Resolving Unknown Components In Finite Field Arithmetic Circuits Using Computer Algebra Methods

Vikas Rao[1], Priyank Kalla[1], Utkarsh Gupta[1] and Florian Enescu[2]
[1]Electrical & Computer Engineering, University of Utah
[2]Mathematics & Statistics, Georgia State University

I. Introduction

Verifying functional correctness of gate level arithmetic circuits is still a significant challenge owing to increasing design size and functional complexity. Considerable amount of manual intervention is required to localize a bug and add correction, thus making it a resource intensive process. Traditional automated debugging techniques based on simulation, decision procedures such as Binary Decision Diagrams(BDDs)[1] and SAT solvers[2], demand bit-blasting of the circuit and are hence considered inefficient models to verify complex datapath designs. Due to inherent algebraic nature of computations in such designs, symbolic algebra algorithms are considered more appropriate for their verification.

Within symbolic algebra environment, a given circuit implementation is modeled as a set of polynomials that generate an ideal. The verification goal here is then to prove that this polynomial ideal satisfies a given golden specification, by performing a series of divisions. If the verification fails, we deem the circuit as buggy and go on to find the faulty gate in order to rectify it. The current challenge and scope of this paper is to realize the correct implementation for this buggy component. Identifying the buggy gate is a much harder problem to solve and is in the future scope of work. Once a particular gate has been identified as suspicious, we label the gate as an unknown component and go on to find the correct functionality implemented by this component such that the entire circuit conforms to the given reference specification.

A. Previous work

The most recent and relevant approach [5][6] resolves the unknown component problem using an incremental *Boolean Satisfiability*(SAT) formulation. The paper models the unknown component in a given circuit(Ckt) as a LUT by using transformation variables(X). The solution to these variables will implement the desired logic function such that the resulting circuit becomes logically equivalent to a given specification $Spec()$. Let $Ckt(X, In)$ be the formula corresponding to the given circuit with possible transformations, where In is the set of all primary inputs to the circuit. This can be formulated naturally as a two-level QBF with a existential quantifier followed by a universal quantifier as shown below:

$$\exists X. \forall In.\ Ckt(X,In) = Spec(In) : \qquad (1)$$

The two level QBF is then solved by repeatedly applying the below SAT formulation:

1) Let Target:$Ckt(X, In) \neq spec(In)$, and let k be the number of test vectors, initialized to zero. Let $TestSet$ be the set of all generated test patterns, initialized to empty set.
2) Check if Target is satisfiable.
3) If SAT, $k = k + 1$ and record the solution as $TestSet = TestSet \cup in_k$. The Target is then updated as Target = (Target$(X, In)) \wedge (Ckt(X, in_k) = Spec(in_k))$, and go to step 2.
4) If UNSAT, we have all the required test set patterns $\{in_1 \dots in_k\}$. Now, check if: $(Ckt(X, in_1) = Spec(in_1)) \wedge (Ckt(X, in_2) = Spec(in_2)) \wedge \dots (Ckt(X, in_k) = Spec(in_k))$ is satisfiable.
5) If SAT, then any solution X is a correct set of transformation, while an UNSAT result proves that there does not exist a correct set of transformation.

By experiment, the approach shows that if the circuit is correct under these input patterns(in_k), it is guaranteed to be correct for all of 2^{In} input patterns.

The work in [7] poses the unknown component formulation as a camouflaged circuit model and tries to de-obfuscate several types of camouflaging techniques using incremental SAT solving. The approach used in [8] inserts logic corrector MUXs on the unknown sub-circuits and relies on SAT solvers to realize the functionality.

Despite using the stat-of-the-art SAT solvers, all the above approaches fail to verify large and complex arithmetic circuits. The solvers still model the problem as decision procedures and, as demonstrated by our experimental results, are shown to be inefficient in solving verification problems on multiplier circuits beyond 12-bits.

Techniques from Farimah et. al [9] deals with automatic debugging and correction using computer algebra concepts. The paper uses function extraction[10] with a specific term order[11] to do equivalence checking, subsequently generating a remainder in case of failure. The approach then finds all possible assignments to variables of the remainder such that it generates a non-zero value. This test set helps arrive at a pruned gate list for bug localization. The procedure then takes every gate in the pruned list, starting from primary inputs, and tries to match the appeared remainders pattern. It does so by computing the difference between the polynomial computed at the output of the suspicious gate against the polynomial computed by a probable set of gate corrections. The coefficient computation[12] during pattern matching relies heavily on the half-adder(HA) based circuit structure. The paper doesn't discuss the ambiguities in weight calculations when the gate structure differs from the given topology. The approach also fails to arrive at a conclusive solution when the circuit is tweaked with some redundancy and hence lacks completeness.

B. Contribution

For a given circuit implementation C, we will utilize concepts from computer algebra to realize the function implemented by the unknown component, and then prove the completeness of the approach by extending it to random logic circuits. For simplicity, we will take a single gate replacement error model as our target design i.e., only one gate in the design incorrectly replaced, for example an AND gate replaced with an XOR/OR gate. The reference golden model can either be a specification polynomial f or a structurally different circuit C_1 implementing the same function. For a given specification polynomial f, we do polynomial reduction until the unknown component gate and arrive at the function implemented by the component by using Gröbner basis based guided ideal membership testing, projection of variety, and quotient of ideals. For the case where the specification is given in terms of a different implementation C_1, we use Gröbner basis based reduction on a miter setup and apply *Nullstellensatz* principles to arrive at the function implemented by the component. This paper seeks to outline the verification challenges and present a complete approach with some preliminary but encouraging experimental results.

II. Preliminaries: Notation and Background

This section reviews some basic concepts from the computer algebra that we utilize in our theory.

Let $\mathbb{F}_q$ denote the finite field of q elements, where $q = p^k$ is some exponent (k is a positive integer) of a prime number p. We denote by $R = \mathbb{F}_2[x_1, \dots, x_n]$ the polynomial ring over variables $x_1, \dots, x_n$ with coefficients in $\mathbb{F}_q$. A polynomial $f \in R$ 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, i.e. power products of the type $x_1^{e_1} \cdot x_2^{e_2} \cdots x_n^{e_n}$, $e_i \in \mathbb{Z}_{\geq 0}$. To systematically manipulate the polynomials, a monomial order $>$ (also called a term order $e.g.$ lexicographical order) is imposed on the polynomial ring such that $X_1 > X_2 > \cdots > X_t$. Subject to $>$, $lt(f) = c_1X_1$, $lm(f) = X_1$, $lc(f) = c_1$, are the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. Also, for a polynomial f, $tail(f) = f - lt(f)$.

Gates of a circuit can be modeled with polynomials in $\mathbb{F}_2[x_1, \dots, x_n]$, where every Boolean logic gate operator is mapped to a polynomial function over $\mathbb{F}_2$:

$$\begin{aligned} z &= \neg a \;\rightarrow\; z + a + 1 \pmod 2 \\ z &= a \wedge b \;\rightarrow\; z + a \cdot b \pmod 2 \\ z &= a \vee b \;\rightarrow\; z + a + b + a \cdot b \pmod 2 \\ z &= a \oplus b \;\rightarrow\; z + a + b \pmod 2 \end{aligned} \qquad (2)$$

Polynomial Reduction via division: Let f, g be polynomials. If $lm(f)$ is divisible by $lm(g)$, then we say that f *is reducible to* r modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{lt(f)}{lt(g)} \cdot g$. Similarly, f can be *reduced w.r.t. a set of polynomials* $F = \{f_1, \dots, f_s\}$ to obtain a remainder r. This reduction is denoted as $f \xrightarrow{F}_+ r$, 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. Algorithm 1 ([3]) shows the step-by-step procedure to perform this reduction.

Algorithm 1 Multivariate Reduction of f by $F = \{f_1, \dots, f_s\}$

```
1:  procedure $multi\_variate\_division(f, \{f_1, \dots, f_s\}, f_i \neq 0)$
2:      $u_i \leftarrow 0;\ r \leftarrow 0,\ h \leftarrow f$
3:      while $h \neq 0$ do
4:          if $\exists i$ s.t. $lm(f_i) \mid lm(h)$ then
5:              choose $i$ least s.t. $lm(f_i) \mid lm(h)$
6:              $u_i = u_i + \frac{lt(h)}{lt(f_i)}$
7:              $h = h - \frac{lt(h)}{lt(f_i)} f_i$
8:          else
9:              $r = r + lt(h)$
10:             $h = h - lt(h)$
11:     return $(\{u_1, \dots, u_s\}, r)$
```

The algorithm initializes h with the polynomial f and cancels its leading term by some polynomial f_i. If the leading term $lt(h)$ cannot be canceled by any $lt(f_i)$, then it is added to the final remainder r and process is repeated until all the terms in h are analyzed. The algorithm also returns the set of quotients $u_1, \dots, u_s$ of division of f by $f_1, \dots, f_s$, respectively.

Polynomial Ideals and Varieties: Given a set of polynomials $F = \{f_1, \dots, f_s\} \in \mathbb{F}_q[x_1, \dots, x_n]$, denote the ideal J generated by F as $J = \langle F \rangle = \langle f_1, \dots, f_s \rangle = \{\sum_{i=1}^{s} h_i \cdot f_i :\ h_i \in R\}$. The ideal J may have many different generators, i.e. it is possible to have $J = \langle f_1, \dots, f_s \rangle = \langle g_1, \dots, g_t \rangle = \cdots = \langle h_1, \dots, h_r \rangle$. A Gröbner basis (GB) G of ideal J is one such set of polynomials $G = GB(J) = \{g_1, \dots, g_t\}$ that is a canonical representation of the ideal.

Definition II.1. [Gröbner Basis] [3]: 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 $lm(f)$.

The Gröbner basis for an ideal J can be calculated using the Buchberger's algorithm (see Alg. 1.7.1 in [3]) which takes as input a set of polynomials $\{f_1, \dots, f_s\}$ and computes the GB $G = \{g_1, g_2, \cdots, g_t\}$. A GB can be *reduced* to eliminate redundant polynomials from the basis. A reduced GB is a canonical representation of the ideal.

Given two ideals $J_1 = \langle f_1, \dots, f_s \rangle, J_2 = \langle h_1, \dots, h_r \rangle$, their sum $J_1 + J_2 = \langle f_1, \dots, f_s, h_1 \dots, h_r \rangle$, and their product $J_1 \cdot J_2 = \langle f_i \cdot h_j : 1 \leq i \leq s, 1 \leq j \leq r \rangle$. Ideals and varieties are dual concepts: $V(J_1 + J_2) = V(J_1) \cap V(J_2)$, and $V(J_1 \cdot J_2) = V(J_1) \cup V(J_2)$. Moreover, if $J_1 \subseteq J_2$ then $V(J_1) \supseteq V(J_2)$.

For an ideal $J = \langle f_1, \dots, f_s \rangle$, the *variety* of J over $\mathbb{F}_q$ is denoted by $V_{\mathbb{F}_q}(J)$ and defined as,

$$V_{\mathbb{F}_q}(J) = \{\mathbf{a} \in \mathbb{F}_q^n : \forall f \in J, f(\mathbf{a}) = 0\}$$

Therefore, $V_{\mathbb{F}_q}(J)$ is the set of all the points in $\mathbb{F}_q^n$ that are solutions to $f_1 = f_2 = \cdots = f_s = 0$. Varieties can be different when restricted to the given field $\mathbb{F}_q$ or considered over its algebraic closure $\overline{\mathbb{F}_q}$.

For all elements $\alpha \in \mathbb{F}_q, \alpha^q = \alpha$. Therefore, the polynomial $x^q - x$ vanishes (the polynomial evaluation is zero) everywhere in $\mathbb{F}_q$, and is called the vanishing polynomial of the field. We denote by $J_0 = \langle x_1^q - x_1, \dots, x_n^q - x_n \rangle$ the ideal of all vanishing polynomials in the ring R. Then $V_{\mathbb{F}_q}(J_0) = V_{\overline{\mathbb{F}_q}}(J_0) = \mathbb{F}_q^n$. Therefore, given any ideal J, $V_{\mathbb{F}_q}(J) = V_{\overline{\mathbb{F}_q}}(J) \cap \mathbb{F}_q^n = V_{\overline{\mathbb{F}_q}}(J) \cap V_{\overline{\mathbb{F}_q}}(J_0) = V_{\overline{\mathbb{F}_q}}(J + J_0) = V_{\mathbb{F}_q}(J + J_0)$ ([13]).

Reverse Topological Term Order [14]: The computational complexity of Buchberger's algorithm is exponential in the number of variables n. As our work is focused on the circuits, we will describe a term order that renders the set of polynomials for the gates of the circuit, a Gröbner basis itself. This term order is called Reverse Topological Term Order (RTTO).

Let C be an arbitrary combinational circuit. Let $\{x_1, \dots, x_n\}$ denote the set of all variables (signals) in C. Starting from the primary outputs, perform a *reverse topological traversal* of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lex term order $>$ to represent each gate as a polynomial f_i, s.t. $f_i = x_i + tail(f_i)$. Then set of polynomials $\{f_1, \dots, f_s\}$ corresponding to the gates of the circuits is a Gröbner basis when RTTO is used for ordering.

Weak Nullstellensatz and Elimination Theory:

Theorem II.1 (*The Weak Nullstellensatz over finite fields (from Theorem 3.3 in [13])*)**.** *For a finite field $\mathbb{F}_q$ and the ring $R = \mathbb{F}_q[x_1, \dots, x_n]$, let $J = \langle f_1, \dots, f_s \rangle \subseteq R$, and let $J_0 = \langle x_1^q - x_1, \dots, x_n^q - x_n \rangle$ be the ideal of vanishing polynomials. Then $V_{\mathbb{F}_q}(J) = \emptyset \iff 1 \in J + J_0 \iff G = GB(J + J_0) = \{1\}$.*

To find whether a set of polynomials $f_1, \dots, f_s$ have no common zeros in $\mathbb{F}_q$, we can compute the GB G of $\{f_1, \dots, f_s, x_1^q - x_1, \dots, x_n^q - x_n\}$ and see if $G = \{1\}$.

Definition II.2 ([4])**.** Given an ideal $J \subset \mathbb{F}_q[x_1, \dots, x_n]$, the l-th elimination ideal J_l is an ideal in R defined as $J_l = J \cap \mathbb{F}_q[x_{l+1}, \dots, x_n]$.

The ideal J_l is called an elimination ideal because the variables $x_1, \dots, x_{l-1}$ have been eliminated. The next theorem shows how we can obtain the generators of the l-th elimination ideal using Gröbner bases.

Theorem II.2 (*Elimination Theorem [4]*)**.** Given an ideal $J \subset R$ and its GB G $w.r.t.$ the lexicographical (lex) order on the variables where $x_1 > x_2 > \cdots > x_n$, then for every $0 \le l \le n$ we denote by G_l the GB of l-th elimination ideal of J and compute it as:

$$G_l = G \cap \mathbb{F}_q[x_{l+1}, \dots, x_n]$$

We also need to employ notion of difference of varieties in our theoretical section. The equivalent ideal operation is called the quotient of ideals.

Definition II.3. (Quotient of Ideals) If J_1 and J_2 are ideals in a ring R, then $J_1 : J_2$ is the set $\{f \in R \mid f \cdot g \in J_1, \forall g \in J_2\}$ and is called the **ideal quotient** of J_1 by J_2.

In terms of varieties, $V_{\mathbb{F}_q}(J_1 : J_2) = V_{\mathbb{F}_q}(J_1) \setminus V_{\mathbb{F}_q}(J_2)$. The computation of elimination ideal is based on the intersection of ideals. For details on the computation of elimination ideal, we refer the reader to [4].

The computer algebra tools like SINGULAR [15] contain implementations for computing elimination ideals and quotient of ideals.

References

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

[2] 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.

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

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

[5] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design," *Proceedings of the IEEE*, 2015.

[6] S. Jo, T. Matsumoto, and M. Fujita, "SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions," in *IEEE 21st Asian Test Symposium*, 2012.

[7] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[8] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[9] F. Farahmandi and P. Mishra, "Automated Test Generation for Debugging Arithmetic Circuits," in *Design, Automation Test in Europe Conference Exhibition(DATE)*, 2016.

[10] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of Gate-level Arithmetic Circuits by Function Extraction," in *Proceedings of the 52Nd Annual Design Automation Conference*, 2015.

[11] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Multipliers," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012.

[12] S. Ghandali, C. Yu, D. Liu, W. Brown, and M. Ciesielski, "Logic Debugging of Arithmetic Circuits," in *IEEE Computer Society Annual Symposium on VLSI*, 2015.

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

[14] 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.

[15] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 4-1-0 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2016.

[16] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal Verification of Arithmetic Circuits by Function Extraction," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System*, 2016.