Verification of Sequential Galois Field Circuits using Algebraic Geometry *

Xiaojun Sun
University of Utah
Department of Electrical & Computer Engineering
Salt Lake City, USA
xiaojun.sun@utah.edu

Priyank Kalla
University of Utah
Department of Electrical & Computer Engineering
Salt Lake City, USA
kalla@ece.utah.edu

ABSTRACT
Circuits working in Galois fields are increasingly employed in designs like arithmetic component for Elliptic Curve Cryptography (ECC). This work proposes a new method to effectively verify sequential circuits in Galois fields. Algebraic geometry is introduced to describe circuits behavior and expand the definition of implicit state space traversal. Moreover, Gröbner basis representation is adopted for word-level abstraction on circuit variables to address state explosion problem with BDDs. Experiments are run with fast Gröbner basis computation engine to get competitive results.

Categories and Subject Descriptors
EDA5.1 [**Verification**]: Functional, transaction-level, RTL, and gate-level modeling and verification of hardware design

General Terms
Verification, Algorithms

Keywords
Verification, Sequential, Galois Fields, Algebraic Geometry

1. ESSENTIAL THEORY

1.1 Polynomial Representation
A typical sequential circuit is composed as figure 1. The combinational logic has primary input x, state inputs $\{s_0, s_1, \ldots, s_{k-1}\}$, primary output z and state outputs $t_0, t_1, \ldots, t_{k-1}$. State outputs are latched into registers, which will be fed back to state inputs in next clock cycle.

A transition function Δ is a Boolean function about state outputs and all inputs, i.e.

$$t_i = \Delta_i(x, s_0, s_1, \ldots, s_{k-1})$$

*Version 0.3, refined theory part with normal basis, proposed details for designing 2 examples.

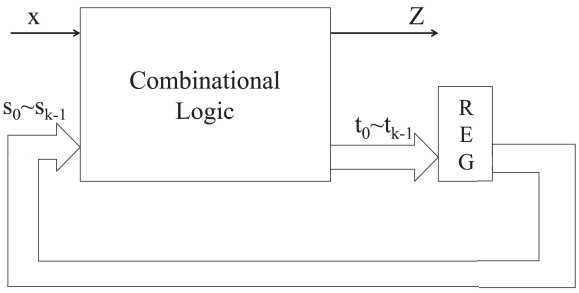

Figure 1: A typical sequential circuit block view

This constrain could also be written in the form of equation $polynomial = 0$:

$$t_i - \Delta_i(x, s_0, s_1, \ldots, s_{k-1}) = 0$$

Solution to this equation is limited within $\{0,1\}$, which is set of all elements in Galois field $\mathbb{F}_2$. Furthermore, there exists a one-to-one corresponding relation from Boolean space to Galois field on higher dimension: $\mathbb{B}^k \to \mathbb{F}_{2^k}$, which means it is possible to find a set of elements in $\mathbb{F}_{2^k}$ representing all Boolean values every bit of a bit vector can take. In Galois field, a as 1 shows. A word-level representation on state inputs/outputs can be attained through above correspondence. If the set of state inputs $\{s_0, s_1, \ldots, s_{k-1}\}$ is denoted by S taking values from $\mathbb{F}_{2^k}$, and set of outputs $\{t_0, t_1, \ldots, t_{k-1}\}$ is denoted by T, a word-level transition function can be written as

$$T - \Delta(x, S) = 0$$

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

$a_3a_2a_1a_0$	Polynomial	$a_3a_2a_1a_0$	Polynomial
0000	0	1000	α^3
0001	1	1001	α^3+1
0010	α	1010	$\alpha^3+\alpha$
0011	$\alpha+1$	1011	$\alpha^3+\alpha+1$
0100	α^2	1100	$\alpha^3+\alpha^2$
0101	α^2+1	1101	$\alpha^3+\alpha^2+1$
0110	$\alpha^2+\alpha$	1110	$\alpha^3+\alpha^2+\alpha$
0111	$\alpha^2+\alpha+1$	1111	$\alpha^3+\alpha^2+\alpha+1$

If $T-\Delta(x,S)$ is taken as a polynomial f, then $f \in \mathbb{F}_{2^k}[x]$ *mod* $P(\alpha)$.

1.2 Gröbner Basis Theory

Gröbner basis is used to calculate desired results out of a set of polynomials. If there exists multiple polynomials $\{f_1, f_2, \ldots, f_r\}$ representing constrains on T,x and S, then all polynomials should simultaneously equal to 0.

$$\begin{cases} f_1 & = 0 \\ f_2 & = 0 \\ \cdots & \\ f_r & = 0 \end{cases}$$

Solution to these equations is set of values of (T,x,S), which is called **Variety** of **ideal** $I = \langle f_1, f_2, \ldots, f_r \rangle$.

Calculate Reduced Gröbner Basis (RGB) from ideal I with **elimination term order**, there exists a polynomial contains only T, if the values of states input S and primary input x are given in the form of polynomials and included by I, the values of T can be computed.

Otherwise, if RGB calculation is manipulated under **abstraction term order**, there exists a polynomial in the form of $T-\mathcal{F}(x,S)$ representing transition function. In next clock cycle, assign new state input S' with T, the next state can be computed again, result is polynomial $T'-\mathcal{F}(x',S')$.

1.3 Normal Basis Representation

Let β be an element in the Galois field F_{2^n} constructed by primitive element α and minimal polynomial $P(\alpha)$. Then a basis in the form $\{\beta, \beta^2, \beta^4, \beta^8, ..., \beta^{2^{n-1}}\}$ is a *Normal Basis*; here β is called *Normal Element*.

For arithmetic operations in Galois fields, squaring and multiplication (with modulus) can be greatly simplified if normal basis representation is adopted to represent operands.

Example 1. Element squaring: In F_{2^n}, all coefficients which can be divided by 2 are reduced, so following equality holds for all field elements a and b:

$$(a+b)^2 = a^2 + b^2$$

Apply this rule on element squaring of standard/polynomial basis:

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

using Fermat's little theorem $\beta^{2^n} = \beta$. This shows that squaring of field elements represented by normal bases can be easily completed by operating a simple right-cyclic rotation.

Example 2. Normal basis multiplication: There are 2 binary vectors representing operands of multiplication in normal basis:

$$A = (a_0, a_1, \ldots, a_{n-1}),\ B = (b_0, b_1, \ldots, b_{n-1})$$

the product is also written in normal basis representation:

$$C = A * B = (c_0, c_1, \ldots, c_{n-1})$$

Describe calculation procedure for the most significant digit of product with a function:

$$c_{n-1} = f(a_0, a_1, \ldots, a_{n-1}; b_0, b_1, \ldots, b_{n-1})$$

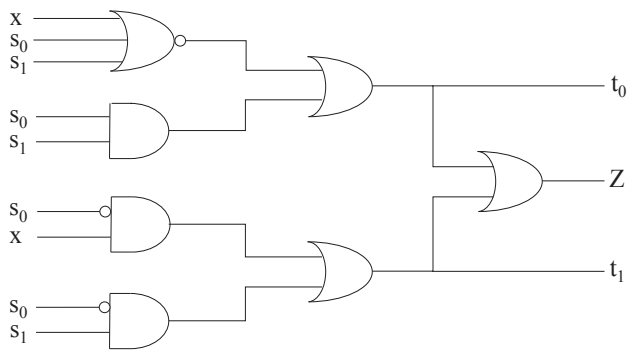

Figure 2: Gate-level circuit of combinational logic block of sample FSM

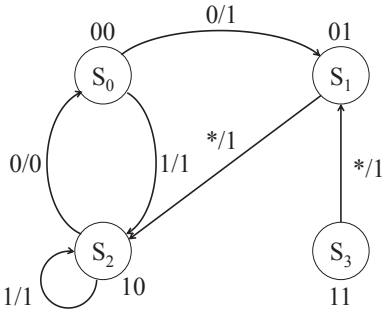

Figure 3: State transition graph of sample FSM

Square both side: $C^2 = A^2 * B^2$, i.e. the second significant digit

$$c_{n-2} = f(a_{n-1}, a_0, a_1, \ldots, a_{n-2}; b_{n-1}, b_0, b_1, \ldots, b_{n-2})$$

By this method it is easy to get all digits of product C.

2. ALGEBRAIC GEOMETRY APPROACH

2.1 State Space Traversal

The first example is a mealy finite state machine (FSM). Gate-level circuits of combinational logic is figure 2, x is primary input, $\{s_0, s_1\}$ are state inputs, Z is primary output and $\{t_0, t_1\}$ are state outputs. Word-level state input is $S = s_0 + s_1 \cdot \alpha$, output is $T = t_0 + t_1 \cdot \alpha$. In Galois field $\mathbb{F}_{2^2}$, bit-level variables x, s_0, s_1, Z, t_0, t_1 can take values from $\{0,1\}$, while word-level variables S, T may take values from $\{0, 1, \alpha, 1+\alpha\}$. State transition graph (STG) showed in figure 3 uses 2-bit Boolean vector to represent 4 states $\{S_0, S_1, S_2, S_3\}$, which could be converted to elements in $\mathbb{F}_{2^2}$ similarly as table 1 shows.

One important technique to check sequential equivalence is **state space traversal**. In this example, Breath-First-Search (BFS) is employed for traversal, which can be described with following algorithm:

Our approach is implementing state space traversal by refined BFS algorithm combined with polynomial representation and Gröbner basis theory. In practice, this example will show how to apply concepts and techniques from algebraic geometry on each line of above BFS algorithm.

Algorithm 1: Breadth-first Traversal Algorithm

Input: Transition functions Δ, initial state S^0

1 $from^0 = reached = S^0$;
2 **repeat**
3 $\quad i \leftarrow i+1$;
4 $\quad to^i \leftarrow \text{Img}(\Delta, from^{i-1})$;
5 $\quad new^i \leftarrow to^i \cap \overline{reached}$;
6 $\quad reached \leftarrow reached \cup new^i$;
7 $\quad from^i \leftarrow new^i$;
8 **until** $new^i == 0$;
9 **return** *reached*

2.2 States and Varieties of Ideal

THEOREM 1. *State variables S,T and sets of states such as $from^i, to^i$ can always be represented as varieties of ideals.*

For example in Line 1 of BFS algorithm, assume initial state is S_3 in figure 3, then corresponding polynomial $f = \mathcal{F}(S^0) = S^0 - 1 - \alpha$. Consider an ideal I with only one generator f, its variety $V(I) = \{\gamma \mid \gamma \in \mathbb{F}_{2^2}, \gamma - 1 - \alpha = 0\}$, which equals to $\{1+\alpha\}$, the only valid value S^0 can take.

If an ideal contains multiple polynomial specifications, it is necessary to compute Gröbner basis with elimination term order to get one polynomial only on desired variable. In first iteration of Line 4, to^i has multiple specifications, some of them are transition functions on bit-level:

$$\begin{aligned} f_1: &\quad t_0 - (\bar{x} \text{ and } \overline{s_0} \text{ and } \overline{s_1}) \text{ or } (s_0 \text{ and } s_1) \\ f_2: &\quad t_1 - (\overline{s_0} \text{ and } x) \text{ or } (p \text{ and } \overline{s_1}) \end{aligned}$$

And some are bits-to-word definitions fitting polynomial representation of elements in $\mathbb{F}_{2^2}$:

$$\begin{aligned} f_3: &\quad S - s_0 - s_1\alpha \\ f_4: &\quad T - t_0 - t_1\alpha \end{aligned}$$

And an polynomial about initial state as mentioned above:

$$f_5: S - 1 - \alpha$$

And the rests are vanishing polynomials for every variable, bit-level and word level: $f_6 : x^2 - x$; $f_7 : t_0^2 - t_0$; $f_8 : t_1^2 - t_1$; $f_9 : S^4 - S$; $f_{10} : s_0^2 - s_0$; $f_{11} : s_1^2 - s_1$; $f_{12} : T^4 - T$

Transition equations here contains some Boolean operators, they can be re-written in terms of operations in Galois fields. In $\mathbb{F}_2$, let $TRUE = 1, FALSE = 0$, for either element a in this field, considering $0+0 = 1+1 \equiv 0 \ (mod\ 2)$ and $0+1 = 1+0 \equiv 1 \ (mod\ 2)$, the inverse of a is: $\bar{a} = 1 + a$. Similarly all Boolean operators can be converted within $\mathbb{F}_2$, table 2 gives part of them and their corresponding operations in $\mathbb{F}_2$: Also note that there is no specification on initial primary input x, in Gröbner basis approach this means x is smoothed by reversely using Shannon's expansion. Using elimination term order: $others > bit\text{-}level\ inputs/outputs > S > T$, compute Gröbner basis for ideal $J = \langle f_1, f_2, \ldots, f_{12} \rangle$, the result will include one polynomial f_T contains only variable T. In this example, $f_T = T + 1$. This polynomial equals to $T - 1$ since coefficients of polynomial representation in $\mathbb{F}_{2^2}$ are limited within $\mathbb{F}_2$, where $1 \equiv -1 \ (mod\ 2)$. Solution to $f_T = 0$ is $T = 1$, which shows that next state the machine just reached is $\{S_1(01)\}$.

Table 2: Some Boolean operators and corresponding operations in $\mathbb{F}_2$

Boolean operator	operation in $\mathbb{F}_2$
$\bar{a}$	$1+a$
a *and* b	ab
a *or* b	$a+b+ab$
$a \oplus b$	$a+b$

2.3 Application of Algebraic Geometry

There are some difficulties with polynomial representation when executing Line 5 and Line 6, it is necessary to explore how **Union**, **Intersection** and **Complement Set** works in Galois field $\mathbb{F}_{2^k}$. Since state set variables $from^i, to^i, reached$ all have single polynomial, consider an ideal I with single generator $I = \langle f \rangle$. Example: assume $I = \langle f \rangle = \langle T^2 + (1+\alpha) \cdot T + \alpha \rangle$, its variety $V(I) = \{a \mid a \in \mathbb{F}_{2^2} \text{ and } f(a) = 0\} = \{1, \alpha\}$. So it is reasonable to specify: union, intersection and complement set mentioned in this paper are all functions on **varieties**. To better discuss this problem, introduce following concepts from algebraic geometry:

Definition 1. (**Sum of Ideals**) If I and J are ideals in $k[x_1, \ldots, x_n]$, then the **sum** of I and J, denoted by $I + J$, is the set

$$I + J = \{f + g \mid f \in I \text{ and } g \in J\}. \tag{1}$$

Furthermore, if $I = \langle f_1, \ldots, f_r \rangle$ and $J = \langle g_1, \ldots, g_s \rangle$, then $I + J = \langle f_1, \ldots, f_r, g_1, \ldots, g_s \rangle$.

Definition 2. (**Product of Ideals**) If I and J are ideals in $k[x_1, \ldots, x_n]$, then the **product** of I and J, denoted by $I \cdot J$, is defined to be the ideal generated by all polynomials $f \cdot g$ where $f \in I$ and $g \in J$. Furthermore, let $I = \langle f_1, \ldots, f_r \rangle$ and $J = \langle g_1, \ldots, g_s \rangle$, then

$$I \cdot J = \langle f_i g_j \mid 1 \le i \le r, 1 \le j \le s \rangle. \tag{2}$$

Definition 3. (**Quotient of Ideals**) If I and J are ideals in $k[x_1, \ldots, x_n]$, then $I : J$ is the set

$$\{f \in k[x_1, \ldots, x_n] \mid f \cdot g \in I, \forall g \in J\} \tag{3}$$

and is called the **ideal quotient** of I by J.

These concepts can lead the way to solution by adopting following theorems:

THEOREM 2. *If I and J are ideals in $k[x_1, \ldots, x_n]$, then $\mathbf{V}(I + J) = \mathbf{V}(I) \bigcap \mathbf{V}(J)$ and $\mathbf{V}(I \cdot J) = \mathbf{V}(I) \bigcup \mathbf{V}(J)$.*

THEOREM 3. *If I, J are ideals with only one generator, then $\mathbf{V}(I : J) = \mathbf{V}(I) - \mathbf{V}(J)$.*

Proof to these theorems are referred to (that yellow book?) and (my write-up?).

For varieties intersection $\{1\} \bigcap \{1, \alpha\}$, calculate ideal sum $\langle T + 1, T^2 + (1+\alpha) \cdot T + \alpha \rangle = \langle T + 1 \rangle$, its variety is $\{1\}$; for varieties union $\{1, 1+\alpha\} \bigcup \{1+\alpha\}$, calculate ideal product $\langle (T+1+\alpha) \cdot (T^2 + (1+\alpha) \cdot T + \alpha) \rangle = \langle T^3 + 1 \rangle$, its variety is $\{1, \alpha, 1+\alpha\}$; for

complement set of variety $\{1,\alpha\}$, the universal set is the variety of ideal of vanishing polynomial $V(\langle T^4+T\rangle)=\{0,1,\alpha,1+\alpha\}$, so $\overline{V(\langle T^2+(1+\alpha)\cdot T+\alpha\rangle)}=V(\langle T^4+T\rangle)-V(\langle T^2+(1+\alpha)\cdot T+\alpha\rangle)$, which equals to $V(\langle T^4+T\rangle:\langle T^2+(1+\alpha)\cdot T+\alpha\rangle)=V(\langle T^2+(1+\alpha)\cdot T\rangle)$, the result is $\{0,1+\alpha\}$.

From discussion above, the BFS traversal is completely implemented in Galois field. If the initial input is S_3: $S+1+\alpha$, the final return value will be set of reachable states: T^4+T, i.e. the universal set, $\{S_0,S_1,S_2,S_3\}$.

2.4 Discussion on Ideal quotient and its generator

In nature, we are looking varieties as the solutions to specific polynomial $f=0$ WITHIN $\mathbb{F}_{2^n}$, because (affine) varieties are defined within algebraic closure k^n but we only care about those fall in $\mathbb{F}_{2^n}$.

Theorem 2 related to DEF. 1 and 2 counts on ideal generators, so there will be no ambiguity on new ideals we get from this theorem. It also appeared in Avrunin's CAV paper. The only problem is what ideal quotient operation will give us. This should be further discussed because the completeness of our algebraic geometry approach will benefit from it. *Singular* has the function of ideal quotient, so ask from the author maybe another choice.

If we put this conception aside, it is also feasible to define complement set from "solution to equation" sense. Assume $f(x)=x+1+\alpha=0$, its solution within $\mathbb{F}_{2^2}$ is $\{1+\alpha\}$; meanwhile solutions to vanishing polynomial $v(x)=x^2+x=0$ are $\{0,1,\alpha,1+\alpha\}$. Now compute solutions to

$$g(x)=\frac{v(x)}{f(x)}=0$$

, this requires $v(x)=0$ and $f(x)\neq 0$, which means the complement set of solution to $f(x)=0$.

Issues to this interpretation: (a) Is $v(x)$ always divisible by $f(x)$? (b) Is $v(x)=0$ and $f(x)\neq 0$ equivalent with $g(x)=0$ on their solutions? Could $f(x)$ equal to 0?

3. FUNCTIONAL VERIFICATION

3.1 Sequential Galois Arithmetic Multiplier

Our experiment is to verify the function of a Galois field multiplier after certain clock cycles. The following design is Sequential Multiplier with Parallel Output (SMPO), a Normal Basis multiplier based on Massey-Omura algorithm. Inputs and outputs are all 5-bit word taking value from $\mathbb{F}_{2^5}$. After loading operands A and B, setting all output latches to 0 and running for 5 iterations, the output should be $R=A\cdot B \pmod{x^5+x^2+1}$. Similarly, build elimination ideal for all gates/operations and induce initial states of latches. However, instead of eliminating all variables to one, this example adopts abstraction term order and keeps the polynomial contains function between output and inputs. Here the lex ordering is $\{others\}>R>\{A,B\}$, and objective polynomial is $R+\mathcal{F}(A,B)$.

Apply this approach to calculate image in BFS traversal, but modify algorithm 1 to make it adapt 5 steps reachable states enumeration. The result is $R+AB$, which validates the function of this circuit.

3.2 Gröbner basis based Approach

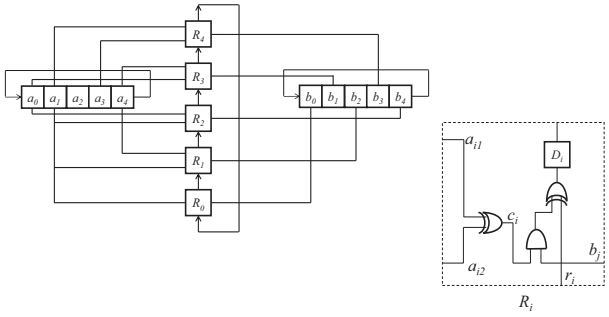

Figure 4: 5-bit SMPO

For 5-bit normal basis multiplication, the i-th digit of product can be written as

$$R_i=b_ia_{i+1}+b_{i+1}(a_i+a_{i+3})+b_{i+2}(a_{i+3}+a_{i+4})+b_{i+3}(a_{i+1}+a_{i+2})+b_{i+4}(a_{i+2}+a_{i+4}),\ 0\le i\le 4$$

It is possible to calculate every conjunctive term in the product simultaneously within one clock cycle, on distinct bits. Use cyclic similarity of above function, following executions are operated:

Example 3. Sequential Multiplier Protocol:

- **Initial** $R_0=R_1=R_2=R_3=R_4=0$
- **Clock 1** $R_0=a_1b_0, R_1=b_2(a_1+a_4), R_2=b_4(a_0+a_1), R_3=b_1(a_4+a_0), R_4=b_3(a_1+a_3)$
- **Clock 2** $R_0=b_3(a_1+a_3)+a_0b_4, R_1=a_1b_0+b_1(a_0+a_3), R_2=b_2(a_1+a_4)+b_3(a_4+a_0), R_3=b_4(a_0+a_1)+b_0(a_3+a_4), R_4=b_1(a_4+a_0)+b_2(a_0+a_2)$
- ...
- **Clock 5** $R_0=c_0, R_1=c_1, R_2=c_2, R_3=c_3, R_4=c_4$, i.e. $R=A\cdot B$.

In BFS algorithm, the **Image** function is implemented by constructing an elimination ideal then compute its Gröbner basis.

Example 4. For 5-bit SMPO, the ideal consists of (for the first clock cycle):

(a) **Gate descriptions:** $a_1+a_4+c_1, a_1+a_0+c_2, a_0+a_4+c_3, a_1+a_3+c_4, a_1b_0+r_4+R_0, c_1b_2+r_0+R_1, c_2b_4+r_1+R_2, c_3b_1+r_2+R_3, c_4b_3+r_3+R_4$;

(b) **Word-level variables:** $A+a_0\alpha^5+a_1\alpha^{10}+a_2\alpha^{20}+a_3\alpha^9+a_4\alpha^{18}, B+b_0\alpha^5+b_1\alpha^{10}+b_2\alpha^{20}+b_3\alpha^9+b_4\alpha^{18}, r+r_0\alpha^5+r_1\alpha^{10}+r_2\alpha^{20}+r_3\alpha^9+r_4\alpha^{18}, R+R_0\alpha^5+R_1\alpha^{10}+R_2\alpha^{20}+R_3\alpha^9+R_4\alpha^{18}$;

(c) **Vanishing polynomials:** $a_0^2+a_0, a_1^2+a_1, a_2^2+a_2, a_3^2+a_3, a_4^2+a_4, b_0^2+b_0, b_1^2+b_1, b_2^2+b_2, b_3^2+b_3, b_4^2+b_4, r_0^2+r_0, r_1^2+r_1, r_2^2+r_2, r_3^2+r_3, r_4^2+r_4, R_0^2+R_0, R_1^2+R_1, R_2^2+R_2, R_3^2+R_3, R_4^2+R_4, c_1^2+c_1, c_2^2+c_2, c_3^2+c_3, c_4^2+c_4, A^{32}+A, B^{32}+B, r^{32}+r, R^{32}+R$;

(d) **Feedback input:** r_{in}.

Polynomial r_{in} equals to $from^{i-1}$ in Line 4, BFS algorithm. Under abstraction term ordering, polynomial to^i is assigned with a polynomial in result Gröbner basis which has the form $R+\mathcal{F}(A,B)$. Since all outputs are connected to feedback inputs in SMPO, Line 5 and 6 will be neglected. Line 7 is finished by replace current output R with previous output r. Initially $r_{in}=0$.

In next clock cycle, $r_{in}=r+\mathcal{F}(A,B)$ updated from latest result; simultaneously operands A and B have been cyclically shifted, so gate descriptions in (a) should be modified accordingly. The loop is operated for 5 times, result from the last step's Gröbner basis should be polynomial $R+AB$.

This experiment can be repeated on n-bits SMPO after running for n clock cycles.

4. FAST GRÖBNER BASIS COMPUTATION

4.1 Refined abstraction term order (RATO)

A lexicographic order constrained by following relation $>_r$: "circuit variables ordered reverse topologically" > "designated word-level output" > "word-level inputs" is called the *Refined Abstraction Term Order (RATO)*.

In Buchberger's algorithm, the specification polynomial (Spoly) for each pair is calculated. In RATO, most polynomials have relatively prime leading terms/monomials (which means $Spoly \xrightarrow{J+J_0}_+ 0$) except one pair: word-level polynomial corresponding to outputs and its leading bit-level variable's gate description polynomial. Its remainder r lets following lemma hold:

LEMMA 1. *r will only contain primary inputs (bit-level and word-level) and word-level output; furthermore, there will be one and only one term containing word-level output whose monomial is word-level output itself rather than higher order form.*

PROOF. First proposition is easy to prove by contradiction. Second part, the candidate pair of polynomials only have one term of single word-level output variable (say it is R) and this term is the last term under RATO, which means there is only one term with R in Spoly. Meanwhile in other polynomials from $J+J_0$ there is no such term containing R, so this term will be kept to remainder r, in first degree. □

Example 5. The elimination ideal for 5-bit SMPO (Ex.4) could be rewritten under RATO:

$$\begin{aligned}&(R_0,R_1,R_2,R_3,R_4)>(r_0,r_1,r_2,r_3,r_4)\\&>(c_1,c_2,c_3,c_4,b_0,b_1,b_2,b_3,b_4)\\&>(a_0,a_1,a_2,a_3,a_4)>R>r>(A,B)\end{aligned}$$

Thus the candidate pair is (f_w,f_g), $f_w=R_0+r_4+b_0\cdot a_1$, $f_g=R_0\alpha^5+R_1\alpha^{10}+R_2\alpha^{20}+R_3\alpha^9+R_4\alpha^{18}+R$. Result after reduction is:

$$\begin{aligned}&Spoly(f_w,f_g)\xrightarrow{J+J_0}_+\\&r_1+(\alpha)r_2+(\alpha^4+\alpha^2)r_3+(\alpha^3+\alpha^2)r_4+(\alpha^3)b_1a_1+(\alpha^4+\alpha^2)b_1a_2\\&+(\alpha^3+\alpha+1)b_1a_3+(\alpha^3+\alpha)b_1a_4+(\alpha+1)b_1A+(\alpha^4+\alpha^2+\alpha)b_2a_1\\&+(\alpha^4+\alpha^3+\alpha^2+\alpha)b_2a_4+(\alpha^3+\alpha^2+1)b_3a_1+(\alpha)b_3a_3\\&+(\alpha^2+\alpha+1)b_4a_1+(\alpha+1)b_4a_2+(\alpha^4+\alpha^2)b_4a_3+(\alpha^4+\alpha^3+\alpha+1)b_4a_4\\&+(\alpha^3+1)b_4A+(\alpha^4+\alpha^3+\alpha^2+1)a_1B+(\alpha^4+\alpha^3+\alpha^2+1)R\end{aligned}$$

The remainder satisfies Lemma 1.

4.2 Bit-level Variable Substitution (BLVS)

The remainder from *Spoly* contains some bit-level variables, and our objective is to get a polynomial contains only word-level variables (such as $R+\mathcal{F}(A,B)$). One possible method is to rewrite bit-level variables in term of function on word-level variables, i.e. $a_i=\mathcal{G}(A)$, then do substitution. A Gaussian-elimination-fashion approach could be applied to compute corresponding $\mathcal{G}(A)$ efficiently.

Example 6. **Objective**: Abstract polynomial $a_i+\mathcal{G}_i(A)$ from f_0 : $a_0\alpha^5+a_1\alpha^{10}+a_2\alpha^{20}+a_3\alpha^9+a_4\alpha^{18}+A$.

First, compute $f_0^2: a_0\alpha^{10}+a_1\alpha^{20}+a_2\alpha^9+a_3\alpha^{18}+a_4\alpha^5+A^2$. Apparently variable a_0 can be eliminated by operation

$$\begin{aligned}f_1=f_0\times\alpha^5+f_0^2:&\\a_1+(\alpha)a_2+(\alpha^4+\alpha^2)a_3+(\alpha^3+\alpha^2)a_4&\\+(\alpha^4+\alpha^3+\alpha^2+1)A^2+(\alpha^2+\alpha)A&\end{aligned}$$

Recursively eliminate a_1 from f_1, a_2 from f_2, etc. The final polynomial f_4 has the form $a_4+\mathcal{G}_4(A)=a_4+(\alpha^4+\alpha^3+\alpha)A^{16}+(\alpha^3+\alpha^2)A^8+(\alpha^4+1)A^4+(\alpha^2+1)A^2+(\alpha+1)A$. Recursively substitute $\mathcal{G}_4(A)$ back to f_3, etc. The result is a set of polynomials

$$\{g_i \mid g_i : a_i+\mathcal{G}_i(A)\}$$

In this approach it is easy to get word-level variable representation for each bit-level primary inputs. By substitution, a new polynomial in the form $R+\mathcal{F}(A,B)$ could be attained.

Discussion (a) This approach won't conduct to a result reducing to 0, because *Spoly*'s remainder contains information from the whole system, the substitution will result something new, i.e. an abstraction of the system; (b) The Gaussian-like approach is a preprocessing of a variable, and could be immediately reused to other words with the same length; (c) For n-bits SMPO, the fast GB is very efficient: for each cycle, first compute a *Spoly* in $O(1)$, then do multi-division by $O(n)$ polynomials (unknown complexity, maybe lower if using F4-style reduction?), the substitution cost at most $O(n^3)$ time (assuming $O(1)$ for each term). In total repeat for n cycles. The complexity should be much lower compared to Buchberger's algorithm.

APPENDIX

A. NORMAL BASIS MULTIPLICATION

A.1 λ-Matrix

λ-Matrix is defined with cross-product terms from multiplication. That is

$$Product\ C = (\sum_{i=0}^{n-1} a_i \beta^{2^i})(\sum_{j=0}^{n-1} b_j \beta^{2^j}) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \beta^{2^i}\beta^{2^j}$$

The expressions $\beta^{2^i}\beta^{2^j}$ are referred to as cross-product terms, and can be represented by normal basis, i.e.

$$\beta^{2^i}\beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ij}^{(k)} \beta^{2^k},\ \ \lambda_{ij}^{(k)} \in F_2.$$

Substitution yields, get the expression for k-th digit of product:

$$c_k = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \lambda_{ij}^{(k)} a_i b_j$$

$\lambda_{ij}^{(k)}$ is the entry with coordinate (i, j) in k-th λ-Matrix.

Example 7. λ-Matrix: A binary $n \times n$ matrix M could be employed to describe the "function" mentioned in Ex.2: $c_{n-1} = f(A,B) = A \cdot M \cdot B^T$, B^T denotes vector transposition. More specifically, denote the matrix by *k-th λ-Matrix*: $c_k = A \cdot M^{(k)} \cdot B^T$. Then $c_{k-1} = A \cdot M^{(k-1)} \cdot B^T = rotate(A) \cdot M^{(k)} \cdot rotate(B)^T$, which means $M^{(k)}$ is generated by right and down cyclic shifting $M^{(k-1)}$.

In $\mathbb{F}_{2^3}$ constructed by $\alpha^3 + \alpha + 1$, let $\beta = \alpha^3$, $N = \{\beta, \beta^2, \beta^4\}$ is a normal basis. 0-th λ-Matrix

$$M^{(0)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

i.e.,

$$c_0 = (a_0\ a_1\ a_2) \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}.$$

A.2 Optimal Normal Basis

Definition 4. The number of non-zero entries in λ-Matrix is known as *Complexity*(C_N).

THEOREM 4. *If N is a normal basis for $\mathbb{F}_{p^n}$ with λ-matrix $M^{(k)}$, then non-zero entries in matrix $C_N \geq 2n-1$.*

Proof omitted.

Definition 5. If there exists a set of normal basis satisfying $C_N = 2n-1$, this normal basis is named as *Optimal Normal Basis (ONB)*.

There are 2 types of optimal normal basis existing. Rules for creating Type-I ONB over $\mathbb{F}_{2^n}$ are:

- n+1 must be prime.
- 2 must be primitive in $\mathbb{Z}_{n+1}$.

Rules for creating Type-II ONB over $\mathbb{F}_{2^n}$ are:

- 2n+1 must be prime. And either
- 2 must be primitive in $\mathbb{Z}_{2n+1}$, OR
- $2n+1 \equiv 3 \mod 4$ AND 2 generates the quadratic residues in $\mathbb{Z}_{2n+1}$

There are corresponding criteria for simply creating λ-Matrix for either type of ONB:

LEMMA 2. *Type-I ONB implies a λ-Matrix that each nonzero entry $M_{i,j}$ satisfies*

$$\begin{cases} 2^i + 2^j = 1 \bmod n+1 \\ 2^i + 2^j = 0 \bmod n+1 \end{cases}$$

Type-II ONB implies a λ-Matrix that each nonzero entry $M_{i,j}$ satisfies

$$\begin{cases} 2^i + 2^j = 1 \bmod 2n+1 \\ 2^i + 2^j = -1 \bmod 2n+1 \\ 2^i - 2^j = 1 \bmod 2n+1 \\ 2^i - 2^j = -1 \bmod 2n+1 \end{cases}$$

Proof omitted. (Issues here: We knew type-I & II definition can deduce corresponding λ-Matrix, how about reverse implication? i.e. can we prove the equivalence?)

A.3 Design a Normal Basis Multiplier using λ-Matrix

Imitating the structure of SMPO, just specify the gates' connections for the first clock cycle, then following cycles are automatically completed by cyclic shifting operands A and B. The whole design procedure is based on a $n \times n$ k-th λ-Matrix.

First figure out the first row (*row* 0) in λ-Matrix, for any nonzero entry $M_{0,j}$ (there will be only 1 if taking 0-th λ-Matrix), place an AND gate $a_j \wedge b_0$. Connect different a_j with a XOR gate before place the AND gate if there are 2 or more nonzero entries in this row;

Secondly, repeat this for each row. Note for nonzero entry $M_{i,j}$, corresponding index of a_u and b_v is incremented by i, i.e. $u = j + i \pmod n$, $v = i + i \pmod n$;

Finally, connect the output of AND gate for row i to one input of a XOR gate, then connect the output of XOR gate to a register/flip-flop. The output of register/flip-flop is connected to the other input of XOR gate, but at next row. All these gates and connections for row i is called unit R_i. (Fig.4)

B. FIND OPTIMAL NORMAL BASIS

B.1 Transforming to SAT

From last section we learned it is easy to compute λ-Matrix when given certain type of optimal normal basis. However in our approach, the mapping between optimal normal basis and standard basis should be explicitly specified. In other words, let *Optimal Normal Element* $\beta = \alpha^k$, the exponential k on primitive element α needs to be calculated. Otherwise word-level variable interpreting polynomial $a_0\beta + a_1\beta^2 + \cdots + a_{n-1}\beta^{2^{n-1}}$ will be an unknown polynomial and GB computation is impossible.

Currently finding the optimal normal element β in 2^n (over field $\mathbb{F}_{2^n}$) elements is NP-hard. However, it is possible to improve the performance by transforming this problem to SAT problem which can be solved by efficient SAT solver. This is what our tool is managing to do.

Given 0-th λ-Matrix $M^{(0)}$, other λ-Matrices can be obtained by cyclic right-down shifting, i.e. entry $M^{(k)}_{i,j} = M^{(0)}_{i-k,j-k}$. Note all index operation results should modulus n. β^3 is an element which can be represented as product of normal basis multiplication: $\beta^3 = \beta \cdot \beta^2$. Consider the following equation

$$c_k = (1\ 0\ 0\ \cdots\ 0)\, M^{(k)} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

c_k is the k-th coefficient in normal basis expression of β^3: $\beta^3 = c_0\beta + c_1\beta^2 + \cdots + c_{n-1}\beta^{2^{n-1}}$. Thus, β^3 could be written as

$$\beta^3 = \sum_{k=0}^{n-1} M^{(k)}_{0,1}\beta^{2^k} = \sum_{k=0}^{n-1} M^{(0)}_{-k,1-k}\beta^{2^k}$$

Meanwhile, the standard basis representation of optimal normal element can be assumed as

$$\beta = a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1}$$

Do a substitution, vanish every coefficient. Finally we can get a linear system of n boolean equations (composed by XOR and AND), with n unknowns $a_0, a_1, \ldots, a_{n-1}$. Examples are in the write-up by Florian's student.

B.2 Solving XOR-SAT

Solving the linear system of boolean equations is a tricky problem, because ordinary SAT solver only support Conjunctive Normal Form (CNF) inputs; while a XOR-SAT based solver *CryptoMiniSAT* only support clauses composed by single literals and XOR connectives ($a \oplus b \oplus (c \wedge d)$ is illegal input), I call it "XOR-CNF". Basically there are 2 approaches to think about our tool flow:

(a) Use another engine (Commercial Symbolic Computing System such as *Wolfram Mathematica*, or Circuit Based Synthesizer such as *ABC*) to transform all equations to CNF clauses. Advantage: Convenient, even no need to write our own tool in scripts; Disadvantage: cannot guarantee the efficiency, CNF may explode if we have hundreds of equations with hundreds of XOR terms.

(b) Imitate clause learning technique in CDCL, create new clauses to satisfy the input format of *CryptoMiniSAT*. Advantage: *CryptoMiniSAT* is a tool with performance enhancing on XOR clauses, and since the number of XOR clauses will not explode (I will explain later), maybe the efficiency can be guaranteed (somehow); Disadvantage: need to build our own tool for experiments.

Let me have a tiny statement on the second choice. First is the approach, maybe better illustrated by an example:

Example 8. Change boolean equation $a \oplus b \oplus c \oplus bc = 0$ to XOR-CNF. First, a SAT-solver always want whole CNF to be TRUE. Fortunately, we can negate a XOR clause by just negate one literal: $\overline{a} \oplus b \oplus c \oplus bc = 1$. Then we need to solve the non-single-literal term. Replace term bc with new literal d, and create a new clause to make $d = bc$, which is $(\overline{b} \vee \overline{c} \vee d) \wedge (b \vee \overline{d}) \wedge (c \vee \overline{d})$, adding 3 short clauses.

You may ask why the number and size of clauses won't explode, the answer is easy: we only have 2-literal terms in worst case! Note all equations come from

$$\beta^3 + c_0\beta + c_1\beta + \cdots + c_{n-1}\beta^{2^{n-1}} = 0$$

Even exponentials will never generate higher order coefficients in $\mathbb{F}_{2^n}$, so the only term generates higher-order coefficients is

$$\beta^3 = (a_0 + a_1\alpha + \cdots a_{n-1}\alpha^{n-1})(a_0 + a_1\alpha^2 + \cdots + a_{n-1}\alpha^{2(n-1)})$$

Apparently cross-product terms here have at most 2 literals (degree 2).

Acknowledging this fact, for n unknowns there will be at most $\binom{n}{2} = O(n^2)$ 2-literal terms, so the number of newly created clause will also be $O(n^2)$, with fixed tiny size.

By either approach, we finally get one legal assignment for bit vector $(a_0, a_1, \ldots, c_{n-1})$ from the SAT-solver, by that we can compute β. We can directly use it as ONB, or setup a loop to keep squaring β to find the minimum k for α^k, and adopt that one as ONB.

Discussion The tool flow is feasible enough, so only issue is the soundness from the ONB theory to linear system of boolean equations. I asked following questions to Florian's student: 1. Solutions for linear system of equations in $\mathbb{F}_2$ (or say $\mathbb{Z}_2$). For example, in last page of ONB.pdf194 KB, there are 5 variables and 5 equations, provide 5 conjugate solutions forming a set of (optimal) normal basis. Is it possible to dig into this, say to prove all solutions are conjugates $(x, x^2, x^4, x^8, \ldots)$? 2. You chose β^3 to deduce those equations. Does solution to β^3 necessarily cover (or equal to) solutions for given lambda-Matrix M0? In other words, if I choose β^5 or β^6 to construct those equations, can I get the same answer?

If above concerns are resolved, we can come up with a complete proof.