Resolving Unknown Component In Finite Field 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*—Automatic correction of unknown components is a resource intensive process. Recent developments in resolving these unknown gates rely on incremental SAT solving. Despite using state-of-the-art SAT solvers, these approaches fail to verify finite field multipliers beyond 12-bits and hence are infeasible in a practical setting. Current formal datapath verification methods using symbolic computer algebra concepts rely heavily on the textbook structure of the circuits to realize an unknown component. These approaches model circuit as a set of polynomials over integer rings, and use function extraction, simulation, and term rewriting using coefficient computation to arrive at a solution. The approach is not complete in the sense that the procedure cannot be extended to random logic circuits and finite field circuits due to ambiguities in coefficient calculation. The approach also fails to verify circuits with redundant gates. To overcome these limitations, this paper describes a formal approach using finite field theory to automatically realize the function implemented by an unknown component, and verify the same. The paper describes theory on resolving a single unknown component using ideal membership testing and Gröbner basis based reduction. We go onto pose the problem as a synthesis challenge and try to find multiple solutions to the unknown component using quotient of ideals concept. The paper also presents results on some preliminary experiments performed over finite field multipliers to compare efficiency of our approach against recent approaches.**

I. Introduction

Verifying functional correctness of gate level arithmetic circuits is still a significant challenge owing to increasing design size and functional complexity. Considerable amount of manual intervention is required to localize a bug and add 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 inherent algebraic nature of computations in such designs, symbolic algebra algorithms are considered more appropriate for their verification.

Within 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, by performing a series of reductions under a defined term order. If the 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 and is in the future scope of work. Once a particular gate has been identified as suspicious, we label the gate as an unknown component and go on to find the correct functionality 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 *Boolean Satisfiability*(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 will implement 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 num ber of test vectors, initialized to zero. Let $TestSet$ be the set of all generated test patterns, initialized to 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 $\{in_1 \dots in_k\}$. Now, check if: $(Ckt(X, in_1) = Spec(in_1)) \wedge (Ckt(X, in_2) = Spec(in_2)) \wedge \dots (Ckt(X, in_k) = Spec\ (in_k))$ 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 the 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.

Techniques from Farimah 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(HA) 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. The approach also doesn't talk about finite field arithmetic circuits as the experiments are illustrated with only integer arithmetic circuits.

B. Contribution

The paper discusses a single gate replacement error model as the target design i.e., only one gate in the design incorrectly replaced, for example an AND gate replaced with an XOR/OR gate. We are given a circuit implementation C, modeled as a set of polynomials $F = \{f_1, \dots, f_s\}$, with one of these gates f_i marked as unknown component. We then utilize concepts from symbolic computer algebra to realize the function implemented by this unknown component. The reference golden model can either be a specification polynomial f or a different circuit C_1 implementing the same function as circuit C. For a given specification polynomial f, we do polynomial reduction until the unknown component gate and arrive at the function implemented by the component by using Gröbner basis based guided ideal membership testing, and elimination ideal. For the case where the specification is given in terms of a different implementation C_1, we use Gröbner basis based reduction on a miter setup to arrive at a solution. This paper seeks to put forth the underlying theory, outline the verification challenges, and present a complete approach to resolve unknown component in finite field arithmetic circuits. We also discuss some preliminary but encouraging experimental results and draw comparison to the SAT based approach.

II. Preliminaries: Notation and Background

This section reviews some basic concepts from symbolic computer algebra that we will utilize in our theory.

Let $\mathbb{F}_q$ denote the finite field of q elements, where $q = p^k$ is some exponent (k is a positive integer) of a prime number p. We denote by $R = \mathbb{F}_2[x_1, \dots, x_n]$ the polynomial ring over 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 *e.g.* lexicographical order) is imposed on the polynomial ring such that $X_1 > X_2 > \cdots > X_t$. Subject to $>$, $lt(f) = c_1X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. Also, for a polynomial f, $tail(f) = f - lt(f)$.

Gates of a circuit can be modeled with polynomials in $\mathbb{F}_2[x_1, \dots, x_n]$, where every Boolean logic gate operator is mapped to a polynomial function over $\mathbb{F}_2$:

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

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 ([11]) shows the step-by-step procedure to perform this reduction.

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

```
1:  procedure $multi\_var\_division(f, \{f_1, \dots, f_s\}, f_j \neq 0)$
2:      $u_j \leftarrow 0;\ r \leftarrow 0,\ h \leftarrow f$
3:      while $h \neq 0$ do
4:          if $\exists j$ s.t. $lm(f_j) \mid lm(h)$ then
5:              choose $j$ least s.t. $lm(f_j) \mid lm(h)$
6:              $u_j = u_j + \frac{lt(h)}{lt(f_j)}$
7:              $h = h - \frac{lt(h)}{lt(f_j)} f_j$
8:          else
9:              $r = r + lt(h)$
10:             $h = h - lt(h)$
11:     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 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.

Polynomial Ideals: Given a set of polynomials $F = \{f_1, \dots, f_s\} \in \mathbb{F}_q[x_1, \dots, x_n]$, denote the ideal J generated by F as $J = \langle F \rangle = \langle f_1, \dots, f_s \rangle = \{\sum_{j=1}^{s} h_j \cdot f_j : h_j \in R\}$. The ideal J may have many different 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) G of ideal J is one such set of polynomials $G = GB(J) = \{g_1, \dots, g_t\}$ that is a canonical representation of the ideal.

Definition II.1. [**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 $i \in \{1, \cdots, t\}$ such that $lm(g_j)$

divides $lm(f)$.

The Gröbner basis for an ideal J can be calculated using the Buchberger's algorithm (see Alg. 1.7.1 in [11]) which takes as input a set of polynomials $\{f_1,\dots,f_s\}$ and computes the GB $G=\{g_1,g_2,\cdots,g_t\}$. A GB can be *reduced* to eliminate redundant polynomials from the basis. A reduced GB is a canonical representation of the ideal.

Lemma II.1. *Ideal Membership Testing(From Section 2.1 in [11])* Given $F=\{f_1,\dots,f_s\}$, let $G=\{g_1,g_2,\cdots,g_t\}$ be the GB for $J=\langle f_1,\dots,f_s\rangle$ with respect to a fixed term ordering. It is possible to determine if a polynomial $f\in\mathbb{F}_q[x_1,\dots,x_n]$ is in the ideal J, defined as Ideal Membership Testing by checking: $f\in J \iff f\xrightarrow{G}_+ 0$

By applying multivariate division, we can write:

$f=u_1g_1+u_2g_2+\cdots+u_tg_t; u_i\rightarrow$ quotients of division

While computing G, we can keep track of the quotients during reduction and express G in terms of F.

$$\begin{bmatrix}g_1\\g_2\\\vdots\\g_t\end{bmatrix}=M\cdot\begin{bmatrix}f_1\\f_2\\\vdots\\f_s\end{bmatrix}\quad :M \text{ is a } t\times s \text{ polynomial matrix.}$$

Ideal membership test is implemented as procedure *lift* in SINGULAR [12]. It takes a polynomial f and ideal J as input parameters and returns a set of polynomials $\{u_1,\dots,u_s\}$ such that $f=u_1f_1+\cdots+u_sf_s$.

Ideal and Variety operations:

Given two ideals $J_1=\langle f_1,\dots,f_s\rangle, J_2=\langle e_1,\dots,e_r\rangle$, their sum $J_1+J_2=\langle f_1,\dots,f_s,e_1\dots,e_r\rangle$, and their product $J_1\cdot J_2=\langle f_j\cdot e_k:1\le j\le s,1\le k\le r\rangle$. Ideals and varieties are dual concepts: $V(J_1+J_2)=V(J_1)\cap V(J_2)$, and $V(J_1\cdot J_2)=V(J_1)\cup V(J_2)$. Moreover, if $J_1\subseteq J_2$ then $V(J_1)\supseteq V(J_2)$.

For an ideal $J=\langle f_1,\dots,f_s\rangle$, the *variety* of J over $\mathbb{F}_q$ is denoted by $V_{\mathbb{F}_q}(J)$ and defined as,

$V_{\mathbb{F}_q}(J)=\{\mathbf{a}\in\mathbb{F}_q^n:\forall f\in J, f(\mathbf{a})=0\}$

Therefore, $V_{\mathbb{F}_q}(J)$ is the set of all the points in $\mathbb{F}_q^n$ that are solutions to $f_1=f_2=\cdots=f_s=0$. Varieties can be different when restricted to the given field $\mathbb{F}_q$ or considered over its algebraic closure $\overline{\mathbb{F}_q}$.

For all elements $\alpha\in\mathbb{F}_q,\alpha^q=\alpha$. Therefore, the polynomial x^q-x vanishes (the polynomial evaluation is zero) everywhere in $\mathbb{F}_q$, and is called the vanishing polynomial of the field. We denote by $J_0=\langle x_1^q-x_1,\dots,x_n^q-x_n\rangle$ the ideal of all vanishing polynomials in the ring R. Then $V_{\mathbb{F}_q}(J_0)=V_{\overline{\mathbb{F}_q}}(J_0)=\mathbb{F}_q^n$. Therefore, given any ideal J, $V_{\mathbb{F}_q}(J)=V_{\overline{\mathbb{F}_q}}(J)\cap\mathbb{F}_q^n=V_{\overline{\mathbb{F}_q}}(J)\cap V_{\overline{\mathbb{F}_q}}(J_0)=V_{\overline{\mathbb{F}_q}}(J+J_0)=V_{\mathbb{F}_q}(J+J_0)$ ([13]).

Definition II.2. *Reverse Topological Term Order [14]:* The computational complexity of Buchberger's algorithm is exponential in the number of variables n. As our work is focused on the circuits, we will describe a term order that renders the set of polynomials for the gates of the circuit, a Gröbner basis itself. This term order is called Reverse Topological Term Order (RTTO).

Let C be an 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 $x_k>x_j$ if x_k appears earlier in the reverse topological order. Impose a lex term order $>$ to represent each gate as a polynomial f_j, s.t. $f_j=x_k+tail(f_j)$. Then set of polynomials $\{f_1,\dots,f_s\}$ corresponding to the gates of the circuits is a Gröbner basis when RTTO is used for ordering.

Theorem II.1 (*The Weak Nullstellensatz over finite fields (from Theorem 3.3 in [13])*). *For a finite field $\mathbb{F}_q$ and the ring $R=\mathbb{F}_q[x_1,\dots,x_n]$, let $J=\langle f_1,\dots,f_s\rangle\subseteq R$, and let $J_0=\langle x_1^q-x_1,\dots,x_n^q-x_n\rangle$ be the ideal of vanishing polynomials. Then* $V_{\mathbb{F}_q}(J)=\emptyset\iff 1\in J+J_0\iff G=GB(J+J_0)=\{1\}$.

To find whether a set of polynomials $f_1,\dots,f_s$ have no common zeros in $\mathbb{F}_q$, we can compute the GB G of $\{f_1,\dots,f_s,x_1^q-x_1,\dots,x_n^q-x_n\}$ and see if $G=\{1\}$.

Definition II.3 ([15]). Given an ideal $J\subset\mathbb{F}_q[x_1,\dots,x_n]$, the l-th elimination ideal J_l is an ideal in R 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. The next theorem shows how we can obtain the generators of the l-th elimination ideal using Gröbner bases.

Theorem II.2 (*Elimination Theorem [15]*). 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 $0\le l\le n$ we denote by G_l the GB of l-th elimination ideal of J and compute it as:

$$G_l=G\cap\mathbb{F}_q[x_{l+1},\dots,x_n]$$

We also need to employ notion of difference of varieties in our theoretical section. The equivalent ideal operation is called the quotient of ideals.

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.

In terms of varieties, $V_{\mathbb{F}_q}(J_1:J_2)=V_{\mathbb{F}_q}(J_1)\setminus V_{\mathbb{F}_q}(J_2)$. The computation of elimination ideal is based on the intersection of ideals. For details on the computation of elimination ideal, we refer the reader to [15].

The computer algebra tools like SINGULAR [12] contain implementations for computing elimination ideals and quotient of ideals.

III. Theory and Procedure

A. Reference specification is polynomial f

Consider a specification polynomial f and its circuit implementation C, modeled as polynomials $F=\{f_1,\dots,f_s\}\in\mathbb{F}_q[x_1,\dots,x_n]$. Generator of polynomials is given as $J=\langle F\rangle$, while J_0 is the set of all vanishing polynomials. Let us consider RTTO>(II.2) for the circuit. We will assume $f_i:1\le i\le s$ to be the unknown component which is of the special form:

$$f_i=x_k+P \tag{3}$$

where x_k is the leading monomial, and P is the tail representing desired solution in variables: x_j s.t. $x_k>x_j$ in the order.

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:

$$f \in J + J_0; f \in \langle f_1, .., f_s \rangle + \langle x_l^q - x_l \rangle; 1 \le l \le n \quad (4)$$

Using Ideal membership testing, we can rewrite f in terms of its generators as:

$f = h_s f_s + h_{s-1} f_{s-1} + \cdots + h_i f_i + \cdots + h_1 f_1 + H \langle x_l^q - x_l \rangle$

where $H, h_m : 1 \le m \le s$ are arbitrary elements from $\mathbb{F}_q$. From equation 3:

$f = h_s f_s + h_{s-1} f_{s-1} + \cdots + h_i x_k + h_i P + \cdots + h_1 f_1 + H \langle x_l^q - x_l \rangle$

$f - h_s f_s - \cdots - h_i x_k = h_i P + \cdots + h_1 f_1 + H \langle x_l^q - x_l \rangle$

$f - h_s f_s - \cdots - h_i x_k \in \langle h_i, f_{i-1}, \ldots, f_1, x_l^q - x_l \rangle$

We shall call the intermediate remainder computed on the left hand side as g.

$$g \in \langle h_i, f_{i-1}, \ldots, f_1, x_l^q - x_l \rangle \quad (5)$$

Given polynomials $h_i, g, f_{i-1}, \ldots, f_1$, we compute $h_i^{'} = P$ such that:

$g = h_i^{'} h_i + h_{i-1}^{'} f_{i-1} + \cdots + h_1^{'} f_1 + H^{'} \langle x_l^q - x_l \rangle$

The computed $h_i^{'} = P$ is a solution to the function implemented by the unknown gate. This linear combination computation is done using $lift$(II.1) implementation in SINGULAR[12].

We will also have cases, when h_i ends up being a constant, in which case $lift$ returns g itself as a solution $h_i^{'}$. To arrive at a implementable solution, 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$\{f_{i-1}, \ldots, f_1\}$.

$$h_i^{'} * h_i^{-1} \xrightarrow{f_{i-1}} \xrightarrow{f_{i-2}} \ldots \xrightarrow{f_1} P \quad (6)$$

Despite being a correct solution, the above approach doesn't guarantee the solution to be in the immediate support variables of f_i due to RTTO>. To determine a solution in immediate support variable set x_j of f_i, we use an elimination term order(II.2) for the variables x_k followed by x_j. We can then compute a GB using this elimination term order with the intermediate solution P added as tail of f_i. This GB will have one and only one polynomial which is of the form $x_k + \mathcal{F}(x_j)$, where $\mathcal{F}$ is the function implemented by the gate, and is the most desired solution.

Since we found two solutions, it is given that P is not unique. We can explore more such solutions which might satisfy the unknown component functionality. Given P as one of the solutions, under RTTO> we have:

$g = P * h_i + h_{i-1}^{'} f_{i-1} + \cdots + h_1^{'} f_1 + H^{'} \langle x_l^q - x_l \rangle$;

Since g can be computed as any linear combination of polynomials, we can rewrite the equation as:

$P * h_i + h_{i-1}^{'} f_{i-1} + \cdots + h_1^{'} f_1 + H^{'} \langle x_l^q - x_l \rangle = P^{'} * h_i + h_{i-1}^{''} f_{i-1} + \cdots + h_1^{''} f_1 + H^{''} \langle x_l^q - x_l \rangle$;

Rearranging the terms:

$(P - P^{'}) h_i = (h_{i-1}^{'} - h_{i-1}^{''}) f_{i-1} + \cdots + (h_1^{'} - h_1^{''}) f_1 + (H^{'} - H^{''}) x_l^q - x_l$;

$(P - P^{'}) h_i \in \langle f_{i-1}, \ldots, f_1, x_l^q - x_l \rangle$;

By definition of Quotient of Ideals(II.4):

$$P - P^{'} \in \langle f_{i-1}, \ldots, f_1, x_l^q - x_l \rangle : h_i; \quad (7)$$

There can be many $P^{'}$ which might satisfy the above membership test. We can pick any polynomial from the quotient operation, add the previous solution P and compute a new $P^{'}$. All such $P^{'}$ computed are valid solutions and will satisfy the membership test with specification polynomial f. The following example for a 2-bit Mastrovito multiplier circuit demonstrates the above procedure in detail.

Example III.1. Consider the 2-bit Mastrovito multiplier given in fig. 1 with variables from ring $\mathbb{R} = \mathbb{F}_2[a_0, b_0, a_1, b_1, s_0, s_1, s_2, s_3, r_0, z_0, z_1, Z, A, B]$. Let us assume f_2 to be the unknown gate in the design which is of the form $f_2 = s_3 + P$.

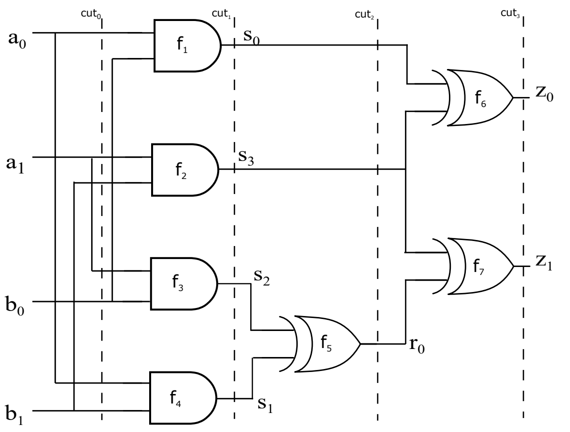

Fig. 1: 2-bit Mastrovito multiplier

The 2x2 Mastrovito multiplier with specification $f : Z + A \cdot B$, is constructed 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) $Z = z_0 + z_1 * \alpha; A = a_0 + a_1 * \alpha; B = b_0 + b_1 * \alpha$; 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: $\{Z\} > \{A > B\} > \{z_0 > z_1\} > \{r_0 > s_0 > s_3\} > \{s_1 > s_2\} > \{a_0 > a_1 > b_0 > b_1\}$

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

$f_1 : s_0 + a_0 * b_0$;	$f_5 : r_0 + s_1 + s_2$;	$f_8 : A + a_0 + a_1 * \alpha$;
$f_2 : s_3 + P$;	$f_6 : z_0 + s_0 + s_3$;	$f_9 : B + b_0 + b_1 * \alpha$;
$f_3 : s_2 + a_1 * b_0$;	$f_7 : z_1 + r_0 + s_3$;	$f_{10} : Z + z_0 + z_1 * \alpha$;
$f_4 : s_1 + a_0 * b_1$;		

We shall add the ideal of vanishing polynomials J_0 for primary inputs, outputs, and intermediate variables.

$f_{11} : a_0^2 + a_0$;	$f_{15} : s_0^2 + s_0$;	$f_{19} : r_0^2 + r_0$;	$f_{23} : A^4 + A$;
$f_{12} : a_1^2 + a_1$;	$f_{16} : s_1^2 + s_1$;	$f_{20} : z_0^2 + z_0$;	$f_{24} : B^4 + B$;
$f_{13} : b_0^2 + b_0$;	$f_{17} : s_2^2 + s_2$;	$f_{21} : z_1^2 + z_1$;	
$f_{14} : b_1^2 + b_1$;	$f_{18} : s_3^2 + s_3$;	$f_{22} : Z^4 + Z$;	

$F = \{f_1, \ldots, f_{10}\}; J = \langle F \rangle = \langle f_1, \ldots, f_{10} \rangle; J_0 = \langle f_{11}, \ldots, f_{24} \rangle$

For a correct implementation, specification f vanishes on circuit implementation: $f \in \langle f_1, f_2, f_3, \ldots, f_{10} \rangle + \langle f_{11}, f_{12}, \ldots, f_{24} \rangle$, where tail of f_2 is unknown.

We know that under RTTO>, the given set of circuit polynomials in itself form a GB. Hence to compute g, we start reducing the specification polynomial f using polynomials from the set $\langle J + J_0 \rangle$. We will use the following notations for reduction: '[]' to represent quotient-h_i's, '()' to represent divisor-f_i's, and '$\{\}$' to represent the partial remainder of every reduction step-fp_i's.

$f \xrightarrow{f_{10}} [1](Z + z_0 + z_1 * \alpha) + \{A * B + z_0 + \alpha * z_1\} \rightarrow fp_1$

$fp_1 \xrightarrow{f_8} [B](A + a_0 + \alpha * a_1) + \{B * a_0 + \alpha * B * a_1 + z_0 + \alpha * z_1\} \rightarrow fp_2$

$fp_2 \xrightarrow{f_9} [a_0 + \alpha * a_1](B + b_0 + \alpha * b_1) + \{z_0 + \alpha * z_1 + a_0 * b_0 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1\} \rightarrow fp_3$

$fp_3 \xrightarrow{f_6} [z_0 + s_0 + s_3](1) + \{\alpha * z_1 + s_0 + s_3 + a_0 * b_0 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1\} \rightarrow fp_4$

$fp_4 \xrightarrow{f_7} [z_1 + r_0 + s_3](\alpha) + \{\alpha * r_0 + s_0 + (\alpha + 1) * s_3 + a_0 * b_0 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1\} \rightarrow fp_5$

$fp_5 \xrightarrow{f_5} [r_0 + s_1 + s_2](\alpha) + \{s_0 + (\alpha + 1) * s_3 + \alpha * s_1 + \alpha * s_2 + a_0 * b_0 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1\} \rightarrow fp_6$

$fp_6 \xrightarrow{f_1} [s_0 + a_0 * b_0](1) + \{(\alpha + 1) * s_3 + \alpha * s_1 + \alpha * s_2 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1\} \rightarrow fp_7$

$fp_7 \xrightarrow{lt(f_2)} [s_3](\underbrace{\alpha+1}_{h_2}) + \{\underbrace{\alpha s_1 + \alpha s_2 + \alpha a_0 b_1 + \alpha a_1 b_0 + (\alpha+1) a_1 b_1}_{g}\}$

Reduction order for $f : f \xrightarrow{f_{10}} \xrightarrow{f_8} \xrightarrow{f_9} \xrightarrow{f_6} \xrightarrow{f_7} \xrightarrow{f_5} \xrightarrow{f_1} \xrightarrow{lt(f_2)} g$

Since, we know g, h_2, f_3, f_4, J_0, we can formulate it as ideal membership testing problem using equation(4):

$$g \in \langle h_2, f_3, f_4 \rangle + \langle J_0 \rangle$$

This ideal membership test can be done using $lift$ procedure in SINGULAR. The procedure takes polynomial f, and ideal J in row matrix form($[J + J_0]$) as inputs, and returns a column matrix $[U]$ as output such that: $f = [U] \cdot [J + J_0]$

$$g = \begin{bmatrix} h_2' & h_3' & \dots & H \end{bmatrix} \cdot \begin{bmatrix} h_2 \\ f_3 \\ \vdots \\ x_n^2 + x_n \end{bmatrix}$$

The matrix output can be written in a linear combination as follows: $\alpha * s_1 + \alpha * s_2 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1 = [\alpha * s_1 + \alpha * s_2 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1](\alpha + 1) + [0](s_2 + a_1 * b_0) + [0](s_1 + a_0 * b_1) + \dots + [0](x_n^2 + x_n)$;

The $lift$ procedure uses GB to compute the linear combination. Since the generator set has a constant polynomial($\alpha + 1$), the procedure considers it as constant one during membership computation. Hence the output in this case will be a 1x1 matrix with g itself projected as the solution. To normalize the ignored constant, we need to divide(multiply the inverse) the solution g by the constant($\alpha + 1$) and reduce it with polynomials $\{f_3, f_4\}$ in order to arrive at a implementable solution.

$(\alpha + 1)^{-1} = \alpha$

$h_2'' = \alpha * h_2' = \alpha * (\alpha * s_1 + \alpha * s_2 + \alpha * a_0 * b_1 + \alpha * a_1 * b_0 + (\alpha + 1) * a_1 * b_1)$

$h_2'' = (\alpha + 1) * s_1 + (\alpha + 1) * s_2 + (\alpha + 1) * a_0 * b_1 + (\alpha + 1) * a_1 * b_0 + a_1 * b_1$

$h_2'' \xrightarrow{f_3} \xrightarrow{f_4} \underbrace{a_1 * b_1}_{\text{P}}$

This computed P is a valid solution and can be used as tail of f_2. If the resulting P were not a solution in immediate support variables of $s_3(a_1, b_1)$, we could have arrived at this desired solution by devising a new term order. The new term order will have variables (s_3, a_1, b_1) moved to the end on the variable order. We could then compute a GB using the modified term order with the intermediate solution P added as tail of f_2. This GB will have one and only one polynomial which is of the form $s_3 + \mathcal{F}(a_1, b_1)$, where $\mathcal{F}$ is the function implemented by the gate for these variables.

Modified term order $\{Z\} > \{A > B\} > \{z_0 > z_1\} > \{r_0 > s_0\} > \{s_1 > s_2\} > \{a_0 > b_0 > s_3 > a_1 > b_1\}$.

To illustrate the computation of multiple solutions, let's take the current solution $a_1 * b_1$ as our P. With RTTO> as our term order, from equation(7):

$a_1 * b_1 - P' \in \{f_3, f_4, x_l^q - x_l\} : h_2;$

The result from quotient of ideal operation is given as:

$$\begin{aligned} f_3 : s_2 + a_1 * b_0; f_{13} : b_0^2 + b_0; f_{23} : A^4 + A; \\ f_4 : s_1 + a_0 * b_1; f_{14} : b_1^2 + b_1; f_{24} : B^4 + B; \\ f_{11} : a_0^2 + a_0; f_{16} : s_1^2 + s_1; \\ f_{12} : a_1^2 + a_1; f_{17} : s_2^2 + s_2; \end{aligned}$$

Any polynomial from the above list when added in tail of f_2 along with P satisfies the solution set for the given unknown component.

B. Circuit implementation as reference

Consider a circuit implementation C, modeled as polynomials $F = \{f_1, \dots, f_s\} \in \mathbb{F}_q[in_j, x_1, \dots, x_n]$, 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 $Q = \{q_1, \dots q_r\} \in \mathbb{F}_q[in_j, y_1, \dots, y_m]$, 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:

$$f : t * (x_n - y_m) \qquad (8)$$

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.

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

The problem formulation is now exactly same as equation(4) 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 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$^{\text{TM}}$ 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. Fig. 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 Fig. 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.

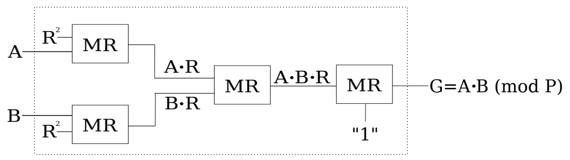

Fig. 2: Montgomery multiplication.

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 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, #Gates = No. of gates, Time-Out = 12 hrs, K = 10^3, M = 10^6

k	**[3]**			**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] 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.

[13] S. Gao, "Counting Zeros over Finite Fields with Gröbner Bases," Master's thesis, Carnegie Mellon University, 2009.

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

[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.