FORMAL VERIFICATION OF SEQUENTIAL GALOIS FIELD ARITHMETIC CIRCUITS USING ALGEBRAIC GEOMETRY

Abstract – *Sequential Galois field ($\mathbb{F}_{2^k}$) arithmetic circuits take k-bit inputs and produce a k-bit result, after k-clock cycles of operation. Formal verification of sequential arithmetic circuits with large datapath size is beyond the capabilities of contemporary verification techniques. To address this problem, this paper describes a verification method based on algebraic geometry that: i) implicitly unrolls the sequential arithmetic circuit over multiple (k) clock-cycles; and ii) represents the function computed by the state-registers of the circuit, canonically, as a multi-variate word-level polynomial over $\mathbb{F}_{2^k}$. Our approach employs the Gröbner basis theory over a specific elimination ideal. Moreover, an efficient implementation is described to identify the k-cycle computation performed by the circuit at word-level. We demonstrate the feasibility of our approach by verifying up to 100-bit sequential Galois field multipliers, whereas conventional techniques fail beyond 23-bit circuits.*

I. INTRODUCTION

Galois field (GF) arithmetic finds application in areas such as cryptography, error control coding, VLSI testing, etc. In most hardware applications, fields of the type $\mathbb{F}_{2^k}$ are widely chosen. Such *binary* GFs are k-dimensional extensions of the base field $\mathbb{F}_2$; this allows for the design of efficient (AND-XOR) arithmetic architectures and algorithms for hardware design. Due to their deployment in communications and security-related applications, there is a critical need to *formally verify* the correctness of hardware implementations of GF arithmetic. In most applications, however, the datapath size (bit-vector operand size) k is very large. Most conventional formal verification methods are unable to cope with the large size and complexity of GF circuits.

GF arithmetic circuits over $\mathbb{F}_{2^k}$ take k-bit vectors as inputs and produce k-bit outputs. For example, a GF modulo multiplier computes $R = A \times B \pmod{P(x)}$, where: i) $A = a_{s(0)} + a_{s(1)}\alpha + \cdots + a_{s(k-1)}\alpha^{k-1} = \sum_{i=0}^{k-1} a_{s(i)}\alpha^i$, $B = \sum_{i=0}^{k-1} b_{s(i)}\alpha^i$ denote the k-bit inputs, $R = \sum_{i=0}^{k-1} r_{s(i)}\alpha^i$ is the output, and $a_{s(i)}, b_{s(i)}, r_{s(i)} \in \mathbb{F}_2$; ii) $P(x)$ is the given primitive polynomial used to construct $\mathbb{F}_{2^k}$; and iii) $P(\alpha) = 0$, i.e. α is the primitive element of the field. In the above, the elements are represented in *standard basis notation* (denoted by subscript "s" on the bits). As the datapath size k increases, combinational GF designs become prohibitively large; sequential GF circuits are therefore desirable.

Sequential GF circuits operate as follows: k-bit input operands are loaded into k-bit state registers (flip-flops), and the circuit is executed for k clock-cycles; after which the k-bit result is available in the output registers. Data representation and circuit design for sequential GF circuits is mostly based on *normal basis* [1]. Data is represented as $A = a_{n(0)}\beta + a_{n(1)}\beta^2 + a_{n(2)}\beta^{2^2} + \cdots + a_{n(k-1)}\beta^{2^{k-1}}$, where $\beta \in \mathbb{F}_{2^k}$ is the *normal element*, and $\{\beta, \beta^2, \ldots, \beta^{2^i}, \ldots, \beta^{2^{k-1}}\}$ forms the normal basis. Standard and normal basis representations can be derived from each other, i.e. β can be derived from α and vice-versa. It has been shown that architectures for multiplication and squaring can be efficiently designed using normal bases [2] [3] [1] as sequential circuits.

Example I.1. *For $a, b \in \mathbb{F}_{2^k}, (a+b)^2 = a^2 + b^2$. Applying this rule for element squaring:*

$$\begin{aligned} B =& (b_0\beta + b_1\beta^2 + b_2\beta^4 + \cdots + b_{k-1}\beta^{2^{k-1}}) \\ B^2 =& b_0^2\beta^2 + b_1^2\beta^4 + b_2^2\beta^8 + \cdots + b_{k-1}^2\beta^{2^k} \\ =& b_{k-1}\beta + b_0\beta^2 + b_1\beta^4 + \cdots + b_{k-2}\beta^{2^{k-1}} \end{aligned}$$

as $\beta^{2^k} = \beta$ due to Fermat's little theorem, and $b_i^2 = b_i$.

The above example shows that squaring of elements represented in normal bases can be implemented simply by a cyclic right-shift operation. However, multiplication of two elements in normal basis is still a complex operation — and its design and verification is still challenging. Recent literature has addressed formal equivalence proofs of *combinational* GF arithmetic circuits [4] [5] [6]. This paper addresses verification of sequential GF circuits, which has not been addressed before.

Problem Statement: We are given: i) the Galois field $\mathbb{F}_{2^k} = \mathbb{F}_2[X] \pmod{P(X)}, P(\alpha) = 0$, along with the *normal basis representation*, i.e. β is also known; ii) a word-level specification polynomial $R = \mathcal{F}(A, B) \pmod{P(X)}$; where $A = \sum_{i=0}^{k-1} a_i\beta^{2^i}$, $B = \sum_{i=0}^{k-1} b_i\beta^{2^i}$, $R = \sum_{i=0}^{k-1} r_i\beta^{2^i}$, $a_i, b_i, r_i \in \mathbb{F}_2$, and iii) a sequential circuit (S) implementation of the polynomial computation. Our objective is to prove or disprove that S is a k-cycle implementation of R.

Approach & Contributions: The sequential GF arithmetic circuit can be viewed as a *restricted* Mealy finite state machine (FSM), as shown in Fig. 1. The FSM contains no primary inputs or outputs. The operands are loaded into state registers A, B (as initial states), and after k-clock cycles of operation, the result $R = \mathcal{F}(A, B)$ is stored in the R register.

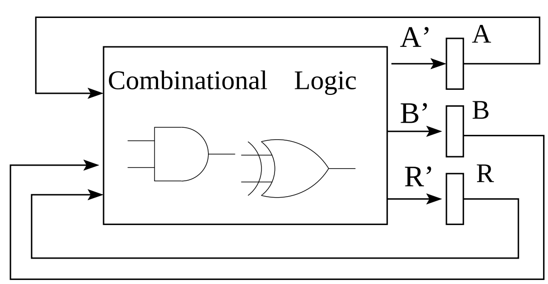

Fig. 1: A typical normal basis GF sequential circuit model. $A = (a_0, \ldots, a_{k-1})$ and similarly B, R are k-bit registers; A', B', R' denote next-state inputs.

A straight-forward approach to verify such a sequential circuit may consist of unrolling the circuit for k time-frames, and performing a (combinational) equivalence check between the unrolled machine and the specification polynomial. Such a technique is grossly inefficient for large circuits. Therefore, we propose a method that implicitly (symbolically) represents the unrolled computation, *canonically, as a word-level multivariate polynomial.* We show that the k-cycle polynomial representation can be derived *iteratively* by performing a sequence of Gröbner basis (*GB*) computations of the ideal generated by the polynomials corresponding to the circuit. The approach requires the use of a specific elimination term order for the *GB* computation, based on the circuit's topology. Once the canonical polynomial is derived, it can be checked against the specification polynomial for verification.

Computing Gröbner bases with elimination orders is infeasible for large circuits. To overcome this complexity, we draw inspirations from [6], and exploit the binomial expansion over GFs to engineer a new, efficient implementation to derive the word-level polynomial. We demonstrate the feasibility of our approach by verifying (and also detecting bugs in) up to 100-bit sequential GF multipliers (containing 300 flip-flops), whereas conventional techniques fail beyond 23-bit circuits. Finally, our approach can be construed as a word-level, implicit traversal of the underlying FSM of the sequential GF circuit, wherein the set of states is encoded as the variety of an elimination ideal related to the FSM's transition function.

Paper Organization: The paper is organized as follows. The following section reviews previous work, and contrasts it against the new contributions of this paper. Sec III briefly describes the architecture of the sequential normal basis GF multipliers verified in this paper. Section IV covers preliminary concepts and notation. Section V describes our approach, computational improvements for which are described in Section VI. Experimental results are described in Section VII. Finally, Section VIII concludes the paper.

II. Review of Previous Work

Verification of a combinational GF arithmetic circuit C against a polynomial specification $\mathcal{F}$ has been addressed in [4] [5] [6]. Verification problems in [4] [5] are formulated using Nullstellensatz and decided using the Gröbner basis algorithm.

The paper [6] performs verification by deriving a canonical word-level polynomial representation $\mathcal{F}$ from the circuit C. Their approach views any arbitrary Boolean function (circuit) $f:\mathbb{B}^k\rightarrow\mathbb{B}^k$ as a polynomial function $f:\mathbb{F}_{2^k}\rightarrow\mathbb{F}_{2^k}$, and derives a canonical polynomial representation $\mathcal{F}$ over $\mathbb{F}_{2^k}$. They show that this can be achieved by computing a reduced Gröbner basis w.r.t. an *abstraction term order* derived from the circuit. Subsequently, they propose a refinement of this abstraction term order (called RATO), that enables to compute the Gröbner basis of a smaller subset of polynomials. The authors show that their approach can prove correctness of up to 571-bit combinational GF multipliers.

Since we are also interested in deriving a polynomial representation of the computation performed by the sequential circuit S, we draw inspirations from [6] – particularly the use of RATO – for sequential verification. However, the approach of [6] suffers from a few limitations: i) When the GF polynomial implemented by the circuit is dense (say, due to the presence of a bug in the design), their approach is computationally infeasible; ii) The use of RATO still requires a Gröbner basis computation (even though on a subset of polynomials) to derive the polynomial $\mathcal{F}$, which can lead to a memory explosion. Experiments in [6] are only successful for hierarchically designed and bug-free GF circuits. *While we do employ RATO as the term order for sequential verification, we further present an efficient symbolic computation approach that does not suffer from these limitations*.

The problem addressed in this paper is not suitable to be solved by conventional bit-level sequential equivalence [7] or (bounded) model checking frameworks based on interpolation [8] or property directed reachability [9]. This is mostly due to the word-level and GF polynomial nature of the specification (property) $\mathcal{F}$, which is also only valid in one state of the machine. The use of algebraic geometry has been proposed for model checking [10] [11] [12]; however, these approaches are a straight-forward application of *bit-level* Boolean Gröbner basis engines in lieu of BDDs or SAT solvers.

III. Sequential GF Multiplier Design

Let us briefly describe the fundamentals behind the design of normal basis sequential GF multipliers, so as to put in perspective the type of designs that have been verified in this paper. Let $R=\sum_{i=0}^{k-1} r_i\beta^{2^i}$, $A=\sum_{i=0}^{k-1} a_i\beta^{2^i}$, $B=\sum_{i=0}^{k-1} b_i\beta^{2^i}$, then

$$R=A\cdot B=(\sum_{i=0}^{k-1} a_i\beta^{2^i})(\sum_{j=0}^{k-1} b_j\beta^{2^j})=\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} a_i b_j\beta^{2^i}\beta^{2^j}$$

The expressions $\beta^{2^i}\beta^{2^j}$ are called cross-product terms and they can also be represented in normal basis:

$$\beta^{2^i}\beta^{2^j}=\sum_{n=0}^{k-1}\lambda_{ij}^{(n)}\beta^{2^n},\quad \lambda_{ij}^{(n)}\in\mathbb{F}_2.$$

From the above two equations, one can see that the expression for the n^{th} digit of product $R=(r_0,\dots,r_n,\dots r_{k-1})$ is:

$$r_n=\sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\lambda_{ij}^{(n)}a_i b_j = A\cdot M_n\cdot B^T,\quad 0\leq n\leq k-1$$

where $M_n=(\lambda_{ij}^{(n)})$ is a binary $k\times k$ matrix over $\mathbb{F}_2$, and it is called the λ-matrix. Moreover, let $r_n=A\cdot M^{(n)}\cdot B^T$. Then $r_{n-1}=A\cdot M^{(n-1)}\cdot B^T=rotate(A)\cdot M^{(n)}\cdot rotate(B)^T$. This implies that $M^{(n)}$ is generated by right and down cyclic shifting $M^{(n-1)}$. Therefore, the hardware design of sequential GF multipliers is based on mappings of $A\cdot M_n\cdot B^T$ into AND-XOR gates and cyclic shift operations.

This paper verifies the implementation of two distinct architectures of *sequential multipliers with parallel output (SMPO)*, namely: i) the Agnew-SMPO [2] by G. B. Agnew, which is a straight-forward implementation of the λ-matrix; and ii) the more recent, more complicated, yet very efficient RH-SMPO [3], by Reyhani-Masoleh and Hasan, depicted in Fig. 2.

IV. Preliminaries

Let $q=2^k$ and $\mathbb{F}_q[x_1,\dots,x_d]$ be the polynomial ring with indeterminates $x_1,\dots,x_d$. A polynomial $f=c_1X_1+c_2X_2+\cdots+c_tX_t$ is a finite sum of terms, where $c_1,\dots,c_t$ are coefficients and $X_1,\dots,X_t$ are monomials. A monomial ordering

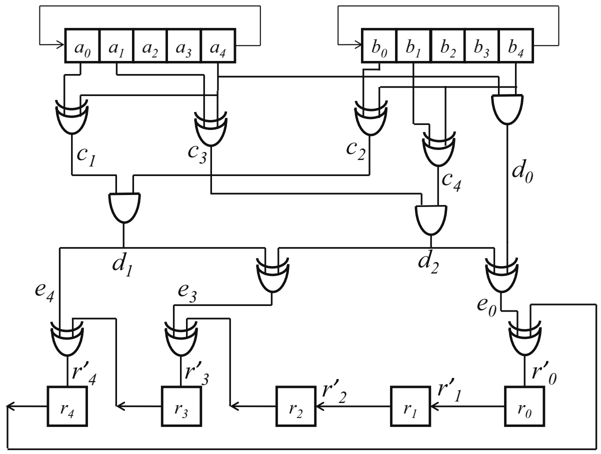

Fig. 2: A 5-bit RH-SMPO

$X_1 > X_2 \cdots > X_t$ is imposed on the polynomials to process them systematically. Then, $LT(f) = c_1X_1, LM(f) = X_1$ denote the leading term and the leading monomial of f, respectively. Given polynomials f, g, if cX is a term in f that is divisible by $LT(g)$, then $f \xrightarrow{g} r$ denotes a one-step reduction (division) of f by g, resulting in remainder $r = f - \frac{cX}{LT(g)} \cdot g$. Similarly, f reduces to r modulo the set of polynomials $F = \{f_1, \ldots, f_s\}$, denoted $f \xrightarrow{F}_+ r$, such that no term in r is divisible by the $LT(f_i)$ of any polynomial in $f_i \in F$.

Let $f_1, \ldots, f_s$ be polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$, then we set $J = \langle f_1, \ldots, f_s \rangle = \left\{\sum_{i=1}^{s} h_i f_i : h_1, \ldots, h_s \in \mathbb{F}_q[x_1, \ldots, x_d]\right\}$. $J = \langle f_1, \ldots, f_s \rangle$ forms an ideal in $\mathbb{F}_q[x_1, \ldots, x_d]$, and $f_1, \ldots, f_s$ are its generators. Let $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{F}_q^d$ be a point. For any ideal $J = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{F}_q[x_1, \ldots, x_d]$, the *affine variety* of J over $\mathbb{F}_q$ is: $V(J) = \{\mathbf{a} \in \mathbb{F}_q^d : \forall f \in J, f(\mathbf{a}) = 0\}$. The variety $V(J)$ corresponds to the set of all solutions to $f_1 = \cdots = f_s = 0$. Any finite set of points can be construed as the variety of an ideal — a concept we exploit to model the set of (finite) states of the machine.

An ideal may have many generating sets. The set $G = \{g_1, \ldots, g_t\}$ is called a Gröbner basis of J if and only if the leading term of all polynomials in J is divisible be the leading term of some polynomial g_i in G: i.e. $\forall f \in J, \exists g_i \in G\ s.t.\ LT(g_i) \mid LT(f)$. The famous Buchberger's algorithm, given in textbook [13], is used to compute a Gröbner basis (GB). Operating on input $F = \{f_1, \ldots, f_s\}$, and subject to the imposed term order $>$, it derives $G = GB(J) = \{g_1, \ldots, g_t\}$. Buchberger's algorithm repeatedly computes S-polynomials. For pairs $(f_i, f_j) \in F$, $Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j$, where $L = LCM(LM(f_i), LM(f_j))$. Reducing $Spoly(f_i, f_j) \xrightarrow{F}_+ r$ cancels the leading terms of f_i, f_j and gives a polynomial r with a new leading term. This remainder r is added to the current basis and $Spoly(f_i, f_j)$ computations are repeated for all pairs of polynomials until all S-polynomials reduce to 0.

Fermat's little theorem: For any $\alpha \in \mathbb{F}_q, \alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ vanishes $(= 0)$ over $\mathbb{F}_q$, and is called a vanishing polynomial. We denote by $J_0 = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$ the ideal of all vanishing polynomials in $\mathbb{F}_q[x_1, \ldots, x_d]$. When $q = 2^k, x^q - x = x^q + x$ as $-1 = +1$ over $\mathbb{F}_{2^k}$.

Gröbner bases can be used to *eliminate* variables from an ideal. Given ideal $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$, the l^{th} elimination ideal J_l is the ideal of $\mathbb{F}_q[x_{l+1}, \ldots, x_d]$ defined by $J_l = J \cap \mathbb{F}_q[x_{l+1}, \ldots, x_d]$. Variable elimination can be achieved by computing a Gröbner basis of J w.r.t. elimination orders:

Theorem IV.1. *(Elimination Theorem[13]) Let $J \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ be an ideal and let G be a Gröbner basis of J with respect to a lexicographic ordering where $x_1 > x_2 > \cdots > x_d$. Then for every $0 \leq l \leq d$, the set $G_l = G \cap \mathbb{F}_{2^k}[x_{l+1}, \ldots, x_d]$ is a Gröbner basis of the l-th elimination ideal J_l.*

A. Polynomial Abstraction with Elimination & Gröbner Bases

The authors of [6] showed that for any combinational logic block, a canonical word-level polynomial representation can be derived through Gröbner bases computed with elimination orders. Our approach is based on their result, which we reproduce here:

Lemma IV.1. *(From [6]) Given a combinational circuit C with k-bit input $A = (a_0, \ldots, a_{k-1})$ and k-bit output $R = (r_0, \ldots, r_{k-1})$. Denote by $x_1, \ldots, x_d$ all the bit-level variables of C. Let $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d, R, A]$ denote all the polynomials corresponding to the logic gates of the circuit. Let $J_0 = \langle x_1^2 - x_1, \ldots, x_d^2 - x_d, R^q - R, A^q - A \rangle$ be the vanishing ideal, so that $J + J_0 = \langle f_1, \ldots, f_s, \quad x_1^2 - x_1, \ldots, x_d^2 - x_d, R^q - R, A^q - A \rangle$. Compute Gröbner basis $G = GB(J + J_0)$ w.r.t. lex term order with $x_1 > x_2 > \cdots > x_d > R > A$. Then $G_d = G \cap \mathbb{F}_{2^k}[R, A]$ eliminates the internal variables $x_1, \ldots, x_d$ of the circuit. G_d also contains the word-level polynomial $R = \mathcal{F}(A)$ which canonically represents the function of the circuit.*

The authors referred to the elimination (lex) order with $\{$*internal variables* $x_1 > \cdots > x_d\} > \{$*word-level output* $R\} > \{$*word-level input* $A\}$ as the *abstraction term order (ATO)*. We will now show how to repeatedly apply Gröbner basis computations with ATO to verify sequential GF circuits.

V. Verification of Sequential GF Circuits

We follow the sequential GF circuit model of Fig. 1, with word-level variables A, B, R denoting *present states (PS)* and A', B', R' denoting *next states (NS)* of the machine; where $A = \sum_{i=0}^{k-1} a_i \beta^{2^i}$ for the PS variables and $A' = \sum_{i=0}^{k-1} a_i' \beta^{2^i}$ for NS variables, and so on. The normal element β is given. Variables $R(R')$ correspond to those that store the result, and $A, B(A', B')$ store input operands. *E.g.*, for a GF multiplier, A_{init}, B_{init} (and $R_{init} = 0$) are the initial values (operands) loaded into the registers, and $R = \mathcal{F}(A_{init}, B_{init}) = A_{init} \times B_{init}$ is the final result after k-cycles. Our approach aims to find this polynomial representation for R.

Each gate in the combinational logic is represented by a Boolean polynomial. To this set of Boolean polynomials, we append the polynomials that define the word-level to bit-level relations for PS and NS variables ($A = \sum_{i=0}^{k-1} a_i \beta^{2^i}$). We denote this set of polynomials as ideal $J = \langle f_1, \dots, f_s\rangle \subset \mathbb{F}_{2^k}[x_1, \dots, x_d, R, R', A, A', B, B']$. The ideal of vanishing polynomials J_0 is also included, and then the implicit FSM unrolling problem is setup for abstraction.

The configurations of the flip-flops are the states of the machine. *Since the set of states is a finite set of points, we can consider it as the variety of an ideal related to the circuit (from Section IV)*. Moreover, since we are interested in the *function encoded* by the state variables (over k-time frames), we can *project this variety* on the word-level state variables, starting from the initial state A_{init}, B_{init}. Projection of varieties (geometry) corresponds to elimination ideals (algebra), and can be analyzed via Gröbner bases. Therefore, we employ a Gröbner basis computation with ATO: we use a *lex term order* with *bit-level variables > word-level NS outputs > word-level PS inputs*. This allows to eliminate all the bit-level variables and derives a representation only in terms of words. Consequently, k-successive Gröbner basis computations implicitly unroll the machine, and provide word-level algebraic k-cycle abstraction for R' as $R' = \mathcal{F}(A_{init}, B_{init})$.

Algorithm 1 describes our approach. In the algorithm, $from^i$ and to^i are polynomial ideals whose varieties are the valuations of word-level variables R, A, B and R', A', B' in the i-th iteration; and the notation "$\setminus$" signifies that the *NS* in iteration (i) becomes the *PS* in iteration $(i+1)$.

ALGORITHM 1: Abstraction via implicit unrolling for Sequential GF circuit verification

Input: Circuit polynomial ideal J, vanishing ideal J_0, initial state ideal $R(=0), \mathcal{G}(A_{init}), \mathcal{H}(B_{init})$
$from^0(R,A,B) = \langle R, \mathcal{G}(A_{init}), \mathcal{H}(B_{init})\rangle$;
$i = 0$;
repeat
  $i \leftarrow i+1$;
  $G \leftarrow$GB$(\langle J + J_0 + from^{i-1}(R,A,B)\rangle)$ with ATO;
  $to^i(R',A',B') \leftarrow G \cap \mathbb{F}_{2^k}[R',A',B',R,A,B]$;
  $from^i \leftarrow to^i(\{R,A,B\} \setminus \{R',A',B'\})$;
until $i == k$;
return $from^k(R_{final})$

Example V.1. *We demonstrate our approach to verify the 5-bit RH-SMPO circuit of Fig.2. The normal element β in $\mathbb{F}_{2^5}$ is known to be $\beta = \alpha^5$, where α is the primitive element. The circuit can be described by the ideal:*

$$
\begin{aligned}
J =\, & d_0 + a_4b_4, c_1 + a_0 + a_4, c_2 + b_0 + b_4, d_1 + c_1c_2, c_3 + a_1a_4,\\
& c_4 + b_1b_4, d_2 + c_3c_4, e_0 + d_0 + d_1, e_3 + d_1 + d_2, e_4 + d_2,\\
& R_0 + r_4 + e_0, R_1 + r_0, R_2 + r_1, R_3 + r_2 + e_3, R_4 + r_3 + e_4,\\
& A + a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18},\\
& B + b_0\alpha^5 + b_1\alpha^{10} + b_2\alpha^{20} + b_3\alpha^9 + b_4\alpha^{18},\\
& R' + r'_0\alpha^5 + r'_1\alpha^{10} + r'_2\alpha^{20} + r'_3\alpha^9 + r'_4\alpha^{18},\\
& R + r_0\alpha^5 + r_1\alpha^{10} + r_2\alpha^{20} + r_3\alpha^9 + r_4\alpha^{18};
\end{aligned}
$$

In the first iteration, $from^0 = \{R,\ A_{init} + a_0\alpha^5 + a_1\alpha^{10} + a_2\alpha^{20} + a_3\alpha^9 + a_4\alpha^{18},\ B_{init} + b_0\alpha^5 + b_1\alpha^{10} + b_2\alpha^{20} + b_3\alpha^9 + b_4\alpha^{18}\}$ *denotes the initial state.*

After the GB computation is performed with ATO, we find $to^1 = \{R'(\alpha^4 + \alpha^3 + 1)A_{init}^{16}B_{init}^{16} + (\alpha^4 + \alpha^2)A_{init}^{16}B_{init}^4 + (\alpha^3 + 1)A_{init}^{16}B_{init}^2 + (\alpha^4 + \alpha^3 + 1)A_{init}^{16}B_{init} + (\alpha^4 + \alpha^3 + \alpha^2 + 1)A_{init}^8B_{init}^8 + (\alpha^4 + \alpha^3 + \alpha + 1)A_{init}^8B_{init}^4 + (\alpha^3 + \alpha + 1)A_{init}^8B_{init}^2 + (\alpha^4 + \alpha^2)A_{init}^8B_{init} + (\alpha^4 + \alpha^2)A_{init}^4B_{init}^{16} + (\alpha^4 + \alpha^3 + \alpha + 1)A_{init}^4B_{init}^8 + (\alpha^2)A_{init}^4B_{init}^4 + (\alpha^3 + \alpha^2 + \alpha + 1)A_{init}^4B_{init}^2 + (\alpha^4 + \alpha^3 + \alpha + 1)A_{init}^4B_{init} + (\alpha^3 + 1)A_{init}^2B_{init}^{16} + (\alpha^3 + \alpha + 1)A_{init}^2B_{init}^8 + (\alpha^3 + \alpha^2 + \alpha + 1)A_{init}^2B_{init}^4 + (\alpha^3 + \alpha^2 + \alpha)A_{init}^2B_{init}^2 + (\alpha^4 + \alpha)A_{init}^2B_{init} + (\alpha^4 + \alpha^3 + 1)A_{init}B_{init}^{16} + (\alpha^4 + \alpha^2)A_{init}B_{init}^8 + (\alpha^4 + \alpha^3 + \alpha + 1)A_{init}B_{init}^4 + (\alpha^4 + \alpha)A_{init}B_{init}^2 + (\alpha^3 + \alpha + 1)A_{init}B_{init}\}$ *— a polynomial in variables* R', A_{init}, B_{init}.

Continuing this way, in the 5^{th} *iteration:* $from^4 = \{R + (\alpha^3 + \alpha + 1)A_{init}^{16}B_{init}^{16} + (\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1)A_{init}^{16}B_{init}^8 + (\alpha^4 + \alpha)A_{init}^{16}B_{init}^4 + (\alpha^3 + 1)A_{init}^{16}B_{init}^2 + (\alpha^3 + \alpha + 1)A_{init}^{16}B_{init} + (\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1)A_{init}^8B_{init}^{16} + (\alpha^3 + 1)A_{init}^8B_{init}^8 + (\alpha^4 + \alpha^2 + \alpha)A_{init}^8B_{init}^4 + (\alpha^2 + \alpha)A_{init}^8B_{init}^2 + (\alpha^3 + \alpha^2 + 1)A_{init}^8B_{init} + (\alpha^4 + \alpha)A_{init}^4B_{init}^{16} + (\alpha^4 + \alpha^2 + \alpha)A_{init}^4B_{init}^8 + (\alpha^4 + \alpha^2 + \alpha)A_{init}^4B_{init}^4 + (\alpha^2 + \alpha)A_{init}^4B_{init} + (\alpha^3 + 1)A_{init}^2B_{init}^{16} + (\alpha^2 + \alpha)A_{init}^2B_{init}^8 + (\alpha^4 + \alpha^2)A_{init}^2B_{init}^2 + (\alpha^3 + \alpha^2 + 1)A_{init}^2B_{init} + (\alpha^3 + \alpha + 1)A_{init}B_{init}^{16} + (\alpha^3 + \alpha^2 + 1)A_{init}B_{init}^8 + (\alpha^2 + \alpha)A_{init}B_{init}^4 + (\alpha^3 + \alpha^2 + 1)A_{init}B_{init}^2 + (\alpha)A_{init}B_{init},\ \ A_{init} + a_1\alpha^5 + a_2\alpha^{10} + a_3\alpha^{20} + a_4\alpha^9 + a_0\alpha^{18},\ \ B_{init} + b_1\alpha^5 + b_2\alpha^{10} + b_3\alpha^{20} + b_4\alpha^9 + b_0\alpha^{18}\}$ *denotes the current state values.*

Finally, by computing GB with ATO, we obtain: $to^5 = \{\mathbf{R' + A_{init}B_{init}},\ A_{init} + a'_0\alpha^5 + a'_1\alpha^{10} + a'_2\alpha^{20} + a'_3\alpha^9 + a'_4\alpha^{18},\ B_{init} + b'_0\alpha^5 + b'_1\alpha^{10} + b'_2\alpha^{20} + b'_3\alpha^9 + b'_4\alpha^{18}\}$ *as the image. The final result is* $from^5(R_{final}) = R_{final} + A_{init} \cdot B_{init}$, *which verifies the multiplier.*

VI. Improving our Approach

Computing Gröbner bases with elimination orders is infeasible for large circuits; the complexity of Buchberger's algorithm to compute $GB(J + J_0)$ in $\mathbb{F}_q$ is $q^{O(d)}$ [5]. To overcome this complexity, [6] proposed a refinement of ATO (called RATO), and simplified the Gröbner basis computation. They exploited Buchberger's product criteria [14], which states that: *If the leading monomials of f_i, f_j are relatively prime, then $Spoly(f_i, f_j)$ reduces to zero modulo the generating set, i.e.* $Spoly(f_i, f_j) \xrightarrow{F}_+ 0$. This concept was exploited in RATO as follows:

Perform a reverse topological sorting of the nodes in the combinational logic, and define a *lex* term order by the following relation $>_r$: *bit-level circuit variables ordered reverse topologically > word-level output variables > word-level input variables*. Representing the polynomial ideal J in RATO has the effect that there exists *one and only one pair of polynomials* in J that do not have relatively prime leading terms (see Section 5 in [6] for details). All other polynomial pairs will have leading terms that are relatively prime, so these polynomial pairs are not considered in Buchberger's algorithm. The authors of [6] exploited this concept and showed how the Gröbner basis of $(J + J_0)$ can be computed by a *subset* of polynomials, which improves the scalability of their approach. Their approach, however, cannot circumvent the Gröbner basis computation altogether. Consequently, their approach fails to derive a canonical polynomial abstraction when the representation is dense, and contains monomials of high-degrees (e.g. in case of buggy designs).

It turns out that RATO can be applied to sequential circuits in much the same way: $\{$*bit-level circuit variables ordered reverse topologically* $x_1 > \dots x_d\} > \{$*word-level NS variables* $R' > A' > B'\} > \{$*word-level PS variables* $R > A > B\}$. Importantly, *we show that using RATO, the polynomial abstraction can be derived without resorting to a Gröbner basis computation.* Perform the following operations:

1) Represent the polynomials of the sequential circuit S using RATO.
2) Due to RATO, only one pair of polynomials (f_i, f_j) will have leading terms that will not be relatively prime.
3) Reduce $Spoly(f_i, f_j) \xrightarrow{F}_+ h$.

As described in [6], remainder h will contain: i) the word-level variables, and ii) bit-level inputs to the combinational logic, i.e. bit-level present-state variables. All other internal circuit variables will not appear in h, as they will be canceled by division due to RATO.

Example VI.1. *Let us re-visit Example V.1 and the RH-SMPO circuit of Fig. 2. For this circuit, the term order under RATO is lex with:* $\{r_0', r_1', r_2', r_3', r_4'\} > \{r_0, r_1, r_2, r_3, r_4\} > \{e_0, e_3, e_4\}, \{d_0, d_1, d_2\}, \{c_1, c_2, c_3, c_4\}\{a_0, a_1, a_2, a_3, a_4, b_0, b_1, b_2, b_3, b_4\} > R' > R > \{A, B\}$.

Among all the generators of the ideal J from Ex.V.1, using RATO ***we find only one pair of polynomials*** *whose leading monomials are not relatively prime:* $(f_i, f_j), f_i = r_0' + r_4 + e_0, f_j = r_0'\alpha^5 + r_1'\alpha^{10} + r_2'\alpha^{20} + r_3'\alpha^9 + r_4'\alpha^{18} + R'$. *Then:*

$$
\begin{aligned}
&Spoly(f_i, f_j) \xrightarrow{J+J_0}_+ \\
&(\alpha^3+\alpha^2+\alpha)r_1 + (\alpha^4+\alpha^3+\alpha^2)r_2 + (\alpha^2+\alpha)r_3 + (\alpha)r_4 \\
&+(\alpha^3+\alpha^2)a_1b_1 + (\alpha^4+\alpha^3+\alpha^2+\alpha)a_1b_2 + (\alpha^2+\alpha)a_1b_3 \\
&+(\alpha^2+1)a_1b_4 + (\alpha^4+1)a_1B + (\alpha^4+\alpha)a_2b_1 + (\alpha^4+\alpha^3+\alpha)a_2b_2 \\
&+(\alpha^3+1)a_2b_3 + (\alpha^3+\alpha^2+1)a_2b_4 + (\alpha^3+\alpha^2)a_2B + (\alpha^2+\alpha)a_3b_1 \\
&+(\alpha^3+1)a_3b_2 + (\alpha+1)a_3b_3 + (\alpha^4+\alpha^2+\alpha)a_3b_4 \\
&+(\alpha^4+\alpha^3+\alpha)a_3B + (\alpha^3+1)a_4b_1 + a_4b_2 + (\alpha^4+\alpha^2+\alpha)a_4b_3 \\
&+(\alpha^4+\alpha^3+1)a_4b_4 + (\alpha^2+\alpha)a_4B + (\alpha^4+1)b_1A + (\alpha^3+\alpha^2)b_2A \\
&+(\alpha^4+\alpha^3+\alpha)b_3A + (\alpha^2+\alpha)b_4A + (\alpha^4+\alpha^2+\alpha+1)R' + AB
\end{aligned}
$$

As shown above, the remainder h contains both bit-level variables and word-level *state variables* a_i, b_i, r_i, R, A, B. We desire a polynomial in only word level variables (e.g. $R' + \mathcal{F}(A, B)$), without computing a Gröbner basis. This can be achieved if we represent the bit-level state variables a_i, b_i, r_i in terms of their word-level variables A, B, R, respectively. This can be achieved due to the following property of finite fields:

Lemma VI.1. *For* $\alpha_1, \dots, \alpha_t \in \mathbb{F}_{2^k}$, $(\alpha_1 + \alpha_2 + \cdots + \alpha_t)^{2^i} = \alpha_1^{2^i} + \alpha_2^{2^i} + \cdots + \alpha_t^{2^i}$, *for* $i = 1, 2, \dots$.

Consider the polynomials that define the relationship between the word-level and bit-level variables. Let $A = a_0\beta + a_1\beta^2 + \cdots + a_{k-1}\beta^{2^{k-1}}$. Due to Lemma VI.1, we have $A^2 = a_0\beta^2 + a_1\beta^4 + \cdots + a_{k-1}\beta^{2\cdot 2^{k-1}}$ (as $a_i^2 = a_i$). By repeated squaring:

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

Consider the above as k *linear equations* with $a_0, \dots, a_{k-1}$ as k-unknowns, with β and its powers as coefficients, and $A, A^2, \dots\dots, A^{2^{k-1}}$ as k constants. Then, we can solve for the unknowns $a_0, \dots, a_{k-1}$ and obtain expressions for them in terms of A and β. The problem can be setup in matrix form:

$$
\begin{pmatrix}
\beta & \beta^2 & \cdots & \beta^{2^{k-1}} \\
\beta^2 & \beta^4 & \cdots & \beta^{2^{k-1}\cdot 2} \\
\vdots & \vdots & \ddots & \vdots \\
\beta^{2^{k-1}} & \beta^{2^{k-1}\cdot 2} & \cdots & \beta^{2^{2(k-1)}}
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix}
=
\begin{pmatrix} A \\ A^2 \\ \vdots \\ A^{2^{k-1}} \end{pmatrix}
$$

Gaussian elimination on this system of equations can be applied, and each bit-level variable a_i can be represented as a function of the word-level variables: $a_i = g_i(A)$. These bit-level variables can be easily substituted in the remainder h obtained by reduction due to RATO. What we will obtain is a word-level polynomial of the form $h : R' + \mathcal{F}(A, B)$ which canonically represents the k-cycle function of the circuit. It is also easy to show that this polynomial is a unique, canonical representation because it is reduced modulo $J + J_0$.

VII. Experimental Results

We have implemented our approach within the SINGULAR symbolic algebra computation system [v. 3-1-6] [15]. Using our implementation, we have performed experiments to verify two SMPO architectures — Agnew-SMPO [2] and the RH-SMPO [3] — over $\mathbb{F}_{2^k}$, for various datapath/field sizes. Bugs are also introduced into the SMPO designs by modifying a few gates in the combinational logic block. Experiments using SAT-, BDD-, and AIG-based solvers are also conducted and results are compared against our approach. Our experiments run on a desktop with 3.5GHz Intel Core™ i7 Quad-core CPU, 16 GB RAM and 64-bit Linux.

Evaluation of SAT/ABC/BDD based methods: To verify circuit S against the polynomial $\mathcal{F}$, we unroll the SMPO over k time-frames, and construct a miter against a combinational implementation of $\mathcal{F}$. A (pre-verified) $\mathbb{F}_{2^k}$ Mastrovito multiplier [16] is used as the *spec* model. This miter is checked for SAT using the *Lingeling* [17] solver. We also experiment with the Combinational Equivalence Checking (CEC) engine of the ABC tool [18], which uses AIG-based reductions to identify internal AIG equivalences within the miter to efficiently solve verification. The BDD-based VIS tool [19] is also used for equivalence check. The run-times for verification of (unrolled) RH-SMPO against Mastrovito *spec* are given in Table I – which shows that the techniques fail beyond 23 bit fields.

TABLE I: Run-time for verification of bug-free RH-SMPO circuits for SAT, ABC and BDD based methods. *TO* = timeout 14 hrs

	Word size of the operands k-bits			
Solver	11	18	23	33
Lingeling	593	*TO*	*TO*	*TO*
ABC	6.24	*TO*	*TO*	*TO*
BDD	0.1	11.7	1002.4	*TO*

CEC between unrolled RH-SMPO and Agnew-SMPO also suffers the same fate (results omitted). In fact, both SMPO designs are based on slightly different mathematical concepts and their computations in all clock-cycles, except for the k^{th} one, are also different. These designs have no internal logical/structural equivalencies, and verification with SAT/BDDs/ABC is infeasible. Their dissimilarity is depicted in Table II, where N_1 depicts the number of AIG nodes in the miter prior to *fraig_sweep*, and the nodes after *fraiging* are recorded as N_2; so $\frac{N_1-N_2}{N_1}$ reflects the proportion of equivalent nodes in original miter, which emphasizes the (lack of) *similarity* between two designs.

TABLE II: Similarity between RH-SMPO and Agnew's SMPO

Size k	11	18	23	33
N_1	734	2011	3285	6723
N_2	529	1450	2347	4852
Similarity	27.9%	27.9%	28.6%	27.8%

Evaluation of Our Approach: Our algorithm inputs the circuit given in BLIF format, derives RATO, and constructs the polynomial ideal from the logic gates and the register/data-word description. We perform one *Spoly* reduction, followed by the bit-level to word-level substitution, in each clock cycle. After k iterations, the final result polynomial R is compared against the spec polynomial. The run-times for verifying bug-free and buggy RH-SMPO and Agnew-SMPO are shown in Table III and Table IV, respectively. We can verify, as well as catch bugs in, up to 100-bit multipliers. Beyond 100-bit fields, our approach is infeasible – mostly due to the fact that the intermediate abstraction polynomial R is very dense and contains high-degree terms, which can be infeasible to compute. However, it should be noted that if we do not use the proposed bit-level to word-level substitution, and compute reduced Gröbner bases with RATO, then our approach does not scale beyond 33-bit datapaths.

TABLE III: Run-time (seconds) for verification of bug-free and buggy RH-SMPO using our approach

Operand size k	33	51	65	81	89	99
#variables	4785	11424	18265	28512	34354	42372
#polynomials	3630	8721	13910	21789	26255	32373
#terms	13629	32793	52845	82539	99591	122958
Runtime(bug-free)	112.6	1129	5243	20724	36096	67021
Runtime(buggy)	112.7	1129	5256	20684	36120	66929

TABLE IV: Run-time (seconds) for verification of bug-free and buggy Agnew's SMPO our approach

Operand size k	36	66	82	89	100
#variables	6588	21978	33866	39872	50300
#polynomials	2700	8910	13694	16109	20300
#terms	12996	43626	67322	79299	100100
Runtime(bug-free)	113	3673	15117	28986	50692
Runtime(buggy)	118	4320	15226	31571	58861

VIII. Conclusions

This paper has described a method to verify sequential Galois field multipliers over $\mathbb{F}_{2^k}$ using computer algebra and algebraic geometry based approach. As sequential Galois field circuits perform the computations over k clock-cycles, verification requires an efficient approach to unroll the computation, and represent it as a canonical word-level multi-variate polynomial. Using algebraic geometry, we show that the unrolling of the computation at word-level can be performed by Gröbner bases and elimination term orders. Subsequently, we show that the complex Gröbner basis computation can be eliminated by means of a bit-level to word-level substitution, which is implemented using the binomial expansion over Galois fields and Gaussian elimination. Our approach is able to verify up to 100-bit sequential circuits, whereas contemporary techniques fail beyond 23-bit datapaths.

References

[1] S. Gao, *Normal Basis over Finite Fields*, PhD thesis, University of Waterloo, 1993.

[2] Gordon B. Agnew, Ronald C. Mullin, IM Onyszchuk, and Scott A. Vanstone, "An implementation for a fast public-key cryptosystem", *Journal of CRYPTOLOGY*, vol. 3, pp. 63–79, 1991.

[3] Arash Reyhani-Masoleh and M Anwar Hasan, "Low complexity word-level sequential normal basis multipliers", *Computers, IEEE Transactions on*, vol. 54, pp. 98–110, 2005.

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

[5] 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, pp. 1409–1420, 2013.

[6] T. Pruss, P. Kalla, and F. Enescu, "Equivalence verification of large galois field arithmetic circuits using word-level abstraction via gröbner bases", *in Proc. Design Automation Conference (DAC)*, 2014.

[7] O. Coudert and J.C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits", *in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 126–129, 1990.

[8] K. L. McMillan, "Interpolation and SAT based Model Checking", *in Computer-Aided Verification*, 2003.

[9] A. Bradley, "Sat-based Model Checking Without Unrolling", *in Verification, Model Checking and Abstract Interpretation (VMCAI)*, pp. 70–87, 2011.

[10] G. Avrunin, "Symbolic Model Checking using Algebraic Geometry", *in Computer Aided Verification Conference*, pp. 26–37, 1996.

[11] Q. Tran and M. Y. Vardi, "Gröbner Basis Computation in Boolean Rings for Symbolic Model Checking", *in IASTED Conf. on Modelling and Simulation*, 2007.

[12] Michael Brickenstein and Alexander Dreyer, "Polybori: A Framework for Gröbner Basis Computations with Boolean Polynomials", *Journal of Symbolic Computation*, vol. 44, pp. 1326–1345, September 2009.

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

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

[15] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, "Singular 3-1-6 — A computer algebra system for polynomial computations", http://www.singular.uni-kl.de, 2012.

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

[17] Armin Biere, "Lingeling, plingeling and treengeling entering the sat competition 2013", *Proceedings of SAT Competition 2013; Solver and*, p. 51, 2013.

[18] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", *in Computer Aided Verification*, vol. 6174, pp. 24–40. Springer, 2010.

[19] Robert K Brayton, Gary D Hachtel, Alberto Sangiovanni-Vincentelli, Fabio Somenzi, Adnan Aziz, Szu-Tsung Cheng, Stephen Edwards, Sunil Khatri, Yuji Kukimoto, Abelardo Pardo, et al., "Vis: A system for verification and synthesis", *in Computer Aided Verification*, pp. 428–432. Springer, 1996.