ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/272200796

Groebner Basis Based Formal Verification of Large Arithmetic Circuits Using Gaussian Elimination and Cone-based Polynomial...

Article *in* Microprocessors and Microsystems · February 2015

DOI: 10.1016/j.micpro.2015.01.007

CITATIONS
10

READS
108

2 authors:

Farimah Farahmandi
University of Florida
9 PUBLICATIONS **39** CITATIONS
SEE PROFILE

Bijan Alizadeh
University of Tehran
84 PUBLICATIONS **279** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Analytical Imprecision Optimization of Fixed-point Digital Circuits View project

Formal Verification and Correction of Dynamic Power Management in Modern Processors View project

All content following this page was uploaded by Bijan Alizadeh on 10 June 2015.

The user has requested enhancement of the downloaded file.

Microprocessors and Microsystems 39 (2015) 83–96

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Groebner basis based formal verification of large arithmetic circuits using Gaussian elimination and cone-based polynomial extraction

Farimah Farahmandi [a], Bijan Alizadeh [a,b,*]

[a] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran 14395-515, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.O.Box 19395-5746, Tehran, Iran

ARTICLE INFO

Article history:
Available online 12 February 2015

Keywords:
Formal verification
Arithmetic circuits
Groebner basis
Gaussian elimination
Repetitive components

ABSTRACT

Verification of arithmetic circuits is essential as they form the main part of many practical designs such as signal processing and multimedia applications. In these applications, the size of the datapath could be very large so that contemporary verification methods would be almost incapable of verifying such circuits in reasonable time and memory usage. This paper addresses formal verification of large integer arithmetic circuits using symbolic computer algebra techniques. In order to efficiently verify gate level arithmetic circuits, we model the circuit and the specification with polynomial system and the verification problem is formulated as membership testing of the given specification polynomial in corresponding ideal of the circuit polynomials. The membership testing needs Groebner basis reduction. In order to overcome the intensive polynomial reduction needed in Groebner basis computation so that we can deal with verifying large arithmetic circuits, the fanout-free regions (cones) of the circuit are extracted and represented as corresponding polynomials automatically. For further improvement, we make use of Gaussian elimination concept to perform specification polynomial reduction w.r.t Groebner basis using a matrix representation of the problem. To evaluate the effectiveness of our verification technique, we have applied it to very large arithmetic circuits with different architectures. The experimental results show that the proposed verification technique is scalable enough so that large arithmetic circuits can efficiently be verified in reasonable run time and memory usage.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

As the size and complexity of digital systems increase continuously, design verification and debug are quickly becoming more important. From a verification point of view, one of the most difficult parts in complicated digital designs is arithmetic datapaths and their components, such as multipliers and dividers. Arithmetic circuits are the main part in many computational intensive designs, e.g. Digital Signal Processing (DSP) units in multimedia applications. The complicated and specialized nature of these arithmetic architectures requires custom design which makes the implementation error-prone. Therefore, verification of arithmetic circuits in a fast and precise way plays an important role in a digital system design flow.

Formal verification methods make use of mathematical techniques to insure the integrity of a design with respect to some desired characteristics. Several formal methods have been proposed to verify the correctness of arithmetic circuits in higher level of abstraction. Most of them are based on canonical graph-based representations like Binary Decision Diagrams (BDDs), which are not scalable because they suffer from space and time explosion problems when dealing with large arithmetic circuits especially multipliers [16,17].

On the other hand, recently some techniques for verification of bit-level implementations using the theory of Groebner basis have been proposed [6,7,9]. However, these techniques are computationally intensive and are not scalable to large arithmetic circuits.

Addressing the above problems, this paper proposes a formal verification technique to verify gate level circuits that implement integer arithmetic circuits so that their specifications are given as polynomial functions f_{spec}. The goal of the verifier is to guarantee the equivalence of f_{spec} and *gate level implementation* f_{imp}. In fact our proposed method uses Groebner basis theory to effectively verify large arithmetic circuits. This theory enables us to formulate the verification problem as an ideal membership testing. Fig. 1 shows our proposed verification technique which takes a gate level circuit (f_{imp}) and its high level specification (f_{spec}) as inputs. In order to check whether f_{imp} is functionally equivalent to f_{spec} or not, first of all, the gate level circuit is converted to Boolean polynomials

* Corresponding author at: School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran 14395-515, Iran.
E-mail addresses: f.farahmandi@ut.ac.ir (F. Farahmandi), b.alizadeh@ut.ac.ir (B. Alizadeh).

http://dx.doi.org/10.1016/j.micpro.2015.01.007
0141-9331/© 2015 Elsevier B.V. All rights reserved.

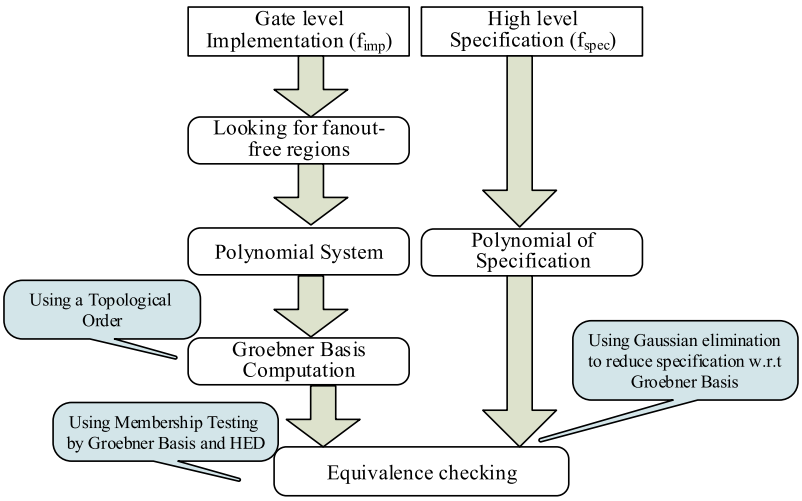

Fig. 1. Proposed verification technique for large arithmetic circuits.

by extraction of fanout free regions (which are referred to as *cones*) and mapping cones to polynomials. We automatically exploit the cones in order to achieve a smaller number of circuit polynomials. We will discuss how to find them based on a backtracking algorithm and obtaining their equivalent polynomials in Section 4. As we will see in Section 3, the ideal membership testing requires Groebner basis computation which is computationally extensive. In order to remove the need of this computation, a topological order of polynomials' variables (eke polynomials' terms) is utilized in Section 4. This ordering eliminates the computation of Groebner basis as well as making the initial set of the circuit polynomials (**F**) Groebner basis itself. Then the equivalence checking is performed by polynomial manipulation and reduction of the specification polynomial (f_{spec}) over Groebner basis polynomials (**F**) [10]. We call this step Groebner basis reduction or polynomial reduction.

As will be discussed in Section 3, the mathematical manipulations (Groebner basis reduction) contain polynomial divisions and multiplications which are complicated in terms of run time and memory usage as the intermediate polynomials may become very large. In order to overcome these intensive computations, we utilize the modification of F4 algorithm [24] for computing Groebner basis reduction on a matrix which represents the verification problem. By using this method, we will be able to verify large arithmetic circuits in reasonable run time and memory space usage as experimentally shown in Section 6. In summary, our contributions in this work are as follows:

- Extracting a finite set of Boolean polynomials from the gate level implementation by looking for fanout free regions (*cones*). In contrast to [18–20], this method is applicable even when half adder and full adders cannot be extracted due to the local optimizations which are usually done by synthesis tools (Section 4).
- In contrast to [29] in which we looked only for repetitive components (which needs some information about repetitive components provided as a library), the proposed method in this work enables us to reduce the number of polynomials significantly so that the verification time decreases dramatically while we do not have any information about the repetitive components.
- Determining a suitable variable ordering for cones' polynomials in such a way that the leading terms of the circuit polynomials become relatively prime to avoid Groebner basis computation (Sections 3 and 4).
- Improving F4 algorithm and using Gaussian elimination to perform polynomial reduction efficiently so that a sequence of polynomial divisions can be efficiently computed (Section 5).

The paper is organized as follows: Section 2 provides a brief review of related work. In Section 3 we briefly describe some mathematical background as well as Horner Expansion Diagrams (HEDs) [14,15]. Section 4 presents our proposed verification technique in detail. Section 5 describes our approach to use Gaussian elimination in order to perform the Groebner basis reductions. The experimental results are shown in Section 6 and Finally, Section 7 concludes the paper.

2. Related work

There is a large amount of literature on equivalence verification of arithmetic circuits against their specifications. A variety of canonical, directed acyclic graph (DAG) representations such as BDDs [2], *BMDs [17] and their various word-level extensions [25] have been used for verification of Boolean functions. However, they are not able to deal with large arithmetic circuits especially multipliers due to bit-blasting as well as existence of different architectures. A combination of TEDs [4] has been proposed to represent multivariate polynomials. However, TEDs need word level information of circuits which are not available at the gate level. Decision making procedures and term-rewriting have been proposed in [3,32]. However, none of the above-mentioned methods are able to solve the verification problem of integer arithmetic circuits practically as they utilize modular arithmetic concept.

Extracting arithmetic operations such as half-adders (HAs) and full-adders (FAs) from the gate-level implementation and generating an Arithmetic Bit Level (ABL) model to compare with the high-level specification is another approach in verification of arithmetic circuits [18]. This technique checks different XOR tree structures exhaustively to verify carry signals. The complexity of this checking is exponential in terms of run-time. More efficient approaches are presented in [19,34] which make use of reverse-engineering and bit level adder (BLA) representation to derive a network of half-adders from the gate level implementation of multipliers and dividers. However, such methods suffer from dealing with *pin-swap* optimizations due to the fact that swapping of the partial product bits does not change the functionality of the design. The debugging algorithm in [20] can be useful for solving the above mentioned problems.

Several methods based on computer symbolic algebra are taken into account for verification of arithmetic circuits as well. In [5], arithmetic circuits are described with weighted number systems so arithmetic formula and equivalence checking can be performed

by formula manipulations based on Groebner basis. This technique requires hierarchical information of circuit which is often unavailable. Symbolic algebra methods also have been used for verification of arithmetic circuits over finite rings [21]. This method checks the difference between two polynomial expressions by utilizing "vanishing polynomial" theory which actually limits its applicability to verification of arithmetic circuits due to using a word level representation of the datapaths.

The theorem provers [27], high-level/RTL Synthesis tools [28] and symbolic algebra tools have been integrated for verification purposes. However, applying them to integer arithmetic circuits is not straightforward as their models are over rational numbers' field. The authors of [13] have presented a computer algebra based technique to model and verify multiplier circuits over Galois fields F_2^k. They have shown how to model Galois field multipliers as a polynomial system in F_2^k. They have also shown how to formulate the verification problem as a membership test in a corresponding ideal. In order to overcome the cost of Buchberger's algorithm [6], they have analyzed the circuit topology and derived a suitable term order to represent polynomials. This approach, however, is not applicable to integer multipliers due to carry propagation issues.

The authors of [7] have proposed a formal verification technique in which ABL components [18] are modeled by polynomials over unique ring and their normal forms are computed with respect to the Groebner basis over rings Z_2^k using computer algebra techniques. In order to overcome the expensive Groebner basis computation problem, they have proposed a technique to directly generate individual output polynomials in terms of primary inputs. However, there is no systematic way for comparing such polynomials against the specification so preprocessing of such very large polynomials is needed for computation of the normal form algorithm. In [11,9], arithmetic circuits are represented as a network of half adder, full adders, and inverters and modeled as a system of linear equations. Functional correctness of the gate level implementation is proved by computing its algebraic signature (which is the outcome of standard linear programming solvers) and comparing it with the reference signature provided by the designer. It should be noted that if half adder and full adders could not be extracted due to optimizations done by synthesis tools, this technique would not be applicable.

3. Preliminaries

In this section, we briefly describe mathematical concepts including Groebner basis, Horner Expansion Diagram (HED) and the F4 algorithm. The mathematical backgrounds of Groebner basis are mostly based on [10].

3.1. Algebraic preliminaries

Let $M = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ be a monomial in variables $x_1, x_2, \ldots, x_n$ where all of the exponents $\alpha_1, \alpha_2, \ldots, \alpha_n$ and variables are non-negative integers. Let K be a computable field and $\mathbf{K}[x_1, x_2, \ldots, x_n]$ be the polynomial ring in n variables. A finite linear combination of monomials in $x_1, x_2, \ldots, x_n$ is a polynomial f. Polynomial $f \in K[x_1, x_2, \ldots, x_n]$ is written as $f = c_1M_1 + c_2M_2 + \cdots + c_dM_d$, where $c_1, c_2, \ldots, c_d$ are coefficients and $M_1, M_2, \ldots, M_d$ are monomials.

Let $f_1, f_2, \ldots, f_s$ be polynomials in $K[x_1, x_2, \ldots, x_n]$. Then $\langle f_1, f_2, \ldots, f_s\rangle = \{\sum_{i=1}^{s} h_i f_i : h_1, h_2, \ldots, h_s \in K[x_1, x_2, \ldots, x_n]\}$ is called an ideal I. So the finite set of polynomials $F = \{f_1, f_2, \ldots, f_s\}$ is called generator or basis of ideal I.

It has been shown in [12] that every arbitrary ideal other than {0} has a basis with specific properties which is called *Groebner basis*. Groebner basis enables us to solve ideal membership problem. To describe Groebner basis, first we need to cover some basic definitions.

Definition 1. A monomial ordering on the set of monomials is any relation ">" on $Z_{\geqslant 0}^n$ with the following properties:

i. Every nonempty set has the smallest element under >.
ii. If $\alpha > \beta$ and $\gamma \in Z_{\geqslant 0}^n$ then $\alpha + \gamma > \beta + \gamma$.

It is a well-ordering on the set of all monomials such that multiplication with a monomial preserves the ordering. With respect to the monomial ordering, we recall some definitions to obtain Groebner basis from a finite set of polynomials.

Definition 2. Let $f = \sum_\alpha a_\alpha X^\alpha$ be a nonzero polynomial in $K[x_1, x_2, \ldots, x_n]$ and > be a monomial order.

i. LM (f) is the leading monomial (the largest monomial) of f with respect to >.
ii. LC (f) is the leading coefficient of f with respect to >.
iii. LT (f) is the leading term of f with respect to >. The initial term of f is LT (f) = LM (f) × LC (f).

Definition 3 (*normal form*). Let f, g and t be polynomials in $K[x_1, x_2, \ldots, x_n]$. We say that $g \neq 0$ is reducible to t by f if there is a term M which can be divided by LM (f) in g and $t = g - \frac{c \times M}{\mathrm{LC}(f) \times \mathrm{LM}(f)} \times f$. Where coefficient of M is c. It is denoted by $g \underset{f}{\rightarrow} t$. Let F be a polynomial set in $K[x_1, x_2, \ldots, x_n]$, we say g is reducible to h with respect to F if there is a sequence of polynomials $f_1, f_2, \ldots, f_s \in F$ that $\underset{f_1}{\rightarrow} \underset{f_1}{\rightarrow} \cdots \underset{f_s}{\rightarrow} h$. We can also represent it by $g \underset{F}{\rightarrow} h$. If we cannot reduce h with respect to F anymore, we say h is normal form of g.

An ideal I may have many different generators: it is possible to have sets of polynomials $F = \{f_1, \ldots, f_s\}$ and $G = \{g_1, \ldots, g_t\}$ such that $I = \langle f_1, \ldots, f_s\rangle = \langle g_1, \ldots, g_t\rangle$. Some generating sets are better because they represent ideal and its attribute better. A Groebner basis is one of them because it can answer to many polynomial decision questions such as polynomial membership testing in an ideal. We have utilized the ability of Groebner basis to determine the membership status of a polynomial in an ideal in our verification technique.

Definition 4 (*membership testing*). The set G of ideal $\mathbf{I}$ is a Groebner basis if and only if for all polynomial $f \in \mathbf{I}$ the remainder of reducing f by polynomials of G is zero. This process is called membership testing of f over ideal $\mathbf{I}$ and denoted by $\forall f \in \boldsymbol{I}$, $f \underset{G}{\rightarrow} 0$.

Definition 5. Let $I \subset K[x_1, x_2, \ldots, x_n]$ be an ideal other than {0}. The ideal of leading terms generated by the elements of LT ($\mathbf{I}$) is denoted as $\langle \mathrm{LT}(\mathbf{I})\rangle$ where $LT(I) = \{cx^\alpha : \text{there exists } f \in I \text{ with } LT(f) = cx^\alpha\}$.

Definition 6 (*Groebner basis*). With respecting to a monomial order, a finite subset $G = \{g_1, g_2, \ldots, g_t\}$ of an ideal $\mathbf{I}$ is said to be a Groebner basis of $\mathbf{I}$ *if* $\langle LT(g_1), \ldots, LT(g_t)\rangle = \langle LT(\boldsymbol{I})\rangle$.

```
Input: F=(f1,f2,…,fs), ideal I = <f1,f2,…,fs>≠{0}
Output: G = (g1, g2, …, gt) for ideal I
1   G := F;
2   V := G × G;
3   WHILE V ≠ 0
4     FOR each pair (f,g) ∈ V DO
5       V := V - (f,g);
6       Spoly(f,g) --G--> r;
7       IF r ≠ 0 THEN
8         G := G ∪ {r};
9         V := V ∪ (G × r);
```

Fig. 2. Buchberger's algorithm.

To compute Groebner basis over a field, Buchberger's algorithm in Fig. 2 is used. It makes use of a polynomial reduction technique named S-polynomial as defined below.

Definition 7 (*S-polynomial*). Let $f, g \in K[x_1, x_2, \ldots, x_n]$ be nonzero polynomials. The S-polynomial of f and g is defined as Spoly $(f,g) = \frac{\text{LCM}(\text{LM}(f),\text{LM}(g))}{LT(f)} \times f - \frac{\text{LCM}(\text{LM}(f),\text{LM}(g))}{LT(g)} \times g$, where LCM(a, b) is a notation for the least common multiple of a and b.

Example 1. Let $f = 6x_1^4x_2^5 + 24x_1^2 - x_2$ and $g = 2x_1^2x_2^7 + 4x_2^3 + 2x_3$ and we have $x_1 > x_2 > x_3$. The S-polynomial of f and g is defined below:

$$\text{LM}(f) = x_1^4x_2^5,\ \text{LM}(g) = x_1^2x_2^7 \text{ and } \text{LCM}(x_1^4x_2^5, x_1^2x_2^7) = x_1^4x_2^7$$

$$\text{Spoly}\ (f,g) = \frac{x_1^4x_2^7}{6x_1^4x_2^5} \times f - \frac{x_1^4x_2^7}{2x_1^2x_2^7} \times g = 4x_1^2x_2^2 - \frac{1}{6}x_2^3 - 2x_1^2x_2^3 - x_1^2x_3.$$

It is obvious that S-polynomial computation cancels leading terms of the polynomials. As shown in Fig. 2, Buchberger's algorithm first calculates all S-polynomials (lines 4–6 of Fig. 2) and then add non-zero S-polynomials to the basis G (line 8). This process repeats until all of the computed S-polynomials become zero with respect to G. It is obvious that Groebner basis can be extremely large so its computation may take a long time and it may need large storage memory as well. The time and space complexity of this algorithm are exponential in terms of the sum of the total degree of polynomials in F, plus the sum of the lengths of the polynomials in F [12]. So when the size of F increases, the verification process would be very slow or in the worst-case would be infeasible.

Definition 8. $V(I)$ is the affain variety of ideal I such that $V(I) = \{(a_1, a_2, \ldots, a_n) | f((a_1, a_2, \ldots, a_n) = 0 \quad for\ all\ f \in I\}$ where $a_i \in \mathbf{K}$. For every generating set of ideal I such as Groebner basis G, we have $V(I) = V(G)$.

Definition 9. f is a member of ideal I if it vanishes on $V(I)$ or $V(I) \subset V(f)$.

This definition says that f would be a member of I if it agrees with all the solutions $f_1 = f_2 = \cdots = f_s = 0$. This problem is called "variety subset" in [10]. In other words, in order to find out whether polynomial f is a member of ideal I or not, we need to check the possibility of vanishing f on $V(I)$. On the other hand, regarding Definition 8, if G is Groebner basis of ideal I, $V(I) = V(G)$. This means that instead of checking whether f is vanishes on $V(I)$, we can just check whether f can be reduced to 0 by polynomials of G.

3.2. F4 Algorithm

In computer algebra, the F4 algorithm [24] computes the Groebner basis efficiently. The mathematical principle of this algorithm is the same as that of the Buchberger's algorithm. However, in order to expedite the Buchberger's algorithm, the F4 algorithm avoids classic reduction procedure which may need many intermediate computations. Instead, it uses symbolic pre-computation and sparse linear algebra methods to make the computation parallel in two ways: (1) selecting a subset of pairs (in contrast to Buchberger's algorithm that chooses one pair; line 4 of Fig. 2) and computing more than one S-polynomial in each iteration and (2) using sparse linear algebra on a matrix (in contrast to Buchberger's algorithm that chooses a reducer among current set of G; line 6 of Fig. 2) to compute the normal form.

The F4 algorithm takes a finite set of polynomials, F, and a set of ordered monomial as inputs and constructs a sparse matrix such that the element M_{ij} shows the coefficient of the jth monomial of the ith polynomial. The matrix may grow up because new polynomials followed by new monomials might be increased during the computation of the algorithm. The algorithm makes use of a well-known reduction technique on matrix, i.e., Gaussian elimination, to perform the normal form computation of the Buchberger's algorithm (according to Definition 3). The F4 algorithm generates a set of polynomials, G, as output such that G is Groebner basis of F.

3.3. Horner Expansion Diagram (HED)

According to Definition 9, in order to check whether the polynomial specification (f_{spec}) is a member of the Groebner basis computed from the circuit polynomials, we should check whether f_{spec} is reduced to zero by the Groebner basis. Such a reduction phase can be done by division of f_{spec} over the Groebner basis polynomials. In order to perform such a division process efficiently we make use of a canonical decision diagram called HED which is successfully used for equivalence checking and debugging purposes [8,26]. The HED is a binary graph-based representation where the algebraic expression $F(X,Y,\ldots)$ is expressed by a first-order linearization of the Taylor series expansion [14,15]. Suppose variable X is the top variable of $F(X,Y,\ldots)$. Eq. (1) shows $F(X,Y,\ldots)$, where *const* is independent of the variable X, while *linear* is the coefficient of variable X.

$$\begin{aligned} F(X,Y,\ldots) &= F(X=0,\ldots) + X \times [F'(X=0,\ldots) + \ldots] \\ &= const + X \times linear \end{aligned} \tag{1}$$

HED is a directed acyclic graph $G = (\text{VR}, \text{ED})$ with vertex set VR and edge set ED. While the vertex set VR consists of two types of vertices: Constant (C) and Variable (V), the edge set indicates integer values as weight attribute. A Constant node v has a value $\text{val}(v) \in Z$ as its attribute. A Variable node v has three attributes: an integer variable $\text{var}(v)$ and two children $\text{const}(v)$ and $\text{linear}(v) \in \{V, C\}$. Hence, each vertex v in HED denotes an integer function $f(y)$ defined recursively as follows:

- If $y \in C$ (is a Constant node), then $f(y) = \text{val}(y)$.
- If $y \in V$ (is a Variable node), then $f(y) = \text{const}(y) + \text{var}(y) \times \text{linear}(y)$.

Fig. 3 illustrates how $f(x,y,z) = 24 - 8z + 12y + 12yz - 6x - 6x^2z$ is represented by the *HED*. Let the ordering of variables be $x > y > z$. First the decomposition w.r.t. x is taken into account. As shown in Fig. 3(a), after rewriting $f(x,y,z) = (24 - 8z + 12y + 12yz) + x(-6 - 6xz)$ based on (1), *const* and *linear* parts will be $24 - 8z + 12y + 12yz$ and $-6 - 6xz$, respectively. The *linear* part is decomposed w.r.t. variable x again due to x^2 sub-monomial. After

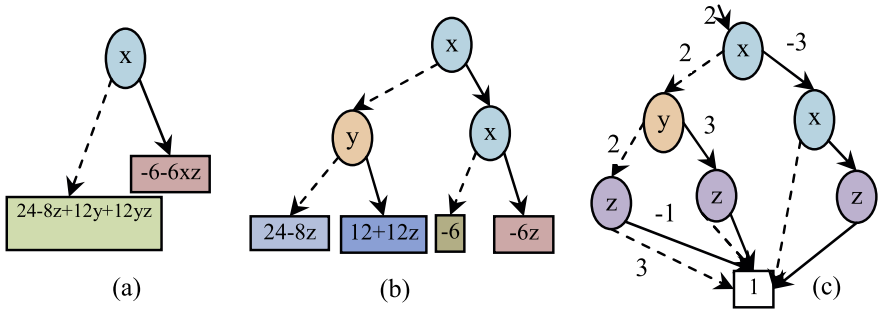

Fig. 3. HED representation of $24 - 8z + 12y + 12yz - 6x - 6x^2z$: (a) decomposition w.r.t. variable x, (b) decomposition w.r.t. variables x and y, and (c) decomposition w.r.t. variables x, y and z.

that, the decomposition is performed w.r.t. variable y and then z as shown in Fig. 3(b). In order to reduce the size of the *HED* representation, redundant nodes are removed and isomorphic sub-graphs are merged. In Fig. 3(b), $24 - 8z$, $12 + 12z$ and $-6z$ are rewritten by $8[3 + z(-1)]$, $12[1 + z(1)]$ and $-6[0 + z(1)]$, respectively. In order to normalize the weights, $gcd(12, 12) = 12$, $gcd(8, 12) = 4$ and $gcd(-6, -6) = -6$ are taken to extract common factors. Finally, Fig. 3(c) shows the normalized graph where $gcd(4, -6) = 2$ is taken to extract the common factor between out-going edges from *x node*. In this representation, dashed and solid lines indicate *const* and *linear* parts, respectively. Note that in order to have a simpler graph; paths to 0-terminal have not been drawn in Fig. 3(c). Obviously, we can access to constant and linear parts of f_{spec} w.r.t. a top variable with O(1). We have described the merit of using the HED in Section 5.

4. Proposed verification technique

Formally, the verification problem of an arithmetic circuit is characterized by a given specification like $f_{spec} := Y - (A * B + C * D)$ where A–D are decimal representation of circuit's primary inputs and Y is decimal representation of the circuit's primary outputs. The decimal representation can be achieved by a polynomial like $A = \sum_{i=0}^{n} 2^i a_i$ where $a_i \in \mathbb{B} = \{0, 1\}$ and a_0 and a_n are the least significant and most significant bits of primary input A, respectively.

Our goal is to check the correctness of gate level implementation (f_{imp}) against the specification polynomial (f_{spec}). In the first step, gate level circuit is modeled to a series of polynomials. We denote these polynomials as $\{f_1, \ldots, f_s\}$ over $K[x_1, \ldots, x_n]$. The specification polynomial is also considered as $f_{spec} \in K[x_1, \ldots, x_n]$. The generated Ideal $\mathbf{I} = \langle f_1, \ldots, f_s \rangle$ is taken into account and therefore the verification problem would become a membership testing of f_{spec} over ideal $\mathbf{I}$. As discussed before, to check whether f_{spec} is a member of ideal $\mathbf{I}$, its Groebner basis should be computed and then we need to check whether f_{spec} is reduced by the polynomials of the Groebner basis.

Keep in mind that the most time consuming task of Buchberger's algorithm is the S-polynomials computation and reduction. A simple idea to speed up such computations is to somehow remove this step from the algorithm. On the other hand, it is obvious that S-polynomial of a pair f_i and f_j whose leading power products are relatively prime, can be ignored. In other words, if $LCM(LM(f_i), LM(f_j)) = LM(f_i) \times LM(f_j)$ then $Spoly(f_i, f_j) \underset{G}{\rightarrow} 0$. This criterion is indicating that on those cases where the leading monomials of f and g are relatively prime, $Spoly(f, g)$ is always reduced to 0. Thus in Buchberger's algorithm, we do not need to compute $Spoly(f, g)$. Therefore analyzing and deriving a suitable term order from the given circuit can be quite useful, because it causes every polynomial pair (f, g) in the generating set has relatively prime leading

$$
\begin{aligned}
&z = NOT(a) \;\Rightarrow\; f = z - (1 - a)\\
&z = AND(a, b) \Rightarrow f = z - ab\\
&z = OR(a, b) \Rightarrow f = z - (a + b - ab)\\
&z = XOR(a, b) \Rightarrow f = z - (a + b - 2ab)
\end{aligned}
$$

Fig. 4. Polynomial model of each gate.

monomials, then $Spoly(f, g) \underset{G+}{\rightarrow} 0$ for all pairs f and g. Consequently, the polynomials $\{f_1, f_2, \ldots, f_s\}$ extracted from the circuit (corresponding to ideal $\mathbf{I}$) and represented using such a term order would itself constitute a Groebner basis of $\mathbf{I}$. Although such a term order is derived and the very same concept is proposed in [13], it has been applied only to Galois field multipliers while in this work we are dealing with integer arithmetic circuits including integer multipliers. Note that, in our case, the variables are all Boolean, so *their degrees never increase*. Another point is the fact that the outputs of gates (cones) are represented as individual variables in the set of polynomials.

We derive an order for polynomial terms based on a topological analysis of the circuit. Since the circuit is acyclic, if we can represent each cone polynomial such that its output places in a higher order than its inputs, then every two polynomials have relatively prime leading monomial and therefore $\{f_1, \ldots, f_s\}$ is itself Groebner basis. Note that, in our case, the variables are all binaries, so their degrees never increase. Another point is the fact that the outputs of cones are represented as individual variables in the set of polynomials.

In order to show how such a term ordering reduces the computational complexity of Buchberger's algorithm let us consider gate level implementation of a two bit multiplier shown in Fig. 5 (in this circuit, we suppose every gate is a cone). To make every polynomial pair (f, g) relatively prime, the following variable ordering is taken into account: $\{y_3 > y_2 > y_1 > y_0\} > \{w_3 > w_2\} > \{w_0 > w_1\} > \{a_1 > b_1 > a_0 > b_0\}$. Polynomials extracted from the specification (f_{spec}) and the implementation ($\{f_0, \ldots, f_7\}$: which is Groebner basis because each polynomial pair (f_i, f_j) are relatively prime, are described as follows:

$$
\begin{aligned}
&f_{spec} := 8y_3 + 4y_2 + 2y_1 + y_0 - (2a_1 + a_0)(2b_1 + b_0)\\
&f_0 := y_0 - a_0 b_0; \qquad LM(f_0) = y_0\\
&f_1 := w_0 - a_1 b_0; \qquad LM(f_1) = w_0\\
&f_2 := w_1 - a_1 b_1; \qquad LM(f_2) = w_1\\
&f_3 := w_2 - a_1 b_1; \qquad LM(f_3) = w_2\\
&f_4 := y_1 - (w_0 + w_1 - 2w_0 w_1); \qquad LM(f_4) = y_1\\
&f_5 := w_3 - w_0 w_1; \qquad LM(f_5) = w_3\\
&f_6 := y_2 - (w_3 + w_2 - 2w_3 w_2); \qquad LM(f_6) = y_2\\
&f_7 := y_3 - w_3 w_2; \qquad LM(f_7) = y_3
\end{aligned}
$$

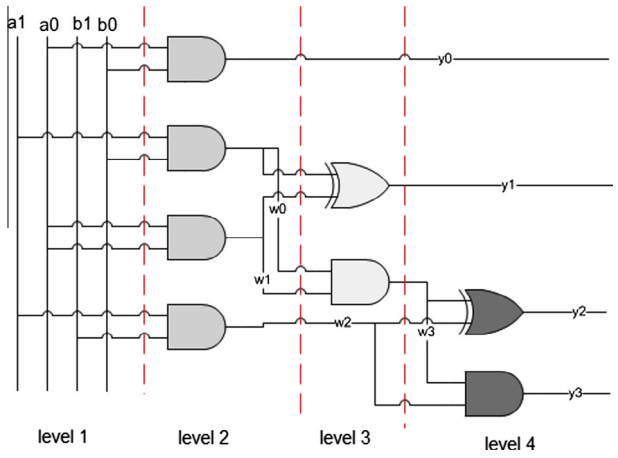

Fig. 5. Gate level implementation of a two bit unsigned multiplier.

After computing Groebner basis $\{f_1,\ldots,f_s\}$, the second step is to reduce the specification polynomial (f_{spec}) with respect to $\{f_1,\ldots,f_s\}$. In order to facilitate the reduction process, we can simultaneously reduce f_{spec} with respect to those polynomials which have outputs at the same level. This step is done with several consecutive polynomial divisions. First, the f_{spec} is divided by f_7 as $LT(f_{spec}) = 8 * y_3$ and $LT(f_7) = y_3$. Thus, $LT(f_{spec})$ will be canceled by f_7. The remainder of this (R) step is divided by one of the $f_0, \ldots, f_6$ which can cancel the $LT(R)$. The process will go on until we have a remainder that cannot be divided any further by any of polynomials in **F**. In this example, we will have zero for the remainder. The zero shows that the design has implemented the specification correctly. Unfortunately, as for circuits with more inputs, the number of variables in polynomials increases greatly and therefore specification reduction over ideal Groebner basis of circuit's polynomials becomes impractical and we get a timeout for large circuits. Our idea to alleviate this issue is to attack the problem from two aspects: (1) Reduce the number of circuit polynomials (Section 4.1 and Section 4.2) and (2) Performing Groebner reduction more effectively (Section 5).

4.1. How to reduce circuit polynomials

In order to reduce the number of polynomials, in [29] we used automata to extract those components which are being repeated in the circuit and create only one polynomial for all of the existing gates in these components. To find repetitive components, a tree-based matching algorithm which utilizes Automata [22] is applied. We have manually defined our components in a cell-library and we have used automaton to represent the cell-library. This approach is based on an encoding of the trees by strings of characters and on a string recognition algorithm. For applying this method, we consider our circuit as a rooted acyclic graph which is called subject graph. The graph associated with library elements are called pattern graphs which are also acyclic and rooted. The subject graph is visited in a bottom up fashion and matches between cell libraries and subject graph are found.

Fig. 6 shows a part of a large adder circuit [1]. In this circuit the signal u computes the sum of signals a–e. This sum is computed in a tree fashion to minimize delays. Signal v computes the corresponding carry. As we can see in the figure, extracting half adders and full adders is a hard task and almost impossible. However, some repetitive components can be found by automata based on our cell library which decreases the number of polynomials significantly.

By applying this technique to the circuit in Fig. 6, the number of polynomials decreases from 14 to 7. The number of variables is also reduced from 19 to 12. The corresponding polynomials are as follows:

$$f_1 := u - (a + b + h - 2ab - 2ah - 2bh + 4abh)$$
$$f_2 := h - (c + d + e - 2ce - 2de - 2dc + 4cde)$$
$$f_3 := f - (d + e - 2de)$$
$$f_4 := j - (a + f - 2af)$$
$$f_5 := k - (cj + bj + cb - 2cjb)$$
$$f_6 := l - (ad + ed - 2aed)$$
$$f_7 := v - (k + l - 2kl)$$

As mentioned in [29], first we set the cell library manually, and then we build the pattern graphs and automata automatically, and search the subject graph in order to find repetitive components. If

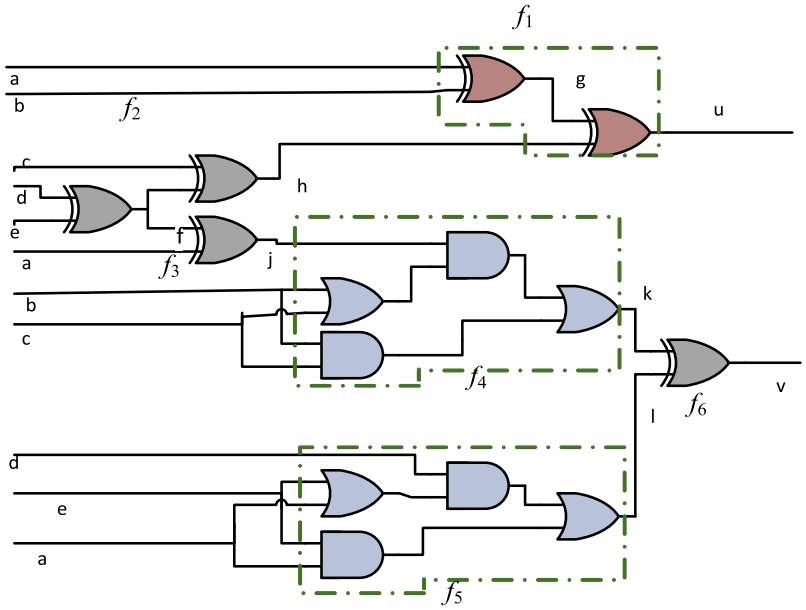

Fig. 6. Finding repetitive components in an optimized circuit.

we are aware of the structure of the circuit, the patterns in the cell library can be chosen wisely and repetitive components can be found more efficiently; therefore, the number of circuit polynomials will be reduced significantly. Otherwise, the automata searches the subject tree for some components which are not repetitive in the circuit, and it is a waste of the time as the number of circuit polynomials may not reduce dramatically. To address this issue, we have improved the method of [29] by looking for fanout-free regions of the circuit and modeling them to polynomials that is explained in the following subsection.

4.2. Cone extraction algorithm

In order to solve the above mentioned problem and being able to apply our method on different arithmetic circuit structures, we take the fanout-free regions (which are called *cones*) of the arithmetic circuit and represent them as a single polynomial. This method works even if repetitive components and the cell library components cannot be identified.

We use the algorithm shown in Fig. 7 to extract the cones of the circuit. The inputs of this algorithm are the gate level implementation of the circuit and the list of its fanouts. Please note that the list of fanouts can be automatically extracted. For doing so, the circuit is modeled as an acyclic graph such that every gate is a node in this graph. When the output degree of a node is greater than one, a fanout is detected. In order to cover the circuit completely, we consider primary outputs as fanouts.

In the algorithm shown in Fig. 7, *FO* is the list of circuit fanouts and $G(V,E)$ is the graph model of the gate level arithmetic circuit. Set V and E represent the gates and their connections, respectively. This graph contains some further information about level of each gate, the number of inputs of gates and their fanouts. The output of this algorithm is a list of circuit polynomials that each polynomial is equivalent to a cone.

The algorithm traverses the circuit from the primary outputs, picks a fanout (line 3 of Fig. 7) and backtracks it, until the algorithm reaches either another fanout or primary inputs of the circuit (line 5). The found region which is called a *cone* is represented as a polynomial. This fanout is also marked as seen (line 8). The algorithm continues until there is no unseen fanout in its input list of fanouts (line 9). In order to derive a polynomial for each cone, first we model each gate with a polynomial based on equations shown in Fig. 4. Then, we treat each of these polynomials as an input of the next connected gate and combine them to have a single polynomial which represents the related cone. By definition, the cones have just one output signal which is member of $\mathbb{B}$. The correspondence polynomial of the output signal is equal to the

```
Inputs: G(V,E) and FO = {g1, g2, …, gt}
Output: F = {f1, f2, …, ft} //circuit polynomails
1   F := {};
2   DO
3       fo := Choose a fanout from FO;
4       New cone C;
5       Backtrack fo until reach either primary
                  inputs or another fanout;
6       Add all gates to C;
7       F := F ∪ makePoly(C);
8       Mark fo as seen;
9   WHILE (there is still unseen fanout)
```

Fig. 7. Fanout free region and circuit polynomials extraction algorithm (Cone Extraction Algorithm).

mathematical manipulation of input signal. So it should also be member of $\mathbb{B}$.

Let us consider a part of a large adder circuit shown in Fig. 6 again. Although extracting half adders and full adders is not possible in most cases, we could extract cones (as shown in Fig. 8) by applying Cone Extraction Algorithm to this circuit. The corresponding polynomials are as follows:

$$f_1 := u - (a + b + c + f - 2ab - 2ac - 2bc - 2af - 2bf - 2cf + 4abc + 4abf + 4acf + 4bcf - 8abcf$$

$$f_2 := f - (d + e - 2de)$$

$$f_3 := v - (ab + ac + ad + ae + de + bf + cf - 2abc - 2abd - 2acd - 2abe - 2ace - 2ade - 2abf - 2acf - 2bcf + 2abcd + 2abce + 2abde + 2acde - 2bcde + 4abcf + 2abdf + 2acdf + 2abef + 2acef - 2bdef - 2cdef - 4abcdf - 4abcef + 4bcde)$$

4.3. Verification problem formulation

After using Cone Extraction Algorithm, the gate level implementation of the circuit is modeled as a set of polynomials. If $cone_i$ has m_i inputs, its correspondence polynomial maps $\mathbb{B}^{m_i}$ to $\mathbb{B}$ (i.e., $f_i : \mathbb{B}^{m_i} \rightarrow \mathbb{B}$). We denote these polynomials as $F = \{f_1, \ldots, f_s\}$.

$$cone_i : \ f_i(x_1, x_2, \ldots, x_m) \ mod 2$$

After extracting polynomials from the circuit, we need to generate a Groebner basis. This phase, however, is not needed because instead of computing Groebner basis, we have derived an order for polynomial terms in such a way that the circuit polynomials become themselves Groebner basis, as discussed before. To this end, the only phase should be done is to see whether the specification is reduced to zero by the circuit polynomials or not.

If the circuit has M primary inputs and N primary outputs, the specification polynomial is considered as $f_{spec} : \mathbb{B}^M \rightarrow \mathbb{B}^N$. As $\mathbb{B} \subset \mathbb{Z}$ and both specification polynomials and circuit polynomials have coefficients in $\mathbb{Z}$ so we consider ring $\mathbb{Z}[x_1, x_2, \ldots, x_n]/2^N$ as computational ring $\mathbb{Z}[X]/2^N$. Therefore, the verification problem becomes a membership testing of f_{spec} in ideal $\mathbf{I}\langle f_1, \ldots, f_s\rangle$ over $\mathbb{Z}[X]/2^N$. With respect to Definition 9, f_{spec} is a member of Ideal $\mathbf{I}$ if it vanishes on $V(I)$. On the other hand, we know that $V(I) = V(G)$ if G is Groebner basis of ideal $\mathbf{I}$. Therefore, f_{spec} would be a member of Ideal $\mathbf{I}$ if it can be reduced to zero by polynomials of G. To check this, instead of using the Buchberger's algorithm we have employed the concept of F4 algorithm that will be explained in the next section.

5. Improving Groebner basis reduction

The polynomial reduction (polynomial division) f w.r.t Groebner basis is the most computationally intensive part of our verification technique. This reduction becomes a bottleneck when the circuit is very large and the polynomial set $F = \{f_1, f_2, \ldots, f_s\}$ is extremely large. This reduction can be done by using existing computer algebra systems e.g., SINGULAR [23] which is a general-purpose computer algebra tool. It should be noted that it does not have a special data structure for Groebner basis reduction in the case of arithmetic verification problems. Moreover, SINGULAR also limits the number of variables so that it can be used in the system up to 32,767 variables and therefore its usage is limited to small circuits [30].

The polynomial reduction of a set is equal to some sequential polynomial division with respect to a specified monomial ordering. Considering P as the number of polynomials, this division should

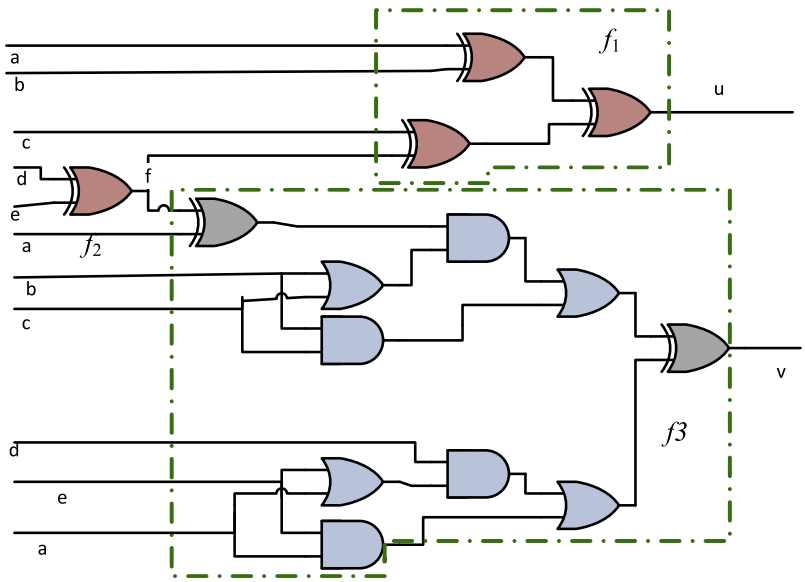

Fig. 8. Finding fanout free regions (cones) in an optimized circuit.

be performed P times. In each iteration, regarding the top variable of each polynomial, we need to divide the terms of the polynomial into two sections: the terms which are independent of the top variable, and the terms which are served as the coefficient of the top variable. Division is continued by multiplication of coefficient of the top variable into polynomial whose leading term is equal to the top variable and subtracting the product from polynomial specification. In each iteration, we also need to simplify polynomials by eliminating terms with same monomials and reduce them to one term. Obviously, the complexity of the entire algorithm in terms of CPU time is really high.

In [29], we have performed such division process by using a canonical word-level decision diagram called Horner Expansion Diagram (HED) [14,15]. As explained in Section 3.3, the HED is a binary graph-based representation which is able to represent polynomial function by factorizing variables recursively. In this work, however, in order to further improve the computation time of the reduction process, we make use of Gaussian elimination which implements polynomial division using row-reduction on a matrix representation of the circuit polynomials. It is obvious that the most computationally intensive part of our verification method, i.e., sequential divisions, has been modeled by Gaussian elimination. The algorithm generates rows of matrix M of specification and circuit polynomials corresponding to our verification problem. The rows and columns of the matrix are setup in such a way that polynomial division can be subsequently performed by applying Gaussian elimination on matrix M.

Now let us show how the Groebner basis reduction $(f_{spec} \underset{G}{\rightarrow} +r)$ can be represented and solved on a matrix. Fig. 9 shows our proposed algorithm. In this algorithm, the polynomial f_{spec} and the set of polynomials $F = \{f_1, f_2, \ldots, f_s\}$ correspond to the specification and circuit implementation, respectively. The set V specifies variable ordering with respect to the topological analysis of the circuit. Note that each variable corresponds to either primary inputs/outputs or internal wires. The set *Mons* is considered as an ordered list of monomials of $F \cup f_{spec}$ which forms the columns of matrix M. As our goal is to reduce the specification polynomial f_{spec} w.r.t. the circuit polynomials (which is Groebner basis), we insert f_{spec} as the first row of the matrix M ($M_{\text{FirstRow}} = \{f_{spec}\}$: line 1 of Fig. 9). Then, for every item in V which is not primary input, this algorithm finds a polynomial $f_t \in F$ such that $\text{LM}(f_t)$ is equal to the ith item of V. This way, it cancels the leading term of f_{spec} by using *TwoRowGausRed* function. This function subtracts the second row from the first row in order to eliminate the first nonzero element of the first row. Our topological order ensures that after encountering the first primary input, the set V contains no other internal variables or primary outputs. This algorithm ends when set V reaches a primary input because there is no circuit polynomial which its leading monomial is equal to primary inputs. Please note that, *Mons* is constructed and sorted once to fill the rows of matrix M. The output of this algorithm is the last row of M whose columns correspond to monomials of *Mons*. This row shows the result of our verification problem $f \underset{G}{\rightarrow} +r$ as a polynomial which is the remainder of Groebner basis reduction of specification polynomial.

One interesting point to be noted here is that, in contrast to the original F4 algorithm [24] which produces all of the rows of the matrix and then tries to reduce the matrix by using Gaussian elimination, our proposed algorithm produces only two rows of this matrix in each iteration and performs the Gaussian elimination on them (*TwoRowGausRed* function: line 7 of Fig. 9). The first row is the intermediate polynomial specification which gets its value from the previous iteration. The algorithm finds the second row in such a way that it can eliminate the first nonzero elements of the first row. Because the columns of the matrix are corresponding to the ordered monomial set, the first nonzero element of intermediate specification is always its leading term. In other words, the algorithm finds a polynomial from the circuit polynomials to cancel the leading term of an intermediate specification polynomial by finding a polynomial which its leading term is equal to the leading term of intermediate specification. The following example shows how our proposed technique works.

Example 2. Let us consider the verification problem of two bit integer multiplier shown in Fig. 4. The specification polynomial is $f_{spec} = 8y_3 + 4y_2 + 2y_1 + y_0 - (4a_1b_1 + 2a_0b_1 + 2a_1b_0 + a_0b_0)$ and circuit polynomials are $f_0 = y_0 - a_0b_0$, $f_1 = w_0 - a_1b_0$, $f_2 = w_1 - a_0b_1$, $f_3 = w_2 - a_1b_1$, $f_4 = y_1 - (w_0 - 2w_0w_1 + w_1)$, $f_5 = w_3 - w_0w_1$, $f_6 = y_2 - (w_3 - 2w_3w_2 + w_2)$ and $f_7 = y_3 - w_3w_2$. As discussed in Section 4, the set $F = \{f_0, f_1, \ldots, f_7\}$ is itself Groebner basis. The final step of the verification problem is to reduce f_{spec}

```
Inputs: Specification polynomial f, circuit
   polynomials F = {f1, f2, …, fs} and variable set
   V= {v1, v2, …, vt} with respect to monomial order
Output: The remainder of Groebner Basis reduction
    /* Columns of M sorted by monomial order > */
1   Mons = sort monomials of F ∪ f;
2   MFirstRow = {f} /*First row of matrix M*/
3   FOR (i=0; i< size(V); i++)
4       IF (V[i] ≠ primary inputs)
5           Identify ft ∈ F such that LM(ft) = V[i];
6           MSecondRow = {ft};
7           MFirstRow = TwoRowGausRed(MFirstRow, MSecondRow);
8       ELSE Break;
9   RETURN MFirstRow;
```

Fig. 9. Improved Groebner basis reduction algorithm which performs Gaussian elimination on a matrix representation of the verification problem.

w.r.t. F (Groebner basis). The topological ordering extracted from the circuit would be $\{y_3 > y_2 > y_1 > y_0\} > \{w_3 > w_2\} > \{w_0 > w_1\} > \{a_1 > b_1 > a_0 > b_0\}$ and therefore $V = \{y_3, y_2, y_1, y_0, w_3, w_2, w_0, w_1, a_1, b_1, a_0, b_0\}$.

The algorithm in Fig. 9 constructs the reduction of matrix M as the following steps. The initial value of $M_{FirstRow}$ is equal to f_{spec} and $Mons = \{y_3, y_2, y_1, y_0, w_3w_2, w_3, w_2, w_0w_1, w_0, w_1, a_1b_1, a_0b_1, a_1b_0, a_0b_0\}$. When $i = 0$, $V[i] = y_3$ which is equal to the leading monomial of f_{spec} so the algorithm finds f_t so that its leading monomial is equal to y_3. It finds $f_7 = y_3 - w_3w_2$. Therefore, $M_{FirstRow} = \{f_{spec}\}$ and $M_{SecondRow} = \{f_7\}$. Then, *TwoRowGausRed* function subtracts $8 \times f_7$ from f_{spec} in order to cancel its leading term y_3. When $i = 1$, $V[i] = y_2$ so $f_t = f_6 = y_2 - (w_3 + w_2 - 2w_3w_2)$. Therefore, $M_{FirstRow} = \{f_{spec} - 8 \times f_7\}$ and $M_{SecondRow} = \{f_6\}$. By continuing in this manner as shown in Fig. 10, when i becomes 8, $V[i]$ contains a primary input and therefore the algorithm finishes. The last row of the reduced matrix corresponds to the reduction $f_{spec} \xrightarrow[G]{} +r$ where $r = 0$ which means the circuit is correctly implemented the two bit integer multiplier.

Although the method of [30] has been applied to Galois field arithmetic circuits, by applying it on integer arithmetic circuits, it would encounter many serious problems. Their approach finds a polynomial for each gate so that N circuit polynomials are obtained if the gate level netlist contains N gates. The number of variables is equal to the number of polynomials plus the number of primary inputs. Table 1 shows that the number of monomials is approximately 2.6 times more than the number of variables in integer array multipliers. The F4 algorithm in [30] executes *mon* times where *mon* is the number of monomials of all polynomials whereas our algorithm executes *var* times where *var* is the number of variables. It is worth noting that by finding the cones, as explained in Section 4.2, the number of polynomials, variables and monomials dramatically decreases. Table 1 shows the merit of our work. Extracting the cones decreases the number of polynomials approximately 2 times. Another point is that as all of leading monomials contain only one variable, the procedure of leading monomial cancelation does not return any new monomial and therefore, unlike [30], we do not need to update and sort the set *Mons* in each iteration. Hence, at the beginning of the algorithm, *Mons* is constructed and sorted once.

Furthermore, in contrast to F4 algorithm used in [30], the algorithm shown in Fig. 9 does not build the whole verification matrix once. It produces only two rows of this matrix in each iteration and performs the Gaussian elimination on them. Therefore, it does not face memory problem in large arithmetic circuits that they have more than 115,000 gates and 300,000 monomials (these parameters show the size of matrix that the algorithm in [30] can deal with in the case of 128 bits integer multiplier). Although they have applied their method on Galois field multipliers which have less complication in comparison with integer arithmetic circuits (because Galois field arithmetic circuits skip carry chains) our verification time is better than that of [30]. Our verification method is implemented as a fully automated tool. Our method takes gate-level implementation of arithmetic circuit f_{imp} and its correspondence specification polynomial f_{spec} as inputs. The tool extracts the cones of the circuit and represents them by equivalence polynomials. It also derives a monomial ordering by considering the level of outputs and inputs of each cone to represent their polynomials.

As the matrix M is being constructed, using algorithm shown in Fig. 9, the Gaussian elimination is also performed to reduce f_{spec} by the circuit polynomials. The following example shows our complete procedure to verify a two bits integer adder.

Example 3. Consider a two bit adder with carry in and carry out. We want to verify the equivalence between its implementation and specification. The specification polynomial is $f_{spec} = 4carry + 2sum_1 + sum_0 - (2a_1 + a_0 + 2b_1 + b_0 + cin)$. Fig. 11 shows the gate level implementation. The first step of our verification problem is extracting polynomial circuits. Each gate has an algebraic functionality and we can derive a polynomial for it. In order to reduce the number of polynomials, the algorithm shown in Fig. 9 is used to extract the cones of this circuit. In Fig. 11 these regions are shown with gates in different colors. Then, the polynomial of each region is extracted and therefore the circuit polynomials will be as follows:

$$f_1 = sum_0 - (a_0 + b_0 + cin - 2a_0b_0 - 2a_0cin - 2b_0cin + 4a_0b_0cin)$$
$$f_2 = w_1 - (a_0b_0 + cinb_0 + a_0cin - 2a_0b_0cin)$$
$$f_3 = sum_1 - (a_1 + b_1 + w_1 - 2a_1b_1 - 2a_1w_1 - 2b_1w_1 + 4a_1b_1w_1)$$
$$f_4 = carry - (a_1b_1 + a_1w_1 + b_1w_1 - 2a_1b_1w_1)$$

By considering the output level of each cone, the variable ordering would be: $carry > sum_1 > sum_0 > w_1 > a_1 > b_1 > a_0 > b_0 > cin$.

	y_3	y_2	y_1	y_0	$w_3.w_2$	w_3	w_2	$w_0.w_1$	w_0	w_1	a_1b_1	a_0b_1	a_1b_0	a_0b_0
First Row = f_{spec}	8	4	2	1	0	0	0	0	0	0	-4	-2	-2	-1
Second Row = f_7	1	0	0	0	-1	0	0	0	0	0	0	0	0	0
$R_1 = f_{spec} - 8f_7$	0	4	2	1	8	0	0	0	0	0	-4	-2	-2	-1
First Iteration														
First Row = R_1	0	4	2	1	8	0	0	0	0	0	-4	-2	-2	-1
Second Row = f_6	0	1	0	0	2	-1	-1	0	0	0	0	0	0	0
$R_2 = R_1 - 4f_6$	0	0	2	1	0	4	4	0	0	0	-4	-2	-2	-1
Second Iteration														
First Row = R_2	0	0	2	1	0	4	4	0	0	0	-4	-2	-2	-1
Second Row = f_4	0	0	1	0	0	0	0	2	-1	-1	0	0	0	0
$R_3 = R_2 - 2f_4$	0	0	0	1	0	4	4	-4	2	2	-4	-2	-2	-1
Third Iteration														
First Row = R_3	0	0	0	1	0	4	4	-4	2	2	-4	-2	-2	-1
Second Row = f_0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1
$R_4 = R_3 - f_0$	0	0	0	0	0	4	4	-4	2	2	-4	-2	-2	0
Fourth Iteration														
First Row = R_4	0	0	0	0	0	4	4	-4	2	2	-4	-2	-2	0
Second Row = f_5	0	0	0	0	0	1	0	-1	0	0	0	0	0	0
$R_5 = R_4 - 4f_5$	0	0	0	0	0	0	4	0	2	2	-4	-2	-2	0
Fifth Iteration														
First Row = R_5	0	0	0	0	0	0	4	0	2	2	-4	-2	-2	0
Second Row = f_3	0	0	0	0	0	0	1	0	0	0	-1	0	0	0
$R_6 = R_5 - 4f_3$	0	0	0	0	0	0	0	0	2	2	0	-2	-2	0
Sixth Iteration														
First Row = R_6	0	0	0	0	0	0	0	0	2	2	0	-2	-2	0
Second Row = f_1	0	0	0	0	0	0	0	0	1	0	0	0	-1	0
$R_7 = R_6 - 2f_1$	0	0	0	0	0	0	0	0	0	2	0	-2	0	0
Seventh Iteration														
First Row = R_7	0	0	0	0	0	0	0	0	0	2	0	-2	0	0
Second Row = f_2	0	0	0	0	0	0	0	0	0	1	0	-1	0	0
$R_8 = R_7 - 2f_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Eighth Iteration														

Fig. 10. Example of proposed Groebner basis reduction technique using Gaussian elimination.

Table 1
Comparison of the number of polynomials, variables and monomial in two verification approaches.

Size	#Gates	Technique in [30]			Our proposed method		
		#Polys	#Vars	#Mons	#Polys	#Vars	#Mons
4	88	88	96	110	40	48	92
8	400	400	416	934	176	192	440
16	1696	1696	1728	4326	736	768	1904
32	6976	6976	7040	18,046	3008	3072	7904
64	28,288	28,288	28,416	73,261	12,160	12,288	32,192
128	113,920	113,920	114,176	294,779	48,896	49,152	129,920
Average improvement in comparison with the technique in [30]					2.33×	2.32×	3.98×

With respect to this variable ordering the polynomial set $F = f_1, f_2, f_3, f_4$ is itself a Groebner basis of ideal $I = \langle f_1, f_2, f_3, f_4 \rangle$. Therefore, the final step of our verification problem is the reduction of f_{spec} w.r.t. Groebner basis, i.e., set F. This step is performed with Gaussian elimination which is shown in Fig. 12. As we can see in this figure, the last row of the matrix contains nothing but zero; which means that the outcome of the verification problem returns true and the gate level implementation is functionally equivalent to the specification.

6. Experimental setup and results

In order to evaluate our proposed arithmetic circuit verification technique, it was implemented in JAVA applied to several arithmetic circuits. All experiments were conducted on a 2.4 GHz with Intel Core™ i5 processor and 4 GB RAM running Linux.

The proposed verification flow has five different parts as shown in Fig. 13. First, gate level netlist of the arithmetic circuit, which its functionality is being verified, is fed into the tool as input. In order

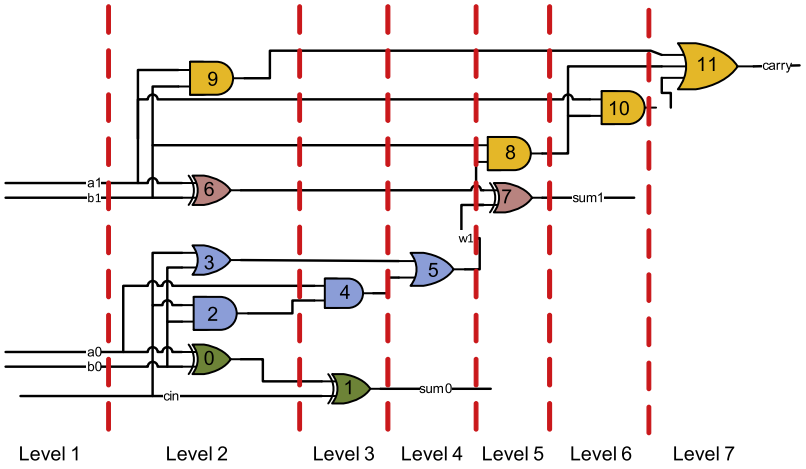

Fig. 11. The gate level implementation of two bit adder with carry-in and carry-out. The gates with same colors show a fanout free region.

$$
\begin{array}{l}
\\
f_{spec} \\
R1 = f_{spec} - 4f_4 \\
R2 = R2 - f_3 \\
R3 = R2 - 2f_1 \\
R4 = R3 - 2f2
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccccccccc}
\text{carry} & \text{sum}_1 & \text{sum}_0 & w_1 & a_1b_1w_1 & a_1w_1 & b_1w_1 & a_1b_1 & a_1 & b_1 & a_0b_0 & a_0\text{cin} & b_0cin & a_0b_0cin & a_0 & b_0 & cin
\end{array} \\
\begin{pmatrix}
4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\
0 & 2 & 1 & 0 & -8 & 4 & 4 & 4 & -2 & -2 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\
0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & -4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

Fig. 12. Reduction result based on Gaussian elimination.

to generate the gate level implementation of such circuits, their high level descriptions are synthesized using Xilinx synthesis tool or any other commercial synthesis tools. This netlist is converted into a graph. In order to produce the graph efficiently, every gate is considered as a node in the graph, which has a unique output and output's name. Then a hash map is taken into account, with outputs' names of the gates as keys and gates themselves as its values. Using this hash map, the corresponding graph of the arithmetic circuit is deployed. In the next step, circuit's cones are found which are used in our main algorithm shown in Fig. 9 as the input. In the third step, these cones are described as polynomials. For every gate existing in a cone, we extract a polynomial based on the equations listed in Fig. 4. Then, these polynomials, which are derived from gates, are combined and one polynomial is produced to represent the cone. In this step, for every cone in the graph, we have one polynomial. These polynomials create an ideal.

In another step, reduction of the specification using Groebner basis (the set of polynomials extracted from the circuit) is performed. For this purpose, we use the algorithm shown in Fig. 9. Please note that in all steps of developing our tool, we have tried to optimize the techniques as much as possible, which can be sensed from the results. For doing so, wherever it was necessary to retrieve data, hash sets were used; whenever it was required to find an item, the lookup was done by having a hash map, which led us to have every item in O(1); every sorting was done using quick sort and tree sort, which are the fastest sorting algorithms in our case. In the last step, we analyze the Groebner basis reduction; if the outcome is zero, it means that the gate level circuit has been implemented correctly; otherwise it indicates that there has been a design error in the implementation.

The developed tool has been evaluated on different arithmetic circuits. To compare the results of this work with other related works, we have considered three experiments:

Experiment 1: in the first experiment, which is labeled as "*Without using reduction techniques*" in the tables, like [13], for each gate one polynomial (regarding the specified topological order) is created, and Groebner basis reduction is performed as a sequence of divisions.

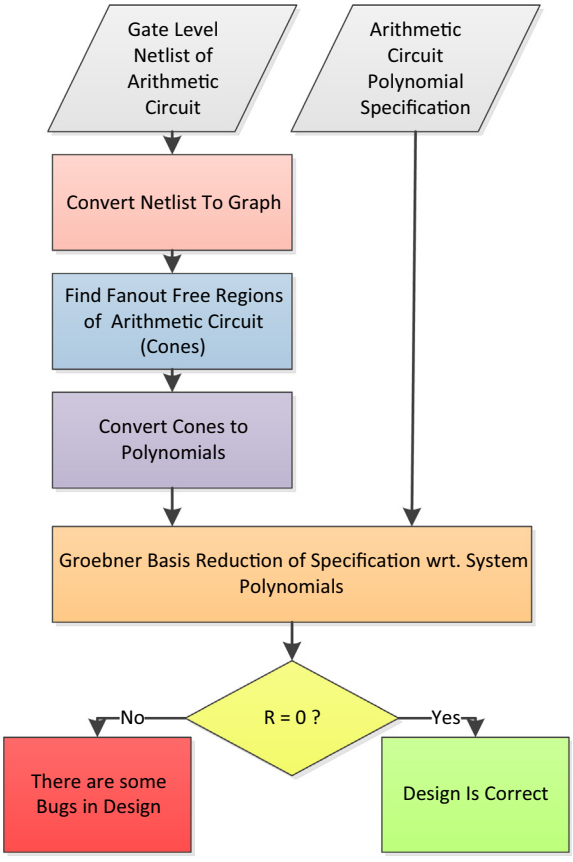

Fig. 13. Flowchart of our verification tool.

Experiment 2: in the second experiment, to improve the performance, an attempt for decreasing the number of polynomials was made. To this end, a library containing those components which are likely to be occurring in the circuits more than the others has been manually created. Based on this library, an automaton is defined, and using this automaton, we look for the components defined in the library in the circuit graph. For every repetitive component and the rest of existing gates of the circuit, a polynomial is defined [29]. This is obvious that this method is efficient when we have information about the structure of arithmetic circuits. The polynomials are gained based on the topological order, which was described earlier. As a result, these polynomials are Groebner basis themselves. To expedite the Groebner basis reduction procedure, every polynomial is represented by the HED. The results of this experiment are labeled as "*Using Automata and HED*" in the tables.

Experiment 3: in the third experiment, the method in [30] has been applied on integer arithmetic circuits. In this method, for each gate existing in the netlist, one polynomial is generated, regarding topological order. To reduce these polynomials, the F4 algorithm has been utilized. As described in Section 5, this work needs to build the whole matrix for implementing F4 algorithm, which decreases its performance significantly. The experimental results of this method are shown as the third experiment in the tables, labeled as "*Technique in* [30]".

To better analyze the performance of the proposed technique in this paper, the number of polynomials and the number of variables are also considered as comparison parameters. Note that in all tables *N*, *#vars* and *#polys* are the size of operands, the number of variables and the number of polynomials, respectively. *MO* shows lack of memory during the execution. Verification experiments are conducted with several different arithmetic structures like carry-lookahead adders, ripple carry adders, array multipliers and their combinations. We also applied our method on optimized circuits where extracting HAs and FAs is really hard and almost impossible.

In Tables 2 and 3, the results of unsigned array multipliers, where the bit width varies from 3 bits to 128 bits, are shown. Table 2 shows that *Experiment 1* is giving timeouts for 64 bits or more. The timeouts, *TO*, is set to 4 h. We have compared these results with those of binary decision diagrams, equivalence checker and SMT solvers in columns *BDD*, *ABC* [33] and *Yices* [31], respectively. As it can be seen, *BDD*, *ABC* and Yices have very poor results. *Experiment 2* is taking 7 times more than the proposed method of this paper. *Experiment 3* suffers from memory out of 96 bits or more and as it can be seen in the table that it is really slow.

Table 3 shows the results of the proposed method for both bug free and buggy multipliers. Our method is able to verify 160 bits multipliers in less than 33 min even in the case of multiple bugs. Table 4 shows the verification time of carry look-ahead adders. As it can be seen in this table, *Experiment 1* is giving timeout for 96 bit or more. The performance of *Experiment 2* is not good enough because repetitive components cannot be extracted efficiently in carry look–ahead adders. The result of *Experiment 3* shows that the technique in [30] is more than four orders of magnitudes slower than our proposed method and it is giving memory out (MO) for 128 bits and more.

It should be noted that we have improved our previous work [29] in two ways: (1) by looking for cones in the circuit in order to reduce the number of polynomials when we do not have any information about the repetitive components in the circuit and (2) using a two-row matrix representation and Gaussian elimination to perform the reduction process more efficiently. This way, we are able to verify large arithmetic circuits in appropriate run times.

Table 5 shows the verification time of arithmetic circuits that implement the function of $A \times B + C \times D$. In these circuits the bit width varies from 4 bits to 128 bits. As shown in this table, *Experiment 1* faces timeouts for 64 bits or more. *Experiment 2* is more than 10 times slower than the proposed method. *Experiment 3* faces memory out for 96 bits or more. The results show that the proposed method is three orders of magnitudes faster than other techniques mentioned in Table 5.

Table 6 shows the verification result of arithmetic circuits that their function is $A \times B + C + D$. The parameter *N* shows the size of inputs *A* and *B* which vary from 4 bits to 128 bits. In this table, if

Table 2
Verification time of $N \times N$ array multipliers using several methods (MO = out of 4 GB memory; TO = timeout of 4 h; CPU time is given in seconds).

Size (*N*)	Without using reduction techniques			BDD	ABC	Yices	Using automata and HED [29]			Technique in [30]		
	#vars	#polys	CPU time	CPU time	CPU time	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time
3	39	33	0.87	0.09	3.87	0.02	15	7	0.00	39	33	0.43
4	80	72	1.38	0.19	5.24	0.05	21	13	0.00	80	72	0.65
8	384	368	14.16	0.343	35.72	2.54	63	47	0.05	384	368	0.86
16	1664	1632	110.76	MO	253.09	TO	235	203	0.38	1664	1632	2.14
32	6912	6848	4054.32	MO	5405.41	TO	959	895	5.72	6912	6848	712.59
64	28,160	28,032	TO	MO	TO	TO	3004	2876	143.95	28,160	28,032	1754.02
96	63,360	63,168	TO	MO	TO	TO	7931	7739	701.54	63,360	63,168	MO
128	212,862	212,606	TO	MO	TO	TO	24,751	24,495	4965.29	212,862	212,606	MO

Table 3
Verification time of bug-free and buggy $N \times N$ array multipliers using our proposed method (CPU time is given in seconds).

Size (*N*)	#vars	#polys	Bug-free circuit CPU time	Buggy circuit (Single) CPU time	Buggy circuit (Multiple) CPU time
3	27	21	0.01	0.03	0.03
4	48	40	0.02	0.02	0.05
8	192	176	0.06	0.09	0.08
16	768	736	0.17	0.18	0.13
32	3072	3008	1.21	1.51	1.57
64	12,288	12,160	20.08	22.81	23.08
96	28,950	28,758	121.61	123.78	125.09
128	49,152	48,896	535.51	572.85	583.14
160	94,440	94,120	1762.8	1894.2	1925.08

Table 4
Verification time of carry-lookahead adders (MO = out of 4 GB memory; TO = timeout of 4 h; CPU time is given in seconds).

Size (N)	Without using reduction techniques			Using Automata and HED [29]			Technique in [30]			Our proposed method		
	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time
4	36	28	0.69	17	9	0.01	36	28	0.05	23	15	0.04
8	143	127	5.13	49	33	0.19	143	127	0.32	34	18	0.15
16	891	759	41.05	226	194	0.29	891	759	0.74	341	309	0.11
32	4044	3980	1613.27	587	523	4.09	4044	3980	39.87	1521	1457	0.84
64	11,656	11,528	5701.06	2191	2063	90.85	11,656	11,528	683.03	3139	3011	1.97
96	29,974	29,782	TO	5519	5327	276.14	29,974	29,782	1098.00	7018	6826	5.04
128	121,030	120,774	TO	19,327	14,071	1558.31	121,030	120,774	MO	25,288	25,032	135.2
Average improvement in comparison with "without using reduction techniques"				6.10×	12.95×	497.50×	0.00×	0.00×	0.00004×	7.72×	7.85×	12106.61×

Table 5
Verification time of A×B + C × D (MO = out of 4 GB memory; TO = timeout of 4 h; CPU time is given in seconds).

Size (*N*)	Without using reduction techniques			Using Automata and HED [29]			Technique in [30]			Our proposed method		
	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time
4	176	168	6.58	74	58	0.06	176	168	0.07	112	96	0.01
8	980	964	54.78	278	246	0.47	980	964	0.69	416	384	0.14
16	3884	3852	1553.87	1135	1071	6.84	3884	3852	7.54	1600	1536	0.57
32	13,824	13,760	7512.00	7970	7842	706.32	13,824	13,760	1426.87	6272	6144	5.02
64	60,315	60,187	TO	23,128	22,872	4636.03	60,315	60,187	8531.02	24,832	24,576	122.58
96	177,110	176,918	TO	53,709	53,325	7848.61	177,110	176,918	MO	64,083	62,891	612.62
128	327,981	327,725	TO	104,023	103,511	13707.05	327,981	327,725	MO	140,680	140,604	1235.05
Average improvement in comparison with "without using reduction techniques"				3.07×	3.09×	64.05×	0.00×	0.00×	1.60×	2.46×	1.55×	1316.37×

Table 6
Verification time of optimized circuit $A \times B + C + D$ where the first adder is a carry-lookahead adder and the second one is a ripple carry adder (MO = out of 4 GB memory; TO = timeout of 4 h; CPU time is given in seconds).

Size (*N*)	Without using reduction techniques			Using Automata and HED [29]			Technique in [30]			Our proposed method		
	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time	#vars	#polys	CPU time
4	102	94	1.79	51	43	0.05	102	94	0.06	56	48	0.01
8	541	525	20.37	289	273	0.47	541	525	0.89	271	255	0.10
16	2170	2138	185.79	806	774	4.99	2170	2138	9.02	724	692	0.29
32	9098	9034	4680.20	3744	3680	183.75	9098	9034	212.15	3206	3142	9.65
64	36,839	36,839	TO	19,672	19,544	3847.17	36,839	36,839	13238.0	17,585	17,457	174.01
96	99,976	99,784	TO	44,402	44,210	7400.14	99,976	99,784	TO	37,042	36,850	365.24
128	387,260	387,004	TO	175,477	175,221	TO	387,260	387,004	MO	83,320	83,064	6872.26
Average improvement in comparison with "without using reduction techniques"				2.19×	2.20×	2.97×	0.00×	0.00×	1.49×	3.77×	3.78×	349.91×

the size of *A* and *B* inputs are *N*, the size of inputs *C* and *D* will be 2*N* and 2*N* + 1, respectively. The addition needed in $A \times B + C$ has been implemented by carry look-ahead adders. These circuits are delay-optimized designs obtained by commercial synthesis tool. The netlist of the optimized circuit is the input of our verification tool. The goal of this experiment is to show that the proposed method does not depend on the structure of arithmetic circuits and it has comparable results for optimized circuits in which extracting repetitive components like HAs and FAs are not easy. The results in Table 6 show that all of the experiments except the proposed method are failed to verify these circuits.

7. Conclusion and future works

A formal method to verify arithmetic circuits using computer algebra techniques has been proposed in this paper. Our method formally proves that the given f_{spec} and gate-level combinational arithmetic circuit f_{imp} are equivalent. The verification problem is formulated as membership testing of the specification polynomial f_{spec} in an ideal where ideal $\mathbf{I} = \langle f_1, f_2, \ldots, f_s \rangle$ generated by extracted polynomials from the circuit. Subsequently, a Groebner basis **G** of the ideal **I** should be computed and the ideal membership test can be decided via Groebner basis reduction. We analyze the circuit topology and drive an order that makes the initial polynomial set itself a Groebner basis. The final step of verification is the reduction of the polynomial f_{spec} w.r.t Groebner basis. To have this reduction efficiently, we have performed it through Gaussian elimination on a matrix representation of the problem.

In our future work, we will focus on formal debugging of arithmetic circuits based on Groebner basis method. We will detect and correct single and multiple design errors that occurred in arithmetic circuits based on analyzing the remainder of the Groebner basis reduction.

Acknowledgment

This research was in part supported by a grant from IPM (No. CS1393-4-46).

References

[1] D. Stoffel, W. Kunz, Equivalence checking of arithmetic circuits on the arithmetic bit level, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 23 (5) (2006) 586–597.
[2] R. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. (TC) 35 (8) (1986) 677–691.
[3] C.W. Barlett, D.L. Dill, J.R. Levitt, A decision procedure for bit-vector arithmetic, in: Proc. of the 35th Annual Design Automation Conference (DAC), June 1998, pp. 522–527.
[4] M. Ciesielski, P. Kalla, S. Askar, Taylor expansion diagrams: a canonical representation for verification of data flow designs, IEEE Trans. Comput. 55 (9) (2006) 1188–1201.
[5] Y. Watanabe, N. Homma, T. Aoki, T. Higuchi, Application of symbolic computer algebra to arithmetic circuit verification, in: Proc. of Intl. Conf. on Computer Design (ICCD), 2007, pp. 25–32.
[6] B. Buchberger, An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimensional Polynomial Ideal, Ph.D. Thesis, Institute of Mathematics, Univ. Innsbruck, Innsbruck, Austria, 1965.
[7] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, STABLE: a new QF-BV SMT solver for hard verification problems combining Boolean reasoning with computer algebra, Proc. Design Autom. Test Eur. (DATE) (2011) 1–6.
[8] B. Alizadeh, P. Behnam, Formal equivalence verification and debugging techniques with auto-correction mechanism for RTL designs, Elsevier J. Microprocess. Microsyst. (MICPRO) 37 (8) (2013) 1108–1121.
[9] M.A. Bastish, T. Ahmad, A. Rossi, M. Ciesielski, Algebraic approach to arithmetic design verification, Proc. Formal Methods Comput.-Aid. Des. (FMCAD) (2011) 67–71.
[10] D. Cox, J. Little, D. O'Shea, Ideals, Varieties, and Algorithms, Springer, New York, 1997.
[11] Q. Tran, M. Vardi, Groebner bases computation in boolean rings for symbolic model checking, Proc. Model. Simul. (MOAS) (2007) 440–445.
[12] B. Buchberger, Some properties of Groebner-bases for polynomial ideals, ACM SIGSAM Bull. 10 (4) (1976) 19–24.
[13] J. Lv, P. Kalla, F. Enescu, Efficient Gröbner basis reductions for formal verification of galois field multipliers, Proc. Des. Autom. Test Eur. (DATE) (2012) 899–904.
[14] B. Alizadeh, M. Fujita, Modular datapath optimization and verification based on modular-HED, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 29 (9) (2010) 1422–1435.
[15] B. Alizadeh, A formal approach to debug polynomial datapath designs, in: Proc. of Asia and South Pacific Design Automation Conference (ASPDAC), 2012, pp. 683–688.
[16] S.Y. Huang, K.T. Cheng, Formal Equivalence Checking and Design Debugging, Springer, 1998.
[17] R.E. Bryant, Y.A. Chen, Verification of arithmetic circuits with binary moment diagrams, in: Proc. of Design Automation Conference (DAC), 1995, pp. 535–541.
[18] M. Wedler, D. Stoffel, R. Brinkmann, W. Kunz, A normalization method for arithmetic data-path verification, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 26 (11) (2007) 1909–1922.
[19] O. Sarbishei, B. Alizadeh, M. Fujita, Arithmetic circuit verification without looking for internal equivalences, in: Proc. of IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE), 2008, pp. 7–16.
[20] O. Sarbishei, M. Tabandeh, B. Alizadeh, M. Fujita, A formal approach for debugging arithmetic circuits, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 28 (5) (2009) 742–754.
[21] N. Shekhar, P. Kalla, F. Enescu, Equivalence verification of polynomial datapaths using ideal membership testing, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 26 (7) (2007) 1320–1330.
[22] G.D. Micheli, Synthesis and Optimization of Digital Circuits, McGraw Hill, Inc., New York, 1994.
[23] W. Decker, G.-M. Greuel, G. Pfister, H. Schönemann, SINGULAR 3-1-3 A Computer Algebra System for Polynomial Computations, 2011. <http://www.singular.uni-kl.de>.
[24] J.C. Faugére, A new efficient algorithm for computing Groebner bases (F4), J. Pure Appl. Algebra 139 (June) (1999) 61–88.
[25] S. Horeth, Drechsler, Formal verification of word-level specifications, Proc. Des. Autom. Test Eur. (DATE) (1999) 52–58.
[26] S. Sadeghi-kohan, P. Behnam, B. Alizadeh, M. Fujita, Z. Navabi, Improving polynomial datapath debugging with HEDs, Proc. Eur. Test Sympos. (ETS) (2014) 1–4.
[27] J. Harrison, L. Thery, Extending HOL theorem prover with a computer algebra system to reason about the reals, Proc. Higher Order Logic Theor. Prov. Appl. 780 (1993) 174–184.
[28] J. Smith, G. DeMicheli, Polynomial methods for component matching and verification, in: Proc. of IEEE International Conference on Computer-Aided Design (ICCAD), 1998, pp. 678–685.
[29] F. Farahmandi, B. Alizadeh, Z. Navabi, Effective combination of algebraic techniques and decision diagrams to formally verify large arithmetic circuits, in: IEEE Computer Society Symposium on VLSI (ISVLSI), 2014, pp. 338–343.
[30] J. Lv, P. Kalla, F. Enescu, Efficient Groebner basis reduction for formal verification of galois field arithmetic circuits, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. (TCAD) 32 (9) (2013) 1409–1420.
[31] Yices: An SMT Solver. <http://yices.csl.sri.com/index.shtml>.
[32] Z. Zhou, W. Burleson, Equivalence checking of datapaths based on canonical arithmetic expressions, in: Proc. of the 32th annual Design Automation Conference (DAC), 1995, pp. 546–551.
[33] R. Brayton, A. Mishchenko, ABC: an academic industrial-strength verification tool, in: Proc. of International Conference on Computer-Aided Verification (CAV), 2010, pp. 24–40.
[34] H. Haghbayan, B. Alizadeh, A. Rahmani, P. Liljeberg, H. Tenhunen, Automated formal approach for debugging dividers using dynamic specification, in: Proc. of International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014, pp. 264–269.

Farimah Farahmnadi received her B.S and M.S degrees in computer engineering from University of Tehran, Tehran, Iran, in 2010 and 2013, respectively. She is currently a PhD student at the Embedded System Lab in department of Computer and Information Science and Engineering, University of Florida. Her research interests include formal verification and debugging of digital systems and post-silicon validation.

Dr. Bijan Alizadeh received his B.Sc., M.Sc. and PhD. degrees in computer engineering from the University of Tehran, Iran in 1996, 1999 and 2004, respectively. Prior to joining the University of Tehran in 2010 as an assistant professor, he was an assistant professor at Sharif University of Technology from 2005 to 2007 and a Research Associate in VLSI Design and Education Center (VDEC), University of Tokyo from 2007 to 2010. He has developed many academic tools in verification, synthesis and other aspects of EDA. His current research interests include verification and synthesis in high level and system level designs as well as embedded system design methodologies.

View publication stats