Word-Level Abstraction from Bit-Level Circuits using Gröbner Bases

Tim Pruss*, Priyank Kalla*, Florian Enescu†
*Electrical & Computer Engineering, University of Utah
†Mathematics & Statistics, Georgia State University

***Abstract*—A combinational circuit with k-inputs and k-outputs implements Boolean functions $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$, where $\mathbb{B} = \{0,1\}$. The function can also be construed as a mapping $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$, where $\mathbb{F}_{2^k}$ denotes a Galois field of 2^k elements. Every function over $\mathbb{F}_{2^k}$ is a polynomial function — i.e. there exists a unique, minimal, canonical polynomial $\mathcal{F}$ that describes f. This paper presents novel techniques based on computer-algebra and algebraic-geometry to derive the canonical (word-level) polynomial representation of the circuit as $Y = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$, where A and Y denote, respectively, the input and output bit-vectors of the circuit. We show that this can be achieved by computing a Gröbner basis of a set of polynomials derived from the circuit, using a specific elimination term order. Our approach can be generalized to circuits with arbitrary number of inputs and outputs, as polyfunctions $f : \mathbb{F}_{2^k}^d \rightarrow \mathbb{F}_{2^k}$ or $f : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^m}$.**

Computing Gröbner bases using elimination orders is, however, practically infeasible for large circuits. To overcome this limitation, we present improvements to the computation based on: i) term orders derived from the topological traversal of the given circuit; and ii) the use of FGLM algorithm from algebraic geometry. We apply our approach to “reverse-engineer” hardware implementations of Elliptic Curve Cryptography (ECC) primitives over Galois fields $\mathbb{F}_{2^k}$ — with applications to design verification and function identification.

I. Introduction

Abstraction of word-level functionality from bit-level descriptions of digital circuits is an important fundamental problem in Electronic Design Automation (EDA) and design verification. Word-level abstractions find applications in high-level/RTL datapath synthesis [1], RTL verification [2] [3], word-level SMT and constraint solving [4] [5], etc. Functional abstraction can also be helpful in detecting malicious modifications to a circuit – potentially inserted as hardware Trojan horses – by *reverse-engineering* the function implemented by the circuit[1]. It is further desirable that the obtained word-level abstraction be a *canonical* representation of the function, to facilitate equivalence verification between a specification model and an implementation circuit.

This paper presents a symbolic method to derive the *word-level, canonical, polynomial representation* from a given combinational circuit. Our approach models the problem over Galois fields, and employs concepts from commutative algebra and algebraic geometry — notably, Gröbner bases [7] theory and technology — to derive the word-level abstraction.

The main motivation for this work is to “reverse engineer” (or to identify) the function implemented by a given Galois field arithmetic circuit as used in Elliptic Curve Cryptography (ECC). The main operations of encryption, decryption and authentication in ECC rely on point-addition and point-doubling operations on elliptic curves defined over Galois fields $\mathbb{F}_{2^k}$. These primitive operations are, in turn, implemented as circuits that perform polynomial computations over k-bit vectors [8]. The objective is to apply our approach to a given circuit and *extract the word-level polynomial function (polyfunction)* implemented by it for design verification. While our approach can be applied to any arbitrary combinational circuit, it is most beneficial for arithmetic circuits where canonical Boolean representations prove to be infeasible.

Problem Statement: Given a combinational circuit C with k-bit inputs and k-bit outputs, such that the circuit implements Boolean functions that are mappings between k-dimensional Boolean spaces: $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$. Let $\{a_0, \dots, a_{k-1}\}$ denote the primary inputs of the circuit, and let $\{y_0, \dots, y_{k-1}\}$ denote the bit-level primary outputs. Suppose that we do not know the function implemented by the circuit. We wish to derive a *word-level, symbolic representation* of the function as $Y = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$, where $Y = \{y_0, \dots, y_{k-1}\}$, $A = \{a_0, \dots, a_{k-1}\}$ are, respectively, the output and input bit-vectors (words), and $\mathcal{F}$ denotes a polynomial representation of the function f. We wish to further generalize our approach to any circuit with arbitrary number of inputs (n) and outputs (m), as polyfunctions $f : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^m}$.

Approach: Canonical symbolic DAG representations of Boolean functions, such as ROBDDs [9], FDDs [10], BMDs [11], etc., are based on (variants of) the Shannon's expansion, which is a bit-wise, Boolean decomposition. These are inapplicable to word-level abstraction of modulo-arithmetic circuits over Galois extension fields $\mathbb{F}_{2^k}$. Therefore, we approach this problem from the perspective of *symbolic computer algebra and algebraic geometry*.

The function $f : \mathbb{B}^k \rightarrow \mathbb{B}^k$ is a mapping among 2^k elements. Therefore, f can also be interpreted, algebraically, in the following two ways: i) as a function $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$, i.e. as a function over finite integer rings $\pmod{2^k}$; and ii) as a function $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$, where $\mathbb{F}_{2^k}$ represents the Galois field of 2^k elements. In this work, we will analyze f as a polynomial function over $\mathbb{F}_{2^k}$, for the following reasons.

First of all, *not every function* of the type $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$, is a polynomial function; i.e. every function cannot be represented by a polynomial $\mathcal{F} \pmod{2^k}$. In commutative algebra, identification of such polynomial functions is an important problem: given a function $f : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$, $n \in \mathbb{N}$, identify if f has a polynomial representation. If so, derive a unique, minimal, canonical polynomial $\mathcal{F}$, such that $f \equiv \mathcal{F} \pmod{n}$. The work of [12] addresses such problems in number theory and algebra. *Shekhar et al.* [13] address RTL verification of polynomial datapaths over k-bit vectors using the above concepts. However, in their work, the RTL datapath is already known to be a polynomial function $\pmod{2^k}$. Unfortunately, an arbitrary k-input/k-output combinational circuit cannot always be modeled

[1]A hardware Trojan modifies the functionality of the circuit. Abstraction of the function at word-level can perhaps give some insights into the functional *intent* of the Trojan. This can also be helpful in Trojan classification; such efforts are currently underway within the EDA community [6].

as a polynomial function over $\mathbb{Z}_{2^k}$; therefore, the $\mathbb{Z}_{2^k}$ model is not considered in this work.

On the other hand, there is a well-known "textbook" result [14] which states that over Galois fields ($\mathbb{F}_q$) of q elements, *every function* $f : \mathbb{F}_q \to \mathbb{F}_q$ is a polynomial function. Motivated by this fundamental result, we devise an approach to derive a word-level polynomial function abstraction from the circuit using the $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ polyfunction model. We analyze the circuit, derive a specific set of polynomials modeled over $\mathbb{F}_{2^k}$, and compute a *Gröbner basis* of these polynomials with a specific term order to abstract the canonical polynomial representation. Our approach can be generalized to a circuit with arbitrary number of inputs and outputs, i.e. to model polynomial functions $f : \mathbb{F}_{2^k}^d \to \mathbb{F}_{2^k}$, and further to $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$.

Approach & Contributions:

1) Using polynomial abstractions, we analyze the circuit, and model the gate-level Boolean operators as elements of a multivariate polynomial ring with coefficients in $\mathbb{F}_{2^k}$.
2) Based on the concepts of *Strong Nullstellensatz, Gröbner bases, elimination ideals and projections of varieties* [7], we deduce that the polynomial abstraction problem can be formulated as one of *computing a Gröbner basis* of a set of polynomials derived from the given circuit netlist, using a specific elimination term order $>$.
3) Computing Gröbner bases using elimination term orders is infeasible for large circuits. To overcome this limitation, we investigate the use of the FGLM algorithm [15] to derive the polynomial representation. The FGLM algorithm takes an *already computed* Gröbner basis (G_1) w.r.t. an arbitrary term order $>_1$, and transforms it into another Gröbner basis (G) w.r.t. an elimination term order $>$. In the work of *Lv* [16] it was shown that for any given circuit, there exists a term order $>_1$ that renders the set of polynomials derived from the circuit itself a Gröbner basis. Moreover, $>_1$ can be easily derived by performing a reverse topological traversal of the circuit. Consequently, G_1 is readily available as a Gröbner basis directly by construction. Using FGLM, we can then translate G_1 into the Gröbner basis G w.r.t. the required elimination order $>$.
4) The worst-case complexity of computing a Gröbner basis is known to be doubly exponential in n, the number of variables, and polynomial in d, the degree of the ideal. Most hardware/software verification applications do exhibit such high complexity, as the number of intermediate computations in the Gröbner basis algorithm (Buchbeger's algorithm) [17] tend to explode. Therefore, we conjecture that using the FGLM algorithm — which relies on *sparse linear algebra concepts* — perhaps this complexity can be avoided. We are currently implementing a domain specific FGLM algorithm — specifically for the abstraction problems related to Galois field crypto-circuits. Experimental results with FGLM are not yet available.

Paper Organization: The next section reviews previous work in the area of canonical representations functions, word-level abstractions, and also polynomial interpolation literature in symbolic computing. Section III reviews Galois field theory and describes the arithmetic circuits that we wish to verify using our approach. Section IV reviews preliminary computer-algebra concepts related to ideals, varieties, Gröbner bases, elimination ideals and Nullstellensatz. Section V describes our results on polynomial abstraction from circuits, and demonstrates the application of our work using preliminary experiments. The application of the FGLM algorithm is covered in Section VI. Finally, Section VII concludes the proposal.

II. Review of Previous Work

Canonical Decision Diagrams: The Reduced Ordered Binary Decision Diagram (ROBBD) [9], based on the Shannon's decomposition, was the first significant contribution in this area. Motivated by the success of BDDs, variants of the Shannon's decomposition principle (Davio, Reed-Muller, etc.) were explored to develop other functional decision diagrams: FDDs [10], ADDs [18], HDDs [19] and EVBDDs [20]. Binary Moment Diagrams (BMDs) [11], and its derivative K*BMDs [21], depart from the Boolean decomposition and perform the decomposition of a *linear* function based on its two moments. The decomposition is still point-wise, binary, w.r.t. each Boolean variable; these do not serve the purpose of word-level abstraction from bit-level representations.

Taylor Expansion Diagrams (TEDs) [22] are a word-level canonical representation of a *polynomial expression*, but they do not represent a *polynomial function* canonically. While [13] and [23] provide canonical representations of polynomial functions, they do so over $\mathbb{Z}_{2^k}$ and not over $\mathbb{F}_{2^k}$.

MODDs [24] [25] are a DAG representation of the characteristic function of a circuit over Galois fields $\mathbb{F}_{2^k}$. MODDs come very close to satisfying our requirements as a canonical word-level representation that can be employed over Galois fields, as it essentially interpolates a polynomial from the characteristic function. However, MODDs do not scale very well for large circuits — this is because every node in the DAG can have up to k children and the normalization operations are very complicated for MODDs. They suffer from the size explosion problem during intermediate computations. They are known to be infeasible beyond 32-bit circuits.

Verification of Galois field circuits: In [26], the authors present the BLUEVERI tool from IBM for verification of Galois field circuits for error correcting codes against an algorithmic spec. The implementation consists of a set of (pre-designed 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 formulation. For final verification, the polynomial system is given to a computer algebra tool (SINGULAR [27]) to *compute* a reduced Gröbner basis. However, improvements to the core Gröbner basis computational engine are not the subject of their work.

In [16] *Lv et al.* present computer algebra techniques for formal verification of Galois field arithmetic circuits. Given a specification polynomial f, and a circuit C, they formulate the verification problems as an ideal membership test using the

Strong Nullstellensatz and Gröbner bases. In [16], the authors show that for any combinational circuit, *there exists a term order $>_1$ that renders the set of polynomials of the circuit itself a Gröbner basis — and this term order can be easily derived by performing a topological traversal of the circuit*. By exploiting this term order, verification can be significantly scaled to 163-bit (NIST-specified) cryptography circuits. In contrast to the work of [16], we are not given a specification polynomial. Given the circuit C, we want to derive (extract) the word-level specification f. In this work, we build upon the results of [16].

Polynomial Interpolation in Symbolic Computation: The problem of polynomial interpolation is a fundamental problem in symbolic and algebraic computing which finds application in modular algorithms, such as the GCD computation and polynomial factorization. The problem is stated as follows: Given n distinct data points $x_1, \dots, x_n$, and their evaluations at these points $y_1, \dots, y_n$, *interpolate* a polynomial $\mathcal{F}(X)$ of degree $n-1$ (or less) such that $\mathcal{F}(x_i) = y_i$ for $1 \leq i \leq n$. Let t be the number of non-zero terms in $\mathcal{F}$ and let T be the total number of possible terms. When $\frac{t}{T} << 1$, the polynomial $\mathcal{F}$ is *sparse*, otherwise it is *dense*. Much of the work in polynomial interpolation addresses sparse interpolation using the "black-box" model (also called the algebraic circuit model) as shown in Fig. 1.

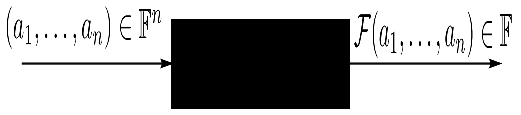
$(a_1, \dots, a_n) \in \mathbb{F}^n$
$\mathcal{F}(a_1, \dots, a_n) \in \mathbb{F}$

Fig. 1: The black-box or the algebraic circuit representation.

Let $\mathcal{F}$ be a multivariate polynomial in n variables $\{x_1, \dots, x_n\}$, with t non-zero terms $(0 < t < T)$, represented with a black-box B. On input $(x_1, \dots, x_n)$, the black-box evaluates $y_i = \mathcal{F}(x_1, \dots, x_n)$. Given also a degree bound d on $\mathcal{F}$, the goal is to interpolate the polynomial $\mathcal{F}$ with a minimum number of *probes* to the black-box. The early work of Zippel [28] and Ben-Or/Tiwari [29] require $O(ndt)$ and $O(T \log n)$ probes, respectively, to the black-box. These bounds have since been improved significantly; the recent algorithm of [30] interpolates with $O(nt)$ probes.

Our problem falls into the category of *dense* interpolation, as we require a polynomial that describes the function at each of the q points of the field $\mathbb{F}_q$. Newton's dense interpolation technique, with the black-box model, bounds the number of probes by $(d+1)^n$ — which exhibits very high complexity. In the logic synthesis area, the work of [31] investigates dense interpolation. Due to the inherent high-complexity, their approach is feasible only for applications over small fields, *e.g.* computing Reed-Muller forms for multi-valued logic.

We can also employ the black-box model by replacing the black-box (algebraic circuit) by the given circuit C; then every *probe* of the black-box would correspond to a *simulation of the circuit*. As we desire a polynomial representation of the entire function, exhaustive simulation would be required, which is infeasible. Therefore, we propose a *symbolic approach* to polynomial interpolation from a circuit using the Gröbner basis computation.

III. Galois Fields, Polynomial Functions & Hardware Design

A Galois field $\mathbb{F}_q$ is a field with a finite number of elements. The number of elements q of the field is a power of a prime integer — i.e. $q = p^k$, where p is a prime integer, and $k \geq 1$ [32]. We are interested in fields where $p = 2$ and $k > 1$ — i.e. *binary Galois extension fields* $\mathbb{F}_{2^k}$ — as they are widely employed in hardware implementations of cryptography primitives.

To construct $\mathbb{F}_{2^k}$, we take the polynomial ring $\mathbb{F}_2[x]$, where $\mathbb{F}_2 = \{0, 1\}$, and an irreducible polynomial $P(x) \in \mathbb{F}_2[x]$ of degree k, and construct $\mathbb{F}_{2^k}$ as $\mathbb{F}_2[x] \pmod{P(x)}$. As a result, all field operations are performed modulo the irreducible polynomial $P(x)$ and the coefficients are reduced modulo $p = 2$. Any element $A \in \mathbb{F}_{2^k}$ can be represented in polynomial form as $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$, where $a_i \in \mathbb{F}_2, i = 0, \dots, k-1$, and α is the root of the irreducible polynomial, *i.e.* $P(\alpha) = 0$.

Example 3.1: Let us construct $\mathbb{F}_{2^4}$ *as* $\mathbb{F}_2[x] \pmod{P(x)}$, *where* $P(x) = x^4 + x^3 + 1 \in \mathbb{F}_2[x]$ *is an irreducible polynomial of degree* $k = 4$. *Let* α *be the root of* $P(x)$, *i.e.* $P(\alpha) = 0$. *Any element* $A \in \mathbb{F}_2[x] \pmod{x^4 + x^3 + 1}$ *has a representation of the type:* $A = a_3x^3 + a_2x^2 + a_1x + a_0$ *where the coefficients* $a_3, \dots, a_0$ *are in* $\mathbb{F}_2 = \{0, 1\}$. *Since there are only 16 such polynomials, we obtain 16 elements in the field* $\mathbb{F}_{16}$. *Each element in can then be viewed as a 4-bit vector over* $\mathbb{F}_2$*:* $\mathbb{F}_{16} = \{(0000), (0001), \dots (1110), (1111)\}$. *Each element also has an exponential representation; all three representations are shown in Table I. For example, consider the element* α^{12}. *Computing* $\alpha^{12} \pmod{\alpha^4 + \alpha^3 + 1} = \alpha + 1 = (0011)$*; hence we have the three equivalent representations.*

TABLE I: Bit-vector, Exponential and Polynomial representation of elements in $\mathbb{F}_{2^4} = \mathbb{F}_2[x] \pmod{x^4 + x^3 + 1}$

$a_3a_2a_1a_0$	Exponential	Polynomial	$a_3a_2a_1a_0$	Exponential	Polynomial
0000	0	0	1000	α^3	α^3
0001	1	1	1001	α^4	α^3+1
0010	α	α	1010	α^{10}	$\alpha^3+\alpha$
0011	α^{12}	$\alpha+1$	1011	α^5	$\alpha^3+\alpha+1$
0100	α^2	α^2	1100	α^{14}	$\alpha^3+\alpha^2$
0101	α^9	α^2+1	1101	α^{11}	$\alpha^3+\alpha^2+1$
0110	α^{13}	$\alpha^2+\alpha$	1110	α^8	$\alpha^3+\alpha^2+\alpha$
0111	α^7	$\alpha^2+\alpha+1$	1111	α^6	$\alpha^3+\alpha^2+\alpha+1$

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 $f : \mathbb{B}^k \to \mathbb{B}^k$. Every such function is also a polynomial function over Galois fields: $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$.

Theorem 3.1: From [14]: 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 **Langrange's interpolation formula** and interpolate a polynomial $\mathcal{F}(x) = \sum_{k=1}^{q} \frac{\prod_{i \neq k}(x - x_i)}{\prod_{i \neq k}(x_k - x_i)} \cdot f(x_k)$, 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 function representation of f.

An important property of Galois 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 $\mathbb{F}_q$. Consequently, 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 $q-1$.

Definition 3.1: Any function $f : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ has a unique canonical representation (UCR) as polynomial $\mathcal{F} \in \mathbb{F}_q[x_1, \ldots, x_d]$ such that all its nonzero monomials are of the form $x_1^{i_1} \cdots x_d^{i_d}$ where $0 \le i_j \le q-1$, for all $j = 1, \ldots d$.

A. Hardware Implementations of Galois Field Arithmetic

Point Addition over Elliptic Curves: The main operations of encryption, decryption and authentication in elliptic curve cryptography (ECC) rely on *point additions* and *doubling* operations on elliptic curves designed over Galois fields. In general, this requires computation of multiplicative inverses over the field - which is expensive. Modern approaches represent the points in projective coordinate systems, *e.g.*, the López-Dahab (LD) projective coordinate [8], which eliminates the need for multiplicative inverses and improves the efficiency of these operations.

Example 3.2: Consider point doubling in LD projective coordinate system. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$. *Let* ($X_3$, Y_3, Z_3) = 2(X_1, Y_1, Z_1), *then* X_3, Y_3, Z_3 *can be computed as:* $X_3 = X_1^4 + b \cdot Z_1^4$; $Z_3 = X_1^2 \cdot Z_1^2$; $Y_3 = bZ_1^4 \cdot Z_3 + X_3 \cdot (aZ_3 + Y_1^2 + bZ_1^4)$.

The multiplication and iterative squaring operations in the above computation are usually implemented using custom-designed Galois field multipliers, such as the Mastrovito [33], Montgomery [34], Barrett multipliers [35], or composite-field multipliers [36] — which are, in turn, hierarchically designed. For example, Montgomery reduction (MR) computes: $MR(A,B) = A \cdot B \cdot R^{-1} \pmod{P(x)}$, where A, B are k-bit inputs, R is suitably chosen as $R = \alpha^k$, R^{-1} is multiplicative inverse of R in $\mathbb{F}_{2^k}$, and $P(x)$ is the irreducible polynomial. Since Montgomery reduction cannot directly compute $A \cdot B$ (mod $P(x)$), we need to pre-compute $A \cdot R$ and $B \cdot R$, as shown in Figure 2.

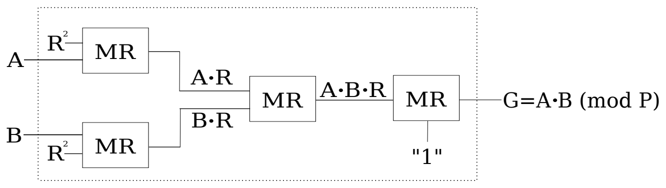

Fig. 2: *Montgomery* multiplication over $\mathbb{F}_{2^k}$ using four Montgomery reductions.

Given a hierarchically designed Montgomery multiplier, we will first extract polynomials AR, BR, ABR from the sub-circuit blocks. By analyzing the interconnection of these sub-circuits at word-level, we can then apply our approach at a higher-level, to extract the function of the entire circuit. Performing such operations hierarchically, we will apply our approach to reverse-engineer point-addition circuits designed using a variety of such Galois field adder and multiplier circuits.

IV. Computer Algebra Preliminaries

We denote a Galois field of q elements by $\mathbb{F}_q$, where $q = 2^k$ in our case. Let $\mathbb{F}_q[x_1, \ldots, x_d]$ be the polynomial ring over $\mathbb{F}_q$ with indeterminates $x_1, \ldots, x_d$. A *monomial* in variables $x_1, \cdots, x_d$ is a product of the form $X = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_d^{\alpha_d}$, where $\alpha_i \ge 0, i \in \{1, \ldots, d\}$. A *polynomial* $f \in \mathbb{F}_q[x_1, \ldots, x_d], f \neq 0$, 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 ordering* $>$ is imposed such that $X_1 > X_2 > \cdots > X_t$. Subject to such an ordering, $lt(f) = c_1X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term, leading monomial* and *leading coefficient* of f, respectively. Similarly, tail(f) = $c_2X_2 + \cdots + c_tX_t$. Division of a polynomial f by polynomial g gives remainder polynomial r, denoted $f \xrightarrow{g}_+ r$. Similarly, f can be reduced (divided) w.r.t. a set of polynomials $F = \{f_1, \ldots, f_s\}$ to obtain a remainder r, denoted $f \xrightarrow{F}_+ r$, such that no term in r is divisible by the leading term of any polynomial in F.

Ideals and varieties: An *ideal* J generated by polynomials $f_1, \ldots, f_s \in \mathbb{F}_q[x_1, \ldots, x_d]$ is: $J = \langle f_1, \ldots, f_s \rangle = \{\sum_{i=1}^s h_i \cdot f_i : \ h_i \in \mathbb{F}[x_1, \ldots, x_d]\}$. 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 $f(\mathbf{a}) = 0$. For any ideal $J = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$, the *affine variety* of J over $\mathbb{F}_q$ is: $V(J) = \{\mathbf{a} \in \mathbb{F}^d : \forall f \in J, f(\mathbf{a}) = 0\}$. In other words, the variety corresponds to the set of all solutions to $f_1 = \ldots f_s = 0$.

For any subset V of $\mathbb{F}_q^d$, the ideal of polynomials that vanish on V, called the *vanishing ideal of* V, is defined as: $I(V) = \{f \in \mathbb{F}_q[x_1, \ldots, x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$.

Proposition 4.1: If a polynomial f vanishes on a variety V, then $f \in I(V)$.

A. Strong Nullstellensatz over Galois Fields

Our problem formulation is derived from Nullstellensatz, which admits a special form over Galois fields. We state the following results of Nullstellensatz over Galois fields, proofs of which can be found in [37].

Let $\mathbb{F}_q$ be a Galois field of q elements. For all elements $A \in \mathbb{F}_q$, we have $A^q - A = 0$. Therefore, for a polynomial $x^q - x$, we have $V(x^q - x) = \mathbb{F}_q$. The polynomials of the form $\{x^q - x\}$ are called the *vanishing polynomials* of the field. Let $F_0 = \{x_1^q - x_1, \ldots, \ x_d^q - x_d\}$, then $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ is the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$. Below, we use the concept of sum of ideals: given ideals $I_1 = \langle f_1, \ldots, f_s \rangle$ and $I_2 = \langle g_1, \ldots, g_t \rangle$, then ideal $I_1 + I_2 = \langle f_1, \ldots, f_s, g_1, \ldots, g_t \rangle$.

Theorem 4.1: Strong Nullstellensatz over $\mathbb{F}_q$*:* For any Galois field $\mathbb{F}_q$, 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, $I(V_{\mathbb{F}_q}(J)) = J + J_0 = J + \langle x_1^q - x_1, \ldots, \ x_d^q - x_d \rangle$.

B. Gröbner Basis of Ideals

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 $V(J) = V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$. Some generating sets are "better" than others, i.e. they are a better representation of the ideal. A *Gröbner basis* is one such representation which allows to solve many polynomial decision questions.

Definition 4.1: [**Gröbner Basis**] [From [38]] 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 for $J \iff \forall f \in J,\ f \neq 0$, there exists $i \in \{1, \cdots, t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f)$.

Gröbner bases theory provides a *decision procedure to test for membership in an ideal*. As a consequence of Definition 4.1, the set G is a Gröbner basis of ideal J, if and only if for all $f \in J$, dividing f by polynomials of G gives 0 remainder: $G = GB(J) \iff \forall f \in J, f \xrightarrow{G}_+ 0$.

Buchberger's algorithm [17], 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 $G = \{g_1, \ldots, g_t\}$. In the algorithm,

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

where $L = \text{LCM}(lm(f), lm(g))$, where $lm(f)$ is the leading monomial of f, and $lt(f)$ is the leading term of f.

ALGORITHM 1: Buchberger's Algorithm

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

We now describe our word-level abstraction problem formulation using Strong Nullstellensatz over $\mathbb{F}_{2^k}$, and its solution using Gröbner bases and Buchberger's algorithm.

V. Word-Level Abstraction using Gröbner basis

We are given a circuit C with k-bit inputs and outputs that performs a polynomial computation $Y = \mathcal{F}(A)$ over $\mathbb{F}_q = \mathbb{F}_{2^k}$. Let $P(x)$ be the *given* irreducible or primitive polynomial used for field construction, and let α be its root, i.e. $P(\alpha) = 0$. Note that we do not know the polynomial representation $\mathcal{F}(A)$ and our objective is to identify (the coefficients of) $\mathcal{F}(A)$. Let $\{a_0, \ldots, a_{k-1}\}$ denote the primary inputs and let $\{y_0, \ldots, y_{k-1}\}$ be the primary outputs of C. Then, the word-level and bit-level correspondences are:

$$A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1};\ \ Y = y_0 + y_1\alpha + \cdots + y_{k-1}\alpha^{k-1}; \tag{1}$$

We analyze the circuit and model all the gate-level Boolean operators as polynomials in $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$. To this set of Boolean polynomials, we append the polynomials of Eqn (1) that relate the word-level and bit-level variables. We model this set of polynomials as $F = \{f_1, \ldots, f_s\}$ over the ring $R = \mathbb{F}_q[x_1, \ldots, x_d, Y, A]$. Here $x_1, \ldots, x_d$ denote, collectively, all the bit-level variables of the circuit — i.e. primary inputs, primary outputs and the intermediate circuit variables — and Y, A the word-level variables. Denote the generated ideal as $J = \langle F \rangle \subset R$. Also, let us denote the (unknown) "specification" of the circuit as a polynomial $Y - \mathcal{F}(A)$; or equivalently as $Y + \mathcal{F}(A)$ and -1 = +1 over $\mathbb{F}_{2^k}$.

As $Y = \mathcal{F}(A)$, clearly $Y + \mathcal{F}(A)$ *agrees with the solutions* to the circuit equations $f_1 = \cdots = f_s = 0$. In computer algebra terminology, this means that $Y + \mathcal{F}(A)$ *vanishes on the variety* $V_{\mathbb{F}_q}(J)$. If $Y + \mathcal{F}(A)$ vanishes on $V_{\mathbb{F}_q}(J)$, then due to Proposition 4.1, $Y + \mathcal{F}(A)$ is a member of the ideal $I(V_{\mathbb{F}_q}(J))$. Strong Nullstellensatz over Galois fields (Theorem 4.1) tells us that $I(V_{\mathbb{F}_q}(J)) = J + J_0$, where $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d, Y^q - Y, A^q - A \rangle$ is the ideal of vanishing polynomials in R. Consolidating these results, we deduce that:

Proposition 5.1: The specification polynomial $Y + \mathcal{F}(A) \in (J + J_0)$.

If the specification polynomial is known, then the verification problem can be solved via membership testing of $Y + \mathcal{F}(A)$ in the ideal $(J + J_0)$ ([16] actually used such a verification formulation). We will now show further that by computing a Gröbner basis of $(J + J_0)$, using a specific elimination term order, we can also identify the polynomial $Y + \mathcal{F}(A)$ which represents the function implemented by the circuit.

Reverse-Engineering $\mathcal{F}$ from C: The variety $V_{\mathbb{F}_q}(J)$ 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 of the circuit C, 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 $\mathbb{F}_q^{d-l}$ is equivalent to *eliminating l variables* from the corresponding ideal.

Definition 5.1: (**Elimination Ideal**) From [7]: 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

$$J_l = J \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d] \tag{2}$$

In other words, the lth elimination ideal does not contain variables $x_1, \ldots, x_l$, nor do the generators of it. This can aid in solving systems of polynomial equations by isolating variables in a set of constraints, as is the purpose of techniques such as Gaussian elimination. Moreover, Gröbner bases may be used to generate an elimination ideal by using an "elimination term order." One such ordering is a pure lexicographic ordering, which features into a theorem:

Theorem 5.1: (**Elimination Theorem**) From [7]: 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

$$G_l = G \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d] \tag{3}$$

is a Gröbner basis of the lth elimination ideal J_l.

We describe the application of elimination ideals using the following example, borrowed from [7].

Example 5.1: Consider polynomials $f_1 : x^2 - y - z - 1$, $f_2 : x - y^2 - z - 1$, $f_3 : x - y - z^2 - 1$ and ideal $J = \langle f_1, f_2, f_3 \rangle \subset \mathbb{C}[x,y,z]$. Let us compute a Gröbner basis G of J w.r.t. lex term order with $x > y > z$. Then $G = \{g_1, \dots, g_4\}$ is obtained as: $g_1 : x - y - z^2 - 1$; $g_2 : y^2 - y - z^2 - z$; $g_3 : 2yz^2 - z^4 - z^2$; $g_4 : z^6 - 4z^4 - 4z^3 - z^2$. Notice that the polynomial g_4 contains only the variable z, and it **eliminates** *variables x,y. Similarly, polynomials g_2, g_3, g_4, contain variables y,z and eliminate x. According to Theorem 5.1, $G_1 = G \cap \mathbb{C}[y,z] = \{g_2, g_3, g_4\}$ and $G_2 = G \cap \mathbb{C}[z] = \{g_4\}$ are the Gröbner bases of the 1^{st} and 2^{nd} elimination ideals of J, respectively.*

In conclusion, Gröbner basis computations w.r.t. pure lexicographic term orders can be used to eliminate variables from an ideal. The above example motivates our approach: Since we want to derive a polynomial representation from a circuit in variables Y, A, we can compute a Gröbner basis of $J + J_0$ w.r.t. an elimination order that eliminates all the (d) bit-level variables of the circuit. Then, the Gröbner basis $G_d = G \cap \mathbb{F}_q[Y,A]$ of the d^{th} elimination ideal of $(J + J_0)$ will contain polynomials in only Y, A. We now prove that the required polynomial representation will be found in G_d. First, let us formally "setup" the abstraction problem:

Problem Setup 5.1: Given a circuit C with k-bit inputs and outputs which computes a polyfunction $\mathcal{F} : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. Let $\{a_0, \dots, a_{k-1}\}$ be the bit-level primary inputs and $\{y_0, \dots, y_{k-1}\}$ be the primary outputs. Let A, Y denote the word-level input and output variables of the circuit, respectively, such that $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$ and $Y = y_0 + \cdots + y_{k-1}\alpha^{k-1}$, where α is a primitive element of $\mathbb{F}_{2^k}$. Let $\mathcal{F}(A)$ be the (unknown) polynomial representation of the function implemented by the circuit such that $Y = \mathcal{F}(A)$.

Denote by $x_i, i = 1, \dots, d$ all the Boolean variables of the circuit – i.e. the input, output and the intermediate variables. Let $R = \mathbb{F}_{2^k}[x_i, Y, A : i = 1, \dots d]$ denote the ring to model the polynomials that describe the circuit functionality. Let ideal $J \subset \mathbb{F}_{2^k}[x_i, Y, A : i = 1 \dots d]$ be generated by the bit-level and word-level polynomials of the circuit. Let $J_0 = \langle x_i^2 - x_i, Y^{2^k} - Y, A^{2^k} - A : i = 1, \dots, d \rangle$ denote the ideal of vanishing polynomials in R. □

Now, we will impose the following elimination order used for abstraction:

Definition 5.2: **Abstraction Term Order $>$:** Using the variable order $x_1 > x_2 > \cdots > x_d > Y > A$, impose a lex term order $>$ on the polynomial ring $R = \mathbb{F}_q[x_1, \dots, x_d, Y, A]$. This elimination term order $>$ is defined as the **Abstraction Term Order**. The relative ordering among $x_1, \dots, x_d$ is not important and can be chosen arbitrarily.

Theorem 5.2: **Abstraction Theorem:** Using the setup and notations from *Problem Setup* 5.1 above, compute a Gröbner basis G of ideal $(J + J_0)$ using the abstraction term order $>$. Then:
(i) G must contain the vanishing polynomial $A^q - A$ as the only polynomial with only A as the support variable;
(ii) G must contain a polynomial of the form $Y + \mathcal{G}(A)$; and
(iii) $Y + \mathcal{G}(A)$ 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 $\mathbb{F}_q$.

Proof: (i) $A^q - A$ is a given generator of J_0. Variable A is also the last variable in the abstraction term order. Moreover, A is an input to the circuit, so A is an independent variable which can take any and all values in $\mathbb{F}_{2^k}$. As a result, $G_{d+1} = G \cap \mathbb{F}_{2^k}[A] = \{A^q - A\}$.

(ii) Since $f : Y + \mathcal{F}(A)$ is a polynomial representation of the function of the circuit, $Y + \mathcal{F}(A) \in J + J_0$, due to Proposition 5.1. Therefore, according to the definition of a Gröbner basis (Definition 4.1), the leading term of $Y + \mathcal{F}(A)$ (which is Y) should be divisible by the leading term of some polynomial $g_i \in G$. The only way $lt(g_i)$ can divide Y is when $lt(g_i) = Y$ itself. Moreover, due to our abstraction (lex) term order, $Y > A$, so this polynomial must be of the form $Y + \mathcal{G}(A)$.

(iii) As $Y = \mathcal{F}(A)$ represents the function of the circuit, $Y + \mathcal{F}(A) \in J + J_0$. Moreover, $V(J + J_0) \subset V(Y + \mathcal{F}(A))$. Project this variety $V(J + J_0)$ onto the co-ordinates corresponding to (A, Y). What we obtain is the *graph of the function $A \mapsto \mathcal{F}(A)$* over $\mathbb{F}_{2^k}$. Since $Y + \mathcal{G}(A)$ is an element of the Gröbner basis of $J + J_0$, $V(J + J_0) \subset V(Y + \mathcal{G}(A))$ too. Therefore, $Y = \mathcal{G}(A)$ gives the same function as $Y = \mathcal{F}(A)$, for all $A \in \mathbb{F}_{2^k}$. ■

As a consequence of Theorem 5.2, if we compute a Gröbner basis G of $J + J_0$ using the abstraction term order, we will find a polynomial of the form $Y + \mathcal{G}(A)$ in the Gröbner basis, such that $Y = \mathcal{G}(A)$ is a polynomial representation of the circuit. However, if the Gröbner basis is not reduced, it is possible to obtain multiple polynomials in G of the form $Y + \mathcal{G}_1(A), Y + \mathcal{G}_2(A), \dots,$; all of which correspond to the same function.

Corollary 5.1: Computing a **reduced** Gröbner basis G_r of $J + J_0$, we will obtain **one and only one polynomial** in G_r of the form $Y + \mathcal{G}(A)$, such that $Y = \mathcal{G}(A)$ is the **unique, minimal, canonical** representation of the function f implemented by the circuit.

The above results trivially extend to circuits with multiple word-level input variables $A^1, \dots, A^n$, and the canonical polynomial representation obtained by computing a reduced Gröbner basis G_r of $J + J_0$ using $>$ is of the form $Y = \mathcal{F}(A^1, \dots, A^n)$. Our approach can also be applied to circuits with n inputs and m outputs, to interpolate polyfunctions $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$. This is achieved by modeling the function over $\mathbb{F}_{2^k}$ where $k = LCM(n,m)$ — i.e., over the smallest single field $\mathbb{F}_{2^k}$ that contains both $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^m}$.

Example 5.2: **Demonstration of our approach:** *Consider a 2-bit multiplier circuit over $\mathbb{F}_{2^2}$ given in Fig. 3, which implements a function $f : \mathbb{F}_4 \times \mathbb{F}_4 \to \mathbb{F}_4$. It is a* **polynomial function***: $Z = A \times B$, $Z, A, B \in \mathbb{F}_4$. Variables a_0, a_1, b_0, b_1 are primary inputs, z_0, z_1 are primary outputs, and s_0, s_1, s_2, s_3, r_0 are intermediate variables of the circuit. Then, we can consider $A = a_0 + a_1\alpha$, $B = b_0 + b_1\alpha$ as the word-level inputs and $Z = z_0 + z_1\alpha$ as the output in $\mathbb{F}_4$. Here $P(x) = x^2 + x + 1$ is the primitive polynomial used to construct the field $\mathbb{F}_4$ and $P(\alpha) = 0$, i.e. α is a root of $P(x)$.*

The functionality of the entire circuit can be described using the following polynomials derived from the Boolean gate-level operators: $f_1 : z_0 + z_1\alpha + Z$; $f_2 : b_0 + b_1\alpha + B$; $f_3 : a_0 + a_1\alpha + A$; $f_4 : s_0 + a_0 \cdot b_0$; $f_5 : s_1 + a_0 \cdot b_1$; $f_6 : s_2 + a_1 \cdot b_0$; $f_7 : s_3 + a_1 \cdot b_1$; $f_8 : r_0 + s_1 + s_2$; $f_9 : z_0 + s_0 + s_3$; $f_{10} : z_1 + r_0 + s_3$. Ideal $J = \langle f_1, \dots, f_{10} \rangle$. Generate J_0 as the ideal of vanishing polynomials. Impose the following **elimination** *term order:* **lex term order** *with "circuit variables" > "Output Z" > "Inputs,*

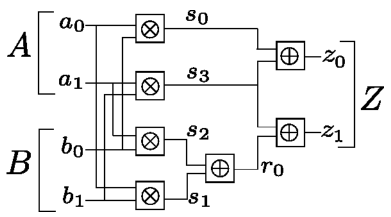

Fig. 3: A 2-bit Multiplier over $\mathbb{F}_{2^2}$. The gate $\otimes$ corresponds to AND-gate, i.e. bit-level multiplication modulo 2. The gate $\oplus$ corresponds to XOR-gate, i.e. addition modulo 2.

A, B" and compute a Gröbner basis G of $J+J_0$, we observe the following polynomials in the basis: $g_1: z_0+z_1\alpha+Z$; $g_2: b_0+b_1\alpha+B$; $g_3: a_0+a_1\alpha+A$; $g_4: s_3+r_0+z_1$;$g_5: s_1+s_2+r_0$; $g_6: s_0+s_3+z_0$; $\mathbf{g_7: Z+AB}$; $g_8: a_1b_1+a_1B+b_1A+z_1$; $g_9: r_0+a_1b_1+z_1$; $g_{10}: s_2+a_1b_0$, *and the polynomials of* J_0*. Notice that the polynomial* $g_7: Z+AB$ *describes* $Z=AB$ *as the (canonical) polynomial function implemented by the circuit; and we were able to* **extract** *this representation using the Gröbner basis computation.*

VI. Preliminary experiments and FGLM

Theorem 6.1: (From [16]:) Let $J+J_0 = \langle f_1,\dots,f_s,\ x_1^q - x_1,\dots,x_d^q - x_d\rangle \subset \mathbb{F}_q[x_1,\dots,x_d]$ be an ideal. The time and space complexity of Buchberger's algorithm to compute a Gröbner basis of $J+J_0$ is bounded by $q^{O(d)}$. assuming that the length of input $f_1,\dots,f_s$ is dominated by $q^{O(d)}$.

In our case $q=2^k$, and when k and d are large, this complexity makes verification infeasible. The impact of this complexity is clearly visible in the results of preliminary experiments (TABLE II). The experiments are conducted with Mastrovito [33] multiplier circuits of various word sizes k. Each multiplier computes the polynomial function $Y=\mathcal{F}(A,B)=A\cdot B$ over $\mathbb{F}_{2^k}$, where Y,A,B are k-bit vectors. We extract the Boolean gate-level operators J and generate vanishing polynomials J_0. Then we compute the Gröbner basis of $J+J_0$ with respect to abstraction term order $>$. The computation was performed using the "slimgb" command in the SINGULAR' computer algebra tool [27]. The resulting Gröbner bases contained a polynomial $Y+A\times B$. These experiments were run on a 64-bit Ubuntu OS running on a 2.4GHz Intel QuadCore processor with 8Gb of memory. The approach is infeasible beyond 40-bit datapath size, as the Gröbner basis engine encounters a memory explosion. BDDs cannot be constructed for any of these circuits.

TABLE II: Gröbner Basis runtime for Mastrovito Multipliers in Singular using Abstraction Term Order $>$.

Word Size (k)	# Polynomials (# gates)	Time (minutes)
16	1,871	2.4
24	3,135	12
32	5,549	22.6
40	8,587	266
48	12,327	NA (Out of Memory)

Proposed FGLM Approach: To make this approach scalable, we are investigating the use of the FGLM algorithm [15]. This algorithm provides a method for converting a Gröbner basis in one term ordering to a Gröbner Basis in another term ordering. We use this method in conjunction with following result which was derived and proved in [16]:

Theorem 6.2: Let C be any arbitrary combinational circuit. Let $\{x_1,\dots,x_d\}$ denote the set of all variables (signals) in the circuit, i.e. the primary input, intermediate and primary output variables. Perform a *reverse topological traversal* of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lex term order to represent each gate as a polynomial f_i, s.t. $f_i = x_i + \text{tail}(f_i)$. Then the set of all polynomials $G_1 = \{f_1,\dots,f_s, x_i^q - x_i : i = 1,\dots,d\}$ constitutes a Gröbner basis.

Using the term ordering specified by [16], which we denote $>_1$, we can obviate the need to compute a Gröbner basis. Imposing the monomial ordering $>_1$ already makes G_1 a Gröbner basis of $J+J_0$. Now we apply the FGLM algorithm to transform the Gröbner basis into another basis w.r.t. the abstraction ordering $>$. This new basis will also definitely contain a polynomial in the form of $Y+\mathcal{F}(A)$ as desired. By using FGLM to convert the Gröbner basis from $>_1$ to $>$, we are hoping to avoid the complexity presented by computing a Gröbner basis over the abstraction ordering $>$.

Example 6.1: Let us revisit the 2-bit multiplier circuit from Example 5.2, shown in Fig. 3.

As before, we can describe the functionality of this circuit, J, using the following polynomials: $f_1: Z+z_0+z_1\alpha$; $f_2: B+b_0+b_1\alpha$; $f_3: A+a_0+a_1\alpha$; $f_4: s_0+a_0\cdot b_0$; $f_5: s_1+a_0\cdot b_1$; $f_6: s_2+a_1\cdot b_0$; $f_7: s_3+a_1\cdot b_1$; $f_8: r_0+s_1+s_2$; $f_9: z_0+s_0+s_3$; $f_{10}: z_1+r_0+s_3$. *By imposing the monomial order* $>_1$, $J+J_0$ *is already a Gröbner basis, where* J_0 *is the ideal of vanishing polynomials of the circuit:* $f_{11}: a_0^2+a_0$; $f_{12}: a_1^2+a_1$; $f_{13}: b_0^2+b_0$; $f_{14}: b_1^2+b_1$; $f_{15}: s_0^2+s_0$; $f_{16}: s_1^2+s_1$; $f_{17}: s_2^2+s_2$; $f_{18}: s_3^2+s_3$; $f_{19}: r_0^2+r_0$; $f_{20}: z_0^2+z_0$; $f_{21}: z_1^2+z_1$. *We denote this Gröbner basis as* $G_1=\{f_1,\dots,f_{21}\}$*. Using the FGLM algorithm ("fglm" command in Singular) we convert* G_1 *from ordering* $>_1$ *to G with the abstraction term ordering* $>$*. The resulting Gröbner basis G contains the polynomial* $Z\cdot(\alpha+1)+A\cdot B\cdot(\alpha+1)$*, which can be equivalently reduced to* $Z+A\cdot B$*.*

FGLM exploits concepts from algebraic geometry and sparse linear algebra to transform the Gröbner basis. While a detailed exposition of FGLM is beyond the scope of this paper, we demonstrate its operation on the 2-bit multiplier shown in Example 6.1.

Example 6.2: FGLM begins by taking the least variable in the the abstraction term ordering, (A in our case). Starting with $m=0$*, it computes* $A^m \xrightarrow{G_1}_+ r$*. It checks whether the remainder r can be represented as a linear combination of any remainders calculated thus far. If so, it adds this representation to G and moves on to the next monomial; otherwise it increments m and repeats* $A^m \xrightarrow{G_1}_+ r$ *computation.*

- $A^0 \xrightarrow{G_1}_+ 1$
- $A^1 \xrightarrow{G_1}_+ a_0+a_1\alpha$
- $A^2 \xrightarrow{G_1}_+ a_0+a_1\cdot(\alpha+1)$
- $A^3 \xrightarrow{G_1}_+ a_0\cdot a_1+a_0+a_1$

- $A^4 \xrightarrow{G_1}_+ a_0 + a_1\alpha = A$

A^4 can be composed of A, so $A^4 - A$ is added to G. Similarly, $B^4 - B$ is also added to G. Continue to the next variable Z in the order as we have "circuit variables" $> Z > A > B$.

- $Z^1 \xrightarrow{G_1}_+ a_0 \cdot b_0 + (\alpha) \cdot a_0 \cdot b_1 + (\alpha) \cdot a_1 \cdot b_0 + (\alpha^2) \cdot a_1 \cdot b_1 = a_0(b_0 + b_1\alpha) + a_1\alpha(b_0 + b_1\alpha) = (a_0 + a_1\alpha)(b_0 + b_1\alpha) = A \cdot B$

Z is found to be a function of A and B, therefore $Z - AB$ is added to G.

FGLM continues to convert the rest of the monomials into the new ordering. However, since we are only looking for $Y = \mathcal{F}(A)$ polynomial representation, we can make this approach even more efficient by restricting the conversion to only the word-level variables of the circuit. This idea is currently under development.

VII. Conclusions

This paper has described an approach to derive a word-level, canonical polynomial representation from combinational circuits using Gröbner bases. Our approach interprets the function of the circuit $f : \mathbb{B}^k \to \mathbb{B}^k$ as a polynomial function over Galois fields $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. By deriving a set of polynomials corresponding to the circuit (ideal $J + J_0$), and computing its Gröbner basis w.r.t. a specific elimination (abstraction) term order, the canonical representation of this circuit can be derived. As Gröbner basis computation exhibits high complexity, we are currently investigating the use of the FGLM algorithm to make our approach scalable.

References

[1] A. Peymandoust and G. DeMicheli, "Application of Symbolic Computer Algebra in High-Level Data-Flow Synthesis", *IEEE Trans. CAD*, vol. 22, pp. 1154–11656, 2003.

[2] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications", *in DATE*, pp. 52–58, 1999.

[3] Z. Zeng, P. Kalla, and M. J. Ciesielski, "LPSAT: A Unified Approach to RTL Satisfiability", *in Proc. DATE*, 2001.

[4] D. Babic and M. Musuvathi, "Modular Arithmetic Decision Procedure", Technical Report TR-2005-114, Microsoft Research, 2005.

[5] R. Brummayer and A. Biere, "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays", *in TACAS 09, Volume 5505 of LNCS*. Springer, 2009.

[6] R.S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions", *in High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*, pp. 166–171, Nov.

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

[8] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)", *in Proceedings of the Selected Areas in Cryptography*, pp. 201–212, London, UK, UK, 1999. Springer-Verlag.

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

[10] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Ordered Kronecker Functional Decision Diagrams", *in Design Automation Conference*, pp. 415–419, 1994.

[11] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams", *in DAC*, 95.

[12] D. Singmaster, "On Polynomial Functions (mod m)", *J. Number Theory*, vol. 6, pp. 345–352, 1974.

[13] N. Shekhar, P. Kalla, and F Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing", *IEEE Transactions on CAD*, vol. 26, pp. 1320–1330, July 2007.

[14] Rudolf Lidl and Harald Niederreiter, *Finite Fields*, Cambridge University Press, 1997.

[15] J. C. Faugére, P. Gianni, D. Lazard, and T. Mora, "Efficient computation of zero-dimentional Gröbner Basis by change of ordering", *Journal of Symbolic Computation*, vol. 16, pp. 329–344, 1993.

[16] J. Lv, P. Kalla, and F. Enescu, "Efficient Groebner Basis Reductions for Formal Verification of Galois Field Multipliers", *in IEEE Design, Automation and Test in Europe*, 2012.

[17] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, PhD thesis, Philosophiesche Fakultät an der Leopold-Franzens-Universität, Austria, 1965.

[18] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications", *in ICCAD*, pp. 188–191, Nov. 93.

[19] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams - Overcoming the Limitation of MTBDDs and BMDs", *in ICCAD*, 95.

[20] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "FGILP: An ILP Solver based on Function Graphs", *in ICCAD*, pp. 685–689, 93.

[21] R. Dreschler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure", *IEEE Design & Test*, vol. 14, pp. 51–59, 1997.

[22] M. Ciesielski, P. Kalla, and S. Askar, "Taylor Expansion Diagrams: A Canonical Representation for Verification of Data-Flow Designs", *IEEE Transactions on Computers*, vol. 55, pp. 1188–1201, 2006.

[23] B. Alizadeh and M. Fujita, "Modular Datapath Optimization and Verification based on Modular-HED", *IEEE Transactions CAD*, pp. 1422–1435, Sept. 2010.

[24] A. Jabir and Pradhan D., "MODD: A New Decision Diagram and Representation for Multiple Output Binary Functions", *in Design, Automation and Test in Europe, DATE*, 2004.

[25] A. Jabir, D. Pradhan, T. Rajaprabhu, and A. Singh, "A Technique for Representing Multiple Output Binary Functions with Applications to Verification and Simulation", *IEEE Transactions on Computers*, vol. 56, pp. 1133–1145, 2007.

[26] A. Lvov, L. Lastras-Montaño, V. Paruthi, R. Shadowen, and A. El-Zein, "Formal Verification of Error Correcting Circuits using Computational Algebraic Geometry", *in Proc. Formal Methods in Computer-Aided Design (FMCAD)*, pp. 141–148, 2012.

[27] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-3 — A computer algebra system for polynomial computations", 2011, http://www.singular.uni-kl.de.

[28] R. Zippel, "Probabilistic algorithms for sparse interpolation", *in Proc. Symp. Symbolic and Algebraic Computation*, pp. 216–226, 1979.

[29] M. Ben-Or and P. Tiwari, "A deterministic algorithm for sparse multivariate polynomial interpolation", *in Proc. Symp. Theory of Computing*, pp. 301–309, 1988.

[30] S. Javadi and M. Monagan, "On sparse polynomial interpolation over finite fields", *in Intl. Symp. Symbolic and Algebraic Computing*, 2010.

[31] Z. Zilic and Z. Vranesic, "A deterministic multivariate interpolation algorithm for small finite fields", *IEEE Trans. Computers*, vol. 51, Sept. 2002.

[32] Robert J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

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

[34] Cetin K. Koc and Tolga Acar, "Montgomery Multiplication in GF(2^k)", *Designs, Codes and Cryptography*, vol. 14, pp. 57–69, April 1998.

[35] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF(2^n) Without Pre-Computational Phase", *in Proceedings of the International Workshop on Arithmetic of Finite Fields*, pp. 77–87, 2008.

[36] Christof Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields", *IEEE Transactions on Computers*, vol. 45, pp. 856–861, July 1996.

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

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