Resolving Unknown Components in Arithmetic Circuits using Computer Algebra Methods

Vikas Rao[1], Utkarsh Gupta[1], Irina Ilioaea[2], Priyank Kalla[1], and Florian Enescu[2]
[1]Electrical & Computer Engineering, University of Utah
[2]Mathematics & Statistics, Georgia State University

***Abstract*—Resolving an unknown component is a fundamental problem encountered in logic synthesis for engineering change orders, post-verification debugging and automatic correction of digital circuits. Contemporary techniques rely on iterative/incremental application of SAT solving and Craig interpolation to realize the functionality of (or resolve) the unknown components. While these techniques have achieved some success for control-dominated applications (random logic circuits), they are infeasible in resolving the unknown components in arithmetic circuits. This paper describes an algebraic approach to resolve the functionality of an unknown component in an arithmetic circuit so that the circuit implementation matches a given specification. Our approach is formulated as a polynomial ideal membership test. We go on to pose the problem as a synthesis challenge and explore the solution space of the unknown component using concepts from the quotient of ideals. We propose a Gröbner basis based algorithm for a systematic, goal driven search for implementable solutions. The paper presents results on some experiments performed over various finite field arithmetic circuits to compare the efficiency of our approach against recent methods.**

I. Introduction

Verifying functional correctness of gate-level arithmetic circuits is still a significant challenge owing to increasing design size and functional complexity. In cases where verification detects the presence of a bug, considerable amount of manual intervention is required to localize the bug and introduce a correction, thus making it a resource intensive process. Traditional automated debugging techniques based on simulation, decision procedures such as Binary Decision Diagrams (BDDs) [1] and SAT solvers [2], demand bit-blasting of the circuit and are hence considered inefficient models to verify complex datapath designs. Due to the inherent algebraic nature of computations in such designs, symbolic algebra algorithms are considered more appropriate for their verification.

Within a symbolic algebra environment, a given circuit implementation is modeled as a set of polynomials that generate an ideal. The verification goal here is then to prove that this polynomial ideal satisfies a given golden specification. This is solved using an ideal membership test by performing a series of Gröbner basis reductions under a defined term order. If verification fails, we deem the circuit as buggy and go on to find the faulty gate in order to rectify it. The current challenge and scope of this paper is to realize the correct implementation for this buggy component. Identifying the buggy gate is a much harder problem to solve; it is part of future work, and beyond the scope of this paper. Once a particular gate has been identified as corrupted and rectifiable, we label the gate as *an unknown component* and go on to find the correct functionality to be implemented by this component such that the entire circuit conforms to the given reference specification.

A. Previous work

The most recent and relevant approach [3], [4] resolves the unknown component problem using an incremental SAT formulation. The paper models the unknown component in a given circuit(Ckt) as a LUT by using transformation variables(X). The solution to these variables implements the desired logic function so that the resulting circuit becomes logically equivalent to a given specification $Spec()$. Let $Ckt(X, In)$ be the formula corresponding to the given circuit with possible transformations, where In is the set of all primary inputs to the circuit. This can be formulated naturally as a two-level QBF with a existential quantifier followed by a universal quantifier as shown below:

$$\exists X.\forall In.\ Ckt(X,In) = Spec(In) : \tag{1}$$

The two level QBF is then solved by repeatedly applying the below SAT formulation:

1) Let Target=($Ckt(X, In) \neq spec(In)$). Let k be the number of test vectors, initialized to zero. Let $TestSet$ be the set of all generated test patterns, initialized to the empty set.
2) Check if Target is satisfiable.
3) If SAT, $k = k + 1$ and record the solution as $TestSet = TestSet \cup in_k$. The Target is then updated as Target = (Target$(X, In)) \wedge (Ckt(X, in_k) = Spec(in_k))$, and go to step 2.
4) If UNSAT, we have all the required test set patterns $\{in_1 \ldots in_k\}$. Now, check if: $(Ckt(X, in_1) = Spec(in_1)) \wedge (Ckt(X, in_2) = Spec(in_2)) \wedge \ldots (Ckt(X, in_k) = Spec\ (in_k))$ is satisfiable.
5) If SAT, then any solution X is a correct set of transformation, while an UNSAT result proves that there does not exist a correct set of transformation.

The work in [5] poses the unknown component formulation as a camouflaged circuit model and tries to de-obfuscate several types of camouflaging techniques using incremental SAT solving. The approach used in [6] inserts logic corrector MUXs on the unknown sub-circuits and relies on SAT solvers to realize the functionality.

Despite using state-of-the-art SAT solvers, all the above approaches fail to verify large and complex finite field arithmetic circuits. The solvers still model the problem as decision procedures and, as demonstrated by our experimental results, are shown to be inefficient in solving verification problems on multiplier circuits beyond 12-bits.

The technique from Farahmandi et al. [7] deals with automatic debugging and correction using computer algebra concepts. The authors use function extraction [8] with a specific term order [9] to do equivalence checking, subsequently generating a remainder in case of failure. The approach then finds all possible assignments to variables of the remainder such that it generates a non-zero value. This test set helps arrive at a pruned gate list for bug localization. The procedure then takes every gate in the pruned list, starting from primary inputs, and tries to match the appeared remainders pattern. It does so by computing the difference between the polynomial computed at the output of the suspicious gate against the polynomial computed by a probable set of gate corrections. The coefficient computation [10] during pattern matching relies heavily on the half-adder based circuit structure. The paper doesn't discuss the ambiguities in weight calculations when the gate structure differs from the given topology. The approach is not complete in the case when there are redundant gates in the circuit as we found through our experiments.

B. Contribution

We are given a gate-level circuit C, with one of the gates $\mathcal{G}_i \in C$ marked as unknown component. The problem is to compute the function implemented by this gate such that it matches a given specification polynomial f or a given golden reference model circuit C. We utilize concepts from symbolic computer algebra to realize the function implemented by this unknown component. The circuit is modeled by a set of polynomials $F = \{f_1, \dots, f_s\}$, with f_i being the unknown polynomial corresponding to the gate $\mathcal{G}_i$. We consider the ideal generated by these polynomials F, and exploit concepts from ideal membership testing to compute the function implemented by the unknown component. Using the concepts of ideal membership, elimination ideals, quotients of ideals – and their computation using the Gröbner basis algorithm – we show how multiple functions for the unknown component can be explored. This paper seeks to put forth the underlying theory, outline the verification challenges, and present a complete approach to resolve an unknown component in finite field arithmetic circuits. We also discuss some experimental results and draw a comparison to the SAT-based approach.

So far, the theory is developed and validated only for finite field arithmetic circuits. We believe that our approach is also applicable to integer arithmetic circuits. However, we do not yet have a *provably complete* algorithmic approach as a quantification procedure over integer rings, though we exepect to have it resolved by the time the workshop will be held. So our claims, approach and experiments are restricted to finite field arithmetic circuits.

II. Preliminaries: Notation and Background

This section reviews some basic concepts from symbolic computer algebra that we utilize in this paper.

Let $\mathbb{F}_q$ denote the finite field of q elements, where $q = p^k$ is a prime power. To model functions over k-bit vector operands, we use $q = 2^k$, i.e. the finite field $\mathbb{F}_{2^k}$ of 2^k elements. The field $\mathbb{F}_{2^k}$ is constructed as $\mathbb{F}_{2^k} = \mathbb{F}_2[X] \pmod{P(X)}$, where $\mathbb{F}_2$ is the field of two elements $\{0, 1\}$, and $P(X)$ is an irreducible polynomial of degree k. Moreover, we use α to denote a root of the irreducible polynomial, i.e. $P(\alpha) = 0$.

Let $R = \mathbb{F}_q[x_1, \dots, x_n]$ be the polynomial ring in variables $x_1, \dots, x_n$ with coefficients in $\mathbb{F}_q$. A polynomial $f \in R$ is written as a finite sum of terms $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$. Here $c_1, \dots, c_t$ are coefficients and $X_1, \dots, X_t$ are monomials, i.e. power products of the type $x_1^{e_1} \cdot x_2^{e_2} \cdots x_n^{e_n}$, $e_j \in \mathbb{Z}_{\geq 0}$. To systematically manipulate the polynomials, a monomial order $>$ (also called a term order) is imposed on the polynomial ring. Subject to $>$, $X_1 > X_2 > \cdots > X_t$, and $lt(f) = c_1X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. Also, for a polynomial f, $tail(f) = f - lt(f)$. In this work, we are mostly concerned with *lexicographic* (lex) term orders.

Logic gates of a circuit can be modeled with polynomials in $\mathbb{F}_2$. As $\mathbb{F}_{2^k} \supset \mathbb{F}_2$, these polynomials can also be construed as polynomials in $\mathbb{F}_{2^k}$. The mapping $\mathbb{B} \mapsto \mathbb{F}_2$ is given as:

$$
\begin{aligned}
z &= \neg a \rightarrow z + a + 1 \pmod 2\\
z &= a \wedge b \rightarrow z + a \cdot b \pmod 2\\
z &= a \vee b \rightarrow z + a + b + a \cdot b \pmod 2\\
z &= a \oplus b \rightarrow z + a + b \pmod 2
\end{aligned} \tag{2}
$$

Polynomial Reduction via division: Let f, g be polynomials. If $lm(f)$ is divisible by $lm(g)$, then we say that *f is reducible to r* modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{lt(f)}{lt(g)} \cdot g$. Similarly, f can be *reduced w.r.t. a set of polynomials* $F = \{f_1, \dots, f_s\}$ to obtain a remainder r. This reduction is denoted as $f \xrightarrow{F}_+ r$, and the remainder r has the property that no term in r is divisible by the leading term of any polynomial f_j in F. Algorithm 1 (from [11]) shows a step-by-step procedure to perform this reduction.

Algorithm 1 Multivariate Reduction of f by $F = \{f_1, \dots, f_s\}$

procedure $multi\_var\_division(f, \{f_1, \dots, f_s\}, f_j \neq 0)$
  $u_j \leftarrow 0;\ r \leftarrow 0,\ h \leftarrow f$
  while $h \neq 0$ **do**
    if $\exists j$ s.t. $lm(f_j) \mid lm(h)$ **then**
      choose j least s.t. $lm(f_j) \mid lm(h)$
      $u_j = u_j + \frac{lt(h)}{lt(f_j)}$
      $h = h - \frac{lt(h)}{lt(f_j)} f_j$
    else
      $r = r + lt(h)$
      $h = h - lt(h)$
  return $(\{u_1, \dots, u_s\}, r)$

The algorithm initializes h with the polynomial f and cancels its leading term by some polynomial f_j. If the leading term $lt(h)$ cannot be canceled by any $lt(f_j)$, then it is added to the final remainder r and the process is repeated until all the terms in h are analyzed. The algorithm also returns the set of quotients $\{u_1, \dots, u_s\}$ of division of f by $\{f_1, \dots, f_s\}$, respectively.

Definition II.1. Given a ring $R = \mathbb{F}_q[x_1, \dots, x_n]$ and a set of polynomials $F = \{f_1, \dots, f_s\}$ from R, the ideal generated by F is $J = \langle F \rangle \subseteq R$:

$$J = \langle f_1, \dots, f_s \rangle = \{h_1 \cdot f_1 + \cdots + h_s \cdot f_s : h_1, \dots, h_s \in R\}. \quad (3)$$

The polynomials $f_1, \dots, f_s$ are called the generators or the basis of ideal J.

An ideal may have many different sets of generators, i.e. it is possible to have $J = \langle f_1, \dots, f_s \rangle = \langle g_1, \dots, g_t \rangle = \cdots = \langle h_1, \dots, h_r \rangle$. A Gröbner basis (GB) of an ideal is one such generating set $G = \{g_1, \dots, g_t\}$ that is a canonical representation of the ideal.

Definition II.2. [**Gröbner Basis**] [11]: For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner basis of J iff $\forall f \in J$, $f \neq 0$, there exists $g_i \in G$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \ \forall f \in J : f \neq 0 \ \exists g_i \in G : lm(g_i) \mid lm(f)$.

Then $J = \langle F \rangle = \langle G \rangle$ holds and $G = GB(J)$ forms a basis for J. The Gröbner basis for an ideal J can be computed using the Buchberger's algorithm [12], reproduced in Alg. 2. It takes as input a set of polynomials $\{f_1, \dots, f_s\}$ and computes its GB $G = \{g_1, g_2, \cdots, g_t\}$.

Algorithm 2 Buchberger's Algorithm

Require: $F = \{f_1, \dots, f_s\}$
Ensure: $G = \{g_1, \dots, g_t\}$
$G := F$;
repeat
  $G' := G$
  for each pair $\{f_i, f_j\}, i \neq j$ in G' **do**
    $Spoly(f_i, f_j) \xrightarrow{G'}_+ h$
    if $h \neq 0$ **then**
      $G := G \cup \{h\}$
until $G = G'$

The algorithm takes pairs of polynomials (f_i, f_j) from the basis and computes their S-polynomial $Spoly(f_i, f_j)$:

$$Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j \quad (4)$$

where $L = LCM(lt(f_i), lt(f_j))$. The $Spoly(f_i, f_j)$ is then reduced $w.r.t.$ the polynomials in G to obtain remainder h. If h is non-zero, it is added to G. The process is repeated for all unique polynomial pairs, including those generated by the newly added elements h. The algorithm terminates when there are no new non-zero h generated from the set G. $Spoly(f_i, f_j) \xrightarrow{G}_+ h$ reductions cancel the leading terms of polynomials $\{f_i, f_j\}$, and generate h with new leading terms, providing additional information about the ideal.

Buchberger's algorithm can be easily extended to output not just the Gröbner basis $G = \{g_1, \dots, g_t\}$ but also a $t \times s$ matrix M with polynomial entries such that:

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_t \end{bmatrix} = M \cdot \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} \quad (5)$$

An important property of Gröbner bases is that as a decision procedure, they allow for membership testing of a polynomial in an ideal.

Lemma II.1. (*Ideal Membership Testing)* Let $G = GB(J) = \{g_1, \dots, g_t\}$ and f be any polynomial. Then $f \in J \iff f \xrightarrow{G}_+ 0$.

In other words, a polynomial f is a member of ideal J iff division by the Gröbner basis of J gives remainder 0. Consequently, f can be written as a linear combination (with polynomial coefficients) of the elements of the Gröbner basis:

$$f = u_1 g_1 + u_2 g_2 + \cdots + u_t g_t, \quad (6)$$

where $u_i's$ correspond to the quotients of division $f \xrightarrow{g_1, \dots, g_t}_+ 0$. Eqns. (6) and (5) can be combined to give f as combination of the original polynomials $f_1, \dots, f_s$:

$$f = v_1 f_1 + \cdots + v_s f_s. \quad (7)$$

We utilize this concept to identify the unknown component.

A. *Operations on Ideals*

Given two ideals $J_1 = \langle f_1, \dots, f_s \rangle, J_2 = \langle e_1, \dots, e_r \rangle$, their sum is given as $J_1 + J_2 = \langle f_1, \dots, f_s, e_1 \dots, e_r \rangle$. For all elements $\alpha \in \mathbb{F}_q, \alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ *vanishes* (evaluates to zero) everywhere in $\mathbb{F}_q$, and is called the vanishing polynomial of the field. We denote $F_0 = \{x_1^q - x_1, \dots, x_n^q - x_n\}$ the set of vanishing polynomials, and similarly $J_0 = \langle F_0 \rangle$ denotes the ideal of all vanishing polynomials in the ring R.

Definition II.3 (*Elimination Ideal [13]*)**.** Given an ideal $J = \langle f_1, \dots, f_s \rangle \subset \mathbb{F}_q[x_1, \dots, x_n]$, the l-th elimination ideal J_l is defined as $J_l = J \cap \mathbb{F}_q[x_{l+1}, \dots, x_n]$.

The ideal J_l is called an elimination ideal because the variables $x_1, \dots, x_{l-1}$ have been eliminated. Generators of the l-th elimination ideal can be computed using Gröbner bases.

Theorem II.1 (*Elimination Theorem [13]*)**.** Given an ideal $J \subset R$ and its GB G $w.r.t.$ the lexicographical (lex) order on the variables where $x_1 > x_2 > \cdots > x_n$, then for every $0 \leq l \leq n$ we denote by G_l the GB of l-th elimination ideal of J and compute it as $G_l = G \cap \mathbb{F}_q[x_{l+1}, \dots, x_n]$

Definition II.4. (*Quotient of Ideals*) If J_1 and J_2 are ideals in a ring R, then $J_1 : J_2$ is the set $\{f \in R \mid f \cdot g \in J_1, \forall g \in J_2\}$ and is called the **ideal quotient** of J_1 by J_2, also called the **colon ideal**.

Given generators of J_1 and J_2, the generators of $J_1 : J_2$ can also be computed using Gröbner bases with elimination (lex) term orders. We refer the reader to Section 2.3 in [13] for further details.

III. Theory and Procedure

In this section we describe the theory, and procedures used to arrive at a function and its implementation for the unknown component. We also show how to explore the solution space for implementable functions.

Consider a specification polynomial f and a gate-level circuit C that implements f. Model the circuit by way of polynomials $F = \{f_1, \dots, f_s\} \in \mathbb{F}_q[x_1, \dots, x_n]$, where variables $x_1, \dots, x_n$ denote the nets in the circuit. The set F generates an ideal $J = \langle F \rangle$, and let $J_0 = \langle x_l^q - x_l : 1 \leq l \leq n \rangle$ be the set of all vanishing polynomials.

As described in [14], the equivalence check between f and C can be formulated as an ideal membership test that checks if $f \in J + J_0$. Thus, one can compute a Gröbner basis $G = GB(J + J_0) = \{g_1, \dots, g_t\}$, and check if $f \xrightarrow{G}_+ 0$? If the circuit C indeed implements f, $f \xrightarrow{G}_+ 0$ and f can be written as a linear combination of $g_1 \dots, g_t$, and also of $f_1 \dots, f_s$, by virtue of Eqns. (5) and (6).

The Gröbner basis algorithm has very high exponential complexity ($q^{O(n)}$ in our case). In [14], it was further shown that this complexity can be overcome by deriving a specialized term order by analyzing the topology of the given circuit. It relies on the condition that *when the leading terms of all polynomials in a generating set $F = \{f_1, \dots, f_s\}$ are relatively prime, then F already constitutes a GB, i.e. $F = GB(F)$*. We restate the result:

Proposition III.1. (From [14]) Let C be any arbitrary combinational circuit. Let $\{x_1, \dots, x_n\}$ denote the set of all variables (signals) in C. Starting from the primary outputs, perform a *reverse topological traversal* of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a *lex* term order $>$ to represent each gate as a polynomial f_i, s.t. $f_i = x_i + tail(f_i)$. Then the set of all polynomials $\{f_1, \dots, f_s\}$ forms a Gröbner basis G, as $lt(f_i) = x_i$ and $lt(f_j) = x_j$ for $i \neq j$ are relatively prime. This term order $>$ is called the **Reverse Topological Term Order (RTTO)**.

Imposition of RTTO $>$ on the polynomials of the circuit has the effect of making every gate output variable x_i a leading term of f_i. Since every gate output is unique, $lm(f_i) = x_i, lm(f_j) = x_j \forall i \neq j$ become relatively prime. As a result, the set F already constitutes a GB ($G = F$). Moreover, it was further shown in [14] that under RTTO, the set $F \cup F_0$ forms the Gröbner basis of $J + J_0$. As a result, the verification test can be carried out simply by the division of f modulo the Gröbner basis $F \cup F_0$ and checking if the remainder is 0; i.e. $f \xrightarrow{F, F_0}_+ r$, and checking if $r = 0$? When C does implement f, $r = 0$ and $f = u_1 f_1 + \cdots + u_s f_s + \sum_{i=1}^{n} H_i \cdot (x_i^q - x_i)$.

An important effect of RTTO $>$ is that each gate $\mathcal{G}_i$ is represented by a polynomial of the type $f_i = x_i + \text{tail}(f_i)$. RTTO ensures that every variable x_j that appears in $\text{tail}(f_i)$ satisfies $x_i > x_j$. These properties will be exploited in our technique.

A. The unknown component

Now consider that verification has been performed between f and C, and it is found that C **does not** implement f. Further, assume that post-verification debugging identifies the gate $\mathcal{G}_i \in C$ with output net x_i where a correction can be synthesized and implemented to meet the specification. We consider the gate $\mathcal{G}_i$ as the unknown component and attempt to identify a function for $\mathcal{G}_i$. More precisely, *we have to compute a polynomial $f_i = x_i + tail(f_i)$ that identifies a function implementable at gate $\mathcal{G}_i$* such that the circuit C conforms to the specification f.

First, we address the problem of computing a polynomial f_i of the form $f_i : x_i + P(X)$ as an implementation of $\mathcal{G}_i$, where x_i is the output of the gate $\mathcal{G}_i$, $X \subset \{x_1, \dots, x_n\}$ is a subset of the nets in the circuit that lie in the fanin cone of the gate $\mathcal{G}_i$, and $P(X) = \text{tail}(f_i)$ is a polynomial in X-variables, with coefficients in $\mathbb{F}_q$. Subsequently, we address the problem of identifying $f_i : x_i + P(X_{im})$ as an implementation of $\mathcal{G}_i$ where $X_{im} \subset \{x_1, \dots, x_n\}$ is a *given* set of variables corresponding to the internal nets of the circuit. This case corresponds to resolving the unknown component with more topological constraints imposed on $P(X_{im})$ by the user; say, when the immediate inputs (X_{im}) of the unknown gate $\mathcal{G}_i$ are known. *Note that here we assume that the unknown component can indeed be composed of the given X_{im} variables.*

As described above, for a correct implementation,
$$f \in \langle f_1, .., f_s \rangle + \langle x_l^q - x_l : 1 \leq l \leq n \rangle.$$

We impose RTTO $>$ on the ring, which ensures that the set $\{f_1, \dots, f_s\} \cup \{x_1^q - x_1, \dots, x_n^q - x_n\}$ itself constitutes a Gröbner basis. Thus $f \xrightarrow{f_1, \dots, f_s, x_l^q - x_l}_+ 0$. Using Lemma II.1, we can rewrite f in terms of its generators as:

$$f = h_1 f_1 + h_2 f_2 + \cdots + h_i f_i + \cdots + h_s f_s + \sum_{l=1}^{n} H_l (x_l^q - x_l), \quad (8)$$

where $h_1, \dots, h_s, H_1, \dots, H_n$ are arbitrary polynomials from the ring R. Substituting $f_i = x_i + P$ for the unknown component in Eqn. (8), we have:

$$\begin{aligned} f = {} & h_1 f_1 + \cdots + h_{i-1} f_{i-1} + h_i x_i + h_i P + \cdots + h_s f_s \\ & + \sum_{l=1}^{n} H_l (x_l^q - x_l) \end{aligned} \quad (9)$$

$$\begin{aligned} & f - h_1 f_1 - \cdots - h_{i-1} f_{i-1} - h_i x_i = h_i P + h_{i+1} f_{i+1} + \\ & \cdots + h_s f_s + \sum_{l=1}^{n} H_l (x_l^q - x_l) \end{aligned} \quad (10)$$

Notice that on the L.H.S. of Eqn. (10), the polynomials $f, f_1, \dots, f_{i-1}$ and the variable x_i are known expressions. Therefore, f can be divided by $f_1, \dots, f_{i-1}$ and x_i to obtain the quotients of the division $h_1, \dots, h_i$ and a remainder r where $r = f - h_1 f_1 - \cdots - h_i x_i$. After h_i is computed (as the quotient of this division by x_i), the R.H.S. of Eqn. (10) consists of $h_i, f_{i+1}, \dots, f_s$ and all the vanishing polynomials $x_l^q - x_l$ as known expressions. This implies that:

$$f - h_1 f_1 - \cdots - h_i x_i \quad \in \langle h_i, f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle \quad (11)$$

$$r \quad \in \langle h_i, f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle \quad (12)$$

This ideal membership implies that r can be written as some polynomial combination of the generators $h_i, f_{i+1}, \ldots, f_s, x_l^q - x_l$. This combination can be identified by first computing the Gröbner basis G of the ideal $\langle h_i, f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle$, and then performing the ideal membership test $r \xrightarrow{G}_+ 0$, while utilizing Eqns. (6) and (7). As a result, we can write:

$$r = h_i' h_i + h_{i+1}' f_{i+1} + \cdots + h_s' f_s + \sum_{l=1}^{n} H_l (x_l^q - x_l). \quad (13)$$

Then $P = h_i'$ is a polynomial that forms the solution to the unknown component problem. Algorithmically, as $P = h_i'$ is computed as a quotient of division, P may contain any variables $x_1, \ldots, x_n$ in its support. However, due to the imposition of RTTO $>$, P will contain only those variables x_j in its support set that are less than x_i in the reverse topological order. In this fashion, the polynomial $f_i : x_i + P(X)$ can be identified to implement the function of the gate $\mathcal{G}_i \in C$ so that C correctly implements f.

Note that in Eqn. (12), while $\{f_{i+1}, \ldots, f_s\}$ constitutes a GB under RTTO, $\{h_i, f_{i+1}, \ldots, f_s\}$ may not, so a GB computation may be required. On the other hand, we may also encounter situations when h_i ends up being a constant. When a constant is a member of an ideal J, then $GB(J) = \{1\}$. To arrive at an implementable solution in this case, we divide $h_i^{'}$ by the constant h_i(multiply the inverse of h_i) and reduce the result by rest of the input polynomials$\{f_{i+1}, \ldots, f_s\}$.

$$h_i^{'} * h_i^{-1} \xrightarrow{f_{i+1}} \xrightarrow{f_{i+2}} \ldots \xrightarrow{f_s} P \quad (14)$$

B. Exploring the solution-space for the unknown component

From Eqn. (12), we have that r can be written as a polynomial combination of $h_i, f_{i+1}, \ldots, f_s$. However, this combination need not be unique:

$$r = P \cdot h_i + h_{i+1} f_{i+1} + \cdots + h_s f_s + \sum_{l=1}^{n} H_l (x_l^q - x_l)$$

$$r = P^{'} \cdot h_i + h_{i+1}^{'} f_{i+1} + \cdots + h_s^{'} f_s + \sum_{l=1}^{n} H_l' (x_l^q - x_l),$$

Rearranging the terms from the above two equations:

$$\begin{aligned}(P - P^{'}) h_i =& (h_{i+1} - h_{i+1}^{'}) f_{i+1} + \cdots + (h_s - h_s^{'}) f_s \\ &+ \sum_{l=1}^{n} (H_l - H_l')(x_l^q - x_l).\end{aligned} \quad (15)$$

In other words, $(P - P^{'}) h_i \in \langle f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle$. From the definition of Quotient of Ideals (Definition II.4), we observe that:

$$P - P^{'} \in \langle f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle : \langle h_i \rangle. \quad (16)$$

There can be many polynomials $P^{'}$ which might satisfy the above ideal membership. We now show how to explore more solutions to the unknown component, i.e. given P, how to find a P' that satisfies the above membership.

Let $J_Q = \langle f_{i+1}, \ldots, f_s, x_l^q - x_l \rangle : \langle h_i \rangle$, so $P - P' \in J_Q$. This implies that P is congruent to P' modulo J_Q. In such a case, $P' = P + g$ s.t. $g \in J_Q$. This is because:

$$\begin{aligned} & P' \equiv (P + g) \pmod{J_Q} \\ \implies & P' \equiv P \pmod{J_Q} + g \pmod{J_Q} \\ \implies & P' \equiv P \pmod{J_Q} + 0 \;\; (\text{as } g \in J_Q) \\ \implies & P - P' \equiv 0 \pmod{J_Q} \\ \implies & P - P' \in J_Q \end{aligned}$$

So we can chose any polynomial g from the quotient ideal J_Q and compute $P' = P + g$. Since J_Q is computed using the Gröbner basis algorithm, we can chose $g \in GB(J_Q)$ to get $P' = P + g$.

1) Resolving the unknown component over a given set of variables: Once $P(X)$ is identified, where X may be an arbitrary set of variables, it may be desirable to express the polynomial P in terms of a given set of variables X_{im}. Here, X_{im} may correspond to, say, the known immediate inputs to the gate $\mathcal{G}_i \in C$. This can be easily performed using the concept of elimination term orders and elimination ideals.

Once a $P(X)$ is computed, we also compute $G_Q = GB(J_Q)$ with a different term order where the variables $x_i > X_{im}$ are the last in the order. In other words we use a lex order with: $X_{im} < x_i <$ *all other circuit variables*. Then reducing $P(X) \xrightarrow{G_Q}_+ r(X_{im})$ gives remainder r whose monomials will be composed of only the X_{im} variables. Therefore, $P(X_{im}) = r(X_{im})$ and the unknown component can be resolved as the polynomial $f_i : x_i + r(X_{im})$.

We can further explore different implementation functions for the unknown component. Select a polynomial $g \in G_Q$ such that $g \in G_Q \cap \mathbb{F}_q[X_{im}]$ (elimination ideal). Then $f_i : x_i + (r(X_{im}) + g)$ works as another replacement for the unknown component. This is due to the fact that $(r(X_{im}) + g) \xrightarrow{G_Q}_+ r(X_{im})$. This way, various functions for the unknown component can be explored that depend on a given set X_{im} variables.

C. Demonstration of our approach

Example III.1. Consider an implementation of a 2-bit finite field multiplier as shown in Figure 1. The system is modeled over the ring $R = \mathbb{F}_4[a_0, b_0, a_1, b_1, s_0, s_1, s_2, s_3, s_4, s_5, e_0, e_1, e_2, e_3, r_0, z_0, z_1, Z, A, B]$. The multiplier specification is given as $f : Z + A \cdot B$, and our setup is as follows:

1) Field construction: $\mathbb{F}_4 = \mathbb{F}_2[X] \pmod{\mathcal{P}}$; where $\mathcal{P} = X^2 + X + 1$ is the primitive polynomial used.
2) $Z = z_0 + \alpha z_1; A = a_0 + \alpha a_1; B = b_0 + \alpha b_1$; are the word level polynomials, and α is the root of primitive polynomial s.t. $\mathcal{P}(\alpha) = 0$.

Based on the circuit topology, RTTO$>$ with variable order: $\{Z\} > \{A > B\} > \{z_0 > z_1\} > \{r_0\} > \{e_0 > e_1\} >$

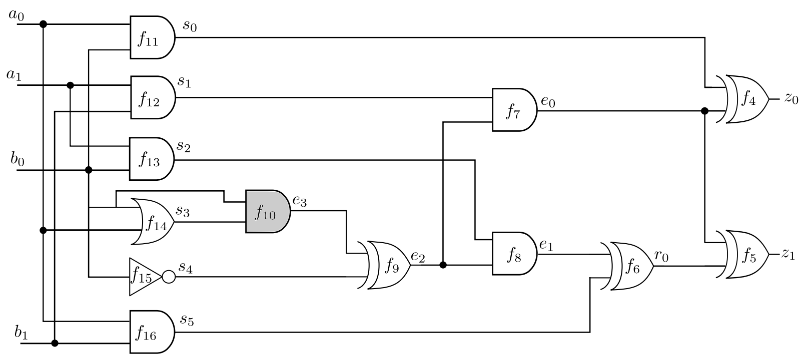

Fig. 1: A 2-bit finite field multiplier

$\{e_2\} > \{e_3\} > \{s_0 > s_1 > s_2 > s_3 > s_4 > s_5\} > \{a_0 > a_1 > b_0 > b_1\}$

Let F be the set of all polynomials implementing the circuit which is given as:

$$\begin{aligned} f_1 &: Z + z_0 + \alpha z_1; & f_9 &: e_2 + e_3 + s_4; \\ f_2 &: A + a_0 + \alpha a_1; & f_{10} &: e_3 + b_0 + s_3; \\ f_3 &: B + b_0 + \alpha b_1; & f_{11} &: s_0 + a_0 b_0; \\ f_4 &: z_0 + s_0 + e_0; & f_{12} &: s_1 + a_1 b_1; \\ f_5 &: z_1 + e_0 + r_0; & f_{13} &: s_2 + a_1 b_0; \\ f_6 &: r_0 + e_1 + s_5; & f_{14} &: s_3 + a_0 + b_0 + a_0 b_0; \\ f_7 &: e_0 + s_1 e_2; & f_{15} &: s_4 + b_0 + 1; \\ f_8 &: e_1 + s_2 e_2; & f_{16} &: s_5 + a_0 b_1; \end{aligned}$$

Let $J_0 = \langle x_l^4 - x_l \rangle$ denote the ideal of vanishing polynomials in R. Then $F = \{f_1, \dots, f_{16}\}, J = \langle F \rangle = \langle f_1, \dots, f_{16} \rangle$

For a correct implementation, the specification f is in $J + J_0$.

$f \in \langle f_1, f_2, f_3, \dots, f_{16} \rangle + J_0$

$f \xrightarrow{GB(f_1 \dots f_{16}, x_l^4 - x_l)} 0$

Now, let us assume the marked gate f_{10} is the *unknown component* in the design which is of the form $f_{10} = e_3 + P$, where P is the unknown function to be implemented by the gate. We know that under RTTO $>$, the given set of circuit polynomials F itself form a GB. Hence to compute r, we start reducing the specification polynomial f using polynomials from $\langle J + J_0 \rangle$.

We will use the following notations for reduction: '[]' to represent quotient-h_j's, '()' to represent divisor-f_j's, and '{}' to represent the partial remainder of every reduction step-fp_j's.

$f \xrightarrow{f_1} [1](Z + z_0 + \alpha z_1) + \{AB + z_0 + \alpha z_1\} \rightarrow fp_1$

$fp_1 \xrightarrow{f_2} [B](A + a_0 + \alpha a_1) + \{Ba_0 + \alpha B a_1 + z_0 + \alpha z_1\} \rightarrow fp_2$

$fp_2 \xrightarrow{f_3} [a_0 + \alpha a_1](B + b_0 + \alpha b_1) + \{z_0 + \alpha z_1 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_3$

$fp_3 \xrightarrow{f_4} [1](z_0 + e_0 + s_0) + \{\alpha z_1 + e_0 + s_0 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_4$

$fp_4 \xrightarrow{f_5} [\alpha](z_1 + r_0 + e_0) + \{\alpha z_1 + e_0 + s_0 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_5$

$fp_5 \xrightarrow{f_6} [\alpha](r_0 + e_1 + s_5) + \{(\alpha+1) e_0 + \alpha e_1 + s_0 + \alpha s_5 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_6$

$fp_6 \xrightarrow{f_7} [\alpha+1](e_0 + e_2 * s_1) + \{\alpha e_1 + (\alpha+1) e_2 s_1 + s_0 + \alpha s_5 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_7$

$fp_7 \xrightarrow{f_8} [\alpha](e_1 + e_2 * s_2) + \{(\alpha+1) e_2 s_1 + \alpha e_2 s_2 + s_0 + \alpha s_5 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_8$

$fp_8 \xrightarrow{f_9} [(\alpha+1) s_1 + \alpha s_2](e_2 + e_3 + s_4) + \{(\alpha+1) e_3 s_1 + \alpha e_3 s_2 + s_0 + (\alpha+1) s_1 s_4 + \alpha s_2 s_4 + \alpha s_5 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1) a_1 b_1 + \alpha a_1 b_0\} \rightarrow fp_9$

$$fp_9 \xrightarrow{lt(f_{10})} \underbrace{[(\alpha+1)s_1+\alpha s_2]}_{h_{10}}(e_3) + \underbrace{\{s_0+(\alpha+1)s_1s_4+\alpha s_2s_4+\alpha s_5+\alpha a_0b_1+a_0b_0+(\alpha+1)a_1b_1+\alpha a_1b_0\}}_{r}$$

Reduction order for f :

$f \xrightarrow{f_1} \xrightarrow{f_2} \xrightarrow{f_3} \xrightarrow{f_4} \xrightarrow{f_5} \xrightarrow{f_6} \xrightarrow{f_7} \xrightarrow{f_8} \xrightarrow{f_9} \xrightarrow{lt(f_{10})} r$

Given: $r, h_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, J_0$, the problem can be formulated as an ideal membership test using (12) such that:

$$r \in \langle h_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16} \rangle + \langle J_0 \rangle$$

The above ideal membership can be solved by expressing r as a linear combination of the ideal members (Lemma II.1). $r = Ph_{10} + h_{11} f_{11} + h_{12} f_{12} + h_{13} f_{13} + h_{14} f_{14} + h_{15} f_{15} + h_{16} f_{16}$

In our example, polynomial r can be expressed as

$r = [b_0] h_{10} + [1] f_{11} + [\alpha+1] f_{12} + [\alpha s_4 + \alpha b_0] f_{13} + [0] f_{14} + [(\alpha+1) s_1 + \alpha a_1 b_0] f_{15} + [\alpha] f_{16} + [0] f_{17} + [0] f_{18} + [0] f_{19} + [0] f_{20};$

Thus computed $P = b_0$ is a solution to the *unknown component* f_{10}; i.e. $f_{10} : e_3 + b_0$.

Given a solution P, we can explore the solution space for the gate f_i in terms of variables x_j such that $x_i > x_j$ in the variable order. In our example, r can be written as:

$$\begin{aligned} r &= Ph_{10} + h_{11} f_{11} + h_{12} f_{12} + h_{13} f_{13} + h_{14} f_{14} \\ &\quad + h_{15} f_{15} + h_{16} f_{16} + HJ_0 \\ r &= P' h_{10} + h'_{11} f_{11} + h'_{12} f_{12} + h'_{13} f_{13} + h'_{14} f_{14} \\ &\quad + h'_{15} f_{15} + h'_{16} f_{16} + H' J_0 \end{aligned}$$

Re-writing the above two equations:

$$(P-P')h_{10} = (h_{11}-h'_{11})f_{11} + (h_{12}-h'_{12})f_{12} + \cdots + (h_{16}-h'_{16})f_{16} + (H-H')J_0 \tag{17}$$

$$(P-P')h_{10} \in \langle f_{11}, f_{12}, \ldots, f_{16}, J_0\rangle \tag{18}$$

$$(P-P') \in \langle f_{11}, f_{12}, \ldots, f_{16}, J_0\rangle : h_{10} \tag{19}$$

$$(P-P') \in Q \tag{20}$$

The above expression for J_Q represents the quotient of ideals operation (Definition II.4). We can pick any polynomial within desired variable subset x_j from the result of J_Q and add it to the computed solution P to arrive at a new solution.

Under the current $RTTO >$ variable order, the quotient of ideals operation results in the following polynomials:
$g[1] = b_1b_0 + b_1 + b_0 + 1$
$g[2] = (\alpha+1)b_1 + (\alpha+1)*b_1*b_0 + (\alpha+1)*b_0 + (\alpha+1)$
$g[3] = a_1 + 1$
$g[4] = s_5 + a_0b_1$
$g[5] = s_4 + b_0 + 1$
$g[6] = s_3 + a_0b_0 + a_0 + b_0$
$g[7] = s_2 + b_0$
$g[8] = s_1 + b_1$
$g[9] = s_0 + a_0b_0$

Any $P + g[k]$, where $1 < k < 9$, will work as a solution for the *unknown component* f_{10}.

Now assume that we know the immediate input variables of the polynomial f_{10} as $X_{im} = (b_0, s_3)$, we can compute a solution in terms of these variables by using the elimination ideal $J_Q \cap \mathbb{F}_q[b_0, s_3]$. The quotient operation with the elimination ideal results in:
$g[1] = s_3b_0 + b_0$

Since, there is only one g from the operation, $P + g[1] = s_3b_0$ also works as a solution for the *unknown component*: $f_{10} : e_3 + s_3b_0$.

D. Circuit implementation as reference

Consider a circuit implementation C, modeled as polynomials $F = \{f_1, \ldots, f_s\} \in \mathbb{F}_q[in_j, x_1, \ldots, x_n]$, with $J_1 = \langle F\rangle$, in_j as the set of all primary inputs, and x_n as the word level output. Let us assume $f_i : 1 \leq i \leq s$ to be the unknown component which is of the special form:

$$f_i = x_k + P$$

Let us consider a different circuit C_1 as the golden specification which implements the same function as C. The reference circuit is modeled as polynomials $Q = \{q_1, \ldots q_r\} \in \mathbb{F}_q[in_j, y_1, \ldots, y_m]$, with $J_2 = \langle Q\rangle$, in_j as the set of all primary inputs, and y_m as the word level output.

To formulate the problem, we will derive a new circuit structure using the above two implementations (C, C_1). Primary input set in_j will be used as the common set of inputs for both the circuits, and the word level outputs (x_n, y_m) will be mitered using an XOR gate. A new specification polynomial f is derived using the above setup as:

$$f : t*(x_n - y_m) \tag{21}$$

where, t is the final output of miter gate.

Now, for a correct implementation, specification f should vanish on the variety of ideal generated by the circuit polynomials i.e., f will be in the ideal generated by the circuit:

$f \in J_1 + J_2 + J_0$: where J_0 is the set of all vanishing polynomials from circuits C, C_1 and miter output t.

$f \in \langle f_1, \ldots, f_s\rangle + \langle q_1, \ldots, q_r\rangle + \langle x_l^q - x_l\rangle + \langle y_u^q - y_u\rangle + \langle t^q - t\rangle$; $1 \leq l \leq n, 1 \leq u \leq m$

The problem formulation is now exactly same as (**??**) with f_i from circuit C as the unknown gate. Now, we will follow the same procedure as in the first notion to realize the function of the unknown component. Once a solution has been computed, we can verify the circuit using principles from weak *Nullstellensatz* by checking if $GB(J_1 + J_2 + J_0) = \{1\}$.

IV. Experiments

This section presents the results of our experiments on the finite-field multipliers. We compare an implementation of section III against the incremental SAT based approach presented in [3]. We have implemented the approach presented in [3] using Python with PICOSAT as the underlying SAT solver. The experiments were performed on a 4.0GHz Intel(R) Core™ i7-6700K Quad-Core CPU with 32 GB of RAM.

A. Mastrovito Multipliers

Modular multiplication is an important operation used in cryptography. A Mastrovito multiplier architecture can be used for performing this computation. Mastrovito multipliers compute $Z = A \times B \pmod{P(x)}$ where $P(x)$ is a given primitive polynomial for the datapath size k. The product $A \times B$ is computed using an array multiplier architecture, and then the result is reduced modulo $P(x)$.

B. Montgomery Multipliers

Exponentiation (repeated multiplication) is often required in cryptosystems. For such applications, Montgomery architecture [16], [17], [18] are considered more efficient than Mastrovito multipliers as they do not require explicit reduction modulo $P(x)$ after each step. Figure 2 shows the structure of a Montgomery multiplier. Each MR block computes $A \cdot B \cdot R^{-1}$, where R is selected as a power of a base (α^k) and R^{-1} is the multiplicative inverse of R in $\mathbb{F}_{2^k}$. As this operation cannot compute $A \cdot B$ directly, we need to pre-compute $A \cdot R$ and $B \cdot R$ as shown in the Figure 2. We denote the leftmost two blocks as Block A (upper) and B (lower), the middle block as Block C and the output block as Block D.

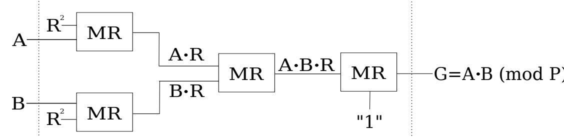

Fig. 2: Montgomery multiplication.

Table I presents the execution times when there is an unknown component in the Mastrovito multiplier and Montgomery multiplier is used as the specification. The labels NO, NM, and NI denote the unknown component placed near output, somewhere in the middle, and near input respectively. While the approach [3] finds a satisfying transformation assignment which can be mapped to a library gate, our approach section III outputs a function in terms of primary input bits which can be implemented as a single gate or sub-circuit. As shown in the table, our approach shows improvement by several orders of magnitude.

TABLE I: Resolving Unknown Component in Mastrovito circuit v/s Reference Specification Montgomery(Time is in seconds); k = Datapath Size of both multipliers, Time-Out = 12 hrs

k	**[3]**			**section III**		
	NO	*NM*	*NI*	*NO*	*NM*	*NI*
9	33.7	36.8	34.9	11.1	5.4	1.4
10	214.3	215	231.4	48.9	29.3	4.9
11	1,999.5	1,927	2,090.7	120.5	96.1	8.9
12	24,085	23,400	8,676	3880.6	2140.3	243.7
13	TO	TO	TO	4143	2735.9	321.6

V. Conclusion and future work

The paper presented an approach to resolve function implemented by a unknown component in finite field arithmetic circuits. We presented a procedure to systematically use Gröbner basis based reduction and ideal membership testing to arrive at a solution, such that the resulting logic function of the circuit conforms to the reference specification. The paper also discussed quotient of ideal concept to define multiple implementable solution set. The experiments showed that our approach is better in several orders of magnitude as compared to recent SAT based approaches, hence demonstrating the effectiveness of the underlying theory. The most desired solution which is in terms of immediate support variables of the unknown component relies on expensive Gröbner basis re-computation with a different term order. To avoid this overhead, as part of our future work, we would like to explore better heuristics to do a guided computation of solution set P, so as to arrive at a solution with specific form in desired variables. The current experiment set deals with one unknown component or sub-circuit and needs to be extended for multiple dependent bugs in a single cone. Also, identifying the bug location, which is the primary concern in the overall scope of automated debugging needs to be addressed as well.

References

[1] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Comp.*, vol. C-35, pp. 677–691, 1986.

[2] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to Combinational Equivalence Checking," in *Proc. Intl. Conf. on CAD (ICCAD)*, 2006, pp. 836–843.

[3] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design," *Proceedings of the IEEE*, 2015.

[4] S. Jo, T. Matsumoto, and M. Fujita, "SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions," in *IEEE 21st Asian Test Symposium*, 2012.

[5] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[6] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[7] F. Farahmandi and P. Mishra, "Automated Test Generation for Debugging Arithmetic Circuits," in *Design, Automation Test in Europe Conference Exhibition(DATE)*, 2016.

[8] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of Gate-level Arithmetic Circuits by Function Extraction," in *Proceedings of the 52Nd Annual Design Automation Conference*, 2015.

[9] 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.

[10] S. Ghandali, C. Yu, D. Liu, W. Brown, and M. Ciesielski, "Logic Debugging of Arithmetic Circuits," in *IEEE Computer Society Annual Symposium on VLSI*, 2015.

[11] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*. American Mathematical Society, 1994.

[12] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal," Ph.D. dissertation, University of Innsbruck, 1965.

[13] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[14] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits," in *IEEE Trans. on CAD*, vol. 32, no. 9, 2013, pp. 1409–1420.

[15] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "Singular 4-1-0 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2016.

[16] C. Koc and T. Acar, "Montgomery Multiplication in GF(2^k)," *Designs, Codes and Cryptography*, 1998.

[17] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," *IEEE Transactions On Computers*, 2002.

[18] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF(2^n) Without Pre-Computational Phase," in *Proceedings of the International Workshop on Arithmetic of Finite Fields*, 2008.