Effective Combination of Algebraic Techniques and Decision Diagrams to Formally Verify Large Arithmetic Circuits

Farimah Farahmandi, Bijan Alizadeh, Zain Navabi
School of Electrical and Computer Engineering, University of Tehran
{f.farahmandi, b.alizadeh, navabi}@ut.ac.ir

Abstract – Arithmetic circuits require a verification process to prove that the gate level circuit is functionally equivalent to a high level specification. This paper presents an automatic equivalence checking technique to verify combinational arithmetic circuits at bit level. In order to efficiently verify gate level arithmetic circuits, we make use of computer algebra based approach so that the circuit and the specification are modeled in polynomial system and the verification problem is formulated as polynomial reduction techniques using Groebner basis of circuit polynomial corresponding ideal. To overcome costly Groebner basis computation as well as intensive polynomial reduction, we make use of a canonical decision diagram named Horner Expansion Diagram (HED), derive a suitable term order to represent and manipulate polynomials efficiently and find repetitive components based on automata. To evaluate the effectiveness of our verification technique, we have applied it to very large arithmetic circuits including multipliers. Preliminary experimental results show that the proposed verification technique is scalable enough so that large multipliers can efficiently be verified in reasonable run time and memory usage.

I. INTRODUCTION

Effort for verification of digital system designs grows as their size and complexity increase. In many practical designs such as Digital Signal Processing (DSP) for multimedia applications, arithmetic circuits are the main part of their datapaths. So verification of arithmetic circuits in a fast and precise way plays an important role in a digital system design flow. Several methods have been proposed to check an arithmetic circuit against its specification at a higher level of abstraction. Most of them are based on canonical graph-based representations like Binary Decision Diagrams (BDDs) that 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 to verification of bit-level implementations using theory of Groebner basis have been proposed [6][8][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 which combines Groebner basis and decision diagrams to effectively verify large arithmetic circuits. Fig. 1 shows our proposed verification technique which takes a gate level circuit and its high level specification as inputs and automatically checks whether the implementation is functionally equivalent to the specification or not. For doing so, first of all, the gate level circuit is converted to Boolean polynomials by looking for repetitive components like cascaded XORs, carry generation logics and so on. In Section V, we will discuss how to find such repetitive components based on automata. Then the specification is described as a polynomial function and finally the equivalence checking is performed by polynomial manipulation and reduction based on Groebner basis [10].

As will be discussed in Section III, to check whether a given polynomial is a member of a polynomial system or not, a sequence of polynomial manipulations including polynomial divisions and multiplications are needed which makes such a processing complicated in terms of the run time and memory requirement. In order to overcome intensive polynomial reduction and Groebner basis computation, we make use of a canonical decision diagram called Horner Expansion Diagram (HED) [14] and also derive a topological order for polynomials' terms as discussed in Section V. This way, we will be able to verify large arithmetic circuits in reasonable run time and memory usage as experimentally shown in Section VI.

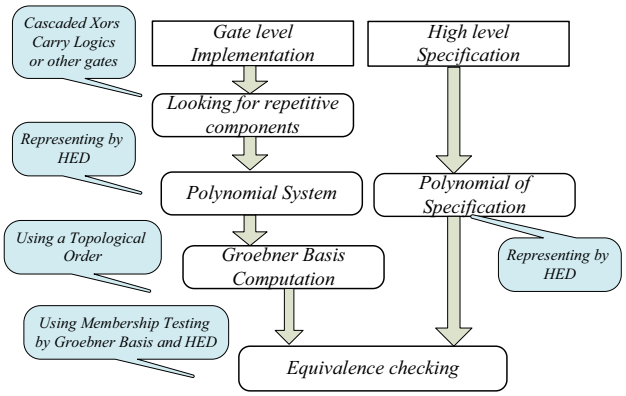

Fig. 1. Proposed verification technique.

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 repetitive components. In contrast to [18][19][20], this method is applicable even when half adders and full adders cannot be extracted due to the local optimizations (Section V).
- Determining a suitable variable ordering in such a way that leading terms of the original polynomials become relatively prime to avoid Groebner basis computation (Sections III and V).
- Using a decision diagram, i.e., HED, to perform polynomial reduction efficiently so that a sequence of polynomial divisions and multiplications can be efficiently computed (Section V).
- Showing empirical results to prove that our proposed verification technique enables us to verify very large industrial arithmetic circuits within practical time.

The paper is organized as follows: Section II provides a brief review of related work. In Section III we briefly describe some mathematical background. Section IV presents our proposed verification technique in more details. This paper ends with the experimental results in Section V and the conclusion in Section

II. RELATED WORK

There is a large amount of literature on equivalence verification of arithmetic circuits against their specifications. In the literature of graph-based canonical representation, the

978-1-4799-3765-3/14 $31.00 © 2014 IEEE
DOI 10.1109/ISVLSI.2014.109

ordered Binary Decision Diagrams (BDDs) based verification techniques have been successfully used to verify small-sized arithmetic circuits [2]. However, they are not able to deal with large arithmetic circuits especially multipliers due to the existence of many varieties of different architectures as well as the lack of compact bit-level canonical representation. Word level extensions of BDDs like *BMDs [3] and TEDs [4] represent arithmetic circuits in more efficient ways but they need word level information of circuits which are not available at the gate level of abstraction.

Another approach in verifying arithmetic circuits is to extract arithmetic operations like half-adders (HAs) and full-adders (FAs) from gate-level implementation and then generate an arithmetic model called Arithmetic Bit Level (ABL) to be compared with the higher-level description [18]. This technique needs an exhaustive process to check different XOR tree structures to verify carry signals, which becomes exponential in terms of run-time complexity. The verification approach presented in [19] utilizes a more efficient reverse-engineering process in extracting a network of half-adders from the gate level implementation. This approach uses a bit level adder (BLA) representation during the adder-extraction process. The BLA model is robust for several arithmetic circuit architectures. However, this method cannot handle *pin-swap* optimizations, related to the cases where partial product bits can be swapped with each other without changing the functionality of the circuit. The debugging algorithm in [20] is a much more efficient approach, which solves almost all of the above-mentioned problems.

Several methods based on computer symbolic algebra are taken into account for verification of arithmetic circuits as well. In [5][6], 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 authors of [13] have presented a formal approach to model and verify multiplier circuits over Galois fields F_2^k using a computer algebra based technique. 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 [7], 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 [8] 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, the same authors have proposed a technique to directly generate individual output polynomials in terms of primary inputs [7]. However, there is no systematic way for comparing such polynomials against the specification and preprocessing of such very large polynomials is crucial for successful performance of the normal form algorithm. In [11] and [9], arithmetic circuits are represented as a network of half adders, 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 using standard linear programming solvers and comparing it with the reference signature provided by the designer. It should be noted that if half adders and full adders could not be extracted due to optimizations done by synthesis tools, such a technique would not be applicable.

III. ALGEBRAIC PRELIMINARIES

In this section, we briefly describe Groebner basis concepts. These mathematical backgrounds are mostly based on [10]. Let m = $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$ be a monomial in $x_1, x_2, \ldots, x_n$ where all of the exponents $\alpha_1, \alpha_2, \ldots, \alpha_n$ and variables are nonnegative integers. Let K be a computable field and $K[x_1, x_2, \ldots, x_n]$ be the polynomial ring in n variables so that its elements restricted to 0 and 1. A finite linear combination of monomials in $x_1, x_2, \ldots, x_n$ is a polynomial *f*. Let $f_1, f_2, \ldots, f_s$ be polynomials in $K[x_1, x_2, \ldots, x_n]$.Then
$< f_1, f_2, \ldots, f_s >= \{ \sum_{i=1}^{s} h_i f_i : h_1, h_2, \ldots, h_s \in K[x_1, x_2, \ldots, x_n] \}$ is 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 3.1*: 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 3.2: *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* $t = g - \frac{c \,.\, M}{LC(f).\, LM(f)} f$. *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* $\underset{f_1}{\rightarrow} \underset{f_2}{\rightarrow} \ldots \underset{f_s}{\rightarrow} h$. *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.*

Definition 3.3: *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 (**I**) is denoted as <LT (**I**)> where* $LT(\boldsymbol{I}) = \{cx^\alpha : ther\ exists\ f \in \boldsymbol{I}\ with\ LT(f) = cx^\alpha\}$.

Definition 3.4: *The set G of ideal **I** is a Groebner basis if and only if for all polynomial* $f \in \boldsymbol{I}$ *the remainder of reducing f by polynomials of G is zero. This process is called membership testing of f over ideal **I** and denoted by* $\forall f \in \boldsymbol{I}, f \underset{G}{\rightarrow} 0$.

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 3.5*: V(I) is the affine variety of ideal I such that*

$V(I) = \{(a_1, a_2, .., a_n) | f((a_1, a_2, .., a_n) = 0 \; for \; all \; f \in I\}$
where $a_i \in \boldsymbol{K}$*.For every generating set of ideal I such as Groebner basis G, we have V(I)= V(G).*

Definition 3.6*: Polynomial f is member of ideal I if it vanishes on V(I) or* $V(I) \subset V(f)$.

In order to find out whether polynomial f is member of ideal I or not, we need to check the possibility of vanishing f on V (I). Regarding to definition 3.7, if set G is Groebner basis of ideal I, V(I) is equal to V(G). So for the membership testing, we need to compute Groebner basis.

Definition 3.7 (S-polynomial): *Let* $f, g \in \mathrm{K}[x_1, x_2, \ldots, x_n]$ *be nonzero polynomials. The S-polynomial of f and g is defined as*
Spoly (f, g) $= \frac{\mathrm{LCM}(\mathrm{LM}(f),\mathrm{LM}(g))}{LT(f)} . f - \frac{\mathrm{LCM}(\mathrm{LM}(f),\mathrm{LM}(g))}{LT(g)} . g.$
Where LCM(a, b) is a notation for the least common multiple of a and b).

Example 3.1: Let $f = 6\, x_1^4 x_2^5 + 24 x_1^2 - x_2$ and $g = 2x_1^2 x_2^7 + 4x_2^3 + 2x_3$ and we have $x_1 > x_2 > x_3$. The S-polynomial of *f* and *g* is defined below as $LCM\ (x_1^4 x_2^5, x_1^2 x_2^7)$ is equal to $x_1^4 x_2^7$.

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

It is obvious that S-polynomial computation cancels leading terms of the polynomials.

```
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.

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 requirement 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* [10]. So it implies that when the size of *F* increases, the verification process may be very slow or in the worst-case may be infeasible.

IV. PROPOSED VERIFICATION APPROACH

Formally, the verification problem of an arithmetic circuit is characterized by a given specification $f_{spec} := (\mathrm{Y} - \sum_i A_i \times B_i + C_i)$ where $A = a_0 + 2a_1 + \cdots + 2^{m-1} a_{m-1}$, $B = b_0 + 2b_1 + \cdots + 2^{n-1} b_{m-1}$ and $C = c_0 + 2c_1 + \cdots + 2^{m-1} c_{m-1}$ and $Y = y_0 + 2y_1 + \cdots + 2^{t-1} y_{t-1}$ where $a_i, b_i, y_i \in \{0, 1\}$. We are also given a gate level circuit with $\{a_0, \ldots, a_{m-1}\}$, $\{b_0, \ldots, b_{m-1}\}$ and $\{c_0, \ldots, c_{m-1}\}$ as the primary inputs and $\{y_0, \ldots, y_{t-1}\}$ as the primary outputs.

To check the equivalence between the polynomial specification (f_{spec}) and the gate level implementation, we model the gate level circuit with a series of polynomials. Each polynomial models the operation of correspondence gate as shown below:

$$z = NOT(a) \Rightarrow f = z - (1 - a)$$
$$z = AND(a, b) \Rightarrow f = z - a.b$$
$$z = OR(a, b) \Rightarrow f = z - (a + b - a.b)$$
$$z = XOR(a, b) \Rightarrow f = z - (a + b - 2.a.b)$$

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 verification test can become a membership test of f_{spec} over ideal **I**. To check whether f_{spec} is a member of ideal **I**, its Groebner Basis needs to be computed.

Please keep in mind that the most time consuming task in Buchberger's algorithm is the reduction of the S-polynomials modulo *G*. A simple idea to speed up such computations is to reduce the number of S-polynomials to be computed. 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 $\mathrm{LCM}(\mathrm{LM}(f_i),\mathrm{LM}(f_j)) = \mathrm{LM}(f_i) \times \mathrm{LM}(f_j)$ then $\mathrm{Spoly}(f_i, f_j) \xrightarrow[G]{} 0$. 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 Buchberger's algorithm does not need to consider 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 monomials, then Spoly(*f, g*)$\xrightarrow[G+]{}$0 for all pairs *f* and *g*. Consequently, the polynomials $\{f_1, f_2, \ldots, f_s\}$ extracted from the circuit (corresponding to ideal **I**) and represented using such a term order would itself constitute a Groebner basis of **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.

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 gate polynomial such that its output places in 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 [13]. Note that, in our case, variables are all binaries, so their degrees never increase. Another point is the fact that outputs of gates 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. 3.

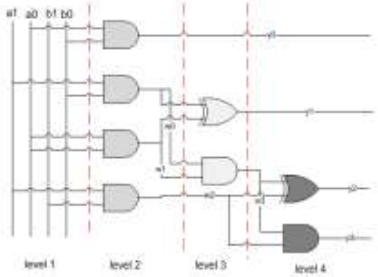

Fig. 3. Gate level implementation of a two bit unsigned multiplier

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 and the implementation are described as follows:

$$f_{spec} := 8y_3 + 4y_2 + 2y_1 + y_0 - (2a_1 + a_0).(2b_1 + b_0)$$

$$f_0 := y_0 - a_0.b_0 \quad LM(f_0) = y_0$$
$$f_1 := w_0 - a_1.b_0 \quad LM(f_1) = w_0$$
$$f_2 := w_1 - a_0.b_1 \quad LM(f_2) = w_1$$
$$f_3 := w_2 - a_1.b_1 \quad LM(f_3) = w_2$$
$$f_4 := y_1 - (w_0 + w_1 - 2w_0.w_1) \quad LM(f_4) = y_1$$
$$f_5 := w_3 - w_0.w_1 \quad LM(f_5) = w_3$$
$$f_6 := y_2 - (w_3 + w_2 - 2w_3.w_2) \quad LM(f_6) = y_2$$
$$f_7 := y_3 - w_3.w_2 \quad LM(f_7) = y_3$$

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. 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 timeout for large circuits. Our idea to alleviate this issue is to partition the circuits into suitable components which are repeated in the circuit. In the following subsections we will discuss our idea in more details.

A. How to Extract Repetitive Components

In order to reduce the numbers of polynomials, we have tried 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. Using this method, the number of polynomials decreases significantly. Please note that unlike the technique presented in [8] which looks for HAs and FAs as repetitive components, we are looking for different forms of repetitive components including HAs and FAs. This way, if the proposed verification technique is unable to map a series of gates to HAs or FAs (due to optimizations), it is still able to verify the circuit by creating other forms of repetitive components and therefore the specification reduction over this set of new polynomials needs to be done.

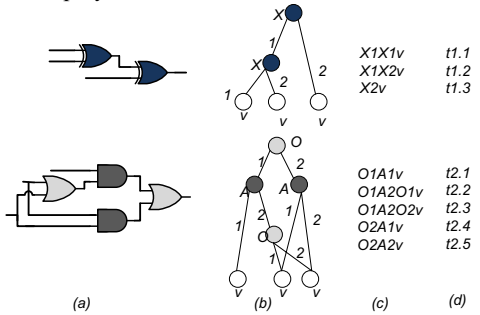

Fig. 4. Cell library example (a) Components of cell library. (b) Pattern trees. (c) Pattern strings. (d) Pattern tree identifiers.

To find repetitive components, a tree-based matching algorithm which utilizes Automata [22] is applied. We define our components in a cell-library and we use 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. This method supports library descriptions in terms of base functions, each one encoded by a single character. 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. We also have corresponding pattern strings and pattern tree identifiers for each library component as shown in Fig. 4.

The automaton processes strings that encode paths in the pattern trees. The automaton consists of a set of states, a set of transitions that are portioned into "goto" and "failure" corresponding to the status of detection of a character in a string and an output function that signals the full detection of a string. The automaton is constructed once for all elements of a given library. A part of automaton which detects library elements of Fig. 4 is shown in Fig. 5.

To find repetitive components, the subject tree is visited in a bottom up fashion, and for each vertex all strings that are corresponding to paths to the leaves are processed. A match exists if the strings are recognized by the automaton as belonging to the same pattern tree. The algorithm complexity is linear in the size of the subject graph.

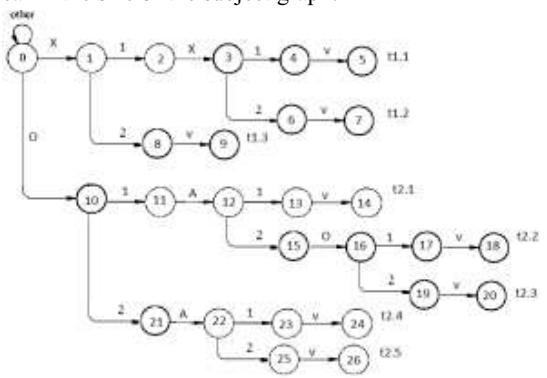

Fig. 5. The matching automaton. The "goto" transitions are indicated by edges.

Fig. 6 shows a part of a large adder circuit [1]. In this circuit the signal u computes the sum of signals a, b, c, d and 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.

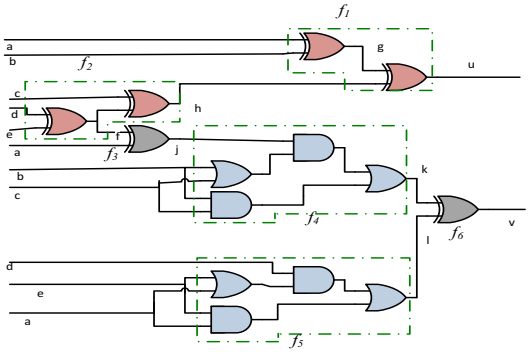

Fig. 6. Finding repetitive components and corresponding polynomials in an optimized circuit

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

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

B. Improving Polynomial Reduction Phase Using HED

As mentioned before, final step of verification is reducing specification polynomials with respect to circuit polynomials which are Groebner basis. 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 be performed P times. In each iteration regarding to the top variable of each polynomial, we need to divide the terms of the polynomial into two sections: the terms which are independent of top variable and the terms which are served as the coefficient of top variable. Division is continued by multiplication of coefficient of top variable into polynomial which is leading term is equal to the top variable and subtracting the product from polynomial specification of that iteration. In each iteration, we also need to simplify polynomials by eliminating terms with same monomials and reduce them to one term. So the complexity of the entire algorithm is really high.

In order to perform such division process efficiently we need a suitable representation. The HED is a binary graph-based representation which is able to represent polynomial function by factorizing variables recursively as shown in (1), where const is a term which is independent of variable y, while linear is another term which is served as the coefficient of variable y [15].

$$F(y,\ldots)=F(y=0,\ldots)+y\times[F'(y=0,\ldots)+\ldots]= \text{const} + y\times \text{linear} \quad (1)$$

Definition 4.1 (HED Representation): *HED is a directed acyclic graph G= (VR, 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 val(v)∈ Z as its attribute. A Variable node v has three attributes: an integer variable var(v) and two children const(v) and linear(v) ∈ {V, C}. Hence, each vertex v in HED denotes an integer function f^v defined recursively as follows:*

- *If v ∈ C (is a Constant node), then f^v = val(v).*
- *If v ∈ V (is a Variable node), then f^v = const(v)+var(v)*linear(v).*

Now suppose we need to compute f_{spec}/f_i where the top variable is x. After representing f_{spec} and f_i in HED, we will have:

$f_{spec} = const(f_{spec}) + linear(f_{spec})*x \quad f_i = x - const(f_i)$

and therefore instead of computing f_{spec}/f_i directly, we just need to consider the following computation:

$$f_{spec} = const(f_{spec}) - linear(f_{spec}) * const(f_i)$$

Obviously, we can access to constant and linear parts of polynomial specification w.r.t. a top variable with O(1).

After completion of the reduction, if the reminder is not equal to 0, it means the design contains errors. Otherwise, the design has been implemented correctly.

V. EXPERIMENTAL RESULTS

In order to evaluate our proposed arithmetic circuit verification technique, it has been implemented in C++ and applied to several arithmetic circuits. In order to generate the gate level implementation of such circuits, their high level descriptions are synthesized using commercial synthesis tools. All experiments were conducted on a 2.4 GHz with Intel Core™ i5 processor and 4GB RAM running Linux.

In the first experiment, we have extracted individual polynomials for all gates of $N\times N$ unsigned multipliers where N varies from 3 to 128. We have used HED neither for representing our polynomials nor for reducing the specification over the polynomials set as a Groebner Basis. Taking this method into account, we encountered time outs for 64 bits and more (the time out, TO, is set to 6 hours) as indicated in major column "without using automata to find repetitive components" of Table 1. Note that in this table, N, #vars and #polys are the size of multiplier (both operands are N-bits), the number of variables and the number of polynomials, respectively. MO shows lack of memory during of execution. In order to compare our results with those of BDDs and SMT solvers, we have converted the polynomial specification into gate-level golden circuit. Then we have created a miter with the specification and implementation and check their equivalency. These results are also shown in minor columns "BDDs" and "Yices" of Table 1. We have also run Singular as a computer algebra system [23] and reported the results in minor column "SINGULAR" of Table 1.

We have improved our method in two ways: 1) by looking for repetitive components in the circuit in order to reduce the number of polynomials and 2) using HED representation to have the reduction process done more efficiently. This way, we were able to verify large arithmetic circuits in appropriate run time. To illustrate the effectiveness of the proposed verification method, we have compared our results with those of Singular. We have used Singular to compute Groebner basis computation and polynomial reduction (ideal membership testing). Our results show that Singular encounters a memory explosion for up to 4-bit multipliers when it is not able to find repetitive components. When the number of polynomials is reduced by using automata, Singular has failed to verify 24-bit multipliers. The results are shown in major column "with using automata to find repetitive components" of Table 1. In the second experiment, we have applied our proposed verification technique to two complex arithmetic circuits: 1) a*b+c*d (the results are shown in Table 2) and a*b+c+d+e (the results are shown in Table 3). As shown in these tables SINGULAR can only verify the correctness up to 16-bits arithmetic circuits and the Groebner basis engine encounters a memory explosion while our method is able to verify 128-bit arithmetic circuits in reasonable run time. We have performed this experiment to show that our method is not limited to just some types of circuits and it can easily be applied to very large arithmetic circuits. Furthermore, the results in Table 3 indicate that our verification method is efficient to verify those circuits in which extracting HAs and FAs can rarely be done.

Table 1. Verification of $N \times N$ multipliers with and without using automata and HED

Without using automata to find repetitive components							With using automata to find repetitive components			
Size (N)	**#vars**	**#polys**	**Without Using HED (sec)**	**SINGULAR (sec)**	**BDD (sec)**	**Yices (sec)**	**#vars**	**#polys**	**Using HED (sec)**	**SINGULAR (sec)**
3	39	33	0.87	2.73	0.09	0.02	15	7	0.00	0.00
4	80	72	1.38	7.42	0.19	0.05	21	13	0.00	0.03
8	384	368	14.16	MO	0.343	2.54	63	47	0.05	3.41
16	1664	1632	110.76	MO	MO	TO	235	203	0.38	7.19
24	3744	3696	617.82	MO	MO	TO	521	473	0.79	MO
32	6912	6848	4054.32	MO	MO	TO	959	895	5.72	MO
64	28160	28032	TO	MO	MO	TO	3004	2876	143.95	MO
96	63360	63168	TO	MO	MO	TO	7931	7739	701.54	MO
128	212862	212606	TO	MO	MO	TO	24751	24495	4965.29	MO

Table 2. Verification of a*b+c*d with using automata and HED representation

Size (N)	**#vars**	**#polys**	**CPU time (sec)**	**SINGULAR (sec)**
3	35	23	0.00	0.03
4	74	58	0.06	4.52
8	278	246	0.47	7.27
16	1135	1071	6.84	MO
24	3064	2968	149.55	MO
32	7970	7842	706.32	MO
64	23128	22872	4636.03	MO
96	53709	53325	7848.61	MO
128	104023	103511	13707.05	MO

Table 3. Verification of delay optimized circuit of a*b+c+d+e with using automata and HED representation

Size (N)	**#vars**	**#polys**	**CPU time (sec)**	**SINGULAR (sec)**
3	54	39	0.04	2.60
4	82	62	0.06	4.52
8	215	175	0.33	6.79
16	1028	948	6.05	MO
24	2489	2369	112.58	MO
32	4915	4755	225.97	MO
64	7836	7516	676.42	MO
96	22040	21560	4370.14	MO
128	40917	40277	5928.14	MO

VI. CONCLUSION

In this paper, a formal method for modeling and verifying arithmetic circuits using a computer algebra based approach has been proposed. The verification test was formulated as a membership testing over the ideal generated by polynomials extracted from the circuit. We analyze the circuit topology and drive an order that makes polynomial set itself a Groebner basis. This test can be done by polynomial specification reduction over Groebner basis. For enhancing our method, instead of generating polynomials for each gate, we have generated polynomials for each repetitive component which has significantly improved the performance. To further improve our method in terms of the run time and memory usage, we have represented the polynomials by a graph based representation called HED.

References

[1] D. Stoffel, and W. Kunz, "Equivalence Checking of Arithmetic Circuits on the Arithmetic Bit Level", in *Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions*, 2004, pp.587-597.

[2] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in *IEEE Trans. on Computers*, August 1986, vol. 35, pp. 677–691.

[3] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning, " in *Proc. Design Automation Conf.*, June 2001, pp. 232-237.

[4] M. Ciesielski, P. Kalla, and S. Askar, "Taylor Expansion Diagrams: A Canonical Representation for Verification of Data Flow Designs," *IEEE Trans. on Computers*, vol. 55, no. 9, pp. 1188–1201, Sept. 2006.

[5] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Application of Symbolic Computer Algebra to Arithmetic Circuit Verification," in *Proc. Intl. Conf. on Computer Design*, 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, and W. Kunz, "STABLE: A new QF-BV SMT Solver for hard Verification Problems combining Boolean Reasoning with Computer Algebra," in *Proc. Design Automation and Test in Europe*, 2011.

[8] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Greuel, "An Algebraic Approach for Proving Data Correctness in Arithmetic Data Paths," in *Proc. Intl. Conf. on Computer-Aided Verification*. July 2008, pp. 473–486, Springer-Verlag Berlin Heidelberg 2008.

[9] M.A. Bastish, T. Ahmad, A.Rossi and M.Ciesielski, "Algebraic Approach to Arithmetic Design Verification," in *Formal Methods in Computer-Aided Design (FMCAD),* 2011.

[10] D. Cox, J. Little, and 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," in *MOAS'07*, 2007, pp. 440-445.

[12] B. Buchberger, "Some properties of Groebner-bases for polynomial ideals," *ACM SIGSAM Bulletin*, vol. 10, no. 4, pp. 19.24, 1976.

[13] J. Lv, P. Kalla and F. Enescu, "Efficient Gröbner Basis reductions for formal verification of galois field multipliers," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 899-904.

[14] B. Alizadeh, and M. Fujita, "Modular Datapath Optimization and Verification based on Modular-HED", IEEE Transactions on Computer-aided design (TCAD), vol. 29, no. 9, pp. 1422-1435, September 2010.

[15] B. Alizadeh, "A Formal Approach to Debug Polynomial Datapath Designs ", in *ASPDAC*, 2012, pp. 683-688.

[16] S. Y. Huang and K. T. Cheng, Formal Equivalence Checking and Design Debugging, Springer, June 1998.

[17] R.E. Bryant and Y.A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Proc. of the 32nd Design Automation Conference*, San Francisco, pp. 535-541, June 1995.

[18] M. Wedler, D. Stoffel, R. Brinkmann, W. Kunz, "A Normalization Method for Arithmetic Data-Path Verification," *IEEE Trans. on CAD,* Vol 26, No 11, 2007, pp. 1909-1922.

[19] O. Sarbishei, B. Alizadeh and 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 and M. Fujita, "A Formal Approach for Debugging Arithmetic Circuits," *IEEE Trans. On CAD,* Vol 28, No 5, May 2009, pp. 742-754

[21] N.Shekhar, P.Kalla and F. Enescu, "Equivalence verification of Polynomial Datapaths Using Ideal Membership Testing," *in IEEE Trans. On Computers,* vol. 55, no. 9, pp. 1188-1201, Sept. 2006.

[22] G. D. Micheli, *synthesis and optimization of digital circuits*. New York: McGraw Hill. Inc, 1994.

[23] Greuel, G.-M., Pfister, G., Schönemann, H., 2011. SINGULAR 3.1.3 A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern. URL: http://www.singular.uni-kl.de.