Efficient Symbolic Computation for Word-Level Abstraction From Combinational Circuits for Verification Over Finite Fields

Tim Pruss, Priyank Kalla, *Senior Member, IEEE*, and Florian Enescu

***Abstract*—This paper introduces a technique to derive a word-level abstraction of the function implemented by a combinational logic circuit. The abstraction provides a canonical representation of the function as a polynomial $Z = \mathcal{F}(A)$ over the finite field $\mathbb{F}_{2^k}$, where Z and A represent the k-bit output and input bit-vectors (words) of the circuit, respectively. This canonical abstraction can be utilized for formal verification and equivalence checking of combinational circuits. Our approach to abstraction is based upon concepts from computational commutative algebra and algebraic geometry. We show that the abstraction $Z = \mathcal{F}(A)$ can be derived by computing a Gröbner basis of the polynomials corresponding to the circuit, using a specific elimination term order derived from the circuit's topology. Computing Gröbner bases using elimination term orders is infeasible for large circuits. To overcome this limitation, we describe an efficient symbolic computation to derive the word-level polynomial. Our algorithms exploit: 1) the structure of the circuit; 2) the properties of Gröbner bases; 3) characteristics of finite fields $\mathbb{F}_{2^k}$; and 4) modern algorithms from symbolic algebra, to derive the canonical polynomial representation. This approach is employed to verify (and detect bugs in) large combinational finite field arithmetic circuits, where contemporary verification techniques are known to be infeasible.**

***Index Terms*—Equivalence checking, finite fields, formal verification, Gröbner bases, word-level abstraction.**

I. Introduction

FORMAL verification techniques can benefit greatly from abstractions of the functionality of the circuits that are being verified. Abstractions may reduce the complexity of analysis of the design and may provide a hierarchical view of the register transfer level (RTL) which may aid in RTL and system-level verification. Word-level abstraction specifically focuses on extracting a word-level representation of the function implemented by a gate-level design. It is desirable for the obtained word-level abstraction to be a canonical representation of the function, to facilitate formal verification and equivalence checking between a specification (golden) model against an optimized implementation.

This paper describes a method to derive a canonical word-level polynomial representation of a given gate-level combinational circuit. This abstraction polynomial is derived over the finite field of 2^k elements ($\mathbb{F}_{2^k}$)—where k corresponds to the size of the input/output bit-vectors (words)—and it represents the function implemented by the circuit. The circuit is modeled as a set of polynomials over $\mathbb{F}_{2^k}$, and concepts from computer-algebra and algebraic geometry (notably, Gröbner bases [1]) over finite fields are applied to derive the abstraction. An efficient algorithmic approach based on new concepts and discoveries is described to make our approach practical. The approach is based on the following mathematical insights.

Mathematical Framework: A combinational circuit C with k-bit inputs and outputs implements Boolean functions that are mappings between k-dimensional Boolean spaces: $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$, where $\mathbb{B} = \{0, 1\}$. The function f, which is a mapping among 2^k elements, can also be construed as a function $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$, i.e., as a function over the finite field of 2^k elements. Over the finite field ($\mathbb{F}_q$) of q elements, every function $f : \mathbb{F}_q \rightarrow \mathbb{F}_q$ is a polynomial function [2]. Moreover, there exists a unique canonical polynomial $\mathcal{F}$ that describes f. Motivated by this fundamental result, we devise an approach to derive a word-level, canonical, and polynomial abstraction of the function as $Z = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$, where $Z = \{z_0, \ldots, z_{k-1}\}$ and $A = \{a_0, \ldots, a_{k-1}\}$ are, respectively, the output and input bit-vectors (words) of the circuit C, and $\mathcal{F}$ denotes a polynomial representation of the circuit's functionality. The approach is generalized to circuits with different input/output bit-vector sizes, i.e., functions of the type $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, modeled as a polynomial over $f : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^m}$.

This abstraction is employed for formal verification and equivalence checking of combinational circuits C_1 and C_2. The circuits can be analyzed separately to derive their corresponding canonical polynomial representations $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. Equivalence test is then performed by simply matching the coefficients of $\mathcal{F}_1$ and $\mathcal{F}_2$.

The approach is targeted for hardware verification of cryptography primitives. Such designs perform polynomial computations over the finite field $\mathbb{F}_{2^k}$, where the datapath size k is very large. For example, the U.S. National Institute for Standards and Technology (NIST) recommends fields $\mathbb{F}_{2^k}$ corresponding to $k = 163, 233, 283, 409$, and 571 bits for

Manuscript received June 9, 2015; revised September 7, 2015; accepted October 10, 2015. Date of publication November 16, 2015; date of current version June 16, 2016. This work was supported by the U.S. National Science Foundation through the Division of Computing and Communication Foundations under Grant CCF-1320335 and Grant CCF-1320385. This paper was recommended by Associate Editor R. Drechsler.

T. Pruss is with Apple Computer, Cupertino, CA 95014 USA.

P. Kalla is with the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112 USA (e-mail: kalla@ece.utah.edu).

F. Enescu is with the Department of Mathematics and Statistics, Georgia State University, Atlanta, GA 30303 USA.

Digital Object Identifier 10.1109/TCAD.2015.2501301

elliptic-curve cryptography (ECC). For other non-ECC based crypto- and error-correcting circuits, k can be 1024 bits or larger! The large size and high complexity of such architectures necessitates hierarchical and custom design [3]–[5]. Custom design raises the potential for bugs in large systems. Arithmetic bugs are known to compromise the security of crypto-systems [6]; therefore, formal verification of such systems is an imperative. Our approach is particularly powerful for formal verification of hierarchical and custom finite field arithmetic architectures, where the specification (golden) models are structurally very dissimilar than their optimized implementations. Contemporary circuit verification techniques (see [7], [8]) are unable to prove equivalence between such large, custom, and modulo-arithmetic circuits.

A. Approach and Contributions

We analyze the given circuits and model the gate-level operators as polynomials with coefficients in $\mathbb{F}_{2^k}$, where k corresponds to the operand-size in the circuit. Using the concepts of Nullstellensatz over finite fields, projections of varieties, elimination ideals, and Gröbner bases [1], we formulate the polynomial abstraction problem as one of computing a Gröbner basis of this set of polynomials, using a specific elimination term order, called the abstraction term order (ATO) $>$.

Computing Gröbner bases using elimination orders is infeasible for large circuits. To overcome this limitation, we present a refinement of this ATO based on the topological analysis of the circuit. This refinement allows us to overcome the complexity of Gröbner basis computations, and derive the abstraction polynomial using efficient symbolic computation algorithms.

The technique is implemented as a standalone custom verification tool for canonical word-level abstraction from gate-level combinational circuits, and employed for formal verification and equivalence checking of finite field arithmetic circuits. Our approach can verify, and also find bugs in, large (up to $k = 1024$ bits) arithmetic circuits, whereas contemporary verification techniques are infeasible. Our approach is, however, not efficient for verification of random-logic and integer arithmetic circuits. This paper also discusses this particular limitation—which is a limitation not so much of our algorithms, but rather a (theoretical) limitation that is inherent in the complexity of the representation.

1) Paper Organization: Section II reviews related previous work in functional abstraction, combinational equivalence checking, and verification of finite field arithmetic circuits. Section III covers preliminary concepts related to finite fields, polynomial functions, and algebraic geometry. Section IV describes the main theoretical results of our approach on polynomial abstraction from circuits. A new, improved, and guided approach to abstraction is described in Section V. Our custom abstraction tool and experiments are described in Section VI. The limitations of our approach are also analyzed. Finally, Section VII concludes this paper.

II. Related Previous Work

A. Canonical Representations

The reduced ordered binary decision diagram [9]—and its variants ordered Kronecker functional decision diagrams, algebraic decision diagrams (ADDs), binary moment diagrams, etc.—are canonical directed acyclic graph (DAG) representations of functions that are employed in design verification. The various decomposition principles behind these diagrams are based on point-wise, binary decomposition, with respect to each (Boolean) variable. As such, these do not fully provide word-level abstraction capabilities from bit-level representations. The Taylor expansion diagram [10] is a word-level canonical representation of a polynomial expression, but it does not represent a polynomial function canonically. The work of [11] and [12] represents polynomial functions canonically, but over finite integer rings $\mathbb{Z}_{2^k}$ and not over $\mathbb{F}_{2^k}$. Multi-output decision diagrams (MODDs) [13] are a DAG representation of the characteristic function of a circuit over finite fields $\mathbb{F}_{2^k}$. However, MODDs do not scale well with respect to the circuit size.

B. Equivalence Checking

Modern equivalence checkers employ techniques based on and-invert-graph reductions [7] and circuit-satisfiability (SAT) solvers [14]. Such techniques are able to identify internal structural equivalences between the specification models (spec) and implementation circuits and reduce the instances for verification. However, when the arithmetic circuits are structurally very dissimilar, these techniques are infeasible in proving equivalence ([8, Tables I and II] depict such experiments). In general, the applications targeted in this paper are hard for SAT/SMT solvers.

C. Computer Algebra Based Verification

Lvov *et al.* [5] presented the BLUEVERI tool from IBM for verification of finite field error correcting circuits against an algorithmic spec. The implementation consists of a set of (predesigned and verified) circuit blocks that are interconnected to form the error correcting system. The spec is given as a set of design constraints on a "check file." Their objective is to prove the equivalence of the implementation against this check file, for which they employ a Nullstellensatz and Gröbner basis formulation. In their setting, the polynomial representation of the subcircuit blocks is already available, whereas our approach identifies such a representation.

Lv *et al.* [8] presented computer algebra techniques for formal verification of finite field arithmetic circuits. Given a specification polynomial f, and a circuit C, they formulate the verification problem as an ideal membership test using Nullstellensatz and Gröbner bases. They show that for any combinational circuit, there exists a term order (derived from the circuit) that renders the set of polynomials itself a Gröbner basis. By exploiting this term order, the need for Gröbner basis computation is avoided and verification is performed only by polynomial division. In contrast to [8], we are not given the specification polynomial f. Given the circuit C, we have to derive (abstract) the word-level specification f.

Among other relevant works, Gao [15] described how to use Gröbner basis techniques to count the zeros of an ideal over $\mathbb{F}_q$. The authors then follow-up with an approach for quantifier elimination over $\mathbb{F}_q$ [16]. Computer algebra techniques have also been employed for verification of integer arithmetic circuits [17].

D. Other Function Extraction Techniques

Ciesielski *et al.* [18] presented an approach to function extraction from bit-level circuits using a network-flow based model. Improvements to this approach are described in [19]. The extracted "signatures" are in terms of bit-level polynomials and do not provide a word-level abstraction.

E. Polynomial Interpolation

Conceptually, our abstraction can be derived using polynomial interpolation. It falls into the category of Newton's dense interpolation, which exhibits very high complexity. In the logic synthesis and very large-scale integration testing area, the work of [20] investigates dense interpolation. Due to its inherently high-complexity, their approach is feasible for applications over smaller fields, e.g., computing Reed–Muller forms for multivalued logic. To make such an approach practical, we propose a symbolic method based on computer algebra and algebraic geometry to derive the canonical polynomial abstraction from the circuit.

III. Preliminaries

A. Finite Fields and Polynomial Functions

A finite field, also called a Galois field, is a field with a finite number of elements. It is denoted as $\mathbb{F}_q$, where q corresponds to the number of elements, and it is always a power of a prime integer—i.e., $q = p^k$ where $p \geq 2$ is a prime integer and $k > 0$ is a positive integer. In this paper, we are concerned with binary Galois extension fields $\mathbb{F}_{2^k}$, where $p = 2$, so that the field contains $q = 2^k$ elements. We use the notations $\mathbb{F}_q$ and $\mathbb{F}_{2^k}$ interchangeably, with q always taken as 2^k.

The field $\mathbb{F}_{2^k}$ is constructed as $\mathbb{F}_{2^k} \equiv \mathbb{F}_2[x] \pmod{P(x)}$, where: 1) $\mathbb{F}_2 = \{0, 1\}$ denotes the finite field of two elements; 2) $\mathbb{F}_2[x]$ is the univariate polynomial ring with coefficients in $\mathbb{F}_2$; and 3) $P(x)$ denotes an irreducible (or primitive) polynomial in $\mathbb{F}_2[x]$ of degree k. $\mathbb{F}_{2^k}$ is a k-dimensional extension of the base field $\mathbb{F}_2$; all the field operations in $\mathbb{F}_{2^k}$ are performed modulo the irreducible polynomial $P(x)$ and the coefficients are reduced modulo $p = 2$ (due to which $-1 = +1$ over $\mathbb{F}_{2^k}$). In this paper, we always choose $P(x)$ to be a primitive polynomial and α as a primitive element.

Any element $A \in \mathbb{F}_{2^k}$ can be represented as $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$, where $a_i \in \mathbb{F}_2, i = 0, \ldots, k-1$, and α is a root of the primitive polynomial, i.e., $P(\alpha) = 0$. Since a k-bit vector $\{a_0, \ldots, a_{k-1}\}$ represents 2^k distinct values, it can be viewed as an element A of $\mathbb{F}_{2^k}$.

1) Polynomial Functions $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$: Arbitrary mappings among k-bit vectors can be constructed; each such mapping generates a function $f : \mathbb{B}^k \to \mathbb{B}^k$. Since every k-bit vector can be construed as an element in $\mathbb{F}_{2^k}$, every such function can be viewed as a mapping over $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. Importantly, every such function is also a polynomial function over $\mathbb{F}_{2^k}$.

Theorem 1 (From [2]): Any function $f : \mathbb{F}_q \to \mathbb{F}_q$ is a polynomial function over $\mathbb{F}_q$, that is there exists a polynomial $\mathcal{F} \in \mathbb{F}_q[x]$ such that $f(a) = \mathcal{F}(a)$, for all $a \in \mathbb{F}_q$.

By analyzing f over each of the q points, one can apply Lagrange's interpolation formula and interpolate a polynomial

$$\mathcal{F}(x) = \sum_{n=1}^{q} \frac{\prod_{i \neq n}(x - x_i)}{\prod_{i \neq n}(x_n - x_i)} \cdot f(x_n) \tag{1}$$

which is a polynomial of degree at most $q - 1$ in x. One can easily see that $\mathcal{F}(a) = f(a)$ for all $a \in \mathbb{F}_q$, and $\mathcal{F}(x)$ is therefore the polynomial representation of the function f.

An important property of finite fields is that for all elements $A \in \mathbb{F}_q, A^q = A$, and hence $A^q - A = 0$. Therefore, the polynomial $x^q - x$ vanishes on all points in $\mathbb{F}_q$. The polynomial $x^q - x$ is also referred to as a vanishing polynomial of $\mathbb{F}_q$. Any polynomial $\mathcal{F}(x)$ can be reduced $\pmod{x^q - x}$ to obtain a canonical representation $\mathcal{F}(x) \pmod{x^q - x}$ with degree at most $q - 1$. The result can be generalized as follows.

Definition 1: Any function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ has a unique canonical representation (UCR) as a polynomial $\mathcal{F} \in \mathbb{F}_q[x_1, \ldots, x_n]$ such that all its nonzero monomials are of the form $x_1^{i_1} \cdots x_n^{i_n}$ where $0 \leq i_j \leq q - 1$, for all $j = 1, \ldots, n$.

B. Hardware Designs Over $\mathbb{F}_{2^k}$ Verified in This Paper

In ECC, the operations of encryption, decryption, and authentication are built upon point-addition and point-doubling operations on elliptic curves over $\mathbb{F}_{2^k}$. These operations are implemented as polynomial computations (ADD, MULT) over $\mathbb{F}_{2^k}$ [21], as shown below.

Example 1: Consider point addition in López–Dahab projective coordinate. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ over $\mathbb{F}_{2^k}$, where X, Y, and Z are k-bit vectors that are elements in $\mathbb{F}_{2^k}$ and similarly, a and b are constants from the field. Let $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1)$ represent point addition over the elliptic curve. Then X_3, Y_3, and Z_3 can be computed as follows:

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

Over finite fields $\mathbb{F}_{2^k}$, multiplication is performed as $Z = A \times B \pmod{P(x)}$, where the multiplier circuit takes bit-level inputs $\{a_0, \ldots, a_{k-1}, b_0, \ldots, b_{k-1}\}$ and produces output $\{z_0, \ldots, z_{k-1}\}$, such that $A = \sum_{i=0}^{i=k-1} a_i\alpha^i$, $B = \sum_{i=0}^{i=k-1} b_i\alpha^i$, and $Z = \sum_{i=0}^{i=k-1} z_i\alpha^i$, where $P(\alpha) = 0$.

In one approach, the bit-wise multiplication $S = A \times B$ is computed using an array multiplier architecture, and then the result S is reduced $\pmod{P(x)}$ to obtain $Z = S \pmod{P(x)}$. Such architectures are termed Mastrovito multipliers [22]. In another approach, Montgomery reduction (MR) operations are used [4] that compute: $\mathrm{MR}(A, B) = A \cdot B \cdot R^{-1} \pmod{P(x)}$,

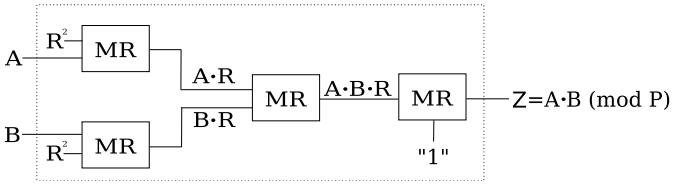

Fig. 1. Montgomery multiplication over $\mathbb{F}_{2^k}$ using four MRs.

where A and B are k-bit inputs, $R = \alpha^k$, $R^{-1} = (1/R)$, and $P(x)$ is the irreducible polynomial. Since MR(A, B) cannot directly compute $A \cdot B \pmod{P(x)}$, we need to precompute $A \cdot R$ and $B \cdot R$, as shown in Fig. 1.

In many non-ECC based crypto-systems, the datapath size k can be extremely large, e.g., $k = 1024$ bits. For such large designs, the concept of composite field arithmetic is used [3], where the field $\mathbb{F}_{2^k}$ is decomposed as $\mathbb{F}_{(2^m)^n}$ for a nonprime $k = m \cdot n$. This decomposition introduces a hierarchy (modularity) in the design by first lifting the base field from $\mathbb{F}_2$ to $\mathbb{F}_{2^m}$, and then constructing $\mathbb{F}_{(2^m)^n}$ as a n-dimensional extension of $\mathbb{F}_{2^m}$. Such circuits comprise m-bit $\mathbb{F}_{2^m}$ adders and multipliers, which are interconnected together to form a $k = m \cdot n$ bit circuit over $\mathbb{F}_{(2^m)^n}$ [3].

Example 2: An example of a composite field multiplier is shown in Fig. 2, where multiplication over $\mathbb{F}_{2^4}$ is decomposed as polynomial computations over $\mathbb{F}_{(2^2)^2}$. As shown in the figure, inputs $A = (a_3, \ldots, a_0)$ and $B = (b_3, \ldots, b_0)$ over $\mathbb{F}_{2^4}$ are first transformed into elements A_0, A_1, B_0, and B_1 over the base field $\mathbb{F}_{2^2}$; these are then interconnected (added and multiplied) to produce the final output $Z = \{Z_0, Z_1\}$.

Associated with both Montgomery multipliers and composite field circuits is a level of hierarchy (modularity) in design. With or without the availability of this hierarchy information, our approach can be applied to identify the function implemented the given circuit. However, when this hierarchy information is available, our approach can perform abstraction hierarchically and iteratively—significantly improving the efficiency of verification. In this paper, we have experimented with both flattened ("bit-blasted") and hierarchical implementations of the above circuits, for both buggy and bug-free implementations.

C. Algebraic Geometry and Symbolic Computation

1) Polynomial Rings and Term Orderings: We model the given combinational circuits with a set of multivariate polynomials with coefficients from the finite field $\mathbb{F}_q$. Let $\mathbb{F}_q[x_1, \ldots, x_d]$ be the polynomial ring in variables $x_1, \ldots, x_d$. A monomial in variables $x_1, \ldots, x_d$ is a power product of the form $X = x_1^{e_1} \cdot x_2^{e_2} \cdots x_d^{e_d}$, where $e_i \in \mathbb{Z}_{\geq 0}, i \in \{1, \ldots, d\}$. A polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_d]$ 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. To systematically manipulate the polynomials, a monomial order $>$ (term order) is imposed on the ring. The monomials of a polynomial $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$ are ordered with respect to $>$, such that $X_1 > X_2 > \cdots > X_t$. Subject to such a term order, $\mathrm{lt}(f) = c_1X_1$, $\mathrm{lm}(f) = X_1$, and $\mathrm{lc}(f) = c_1$, are the leading term, leading monomial, and leading coefficient of f, respectively. Also, denote $\mathrm{tail}(f) = f - \mathrm{lt}(f) = c_2X_2 + \cdots + c_tX_t$. This paper mostly considers terms ordered lexicographically (lex).

2) Polynomial Reduction: Polynomial reduction (division) plays a key role in our abstraction algorithms. Let f and g be polynomials. If a nonzero term cX of f is divisible by the leading term of g, then we say that f is reducible to r modulo g, denoted $f \xrightarrow{g} r$, where $r = f - (cX/\mathrm{lt}(g)) \cdot g$. Similarly, f can be reduced (divided) with respect to a set of polynomials $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r. This reduction is denoted $f \xrightarrow{F}_+ r$, and the remainder r has the property that no term in r is divisible by the leading term of any polynomial f_i in F.

3) The concepts of Ideals, Varieties, and Nullstellensatz:

Definition 2: An ideal J generated by polynomials $f_1, \ldots, f_s \in \mathbb{F}_q[x_1, \ldots, x_d]$ is

$$J = \langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i \cdot f_i : h_i \in \mathbb{F}_q[x_1, \ldots, x_d] \right\}.$$

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

Let $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{F}_q^d$ be a point, and $f \in \mathbb{F}_q[x_1, \ldots, x_d]$ be a polynomial. We say that f vanishes on $\mathbf{a}$ if $f(\mathbf{a}) = 0$. Then, for any ideal $J = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$, the variety of J over $\mathbb{F}_q$ is formally defined as

$$V_{\mathbb{F}_q}(J) = V(f_1, \ldots, f_s) = \left\{ \mathbf{a} \in \mathbb{F}_q^d : \forall f \in J, f(\mathbf{a}) = 0 \right\}.$$

In the context of this paper, the set of polynomials $F = \{f_1, \ldots, f_s\}$ describing the given circuit generates an ideal $J = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$. The variety $V_{\mathbb{F}_q}(J)$ corresponds to the set of all evaluations of the circuit. Then, to formulate our abstraction problem, we need to consider the ideals of polynomials that vanish on a variety V.

Definition 3: For any $V \subseteq \mathbb{F}_q^d$, the ideal of polynomials that vanish on V, called the vanishing ideal of V, is defined as: $I(V) = \{f \in \mathbb{F}_q[x_1, \ldots, x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$. Therefore, if a polynomial f vanishes on a variety V, then $f \in I(V)$.

The notation of sum of ideals is used below: if $I_1 = \langle f_1, \ldots, f_s \rangle$ and $I_2 = \langle h_1, \ldots, h_r \rangle$, then $I_1 + I_2 = \langle f_1, \ldots, f_s, h_1, \ldots, h_r \rangle$. Moreover, $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ is used to denote the ideal of all vanishing polynomials over $\mathbb{F}_q$.

Theorem 2 [Strong Nullstellensatz Over $\mathbb{F}_q$ (From [15])]: Let $J \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal, and let $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ be the ideal of all vanishing polynomials. Let $V_{\mathbb{F}_q}(J)$ denote the variety of J over $\mathbb{F}_q$. Then, $I(V_{\mathbb{F}_q}(J)) = J + J_0$.

4) Gröbner Bases: An ideal J may have many different generators: it is possible to have sets of polynomials $F = \{f_1, \ldots, f_s\}$ and $G = \{g_1, \ldots, g_t\}$ such that $J = \langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$ and $V(J) = V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$. A Gröbner basis is one such generating set that possesses many important properties that allow to solve many polynomial decision problems. In this paper, Gröbner bases are utilized as a canonical representation of an ideal.

Definition 4 ([Gröbner Basis] [1]): For a monomial ordering $>$, a set of nonzero polynomials $G = \{g_1, g_2, \ldots, g_t\}$ contained in an ideal J, is called a Gröbner basis of J iff $\forall f \in J, f \neq 0$, there exists $i \in \{1, \ldots, t\}$ such that $\mathrm{lm}(g_i)$

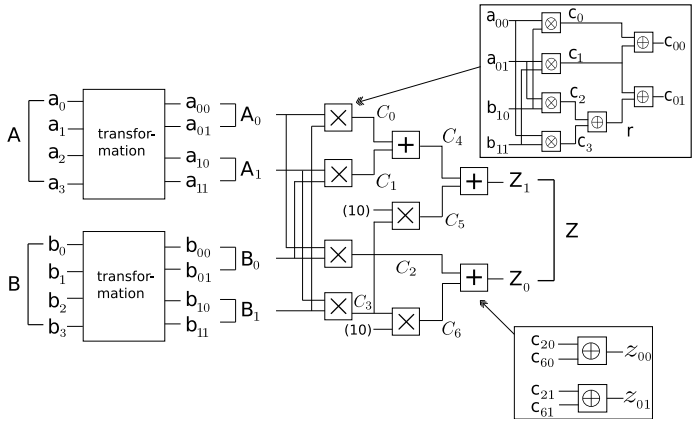

Fig. 2. 4-bit composite field multiplier designed over $\mathbb{F}_{(2^2)^2}$.

Algorithm 1: Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$
Output: $G = \{g_1, \ldots, g_t\}$
$G \coloneqq F$;
repeat
  $G' \coloneqq G$;
  for *each pair* $\{f, g\}, f \neq g$ *in* G' **do**
    $Spoly(f, g) \xrightarrow{G'}_{+} r$;
    if $r \neq 0$ **then**
      $G \coloneqq G \cup \{r\}$;
    end
  end
until $G = G'$;

divides $\mathrm{lm}(f)$; i.e., $G = \mathrm{GB}(J) \Leftrightarrow \ \forall f \in J : f \neq 0, \exists g_i \in G : \mathrm{lm}(g_i) \mid \mathrm{lm}(f)$.

Buchberger's algorithm [23], shown in Algorithm 1, computes a Gröbner basis over a field. Given polynomials $F = \{f_1, \ldots, f_s\}$, the algorithm computes the Gröbner basis $G = \{g_1, \ldots, g_t\}$. The algorithm takes pairs of polynomials (f, g), and computes their S-polynomial (Spoly(f, g))

$$\mathrm{Spoly}(f, g) = \frac{L}{\mathrm{lt}(f)} \cdot f - \frac{L}{\mathrm{lt}(g)} \cdot g$$

where $L = \mathrm{LCM}(\mathrm{lm}(f), \mathrm{lm}(g))$. Spoly$(f, g)$ cancels the leading terms of f and g. Therefore, the computation $\mathrm{Spoly}(f, g) \xrightarrow{G'}_{+} r$ results in a remainder r, which if nonzero, provides an element with new leading term in the generating set. The Gröbner basis algorithm terminates when for all pairs (f, g), $\mathrm{Spoly}(f, g) \xrightarrow{G'}_{+} 0$.

A Gröbner basis G may contain redundant elements. To remove these redundant elements, G is first made minimal and subsequently reduced. Subject to $>$, the reduced Gröbner basis $G = \{g_1, \ldots, g_t\}$ is a UCR of the ideal—a property we utilize for canonical polynomial abstraction.

IV. Word-Level Abstraction using Gröbner Basis

We are given a combinational circuit C with k-bit inputs and outputs, as shown in Fig. 3. Our objective is to derive a

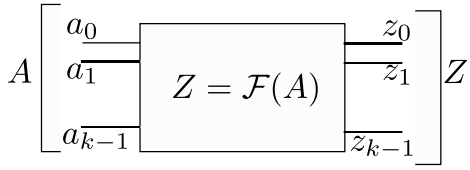

Fig. 3. Polynomial abstraction from a circuit.

canonical word-level abstraction polynomial $Z = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$ for the circuit C. Based on the datapath size k, $q = 2^k$ is chosen to model the circuit as a system of polynomials over $\mathbb{F}_q[x_1, \ldots, x_d, Z, A]$, where $\{x_1, \ldots, x_d\}$ correspond to all the bit-level variables (nets) in the circuit, Z and A are the word-level output and input, respectively. To construct the field, we choose a primitive polynomial $P(x)$ with the fewest terms, as it simplifies our computations.

Every Boolean logic gate in the circuit C is mapped to a polynomial function over $\mathbb{F}_2$ $(\subset \mathbb{F}_{2^k})$

$$\begin{aligned} \mathrm{NOT} &: \neg a \rightarrow a + 1 \pmod 2 \\ \mathrm{AND} &: a \wedge b \rightarrow a \cdot b \pmod 2 \\ \mathrm{OR} &: a \vee b \rightarrow a + b + a \cdot b \pmod 2 \\ \mathrm{XOR} &: a \oplus b \rightarrow a + b \pmod 2. \end{aligned} \tag{3}$$

Let $\{f_1, \ldots, f_s\}$ denote the set of polynomials derived from every Boolean gate in the circuit. Next, the word-level and bit-level correspondences over $\mathbb{F}_{2^k}$ are considered as $A = \sum_{i=0}^{k-1} a_i \cdot \alpha^i$ and $Z = \sum_{i=0}^{k-1} z_i \cdot \alpha^i$, where $P(\alpha) = 0$. These are represented as polynomials

$$\begin{aligned} f_A &: \ a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1} + A \\ f_Z &: \ z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1} + Z. \end{aligned} \tag{4}$$

Denote the ideal generated by all these polynomials as $J = \langle f_1, \ldots, f_s, f_A, f_Z \rangle$. The (unknown) word-level abstraction of the circuit $Z = \mathcal{F}(A)$ can be represented as the spec polynomial $f : Z + \mathcal{F}(A)$. The generators of J encapsulate the functionality of the circuit. Clearly, the spec polynomial $f : Z + \mathcal{F}(A)$ agrees with the solutions to the circuit's equations $f_1 = \cdots = f_s = f_A = f_Z = 0$. In other words, $f(\mathbf{a}) = 0$ for all points $\mathbf{a}$ that are solutions to $f_1 = \cdots = f_s = f_A = f_Z = 0$. In computer algebra terminology, we say that f vanishes on the variety $V_{\mathbb{F}_q}(J)$. This implies that

$f \in I(V_{\mathbb{F}_q}(J))$, due to Definition 3. Strong Nullstellensatz over Galois fields (Theorem 2) tells us that $I(V_{\mathbb{F}_q}(J)) = J + J_0$, where $J_0 = \langle x_1^2 - x_1, \ldots, x_d^2 - x_d, Z^q - Z, A^q - A\rangle$ is the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d, Z, A]$. Note that since the bit-level variables $x_1, \ldots, x_d$ take values in $\mathbb{F}_2$, the vanishing polynomial $x_i^2 - x_i$ is used; whereas $A^q - A$ and $Z^q - Z$ are used for the vanishing polynomials in word-level variables. From these results, we deduce the following

Proposition 1: The (unknown) abstraction polynomial $f : Z + \mathcal{F}(A)$ is a member of the ideal $J + J_0$.

A. Abstraction Using Gröbner Basis

The variety $V(J+J_0)$ is the set of all consistent assignments to the nets (signals) in the circuit C. If we project this variety on the word-level input and output variables, we essentially generate the function f implemented by the circuit. Projection of varieties from d-dimensional space $\mathbb{F}_q^d$ onto a lower dimensional subspace $\mathbb{F}_q^{d-l}$ corresponds to eliminating l variables from the corresponding ideal.

Definition 5 [Elimination Ideal (From [1])]: Given $J = \langle f_1, \ldots, f_s\rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$, the lth elimination ideal J_l is the ideal of $\mathbb{F}_q[x_{l+1}, \ldots, x_d]$ defined by: $J_l = J \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d]$.

In other words, the lth elimination ideal does not contain variables $x_1, \ldots, x_l$, nor do the generators of it. Moreover, Gröbner bases may be used to generate an elimination ideal by using an elimination term order. One such ordering is a pure lex ordering, which features into the theorem.

Theorem 3 [Elimination Theorem (From [1])]: Let $J \subset \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal and let G be a Gröbner basis of J with respect to a lex ordering where $x_1 > x_2 > \cdots > x_d$. Then for every $0 \leq l \leq d$, the set $G_l = G \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d]$ is a Gröbner basis of the lth elimination ideal J_l.

We wish to derive a polynomial representation from a circuit in variables Z and A. For this, we compute a Gröbner basis of $J + J_0$ with respect to an elimination order that eliminates all the (d) bit-level variables of the circuit. Then the Gröbner basis $G_d = G \cap \mathbb{F}_q[Z, A]$ of the dth elimination ideal of $J + J_0$ will contain polynomials in only Z, A. We will show that the desired canonical polynomial representation $f : Z + \mathcal{F}(A)$ will be found in G_d.

Problem Setup 1: Given a circuit C with k-bit inputs and outputs which computes a polynomial function $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. Let $A = \{a_0, \ldots, a_{k-1}\}$ and $Z = \{z_0, \ldots, z_{k-1}\}$ be the inputs and outputs of the circuit, respectively, such that $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$ and $Z = z_0 + \cdots + z_{k-1}\alpha^{k-1}$, where $P(\alpha) = 0$. Let $Z = \mathcal{F}(A)$ be the unknown polynomial function (spec) implemented by the circuit. Denote by $x_i, i = 1, \ldots, d$, all the Boolean (bit-level) variables of the circuit. Let $R = \mathbb{F}_{2^k}[x_i, Z, A : i = 1, \ldots, d]$ denote the corresponding polynomial ring and let ideal $J = \langle f_1, \ldots, f_s, f_A, f_Z\rangle \subset \mathbb{F}_{2^k}[x_i, Z, A : i = 1 \ldots d]$ be generated by the bit-level polynomials of the circuit $(f_1, \ldots, f_s)$ and the word-level designation polynomials (f_A, f_Z). Let $J_0 = \langle x_i^2 - x_i, Z^{2^k} - Z, A^{2^k} - A : i = 1, \ldots, d\rangle$ denote the ideal of vanishing polynomials in R.

The following term order is introduced for the purpose of abstraction.

Definition 6 (ATO $>$): Using the variable order $\{x_1, \ldots, x_d\} > Z > A$, impose a lex term order $>$ on $\mathbb{F}_q[x_1, \ldots, x_d, Z, A]$. This term order is defined as the ATO $>$. The relative ordering among the bit-level circuit variables $x_1, \ldots, x_d$ is not important and can be chosen arbitrarily.

Theorem 4 (Abstraction Theorem): Using the setup and notations given in Problem Setup 1, compute a Gröbner basis G of ideal $J + J_0$ using ATO $>$.

1) G must contain the vanishing polynomial $A^q - A$ as the only polynomial with only A as the support variable.
2) G must contain a polynomial of the form $Z + \mathcal{G}(A)$.
3) $Z + \mathcal{G}(A)$ is such that $\mathcal{F}(A) = \mathcal{G}(A), \forall A \in \mathbb{F}_q$. In other words, $\mathcal{G}(A)$ and $\mathcal{F}(A)$ are equal as polynomial functions over $\mathbb{F}_q$, and that $Z = \mathcal{G}(A)$ is a polynomial representation of the circuit C.

Proof: The proofs of the above three conditions are given below.

1) The vanishing polynomial $A^q - A$ is a given element of the generating set $J + J_0$. Variable A is also the last variable in the ATO. Moreover, A is an input to the circuit, so A is an independent variable which can take any and all values in $\mathbb{F}_q$. Since only a vanishing polynomial contains as solutions all points in $\mathbb{F}_q$, it follows that $G_{d+1} = G \cap \mathbb{F}_q[A] = \{A^q - A\}$.
2) Since $f : Z + \mathcal{F}(A)$ is a polynomial representation of the circuit, $Z + \mathcal{F}(A) \in J + J_0$, due to Proposition 1. Therefore, according to the definition of a Gröbner basis (Definition 4), the leading term of $Z + \mathcal{F}(A)$ (which is Z) should be divisible by the leading term of some polynomial $g_i \in G$. The only way $\mathrm{lt}(g_i)$ can divide Z is when $\mathrm{lt}(g_i) = Z$ itself. Moreover, due to our abstraction (lex) term order, $Z > A$, so this polynomial must be of the form $Z + \mathcal{G}(A)$.
3) As $Z = \mathcal{F}(A)$ represents the function of the circuit, $Z + \mathcal{F}(A) \in J + J_0$. Moreover, $V(J + J_0) \subseteq V(Z + \mathcal{F}(A))$. Project this variety $V(J + J_0)$ onto the co-ordinates corresponding to (A, Z). What we obtain is the graph of the function $A \mapsto \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$. Since $Z + \mathcal{G}(A)$ is an element of the Gröbner basis of $J + J_0$, $V(J + J_0) \subseteq V(Z + \mathcal{G}(A))$ too. Due to this inclusion of varieties, the points that satisfy $J + J_0$ also satisfy $Z + \mathcal{G}(A) = 0$ and $Z + \mathcal{F}(A) = 0$. Therefore, $Z = \mathcal{G}(A)$ gives the same function as $Z = \mathcal{F}(A)$, for all $A \in \mathbb{F}_{2^k}$. ■

Corollary 1: Let $G_{\mathrm{red}} = \{g_1, \ldots, g_t\}$ denote the reduced Gröbner basis of $J + J_0$ with respect to ATO $>$. Then G_{red} contains the one and only polynomial of the form $g_i : Z + \mathcal{F}(A)$, such that $Z = \mathcal{F}(A)$ is the unique, canonical representation of the function f implemented by the circuit.

Theorem 4 and Corollary 1 can be applied to abstract the canonical word-level representation of combinational circuits with an arbitrary number of k-bit operands. If the circuit contains multiple word-level inputs $A_1, \ldots, A_n$, each k-bit wide, then ATO can be extended to include these variables by imposing a lex term order $>$ with $\{x_1, \ldots, x_d\} > Z > A_1 > \cdots > A_n$. Subsequently, the reduced Gröbner basis of $J + J_0$ computed with ATO contains $f : Z + \mathcal{F}(A_1, \ldots, A_n)$ as the only polynomial in variables $Z, A_1, \ldots, A_n$, corresponding to the desired abstraction.

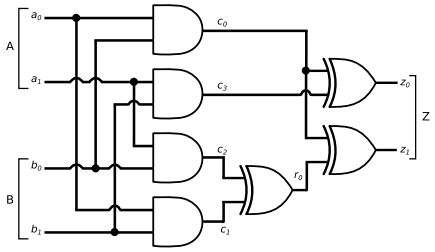

Fig. 4. Arbitrary circuit modeled over $\mathbb{F}_{(2^2)}$.

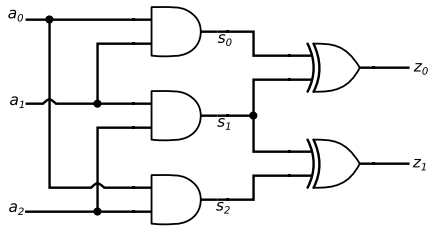

Fig. 5. Abstraction over a circuit with varying word sizes.

Example 3: Consider the circuit of Fig. 4. As the circuit contains 2-bit inputs and outputs, we will abstract a polynomial $Z = \mathcal{F}(A, B)$ over $\mathbb{F}_{2^2}$ by computing a reduced Gröbner basis of the polynomials derived from the circuit. To construct $\mathbb{F}_{2^2}$, we use $P(x) = x^2 + x + 1$, with $P(\alpha) = 0$.

With the mapping rules given in (3), the Boolean equations are transformed into polynomials over $\mathbb{F}_2$

$$\begin{aligned} c_0 &= a_0 \wedge b_0 \rightarrow f_1 : c_0 + a_0 \cdot b_0 \\ c_1 &= a_0 \wedge b_1 \rightarrow f_2 : c_1 + a_0 \cdot b_1 \\ c_2 &= a_1 \wedge b_0 \rightarrow f_3 : c_2 + a_1 \cdot b_0 \\ c_3 &= a_1 \wedge b_1 \rightarrow f_4 : c_3 + a_1 \cdot b_1 \\ r_0 &= c_1 \oplus c_2 \rightarrow f_5 : r_0 + c_1 + c_2 \\ z_0 &= c_0 \oplus c_3 \rightarrow f_6 : z_0 + c_0 + c_3 \\ z_1 &= r_0 \oplus c_0 \rightarrow f_7 : z_1 + r_0 + c_0. \end{aligned}$$

The word-level designation polynomials are: $f_A : a_0 + a_1 \cdot \alpha + A$; $f_B : b_0 + b_1 \cdot \alpha + B$; $f_Z : z_0 + z_1 \cdot \alpha + Z$. Thus the ideal $J = \langle f_1, \ldots, f_7, f_A, f_B, f_Z \rangle$ is generated by the polynomials derived from the circuit. The vanishing polynomials in our system are

$$\begin{array}{lll} f_8 : a_0^2 + a_0 & f_9 : a_1^2 + a_1 & f_{10} : b_0^2 + b_0 \\ f_{11} : b_1^2 + b_1 & f_{12} : c_0^2 + c_0 & f_{13} : c_1^2 + c_1 \\ f_{14} : c_2^2 + c_2 & f_{15} : c_3^2 + c_3 & f_{16} : r_0^2 + r_0 \\ f_{17} : z_0^2 + z_0 & f_{18} : z_1^2 + z_1 & f_{19} : A^4 + A \\ f_{20} : B^4 + B & f_{21} : Z^4 + Z. & \end{array}$$

Then $J_0 = \langle f_8, \ldots, f_{21} \rangle$ and $J + J_0 = \langle f_1, \ldots, f_{21}, f_A, f_B, f_Z \rangle$. Impose the following ATO, i.e., a lex order with $\{z_0 > z_1 > r_0 > c_0 > c_1 > c_2 > c_3 > a_0 > a_1 > b_0 > b_1\} >$ "Output Z" > "Inputs, $A > B$" and compute a reduced Gröbner basis G_{red} of $J + J_0$. The resulting basis contains 14 polynomials

$$\begin{aligned} g_1 &: \boldsymbol{B^4 + B} \\ g_2 &: \boldsymbol{A^4 + A} \\ g_3 &: \boldsymbol{Z + (\alpha + 1) \cdot A^2 \cdot B^2} \\ g_4 &: b_1 + B^2 + B \\ &\vdots \\ g_{14} &: z_0 + \alpha \cdot A^2 \cdot B^2 + (\alpha + 1) \cdot A \cdot B. \end{aligned}$$

As expected, the first two polynomials are the vanishing polynomials in word-level inputs and the polynomial g_3 is the only polynomial in variables Z, A, and B which represents the polynomial function of the circuit C as $Z = (\alpha + 1) \cdot A^2 \cdot B^2$.

B. Generalizing the Approach for Functions $f : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^n}$

When the word sizes of the inputs and output of the circuits vary, the functionality of the circuit must be analyzed over an encompassing field. Let m be the size of the input bit-vector A and n be with size of the output Z such that $m \neq n$. Then the circuit implements a function over $f : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^n}$. In such cases, the abstraction can be performed over $\mathbb{F}_{2^k}$ where $k = \text{LCM}(m, n)$, by virtue of the following result [24].

Lemma 1: The field $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ when $n \mid k$.

By selecting $k = \text{LCM}(m, n)$, the field $\mathbb{F}_{2^k}$ becomes the smallest single field containing both $\mathbb{F}_{2^m}$ and $\mathbb{F}_{2^n}$. Let α, β, and γ be the primitive elements of $\mathbb{F}_{2^k}, \mathbb{F}_{2^m}$, and $\mathbb{F}_{2^n}$, respectively. The word-level designation polynomials now become

$$\begin{aligned} f_A &: a_0 + a_1\beta + \cdots + a_{m-1}\beta^{m-1} + A \\ f_Z &: z_0 + z_1\gamma + \cdots + z_{n-1}\gamma^{n-1} + Z. \end{aligned} \tag{5}$$

Since the analysis is performed over $\mathbb{F}_{2^k}$, β, and γ must be mapped to α. This is accomplished as [24]

$$\begin{aligned} \beta &= \alpha^{(2^k-1)/(2^m-1)} \\ \gamma &= \alpha^{(2^k-1)/(2^n-1)}. \end{aligned} \tag{6}$$

By replacing β and γ in terms of α in (5), the abstraction can be performed as before by computing the reduced Gröbner basis of the ideal $J + J_0$. However, care should be taken to compose the vanishing polynomials: $x_i^2 - x_i$ for the bit-level variables, $A^{2^m} - A$ for the m-bit input, and $Z^{2^n} - Z$ for the n-bit output.

Example 4: Consider the circuit shown in Fig. 5. The input A is 3 bits wide while the output Z is 2 bits. Thus, $A \in \mathbb{F}_{2^3}$ and $Z \in \mathbb{F}_{2^2}$. Let β be the primitive element of $\mathbb{F}_{2^3}$ and γ be the primitive element of $\mathbb{F}_{2^2}$, i.e., $A = a_0 + a_1\beta + a_2\beta^2$ and $Z = z_0 + z_1\gamma$. The function $\mathbb{F}_{2^3} \rightarrow \mathbb{F}_{2^2}$ must be analyzed over $\mathbb{F}_{2^6}$ since $\text{LCM}(2, 3) = 6$. Choose $P(X) = X^6 + X + 1$ with $P(\alpha) = 0$, and find β and γ in terms of α: $\beta = \alpha^{(2^6-1)/(2^3-1)} = \alpha^9$ and $\gamma = \alpha^{(2^6-1)/(2^2-1)} = \alpha^{21}$.

So the word-level polynomials represented in $\mathbb{F}_{2^6}$ are: $f_A : a_0 + a_1\alpha^9 + a_2\alpha^{18} + A$; $f_Z : z_0 + z_1\alpha^{21} + Z$, and the bit-level polynomials are derived from the circuit as before. Collectively, these generate ideal J. On the other hand, in J_0, the vanishing polynomials corresponding to the bit-level variables are included as $\{x_i^2 + x_i\}$, whereas the vanishing polynomials of the word-level variables are composed according to their respective operand sizes: $A^{2^3} + A$ and $Z^{2^2} + Z$. Then, by computing the reduced Gröbner basis of $J + J_0$, the word-level

abstraction of the circuit $Z + \mathcal{F}(A)$ is found to be

$$Z + A^6\left(\alpha^2+\alpha\right) + A^5\left(\alpha^4+\alpha^3+\alpha\right) + A^4\left(\alpha^2+\alpha\right) + A^3\left(\alpha^4+\alpha^3+\alpha^2\right) + A^2\left(\alpha^4+\alpha^3+\alpha^2\right) + A\left(\alpha^4+\alpha^3+\alpha\right).$$

1) Preliminary Experiments: Using the results of Theorem 4 and Corollary 1, we performed some proof-of-concept experiments to evaluate the efficacy of our approach to abstraction. We experimented with Galois field Mastrovito multipliers, and employed the SINGULAR computer algebra tool [25] to derive the abstraction polynomial $Z + A \cdot B$, using the slimgb command to compute the reduced Gröbner basis of $J + J_0$ using ATO. We found that beyond $k = 32$-bit operands, the reduced Gröbner basis computation explodes in both time and space, and the abstraction becomes infeasible.

Computing Gröbner bases using elimination term orders is infeasible for large circuits. The worst-case time and space complexity of computing the Gröbner basis of $J + J_0$ in $\mathbb{F}_q[x_1, \ldots, x_d]$ is known to be bounded by $q^{O(d)}$ [15], which is prohibitive over large fields. To make our approach practical, we need to overcome this complexity. This is described next.

V. Efficient Search for the Abstraction

The aforementioned complexity makes the computation of a reduced Gröbner basis infeasible. However, our abstraction approach "searches" for only one polynomial ($Z+\mathcal{F}(A)$) in the basis. This motivates an investigation into whether it is possible to guide a sequence of $\text{Spoly}(f, g) \xrightarrow{J+J_0}_+ r$ computations to arrive at the desired word-level polynomial, without considering other polynomials in the generating set. For this purpose, we exploit the well-known Buchberger's product criteria.

Lemma 2 (Product Criterion [26]): Let $\mathbb{F}$ be any field, and $f, g \in \mathbb{F}[x_1, \ldots, x_d]$ be polynomials. If the equality $\text{lm}(f) \cdot \text{lm}(g) = \text{LCM}(\text{lm}(f), \text{lm}(g))$ holds, then $\text{Spoly}(f, g) \xrightarrow{G}_+ 0$.

The above result states that when the leading monomials of f, g are relatively prime, then $\text{Spoly}(f, g)$ always reduces to 0 modulo the basis G. Thus $\text{Spoly}(f, g)$ corresponding to the critical pair (f, g) need not be considered in Buchberger's algorithm. Recall that in the ATO (Definition 6), the relative ordering among the bit-level circuit variables $x_1, \ldots, x_d$ is unimportant. This ordering is now further refined to exploit the product criteria. For this purpose, we draw inspirations from [17, Proposition 2] that shows how to derive a term order from the circuit that makes leading terms of all pairs of gate-level polynomials relatively prime.

Definition 7 (Refined ATO (RATO) $>_r$): Starting from the primary outputs of the circuit C, perform a reverse topological traversal toward the primary inputs. Order each variable of the circuit according to its reverse topological level: i.e., $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lex term order $>_r$ on $\mathbb{F}_q[x_1, \ldots, x_d, Z, A]$ with the "bit-level variables $x_1, \ldots, x_d$ ordered reverse topologically" $> Z > A$. This term order $>_r$ is called the RATO.

Denote $F = \{f_1, \ldots, f_s, f_A, f_Z\}$ to be the set of polynomials which generates the ideal $J = \langle F \rangle$ and denote F_0 to be the set of vanishing polynomials which generates the ideal $J_0 = \langle F_0 \rangle$. Let us impose RATO on the ring, and analyze the characteristics of the generating set $F \cup F_0$. First, we consider only the bit-level polynomials $f_1, \ldots, f_s$ derived from the logic gates in the circuit. Due to RATO, each bit-level polynomial will be of the form $f_i = x_i + \text{tail}(f_i)$, where x_i is the output of the corresponding logic gate. Since the same signal cannot be the output of two or more gates, each polynomial pair $(f_i, f_j), i \neq j$ will have relatively-prime leading terms. Consequently, $\text{Spoly}(f_i, f_j) \xrightarrow{F \cup F_0}_+ 0$ for all bit-level polynomials due to the product criteria, and need not be considered in the Gröbner basis computation.

Also, corresponding to each bit-level polynomial $f_i = x_i + \text{tail}(f_i)$, there exists a bit-level vanishing polynomial $x_i^2 + x_i$. While their leading terms are not relatively prime, it was shown in [8, Th. 6.1] that $\text{Spoly}(f_i, x_i^2 + x_i) \xrightarrow{F \cup F_0}_+ 0$. To show this, let us denote $\text{tail}(f_i) = P_i$ so that $f_i = x_i + P_i$. Also, every variable x_j that appears in P_i satisfies $x_i > x_j$. Then $\text{Spoly}(f_i, x_i^2 + x_i) = x_i P_i + x_i$, which can be reduced by the polynomial $f_i \in F$

$$(x_i P_i + x_i) \xrightarrow{x_i + P_i} x_i + P_i^2 \xrightarrow{x_i + P_i} P_i^2 + P_i.$$

Note that since $P_i = \text{tail}(x_i)$ contains only bit-level variables, $P_i^2 + P_i$ is a vanishing polynomial, or $P_i^2 + P_i \xrightarrow{F_0}_+ 0$. Therefore, the S-polynomials $\text{Spoly}(f_i = x_i + \text{tail}(f_i), x_i^2 + x_i) \xrightarrow{F \cup F_0}_+ 0$ for all $i = 1, \ldots, d$; so these also need not be considered in the Gröbner basis computation.

However, there is one (and only one) pair of polynomials $(f_Z, f_{z_i}) \in F$ which do not have relatively prime leading terms, and for which $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_+ r$ results in a new polynomial in the Gröbner basis computation. Here: 1) f_Z is the word-level designation polynomial corresponding the output $f_Z = z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1} + Z$, with some gate output z_i as the leading term and 2) the polynomial $f_{z_i} = z_i + \text{tail}(f_{z_i})$ models the function implemented at the gate, and $\text{lt}(f_{z_i}) = z_i$.

So let us analyze the remainder r obtained as $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_+ r$. Due to RATO, r does not contain any bit-level nonprimary input variables of the circuit C, and it may only depend upon: 1) the word-level variables (Z, A) and 2) the primary input bits $(a_0, \ldots, a_{k-1})$. To show this, assume that r contains a bit-level nonprimary input variable x_j in a term m_j. Since there exists a polynomial $f_j = x_j + \text{tail}(f_j) \in F$, $\text{lt}(f_j) \mid m_j$, and all such terms m_j will be canceled during the reduction $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_+ r$. Variables Z and A never appear as leading terms of any polynomial in F as they appear last in RATO. Similarly, the bit-level primary inputs $a_0, \ldots, a_{k-1}$ also never appear as leading terms of any polynomial in F, as primary inputs are not outputs of any gate. Based on the above discussion, we conclude the following.

Proposition 2: Due to RATO, (f_Z, f_{z_i}) is the only candidate critical pair to be evaluated as $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F, F_0}_+ r$ at the start of Buchberger's algorithm when applied to our setup. Moreover, the obtained remainder r is a function only in variables $a_0, \ldots, a_{k-1}, Z$, and A.

Example 5: Let us revisit Example 3 and the corresponding circuit shown in Fig. 4. Impose RATO: lex term order with $\{z_0 > z_1\} > \{r_0 > c_0 > c_3\} > \{c_1 > c_2\} > \{a_0 > a_1 > b_0 > b_1\} > Z > A$. Then, the set of polynomials $F = \{f_1 \ldots, f_{21}, f_Z, f_A, f_B\}$ shown in Example 3 are already represented in RATO.

Notice that the pair $(f_Z, f_6) \in F$ is the only critical pair with leading terms that are not relatively prime. Due to Proposition 2, computing $\text{Spoly}(f_Z, f_6) \xrightarrow{F \cup F_0}_+ r$, we find that $r = (\alpha+1)a_1b_1 + (\alpha+1)a_1B + (\alpha+1)b_1A + Z + (\alpha+1)AB$. Note that the remainder r contains word-level variables Z, A, and B, and the bit-level primary inputs a_1 and b_1. Intermediate bit-level variables (non primary inputs) do not appear in r.

A. Eliminating Bit-Level Variables

The remainder r obtained in Proposition 2 is a function of the primary input variables, in addition to the word-level variables. In order to derive a purely word-level expression, the bit-level variables need to be eliminated from r. We now derive a functional (polynomial) mapping from each bit-level primary input variable $a_0, \ldots, a_{k-1}$ to the word-level input variable A in the form of $a_i = \mathcal{F}_{a_i}(A)$. Then substituting each $a_i = \mathcal{F}_{a_i}(A)$ in r will result in a purely word-level expression. These mappings are derived as a set of polynomial functions $F_a = \{f_{a_0}, \ldots, f_{a_{k-1}}\}$ in the following form:

$$\begin{aligned} a_0 = \mathcal{F}_{a_0}(A) &\rightarrow f_{a_0} : a_0 + \mathcal{F}_{a_0}(A) \\ \vdots &\rightarrow \vdots \\ a_{k-1} = \mathcal{F}_{a_{k-1}}(A) &\rightarrow f_{a_{k-1}} : a_{k-1} + \mathcal{F}_{a_{k-1}}(A) \end{aligned}$$

where each $\mathcal{F}_{a_i}(A)$ represents some polynomial function of A.

Due to RATO, terms in $\{a_0, \ldots, a_{k-1}\} > A$, thus the leading terms of $f_{a_0}, \ldots, f_{a_{k-1}}$ are $a_0, \ldots, a_{k-1}$, respectively. Then $r \xrightarrow{F_a \cup F_0}_+ r_w$ ensures that the new remainder r_w must contain only word-level variables. In other words, r_w must be in the form $Z + \mathcal{F}(A)$ and is the canonical word-level polynomial representation of the circuit.

Lemma 3 (From [24]): Let $\alpha_1, \ldots, \alpha_t$ be elements in $\mathbb{F}_{2^k}$. Then $(\alpha_1 + \alpha_2 + \cdots + \alpha_t)^{2^i} = \alpha_1^{2^i} + \alpha_2^{2^i} + \cdots + \alpha_t^{2^i}$ for $i \geq 1$.

Lemma 3 can be applied to derive the desired mapping. We take the word-level designation polynomial $f_A : A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$, and compute A^{2^j} for all $0 \leq j < k$

$$\begin{aligned} A &= a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1} \\ A^2 &= a_0^2 + a_1^2\alpha^2 + \cdots + a_{k-1}^2\alpha^{2(k-1)} \\ &= a_0 + a_1\alpha^2 + \cdots + a_{k-1}\alpha^{2(k-1)} \ \left(a_i^2 = a_i\right) \\ A^{2^2} &= a_0 + a_1\alpha^4 + \cdots + a_{k-1}\alpha^{4(k-1)} \\ &\vdots \\ A^{2^{k-1}} &= a_0 + a_1\alpha^{2^{k-1}} + \cdots + a_{k-1}\alpha^{2^{k-1}\cdot(k-1)}. \end{aligned} \tag{7}$$

These equations can be represented in matrix form, $\mathbf{A} = \mathbf{Ma}$, where $\mathbf{A} = [A, A^2, \ldots, A^{2^{k-1}}]^T$, $\mathbf{M}$ is a $k \times k$ matrix of coefficients, and $\mathbf{a} = [a_0, \ldots, a_{k-1}]^T$

$$\begin{bmatrix} A \\ A^2 \\ A^{2^2} \\ \vdots \\ A^{2^{k-1}} \end{bmatrix} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{k-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{(k-1)\cdot 2} \\ 1 & \alpha^4 & \alpha^8 & \ldots & \alpha^{(k-1)\cdot 4} \\ \vdots & \vdots & \vdots & . & \vdots \\ 1 & \alpha^{2^{k-1}} & \alpha^{2\cdot 2^{k-1}} & \ldots & \alpha^{(k-1)\cdot 2^{k-1}} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{k-1} \end{bmatrix}. \tag{8}$$

Treat $\mathbf{a}$ as a vector of k unknowns, $\mathbf{M}$ and $\mathbf{A}$ as matrices of constants. This represents a system of k linear equations in k unknowns $\{a_0, \ldots, a_{k-1}\}$. Then F_a can be derived by solving (8) using Cramer's rule

$$a_i = \frac{|\mathbf{M_i}|}{|\mathbf{M}|}, \quad 0 \leq i \leq k-1 \tag{9}$$

provided that $|\mathbf{M}| \neq 0$. Here $\mathbf{M_i}$ corresponds to the matrix $\mathbf{M}$ where the ith column $[\alpha^i, \alpha^{i\cdot 2}, \ldots, \alpha^{i\cdot 2^{k-1}}]^T$ in $\mathbf{M}$ is replaced by the vector $\mathbf{A} = [A, A^2, \ldots, A^{2^{k-1}}]^T$.

Notice that in (8), elements in every row of $\mathbf{M}$ form a geometric progression; this makes $\mathbf{M}$ a Vandermonde matrix, whose determinant is computed with a simple formula.

Definition 8: Let $\mathbf{V}(x_1, \ldots, x_n)$ denote a square $n \times n$ matrix of the form

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & \vdots & . & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{bmatrix}. \tag{10}$$

Then $\mathbf{V}(x_1, \ldots, x_n)$ is a square Vandermonde matrix, the determinant of which can be computed as

$$|\mathbf{V}(x_1, \ldots, x_n)| = \prod_{1 \leq i < j \leq n} \left(x_j - x_i\right). \tag{11}$$

This determinant is nonzero if each $x_i \in \{x_1, \ldots, x_n\}$ is a distinct element.

Matrix $\mathbf{M}$ in (8) is a $k \times k$ square Vandermonde matrix of the form $\mathbf{V}(\alpha, \alpha^2, \ldots, \alpha^{2^{k-1}})$. In our investigations, we discovered that $|\mathbf{M}| = 1$, which we prove below.

Lemma 4: For matrix $\mathbf{M}$ in (8), $|\mathbf{M}| = 1$ over $\mathbb{F}_{2^k}$.

Proof: $\mathbf{M}$ is a Vandermonde matrix of the form $\mathbf{V}(\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{k-1}}) = \mathbf{V}(\alpha^{2^0}, \alpha^{2^1}, \alpha^{2^2}, \ldots, \alpha^{2^i}, \ldots, \alpha^{2^{k-1}})$. The Vandermonde determinant $|\mathbf{V}(\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{k-1}})|$ is nonsingular if the elements $\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{k-1}}$ are distinct (Definition 8). Since $\mathbb{F}_{2^k}$ is constructed from a primitive polynomial, α is a primitive element, and every α^{2^i} for $0 \leq i < 2^k$ is a distinct nonzero element. Then by Definition 8, $|\mathbf{M}| \neq 0$. Moreover, from (11) it follows that:

$$|\mathbf{M}| = \prod_{0 \leq i < j < k} \left(\alpha^{2^j} - \alpha^{2^i}\right) = \prod_{0 \leq i < j < k} \left(\alpha^{2^j} + \alpha^{2^i}\right). \tag{12}$$

Computing $|\mathbf{M}|^2$, and applying Lemma 3 gives

$$|\mathbf{M}|^2 = \left[\prod_{0 \leq i < j < k} \left(\alpha^{2^j} + \alpha^{2^i}\right)\right]^2 = \prod_{0 \leq i < j < k} \left(\alpha^{2^{j+1}} + \alpha^{2^{i+1}}\right). \tag{13}$$

When $j = k-1$, the expression $(\alpha^{2^{j+1}} + \alpha^{2^{i+1}})$ equals $(\alpha^{2^k} + \alpha^{2^{i+1}})$. Since $\alpha^{2^k} = \alpha$ over $\mathbb{F}_{2^k}$, this reduces to $(\alpha^{2^{i+1}} + \alpha)$. This results in the property that $|\mathbf{M}|^2 = |\mathbf{M}|$. Since $|\mathbf{M}| \in \mathbb{F}_{2^k}$, the only two elements of $\mathbb{F}_{2^k}$ that satisfy $|\mathbf{M}|^2 = |\mathbf{M}|$ are 0 and 1. As already shown $|\mathbf{M}| \neq 0$, it follows that $|\mathbf{M}| = 1$. ■

Applying Lemma 4 to (9) gives the expression for a_i in terms of A: i.e., $a_i = |\mathbf{M_i}|$, where the determinant $|\mathbf{M_i}|$ can be computed symbolically. The polynomials $a_i = \mathcal{F}_{a_i}(A) = |\mathbf{M_i}|$ are independent of the circuit that is given for abstraction, and their form/size depends upon the operand width k of the circuit (or the corresponding field $\mathbb{F}_{2^k}$). Since $\mathbf{M_i}$ is of the form

$$\mathbf{M_i} = \begin{bmatrix} 1 & \alpha & \dots & \alpha^{i-1} & A & \alpha^{i+1} & \dots & \alpha^{k-1} \\ 1 & \alpha^2 & \dots & \alpha^{(i-1)\cdot 2} & A^2 & \alpha^{(i+1)\cdot 2} & \dots & \alpha^{(k-1)\cdot 2} \\ \vdots & \vdots & . & \vdots & \vdots & \vdots & . & \vdots \\ 1 & \alpha^{2^{k-1}} & \dots & \alpha^{(i-1)\cdot 2^{k-1}} & A^{2^{k-1}} & \alpha^{(i+1)\cdot 2^{k-1}} & \dots & \alpha^{(k-1)\cdot 2^{k-1}} \end{bmatrix}$$

and $|\mathbf{M}_i|$ is computed by Laplace expansion along the ith column, it follows that computing the determinant $|\mathbf{M_i}|$ results in polynomials of the form $a_i = c_{i_0}A + c_{i_1}A^2 + c_{i_2}A^4 + \cdots + c_{i_{k-1}}A^{2^{k-1}}$, where $\{c_{i_0}, \dots, c_{i_{k-1}}\} \in \mathbb{F}_{2^k}$.

B. Overall Abstraction Approach

Based on the above concepts, the word-level abstraction approach for a circuit with k-bit input A and k-bit output Z is described as follows.

1) Given a combinational circuit C, with word-level k-bit input A and output Z. Choose a primitive polynomial $P(x)$ of degree k and construct $\mathbb{F}_{2^k}$, and let $P(\alpha) = 0$.
2) Perform a reverse-topological traversal of C to derive RATO: lex with $\{x_1 > x_2 > \cdots > x_d > Z > A\}$, where $\{x_1, \dots, x_d\}$ are bit-level variables of C.
3) Derive the set of bit-level polynomials $\{f_1, \dots, f_s\}$ from each gate in C, and represent them using RATO. These will be in the form $f_i : x_i + \text{tail}(f_i)$ where x_i is the output of the corresponding logic gate.
4) Compose the bit-level to word-level polynomial correspondences: $f_A : a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1} + A; f_Z : z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1} + Z$. Denote $F = \{f_1, \dots, f_s, f_A, f_Z\}$. Compose the set of vanishing polynomials $F_0 = \{x_i^2 + x_i, A^{2^k} + A, Z^{2^k} + A\}$.
5) Select the only critical pair (f_Z, f_{z_i}) in F that does not have relatively prime leading terms. Compute $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_+ r$.
6) Construct matrices $\mathbf{M_0}, \dots, \mathbf{M_{k-1}}$, where $\mathbf{M_i}$ is $\mathbf{M}$ with the column $[\alpha^i, \alpha^{i\cdot 2}, \dots, \alpha^{i\cdot 2^{k-1}}]^T$ replaced by $[A, A^2, \dots, A^{2^{k-1}}]^T$, and $\mathbf{M} = \mathbf{V}(\alpha, \alpha^2, \dots, \alpha^{2^{k-1}})$.
7) Symbolically compute the determinants $|\mathbf{M_i}|$ to find $F_a = \{f_{a_0}, \dots, f_{a_{k-1}}\}$, where $f_{a_i} : a_i + |\mathbf{M_i}|$, for $0 \leq i \leq k-1$. Since this computation is independent of the reduction $\text{Spoly}(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_+ r$, it can be performed in parallel with step 5.
8) Compute $r \xrightarrow{F_a \cup F_0}_+ r_w$. Then r_w is of the form $Z + \mathcal{F}(A)$ and it is the unique, canonical word-level abstraction of C over $\mathbb{F}_{2^k}$.

Example 6: We demonstrate the application of our approach on the circuit of Fig. 4. In Example 5, we have already shown that by imposing RATO and performing the reduction $\text{Spoly}(f_Z, f_6) \xrightarrow{F, F_0}_+ r$ gives

$$r = (\alpha+1)a_1b_1 + (\alpha+1)a_1B + (\alpha+1)b_1A + Z + (\alpha+1)AB.$$

Since r contains the bit-level variable a_1, b_1, find $f_{a_1} : a_1 + |\mathbf{M_1}|$. In this example, $f_A : a_0 + a_1\alpha + A$, so

$$\mathbf{M} = \begin{bmatrix} 1 & \alpha \\ 1 & \alpha^2 \end{bmatrix} \quad \mathbf{M_1} = \begin{bmatrix} 1 & A \\ 1 & A^2 \end{bmatrix}. \tag{14}$$

Computing $|\mathbf{M_1}|$ finds $f_{a_1} : a_1 + A^2 + A$. Similarly, $f_{b_1} : b_1 + B^2 + B$. Now, computing $r \xrightarrow{\{f_{a_1}, f_{b_1}\} \cup F_0}_+ r_w$ finds $\boldsymbol{r_w = Z + (\alpha+1)A^2B^2}$, which is the UCR of the circuit, the same as the one derived by computing a full reduced Gröbner basis in Example 3.

1) Complexity of Our Approach: Given d variables in the polynomial ring with coefficients in $\mathbb{F}_{2^k}$, we have shown that: 1) the worst-case complexity of step 5) is $O(2^{2d})$; 2) the complexity of step 7) is polynomial in k; and 3) the complexity of step 8) is $O(2^{3k})$. Steps 5) and 7) are independent and these are computed in parallel. Moreover, since $d >> k$, the complexity of step 5) subsumes that of step 7). So the worst-case complexity of the abstraction algorithm is $O(2^{2d}) + O(2^{3k})$ (see [27, Th. 6.3]).

VI. Experimental Results

The approach described in Section V can be can be scripted within a computer algebra tool, e.g., SINGULAR [25]. However, the data-structures of such general-purpose computer algebra tools are not specifically geared toward circuit verification problems; this makes the polynomial abstraction for large circuits infeasible. For this reason, we have developed our own custom abstraction tool for circuits, written in C++.

Our word-level abstraction tool is based on fast, efficient polynomial operations over the rings of the type $\mathbb{F}_{2^k}[x_1, \dots, x_d]$. We developed a Galois field library that forms the backbone of the tool. It facilitates the construction and manipulation of elements of the corresponding field $\mathbb{F}_{2^k}$. A monomial M over $\mathbb{F}_{2^k}[x_1, \dots, x_d]$ is represented as a power-product in variables $x_1, \dots, x_d$ with a coefficient $C \in \mathbb{F}_{2^k}$, $M = C \cdot {x_1}^{e_1} \cdot {x_2}^{e_2} \cdots {x_d}^{e_d}$; $e_i \geq 0$. Ring variables are either bit-level or word-level. If x_i is bit-level, then $x_i^2 = x_i$, so its exponent $e_i \in \{0, 1\}$. For word-level variables, $e_i < 2^k$, due to $x_i^{2^k} = x_i$. These degree-reductions are performed after every monomial operation. Since RATO is a lex based ordering, lex is currently the only ordering implemented in the tool. Large exponents are stored as a BIGUNSIGNED object of the open source library BigInt [28], which provides basic functionality for integers of unbounded size. A polynomial object is simply an ordered vector of monomial objects. Additions and multiplications of polynomials are performed over their respective monomials. Finally, polynomial division $f \xrightarrow{F \cup F_0}_+ r$ is implemented as Faugére's $F4$-style reduction [8].

TABLE I
ABSTRACTION OF FLAT MONTGOMERY MULTIPLIERS. TIME GIVEN IN SECONDS AND MEMORY GIVEN IN MB. TO = 3 DAYS (259 200 s)

Field Size (k)		163	233	283	409	571
# of Gates		184K	329K	488K	1.0M	1.97M
Time	Bug Free	6,897	63,805	TO	TO	TO
	Buggy	6,961	64,009	TO	TO	TO
Max Memory		153	325	505	971	2,240

TABLE II
ABSTRACTION OF MONTGOMERY BLOCKS. TIME GIVEN IN SECONDS AND MEMORY IS GIVEN IN MB. TO = 3 DAYS (259 200 s)

Field Size (k)			163	233	283	409	571
# of Gates		Blk A	33K	55K	82K	168K	330K
		Blk B	33K	55K	82K	168K	330K
		Blk Mid	85K	163K	241K	502K	980K
		Blk Out	32K	54K	81K	168K	328K
Time	Bug Free	Blk A	25	142	330	1,322	5,371
		Blk B	25	141	329	1,335	5,241
		Blk Mid	73	408	883	4,471	19,942
		Blk Out	24	140	321	1,338	5,532
	Buggy	Blk A	26	142	331	1,323	5,372
		Blk B	26	141	330	1,336	5,421
		Blk Mid	111	580	1,411	6,829	37,804
		Blk Out	25	141	322	1,339	5,539
Max Mem Per Blk			80	168	254	538	1,129

A. Results

All experiments are conducted on a 64-bit Linux desktop with a 3.5 GHz Intel Core i7 Quad-core CPU and 16 GB of RAM. Table I depicts the time and memory required to derive the polynomial abstraction from bug-free and buggy Montgomery multiplier circuits using our custom tool. These circuits are provided as bit-blasted/flattened gate-level netlists. They compute $Z = A \cdot B$ over $\mathbb{F}_{2^k}$ for a given $k = 163, \ldots, 571$ corresponding to the NIST ECC specification. Unlike as shown in Fig. 1, in this experiment the design hierarchy is not made available to the abstraction tool. Abstraction is feasible for the 233-bit field, beyond which the reduction times-out. However, if the hierarchy is known, it can be exploited by computing the abstraction of each MR block in parallel, as shown in Table II. In this table, "BLK A" and "B" denote the input MR blocks, "BLK Mid" denotes the middle block and "BLK Out" is the output block. First, a polynomial is extracted for each MR block (gate-level to word-level abstraction), and then the approach is reapplied at word-level to derive the input–output relation (solved trivially in < 1 s). We can abstract the word-level polynomial for up to 571-bit (all NIST-specified ECC fields) circuits for both buggy and bug-free implementations. In these experiments, bugs are introduced in the middle MR block, as its abstraction is the most compute intensive.

We have also performed experiments with abstraction and verification of composite field multipliers for up to $k = 1024$ bits, where the field $\mathbb{F}_{2^k}$ is decomposed as $\mathbb{F}_{(2^m)^n}$. For these experiments, the decomposition hierarchy is made available to the tool. Similar to the design shown in Fig. 2, the input $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$ is transformed into multiple word-level inputs over $\mathbb{F}_{2^m}$, $A_0, A_1, \ldots, A_{n-1}$, where each $A_i = a_{i0} + a_{i1}\beta + \cdots + a_{i(m-1)}\beta^{m-1}$ and β is the primitive element of $\mathbb{F}_{2^m}$. The circuit is then composed of m-bit ADD, MULT blocks.

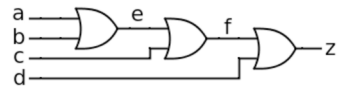

Fig. 6. Polynomial division leading to term explosion.

First, the abstraction for each of these blocks is performed independently. Once these m-bit word-level polynomials are known for the blocks, the final abstraction $Z = \mathcal{F}(A, B)$ is performed. For this, a functional mapping from each A_i to A is derived in the form $A_i = \mathcal{F}_i(A)$. This is also computed similar to the concepts shown in (7) and (8) in Section V; the property $A_i^{2^m} = A_i$ is utilized instead of $a_i^2 = a_i$ as the base field is $\mathbb{F}_{2^m}$ in this case. The results of this final word-level abstraction of buggy and bug-free multipliers over composite fields are shown in Table III. In these experiments, bugs are introduced in the high-level interconnections of the $\mathbb{F}_{2^m}$ blocks, and the $\mathbb{F}_{2^m}$ blocks themselves are kept bug-free.

Another experiment is conducted to evaluate the efficacy of our abstraction procedure when the (ADD, MULT) building-blocks are instantiated in a larger system. To this end, we perform abstraction for ECC point-addition circuits. Using $\mathbb{F}_{2^k}$ Mastrovito multipliers and adders, circuits were designed for ECC point-addition [see (2)]; these polynomials were synthesized/optimized and then abstraction for each word-level output X_3, Y_3, and Z_3 was performed in terms of word-level inputs X_1, Y_1, Z_1, X_2, and Y_2. The results for this abstraction are shown in Table IV, where abstraction is successful for up to 409-bit fields, but infeasible for the 571-bit circuit.

B. Limitations of the Approach

Our approach performs very well for $\mathbb{F}_{2^k}$ Galois field circuits. The design of these circuits is based on AND-XOR logic, where "chains of XOR gates" are often encountered. Our experience shows that the polynomials derived during the reduction procedures are sparse and do not explode. However, for random logic circuits, especially logic containing chains of OR gates, the computations explode and exhibit the worst-case behavior, due to which this technique is not very efficient for abstraction of random logic circuits.

Example 7: Consider the circuit of Fig. 6, apply RATO: lex with $z > f > d > e > c > b > a$. Reduce z modulo the polynomials corresponding to the circuit

$$f_1 : z + fd + f + d \quad f_2 : f + ec + e + c \quad f_3 : e + ba + b + a.$$

The reduction procedure, $z \xrightarrow{f_1, f_2, f_3}_+ r$ is computed as: $z \xrightarrow{z+fd+f+d} fd + f + d \xrightarrow{f+ec+e+c}_+ edc + ed + ec + e + dc + d + c \xrightarrow{e+ba+b+a}_+ dcba + dcb + dca + dba + dc + db + da + d + cba + cb + ca + c + ba + b + a = r$. The size of r corresponds to that of a worst-case representation of a Boolean polynomial in four variables. Each one-step reduction removes the output variable of the gate, but replaces it with two instances of each input variable. This increases the density of the resulting polynomial exponentially.

TABLE III
ABSTRACTION OF COMPOSITE FIELD MULTIPLIERS OVER $\mathbb{F}_{(2^m)^n}$. TIME GIVEN IN SECONDS AND MEMORY IN MB

$k = 128$					$k = 256$					$k = 512$					$k = 1024$				
m	n	Time		Max Mem	m	n	Time		Max Mem	m	n	Time		Max Mem	m	n	Time		Max Mem
		Bug Free	Buggy				Bug Free	Buggy				Bug Free	Buggy				Bug Free	Buggy	
2	64	1	1	4	2	128	15	15	23	2	256	406	408	90	2	512	11,883	12,050	414
4	32	1	1	2	4	64	2	2	4	4	128	53	53	25	4	256	1,520	1,536	106
8	16	1	1	2	8	32	1	1	3	8	64	8	8	4	8	128	209	211	29
16	8	1	1	2	16	16	1	1	2	16	32	2	2	4	16	64	38	37	10
32	4	1	1	2	32	8	1	1	2	32	16	1	1	3	32	32	10	10	5
64	2	1	1	3	64	4	1	1	2	64	8	1	1	3	64	16	4	4	3
-	-	-	-	-	128	2	1	1	2	128	4	1	1	2	128	8	2	2	3
-	-	-	-	-	-	-	-	-	-	256	2	1	1	2	256	4	1	1	3
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	512	2	1	1	3

TABLE IV
TIME AND MEMORY PER ABSTRACTION OF POINT ADDITION CIRCUITS. TIME GIVEN IN SECONDS AND MEMORY GIVEN IN MB. TO = 3 DAYS

Size (k)	233	283	409	571
Max Time (s)	2,831	10,325	109,777	TO
Max Memory per run (MB)	297	535	942	-

VII. CONCLUSION

This paper has described an approach to derive a word-level canonical polynomial representation from a combinational circuit using algebraic geometry and symbolic computation. Given a circuit C with k-bit inputs and outputs, we interpret the function $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$ as a polynomial function $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$. We prove that by computing a reduced Gröbner basis of the ideal generated by the polynomials of the circuit, an input–output relationship can be derived for the circuit as $Z = \mathcal{F}(A)$, where $Z = \{z_0, \ldots, z_{k-1}\}$, $A = \{a_0, \ldots, a_{k-1}\}$ are the k-bit outputs and inputs, respectively. The approach is then generalized to circuits with an arbitrary number of inputs (m) and outputs (n), by modeling the function over $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$, where $k = \text{LCM}(m, n)$. Subsequently, by analyzing the circuit's topology, an efficient symbolic computation is engineered that obviates the need to compute a reduced Gröbner basis. This enables the abstraction and verification of large (up to 1024-bit) Galois field arithmetic circuits, whereas prior approaches are practically infeasible. The complexity of our approach is also analyzed. As part of future work, we are investigating the application of our abstraction approach to verification of sequential Galois field arithmetic circuits, and also to verification over finite integer rings $\mathbb{Z}_{2^k} = \mathbb{Z} \pmod{2^k}$.

REFERENCES

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

[2] R. Lidl and H. Niederreiter, *Finite Fields*. Cambridge, U.K.: Cambridge Univ. Press, 1997.

[3] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 856–861, Jul. 1996.

[4] C. Koc and T. Acar, "Montgomery multiplication in GF(2^k)," *Designs Codes Cryptogr.*, vol. 14, no. 1, pp. 57–69, Apr. 1998.

[5] A. Lvov *et al.*, "Verification of Galois field based circuits by formal reasoning based on computational algebraic geometry," *Formal Methods Syst. Design*, vol. 45, no. 2, pp. 189–212, Oct. 2014.

[6] E. Biham, Y. Carmeli, and A. Shamir, "Bug attacks," in *Proc. Adv. Cryptol.*, Santa Barbara, CA, USA, 2008, pp. 221–240.

[7] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. Int. Conf. CAD (ICCAD)*, San Jose, CA, USA, 2006, pp. 836–843.

[8] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 9, pp. 1409–1420, Sep. 2013.

[9] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[10] M. Ciesielski, P. Kalla, and S. Askar, "Taylor expansion diagrams: A canonical representation for verification of data-flow designs," *IEEE Trans. Comput.*, vol. 55, no. 9, pp. 1188–1201, Sep. 2006.

[11] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence verification of polynomial datapaths using ideal membership testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1320–1330, Jul. 2007.

[12] B. Alizadeh and M. Fujita, "Modular datapath optimization and verification based on modular-HED," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 9, pp. 1422–1435, Sep. 2010.

[13] A. M. Jabir, D. K. Pradhan, T. L. Rajaprabhu, and A. K. Singh, "A technique for representing multiple output binary functions with applications to verification and simulation," *IEEE Trans. Comput.*, vol. 56, no. 8, pp. 1133–1145, Aug. 2007.

[14] F. Lu, L.-C. Wang, K.-T. Cheng, and R. C.-Y. Huang, "A circuit SAT solver with signal correlation guided learning," in *Proc. IEEE Design Autom. Test Europe*, Munich, Germany, 2003, pp. 892–897.

[15] S. Gao, "Counting zeros over finite fields with Gröbner bases," M.S. thesis, Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2009.

[16] S. Gao, A. Platzer, and E. M. Clarke, "Quantifier elimination over finite fields with Gröbner bases," in *Proc. Int. Conf. Algebr. Inform.*, Linz, Austria, 2011, pp. 140–157.

[17] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G.-M. Gruel, "An algebraic approach to proving data correctness in arithmetic datapaths," in *Proc. Comput.-Aided Verificat. Conf.*, Princeton, NJ, USA, 2008, pp. 473–486.

[18] M. Ciesielski, W. Brown, D. Liu, and A. Rossi, "Function extraction from arithmetic bit-level circuits," in *Proc. IEEE Int. Symp. VLSI (ISVLSI)*, Tampa, FL, USA, 2014, pp. 356–361.

[19] M. Ciesielski, C. Yu, D. Liu, W. Brown, and A. Rossi, "Verification of gate-level arithmetic circuits by function extraction," in *Proc. Design. Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.

[20] Z. Zilic and Z. G. Vranesic, "A deterministic multivariate interpolation algorithm for small finite fields," *IEEE Trans. Comput.*, vol. 51, no. 9, pp. 1100–1105, Sep. 2002.

[21] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in GF(2^n)," in *Proc. Int. Workshop Select. Areas Cryptogr.*, Kingston, ON, Canada, 1998, pp. 201–212.

[22] E. D. Mastrovito, "VLSI designs for multiplication over finite fields GF(2^m)," in *Applied Algebra, Algebraic Algorithms Error-Correcting Codes* (LNCS 357). Berlin, Germany: Springer, 1989, pp. 297–309.

[23] B. Buchberger, "Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal," Ph.D. dissertation, Math. Inst., Univ. Innsbruck, Innsbruck, Austria, 1965.

[24] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*. Boston, MA, USA: Kluwer Academic, 1987.

[25] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. (2011). *Singular 3-1-3—A Computer Algebra System for Polynomial Computations*. [Online]. Available: http://www.singular.uni-kl.de
[26] B. Buchberger, "A criterion for detecting unnecessary reductions in the construction of a Groebner bases," in *Proc. Int. Symp. Symb. Algebr. Manipulat.*, Marseille, France, 1979, pp. 3–21.
[27] T. Pruss, "Word-level abstraction from combinational circuits using algebraic geometry," Ph.D. dissertation, Dept. Electr. Comput. Eng., Univ. Utah, Salt Lake City, UT, USA, May 2015.
[28] M. McCutchen. (2010). *C++ Big Integer Library*. [Online]. Available: https://mattmccutchen.net/bigint/

Tim Pruss received the B.S. degree in computer engineering and the Ph.D. degree in electrical and computer engineering from the University of Utah, Salt Lake City, UT, USA, in 2009 and 2015, respectively.

He is a Formal Verification Engineer with Apple Computer, Cupertino, CA, USA.

Priyank Kalla (SM'13) received the B.E. degree in electronics engineering from Saradar Patel University, Vallabh Vidyanagar, India, in 1993, and the M.S. and Ph.D. degrees from the University of Massachusetts Amherst, Amherst, MA, USA, in 1998 and 2002, respectively.

He has been a Faculty Member with the Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT, USA, since 2002, where he is currently an Associate Professor. His current research interests include computer-aided design techniques for synthesis, optimization, and verification of digital very large-scale integration circuits and systems.

Prof. Kalla was a recipient of the National Science Foundation CAREER Award and the ACM Transactions on Design Automation of Electronic Systems 2009 Best Paper Award. He was the Chair of the IEEE High-Level Design Validation and Test Workshop 2009 and the IEEE Circuits and Systems Technical Committee on Computer-Aided Network Design 2011 and 2012.

Florian Enescu received the B.S. degree from the University of Bucharest, Bucharest, Romania, in 1996, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 1999 and 2001, respectively, all in mathematics.

From 2001 to 2004, he was a Scott Assistant Professor of Mathematics with the University of Utah, Salt Lake City, UT, USA. In 2002, he was a Post-Doctoral Fellow with the Mathematical Sciences Research Institute, Berkeley, CA, USA. Since 2004, he has been with the Department of Mathematics and Statistics, Georgia State University, Atlanta, GA, USA, where he is currently a Professor of Mathematics. His current research interests include commutative algebra, primarily positive characteristic methods in ring theory.

Prof. Enescu is a member of the American Mathematical Society.