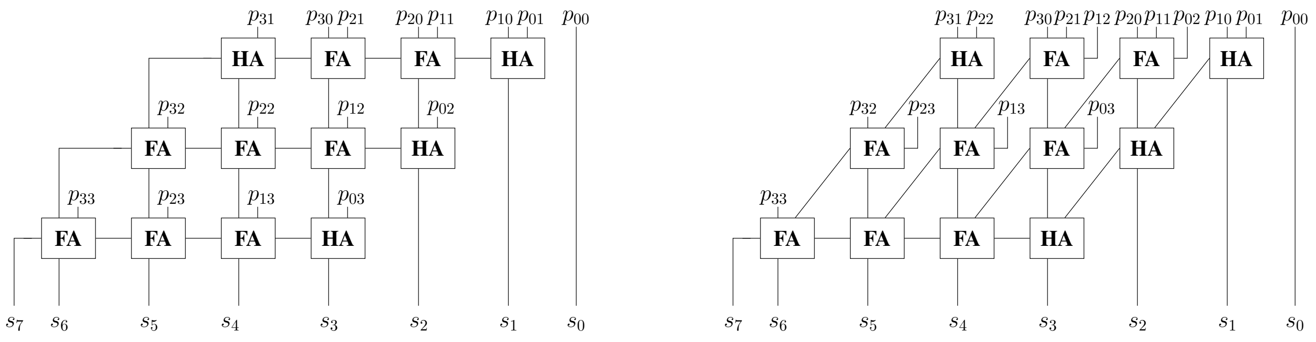
p_{31} p_{30} p_{21} p_{20} p_{11} p_{10} p_{01} p_{00}
HA FA FA HA
p_{32} p_{22} p_{12} p_{02}
FA FA FA HA
p_{33} p_{23} p_{13} p_{03}
FA FA FA HA
s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0
p_{31} p_{22} p_{30} p_{21} p_{12} p_{20} p_{11} p_{02} p_{10} p_{01} p_{00}
HA FA FA HA
p_{32} p_{23} p_{13} p_{03}
FA FA FA HA
p_{33}
FA FA FA HA
s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig. 3. Structure of "btor" (left) and "sp-ar-rc" (right) for $n = 4$ with $p_{ij} = a_i b_j$

TABLE II
INCREMENTAL COLUMN-WISE EQUIVALENCE CHECKING.

mult	n	-Adder Rew.	+Adder Rew.	-s
btor vs. sp-ar-rc	8	1	0	1
btor vs. sp-ar-rc	16	10	1	1
btor vs. sp-ar-rc	32	114	3	2
btor vs. sp-ar-rc	64	TO	15	12
btor vs. sp-ar-rc	128	TO	116	98

Table II shows experiments on incremental equivalence checking (Sect. VI). Mathematica is used as computer algebra system, due to the more flexible input language and supporting more variables. We derive the Gröbner bases of the circuits using the approach of [15] with (4th column) and without (3rd column) applying Adder Rewriting (Sect. IV). In our current implementation we do not use the optimization of Sect. V, since identifying equivalent partial products in two circuits is more complex and error prone. As in Table I we further enhance Adder Rewriting by deleting the polynomials where the leading term is the sum of a full- or half-adder from the Gröbner basis (5th column). We check the equivalence of the "btor" and "sp-ar-rc" multipliers.

Despite their architectural similarity, neither Lingeling [2] nor ABC [1] succeed to verify their equivalence for $n = 16$ within 10 hours, whereas it takes about a second for our incremental column-wise equivalence checking approach using Adder Rewriting.

VIII. Conclusion

In this paper we improved and extended recent algebraic approaches for multiplier circuit verification using computer algebra systems. As first optimization we used full- and half-adders extracted from the circuit to eliminate from the Gröbner basis internal variables of these full- and half-adders.

As second optimization we proposed to rewrite the specification of a multiplier in terms of partial products instead of circuit inputs to further reduce the Gröbner basis. We also presented a new incremental equivalence checking technique, including formal proofs, applicable to all acyclic gate-level circuits. As future work we want to extend these techniques to floating point operations, to even more challenging multiplier architectures and other arithmetic circuits such as dividers.

References

[1] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/. Bitbucket Version 1.01, last change Feb. 27, 2017.

[2] A. Biere. Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016. In *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, pages 44–45. University of Helsinki, 2016.

[3] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, University of Innsbruck, 1965.

[4] B. Buchberger and M. Kauers. Gröbner basis. *Scholarpedia*, 5(10):7763, 2010. http://www.scholarpedia.org/article/Groebner_basis.

[5] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Springer-Verlag New York, 1997.

[6] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 4-1-0. http://www.singular.uni-kl.de, 2016.

[7] N. Homma, Y. Watanabe, T. Aoki, and T. Higuchi. Formal design of arithmetic circuits based on arithmetic description language. *IEICE Transactions*, 89-A(12):3500–3509, 2006.

[8] A. Kandri-Rody and D. Kapur. Computing a Gröbner basis of a polynomial ideal over a Euclidean domain. *Journal of Symbolic Computation*, 6(1):37–57, 1988.

[9] A. Kandri-Rody, D. Kapur, and P. Narendran. An ideal-theoretic approach to work problems and unification problems over finitely presented commutative algebras. In *RTA*, pages 345–364. Springer, 1985.

[10] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE TCAD*, 21(12):1377–1394, 2002.

[11] J. Lv, P. Kalla, and F. Enescu. Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits. *IEEE TCAD*, 32(9):1409–1420, 2013.

[12] A. Niemetz, M. Preiner, and A. Biere. Boolector 2.0 system description. *JSAT*, 9:53–58, 2014 (published 2015).

[13] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, Oxford, UK, 2000.

[14] T. Pruss, P. Kalla, and F. Enescu. Equivalence verification of large Galois field arithmetic circuits using word-level abstraction via Gröbner bases. In *DAC*, pages 152:1–152:6. ACM, 2014.

[15] D. Ritirc, A. Biere, and M. Kauers. Column-wise verification of multipliers using computer algebra. In D. Stewart and G. Weissenbacher, editors, *FMCAD*, pages 23–30. IEEE, 2017.

[16] 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*, pages 1048–1053. IEEE, 2016.

[17] A. Sayed-Ahmed, D. Große, M. Soeken, and R. Drechsler. Equivalence checking using Gröbner bases. In *FMCAD*, pages 169–176. IEEE, 2016.

[18] D. Stoffel and W. Kunz. Equivalence checking of arithmetic circuits on the arithmetic bit level. *IEEE TCAD*, 23(5):586–597, 2004.

[19] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Greuel. An algebraic approach for proving data correctness in arithmetic data paths. In *CAV*, volume 5123 of *LNCS*, pages 473–486. Springer, 2008.

[20] Wolfram Research, Inc. Mathematica, 2016. Version 10.4.

[21] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski. Formal verification of arithmetic circuits by function extraction. *IEEE TCAD*, 35(12):2131–2142, 2016.