BOOLEAN GRÖBNER BASIS REDUCTIONS ON DATAPATH CIRCUITS USING THE UNATE CUBE SET ALGEBRA

Utkarsh Gupta, Priyank Kalla, *Senior Member, IEEE*, Vikas Rao

***Abstract*—Recent developments in formal verification of arithmetic datapaths make efficient use of symbolic computer algebra algorithms. The circuit is modeled as an ideal in polynomial rings, and Gröbner basis (GB) reductions are performed over these polynomials to derive a canonical representation. As they model logic gates of the circuit, the ideals comprise largely of Boolean (or pseudo-Boolean) polynomials. GB reductions modulo these Boolean polynomials tend to cause intermediate expression swell (term explosion problem) – often rendering the approach infeasible in a practical setting. These problems can be overcome by using an implicit data-structure that represents these polynomials compactly, provides the algorithmic flexibility to perform GB-reduction implicitly, and allows to exploit the information derived from the topology of the circuits to improve symbolic computation.**

This paper considers a logic synthesis analogue of GB reductions over Boolean polynomials, by interpreting symbolic algebra as the unate cube set algebra over characteristic sets. By representing Boolean polynomials as characteristic sets using Zero-suppressed BDDs (ZBDDs), implicit algorithms are efficiently designed for GB-reduction for datapath circuits. While polynomial manipulation algorithms have been implemented on ZBDDs before, we show that the imposition of circuit-topology based monomial orders exposes a special structure on the ZBDD representation of the polynomials. The subexpressions employed in the GB-reduction are readily visible as subgraphs on the ZBDDs, which are directly used to compose the result. Our division algorithms effectively cancel multiple monomials implicitly in one-step, simplify the search for divisors, and avoid intermediate size explosion. Experiments performed over various arithmetic architectures demonstrate the efficiency of our algorithms and implementations; our approach is orders of magnitude faster as compared to conventional methods.

I. INTRODUCTION

Automated formal verification and equivalence checking of arithmetic datapath circuits is challenging. Conventional verification techniques, such as those based on binary decision diagrams (BDDs) [1], And-Invert-Graph (AIG) based reductions with SAT or SMT-solvers [2], etc., are infeasible in verifying complex datapath designs. Such designs often implement algebraic computations over bit-vector operands, therefore finite integer rings [3] [4] or finite fields [5] [6] are considered appropriate models to devise decision procedures for verification. For this reason, the verification community has explored the use of algebraic geometry and symbolic algebra algorithms for verification.

In such a setting, the logic gates of the circuit are modeled by way of a set of multivariate polynomials $F = \{f_1, \ldots, f_s\}$ in rings $R[x_1, \ldots, x_n]$. Usually, the coefficients $R = \mathbb{Z}, \mathbb{Z}_{2^k}$, or $\mathbb{F}_{2^k}$, depending on whether the integer, finite integer ring $(\text{mod } 2^k)$, or respectively the finite field (of 2^k elements) model is employed for verification. This set of polynomials F generates an ideal, and for verification it is required to compute a *Gröbner basis (GB)* [7] of this ideal. Reducing the primary output polynomials (or variables) of the circuit modulo this GB results in a unique canonical polynomial expression, and it can be used for equivalence checking.

The GB problem exhibits high computational complexity. Indeed, computing a GB (using Buchberger's [8] or the F_4 algorithm [9]) for large circuits is practically infeasible. Managing this complexity ought to be a major goal of any approach.

1) State-of-the-art & Limitations: Recent approaches [3] [5] have discovered that particularly for circuit verification problems, the expensive GB computation can be avoided altogether. For arbitrary combinational [3] [5] and sequential circuits [10], a specialized term order $>$ can be derived by analyzing the topology of the given circuit. This term order is derived by performing a reverse topological traversal of the circuit, and in this manuscript we refer to it as the *Reverse Topological Term Order* (RTTO). Imposition of RTTO $>$ on the polynomial ring *renders the set of polynomials of the circuit itself a GB.* Subsequently, the verification problems can be solved solely by way of GB-reduction (using multi-variate polynomial division), without any need to explicitly compute a GB. It has now become standard practice to make use of RTTO-style term orders to solve various formal verification problems on digital circuits (see for example [3], [5], [4], [11], [12], [13]), where the early techniques of [3] [5] have been extended and improved to verify integer and floating point arithmetic circuits [11], [13], [14].

A common theme among all these relevant works is that *by virtue of RTTO, they move the complexity of verification from one of computing a GB to that of GB-reduction by way of multivariate polynomial division.* Moreover, since the Gröbner basis is derived from the logic gates of the circuit, *it comprises Boolean polynomials (Boolean polynomials have coefficients* $\in \{0,1\}$ *and degrees* $\in \{0,1\}$*).* For example, in the verification of finite field circuits, the overall problem is modeled over the ring $\mathbb{F}_{2^k}[x_1, \ldots, x_n]$; and since $\mathbb{F}_{2^k} \supset \mathbb{F}_2$, the approaches incorporate computations over many Boolean polynomials. Similarly, the techniques of [4] use a pseudo-Boolean model (pseudo-Boolean polynomials have coefficients $\in \mathbb{Z}$ and degrees $\in \{0,1\}$) which also encompasses computations $(\text{mod } x^2 = x)$; [4] refers to it as arithmetic bit-level (ABL). All the aforementioned approaches will benefit greatly by a *dedicated, domain-specific implementation of GB-reduction w.r.t. Boolean polynomials, carried out on the given circuit under RTTO* $>$. So far, the above techniques [3], [6],

This work has been supported in part by grants from the US National Science Foundation CCF-1320335 and CCF-1619370. The authors are with the Electrical & Computer Engineering Department at the University of Utah in Salt Lake City, USA. Contact author: Priyank Kalla (kalla@ece.utah.edu).

[5], [13], [11], [15] use a general-purpose polynomial division approach, together with explicit representation, for this GB-reduction. Moreover, the overall concept of polynomial division is still utilized in its rudimentary form, involving iterative cancellation of monomials "1-step at a time" on explicit data-structures. Despite recent efforts, such GB-reductions can lead to *a worst-case size explosion problem*, which needs to be addressed.

2) Objective & Rationale: This paper addresses the problem of deriving canonical representations for datapath circuits as Boolean polynomials. These canonical Boolean polynomials are used for equivalence checking of datapath designs. The canonical representation requires a Gröbner basis reduction modulo a set of Boolean polynomials. This reduction can result in space and time explosion and make verification infeasible. To make this GB-reduction on circuits more efficient, this paper describes new techniques, algorithms and implementations, specifically targeted for circuit verification under RTTO $>$. Particularly, we make use of the *implicit* characteristic set representation for storing and manipulating Boolean polynomials using *Zero-Suppressed BDDs (ZBDDs)* [16]. By analyzing the structure of ZBDDs for polynomial representation under RTTO $>$, we show how this GB-reduction can be efficiently implemented using algorithms that specifically manipulate the ZBDD graph, by interpreting Boolean polynomial manipulation as the *algebra of unate cube sets.*

The algebraic objects used to model the polynomial ideals derived from digital circuits are rings of Boolean polynomials. When Boolean functions are represented in $\mathbb{F}_2$ (AND/XOR expressions), and that too as a canonical Gröbner basis, the representation tends to explode. Polynomial representations employed in computer algebra tools, such as the *dense-distributive data-structure* of the SINGULAR computer algebra tool [17], are inefficient for this purpose. Since addition and multiplication $\pmod 2$ are equivalent to XOR and AND operations, respectively, GB-reduction can be viewed as a polynomial analog of a specialized *AND/XOR Boolean decomposition* problem. Moreover, the monomials of a Boolean polynomial are a product of literals in positive polarity, which can be viewed as *unate cubes* in logic synthesis. Clearly, implicit Boolean set representations such as decision diagrams could be employed for Gröbner basis reductions over Boolean polynomials.

3) Technical Contributions: We first describe when and how the GB-reduction encounters a term-explosion (exponential blow-up) under RTTO $>$, which cannot be easily overcome by explicit representations. We show that ZBDDs can avoid this exponential blow-up – thereby justifying their use. We describe how Boolean polynomials can be represented using ZBDDs under the special term ordering constraint imposed due to RTTO $>$. The implementation of classical polynomial division algorithms that iteratively cancel one monomial in every step is described on ZBDDs. Subsequently, *we show that RTTO $>$ imposes a special structure on ZBDDs that allows to implement reduction techniques that implicitly cancel multiple monomials in every step of polynomial division*. Moreover, due to RTTO $>$, the subexpressions that are required for polynomial division are also readily available as subgraphs in the ZBDDs. Our algorithm exploits this special structure, thus improving GB-reduction in both space and time.

Using an implementation integrated with the CUDD [18] package, we perform extensive experiments on datapath circuits for deriving the canonical representation of the functions implemented by them. The benchmark designs include various cryptography primitives, such as finite field multipliers, elliptic curve point addition circuits, and also sequential finite field circuits. Experiments conducted on these benchmarks show *orders of magnitude improvement* using our implementation of GB-reduction, as compared against contemporary methods. In fact, for these benchmarks, our bit-level (Boolean) approach is much faster than the word-level approaches (e.g. [6]).

We also describe the limitations of our approach when it is applied to integer arithmetic circuits. We analyze these limitations and show that for integer arithmetic circuits, a word-level symbolic reasoning engine is needed to control the monomial explosion problem. This is not a limitation of our algorithms, but rather of the bit-level model in verifying integer arithmetic circuits. Finally, we are not concerned with equivalence checking of random logic circuits; AIG-based reductions for SAT-based verification techniques [2] are more suitable and efficient for such applications.

4) Paper Organization: The following section reviews relevant previous work on Gröbner basis based verification of datapath circuits, and the literature on Boolean Gröbner basis and applications. Section III describes the mathematical background on Gröbner basis reductions, the RTTO based term order $>$, and how these concepts are applied to datapath verification. Section IV motivates how and why the Boolean GB-reduction can be viewed as the unate cube set algebra. Section V describes the theory, algorithms and implementations for GB-reduction for Boolean polynomials using ZBDDs. Section VI describes the experiments conducted for verification using our approach, and Sec VII concludes the paper.

II. RELATED PREVIOUS WORK

In the past decade, computer algebra and algebraic geometry based datapath verification has received a lot of attention, where the verification problems are formulated in terms of ideal membership, canonical GB-reductions, or projections of varieties. The work of [19] used a GB-reduction approach to derive canonical representations of (word-level) RTL datapath descriptions over finite rings $\mathbb{Z}_{2^k}$. Using the same finite ring model, [3] addressed data correctness properties of arithmetic bit-level implementations using a Gröbner basis formulation. This paper showed how an efficient term order $>$ can be derived from the circuit to simplify the computation.

In [20] and [5], the authors addressed formal verification of finite field arithmetic circuits using the Strong Nullstellensatz formulation over $\mathbb{F}_{2^k}$. Using a set (F) of polynomials to describe the logic circuit, along with a set of vanishing polynomials (F_0) over the field $\mathbb{F}_{2^k}$, the verification problem was formulated as a (radical) ideal membership test, requiring a Gröbner basis. Drawing inspirations from [3], the authors in [20] also exploited the same concept of deriving a specialized term order $>$ to simplify the ideal membership test. In particular, it was shown that $>$ could be derived by performing a

reverse topological traversal on the circuit. Imposition of this term order $>$ rendered the set of polynomials $F \cup F_0$ itself a Gröbner basis, and verification was then performed simply by a GB-reduction. This GB-reduction was subsequently formulated as Gaussian elimination on a coefficient matrix [5], [6] in the style of the F_4 algorithm; called F_4-style reduction in the sequel.

Formulations of a similar flavor (GB-reduction under the specialized term order $>$ derived from the circuit) were used and integrated with SMT-solvers [4] for verification using a pseudo-Boolean model (akin to $\mathbb{Z}_{2^k}[X] \pmod{X^2 - X}$). More recently, these concepts have been applied to verify integer arithmetic [11], [13], and also floating point circuits [14]. The authors in [15] show that the reduction process can be *parallelized* by performing reduction for each output bit independently.

Polynomial division algorithms form the core computation in the above techniques. Almost all of the aforementioned techniques use a classical polynomial division approach. While some of these approaches do perform the reduction in some specific ways – e.g., mimicking GB-reduction under RTTO $>$ by substitution [13], or using TEDs to perform input-output signature comparisons [21], or the use of F_4-style GB-reduction on a coefficient matrix [5], [6] – the overall concept of polynomial division is still utilized in its rudimentary form, involving iterative cancellation of monomials "1-step at a time" on explicit data-structures.

A. Boolean Gröbner Basis

The symbolic algebra community has studied properties of Boolean Gröbner bases [22] [23]. Boolean GB formulations have also been used for SAT solving [24], e.g. to derive proof refutation, and for model checking [25] [26]. From among these, the work of PolyBori [23] comes closest to ours, and is a source of inspiration for this work. PolyBori proposed the use of ZBDDs to compute Gröbner bases for Boolean polynomials. PolyBori is a *generic* Boolean GB computational engine that caters to many permissible term orders. Its division algorithm is also generally based on the conventional concept of canceling one monomial in every step of reduction. In contrast, our algorithms are tailored for GB-reduction under the RTTO $>$. The efficiency of our approach stems from the observation that the RTTO $>$ imposes a special structure on the ZBDDs, which allows for multiple monomials to be canceled in one division-step, along with simplifying the search for divisors. Experiments show that our approach is an order of magnitude faster than PolyBori.

III. Background: Gröbner Basis Reduction and Canonical Representations

Let $\mathbb{B} = \{0,1\}$ denote the Boolean domain, $\mathbb{F}_2$ the finite field of 2 elements ($\mathbb{B} \equiv \mathbb{F}_2$), and $R = \mathbb{F}_2[x_1, \ldots, x_n]$ the polynomial ring over variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{F}_2$. Operations in $\mathbb{F}_2$ are performed $\pmod 2$, so $-1 = +1$ in $\mathbb{F}_2$. We will use $+, \cdot$ to denote addition and multiplication in R, and $\neg, \vee, \wedge$ and $\oplus$ to denote Boolean negation, OR, AND and XOR operations, respectively.

A polynomial $f \in R$ is written as a finite sum of terms $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$. Here $c_1, \ldots, c_t$ are coefficients and $X_1, \ldots, X_t$ 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) is imposed on the ring. This order $>$ is a total order and a well order on all the monomials of R such that multiplication by a monomial preserves the order[1]. All polynomials in R are represented using $>$. Subject to $>$, when f is written as $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$ such that $X_1 > X_2 > \cdots > X_t$, we call $lt(f) = c_1 X_1$, $lm(f) = X_1$, $lc(f) = c_1$, the *leading term*, *leading monomial* and *leading coefficient* of f, respectively. We also denote tail(f) = $f - lt(f) = c_2 X_2 + \cdots + c_t X_t$. In this work, we are mostly concerned with terms ordered lexicographically (*lex*).

Definition III.1 (Boolean Polynomial)**.** *Let $f = c_1 X_1 + \cdots + c_t X_t$ be a polynomial in $\mathbb{F}_2[x_1, \ldots, x_n]$ such that the coefficients $c_i \in \{0,1\}$, and monomials $X = x_1^{e_1} \cdot x_2^{e_2} \cdots x_n^{e_n}, e_i \in \{0,1\}$. Then f is called a* **Boolean polynomial***. For Boolean polynomials $lt(f) = lm(f)$.*

A gate-level circuit can be modeled with Boolean polynomials, where every Boolean logic gate operator is mapped from $\mathbb{B}$ to a polynomial function over $\mathbb{F}_2$:

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

Polynomial reduction via division: Let f, g be polynomials. If $lt(f)$ is divisible by $lt(g)$, then we say that f *is reducible to* r modulo g, denoted $f \xrightarrow{g} r$, where $r = f - \frac{lt(f)}{lt(g)} \cdot g$. This operation forms the core operation of polynomial division algorithms and it has the effect of canceling the leading term of f. Similarly, f can be *reduced w.r.t. a set of polynomials* $F = \{f_1, \ldots, f_s\}$ 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 (i.e. cannot be canceled) by the leading term of any polynomial f_i in F. Algorithm 1 (Alg. 1.5.1 from [7]) depicts the procedure to perform this classical reduction that cancels one monomial in every iteration of the while-loop.

[1]Lexicographic (*lex*) and degree-lexicographic (*deglex*) are examples of such permissible monomial orders.

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

procedure *multi_variate_reduce*$(f, \{f_1, \ldots, f_s\}, f_i \neq 0)$
  $u_i \leftarrow 0$; $r \leftarrow 0$; $h \leftarrow f$
  while $h \neq 0$ **do**
    if $\exists i$ s.t. $lm(f_i) \mid lm(h)$ **then**
      choose i least s.t. $lm(f_i) \mid lm(h)$
      $u_i = u_i + \frac{lt(h)}{lt(f_i)}$
      $h = h - \frac{lt(h)}{lt(f_i)} f_i$
    else
      $r = r + lt(h)$
      $h = h - lt(h)$
  return $(\{u_1, \ldots, u_s\}, r)$

The algorithm initializes h with the polynomial f and cancels its leading term by some polynomial $f_i \in F$. If the leading term $lt(h)$ cannot be canceled by any $lt(f_i)$, then it is added to the remainder r and the process is repeated until all the terms in h are analyzed.

Polynomial ideals: Given a set of polynomials $F = \{f_1, \ldots, f_s\}$ from the ring $R = \mathbb{F}_2[x_1, \ldots, x_n]$, the ideal generated by F is $J = \langle F \rangle \subseteq R$:

$$J = \langle f_1, \ldots, f_s \rangle = \{h_1 \cdot f_1 + \cdots + h_s \cdot f_s : \ h_1, \ldots, h_s \in R\}, \quad (2)$$

where the polynomials $f_1, \ldots, f_s$ are called the generators (or basis) of the ideal.

For a binary variable x_i, we have $x_i^2 = x_i$. Therefore, the polynomial $x_i^2 - x_i$ vanishes over $\mathbb{F}_2$, and we call it a vanishing polynomial. While manipulating Boolean polynomials it is essential to ensure this indempotency by reducing the polynomials modulo (the ideal of) all vanishing polynomials $\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. Therefore, we essentially operate on the quotient ring of $\mathbb{F}_2[x_1, \ldots, x_n]$ modulo the ideal of vanishing polynomials, i.e. over $\mathbb{F}_2[x_1, \ldots, x_n] / \langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. Then Boolean polynomials are exactly the canonical representatives of residue classes in the aforementioned quotient ring. A significant benefit of using a Boolean data-structure such as ZBDDs is that the reduction $x_i^2 = x_i$ is performed implicitly by the data-structure. For this reason, in the sequel we *omit explicit mention* of reduction modulo the vanishing polynomials for manipulation of Boolean polynomials, and it should be assumed that the reduction $x_i^2 = x_i$ is always performed.

Gröbner basis of ideals: An ideal J may have many different generators, i.e. it is possible to have $J = \langle f_1, \ldots, f_s \rangle = \langle h_1, \ldots, h_r \rangle = \cdots = \langle g_1, \ldots, g_t \rangle$. A Gröbner basis G of ideal J is one such set of polynomials $G = GB(J) = \{g_1, \ldots, g_t\}$ with many important properties that allow to solve many polynomial decision questions.

Definition III.2 (Gröbner basis [7])**.** *For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \ldots, g_t\}$ contained in an ideal J, is called a Gröbner basis of J iff $\forall f \in J$, $f \neq 0$, there exists $g_i \in \{g_1, \ldots, g_t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow \ \forall f \in J : f \neq 0, \ \exists g_i \in G : lm(g_i) \mid lm(f)$.*

Gröbner basis G of an ideal $J = \langle f_1, \ldots, f_s \rangle$ is computed using the Buchberger's algorithm [8], reproduced in Alg. 2.

Algorithm 2 Buchberger's Algorithm

Inputs: $F = \{f_1, \ldots, f_s\}$
Outputs: $G = \{g_1, \ldots, g_t\}$
$G := F$;
repeat
  $G' := G$
  for each pair $\{f_i, f_j\}, i \neq j$ in G' **do**
    $Spoly(f_i, f_j) \xrightarrow{G'}_+ h$
    if $h \neq 0$ **then**
      $G := G \cup \{h\}$
until $G = G'$

The algorithm initializes the set G with the given generators of J *i.e.* $\{f_1, \ldots, f_s\}$. Then it takes pairs of polynomials (f_i, f_j) from the basis and computes their S-polynomial $Spoly(f_i, f_j)$:

$$Spoly(f_i, f_j) = \frac{L}{lt(f_i)} \cdot f_i - \frac{L}{lt(f_j)} \cdot f_j \quad (3)$$

where $L = LCM(f_i, f_j)$. The $Spoly(f_i, f_j)$ is then reduced *w.r.t.* the polynomials in G to obtain remainder h. If h is non-zero, it is added to G. The process is repeated for all unique polynomial pairs, including those generated by the newly added elements h. The algorithm terminates when there are no new non-zero h generated from the set G. $Spoly(f_i, f_j) \xrightarrow{G}_+ h$ reductions cancel the leading terms of polynomials $\{f_i, f_j\}$, and generate polynomials h with new leading terms, providing additional information regarding the ideal.

An important property of a Gröbner basis G is that reduction of a polynomial f modulo G is essentially unique.

Theorem III.1 (Gröbner Basis Reduction, Thm. 1.9.1 in [7])**.** *Let $G = \{g_1, \ldots, g_t\}$ be a Gröbner basis of ideal J, and let f be another polynomial. Then the remainder r obtained by reduction of f modulo G, denoted $f \xrightarrow{G}_+ r$, is called the* **Gröbner basis reduction (GBR)** *of f. The remainder r so obtained by GBR of f is a canonical expression modulo G.*

In other words, for any polynomial f, if $f \xrightarrow{G}_+ r_1$ and $f \xrightarrow{G}_+ r_2$, then $r_1 = r_2 = r$. The remainder r is sometimes also called the *normal form* of f w.r.t. G. The canonicity of r (modulo G) can be exploited for equivalence checking of digital circuits.

Proposition III.1. *Given a circuit C, we can represent all the gates using (Boolean) polynomials $F = \{f_1, \ldots, f_s\}$ in $\mathbb{F}_2[x_1, \ldots, x_n]$ by means of Eqn. (1), and generate ideal $J = \langle F \rangle$. Let z_i, $i = 0, \ldots, k-1, (z_i \in \{x_1, \ldots, x_n\})$ denote one-bit of the k-bit primary output variables of the circuit. Compute a Gröbner basis $G = GB(J) = \{g_1, \ldots, g_t\}$ for the polynomials of the circuit, and perform the GBR $z_i \xrightarrow{G}_+ r_i$ for all $0 \leq i < k$. Then all r_i's are a canonical representation and can be used for formal verification/equivalence checking.*

To derive the canonical representation r_i, it is required to compute a Gröbner basis G of the ideal generated by the polynomials of the circuit. Buchberger's algorithm for computation of a Gröbner basis exhibits high complexity. In general, the worst-case Gröbner basis complexity is doubly-exponential in the input data [27]. However, over ideals that

have a finite number of solutions, the complexity is single exponentially bounded. Particularly over finite fields $\mathbb{F}_q$ of q elements ($q = 2$ in our case), the complexity is bounded by $q^{(O(n))}$, where n is the number of variables [28]. This complexity still needs to be overcome. The work of [5] showed that for formal verification of combinational circuits, the expensive GB computation can be avoided altogether, due to the following results.

Lemma III.1 (Product Criterion [29])**.** *For two polynomials f_i, f_j in any polynomial ring R, if the equality $lm(f_i)\cdot lm(f_j) = LCM(lm(f_i), lm(f_j))$ holds, i.e. if $lm(f_i)$ and $lm(f_j)$ are relatively prime, then $Spoly(f_i, f_j) \xrightarrow{G}_+ 0$.*

Using this criterion we can say that when the leading terms of *all polynomials in the basis* $F = \{f_1, \ldots, f_s\}$ are relatively prime, then all $Spoly(f_i, f_j) \xrightarrow{G}_+ 0$. As no new polynomials are generated in Buchberger's algorithm, F is already a Gröbner basis ($F = GB(J)$). For a combinational circuit C, a specialized term order $>$ can always be derived by analyzing the circuit topology which ensures such a property [3] [5]:

Proposition III.2. *(From [5]) Let C be an arbitrary combinational circuit. Let $\{x_1, \ldots, x_n\}$ 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 the set of all polynomials $\{f_1, \ldots, f_s\}$ forms a Gröbner basis G, as $lt(f_i) = x_i$ and $lt(f_j) = x_j$ for $i \neq j$ are relatively prime. This term order $>$ is called the* **Reverse Topological Term Order (RTTO)**.

Imposition of RTTO on the polynomials of the circuit has the effect of making every gate output variable x_i a leading term of f_i. Since every gate output is unique, $lm(f_i) = x_i, lm(f_j) = x_j$ become relatively prime. As a result, the set F is already a GB ($G = F$), the explosive GB computation is avoided, and verification is performed solely by the canonical GB-reduction: $z_i \xrightarrow{G}_+ r_i$. Note that as $f_i = x_i + \text{tail}(f_i)$, RTTO ensures that every variable x_j that appears in $\text{tail}(f_i)$ satisfies $x_i > x_j$. Moreover, the remainder r_i comprises only primary inputs of the circuit. These properties will be exploited in our algorithms.

Example III.1. Demonstration of the approach: *Consider the circuit given in Fig. 1. Impose RTTO on the circuit. The primary outputs z_0, z_1 are both at level-0, variables r_0, c_0, c_3 are at level-1, c_1, c_2 are at level-2, and the primary inputs a_0, a_1, b_0, b_1 are at level-3. Order the variables $\{z_0 > z_1\} > \{r_0 > c_0 > c_3\} > \{c_1 > c_2\} > \{a_0 > a_1 > b_0 > b_1\}$. Using this variable order, we impose a lex term order on the monomials. Then all the polynomials extracted from the circuit have relatively prime leading terms, as shown in Fig. 2, and $G = F = \{f_1, \ldots, f_7\}$ forms a GB.*

Then the GBRs $z_1 \xrightarrow{G}_+ a_0\cdot b_0 + a_1\cdot b_1$ and $z_0 \xrightarrow{G}_+ a_0\cdot b_1 + a_1\cdot b_0 + a_1\cdot b_1$ are canonical expressions (Boolean polynomials) of the output bits and can be used for equivalence checking.

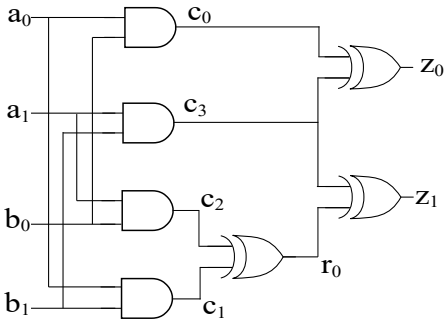

Fig. 1: A 2-bit modulo Multiplier circuit.

$$f_1 : c_0 + a_0\cdot b_0,\ lm = c_0; \quad f_2 : c_1 + a_0\cdot b_1,\ lm = c_1$$
$$f_3 : c_2 + a_1\cdot b_0,\ lm = c_2; \quad f_4 : c_3 + a_1\cdot b_1,\ lm = c_3$$
$$f_5 : r_0 + c_1 + c_2,\ lm = r_0; \quad f_6 : z_0 + c_0 + c_3,\ lm = z_0$$
$$f_7 : z_1 + r_0 + c_3,\ lm = z_1$$

Fig. 2: Polynomials of the circuit under RTTO constitute a GB.

We will now show how to efficiently implement this GBR $z_i \xrightarrow{G}_+ r_i$ on circuits by exploiting the implicit set representation ZBDDs under the imposition of RTTO.

IV. Unate Cube Sets & Boolean Polynomials

A Boolean *variable* represents a dimension of the Boolean space $\mathbb{B}^n$, a *literal* is an instance of a variable x_i or its complement $\neg x_i$. A *cube* is a product of literals which denotes a point or a set of points in the Boolean space. A *cube set* consists of a number of cubes, each of which is a combination of literals. *Unate cube sets* allow the use of only positive literals, not negative/complemented literals.

When cube sets are used to represent Boolean functions, they are usually *binate* cube sets containing negative literals. In binate cube sets, literals x_i and $\neg x_i$ represent $x_i = 1$ and $x_i = 0$, respectively; while the absence of a literal implies a *don't care.* In unate cube sets, literal x_i implies $x_i = 1$ whereas its absence implies $x_i = 0$. For example, the cube set $\{a, bc\}$ corresponds to points $(abc) : \{111, 110, 101, 100, 011\}$ in the binate cube set representation, whereas it represents $(abc) : \{100, 011\}$ in the unate cube set representation.

Each monomial of a Boolean polynomial can be viewed as a unate cube – a product of positive literals – and a Boolean polynomial as a unate cube set. Then the GBR $z_i \xrightarrow{G}_+ r_i$ can be interpreted as operations over unate cube sets, as shown below. Let us (re)consider the one-step division for Boolean polynomials: $f \xrightarrow{g} r$. This division can be interpreted as:

$$f \xrightarrow{g} r = f - \frac{lt(f)}{lt(g)}\cdot g \tag{4}$$
$$= f - \frac{lm(f)}{lm(g)}\cdot g; \quad (\text{coef. } 0{,}1;\ lt(f) = lm(f)) \tag{5}$$
$$= f + \frac{lm(f)}{lm(g)}\cdot g; \quad (-1 = +1 \pmod 2) \tag{6}$$
$$= f \oplus \frac{lm(f)}{lm(g)}\cdot g; \quad (as\ + \pmod 2 = \oplus) \tag{7}$$

Notice that $\frac{lm(f)}{lm(g)}$ is a unate product of literals. i.e. a unate cube. The $\oplus$ operation cancels common cubes from f and $\frac{lm(f)}{lm(g)} \cdot g$. The $\cdot$ operation models the modulo 2 product, where the partial products are added using the $\oplus$ operation. Therefore, we will implement the reduce operation $f \xrightarrow{g} r$ for Boolean polynomials as $r = f \oplus \frac{lm(f)}{lm(g)} \cdot g$ using an implicit representation particularly suited for such unate cube operations, i.e. ZBDDs.

A. Zero Suppressed Binary Decision Diagrams (ZBDDs) for Boolean Polynomials

Binary decision diagrams (BDDs) [1] and their variants have been used as implicit representations for solving many Boolean and pseudo-Boolean optimization problems. Zero-suppressed BDDs [16] are another variant of BDDs that were designed to efficiently manipulate "sets of combinations". A ZBDD is obtained from an ordered BDD by: i) eliminating those vertices whose 1-edge (then-edge) is incident on the 0-terminal; and ii) merging isomorphic subgraphs. Subject to the given variable order, a ZBDD represents a Boolean function canonically. For a detailed description of ZBDDs and their capabilities for solving logic optimization and sparse combinatorial problems, the reader is referred to [16] and [30].

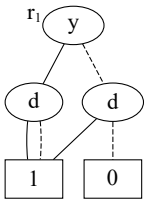

Fig. 3: ZBDD for the polynomial $r_1 = yd + y + d$.

In [30], *Minato* demonstrated that ZBDDs are an efficient data-structure for implicit manipulation (algebra) of unate cube sets. Fig. 3 depicts a ZBDD for the unate cube set $\{yd, y, d\}$ with the variable order $y > d$. The paths beginning from the root node y and terminating in the 1-terminal node correspond to the cubes of the set. A variable is present in a cube if its 1-edge lies on the path; otherwise it is absent from the cube if its 0-edge lies on the path. Consequently, the ZBDD of Fig. 3 can be construed to represent the Boolean polynomial $r_1 = yd + y + d$. *Minato* has shown [16][30] how the set union, intersection and difference operations can be implemented recursively in the ZBDD graphs, and they have been implemented using the *ite-operator* in decision diagrams such as the CUDD [18] package. We extend these operations to accommodate the sum and product operations $\pmod 2$, i.e. polynomial algebra in $\mathbb{F}_2[x_1, \ldots, x_n]$, by manipulating sets of combinations using ZBDDs.

Based on the above discussion, we will: i) model GBR as the algebra of unate cube sets; ii) use ZBDDs as the implicit data-structure for this GBR; and iii) devise efficient implementation of the GBR by exploiting the special structure imposed by RTTO on the ZBDD graph.

V. Theory and Algorithms

Let us first consider a scenario where the GBR $z \xrightarrow{G}_+ r$ on circuits under RTTO can result in a size explosion (expression swell) problem in an explicit representation. We also show on the other hand that an implicit set representation (ZBDD) overcomes this size explosion.

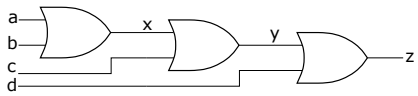

Fig. 4: A chain of OR gates.

Consider the circuit shown in Fig. 4 consisting of a chain of OR gates. Impose RTTO: i.e. *lex* term order with variable ordering as, $z > y > x > d > c > b > a$. The Boolean polynomials for the circuit are:

$$f_1 = z + yd + y + d; \tag{8}$$

$$f_2 = y + xc + x + c; \tag{9}$$

$$f_3 = x + ba + b + a; \tag{10}$$

Under RTTO, the set $G = \{f_1, f_2, f_3\}$ forms a GB. To derive a canonical representation of the function, we have to reduce the output $z \xrightarrow{G}_+ r$. A classical symbolic algebra reduction using an explicit representation is carried out as follows:

1) $z \xrightarrow{f_1} yd + y + d$
2) $yd + y + d \xrightarrow{f_2} y + xdc + xd + dc + d \xrightarrow{f_2} xdc + xd + xc + x + dc + d + c$
3) $xdc + xd + xc + x + dc + d + c \xrightarrow{f_3} xd + xc + x + dcba + dcb + dca + dc + d + c \xrightarrow{f_3} xc + x + dcba + dcb + dca + dc + dba + db + da + d + c \xrightarrow{f_3} x + dcba + dcb + dca + dc + dba + db + da + d + cba + cb + ca + c \xrightarrow{f_3} dcba + dcb + dca + dc + dba + db + da + d + cba + cb + ca + c + ba + b + a = r$

In the first step, z is reduced by f_1 just *once* as that's the only term. In the second step, the result of step one is reduced *twice* by f_2 as the result has two terms containing variable y. Similarly, *four* reductions by f_3 are required to reduce the result of step two into an expression containing only primary inputs (which cannot be reduced further).

Observations: i) Notice that the size of the final remainder corresponds to that of the worst case of a Boolean polynomial: i.e. r contains $2^n - 1$ $(= 15)$ monomial terms for n $(= 4)$ variables. ii) Classical division algorithms reduce the polynomials 1-step at a time, where only one monomial is canceled in each step. iii) The number of 1-step reductions can increase exponentially as GBR progresses across the circuit.

It is clear that any data-structure that *explicitly* represents each monomial will encounter space and time explosion: this includes the dense-distributive representation of SINGULAR computer algebra tool [17], or the ones used by [11], [13]. The F_4-style polynomial reduction of [5], [6] simulates division on a matrix M representing the problem. However, each column of M corresponds to a monomial generated in the division process, therefore [5], [6] also encounter this size explosion.

The use of ZBDDs can help overcome this explosion. Fig. 5 shows the same reduction of z by f_1, f_2, f_3 using ZBDDs

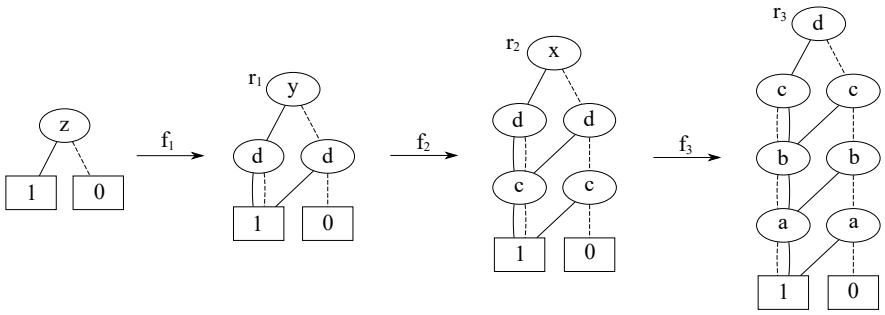

Fig. 5: Reduction of output of the circuit in Fig. 4 by f_1, f_2, f_3.

(exact procedure discussed later). The size of the ZBDDs after complete reduction by f_1, f_2, f_3 increases linearly in the number of nodes. Subsequently, the final remainder has $2 \cdot n - 1$ $(= 7)$ nodes (excluding the terminal 1 and 0 nodes) for n $(= 4)$ variables. Notice that while this controls space explosion, the number of paths (monomials) in the ZBDD is still exponential in the number of variables. A classical division algorithm that cancels only one monomial at a time may still require an exponential number of iterations. We show how to improve upon such a situation.

Problem Setup: Given a circuit C, denote all its *nets* with variables $x_1, \ldots, x_n$. Let the total number of gates in the circuit be s; represent each gate of the circuit with a polynomial f_i in its immediate inputs. Then $F = \{f_1, \ldots, f_s\}$ describes the circuit netlist as a set of Boolean polynomials in $\mathbb{F}_2[x_1, \ldots, x_n]$.

Objective: Impose RTTO on the polynomial ring, so that the set $F = \{f_1, \ldots, f_s\}$ constitutes a Gröbner basis G. For all primary outputs $z_i \in \{x_1, \ldots, x_n\}$, compute $z_i \xrightarrow{G}_+ r_i$ where r_i is the canonical representation of z_i modulo G, and use it for equivalence checking. The representation of the polynomials F of C, and the computation $z_i \xrightarrow{G}_+ r_i$ is to be performed using ZBDDs.

A. ZBDD representation for the polynomials of the circuit

First, a reverse topological traversal of the circuit is performed to derive the variable order $x_1 > x_2 > \cdots > x_n$ as given in Prop. III.2. The same variable order is imposed on the ZBDDs, i.e. x_1 is the variable at the top level in the ZBDDs. A ZBDD is created for each net variable x_i. Using Eqn. (1) the gates of the circuit are modeled as the set of Boolean polynomials $F = \{f_1, \ldots, f_s\}$. ZBDDs for these polynomials are constructed using the $+$ and $\cdot$ binary operations for modulo 2 sum and product of two ZBDDs. Conceptually, the modulo 2 sum $(\oplus)$ operation for two ZBDDs f, g can be implemented as $f + g = f_{cs} \cup g_{cs} - f_{cs} \cap g_{cs}$, where f_{cs} and g_{cs} represent the cube sets for the polynomials f and g respectively. For example, let $f = ab + c$ and $g = c + d$ with the corresponding cube sets $f_{cs} = \{ab, c\}$ and $g_{cs} = \{c, d\}$, then $f_{cs} \cup g_{cs} = \{ab, c, d\}$ and $f_{cs} \cap g_{cs} = \{c\}$. The set difference $f_{cs} \cup g_{cs} - f_{cs} \cap g_{cs}$ is the set $\{ab, d\}$ and the corresponding Boolean polynomial is $ab + d$.

However, experience has shown that such an implementation with the union operation results in large size of intermediate ZBDDs. In order to avoid this intermediate size explosion, we have implemented the $f + g \pmod 2$ operation along similar lines as presented in [23]. The algorithm for this operation is shown in Algorithm 3.

Algorithm 3 Algorithm for performing $f + g \pmod 2$

```
1:  procedure mod_2_sum(f, g)
2:      if f = 0 then
3:          return g
4:      else if g = 0 then
5:          return f
6:      else if f = g then
7:          return 0
8:      else
9:          v1 = top_var(f); v2 = top_var(g);
10:         if index(v1) < index(v2) then
11:             return ite(v1, then(f), else(f) + g)
12:         else if index(v1) > index(v2) then
13:             return ite(v2, then(g), else(g) + f)
14:         else
15:             return ite(v1, then(f) + then(g), else(f) + else(g))
```

Example V.1. *To demonstrate $f + g, f = ab + c, g = c + d$, let the variable ordering be $a > b > c > d$, i.e. index values for these variables are $0, 1, 2, 3$, respectively. The condition $index(v_1 = a) < index(v_2 = c)$ is true. The ite operation places $then(f) = b$ on the solid edge of the root of the new ZBDD and performs $else(f) + g \pmod 2$, as shown in the Fig. 6. During this recursive call, the last condition is true (as $index(v_1 = c) = index(v_2 = c) = 2$). This time the ite operation performs two recursive calls $then(f) + then(g)$ and $else(f) + else(g)$, and so on, to finally construct the ZBDD for the Boolean polynomial $ab + d$.*

A similar recursive algorithm is also implemented for $f \cdot g \pmod 2$ operation where the intermediate partial product terms are added $\pmod 2$ using the Algorithm 3. *E.g.,* $f = ab, g = a + b, f \cdot g = ab + ab = 0$.

B. GBR $z_i \xrightarrow{G}_+ r_i$ under RTTO on ZBDDs

Once the ZBDDs for the circuit have been built and stored in G, we need to perform the canonical Gröbner basis reduction $z_i \xrightarrow{G}_+ r_i$ for each output bit z_i. The polynomial r_i will be a canonical representation of z_i in terms of primary inputs. Reduction $f \xrightarrow{g} r$ requires one to obtain the leading monomials $lm(f), lm(g)$ of f, g (Eqn. 7).

1) Finding the leading monomials under RTTO on ZBDDs: Recall that RTTO imposes a *lex* term order on the polynomials

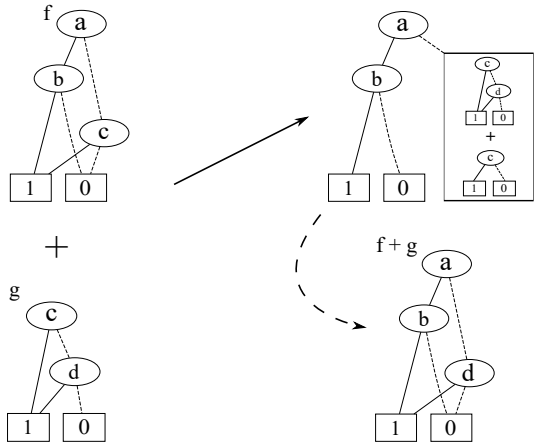

Fig. 6: $f+g \pmod 2$ using ZBDDs

using the variable order $x_1 > \cdots > x_n$. Moreover, the same variable order $x_1 > \cdots > x_n$ is imposed on the ZBDDs. Traversing the then-edges from the root node of a ZBDD to the terminal **1** delivers the leading monomial of that polynomial under RTTO.

2) Classical reduction with ZBDDs: Cancel one monomial in every step: The algorithm for conventional reduction procedure using ZBDDs is shown in Algorithm 4. The input parameters are the ZBDD of the output bit z_i of the circuit and *poly_list* – a list containing the ZBDDs for the set of polynomials $F = \{f_1, \ldots, f_s\}$ corresponding to the gates of the circuit. The algorithm is based on the same principles as the classical division procedure (Algorithm 1).

Algorithm 4 Reduction: Cancel 1 monomial every iteration

```
1:  procedure single_mon_red(z_i, poly_list)
2:      for each g ∈ poly_list do
3:          lead_g = leading_term(g)
4:          lead_z_i = leading_term(z_i)
5:          quotient = ZBDD_Divide(lead_z_i, lead_g)
6:          while quotient ≠ zero do
7:              prod = quotient · g
8:              z_i = z_i + prod
9:              lead_z_i = leading_term(z_i)
10:             quotient = ZBDD_Divide(lead_z_i, lead_g)
11:     return z_i
```

The procedure $leading\_term(g)$ returns the leading term of the ZBDD representation of polynomial g. If g divides f, then the procedure $ZBDD\_Divide(f,g)$ (performs cube division) returns the quotient of the division, else it returns zero. Line 8 iteratively computes $z_i = z_i + \frac{lt(z_i)}{lt(g)} \cdot g$. The polynomial z_i is completely reduced *w.r.t.* the polynomial g in the while loop.

3) Improved Reduction: Cancel multiple monomials in 1 step: Next, we will show how z_i can be reduced by a polynomial g in one step. In the example of Figs. 4 and 5, the primary output z is reduced by f_1 to get r_1. The next step is to reduce r_1 by f_2 to get r_2. To demonstrate our approach we will show how the reduction of r_1 by f_2 can be achieved in one step. There are two monomials in r_1 that contain y, namely yd, y. Both can be canceled by $lt(f_2) = y$ in one step, eliminating the need of the while loop in Algorithm 4.

The polynomial $r_1 = yd + y + d$ can be written as $y \cdot (d+1) + d$. If we perform 1-step reduction of r_1 by f_2 we get the *quotient* $d+1$. This quotient is visible as the polynomial represented by the *then*-node of r_1 (Fig. 7). So the reduction can be performed by multiplying $d+1$ with f_2 and subtracting (adding) this product to $r_1 \pmod 2$.

$$r_1 \xrightarrow{f_2}_{+} r_2 \tag{11}$$
$$= (yd+y+d) + (d+1)\cdot(y+xc+x+c) \pmod 2 \tag{12}$$
$$= 2\cdot(yd+y) + d + (d+1)\cdot(xc+x+c) \pmod 2 \tag{13}$$
$$= \underbrace{d}_{else(r_1)} + \underbrace{(d+1)}_{then(r_1)} \cdot \underbrace{(xc+x+c)}_{else(f_2)} \pmod 2 \tag{14}$$

As shown in Fig. 7, $else(r_1) = d, then(r_1) = d+1$ and $else(f_2) = xc + x + c$. Moreover, $2 \cdot (yd + y) = 0 \pmod 2$. Therefore, in order to reduce number of operations incurred in the division process, we directly use the last step as a formula for reduction:

$$r_1 \xrightarrow{f_2}_{+} r_2 = else(r_1) + then(r_1) \cdot else(f_2) \tag{15}$$

So the reduction process effectively involves just two operations, a modulo 2 sum and a product. *This has the effect of canceling all the terms in r_1 that can be canceled by $lt(f_2)$ in one-go, implicitly canceling multiple monomials in one step.* This is made possible only due to the properties that RTTO imposes on the structure of ZBDDs.

The algorithm for implicit cancellation of multiple monomials in reduction is shown in Algorithm 5, where the notations, z_i and *poly_list*, are the same as in Algorithm 4. Another significant improvement that can be made in the algorithm is w.r.t. the order in which $poly\_list = \{f_1, \ldots, f_s\}$ is populated, and how the search for the divisors $g \in poly\_list$ is greatly simplified.

Due to RTTO, each polynomial $f_i \in F$ is of the form $f_i = x_i + \text{tail}(f_i)$, where each variable $x_j \in \text{tail}(f_i)$ satisfies $x_j < x_i$ (Prop. III.2). Due to this order, the ZBDD representation of the polynomials is of the type $f_1 = x_1 + else(f_1), \ldots, f_s = x_s + else(f_s)$ with variable order $x_1 > \cdots > x_s > \cdots > x_n$. Note that the variables $\{x_{s+1}, \ldots, x_n\}$ are primary inputs, and they are not the output of any logic gate. *We ensure that primary inputs appear last in RTTO.* Then we store elements in *poly_list* also according to the order $f_1 > f_2 > \cdots > f_s$; i.e. $poly\_list[1] = f_1, \ldots, poly\_list[s] = f_s$. Imposition of RTTO on the ZBDDs and on *poly_list* simplifies the check for divisors: *if indices of top-most nodes of ZBDDs of z_i and f_i are equal, then $lm(f_i)$ divides $lt(z_i)$.* The equality of indices indicate that the top variables are the same (as indices are unique). In addition, as f_i is guaranteed to have that variable only in the leading term (*i.e.* leading term is a singleton with only one variable), f_i divides z_i. This allows to replace the cube division (Line 5, Algorithm 4) by an equality check of top indices of ZBDDs.

Example V.2. *Consider the step 2) of division corresponding to Fig. 4, where the polynomial $r_1 = yd + y + d$ needs to be reduced by f_2. RTTO need not differentiate between the relative ordering of the nets y, d for Prop. III.2 to be valid; as both y, d are at the same (reverse) topological level. However, we ensure that the primary inputs always appear*

Algorithm 5 Reduction under RTTO: Cancel multiple monomials

```
1: procedure multi_mon_red(z_i, poly_list)
2:     for each g ∈ poly_list do
3:         if index(g) == index(z_i) then
4:             z_i = else(z_i) + then(z_i) · else(g)
5:     return z_i
```

last in the order. The ZBDDs for r_1 and f_2 are shown in Fig. 7, where RTTO is lex with $y > x > d > c > b > a$. As $index(top\_var(f_2)) = index(top\_var(r_1))$, $lm(f_2) \mid lt(r_1)$. If variable d were placed earlier in the order than y, then the top variables of f_2, r_1 would have been different and we would have had to extract $lm(r_1), lm(f_2)$ and performed cube division $\frac{lm(r_1)}{lm(f_2)}$ to check if $lm(f_2) \mid lm(r_1)$.

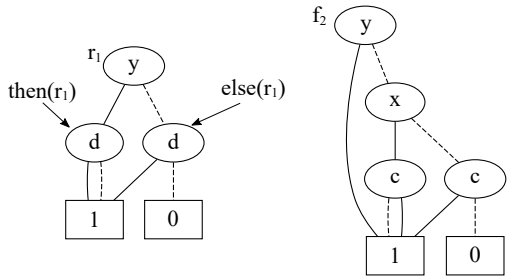

Fig. 7: ZBDDs for polynomial r_1 and f_2.

Therefore, unlike in Algorithm 4, where we need to obtain the leading monomials and compute the quotient $lead\_z_i/lead\_g$, Algorithm 5 only determines if $lead\_g$ can divide z_i at all (in this case the quotient is $then(z_i)$). This can be accomplished by just comparing the indices of top-most nodes of z_i and g. This algorithm significantly reduces the number of iterations, which now exactly equals the size of $poly\_list$. For the example of Fig. 4, the number of iterations is 3 using Algorithm 5, whereas 7 iterations are required using Algorithm 4.

To contrast our approach against PolyBori [23]: PolyBori is a general purpose symbolic computing engine for Boolean polynomials, whereas our approach (Algorithm 5) is specific only to RTTO; it would give an incorrect result of reduction for non-RTTO based term orders. However, our approach exploits the fact that due to RTTO, the subexpressions required in the division process are readily available as the *then-* and *else-*children of the top nodes of the ZBDD (Eqn. (14)). Moreover, an implementation using PolyBori may also generate intermediate monomials that eventually add up to 0 $\pmod 2$ (*e.g.* as in Eqn. (13)). Whereas our approach reduces the size of intermediate computations, and incurs less computation time, by not generating the duplicate monomials.

We have implemented the above GBR procedures using the CUDD package [18]. The circuit under verification is analyzed, RTTO based variable order is imposed on the ZBDDs, and the Boolean polynomials of the circuit are represented as unate cube sets on ZBDDs. The polynomials corresponding to the gates of circuit, $G = \{f_1, \ldots, f_s\}$, are inserted in $poly\_list$ according to the variable order $x_1 > \cdots > x_i > \cdots > x_n$, where $f_i = x_i + else(f_i)$. To perform GBR $z_i \xrightarrow{G}_+ r_i$, Algorithm 5 is invoked and r_i used for equivalence checking.

VI. Experimental Results

This section presents the results of using our implementation (Algorithm 5) for formal verification and equivalence checking of circuits used in cryptography. We compare our results against: i) F4-style reduction [6] which models the reduction as Gaussian elimination on a coefficient matrix; ii) Parallelized approach for performing reductions on Galois field multipliers [15]; and iii) PolyBori's [23] reduction procedure with ZBDDs as underlying data structure. For all tools and experiments, RTTO $>$ is used for constraint representation. The experiments are performed on a 3.5GHz Intel Core™ i7-4770K Quad-Core CPU with 32 GB of RAM. Experiments are conducted for verification of finite field multipliers and polynomial computation modules used as cryptography primitives. The data-path sizes k are selected according to cryptography standards recommended by U.S. National Institute of Standards and Technology (NIST).

A. Mastrovito Multipliers

Modular multiplication is an important computation used in cryptography. A Mastrovito multiplier architecture can be employed for performing this computation over the finite field of 2^k elements, i.e. $\mathbb{F}_{2^k}$. Mastrovito multipliers compute $Z = A \times B \pmod{P(x)}$ where $P(x)$ is a given primitive polynomial for the datapath size k. The product $A \times B$ is computed using an array multiplier architecture, and then the result is reduced modulo $P(x)$. The following example demonstrates the Mastrovito multiplier computation [5].

Example VI.1. *Consider the finite field $\mathbb{F}_{2^4}$ where the operand size is 4. Let the inputs be: $A = a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3$ and $B = b_0 + b_1 \cdot \alpha + b_2 \cdot \alpha^2 + b_3 \cdot \alpha^3$, and the primitive polynomial be $P(x) = x^4 + x^3 + 1$ with α as its root so that $P(\alpha) = 0$. The constituent bits of A and B are $\{a_0, \ldots, a_3\}$ and $\{b_0, \ldots, b_3\}$ respectively, which are the primary inputs of the circuit. First, we perform the multiplication $A \cdot B$ as:*

			a_3	a_2	a_1	a_0
$\times$			b_3	b_2	b_1	b_0
			$a_3 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_0 \cdot b_0$
		$a_3 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_0 \cdot b_1$	
	$a_3 \cdot b_2$	$a_2 \cdot b_2$	$a_1 \cdot b_2$	$a_0 \cdot b_2$		
$a_3 \cdot b_3$	$a_2 \cdot b_3$	$a_1 \cdot b_3$	$a_0 \cdot b_3$			
s_6	s_5	s_4	s_3	s_2	s_1	s_0

The result $S = s_0 + s_1 \cdot \alpha + s_2 \cdot \alpha^2 + s_3 \cdot \alpha^3 + s_4 \cdot \alpha^4 + s_5 \cdot \alpha^5 + s_6 \cdot \alpha^6$, where, $s_0 = a_0 \cdot b_0$, $s_1 = a_0 \cdot b_1 + a_1 \cdot b_0$, $s_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0$, and so on. Here the multiply "$\cdot$" and add "$+$" operations are performed modulo 2, and hence implemented in a circuit using AND and XOR gates. As the coefficients are always reduced $\pmod 2$ in $\mathbb{F}_{2^k}$, there are no carry-chains in the design. Next, the result S is reduced modulo the primitive polynomial $P(x) = x^4 + x^3 + 1$, as:

s_3	s_2	s_1	s_0	
s_4	0	0	s_4	$s_4 \cdot \alpha^4 \pmod{P(\alpha)} = s_4 \cdot (\alpha^3 + 1)$
s_5	0	s_5	s_5	$s_5 \cdot \alpha^5 \pmod{P(\alpha)} = s_5 \cdot (\alpha^3 + \alpha + 1)$
s_6	s_6	s_6	s_6	$s_6 \cdot \alpha^6 \pmod{P(\alpha)} = s_6 \cdot (\alpha^3 + \alpha^2 + \alpha + 1)$
z_3	z_2	z_1	z_0	

The primary output of the circuit is: $Z = \{z_0, \ldots, z_3\}$, *represented as* $Z = z_0 + z_1\alpha + z_2\alpha^2 + z_3\alpha^3$; *where* $z_0 = s_0 + s_4 + s_5 + s_6$; $z_1 = s_1 + s_5 + s_6$; $z_2 = s_2 + s_6$; $z_3 = s_3 + s_4 + s_5 + s_6$.

Table I provides the results for the reductions $z_i \xrightarrow{G}_+ r_i$ for Mastrovito multipliers for each output bit $z_i, 0 \leq i \leq k-1$. The benchmarks are taken from [5] and optimized using ABC [31] with the commands *resyn*2 and *dch* as mentioned in [15]. Algorithm 5 reduces each output bit independently of other bits. Therefore, we have presented the results obtained by running our reduction algorithm both parallelly and sequentially for each output bit. Similarly, the results for implementation in PolyBori are also presented for both cases. The implementation presented in [15] is already parallelized. We parallelized the PolyBori and our implementation by creating individual processes for each output bit z_i that has its own set of variables and gate polynomials, *poly_list* (Algorithm 5). The maximum number of parallel processes is decided upon the memory usage of each process (*i.e.* reducing one bit) for our implementation and the total available memory. The larger benchmarks are run with fewer parallel processes as they consume more memory.

In the table, the column #T represents the number of parallel processes. (S) and (P) refer to the cases when the experiments are run sequentially and parallelly for the output bits z_i, respectively.

TABLE I: Mastrovito Multipliers (Time in seconds); k = Datapath Size, #Gates = No. of gates, #T = No. of threads, Time-Out = 30 hrs, (P): Parallel Execution, (S): Sequential Execution, K = 10^3, M = 10^6, PB: PolyBori, ZR: Algorithm 5

k	#Gates	F4 [6]	#T	[15](P)	PB		ZR	
					(P)	(S)	(P)	(S)
64	11.5K	1.3	20	3.70	3.60	2.21	0.73	**0.27**
128	46K	9.89	20	27.54	23.99	16.76	5.08	**1.63**
163	73.5K	32.61	20	55.96	48.67	33.72	11.41	**3.11**
233	122K	86.30	20	127.61	112.96	77.23	21.77	**3.63**
283	193K	274.68	20	253.05	227.77	157.45	49.89	**11.41**
409	386K	2,528.5	10	716.80	659.64	426.92	163.52	**17.68**
571*	1.6M	TO	3	5,331	CR	CR	2,126.7	**566.4**

The 571-bit multiplier could not be synthesized and mapped with the given memory due to its large size. Therefore, we have provided results for a structured (but unoptimized) 571-bit multiplier benchmark. Our implementation outperforms the explicit approaches of [6] and [15] for Mastrovito multipliers and also PolyBori. For the 571-bit multiplier, the implementation of [6] does not finish for the given time period of 30 hours and the PolyBori implementation crashes (CR). The memory consumption for the Mastrovito multipliers when GBR is performed sequentially varies from 131.6 MB for 64-bit to ∼8.1 GB for 571-bit.

An interesting point to note in Table I is that our implementation takes less time when we are running it sequentially. There is a certain overhead involved when we declare variables and build ZBDDs for each gate of the circuit. In the case of Mastrovito multipliers benchmarks, this overhead is substantially greater than the reduction time for each output bit. Therefore, when we run these benchmarks parallely this overhead increases the overall run time.

Efficiency of our approach: The efficiency of our approach is attributed to the implicit data-structure and the number of iterations (*while* loop in Algorithm 4) that are saved while performing the GBR. Table II shows the number of explicit monomial cancellations that are *avoided in our approach* by exploiting the structure of the ZBDDs, when applied for the GBR $z_i \xrightarrow{G}_+ r_i$ for verification of Mastrovito multipliers.

TABLE II: Number of explicit monomial cancellations (MC) *saved* by using our approach; K = 10^3, M = 10^6

$k=$	64	128	163	233	283	409	571*
MC	22.8K	96.4K	156.2K	171.3K	404.6K	511.3K	1.63M

B. Montgomery Multipliers

Exponentiation operations are often required in cryptosystems. For such applications, Montgomery architectures [32] [33] [34] are considered more efficient than Mastrovito multipliers as they do not require explicit reduction modulo P after each step. Fig. 8 shows the structure of a Montgomery multiplier. Each MR block computes $A \cdot B \cdot R^{-1}$, where R is selected as a power of a base (α^k) and R^{-1} is the multiplicative inverse of R in $\mathbb{F}_{2^k}$. As this operation cannot compute $A \cdot B$ directly, we need to pre-compute $A \cdot R$ and $B \cdot R$ as shown in the Fig. 8. We denote the leftmost two blocks as Block A (upper) and B (lower), the middle block as Block C and the output block as Block D.

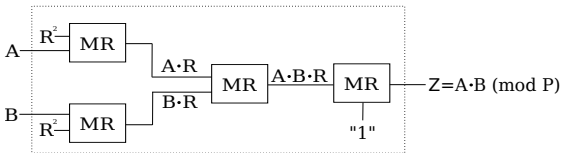

Fig. 8: Montgomery multiplication.

Table III provides the results for GBR on flattened (bit-blasted) and optimized Montgomery multipliers for the sequential and parallel executions of Algorithm 5. The memory consumption for sequential execution varies from 66.7 MB for 64-bit to ∼11.3 GB for 571-bit. Our approach outperforms conventional explicit approaches except for the case of 283 bit multiplier. Also, the reduction time for 283-bit circuit is more than 409-bit circuit. The reason is that the bit-width of the multipliers is not the only factor controlling the GBR time. The irreducible polynomial $P(x)$ also plays an important role in the designing of the multiplier and consequently in the reduction as explained in [15].

TABLE III: Montgomery Multipliers (Time in seconds); k = Datapath Size, #Gates = No. of gates, #T = No. of threads, Time-Out = 30 hrs, (P): Parallel Execution, (S): Sequential Execution, K = 10^3, M = 10^6, PB: PolyBori, ZR: Algorithm 5

k	#Gates	F4 [6]	#T	[15](P)	PB		ZR	
					(P)	(S)	(P)	(S)
64	9.5K	16.29	20	10.69	6.27	9.22	**3.75**	8.37
128	35K	621.90	20	36.19	28.93	34.59	**13.76**	24.73
163	56.5K	2,608.4	20	204.94	167.73	335.2	**141.68**	321.60
233	111K	385.92	20	132.51	119.77	99.36	42.16	**31.88**
283	165K	5,344	20	**704.13**	1,194.2	2,078	1,065.3	2,113
409	340K	7,104	10	697.91	737.23	722.1	303.91	**299.92**
571*	1.97M	TO	3	TO	CR	CR	**43,813**	99,042

Table IV presents the statistics for Montgomery multipliers where the hierarchy of Fig. 8 for the blocks A, B, C, and D is made available. The experiment first reduces the outputs of each individual block modulo the gates of that block, and then reduces the primary outputs modulo these four sets of remainders (ZBDDs), thus exploiting the hierarchy of these circuits. Table IV shows the time for reduction of each block and the time for reducing the primary outputs across the four blocks. The time for reducing the primary outputs across the hierarchical blocks in case of the F4 implementation is $<$1 second, and is not explicitly mentioned in the table. The row labeled *Total* presents the sum of the computation time of reduction across these levels, and the maximum of the time to reduce each MR block (as the reductions for the four blocks are independent of each other and are parallelized). These results again demonstrate the efficiency of our approach against explicit approaches.

TABLE IV: Montgomery Blocks (Time in seconds); k = Datapath Size, #Gates = No. of gates, Time-Out = 30 hrs, Red. = time for reduction, Coll. = time to reduce across the 4 levels. K = 10^3, M = 10^6, PB: PolyBori, ZR: Algorithm 5

k	#Gates	Block	F4 [6]	PB		ZR	
				Red.	Coll.	Red.	Coll.
163	33K	Block A	25	12	16	1	18
	33K	Block B	25	12		1	
	85K	Block C	73	18		7	
	32K	Block D	24	12		1	
	Total		73	34		**25**	
233	55K	Block A	142	32	5	0.14	4
	55K	Block B	141	33		0.14	
	163K	Block C	408	34		2.1	
	54K	Block D	140	32		0.13	
	Total		408	39		**6.1**	
283	82K	Block A	330	79	26	24	90
	82K	Block B	329	78		23	
	241K	Block C	883	173		118	
	81K	Block D	321	80		23	
	Total		883	**199**		208	
409	168K	Block A	1,322	177	28	0.57	29
	168K	Block B	1,335	175		0.57	
	502K	Block C	4,471	192		14	
	168K	Block D	1,338	176		0.56	
	Total		4,471	220		**43**	
571	330K	Block A	5,371	769	1,341	321	1,412
	330K	Block B	5,421	747		332	
	980K	Block C	37,804	3,605		3026	
	328K	Block D	5,539	751		338	
	Total		37,804	4,946		**4,438**	

Equivalence Checking: As a result of the GBR $z_i \xrightarrow{G}_+ r_i$, the function implemented by each output bit z_i of the circuit is represented as a reduced, canonical, Boolean polynomial in terms of the primary inputs, and by using a ZBDD. Thus the equivalence of such vastly different arithmetic circuit implementations (Mastrovito vs Montgomery) can be verified by testing for the equality (isomorphism) of the corresponding ZBDD graphs.

C. Point Addition over Elliptic Curves

Point addition is an important operation required for the task of encryption, decryption and authentication in Elliptic Curve Cryptography (ECC). Modern approaches represent the points in projective coordinate systems, *e.g.*, the L*ó*pez-Dahab (LD) projective coordinate [35], due to which the point addition operation can be implemented as polynomials in the field.

Example VI.2. *Consider point addition in López-Dahab (LD) projective coordinate. Given an elliptic curve:* $Y^2+XYZ = X^3Z+aX^2Z^2+bZ^4$ *over* $\mathbb{F}_{2^k}$*, where* X,Y,Z *are k-bit vectors that are elements in* $\mathbb{F}_{2^k}$ *and similarly,* a,b *are constants from the field. We represent point addition over the elliptic curve as* (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1)*. Then* X_3, Y_3, Z_3 *can be computed as follows:*

$$A = Y_2 \cdot Z_1^2 + Y_1 \qquad B = X_2 \cdot Z_1 + X_1$$
$$C = Z_1 \cdot B \qquad D = B^2 \cdot (C + aZ_1^2)$$
$$Z_3 = C^2 \qquad E = A \cdot C$$
$$X_3 = A^2 + D + E \qquad F = X_3 + X_2 \cdot Z_3$$
$$G = X_3 + Y_2 \cdot Z_3 \qquad Y_3 = E \cdot F + Z_3 \cdot G$$

Each of the polynomials in the above design are implemented as a (gate-level) logic block and are interconnected to obtain final outputs X_3, Y_3 and Z_3.

TABLE V: Point Addition Circuits (Time in seconds); k = Datapath Size, #Gates = No. of gates, Time-Out = 30 hrs, K = 10^3, M = 10^6, PB: PolyBori, ZR: Algorithm 5

k	#Gates	F4 [6]	PB	ZR
64	15.3K	1.78	3.32	**0.72**
128	64K	40.55	27.41	**6.03**
163	104K	130.24	57.57	**13.13**
233	139K	335.60	106.85	**19.62**
283	281K	1,787.96	273.53	**64.48**
409	423K	5,077.50	578.15	**115.20**
571	1.14M	48,162.29	CR	**725.95**

The word-level abstraction approach in [6] presents the results for extracting the above representation for each of $A, B, \ldots, X_3, Y_3, Z_3$ blocks. It first performs a bit-level reduction for every output of each block (GBR $z_i \xrightarrow{G}_+ r_i$), and then a bit-to-word substitution to derive an input-output word-level representation for the circuit. Table V shows the comparison of the time required for bit-level reduction of outputs d_i of the block $D = B^2 \cdot (C + aZ_1^2)$ as done in [6] against our implementation. (Bit-level reductions for other blocks take much less time than that for block D.) This result demonstrates that

our bit-level GBR implementation is in many cases orders of magnitude faster than the F4-style reduction of [6]. Therefore our approach can replace the F4-style bit-level GBR of [6] and improve the overall process of word-level abstraction of datapath designs.

D. Equivalence Checking of Sequential Galois Field Multipliers

The designs discussed so far are combinational implementations of polynomial computations of finite filed circuits. These designs use the standard basis representation $\{1, \alpha, \alpha^2, \ldots, \alpha^{k-1}\}$ to model a k-bit data-word Z in terms of its constituent bits as $Z = z_0 + z_1\alpha + z_2\alpha^2 \cdots + z_{k-1}\alpha^{k-1}$, with α being the primitive element for that field $\mathbb{F}_{2^k}$.

There exists sequential multipliers where k-bit inputs are loaded into k-bit registers, and the k-bit result is available after k clock-cycle execution of the machine. These multipliers use a *normal basis* $\{\beta, \beta^2, \beta^4, \ldots, \beta^{2^{k-1}}\}$ to represent a k-bit data-word S in terms of its constituent bits as $S = s_0\beta + s_1\beta^2 + s_2\beta^4 \cdots s_{k-1}\beta^{2^{k-1}}$, with β being the normal element. The relation between α and β can be used to represent the bits z_i and s_j in terms of each other.

We perform equivalence checking between two different architectures of *sequential multipliers with parallel output (SMPO)*, the Agnew-SMPO (AG-SMPO) by G.B. Agnew [36] and the RH-SMPO by Reyhani-Masoleh and Hasan [37]. The values in the registers for all intermediate states in these multipliers are different because they are based on two different mathematical principles. However, the final state of the output registers (i.e. after k clock cycles) is the same as it is the result of the multiplication. In order to perform equivalence checking, the circuits are unrolled over k time frames, and the GBR $s_i \xrightarrow{G}_+ r_i$ is performed to obtain a canonical r_i (G is the set of polynomials for the unrolled circuit under RTTO). The ZBDDs for respective r_i's (for AG-SMPO and RH-SMPO) are compared to perform equivalence check.

Tables VI and VII present the run-time of our implementation for performing these reductions on RH-SMPO and AG-SMPO architectures respectively, when compared with the approach presented in [6] and PolyBori. The results show that our implementation is about an order of magnitude faster than PolyBori and multiple orders of magnitude faster than the explicit approach of [6].

TABLE VI: RH-SMPO Multipliers (Time in seconds); k = Datapath Size, #Gates = No. of gates, Time-Out = 30 hrs, K = 10^3, PB: PolyBori, ZR: Algorithm 5

$k=$	65	81	89	131	173	233	281	410
#Gates	13.6K	21.4K	25.9K	55.9K	96.5K	177K	258K	546K
F4[6]	9.02	26.65	42.46	294.7	874.3	3,404	7,328	23,610
PB	3.65	6.07	7.42	28.22	47.16	116.63	199.32	637.69
ZR	**0.42**	**0.80**	**1.01**	**3.03**	**3.53**	**8.12**	**13.27**	**52.09**

TABLE VII: AG-SMPO Multipliers (Time in seconds); k = Datapath Size, #Gates = No. of gates, Time-Out = 30 hrs, K = 10^3, PB: PolyBori, ZR: Algorithm 5

$k=$	65	81	89	131	173	233	281	410
#Gates	12.5K	19.5K	23.6K	51.2K	89.4K	162K	236K	503K
F4[6]	8.34	20.46	33.2	221.4	754.1	2,655	5,569	21,938
PB	3.11	6.82	9.21	20.15	44.37	107.12	187.77	578.61
ZR	**0.44**	**0.77**	**0.91**	**2.51**	**3.39**	**7.8**	**12.63**	**43.78**

E. Limitations of our approach: Integer arithmetic circuits

The GBR approach presented in the previous section cannot be applied directly in the case of integer arithmetic circuits. The logical cones of output variables in integer arithmetic circuits have lot of logic sharing (common subexpressions) and generate a large number of non-linear terms. We evaluated our technique on integer arithmetic multiplier circuits, which showed an exponential increase in verification time *w.r.t.* circuit size. This is due to the reason that our approach reduces one output variable at a time and does not consider the logic sharing among the output variables. On the other hand, a word-level reduction approach can cancel the non-linear terms early in the reduction process and avoid intermediate blow-up in the number of monomials.

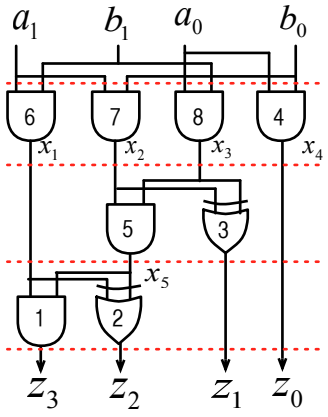

Fig. 9: Integer multiplier circuit

Using the ZBDD-based bit-level GBR for a 7x7 integer multiplier reveals that, when reducing the z_{13} bit (MSB) and z_{12} bit of this circuit, the maximum number of monomials encountered (*i.e.* during $z_{13} \xrightarrow{G}_+ r_{13}$ and $z_{12} \xrightarrow{G}_+ r_{12}$) are 429,889 and 897,955 respectively. However, the modulo-2 sum (XOR) of these ZBDDs contains only 789,604 monomials (during the modulo-2 sum common monomials cancel out) as opposed to 1,327,844 (= 429,889 + 897,955). A better approach would be to perform a word-level reduction similar to the following example.

Consider the integer multiplier circuit given in Fig. 9. A word-level approach would model the output word as $Z = z_0 + 2z_1 + 4z_2 + 8z_3$, and performs the reduction $Z \xrightarrow{G}_+ R$ across the reverse topological levels (RTTO) as depicted in the figure. (Here $+$ is addition over integers). Note that:

$$
\begin{aligned}
z_0 &= x_4 \\
z_1 &= x_2 + x_3 - 2x_2x_3 \\
z_2 &= x_1 + x_5 - 2x_1x_5 \\
&= x_1 + x_2x_3 - 2x_1x_2x_3 \\
z_3 &= x_1x_5 = x_1x_2x_3
\end{aligned}
$$

Therefore, the word-level expression $8z_3 + 4z_2 + 2z_1 + z_0$ in Z cancels out the nonlinear monomials to control intermediate monomial explosion:

$$
\begin{aligned}
Z &= 8z_3 + 4z_2 + 2z_1 + z_0 \\
&= \underline{8x_1x_2x_3} + (4x_1 + \underbrace{4x_2x_3} - \underline{8x_1x_2x_3}) \\
&+ (2x_2 + 2x_3 - \underbrace{4x_2x_3}) + x_4 \\
&= 4x_1 + 2x_2 + 2x_3 + x_4
\end{aligned}
$$

A purely bit-level GBR approach is not suitable for integer arithmetic circuit. However, this is not a limitation of our algorithms and implementations, but rather an issue of bit-level versus word-level models. Integrating the implicit data structure with a word-level representation bit vector can yield significantly better results for such applications.

VII. Conclusion

This paper has presented an approach for formal verification of datapath circuits by deriving a canonical polynomial representation for each output bit z_i of a circuit in terms of the primary inputs using Gröbner basis reduction. The gates of the circuit C are modeled as a set of polynomials G over $\mathbb{F}_2$ where the variables are the nets of the circuit. An order on the variables is derived from the topology of the circuit, and a *lex* term order (RTTO) is imposed on the polynomials. RTTO renders the set G a Gröbner basis itself. The reduction $z_i \xrightarrow{G}_+ r_i$ results in a canonical remainder r_i for each output z_i.

The polynomials in the set G are Boolean polynomials that can be construed as unate cube sets. The unate cube set algebra prowess of ZBDDs is exploited to represent the polynomials implicitly. We show that RTTO imposes a special structure on the ZBDDs, where subexpressions for leading monomials and quotients of the division are readily visible as subgraphs in the ZBDD. We take further advantage of this data structure to improve the classical Gröbner basis reduction method that relies on canceling only 1 monomial in every iteration of division. Our approach cancels multiple monomials in each step of division and generates fewer terms, thus speeding up the reduction. We have performed experiments with various finite field circuits used in cryptography. Our approach achieves significant improvement over recent approaches: the F4-style reduction, a parallelized approach for reduction, and PolyBori.

As part of our future work, we are pursuing investigations to discover a pseudo-Boolean "word-level signature" of the GBR $Z \xrightarrow{G}_+ R$, that could be integrated with implicit representations for integer arithmetic circuits.

Acknowledgment: The authors wish to thank Cunxi Yu of the University of Massachusetts, Amherst for assistance with logic synthesis and optimization of some of the benchmarks used in the experiments.

References

[1] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 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] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths," in *Computer Aided Verification Conference*, 2008, pp. 473–486.

[4] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G.-M. Greuel, "STABLE: A New QBF-BV SMT Solver for Hard Verification Problems Combining Boolean Reasoning with Computer Algebra," in *IEEE Design, Automation and Test in Europe Conference*, 2011, pp. 155–160.

[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, no. 9, 2013, pp. 1409–1420.

[6] T. Pruss, P. Kalla, and F. Enescu, "Efficient Symbolic Computation for Word-Level Abstraction from Combinational Circuits for Verification over Finite Fields," *IEEE Trans. on CAD*, vol. 35, no. 7, pp. 1206–1218, July 2016.

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

[8] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal," Ph.D. dissertation, Philosophiesche Fakultat an der Leopold-Franzens-Universitat, Austria, 1965.

[9] J.-C. Faugére, "A new efficient algorithm for computing Gröbner bases (F_4)," *Journal of Pure and Applied Algebra*, vol. 139, pp. 61–88, June 1999.

[10] X. Sun, P. Kalla, and F. Enescu, "Word-level Traversal of Finite State Machines using Algebraic Geometry," in *Proc. High-Level Design Validation and Test*, 2016.

[11] M. Ciesielski, C. Yu, D. Liu, W. Brown, and A. Rossi, "Verification of Gate-Level Arithmetic Circuits by Function Extraction," in *Proc. Des. Auto. Conf. (DAC)*, 2015.

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

[13] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal verification of integer multipliers by combining gröbner basis with logic reduction," in *Proc. Design Automation and Test in Europe*, 2016, pp. 1048–1053.

[14] Amr Sayed Ahmed and Daniel Groe and Mathias Soeken and Rolf Drechsler, "Equivalence Checking Using Grbner Bases," in *Formal Methods in Computer-Aided Design*, 2016, pp. 169–176.

[15] C. Yu and M. Ciesielski, "Efficient parallel verification of galois field multipliers," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 238–243.

[16] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," in *Design Automation Conference (DAC)*, 1993, pp. 272–277.

[17] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-6 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2012.

[18] F. Somenzi, "CUDD: CU Decision Diagram Package Release 3.0.0," 2015.

[19] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Equivalence Verification of Polynomial Datapaths with Fixed-Size Bit-Vectors using Finite Ring Algebra," in *Intl. Conf. on Computer-Aided Design, ICCAD*, 2005.

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

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

[22] O. M. Hansen and J.-F. Michon, "Boolean Gröbner basis," in *Proc. Boolean Functions Cryptography & Applications*, 2006, pp. 185–201.

[23] M. Brickenstein and A. Dreyer, "Polybori: A Framework for Gröbner Basis Computations with Boolean Polynomials," *Journal of Symbolic Computation*, vol. 44, no. 9, pp. 1326–1345, September 2009.

[24] M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the Gröbner Basis Algorithm to Find Proofs of Unsatisfiability," in *ACM Symposium on Theory of Computing*, 1996, pp. 174–183.

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

[26] M. Y. Vardi and Q. Tran, "Groebner Bases Computation in Boolean Rings for Symbolic Model Checking," in *IASTED*, 2007.

[27] T. W. Dube, "The Structure of Polynomial Ideals and Gröbner bases," *SIAM Journal of Computing*, vol. 19, no. 4, pp. 750–773, 1990.

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

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

[30] S. Minato, "Calculation of Unate Cube Set Algebra using Zero-Suppressed BDDs," in *Proc. Design Automation Conference (DAC)*, 1994, pp. 420–424.

[31] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification," www.eecs.berkeley.edu/alanmi/abc, 2007.

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

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

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

[35] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)," in *Proceedings of the Selected Areas in Cryptography*. London, UK, UK: Springer-Verlag, 1999, pp. 201–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=646554.694442

[36] G. B. Agnew, R. C. Mullin, I. Onyszchuk, and S. A. Vanstone, "An implementation for a fast public-key cryptosystem," *Journal of CRYPTOLOGY*, vol. 3, no. 2, pp. 63–79, 1991.

[37] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity word-level sequential normal basis multipliers," *Computers, IEEE Transactions on*, vol. 54, no. 2, pp. 98–110, 2005.