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*—*Abstraction* plays an important role in digital design, analysis and verification. This paper introduces 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, A represent the k-bit word-level output and input 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 i) the structure of the circuit, ii) the properties of Gröbner bases, iii) characteristics of finite fields $\mathbb{F}_{2^k}$, and iv) modern algorithms from symbolic algebra, to derive the canonical polynomial representation.

A standalone customized tool is developed that implements these concepts to derive the polynomial abstraction. This approach and our tool are used to verify (and detect bugs in) combinational finite field arithmetic circuits – with up to 1024-bit operands – whereas contemporary verification techniques are infeasible.

***Keywords*-Word-Level Abstraction, Formal Verification, Equivalence Checking, Gröbner Bases, Finite Fields.**

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. For instance, a bit-level representation of a multiplier is represented as a collection of logic gates and nets, whereas a word-level abstraction hides the underlying logic and represents the function with bit-vector level inputs and output, e.g. $Z = A \times B$. As the datapath size of the multiplier grows, the bit-level representation may increase (possibly exponentially) in size, while the word-level abstraction does not change. 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.

Word-level abstractions of circuit blocks also have applications in other areas of electronic design automation (EDA), such as in high-level datapath synthesis [1], resource allocation [2], component matching and reuse [3], word-level interpolants [4], SMT-solving [5], etc. Due to their many fundamental applications, it is important to investigate various forms of word-level functional abstractions of hardware designs along with efficient algorithmic techniques to derive them.

This paper describes a method to derive a *canonical word-level polynomial representation from 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 [6] [7]) 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 polynomial abstraction approach is based on the following mathematical insights:

The mathematical framework: A combinational circuit C with k-bit inputs and k-bit outputs implements Boolean functions that are mappings between k-dimensional Boolean spaces: $f : \mathbb{B}^k \to \mathbb{B}^k$, where $\mathbb{B} = \{0, 1\}$. The function f, which is a mapping among 2^k elements, can also be construed as a function $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, i.e. as a function over the finite field of 2^k elements. It is well-known that over the finite field ($\mathbb{F}_q$) of q elements, *every function* $f : \mathbb{F}_q \to \mathbb{F}_q$ is a polynomial function [8]. 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, polynomial abstraction* of the function as $Z = \mathcal{F}(A)$ over $\mathbb{F}_{2^k}$, where $Z = \{z_0, \ldots, z_{k-1}\}$, $A = \{a_0, \ldots, a_{k-1}\}$ 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 $f : \mathbb{B}^n \to \mathbb{B}^m$, modeled as a polynomial over $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^m}$. Note that the function $f : \mathbb{B}^k \to \mathbb{B}^k$ can also be viewed as a mapping over finite integer rings $\mathbb{Z}$ (mod 2^k), i.e. over $f : \mathbb{Z}_{2^k} \to \mathbb{Z}_{2^k}$. However, not every function is a polynomial function over $\mathbb{Z}_{2^k}$, so the finite integer ring model is beyond the scope of this paper.

The polynomial $\mathcal{F}$ can be derived by means of the Lagrange interpolation formula [8] [9]. However, this requires to analyze f over the entire field $\mathbb{F}_{2^k}$, which is exhaustive and infeasible. To make this approach practical, we propose a symbolic method based on computer algebra and algebraic geometry to derive the canonical polynomial abstraction from the circuit. This abstraction is employed for formal verification and equivalence checking of combinational circuits C_1, C_2. The circuits can be analyzed separately to derive their corresponding canonical polynomial representations $\mathcal{F}_1, \mathcal{F}_2$, respectively.

T. Pruss and P. Kalla (kalla@ece.utah.edu) are with the Dept. of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112. F. Enescu is with the Dept. of Mathematics and Statistics, Georgia State University, Atlanta, GA 30303. This work is supported in part by the U.S. National Science Foundation grants CCF-1320335 and CCF-1320385.

Equivalence test is then performed by simply matching the coefficients of $\mathcal{F}_1, \mathcal{F}_2$.

Motivating application: While this approach is theoretically applicable to arbitrary combinational circuits, the main motivation to derive this approach stems from the problem of 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 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 [10] [11] [12] [13]. Custom design raises the potential for bugs in large systems. Arithmetic bugs are known to compromise the security of crypto-systems [14]; 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 (*e.g.* [15] [16]) are unable to prove equivalence between such large, custom, modulo-arithmetic circuits.

A. Approach & 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* [7], 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* $>$.

Computing Gröbner bases using elimination orders is infeasible for large circuits. To overcome this limitation, we present a *refinement* of this abstraction term order 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. By exploiting the *binomial expansion* over $\mathbb{F}_{2^k}$, we further deduce that the symbolic computation problems so derived exhibit a very special structure that further simplify our computations.

This 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. We demonstrate the application of our approach to verify a variety of finite field arithmetic architectures. Our approach can verify, and also find bugs in, large (up to $k = 1024$ bit) arithmetic circuits, whereas contemporary verification techniques are infeasible. Our approach is, however, not efficient for verification of random-logic and integer arithmetic circuits. The 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.

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, 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 the paper.

II. Related Previous Work

Canonical Representations: The Reduced Ordered Binary Decision Diagram (ROBBD) [17] — and its variants OKFDDs, ADDs, BMDs, etc. — are canonical DAG representations of functions that are employed in design verification. The various decomposition principles behind these diagrams are based on point-wise, binary decomposition, w.r.t. each (Boolean) variable. As such, these do not fully provide word-level abstraction capabilities from bit-level representations. The Taylor Expansion Diagram (TED) [18] is a word-level canonical representation of a polynomial *expression*, but it does not represent a polynomial *function* canonically. The work of [19] and [20] represents polynomial functions canonically, but over finite integer rings $\mathbb{Z}_{2^k}$ and not over $\mathbb{F}_{2^k}$. MODDs [21] are a DAG representation of the characteristic function of a circuit over finite fields $\mathbb{F}_{2^k}$. MODDs come close to satisfying our requirements as a canonical word-level representation that can be employed over finite fields. However, MODDs do not scale well w.r.t. the circuit size. MODDs are infeasible in representing functions over larger than 32-bit words [21].

Equivalence Checking: Modern equivalence checkers employ techniques based on And-Invert-Graph (AIG) reductions [15] and circuit-SAT solvers [22]. Such techniques are able to identify internal structural equivalences between the specification models (*Spec*) and implementation (*Impl*) circuits and reduce the instances for verification. However, when the arithmetic circuits are structurally very dissimilar, these techniques are infeasible in proving equivalence (Tables I and II in [16] depict such experiments). In general, the applications targeted in this paper are hard for SAT/SMT solvers.

Computer algebra based verification: In [23] [24] [25] [13], the authors present the BLUEVERI tool from IBM for verification of finite field error correcting circuits 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 and Gröbner basis formulation. In their setting, the polynomial representation of the sub-circuit blocks is already available, whereas our approach identifies such a representation. Moreover, improvements to the core Gröbner basis computational engine are not the subject of their work.

In [26] [16], *Lv et al.* present 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 [16], we are not given the specification polynomial f. Given the circuit C, we have to *derive (extract)* the word-level specification f.

Among other relevant works, [27] describes 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 $\mathbb{F}_q$ [28]. Our problem formulation employs some of the concepts presented in [27] [28]. Computer algebra techniques have also been employed for verification of integer arithmetic circuits [29] [30].

Other function extraction techniques: In [31], the authors present an approach to function extraction from bit-level circuits using a network-flow based model – by interpreting the computation as a flow of binary data through the circuit-network, represented as a pseudo-Boolean expression. Improvements to this approach are described in [32], where the algebraic trasformations are guided by analyzing the structure (levelization) of the circuit. The extracted signatures are in terms of *bit-level* polynomials and do not provide a word-level abstraction. In [3], the authors present an approach that searches for a *linear* word-level abstraction, with integer coefficients, using *BMDs. However, their approach is not complete in the sense that a linear word-level abstraction does not always exist for arbitrary circuits.

Polynomial Interpolation: Conceptually, our abstraction can be derived using polynomial interpolation. It falls into the category of *dense* interpolation (as opposed to the classical multivariate sparse interpolation problem, see [33]), as we require a polynomial that describes the function at each of the q points of the field $\mathbb{F}_q$. However, Newton's dense interpolation exhibits very high complexity. In the logic synthesis and VLSI testing area, the work of [9] 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 multi-valued logic.

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 work, 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: i) $\mathbb{F}_2 = \{0,1\}$ denotes the finite field of 2 elements; ii) $\mathbb{F}_2[x]$ is the univariate polynomial ring with coefficients in $\mathbb{F}_2$; and iii) $P(x)$ 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 work, 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 $a_i \in \mathbb{F}_2, i = 0,\dots,k-1$, and α is a root of the primitive polynomial, *i.e.* $P(\alpha) = 0$. Since a k-bit vector $\{a_0,\dots,a_{k-1}\}$ represents 2^k distinct values, it can be viewed as an element A of $\mathbb{F}_{2^k}$.

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 a primitive polynomial of degree* $k = 4$. *Let* α *be a 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 the 16 elements of the field* $\mathbb{F}_{16}$. *Each element 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	$\alpha^3+\alpha^2+\alpha+1$

There may be more than one primitive polynomials of degree k in $\mathbb{F}_2[x]$, and any of them could be used to construct the field $\mathbb{F}_{2^k}$. Finite fields are unique (up to isomorphism) irrespective of the chosen primitive polynomial. For verification, if the primitive polynomial $P(x)$ is already given, we use it for abstraction. Otherwise, we choose a $P(x)$ of degree k with fewest terms, as the reduction $\pmod{P(x)}$ may result in fewer terms being generated.

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 $\mathbb{F}_{2^k}$ (as shown in the above example), 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 3.1: (From [8]) 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

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

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.

Example 3.2: Let $A = \{a_2, a_1, a_0\}$ and $Z = \{z_2, z_1, z_0\}$ be 3-bit vectors. Consider the function $Z[2:0] = A[2:0] >> 1$, i.e. a 1-bit right shift operation on A. The function maps as follows:

$\{a_2a_1a_0\}$	A	$\rightarrow$	$\{z_2z_1z_0\}$	Z
000	0	$\rightarrow$	000	0
001	1	$\rightarrow$	000	0
010	α	$\rightarrow$	001	1
011	$\alpha+1$	$\rightarrow$	001	1
100	α^2	$\rightarrow$	010	α
101	α^2+1	$\rightarrow$	010	α
110	$\alpha^2+\alpha$	$\rightarrow$	011	$\alpha+1$
111	$\alpha^2+\alpha+1$	$\rightarrow$	011	$\alpha+1$

By applying Lagrange's interpolation formula over $\mathbb{F}_{2^3}$, we obtain $Z = (\alpha^2+1)A^4 + (\alpha^2+1)A^2$, as the canonical polynomial representation of the function, where $P(\alpha) = \alpha^3+\alpha+1 = 0$.

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:

Definition 3.1: Any function $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ has a **unique canonical representation** (UCR) as a polynomial $\mathcal{F} \in \mathbb{F}_q[x_1,\dots,x_n]$ 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 $j = 1,\dots,n$.

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 $\mathbb{F}_{2^k}$ [34], as shown below:

Example 3.3: Consider point addition in López-Dahab (LD) projective coordinate. Given an elliptic curve: $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ over $\mathbb{F}_{2^k}$, where X, Y, Z are k-bit vectors that are elements in $\mathbb{F}_{2^k}$ and similarly, a, 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, Z_3 can be computed as follows:

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

Efficient VLSI architectures for multiplication and squaring have been devised [35] [11] [12] [10], which are employed as modulo-arithmetic and logic units (mALUs) in such cryptosystems [36]. We briefly review such arithmetic architectures on which we have applied our abstraction based approach for verification.

Over finite fields $\mathbb{F}_{2^k}$, multiplication is performed as $Z = A \times B \pmod{P(x)}$, where $A, B \in \mathbb{F}_{2^k}$ are k-bit inputs, Z is the k-bit output, and $P(x)$ is the given primitive polynomial. The multiplier circuit takes bit-level inputs $\{a_0,\dots,a_{k-1},\ b_0,\dots,b_{k-1}\}$ and produces output $\{z_0,\dots,z_{k-1}\}$, 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 [35].

Mastrovito multipliers are inefficient, especially for cryptosystems where multiplication is often performed repeatedly. For such applications, Montgomery Reduction (MR) operations are proposed [11] [12] that compute: $MR(A,B) = A \cdot B \cdot R^{-1} \pmod{P(x)}$, 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 $MR(A,B)$ cannot directly compute $A \cdot B \pmod{P(x)}$, we need to pre-compute $A \cdot R$ and $B \cdot R$, as shown in Fig. 1.

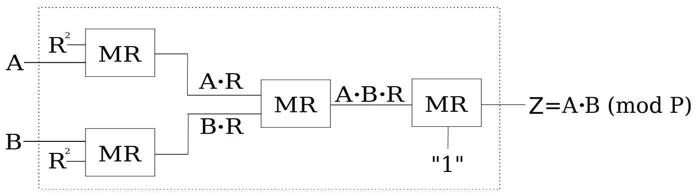


Fig. 1: *Montgomery* multiplication over $\mathbb{F}_{2^k}$ using four MRs.

In many non-ECC based crypto-systems, the datapath size k can be extremely large, e.g. $k = 1024$ bits. To overcome the complexity of such large designs, the concept of *composite field arithmetic* is used [37]. Here, the field $\mathbb{F}_{2^k}$ is decomposed as $\mathbb{F}_{(2^m)^n}$ for a non-prime $k = m \cdot n$, and the circuits are designed over the decomposed field. 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 $\mathbb{F}_{(2^m)^n}$ [10].

Example 3.4: 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 $A = (a_3,\dots,a_0), B = (b_3,\dots,b_0)$ over $\mathbb{F}_{2^4}$ are first transformed into elements A_0, A_1, B_0, 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

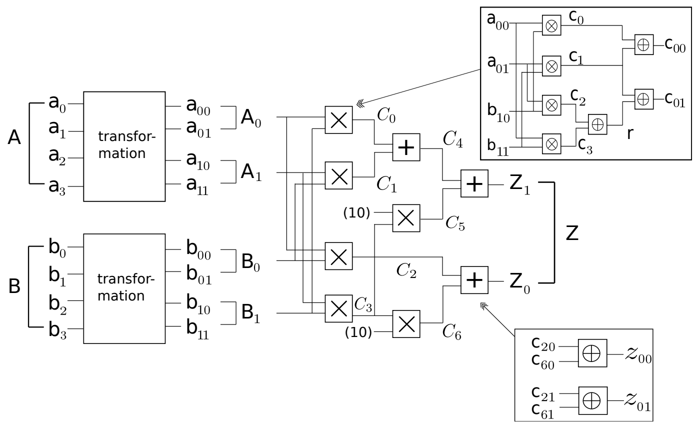


Fig. 2: 4-bit composite field multiplier designed over $\mathbb{F}_{(2^2)^2}$.

with both flattened ("bit-blasted") and hierarchical implementations of the above multipliers, 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 $\mathbb{F}_q[x_1,\dots,x_d]$ be the polynomial ring in variables $x_1,\dots,x_d$. A monomial in variables $x_1,\dots,x_d$ is a power product of the form $X = x_1^{e_1}\cdot x_2^{e_2}\cdots x_d^{e_d}$, where $e_i \in \mathbb{Z}_{\geq 0}, i \in \{1,\dots,d\}$. A *polynomial* $f \in \mathbb{F}_q[x_1,\dots,x_d]$ is written as a finite sum of terms $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$. Here $c_1,\dots,c_t$ are coefficients and $X_1,\dots,X_t$ are monomials. To systematically manipulate the polynomials, a monomial order $>$ (also called a term order) is imposed on the ring. The monomials of any polynomial $f = c_1X_1 + c_2X_2 + \cdots + c_tX_t$ are ordered w.r.t. to $>$, such that $X_1 > X_2 > \cdots > X_t$. Subject to such a term order, $lt(f) = c_1X_1,\ lm(f) = X_1,\ lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. We also denote tail(f) = $f - lt(f) = c_2X_2 + \cdots + c_tX_t$. In this work, we will mostly be concerned with terms ordered lexicographically (*lex*).

2) Polynomial reduction: Polynomial reduction (division) plays a key role in our abstraction algorithms. Let f, g be polynomials. If a non-zero 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 $r = f - \frac{cX}{lt(g)}\cdot g$. Similarly, f can be *reduced (divided) w.r.t. a set of polynomials* $F = \{f_1,\dots,f_s\}$ 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) Ideals, Varieties & Nullstellensatz: To analyze the function implemented by a circuit, we will model the circuit by way of a set of polynomials $F = \{f_1,\dots,f_s\}$, and then analyze the *set of all solutions to* $f_1 = f_2 = \cdots = f_s = 0$. The set of all solutions to a given system of polynomial equations $f_1 = \cdots = f_s = 0$ is called the *variety*, denoted as $V(f_1,\dots,f_s)$. The variety depends not just on the given system of polynomials, but rather on the *ideal* generated by the polynomials.

Definition 3.2: An *ideal* J generated by polynomials $f_1,\dots,f_s \in \mathbb{F}_q[x_1,\dots,x_d]$ is:

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

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

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

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

In the context of this work, the set of polynomials $F = \{f_1,\dots,f_s\}$ describing the given circuit generates an ideal $J = \langle f_1,\dots,f_s\rangle \subseteq \mathbb{F}_q[x_1,\dots,x_d]$. 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 3.3: 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: $I(V) = \{f \in \mathbb{F}_q[x_1,\dots,x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$. Therefore, if a polynomial f vanishes on a variety V, then $f \in I(V)$.

Our abstraction problem is formulated using the *Strong Nullstellensatz* applied over $\mathbb{F}_q$, which is stated below. The proof of this fundamental result can be found in Theorem 3.2 in [27]. The notation of sum of ideals is used below: if $I_1 = \langle f_1,\dots,f_s\rangle$ and $I_2 = \langle h_1,\dots,h_r\rangle$, then $I_1 + I_2 = \langle f_1,\dots,f_s,\ h_1,\dots,h_r\rangle$. Moreover, $J_0 = \langle x_1^q - x_1,\dots,x_d^q - x_d\rangle$ is used to denote the *ideal of all vanishing polynomials* over $\mathbb{F}_q$.

Theorem 3.2: Strong Nullstellensatz over $\mathbb{F}_q$: Let $J \subseteq \mathbb{F}_q[x_1,\dots,x_d]$ be an ideal, and let $J_0 = \langle x_1^q - x_1,\dots,x_d^q - x_d\rangle$ 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 $F = \{f_1,\dots,f_s\}$ and $G = \{g_1,\dots,g_t\}$ such that $J = \langle f_1,\dots,f_s\rangle = \langle g_1,\dots,g_t\rangle$ and $V(J) = V(f_1,\dots,f_s) = V(g_1,\dots,g_t)$. Some generating sets are "better" than others, i.e. they are a better representation of the ideal. A *Gröbner basis* is one such representation that possesses many important properties that allow to solve many polynomial decision questions. In the context of this work, Gröbner bases are utilized as a canonical representation of an ideal.

Definition 3.4: [**Gröbner Basis**] [6]: For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner basis of J iff $\forall f \in J$, $f \neq 0$, there exists $i \in \{1, \cdots, t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \ \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f)$.

Buchberger's algorithm [38], 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* ($Spoly(f, g)$):

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

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

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** $G = G'$;
```

A Gröbner basis G may contain redundant elements. To remove these redundant elements, G is first made *minimal* and subsequently *reduced.*

Definition 3.5: A Gröbner basis $G = \{g_1, \ldots, g_t\}$ for a polynomial ideal J is minimal when: i) $\forall g_i \in G$, $lc(g_i) = 1$; ii) $\forall \ i \neq j$, $\text{lm}(g_i)$ does not divide $\text{lm}(g_j)$.

To obtain a minimal GB, all polynomials g_j are removed from G if there exists a g_i such that $lm(g_i) \mid lm(g_j)$. Then the remaining elements (g_i's) are made monic by dividing each g_i by $lc(g_i)$. This minimal basis is further *reduced* by ensuring that *no term* in g_j is divisible by the leading term $lt(g_i)$ for all $i \neq j$. Subject to $>$, the reduced Gröbner basis $G = \{g_1, \ldots, g_t\}$ is a **unique canonical representation 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 canonical word-level abstraction polynomial $Z = \mathcal{F}(A)$ for the circuit C. As discussed before, one such abstraction exists as a polynomial function over the Galois field $\mathbb{F}_{2^k}$. We now describe a Gröbner basis approach to derive the abstraction polynomial.

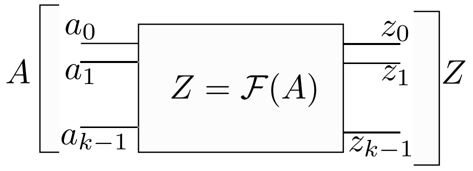


Fig. 3: Polynomial abstraction from a circuit.

A. The Problem Formulation over $\mathbb{F}_{2^k}$

Based on the datapath size k, $q = 2^k$ is chosen to model the circuit as a system of polynomials over $\mathbb{F}_q[x_1, \cdots, x_d, Z, A]$, where $\{x_1, \cdots, x_d\}$ 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 $\mathbb{F}_2$ ($\subset \mathbb{F}_{2^k}$):

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

For example, let $c = a \wedge b$ represent an AND gate. Over $\mathbb{F}_2$, this corresponds to the equation $c = a \cdot b$; its polynomial form is $c - a \cdot b$, or equivalently $c + a \cdot b$ since $-1 = 1$ over $\mathbb{F}_2$.

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 $J = \langle f_1, \cdots, f_s, f_A, f_Z \rangle$. The (unknown) word-level abstraction of the circuit $Z = \mathcal{F}(A)$ can be represented as the *"specification"* (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 3.3. Strong Nullstellensatz over Galois fields (Theorem 3.2) 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 $\mathbb{F}_q[x_1, \cdots, x_d, Z, A]$. 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 that:

Proposition 4.1: The (unknown) abstraction polynomial $f : Z + \mathcal{F}(A)$ is a member of the ideal $J + J_0$.

B. 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 4.1: (*Elimination Ideal*) From [7]: Given $J = \langle f_1,\dots,f_s\rangle \subset \mathbb{F}_q[x_1,\dots,x_d]$, the lth *elimination ideal* J_l is the ideal of $\mathbb{F}_q[x_{l+1},\dots,x_d]$ defined by: $J_l = J\cap\mathbb{F}_q[x_{l+1},\dots,x_d]$.

In other words, the lth elimination ideal does not contain variables $x_1,\dots,x_l$, 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 lexicographic (lex) ordering, which features into the theorem:

Theorem 4.1: (*Elimination Theorem*) From [7]: Let $J \subset \mathbb{F}_q[x_1,\dots,x_d]$ 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 $0 \le l \le d$, the set $G_l = G\cap\mathbb{F}_q[x_{l+1},\dots,x_d]$ is a Gröbner basis of the lth elimination ideal J_l.

Example 4.1: 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 4.1, $G_1 = G\cap\mathbb{C}[y,z] = \{g_2,g_3,g_4\}$ 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.

The above example motivates our approach: since we want to derive a polynomial representation from a circuit in variables Z,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 $G_d = G\cap\mathbb{F}_q[Z,A]$ of the d^{th} 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 4.1: 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 $A = \{a_0,\dots,a_{k-1}\}$ and $Z = \{z_0,\dots,z_{k-1}\}$ 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 $x_i, i = 1,\dots,d$, all the Boolean (bit-level) variables of the circuit. Let $R = \mathbb{F}_{2^k}[x_i,Z,A : i = 1,\dots d]$ denote the corresponding polynomial ring and let ideal $J = \langle f_1,\dots,f_s,f_A,f_Z\rangle \subset \mathbb{F}_{2^k}[x_i,Z,A : i = 1\dots d]$ be generated by the bit-level polynomials of the circuit ($f_1,\dots,f_s$) and the word-level designation polynomials (f_A,f_Z). Let $J_0 = \langle x_i^2-x_i, Z^{2^k}-Z, A^{2^k}-A : i = 1,\dots,d\rangle$ denote the ideal of vanishing polynomials in R. □

We define the following term order for this purpose of abstraction:

Definition 4.2: Abstraction Term Order $>$: Using the variable order $\{x_1,\cdots,x_d\} > Z > A$, impose a lex term order $>$ on $\mathbb{F}_q[x_1,\dots,x_d,Z,A]$. This term order is defined as the **Abstraction Term Order** (ATO) $>$. The relative ordering among the bit-level circuit variables $x_1,\dots,x_d$ is not important and can be chosen arbitrarily.

Theorem 4.2: Abstraction Theorem: Using the setup and notations given in Problem Setup 4.1, compute a Gröbner basis G of ideal $J+J_0$ using ATO $>$. Then:

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:

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 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 $\mathbb{F}_q$. 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 4.1. Therefore, according to the definition of a Gröbner basis (Definition 3.4), 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 $lt(g_i)$ can divide Z is when $lt(g_i) = Z$ 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 4.1: Let $G_{red} = \{g_1,\dots,g_t\}$ denote the *reduced* Gröbner basis of $J+J_0$ w.r.t. 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.

Proof: Assume that there are more than one polynomials in G_{red} containing only variables Z and A. According to Theorem 4.2, one of these polynomials is $g_i : Z+\mathcal{G}(A)$. Clearly, $lt(g_i) = Z$ divides the leading term of all other polynomials g_j in variables (Z,A), as $Z > A$ in ATO. All such polynomials g_j's are redundant and eliminated from the basis when it is reduced to G_{red}. Therefore, only one polynomial of the type $g_i : Z+\mathcal{F}(A)$ appears in the reduced basis.

Moreover, due to the presence of vanishing polynomials $A^q - A \in J + J_0$, $Z + \mathcal{F}(A)$ will be reduced $\pmod{A^q - A}$. Consequently, $Z + \mathcal{F}(A)$ results as the unique, reduced, canonical word-level polynomial representation of the circuit. ■

As a consequence of Theorem 4.2 and Corollary 4.1, if we compute a reduced Gröbner basis G_{red} of $J + J_0$ using the abstraction term order, we will always find the *one and only polynomial* of the form $Z + \mathcal{F}(A)$ in the basis, such that $Z = \mathcal{F}(A)$ is the unique canonical polynomial representation of the circuit. 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. The application of this approach is demonstrated using the example shown below.

Example 4.2: Consider the circuit of Fig. 4. Variables a_0, a_1, b_0, b_1 are primary inputs, z_0, z_1 are primary outputs, and c_0, c_1, c_2, c_3, r_0 are intermediate variables. 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 polynomial derived from the circuit. To construct $\mathbb{F}_{2^2}$, we use the primitive polynomial $P(x) = x^2 + x + 1$, with $P(\alpha) = 0$.

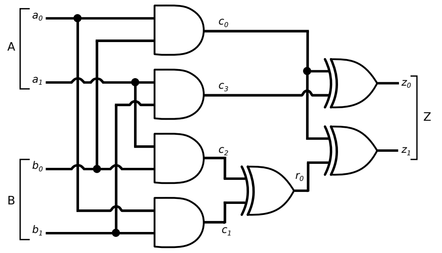


Fig. 4: An arbitrary circuit modeled over $\mathbb{F}(2^2)$.

With the mapping rules given in Eqn. (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 $J = \langle f_1, \cdots, f_7, f_A, f_B, f_Z \rangle$ 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 $J_0 = \langle f_8, \cdots, f_{21} \rangle$, and $J + J_0$ is simply $\langle f_1, \cdots, f_{21}, f_A, f_B, f_Z \rangle$.

Impose the following abstraction term order, 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, B which represents the polynomial function of the circuit C as $Z = (\alpha + 1) \cdot A^2 \cdot B^2$.

C. 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 $k = LCM(m, n)$, by virtue of the following result [39].

Lemma 4.1: The field $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ when $n \mid k$.

By selecting $k = LCM(m, n)$, 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 by means of the following result [39], which can be easily derived by analyzing the multiplicative group structure of the fields:

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

By replacing β and γ in terms of α in Eqn. (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 4.3: 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

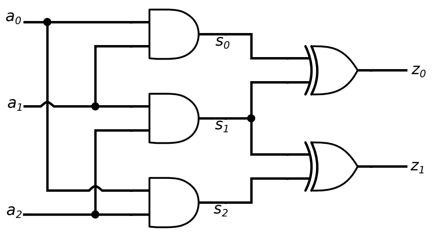


Fig. 5: Abstraction over a circuit with varying word sizes.

over $\mathbb{F}_{2^6}$ since $LCM(2,3)=6$, i.e. $\mathbb{F}_{2^3}$ and $\mathbb{F}_{2^2}$ are subsets of $\mathbb{F}_{2^6}$. Choose $P(X)=X^6+X+1$ as the primitive polynomial to construct this field, where $P(\alpha)=0$, 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:

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

where $\alpha^6+\alpha+1=0$. Simulating this polynomial for all $A\in\mathbb{F}_{2^3}$ results in values of $Z\in\mathbb{F}_{2^2}$ corresponding exactly to the function implemented by the circuit.

We have utilized this generalization of our model for verification of composite field arithmetic circuits.

Preliminary experiments: Using the results of Theorem 4.2 and Corollary 4.1, we performed some proof-of-concept experiments to evaluate the efficacy of our approach to abstraction. We experimented particularly with Galois field Mastrovito multipliers, and employed the SINGULAR computer algebra tool [40] 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 $\mathbb{F}_q[x_1,\dots,x_d]$ is known to be bounded by $q^{O(d)}$ [27], 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 $Spoly(f,g)\xrightarrow{J+J_0}_+ r$ 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 5.1: [Product Criterion [41]] Let $\mathbb{F}$ be any field, and $f,g\in\mathbb{F}[x_1,\cdots,x_d]$ be polynomials. If the equality $lm(f)\cdot lm(g)=LCM(lm(f),lm(g))$ holds, then $Spoly(f,g)\xrightarrow{G}_+ 0$.

The above result states that when the leading monomials of f,g are relatively prime, then $Spoly(f,g)$ always reduces to 0 modulo the basis G. Thus $Spoly(f,g)$ corresponding to the critical pair (f,g) need not be considered in Buchberger's algorithm. Recall that in the Abstraction Term Order (Definition 4.2), the relative ordering among the bit-level circuit variables $x_1,\dots,x_d$ is unimportant. This ordering is now further refined to exploit the product criteria. For this purpose, we draw inspirations from Proposition 2 in [29] 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 5.1: Refined Abstraction Term Order $>_r$: 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 $\mathbb{F}_q[x_1,\dots,x_d,Z,A]$ with the "bit-level variables $x_1,\dots,x_d$ ordered reverse topologically" $>Z>A$. This term order $>_r$ is called the **refined abstraction term order (RATO)**.

Denote $F=\{f_1,\dots,f_s,f_A,f_Z\}$ 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 $f_1,\dots,f_s$ 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, $Spoly(f_i,f_j)\xrightarrow{F\cup F_0}_+ 0$ 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 $f_i=x_i+tail(f_i)$, there exists a bit-level vanishing polynomial $x_i^2+x_i$. While their leading terms are not relatively prime, it was shown in Theorem 6.1 in [16] that $Spoly(f_i,x_i^2+x_i)\xrightarrow{F\cup F_0}_+ 0$. To show this, let us denote $tail(f_i)=P_i$ so that $f_i=x_i+P_i$. Also, every variable x_j that appears in P_i satisfies $x_i>x_j$. Then $Spoly(f_i,x_i^2+x_i)=x_iP_i+x_i$, which can be reduced by the polynomial $f_i\in F$:

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

Note that since $P_i=tail(x_i)$ 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 $Spoly(f_i=x_i+tail(f_i),\ x_i^2+x_i)\xrightarrow{F\cup F_0}_+ 0$ for all $i=1,\dots,d$*; 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 $Spoly(f_Z, f_{z_i}) \xrightarrow{F\cup F_0}_+ r$ results in a new polynomial in the Gröbner basis computation. Here: i) 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 ii) the polynomial $f_{z_i} = z_i + tail(f_{z_i})$ models the function implemented at the gate, and $lt(f_{z_i}) = z_i$.

So let us analyze the remainder r obtained as $Spoly(f_Z, f_{z_i}) \xrightarrow{F\cup F_0}_+ r$. Due to RATO, r does not contain any *bit-level non-primary input variables* of the circuit C, and it may only depend upon: i) the word-level variables (Z,A), and ii) the primary input bits $(a_0,\ldots,a_{k-1})$. To show this, assume that r contains a bit-level non-primary input variable x_j in a term m_j. Since there exists a polynomial $f_j = x_j + tail(f_j) \in F$, $lt(f_j) \mid m_j$, and all such terms m_j will be canceled during the reduction $Spoly(f_Z, f_{z_i}) \xrightarrow{F\cup F_0}_+ r$. Variables Z, 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 that:

Proposition 5.1: Due to RATO, (f_Z, f_{z_i}) is the only candidate critical pair to be evaluated as $Spoly(f_Z, f_{z_i}) \xrightarrow{F, F_0}_+ r$ 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 5.1: Let us revisit Example 4.2 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 4.2 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 5.1, computing* $Spoly(f_Z, f_6) \xrightarrow{F\cup F_0}_+ r$*, 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, B*, and the bit-level primary inputs* a_1, b_1*. Intermediate bit-level variables (non primary inputs) do not appear in r.*

A. Eliminating Bit-Level Variables

The remainder r obtained in Prop. 5.1 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:

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

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 5.2: (From [39]) *Let* $\alpha_1,\ldots,\alpha_t$ *be any 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 all integers* $i \geq 1$.

Lemma 5.2 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 $0 \leq j < k$:

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

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

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

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 Eqn. (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 i^{th} 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 $\mathbf{M}$ in Eqn. (8) exhibits a special structure. 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 5.2: Let $\mathbf{V}(x_1,\ldots,x_n)$ *denote a square* $n \times n$ *matrix of the form*

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

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

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

This determinant is non-zero if each $x_i \in \{x_1, \dots, x_n\}$ is a distinct element.

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

Lemma 5.3: For matrix $\mathbf{M}$ in Eqn. (8), $|\mathbf{M}| = 1$ over $\mathbb{F}_{2^k}$.

Proof: $\mathbf{M}$ is a Vandermonde matrix of the form $\mathbf{V}(\alpha, \alpha^2, \alpha^4, \dots, \alpha^{2^{k-1}}) = \mathbf{V}(\alpha^{2^0}, \alpha^{2^1}, \alpha^{2^2}, \dots, \alpha^{2^i}, \dots, \alpha^{2^{k-1}})$. The Vandermonde determinant $|\mathbf{V}(\alpha, \alpha^2, \alpha^4, \dots, \alpha^{2^{k-1}})|$ is non-singular if the elements $\alpha, \alpha^2, \alpha^4, \dots, \alpha^{2^{k-1}}$ are distinct (Definition 5.2). 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 non-zero element. Then by Definition 5.2, $|\mathbf{M}| \neq 0$.

Moreover, from Eqn. (11) it follows that:

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

Computing $|\mathbf{M}|^2$, and applying Lemma 5.2 gives:

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

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 $|\mathbf{M}| = 1$. ■

This result is demonstrated through the help of an example.

Example 5.2: Over $\mathbb{F}_{2^3}$, let $A = a_0 + a_1\alpha + a_2\alpha^2$, and so $A^2 = a_0 + a_1\alpha^2 + a_2\alpha^4$ and $A^4 = a_0 + a_1\alpha^4 + a_2\alpha^8$. From these equations, the coefficient matrix is derived to be:

$$\mathbf{M} = \begin{bmatrix} 1 & \alpha & \alpha^2 \\ 1 & \alpha^2 & \alpha^4 \\ 1 & \alpha^4 & \alpha^8 \end{bmatrix} \tag{14}$$

Since $\mathbf{M}$ is a Vandermonde matrix of the form $V(\alpha, \alpha^2, \alpha^4)$, its determinant is found by applying Eqn. (11):

$$\begin{aligned} |\mathbf{M}| &= (\alpha^4 - \alpha^2) \cdot (\alpha^4 - \alpha) \cdot (\alpha^2 - \alpha) \\ &= (\alpha^4 + \alpha^2) \cdot (\alpha^4 + \alpha) \cdot (\alpha^2 + \alpha) \end{aligned} \tag{15}$$

Note that $|\mathbf{M}|$ is non-zero since it is a product of non-zero terms. Now compute $|\mathbf{M}|^2$ while applying Lemma 5.2:

$$\begin{aligned} |\mathbf{M}|^2 &= [(\alpha^4 + \alpha^2) \cdot (\alpha^4 + \alpha) \cdot (\alpha^2 + \alpha)]^2 \\ &= (\alpha^8 + \alpha^4) \cdot (\alpha^8 + \alpha^2) \cdot (\alpha^4 + \alpha^2) \\ &= (\alpha + \alpha^4) \cdot (\alpha + \alpha^2) \cdot (\alpha^4 + \alpha^2) \end{aligned} \tag{16}$$

as $\alpha^8 = \alpha$ over $\mathbb{F}_{2^3}$. So $|\mathbf{M}|^2 = |\mathbf{M}|$. Since $|\mathbf{M}| \neq 0$, $|\mathbf{M}| = 1$, as no other element of $\mathbb{F}_{2^3}$ can satisfy this condition. Indeed, evaluating Eqns. (15) and (16) $\pmod{P(\alpha)}$ based on the chosen primitive polynomial $P(x)$ results in $|\mathbf{M}|^2 = |\mathbf{M}| = 1$.

Applying Lemma 5.3 to Eqn. (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 of the circuit k (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 i^{th} column, it follows that computing the determinant $|\mathbf{M_i}|$ results in polynomials of the form $a_i = c_{i_0}A + c_{i_1}A^2 + c_{i_2}A^4 + \dots + c_{i_{k-1}}A^{2^{k-1}}$, where $\{c_{i_0}, \dots, c_{i_{k-1}}\} \in \mathbb{F}_{2^k}$.

B. The 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.
2) Choose a *primitive* polynomial $P(x)$ of degree k and construct $\mathbb{F}_{2^k}$, and let $P(\alpha) = 0$.
3) Perform a reverse-topological traversal of C to derive RATO: lex with $\{x_1 > x_2 > \dots > x_d > Z > A\}$, where $\{x_1, \dots, x_d\}$ are bit-level variables of C.
4) 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 $f_i : x_i + tail(f_i)$ where x_i is the output of the corresponding logic gate.
5) Compose the bit-level to word-level polynomial correspondences: $f_A : a_0 + a_1\alpha + \dots + a_{k-1}\alpha^{k-1} + A$; $f_Z : z_0 + z_1\alpha + \dots + z_{k-1}\alpha^{k-1} + Z$. Denote $F = \{f_1, \dots, f_s, f_A, f_Z\}$. Compose the set of vanishing polynomials $F_0 = \{x_i^2 + x_i, A^{2^k} + A, Z^{2^k} + A\}$.
6) Select *the only critical pair* (f_Z, f_{z_i}) in F that does not have relatively prime leading terms. Compute $Spoly(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_{+} r$.
7) 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 $\mathbf{M} = \mathbf{V}(\alpha, \alpha^2, \dots, \alpha^{2^{k-1}})$.
8) 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 $Spoly(f_Z, f_{z_i}) \xrightarrow{F \cup F_0}_{+} r$, it can be performed in parallel with Step 6.
9) 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 5.3: We demonstrate the application of our approach on the circuit of Fig. 4. In Example 5.1, we have already shown that by imposing RATO and performing the reduction $Spoly(f_Z, f_6) \xrightarrow{F, F_0}_{+} r$ 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

$$\mathbf{M} = \begin{bmatrix} 1 & \alpha \\ 1 & \alpha^2 \end{bmatrix} \qquad \mathbf{M_1} = \begin{bmatrix} 1 & A \\ 1 & A^2 \end{bmatrix} \tag{17}$$

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 unique canonical polynomial representation of the circuit, the same as the one derived by computing a full reduced Gröbner basis in Example 4.2.

Complexity of our approach: In our approach, Steps 6), 8) and 9) are the main computationally intensive procedures. Given d variables in the polynomial ring with coefficients in $\mathbb{F}_{2^k}$, we have shown that: i) the worst-case complexity of Step 6) is $O(2^{2d})$; ii) the complexity of Step 8) is polynomial in k; and iii) the complexity of Step 9) is $O(2^{3k})$. Steps 6) and 8) are independent and these are computed in parallel. Moreover, since $d >> k$, the complexity of Step 6) subsumes that of Step 8). So the worst-case complexity of the abstraction algorithm is $O(2^{2d}) + O(2^{3k})$. Interested readers may refer to Theorem 6.3 in [42] for a detailed proof.

VI. Experimental Results

Using the approach described in Section V, we have performed experiments to abstract canonical word-level representations of galois field multiplier circuits of various designs. The abstraction procedure described above can be scripted using the computer algebra tool SINGULAR [40]. However, SINGULAR has serious limitations that make abstractions of large circuits *impossible*. This is due to the following reasons: i) SINGULAR limits the number of variables to 32,767 in the ring; ii) the size of the exponent (l) of a variable (x^l) is limited to $l < 2^{32}$, whereas our approach requires large exponents (e.g. $A^q + A$); and iii) the (dense-distributive) data structures used by SINGULAR are not specifically designed for circuit verification problems, resulting in large memory usage and slow computation time. In practice, SINGULAR is infeasible for word-level abstraction beyond 32-bit circuits. For this reason, we have developed our own custom abstraction tool in C++, available at http://www.ece.utah.edu/~pruss/abstract.html.

A. Custom Tool Implementation

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. Given any primitive polynomial of some degree k, the library facilitates the construction and manipulation of elements of the corresponding Galois field $\mathbb{F}_{2^k}$. Any element $C \in \mathbb{F}_{2^k}$ can be represented in the form $C = c_{k-1}\cdot\alpha^{k-1} + \cdots + c_2\cdot\alpha^2 + c_1\cdot\alpha + c_0$ where $\{c_0,\dots,c_{k-1}\} \in \mathbb{F}_2$ and α is the primitive element. This structure is stored as an unsigned byte array containing $\{c_0,\dots,c_{k-1}\}$, as shown in Fig. 6.

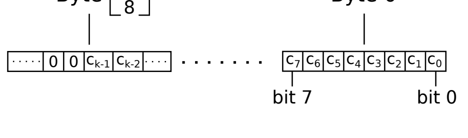


Fig. 6: Object structure of a Galois field element

This compact structure allows addition of any two elements to be computed as a byte-wise XOR operation. The library also supports multiplication and division. The extended Euclidean algorithm is implemented to compute inverses over $\mathbb{F}_{2^k}$.

A monomial M over the ring $\mathbb{F}_{2^k}[x_1,\dots,x_d]$ is 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. Each variable is given a unique unsigned integer ID, which is used to order the terms. Each monomial object contains: i) a Galois field library object; ii) a set of IDs of all variables in the monomial; and iii) a map of variable IDs to their exponents. The exponent mapping is only used for word-level variables as bit-level variables can have an exponent of at most 1. Since exponents can be much larger than what can be stored in a primitive data structure, each exponent is stored as a BigUnsigned object of the open source library BigInt [43], 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.

Example 6.1: Consider the ring $\mathbb{F}_{2^4}[a,b,c,Z]$ with the lex ordering $a > b > c > Z$, where $\{a,b,c\}$ are bit-level variables and Z is a word-level variable, and $P(x) = x^4 + x^3 + 1$ with $P(\alpha) = 0$. Let $M_1 = (\alpha^3+\alpha^2+1)abZ^{10}$ and $M_2 = (\alpha^2+\alpha)bcZ^7$ be monomials. Variables a,b,c,Z are given IDs $0,1,2,3$, respectively. These monomials are stored as:

$$\begin{array}{ccc} & \mathbf{M_1} & \\ \textbf{coef} & \textbf{idSet} & \textbf{idToExp} \\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}\\\hline\end{array} & \{0,1,3\} & 3\rightarrow 10 \end{array} \tag{18}$$

$$\begin{array}{ccc} & \mathbf{M_2} & \\ \textbf{coef} & \textbf{idSet} & \textbf{idToExp} \\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}\\\hline\end{array} & \{1,2,3\} & 3\rightarrow 7 \end{array} \tag{19}$$

where **idToExp** *$3 \rightarrow 7$ implies that the exponent of Z is 7. Since the first element of the* **idSet** *is 0 in M_1 and 1 in M_2, it implies $M_1 > M_2$ in our ordering. $M_1\cdot M_2$ is computed by multiplying the coefficients, merging (union) their* **idSet**, *and adding the exponents of Z (***idToExp***). The exponents are reduced accordingly as $b^2 = b$ and $Z^{16} = Z$, resulting in $M_1\cdot M_2 = (\alpha^2+1)abcZ^2$.*

$$\begin{array}{ccc} & \mathbf{M_1\cdot M_2} & \\ \textbf{coef} & \textbf{idSet} & \textbf{idToExp} \\ \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}\\\hline\end{array} & \{0,1,2,3\} & 3\rightarrow 2 \end{array} \tag{20}$$

Finally, polynomial division $f \xrightarrow{F\cup F_0}_+ r$ is implemented as *Faugére's* $F4$-style reduction [16]. The current implementation of this $F4$-style reduction is a significant improvement over our previous one (Section VII in [16]), which did not provide support for coefficients or exponents beyond $\{0,1\}$.

B. Results

All experiments are conducted on a 64-bit Linux desktop with a 3.5GHz Intel Core$^{\text{TM}}$ i7 Quad-core CPU and 16 GB of RAM. Table II depicts the time and memory required to derive the polynomial abstraction from bug-free and buggy Mastrovito 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 $k = 163, \dots, 571$ corresponding to the NIST ECC specification. Bugs are introduced by interchanging some gates and wires at the output nodes, ensuring that their effect propagates down during the polynomial reduction process.

TABLE II: Abstraction of Mastrovito multipliers. Time given in seconds, memory given in MB. TO = timeout 3 days (259,200 sec).

Field Size (k)		163	233	283	409	571
# of Gates		153K	167K	399K	508K	1.6M
Time (s)	Bug Free	1,443	1,913	11,116	17,848	192,032
	Buggy	1,487	2,106	11,606	20,263	204,194
Max Memory (MB)		213	269	561	845	2,855

Table III depicts the results for abstraction of *flattened Montgomery* multipliers. 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 IV. 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. While each block is an MR block, some have been simplified by constant-propagation, hence they have different gate-counts. First, a polynomial is extracted for each MR block (gate-level to word-level abstraction), and then the approach is re-applied at word-level to derive the input-output relation (solved trivially in < 1 sec). Our approach can extract the 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.

TABLE III: Abstraction of flat Montgomery multipliers. Time given in seconds, memory given in MB. TO = 3 days (259,200 sec).

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 IV: Abstraction of Montgomery blocks. Time given in seconds, memory is given in MB. TO = 3 days (259,200 sec).

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

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 $\mathbb{F}_{2^m}$, $A_0, A_1, \dots, A_{n-1}$, 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. Addition over $\mathbb{F}_{2^m}$ is a bit-wise XOR operation, so this abstraction is trivial. The m-bit multipliers are designed using the Mastrovito-style and their word-level abstractions are derived in similar fashion as the experiments of Table II. Each abstraction can be 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 Eqns. (7)-(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 V. In these experiments, bugs are introduced in the high-level interconnections of the $\mathbb{F}_{2^m}$ blocks, and the $\mathbb{F}_{2^m}$ Mastrovito blocks themselves are kept bug-free.

The above experiments are conducted for abstraction of building-blocks (ADD, MULT) for finite field arithmetic computations. The following experiment is conducted to evaluate the efficacy of our abstraction procedure when these 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 Eqn. (2)); these polynomials were synthesized/optimized and then abstraction for each word-level output X_3, Y_3, 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 VI, where abstraction is successful for up to 409-bit fields, but infeasible for the 571-bit circuit.

TABLE VI: Time and memory per abstraction of point addition circuits. Time given in seconds, memory given in MB. TO = 3 days (259,200 seconds.)

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

The above experiments also demonstrate that we can perform equivalence checking between different circuit implementations by deriving canonical word-level polynomials (Z_1, Z_2) from each circuit independently and then checking if $Z_1 = Z_2$. In the case of Mastrovito (golden model) and Montgomery (implementation) multipliers, our approach can verify their equivalence for up to $k = 571$ bit circuits. *In contrast, contemporary equivalence checking approaches (SAT, SMT, and AIG/ABC) cannot verify their equivalence even for $k = 16$ bits within the timeout limit of 3 days (results omitted for this reason).*

However, simulation, SAT and SMT-based approaches show success with *bug-catching* in equivalence check between a golden model and a buggy implementation. These experiments

$k = 128$				
m	n	Time		Max
		Bug Free	Buggy	Mem
2	64	1	1	4
4	32	1	1	2
8	16	1	1	2
16	8	1	1	2
32	4	1	1	2
64	2	1	1	3
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-

$k = 256$				
m	n	Time		Max
		Bug Free	Buggy	Mem
2	128	15	15	23
4	64	2	2	4
8	32	1	1	3
16	16	1	1	2
32	8	1	1	2
64	4	1	1	2
128	2	1	1	2
-	-	-	-	-
-	-	-	-	-

$k = 512$				
m	n	Time		Max
		Bug Free	Buggy	Mem
2	256	406	408	90
4	128	53	53	25
8	64	8	8	4
16	32	2	2	4
32	16	1	1	3
64	8	1	1	3
128	4	1	1	2
256	2	1	1	2
-	-	-	-	-

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

TABLE V: Abstraction of composite field multipliers over $\mathbb{F}_{(2^m)^n}$. Time given in seconds, memory in MB.

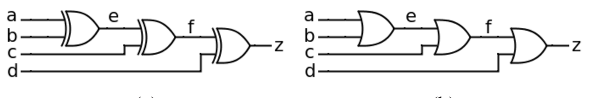
are shown in Table VII, where a Montgomery multiplier implementation is mitered and verified against a Mastrovito golden model. The execution time for bug-catching is reported in the columns. The AIG-based ABC tool outperforms all other solvers. The simulation engine of ABC (employed for FRAIGing) is able to detect bugs in the miter relatively quickly.

TABLE VII: Bug-catching between a golden-model Mastrovito and buggy Montgomery circuit using simulation, SAT and SMT-solvers. Time given in seconds, TO = 3 days (259,200 seconds).

Circuit Size (k)	163	233	283	409	571
ABC	32	6	96	217	401
Lingeling	8	362	12,728	3,323	23,298
Picosat	TO	TO	TO	TO	TO
Boolector	30	41	105	152	19,113
CVC4	11	64	8,660	280	TO
Z3	12	55	10,169	335	TO
Yices	6	7	618	578	11,568

C. 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.



(a) (b)

Fig. 7: Polynomial division comparison

Example 6.2: Consider the circuit of Figure 7 (a), apply RATO: lex with $z > f > d > e > c > b > a$. Reduce z modulo the polynomials corresponding to the circuit:

$$f_1 : z+f+d \quad f_2 : f+e+c \quad f_3 : e+b+a$$

The reduction procedure $z \xrightarrow{f_1,f_2,f_3}_+ r$ will be computed as: $z \xrightarrow{z+f+d} f+d \xrightarrow{f+e+c} e+d+c \xrightarrow{e+b+a} d+c+b+a$. In each reduction, a variable corresponding to a gate output is removed and one copy of each input variable is added, leaving a sparse polynomial. Now consider the same circuit with the XOR gates replaced by OR gates, as shown in Figure 7 (b). The monomial ordering stays the same, but the polynomials derived from each gate have changed:

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

The reduction procedure, $z \xrightarrow{f_1,f_2,f_3}_+ r$ is now computed as: $z \xrightarrow{z+fd+f+d} fd+f+d \xrightarrow{f+ec+e+c}_+ edc+ed+ec+e+dc+d+c \xrightarrow{e+ba+b+a}_+ dcba+dcb+dca+dba+dc+db+da+d+cba+cb+ca+c+ba+b+a = r$. 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.

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 $f : \mathbb{B}^k \to \mathbb{B}^k$ as a polynomial function $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$. 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 $Z = \{z_0, \dots, z_{k-1}\}$, $A = \{a_0, \dots, a_{k-1}\}$ are the k-bit inputs and outputs, respectively. The approach is then generalized to circuits with an arbitrary number of inputs (m) and outputs (n), by modeling the function over $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, where $k = LCM(m,n)$. 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.

The function $f : \mathbb{B}^k \to \mathbb{B}^k$ can also be construed as a mapping over $f : \mathbb{Z}_{2^k} \to \mathbb{Z}_{2^k}$. However, over finite integer rings $\mathbb{Z}_{2^k}$, not every function is a polynomial function. As the concepts of elimination ideals and Gröbner bases are applicable over rings too, it is *conjectured* that the abstractions so derived may provide an *over-approximation* of the function implemented by the circuit. These issues are currently under investigation for the purpose of abstraction and verification of integer arithmetic circuits and RTL designs.

References

[1] A. Peymandoust and G. DeMicheli, "Application of Symbolic Computer Algebra in High-Level Data-Flow Synthesis," *IEEE Transactions CAD*, vol. 22, no. 9, pp. 1154–11 656, 2003.

[2] J. Smith and G. DeMicheli, "Polynomial Methods for Allocating Complex Components," in *IEEE Design, Automation and Test in Europe (DATE) Conf.*, 1999.

[3] P. Dasgupta, P. Chakrabarti, A. Nandi, S. Krishna, and A. Chakrabarti, "Abstraction of linear word-level arithmetic functions from bit-level component descriptions," in *Des. Auto. & Test Eur.*, 2001, pp. 4–8.

[4] A. Griggio, "Effective word-level interpolation for software verification," in *Formal Methods in CAD*, 2011, pp. 28–36.

[5] B. Brady, R. Bryant, S. Seshia, and J. O'Leary, "ATLAS: Automatic Term-level abstraction of RTL designs," in *Proc. Memocode*, 2010, pp. 31–40.

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

[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] R. Lidl and H. Niederreiter, *Finite Fields*. Cambridge University Press, 1997.

[9] Z. Zilic and Z. Vranesic, "A deterministic multivariate interpolation algorithm for small finite fields," *IEEE Trans. Comp.*, vol. 51, no. 2, 2002.

[10] C. Paar, "A New Architecture for A Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 856–861, July 1996.

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

[12] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," *IEEE Transactions On Computers*, vol. 51, no. 5, May 2002.

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

[14] E. Biham, Y. Carmeli, and A. Shamir, "Bug Attacks," in *Proceedings on Advances in Cryptology*, 2008, pp. 221–240.

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

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

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

[18] 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, no. 9, pp. 1188–1201, 2006.

[19] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing," *IEEE Transactions on CAD*, vol. 26, no. 7, pp. 1320–1330, July 2007.

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

[21] A. Jabir and *et al.*, "A Technique for Representing Multiple Output Binary Functions with Applications to Verification and Simulation," *IEEE Trans. on Comp.*, vol. 56, no. 8, pp. 1133–1145, 2007.

[22] F. Lu, L. Wang, K. Cheng, and R. Huang, "A Circuit SAT Solver With Signal Correlation Guided Learning," in *IEEE Design, Automation and Test in Europe*, 2003, pp. 892–897.

[23] L. Lastras-Montaño, P. Meany, E. Stephens, B. Trager, J. O'Conner, and L. Alves, "A new class of array codes for memory storage," in *Proc. Information Theory and Applications Workshop*, 2011, pp. 1–10.

[24] L. Lastras, A. Lvov, B. Trager, S. Winograd, V. Paruthi, A. El-Zhein, R. Shadowen, and G. Janssen, "New Formal Verification Techniques for Algorithms over Finite Fields," 2012, presented at Intl. Workshop on Internation Theory and Applications. Abstract of the paper available at: http://ita.ucsd.edu/workshop/12/talks.

[25] 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)*, 2012, pp. 141–148.

[26] J. Lv, "Scalable Formal Verification of Finite Field Arithmetic Circuits using Computer Algebra Techniques," Ph.D. dissertation, Univ. of Utah, Aug. 2012.

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

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

[29] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths," in *Computer Aided Verification Conference*, 2008, pp. 473–486.

[30] F. Farahmandi and B. Alizadeh, "Groebner basis based formal verification of large arithmetic circuits using Gaussian elimination and cone-based polynomial extraction," *Microprocessors and Microsystems*, vol. 39, no. 83-96, 2015.

[31] M. Ciesielski, W. Brown, D. Liu, and A. Rossi, "Function Extraction from Arithmetic Bit-level Circuits," in *Intl. Symp. VLSI (ISVLSI)*, 2014.

[32] M. Ciesielski and *et al.*, "Verification of Gate-Level Arithmetic Circuits by Function Extraction," in *Proc. Des. Auto. Conf. (DAC)*, 2015.

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

[34] J. L*ó*pez, R. Dahab, and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)." Springer-Verlag, 1998, pp. 201–212.

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

[36] Y. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1514–1527, Nov. 2008.

[37] B. Sunar, E. Savas, and C. Koc, "Constructing Composite Field Representations for Efficient Conversion," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1391–1398, November 2003.

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

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

[40] 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.

[41] B. Buchberger, "A criterion for detecting unnecessary reductions in the construction of a groebner bases," in *EUROSAM*, 1979.

[42] T. Pruss, "Word-Level Abstraction from Combinational Circuits using Algebraic Geometry," Ph.D. dissertation, University of Utah, May 2015.

[43] M. McCutchen, "C++ big integer library," https://mattmccutchen.net/bigint/, 2010.