POST-VERIFICATION DEBUGGING AND RECTIFICATION OF FINITE FIELD ARITHMETIC CIRCUITS USING COMPUTER ALGEBRA TECHNIQUES

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

Abstract: *Formal verification of arithmetic circuits checks whether or not a gate-level circuit correctly implements a given specification model. In cases where this equivalence check fails – the presence of a bug is detected – it is required to: i) debug the circuit, ii) identify a set of nets (signals) where the circuit might be rectified, and iii) compute the corresponding rectification functions at those locations. This paper addresses the problem of post-verification debugging and correction (rectification) of finite field arithmetic circuits. The specification model and the circuit implementation may differ at any number of inputs. We present techniques that determine whether the circuit can be rectified at one particular net (gate output) – i.e. we address single-fix rectification.*

Starting from an equivalence checking setup modeled as a polynomial ideal membership test, we analyze the ideal membership residue to identify potential single-fix rectification locations. Subsequently, we use Nullstellensatz principles to ascertain if indeed a single-fix rectification can be applied at any of these locations. If a single-fix rectification exists, we derive a rectification function by modeling it as the synthesis of an unknown component problem. Our approach is based upon the Gröbner basis algorithm, which we use both as a decision procedure (for rectification test) as well as a quantification procedure (for computing a rectification function). Experiments are performed over various finite field arithmetic circuits that demonstrate the efficacy of our approach, whereas SAT-based approaches are infeasible.

I. INTRODUCTION

Past few years have seen extensive investigations into formal verification of arithmetic circuits. Circuits that implement polynomial computations over large bit-vector operands are hard to verify using methods such as SAT/SMT-solvers, decision diagrams, etc. Recent techniques have investigated the use of polynomial algebra and algebraic geometry techniques for their verification. These include verification of integer arithmetic circuits [1] [2] [3] and also finite field circuits [4] [5]. While these are successful in proving correctness or detecting the presence of bugs, the problem of debugging and correction of arithmetic circuits has only just begun to be addressed [6], [7].

In this paper, we address the problem of rectification of buggy finite field arithmetic circuits. A specification model (*Spec*) is given either as a polynomial description f over a finite field, or as a golden model of a finite field arithmetic circuit. The finite field considered is the field of 2^k elements (denoted by $\mathbb{F}_{2^k}$), where k corresponds to the operand-width (bit-vector word length). An implementation circuit C is also given. Equivalence checking is performed between the *Spec* and the circuit C, and the presence of a bug is detected. No restrictions on the number, type, or locations of the bugs are assumed.

We perform error-diagnosis and a subset $\mathcal{N}$ of the nets of the circuit is identified as *potential rectification locations.* Given the *Spec*, the buggy implementation circuit C, and the set $\mathcal{N}$ of potential rectifiable locations, our objective is to determine whether or not the buggy circuit can be rectified at *one particular net* $x_i \in \mathcal{N}$. This is called **single-fix rectification** in literature [8]. If a single-fix rectification does exist at net x_i in the buggy circuit, then our subsequent objective is to derive a polynomial function $U(X_{PI})$ in terms of the set of primary input variables X_{PI}. This polynomial can be translated (synthesized) into a logic subcircuit such that $x_i = U(X_{PI})$ acts as the rectification function for the buggy circuit C so that C matches the specification.

Our techniques and algorithms are based on symbolic computer algebra and algebraic geometry – particularly on the concepts of the Strong Nullstellensatz and Gröbner bases [9]. We show how to apply our techniques to rectify finite field arithmetic circuits, where conventional SAT-solver based rectification approaches are infeasible.

The paper is organized as follows: The following section reviews related previous work. Section III covers preliminary concepts. The formulation of the verification test is described in Section IV. Section V describes conditions for rectification at a particular net. Section VI describes how rectification function can be synthesized once single-fix rectification is deemed possible. Section VII describes our experimental results and Section VIII concludes the paper.

II. PREVIOUS WORK

Automated diagnosis and rectification of digital circuits has been addressed in [10], [11]. The paper [12] presents algorithms for synthesizing Engineering Change Order (ECO) patches - an analogous problem to rectification. The use of interpolation for ECO has been presented in [8], [13]. The single-fix rectification function approach in [13] has been extended in [8] to generate multiple partial-fix functions. As these approaches are SAT based, they are not efficient for arithmetic circuits. In contrast to these works, our work presents a word-level formulation for single-fix rectification. Computer algebra has been utilized for circuit debugging and rectification in [6], [7]. These approaches rely heavily on the structure of the arithmetic circuit for coefficient calculation.

Moreover, if the arithmetic circuits contain redundancies, then we have shown that their approach is incomplete in that it cannot resolve the rectification question. We have uploaded an appendix at [14] for detailed discussion on these issues.

Once rectification is deemed feasible, the problem of finding the rectification function has been considered as a partial synthesis problem. The most recent and relevant approach [15], [16] resolves the unknown component problem using an incremental SAT formulation.

The approach used in [17] inserts logic corrector MUXs on the unknown sub-circuits and relies on SAT solvers to realize the functionality. The authors in [18] present a QBF formulation for answering whether a partial implementation can be extended to a complete design that models a given specification.

Despite using state-of-the-art SAT solvers, all the above approaches fail to verify large and complex finite field arithmetic circuits. We demonstrate the efficiency of our implementation by comparing the results against the most recent SAT based approach [15] showing improvement by several orders of magnitude.

III. Preliminaries: Notation and Background

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 $\mathbb{F}_2 = \{0, 1\}$ is the field of two elements, and $P(X)$ is a given irreducible polynomial of degree k with α as one of its root, i.e. $P(\alpha) = 0$.

Let $R = \mathbb{F}_q[x_1, \ldots, x_n]$ be the polynomial ring in variables $x_1, \ldots, x_n$ 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 $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ 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.

1) 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* $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r. This reduction is denoted as $f \xrightarrow{F}_+ r$, where the remainder r is said to be *reduced* – i.e. no term in r is divisible by the leading term of any polynomial f_j in F. Algorithm 1 (Alg. 1.5.1 from [9]) depicts a procedure for this reduction. Along with the remainder r, the algorithm also returns the set of quotients $\{u_1, \ldots, u_s\}$ of division of f by $\{f_1, \ldots, f_s\}$, respectively, such that $f = u_1 \cdot f_1 + \cdots + u_s \cdot f_s + r$.

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

```
1:  procedure multi_var_division(f, {f_1, ..., f_s}, f_j ≠ 0)
2:      u_j ← 0;  r ← 0,  h ← f
3:      while h ≠ 0 do
4:          if ∃j s.t. lm(f_j) | lm(h) then
5:              choose j least s.t. lm(f_j) | lm(h)
6:              u_j = u_j + lt(h)/lt(f_j)
7:              h = h − lt(h)/lt(f_j) f_j
8:          else
9:              r = r + lt(h)
10:             h = h − lt(h)
11:     return ({u_1, ..., u_s}, r)
```

2) Polynomial Ideals, Varieties and Gröbner Bases:

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

$$J = \langle f_1, \ldots, f_s \rangle = \{h_1 \cdot f_1 + \cdots + h_s \cdot f_s : h_1, \ldots, h_s \in R\}. \quad (1)$$

The polynomials $f_1, \ldots, f_s$ form the basis of ideal J.

Let $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$ be a point in the affine space, and f a polynomial in R. If $f(\boldsymbol{a}) = 0$, we say that f *vanishes* on $\boldsymbol{a}$. We have to analyze the *set of all common zeros* of the polynomials of F that lie within the field $\mathbb{F}_q$. This zero set is called the *variety*, which depends on the ideal generated by the polynomials. We denote it by $V(J)$, where: $V(J) = V_{\mathbb{F}_q}(J) = V_{\mathbb{F}_q}(f_1, \ldots, f_s) = \{\boldsymbol{a} \in \mathbb{F}_q^n : \forall f \in J, f(\boldsymbol{a}) = 0\}$.

An ideal may have many different sets of generators, i.e. it is possible to have $J = \langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle = \cdots = \langle h_1, \ldots, h_r \rangle$, such that $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t) = \cdots = V(h_1 \ldots, h_r)$. A Gröbner basis (GB) of an ideal is one such generating set $G = \{g_1, \ldots, g_t\}$, which possesses many important properties that allow to solve many polynomial decision problems.

Definition III.2. [**Gröbner Basis**] [9]: 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 [19]. It takes as input a set of polynomials $\{f_1, \ldots, f_s\}$ and computes its GB $G = \{g_1, g_2, \cdots, g_t\}$. The reader may refer to Algorithm 1.7.1 in [9] for a detailed explanation.

Buchberger's algorithm can be easily extended to output not just the Gröbner basis $G = \{g_1, \ldots, g_t\}$ but also a $t \times s$ matrix

M with polynomial entries such that:

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_t \end{bmatrix} = M \cdot \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} \tag{2}$$

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

Proposition III.1. (*Ideal Membership Testing)* Let $G = GB(J) = \{g_1, \ldots, g_t\}$ be the Gröbner basis of ideal J, and f be any polynomial. Then $f \in J \iff f \xrightarrow{G}_+ 0$.

Therefore, if $f \in J$, f can be written as a linear combination (with polynomial coefficients) of the elements of the Gröbner basis:

$$f = u_1 g_1 + u_2 g_2 + \cdots + u_t g_t, \tag{3}$$

where u_i's correspond to the quotients of division $f \xrightarrow{g_1,\ldots,g_t}_+ 0$. Subsequently, Eqns. (3) and (2) can be combined to give f as combination of the original polynomials $f_1, \ldots, f_s$:

$$f = v_1 f_1 + \cdots + v_s f_s. \tag{4}$$

Given two ideals $J_1 = \langle f_1, \ldots, f_s \rangle, J_2 = \langle h_1, \ldots, h_r \rangle$, the sum $J_1 + J_2 = \langle f_1, \ldots, f_s, h_1 \ldots, h_r \rangle$, and their product $J_1 \cdot J_2 = \langle f_i \cdot h_j : 1 \leq i \leq s, 1 \leq j \leq r \rangle$. 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 $V(J_1) \supseteq V(J_2)$.

3) The Strong Nullstellensatz in Finite Fields: For any element $\alpha \in \mathbb{F}_q$, we have that $\alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ vanishes everywhere in $\mathbb{F}_q$, and we call it a *vanishing polynomial*. Let $J_0 = \langle x_1^q - x_1, \ldots, x_n^q - x_n \rangle$ be the ideal of all vanishing polynomials in the ring R. Then, $V_{\mathbb{F}_q}(J_0) = V_{\overline{\mathbb{F}_q}}(J_0) = \mathbb{F}_q^n$. Moreover, given any ideal J, $V_{\mathbb{F}_q}(J) = V_{\overline{\mathbb{F}_q}}(J) \cap \mathbb{F}_q^n = V_{\overline{\mathbb{F}_q}}(J) \cap V_{\overline{\mathbb{F}_q}}(J_0) = V_{\overline{\mathbb{F}_q}}(J + J_0) = V_{\mathbb{F}_q}(J + J_0)$.

Definition III.3. Given an ideal $J \subset R$ and $V(J) \subseteq \mathbb{F}_q^n$, the *ideal of polynomials that vanish on* $V(J)$ is $I(V(J)) = \{f \in R : \forall \boldsymbol{a} \in V(J), f(\boldsymbol{a}) = 0\}$.

If f vanishes on $V(J)$, then $f \in I(V(J))$. The Strong Nullstellensatz, which has a special form over finite fields, characterizes the ideal $I(V(J))$.

Theorem III.1 (*The Strong Nullstellensatz over finite fields (Theorem 3.2 in [20]))***.** For any ideal $J \subset \mathbb{F}_q[x_1, \ldots, x_n]$, $I(V_{\mathbb{F}_q}(J)) = J + J_0$.

IV. The Verification Test

Given a specification polynomial f, and a circuit C, the verification test is formulated as presented in [4]. The circuit is modeled by a set of multivariate polynomials $F = \{f_1, \ldots, f_s\}$ in the ring $R = \mathbb{F}_{2^k}[x_1, \ldots, x_n]$ for the given data-path (operand) size k, where $x_1, \ldots, x_n$ denote the nets (signals) in the circuit. As the circuit comprises Boolean logic gates, they are modeled as polynomials in $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$:

$$\begin{aligned} z &= \neg a \;\rightarrow\; z + a + 1 \pmod 2 \\ z &= a \wedge b \;\rightarrow\; z + a \cdot b \pmod 2 \\ z &= a \vee b \;\rightarrow\; z + a + b + a \cdot b \pmod 2 \\ z &= a \oplus b \;\rightarrow\; z + a + b \pmod 2 \end{aligned} \tag{5}$$

The set of polynomials F generates an ideal, which we denote by $J = \langle F \rangle$. When C correctly implements f, then f agrees with every evaluation of all the nets in C. In other words, f vanishes on $V(J)$, or equivalently $f \in I(V(J))$. The Strong Nullstellensatz in finite fields (Thm. III.1) tells us that $I(V(J)) = J + J_0$, where $J_0 = \langle x_i^q - x_i : i = 1, \ldots, n \rangle$. Thus, the verification test can be formulated as ideal membership testing of f in $J + J_0$ using Gröbner bases: to check if $f \xrightarrow{GB(J+J_0)}_+ 0$?

The Gröbner basis computation $GB(J + J_0)$ in $R = \mathbb{F}_q[x_1, \ldots, x_n]$ exhibits high complexity, as it is shown to be bounded by $q^{O(n)}$ [20]. In [4], the authors showed that the expensive Gröbner basis computation can be avoided altogether for this verification test. It was shown that for any arbitrary combinational circuit, a specialized term order can be derived by analyzing the topology of the given circuit. Imposition of this term order on R renders the set of polynomials $F = \{f_1, \ldots, f_s\}$ itself a Gröbner basis. Based on Buchberger's product criteria, their approach exploits the fact that *when the leading terms of all polynomials in F are relatively prime, then F already constitutes a Gröbner basis.*

Definition IV.1. Let C be an arbitrary combinational circuit described by a set of polynomials $F = \{f_1, \ldots, f_s\}$ with variables $\{x_1, \ldots, x_n\}$. Starting from the primary outputs, perform a reverse topological traversal of C 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 $F = \{f_1, \ldots, f_s\}$ forms a Gröbner basis, 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)**.

Our formulations also contain k-bit word-level variables corresponding to the input and output word-level operands. These variables can also be accommodated in RTTO $>$ by imposing a lex term order with the variable order *"Output word $>$ input words $>$ bit-level variables ordered reverse topologically"*. In [4], the authors analyzed the effect of such a term order further on ideal generators that include the vanishing polynomials. Let $X_{PI} \subset \{x_1, \ldots, x_n\}$ be the primary input variables of the circuit. Let $F_0^{PI} = \{x_i^2 - x_i : x_i \in X_{PI}\}$ denote the set of bit-level vanishing polynomials in primary inputs. We utilize the following result from [4].

Proposition IV.1. (Corollary 6.1 in [4]) Using RTTO $>$ to represent the polynomials in R, the set $F \cup F_0^{PI}$ constitutes a Gröbner basis of $J + J_0$.

The benefit of using RTTO $>$ is that the verification test can be performed solely by way of polynomial division $f \xrightarrow{F,F_0^{PI}}_+ r$, and by checking whether or not $r = 0$? If $r = 0$, then C implements f. Otherwise when $r \neq 0$, there exists a bug in the design. Moreover, RTTO $>$ ensures that when $r \neq 0$, r comprises only primary input variables X_{PI}. Any assignment to X_{PI} that makes $r \neq 0$ generates a counter-example that can be used for debugging.

We use the verification setup under RTTO $>$ (i.e. Def. IV.1 and Prop. IV.1) to rectify the circuit. Our approach begins when the verification test detects the presence of a bug in the design, i.e. $f \xrightarrow{F,F_0^{PI}}_+ r$ with $r \neq 0$. In the sequel, we will use the circuit shown in Fig. 1 as a running example to demonstrate our approach to debugging and rectification. The circuit is a modified version of a Mastrovito multiplier [21], where extra redundant logic was first added in the circuit, and then a bug was introduced in the redundant logic.

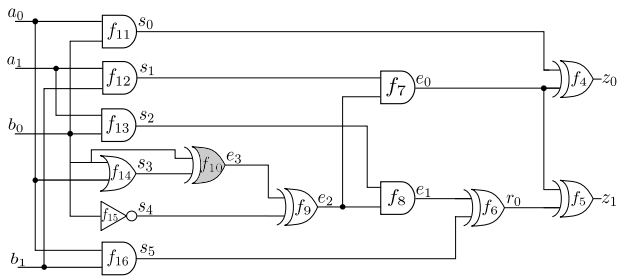

Fig. 1: Design verification of a 2-bit finite-field multiplier. The circuit is buggy, with the bug introduced at net e_3. A correct implementation includes an AND gate at e_3, which is replaced by an XOR gate to introduce a bug.

Example IV.1. We perform verification of the design of a 2-bit finite field multiplier in $\mathbb{F}_4$, where the output Z is to be computed as $A \cdot B$, where $Z = \{z_1, z_0\}, A = \{a_1, a_0\}, B = \{b_1, b_0\}$ are the given 2-bit operands. Assume further that $P(X) = X^2 + X + 1$ is the irreducible polynomial used to construct $\mathbb{F}_4 = \mathbb{F}_2[X] \pmod{P(X)}$, with $P(\alpha) = 0$.

The implemented circuit C is given as shown in Fig. 1. Denote polynomial $f : Z + A \cdot B$ as the design specification. For the verification test, we perform a reverse topological traversal of the circuit to derive RTTO $>$, i.e. a *lex* term order with variable order: $\{Z\} > \{A > B\} > \{z_0 > z_1\} > \{r_0\} > \{e_0 > e_1\} > \{e_2\} > \{e_3\} > \{s_0 > s_1 > s_2 > s_3 > s_4 > s_5\} > \{a_0 > a_1 > b_0 > b_1\}$.
The polynomials describing the circuit are given as:

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

Then $F = \{f_1, \ldots, f_{16}\}$, $F_0^{PI} = \{a_0^2 - a_0, a_1^2 - a_1, b_0^2 - b_0, b_1^2 - b_1\}$, and $F \cup F_0^{PI}$ constitutes a Gröbner basis of $J + J_0$. Computing $f \xrightarrow{F,F_0^{PI}}_+ r$ gives $r = (\alpha+1)a_0a_1b_1b_0 + (\alpha+1)a_0a_1b_1 + (\alpha+1)a_1b_1b_0 + (\alpha)a_1b_0$. Since $r \neq 0$, the presence of a bug in the design is detected. Our objective now is to identify a net where rectification can be performed, and then to subsequently identify a rectification function.

V. Identification of the Rectification Target

After the presence of a bug is detected, we address the problem of single-fix rectification of C. In this section, we present an approach that ascertains whether or not a single-fix rectification can be applied at a given (target) net x_i in C. In principle, our approach can be applied at every net x_i in C to see if C at all admits single-fix rectification. However, it is possible to prune the search for these target nets x_i by analyzing the non-zero remainder obtained by the Gröbner basis reduction $f \xrightarrow{F,F_0^{PI}}_+ r$. We show how to construct a subset $\mathcal{N} \subseteq \{x_1, \ldots, x_n\}$ as possible rectification targets. This rectification target pruning approach is inspired from [7]. Then we present our rectification theorem and the search for a rectification function.

1) Potential rectification target nets: The circuit C has k-bit operands, and the output is expressed as $Z = \sum_{i=0}^{k-1} z_i \alpha^i$. Then the non-zero remainder r can be partitioned based on the coefficients of the monomials in r and re-expressed as:

$$r = \alpha^0(r_0) + \alpha^1(r_1) + \cdots + \alpha^{k-1}(r_{k-1}) \tag{6}$$

Non-zero terms r_i (with coefficient α^i) imply that the effect of the bug is observable at the bit-level output z_i. We consider the transitive fanin cones of logic of the output bits z_i. When a bug affects multiple outputs, a single-fix rectification might exist only at the nets that lie in the intersection of the respective fanin-cones of the affected outputs. In our experiments, we include these nets in $\mathcal{N}$ to check if any one of them admits a single-fix rectification.

Example V.1. As shown in Ex. IV.1, $f \xrightarrow{F,F_0^{PI}}_+ r = (\alpha+1)a_0a_1b_1b_0 + (\alpha+1)a_0a_1b_1 + (\alpha+1)a_1b_1b_0 + (\alpha)a_1b_0$. We re-write the remainder $r = \alpha^0 r_0 + \alpha^1 r_1 = \alpha \cdot (a_0a_1b_1b_0 + a_0a_1b_1 + a_1b_1b_0 + a_1b_0) + 1 \cdot (a_0a_1b_1b_0 + a_0a_1b_1 + a_1b_1b_0)$. Since both r_0 and r_1 are non-zero, the bug affects both primary outputs z_0, z_1. By identifying the nets that lie in the intersection of the fanin cones of z_0, z_1, we construct $\mathcal{N} = \{s_4, s_3, s_2, s_1, e_3, e_2, e_0\}$ as potential rectifiable locations.

2) Confirming a rectification target: After post-verification debugging is performed to identify a set of nets $\mathcal{N} \subseteq \{x_1, \ldots, x_n\}$ that are potential rectification target nets, we now present an approach that *confirms* whether or not the circuit can indeed be single-fix-rectified at net x_i. *Single-fix-rectification at target net x_i means that there exists a polynomial function $U(X_{PI})$ which, when implemented at net x_i, ensures that the circuit C would correctly implement the specification f.* Note that $x_i = U(X_{PI})$ is a polynomial

function of the type $\mathbb{F}_2^{|X_{PI}|} \rightarrow \mathbb{F}_2$ as it implements a subcircuit at net x_i.

In the set of polynomials F, we replace $f_i = x_i + U(X_{PI})$ as the polynomial for the rectification function at x_i, where $U(X_{PI})$ is a hitherto unknown/unresolved polynomial function component. In other words, F is updated to $F = \{f_1, \ldots, f_{i-1}, f_i = x_i + U(X_{PI}), f_{i+1}, \ldots, f_s\}$. We state and prove the *rectification theorem that checks for the existence of $U(X_{PI})$ as a single-fix rectification function at x_i.*

Theorem V.1 (Rectification Theorem)**.** Given the specification polynomial f, and the implementation circuit C, derive RTTO $>$ to represent the polynomials. Using RTTO $>$, construct two ideals:

- $J_L = \langle F_L \rangle$, where $F_L = \{f_1, \ldots, f_{i-1}, f_i = x_i + 1, f_{i+1}, \ldots, f_s\}$;
- $J_H = \langle F_H \rangle$, where $F_H = \{f_1, \ldots, f_{i-1}, f_i = x_i, f_{i+1}, \ldots, f_s\}$;

where the polynomials $f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_s$ are the same as in the generators of ideal J (representing the circuit), and f_i is replaced with $f_i = x_i + 1$ in J_L and $f_i = x_i$ in J_H, respectively. Perform the reductions:

- $f \xrightarrow{F_L, F_0^{PI}}_+ r_L$
- $f \xrightarrow{F_H, F_0^{PI}}_+ r_H$

Let $V_{\mathbb{F}_q}(r_L), V_{\mathbb{F}_q}(r_H)$ denote the varieties of r_L and r_H, respectively, over the given field $\mathbb{F}_q$. Then the buggy circuit C admits a single-fix rectification at the net (gate output) x_i **if and only if** $V_{\mathbb{F}_q}(r_L) \cup V_{\mathbb{F}_q}(r_H) = \mathbb{F}_q^{|X_{PI}|} = V(J_0^{PI})$.

Proof. As rectification at net x_i makes the circuit C match the specification f, f should vanish on $V(J)$. Thus, the rectification condition can be equivalently stated as: "f vanishes on $V_{\mathbb{F}_q}(J) \iff V_{\mathbb{F}_q}(r_L) \cup V_{\mathbb{F}_q}(r_H) = \mathbb{F}_q^{|X_{PI}|}$."

(i) To prove $\Rightarrow$: Let $x_{PI} \in \mathbb{F}_q^{|X_{PI}|}$ be an assignment to the primary input variables of C. For every point x_{PI}, there exists a corresponding assignment x_{int} to the rest of the variables of the circuit. For each primary input assignment, the target net x_i evaluates to either $x_i = 0$ or $x_i = 1$. When $x_i = 0$, then J_H vanishes on the point (x_{PI}, x_{int}). Likewise, when $x_i = 1$, J_L vanishes on (x_{PI}, x_{int}). Since $f \xrightarrow{J_H, J_0}_+ r_H$ and $f \xrightarrow{J_L, J_0}_+ r_L$, and f vanishes on the point (x_{PI}, x_{int}), we obtain that either $r_H(x_{PI}) = 0$ or $r_L(x_{PI}) = 0$. In other words, for every primary input assignment x_{PI}, either r_L or r_H vanishes. This implies that $V(r_L) \cup V(r_H) = \mathbb{F}_q^{|X_{PI}|} = V(J_0^{PI})$.

(ii) To prove "$\Leftarrow$": Say there exists an assignment to the primary inputs $x_{PI} \in \mathbb{F}_q^{|X_{PI}|}$ such that r_H vanishes on x_{PI}, i.e. $r_H(x_{PI}) = 0$. Corresponding to x_{PI}, there exists an assignment to the rest of the variables of the circuit x_{int}. As $f \xrightarrow{J_H, J_0}_+ r_H$, we have that f is a member of the ideal $J_H + J_0 + \langle r_H \rangle$. Therefore, when $r_H(x_{PI}) = 0$, the ideal J_H also vanishes on (x_{PI}, x_{int}), and J_0 by definition vanishes everywhere. This implies that $f(x_{PI}, x_{int}) = 0$. Similarly, the argument also holds that when $r_L(x_{PI}) = 0$, then $f(x_{PI}, x_{int}) = 0$. This proves that for all primary inputs if r_L or r_H vanishes, then f vanishes too; and that completes the proof.

□

Note that the check "Is $V_{\mathbb{F}_q}(r_L) \cup V_{\mathbb{F}_q}(r_H) = \mathbb{F}_q^{|X_{PI}|} = V(J_0^{PI})$?" can be performed as shown below, where the union of varieties corresponds to the product of ideals.

$$\begin{aligned} V_{\mathbb{F}_q}(r_L) \cup V_{\mathbb{F}_q}(r_H) &= V_{\mathbb{F}_q}(r_L \cdot r_H) = V_{\overline{\mathbb{F}_q}}(\langle r_L \cdot r_H \rangle + J_0^{PI}) \\ &= V_{\mathbb{F}_q}(\langle r_L \cdot r_H \rangle + J_0^{PI}) \end{aligned}$$

Thus, to check for single-fix rectification at the net x_i, we need to compute the Gröbner basis $G = GB(\{r_L \cdot r_H\} \cup F_0^{PI})$ and see if G exactly equals F_0^{PI}.

Example V.2. Continuing with our running example, we demonstrate the rectification checks at nets e_3, s_1. As the bug was introduced at e_3, it is obvious that the circuit is rectifiable at e_3. For the rectification check at e_3, we mark the polynomial f_{10} for modification:

- $J_L = \langle F_L \rangle$, where $F_L = \{f_1, \ldots, f_{10} = e_3 + 1, \ldots, f_{16}\}$,
- $J_H = \langle F_H \rangle$, where $F_H = \{f_1, \ldots, f_{10} = e_3, \ldots, f_{16}\}$.

Reducing the specification $f : Z + A \cdot B$ modulo these ideals, we get:

- $r_L = f \xrightarrow{F_L, F_0^{PI}}_+ (\alpha + 1)a_1b_1b_0 + (\alpha + 1)a_1b_1$
- $r_H = f \xrightarrow{F_H, F_0^{PI}}_+ (\alpha + 1)a_1b_1b_0 + (\alpha)a_1b_0$

When we compute the Gröbner basis $G = GB(r_L \cdot r_H, F_0^{PI})$, we obtain $G = \{a_0^2 - a_0, a_1^2 - a_1, b_0^2 - b_0, b_1^2 - b_1\}$, corresponding to the ideal of all vanishing polynomials in primary inputs. This implies the existence of a rectification function at e_3.

In fact, the rectification test also passes for the net s_4; implying that the bug at e_3 can indeed be rectified at a different gate which does not lie in the fanin cone of e_3. However, the rectification test fails at net s_1. When the problem is formulated by modifying the polynomial f_{12} at net s_1, the corresponding computation for $G = GB(r_L \cdot r_H, F_0^{PI})$ results in $G = \{a_0^2 - a_0, b_0^2 - b_0, a_1^2 - a_1, b_1^2 - b_1, \ \ a_1b_0, \ a_0a_1b_1 + (\alpha)a_0a_1b_0\}$. Due to the presence of the last 2 polynomials, $G \neq F_0^{PI}$, and rectification is not possible at net s_1. In our experiments, the rectification check is performed on subset $\mathcal{N}$ starting from the net closest to the primary inputs with the intent of reducing variables in computed rectification function.

VI. Computing a Rectification Function

After the confirmation that the circuit indeed admits a rectification function at net x_i, our objective is to compute a rectification function $x_i = U(X_{PI})$. We call U the *unknown component* which has to be resolved. Due to the presence of internal *don't care* conditions, there may exist one or more polynomial functions U that may rectify the circuit. Our approach computes one of the candidate functions U, and proceeds as follows.

Once again, we use RTTO $>$ to represent the set of polynomials of the circuit. *The polynomial corresponding to the target net x_i is replaced by the polynomial $f_i = x_i + U(X_{PI})$,*

where $lm(f_i) = x_i$ and $tail(f_i) = U(X_{PI})$. In other words, the set F is updated to $F = \{f_1, \ldots, f_i = x_i + U, \ldots, f_s\}$. Notice that due to RTTO $>$, the set F still constitutes a Gröbner basis, as all polynomials in F have leading terms that are relatively prime. Moreover, by virtue of Prop. IV.1, the set $F \cup F_0^{PI}$ also constitutes a Gröbner basis. Thus, for a correct implementation, the condition $f \xrightarrow{F \cup F_0^{PI}}_+ 0$ still holds. Using Prop. III.1 and Eqn. 3, we can rewrite f in terms of these generators as:

$$f = h_1 f_1 + h_2 f_2 + \cdots + h_i f_i + \cdots + h_s f_s + \sum_{x_l \in X_{PI}} H_l(x_l^2 - x_l) \tag{7}$$

where $h_1, \ldots, h_s, H_l$ are arbitrary polynomials from the ring R. Substituting $f_i = x_i + U$ for the unknown component in Eqn. (7), we have:

$$f = h_1 f_1 + \cdots + h_{i-1} f_{i-1} + \boldsymbol{h_i x_i} + \boldsymbol{h_i U} + \cdots + h_s f_s + \sum_{x_l \in X_{PI}} H_l \cdot (x_l^2 - x_l) \tag{8}$$

$$f - h_1 f_1 - \cdots - h_{i-1} f_{i-1} - \boldsymbol{h_i x_i} = \boldsymbol{h_i U} + h_{i+1} f_{i+1} + \cdots + h_s f_s + \sum_{x_l \in X_{PI}} H_l \cdot (x_l^2 - x_l) \tag{9}$$

Notice that on the L.H.S. of Eqn. (9), the polynomials $f, f_1, \ldots, f_{i-1}$ and the monomial x_i are *known quantities/expressions*. Therefore, f can be divided by $f_1, \ldots, f_{i-1}$, and by x_i, to obtain the respective quotients of the division $h_1, \ldots, h_i$ 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. (9) consists of $h_i, f_{i+1}, \ldots, f_s$ and all the vanishing polynomials $x_l^2 - x_l$ as known expressions. This implies that:

$$f - h_1 f_1 - \cdots - h_i x_i \in \langle h_i, f_{i+1}, \ldots, f_s, x_l^2 - x_l \rangle \tag{10}$$

$$r \in \langle h_i, f_{i+1}, \ldots, f_s, x_l^2 - x_l \rangle \tag{11}$$

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. (3) and (4). As a result, we can write:

$$r = h_i' h_i + h_{i+1}' f_{i+1} + \cdots + h_s' f_s + \sum H_l(x_l^2 - x_l) \tag{12}$$

Then $U = h_i'$ is a polynomial function that forms the solution to the unknown component problem. Algorithmically, as $U = h_i'$ is computed as a quotient of division, U may contain any variables $X \subseteq \{x_1, \ldots, x_n\}$ in its support. However, due to the imposition of RTTO $>$, U will contain only those variables x_j in its support set that are less than x_i in the reverse topological order. Once such a polynomial U is obtained, it can be easily expressed in terms of the primary input variables. To achieve such a normalization, U can be reduced modulo the set of polynomials $\{f_j = x_j + tail(f_j)\}$ such that x_j lies in the fanin cone of U. Performing this division also in a reverse topological fashion results in U being expressed in primary inputs only. In this fashion, the polynomial $f_i : x_i + U(X_{PI})$ can be identified to implement the function of a subcircuit at the net x_i so that C correctly implements f.

Note that in Eqn. (11), while $\{f_{i+1}, \ldots, f_s\}$ constitutes a GB under RTTO, the set $\{h_i, f_{i+1}, \ldots, f_s\}$ may not. So a GB computation is required. On the other hand, we may also encounter situations when h_i results as being a constant in the field $\mathbb{F}_q$. When a constant is a member of an ideal J, then $GB(J) = \{1\}$. To arrive at an implementable solution in this case, we multiply r by the inverse of h_i (h_i^{-1}) and reduce the result modulo the rest of the polynomials$\{f_{i+1}, \ldots, f_s\}$.

$$r \cdot h_i^{-1} \xrightarrow{f_{i+1}} \xrightarrow{f_{i+2}} \ldots \xrightarrow{f_s}_+ U. \tag{13}$$

We now demonstrate the application of this approach on our running example.

Example VI.1. In Ex. V.2, we showed that rectification is possible at the net e_3, i.e. there exists a polynomial $f_{10} : e_3 + U$ that can rectify the circuit. Using the same term order as in the previous examples, we mark $f_{10} = e_3 + U$ as the unknown component, and include it in the set $F = \{f_1, \ldots, f_{10} = e_3 + U, \ldots, f_{16}\}$. Based on Eqns. (9)-(11), we begin reducing the specification polynomial f modulo the set $\{f_1, \ldots, f_9, e_3\} \cup F_0$. The reduction order for f based on RTTO $>$ is: $f \xrightarrow{f_1} \xrightarrow{f_2} \xrightarrow{f_3} \xrightarrow{f_4} \xrightarrow{f_5} \xrightarrow{f_6} \xrightarrow{f_7} \xrightarrow{f_8} \xrightarrow{f_9} \xrightarrow{lt(f_{10})} r$.

We will use the following notations to depict this reduction: '[]' to represent quotients of division h_j's, '()' to represent the divisors f_j's, and '{}' to represent the (partial) remainder fp_j obtained after every reduction step.

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

$fp_1 \xrightarrow{f_2} [B](A + a_0 + \alpha a_1) + \{Ba_0 + \alpha Ba_1 + z_0 + \alpha z_1\} \rightarrow fp_2$

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

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

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

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

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

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

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

Finally, the obtained remainder fp_9 is reduced by $lt(f_{10}) = e_3$ to obtain the quotient h_{10} and the remainder r:

$$fp_9 \xrightarrow{lt(f_{10})} \underbrace{[(\alpha + 1)s_1 + \alpha s_2]}_{h_{10}}(e_3) + \underbrace{\{s_0 + (\alpha+1)s_1 s_4 + \alpha s_2 s_4 + \alpha s_5 + \alpha a_0 b_1 + a_0 b_0 + (\alpha+1)a_1 b_1 + \alpha a_1 b_0\}}_{r}$$

Now that we have $r, h_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}$ available as known expressions, the unknown component problem can be formulated as an ideal membership test using Eqn. (11) such that:

$$r \in \langle h_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16} \rangle + \langle F_0^{PI} \rangle.$$

The above ideal membership can be solved by first computing the Gröbner basis of the generators and then expressing r as a linear combination of the ideal members:

$r = U \cdot h_{10} + h_{11} f_{11} + h_{12} f_{12} + h_{13} f_{13} + h_{14} f_{14} + h_{15} f_{15} + h_{16} f_{16}$

In this case, the ideal membership test results in the polynomial r being 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, $U = b_0$ is a solution to the *unknown component* f_{10}, i.e. $f_{10} = e_3 + b_0$. This depicts that e_3 implements just the primary input net b_0, thus also identifying redundancy in the design.

VII. Experiments

This section presents experimental results using our approach to debug the circuits and perform a single-fix rectification. We compare results of our implementation against the incremental SAT-based approach presented in [15] wherever it's relevant. The approach presented in [15] is implemented using PICOSAT [22]. The experiments were performed on a 3.5GHz Intel(R) Core™ i7-4770K Quad-Core CPU with 32 GB of RAM.

We have performed experiments for the cases when the bugs are present near the input, middle, or near the output of the circuit, represented using labels NI, NM, and NO respectively in the tables. All the algorithms were implemented in SINGULAR [23].

1) Verification between a word level specification v/s Mastrovito implementation: Table I presents the results of our approach when the bugs are placed in a Mastrovito multiplier implementation compared against a specification, which is given in terms of a word level polynomial f. A Mastrovito multiplier has word level specification $Z = A \times B \pmod{P(x)}$, where $P(x)$ is a given primitive polynomial for the datapath size k. Bugs in the circuit are introduced, and the presence of the bugs is detetced. Then we apply our approach to check for single-fix rectification interatively on the nets selected in $\mathcal{N}$. If rectification is feasible at x_i, the unknown component problem is solved to identify a rectification function.

We are able to verify and debug the circuits for upto 64-bits within our stipulated Time Out (TO) period.

2) Word level specification v/s Point addition implementation: Point addition is an operation required for the task of encryption, decryption and authentication in Elliptic Curve Cryptography (ECC). Modern approaches represent the points in projective coordinate systems, *e.g.*, the López-Dahab (LD) projective coordinate, due to which the point addition operation can be implemented as polynomials in the field.

Example VII.1. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ over $\mathbb{F}_{2^k}$, where $X, Y, Z \in \mathbb{F}_{2^k}$ and similarly, a, b are constants from the field. Point addition over the elliptic curve is (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1). Then X_3, Y_3, Z_3 can be computed as follows:

$$\begin{aligned} A &= Y_2 \cdot Z_1^2 + Y_1 & B &= X_2 \cdot Z_1 + X_1 \\ C &= Z_1 \cdot B & D &= B^2 \cdot (C + aZ_1^2) \\ Z_3 &= C^2 & E &= A \cdot C \\ X_3 &= A^2 + D + E & F &= X_3 + X_2 \cdot Z_3 \\ G &= X_3 + Y_2 \cdot Z_3 & Y_3 &= E \cdot F + Z_3 \cdot G \end{aligned}$$

Each of the polynomials in the above design are implemented as a (gate-level) logic block and are interconnected to obtain final outputs X_3, Y_3 and Z_3. Table II shows the comparison of the time required for debugging and rectification for the implementation of the block $D = B^2 \cdot (C + aZ_1^2)$.

TABLE II: Single fix rectification debug in Point Addition circuits against word level specification. Time is in seconds; k = Datapath Size, #Gates = No. of gates, K = 10^3, *a*=verification time, *b*=time for rectification check, *c*=time for component correction computation, *d*=total time

k	#Gates	Our implementation											
		NI				*NM*				*NO*			
		a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
8	244	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	1.2K	1.3	3.9	1.5	6.7	1.2	3.7	2	6.9	1.2	3.7	1.8	6.7
32	3.9K	37	112	77	226	38	110	22	170	37	108	35	180

3) Word level specification v/s Barrett reduction implementation: Barrett reduction is the other widely used multiplier design method adopted in cryptography system designs. Based on Barrett reduction, a multiplier can be designed in two steps: multiplication $R = A \times B$ and a subsequent Barrett reduction G = R (mod P). Table III shows results for debugging and rectification of Barrett multipliers against a polynomial specification.

TABLE III: Single fix rectification debug in Barrett reduction circuits against word level specification. Time is in seconds; k = Datapath Size, #Gates = No. of gates, K = 10^3, *a*=verification time, *b*=time for rectification check, *c*=time for component correction computation, *d*=total time

k	#Gates	Our implementation											
		NI				*NM*				*NO*			
		a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
8	134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	427	0.1	0.0	0.0	0.1	0.0	0.2	0.1	0.3	0.0	0.1	0.3	0.4
32	1.4K	0.4	1.4	0.1	1.9	0.5	1.5	0.1	2.1	1.2	2.2	1.1	4.5
64	4.9K	19	58	5.4	82	21	60	1.7	83	63	104	141	308

Since the SAT-based approach cannot be applied against a word level specification polynomial, we perform experiments while using another multiplier implementation as the specification.

4) Verification between a specification and implementation given as gate level circuits: Mastrovito v/s Montgomery multipliers: Montgomery architectures [24] are considered more efficient than Mastrovito multipliers for exponentiation, as they do not require explicit reduction modulo $P(x)$ after each step.

Table IV presents the results of our approach to debug and rectification with the bugs placed in the Montgomery

TABLE I: Single fix rectification debug in Mastrovito circuit against word level specification. Time is in seconds; k = Datapath Size, #Gates = No. of gates, K = 10^3, *a*=verification time, *b*=time for rectification check, *c*=time for component correction computation, *d*=total time

k	#Gates	Our implementation											
		NI				*NM*				*NO*			
		a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
9	0.23K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.29K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.35K	0.0	0.0	0.1	0.1	0.0	0.0	0.1	0.1	0.0	0.0	0.1	0.1
12	0.97K	0.1	0.5	0.4	0.9	0.2	0.5	0.4	1.1	0.5	0.8	0.4	1.7
13	0.82K	0.1	0.3	0.2	0.6	0.2	0.6	0.2	1.0	0.7	0.8	0.2	1.7
16	1.8K	0.9	2.6	1.0	4.5	1.1	3.5	1.0	5.6	2.8	5.3	1.0	9.1
32	5.4K	36	110	42	188	40	160	47	247	38	240	150	428
64	21.8K	2210	7100	2432	9532	2200	8000	2575	12775	2150	7840	10020	20010

TABLE IV: Rectification for Mastrovito circuit with Montgomery circuit as specification. Time is in seconds; k = Datapath Size, #Gates = No. of gates, (TO): Time-Out = 3 hrs, K = 10^3, *a*=verification time, *b*=time for rectification check, *c*=time for component correction computation, *d*=total time

k	#Gates	Incremental SAT **[15]**			Our Approach											
		NI	*NM*	*NO*	*NI*				*NM*				*NO*			
					a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
9	0.6K	35	37	33	0.1	0.5	0.2	0.8	0.2	0.2	0.1	0.5	1.8	2.2	0.6	4.6
10	0.7K	231	215	214	0.3	1	0.5	1.8	0.3	1	0.8	2.1	4.7	5.4	0.2	10
11	0.9K	2090	1927	2000	0.6	2	1	3.6	0.8	2	32	35	9	10	0.4	19
12	1.6K	8676	23400	24085	3.2	9.6	3.5	16	3.2	9.3	12	24	155	160	1.6	316
13	1.7K	TO	TO	TO	3.3	10	4.5	18	3.5	10	22	35	170	177	1.6	349
16	3K	TO	TO	TO	27	81	35	143	28	83	48	159	210	176	2.5	389
32	9.8K	TO	TO	TO	2060	6595	1870	10525	2100	7320	1289	10709	2215	7870	1204	11289

multiplier with a Mastrovito multiplier circuit used as the specification. While the approach [15] finds a satisfying transformation assignment which can be mapped to a library gate, our approach debugs the circuit and finds a single fix rectification function. As shown in the table, our approach shows improvement by several orders of magnitude over [15].

It takes considerable amount of time for verification and rectification check when the bug is close to the output. We are working on further improving the experiments by employing better data structures like ZBDDs ([25]), and devising better heuristics to perform rectification check. Due to several limitations w.r.t the number of ring variables that can be declared in SINGULAR, we have had to restrict our experiments within 64-bit data-path size.

VIII. Conclusions

This paper has presented a fully automated debug approach for single fix rectification of finite field arithmetic circuits. Given a specification and its circuit implementation, we verify the circuit. If verification detects a bug, we identify all potential single-fix rectification target nets, and perform rectification check at each of these nets. If a net admits single-fix rectification, we compute a corresponding rectification function. The underlying theory and algorithms are based on Gröbner basis reductions, Nullstellensatz, and ideal membership test. The experimental results demonstrate the efficacy of our approach for finite field arithmetic circuits where we achieve several orders of magnitude improvement as compared to recent SAT-based approach. As part of our future work, we are working on improving the efficiency of our implementation to target higher bit-widths. We are also investigating how the current procedure can be extended to cover integer arithmetic circuits. Further research also includes exploring the current approach for the case of multi-fix rectification.

References

[1] D. Ritirc, A. Biere, and M. Kauers, "Column-Wise Verification of Multipliers Using Computer Algebra," in *Formal Methods in Computer-Aided Design (FMCAD)*, 2017.

[2] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of Gate-level Arithmetic Circuits by Function Extraction," in *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[3] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal verification of Integer Multipliers by Combining Gröbner Basis with Logic Reduction," in *Design, Automation Test in Europe Conference Exhibition*, 2016.

[4] 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.

[5] A. Lvov, L. Lastras-Montano, B. Trager, V. Paruthi, R. Shadowen, and A. El-Zein, "Verification of Galois field based circuits by formal reasoning based on computational algebraic geometry," *Formal Methods in System Design*, vol. 45, no. 2, pp. 189–212, Oct 2014.

[6] F. Farahmandi and P. Mishra, "Automated Debugging of Arithmetic Circuits Using Incremental Gröbner Basis Reduction," in *IEEE International Conference on Computer Design (ICCD)*, 2017.

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

[8] K. F. Tang, C. A. Wu, P. K. Huang, and C. Y. Huang, "Interpolation-Based Incremental ECO Synthesis for Multi-Error Logic Rectification," in *Proc. Design Automation Conf. (DAC)*, 2011, pp. 146–151.

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

[10] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the Diagnosis and the Rectification of Design Errors with PRIAM," in *Proc. ICCAD*, 1989, pp. 30–33.

[11] H. T. Liaw, J. H. Tsaih, and C. S. Lin, "Efficient Automatic Diagnosis of Digital Circuits," in *Proc. ICCAD*, 1990, pp. 464–467.

[12] C. C. Lin, K. C. Chen, S. C. Chang, and M. Marek-Sadowska, "Logic Synthesis for Engineering Change," in *Proc. Design Automation Conf. (DAC)*, 1995, pp. 647–652.

[13] B. H. Wu, C. J. Yang, C. Y. Huang, and J. H. R. Jiang, "A Robust Functional ECO Engine by SAT Proof Minimization and Interpolation Techniques," in *International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 729–734.

[14] Vikas Rao, "Discussions on recent automatic debugging approaches," available at http://eng.utah.edu/~utkarshg/cex.pdf, 2018.

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

[16] 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.

[17] 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.

[18] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence Checking of Partial Designs Using Dependency Quantified Boolean Formulae," in *IEEE International Conference on Computer Design (ICCD)*, 2013.

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

[20] S. Gao, A. Platzer, and E. Clarke, "Quantifier Elimination over Finite Fields with Gröbner Bases," in *Intl. Conf. Algebraic Informatics*, 2011.

[21] E. Mastrovito, "VLSI Designs for Multiplication Over Finite Fields GF(2^m)," *Lecture Notes in CS*, vol. 357, pp. 297–309, 1989.

[22] A. Biere, "Picosat Essentials," *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2008.

[23] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 4-1-0 — A computer algebra system for polynomial computations," 2016.

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

[25] S.-i. Minato, "Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems," in *Proceedings of the 30th International Design Automation Conference*, 1993, pp. 272–277.