Boolean Gröbner Basis Reductions on Datapath Circuits using the Unate Cube Set Algebra

***Abstract*—Recent developments in formal datapath verification make efficient use of symbolic computer algebra algorithms for formal verification. The circuit is modeled as a set of polynomials over Boolean (or pseudo-Boolean) rings, and Gröbner basis (GB) reductions are performed over these polynomials to derive a canonical representation. GB reductions of Boolean polynomials tend to cause intermediate expression swell (term explosion problem) – often making the approach infeasible in a practical setting. To overcome these problems, this paper describes a logic synthesis analogue of GB reductions over Boolean polynomials, using the unate cube set algebra over characteristic sets. By representing Boolean polynomials as characteristic sets using Zero-suppressed BDDs (ZBDDs), implicit algorithms can be efficiently designed for GB-reduction on digital circuits. We show that imposition of circuit-topology based monomial orders on ZBDDs enables an implicit implementation of polynomial division, canceling multiple monomials in one-step. Experiments performed over various finite field arithmetic architectures demonstrate the efficiency of our algorithms and implementations as compared to conventional explicit 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] or finite fields [4] 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 circuit is modeled by way of a set of polynomials that generate an ideal, and the verification problem is formulated using Gröbner basis (GB) reduction techniques [5].

The GB problem exhibits high computational complexity. Indeed, computing a GB (using Buchberger's [6] or the F_4 algorithm [7]) for large circuits is practically infeasible. Managing this complexity ought be a major goal of any approach.

State-of-the-art & Limitations: Recent approaches [3] [8] have discovered that particularly for circuit verification problems, the expensive GB computation can be avoided altogether. For arbitrary combinational [3] [8] and sequential circuits [9], 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. The techniques of [3] [8] have been extended and improved further to verify arithmetic circuits. For instance, [10] and [11] get more insights from the circuit structure that dictate specific rules on the order of polynomials chosen in GB-reduction – by accounting for topological levels, reconvergent fanouts, AND-XOR gates with common inputs, etc.

A common theme among all these relevant works is that *they move the complexity of verification from one of computing a GB to that of GB-reduction (multivariate polynomial division).* These will benefit greatly by a dedicated, domain-specific implementation of GB-reduction carried out on the given circuit under RTTO $>$. So far, the above techniques [3], [4], [8], [11] use a general-purpose polynomial division approach, together with explicit set representation, for this GB-reduction. While some of these approaches do perform the reduction in some specific ways – e.g., mimicking GB-reduction under RTTO $>$ by substitution [11], or using TEDs to perform input-output signature comparisons [10], or the use of F_4-style GB-reduction on a coefficient matrix [8] – 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. We show in the sequel, that despite recent efforts, such GB-reductions can still lead to *a worst-case size explosion problem.*

Proposed Solution: To make this GB-reduction on circuits more efficient, this paper describes new techniques and implementations, specifically targeted for circuit verification under RTTO $>$. In particular, we make use of implicit characteristic set representation of Zero-Suppressed BDDs [12]. 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.

Rationale: The algebraic objects used to model the polynomial ideals derived from digital circuits are rings of Boolean polynomials. When Boolean functions are represented in $\mathbb{F}_2$ using 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 [13], are inefficient for this purpose. Since addition and multiplication (mod 2) are equivalent to AND and XOR operations, respectively, GB-reduction can be viewed as a specialized *AND/XOR Boolean function decomposition* problem. Clearly, implicit Boolean set representations such as decision diagrams could be employed for this purpose. The decision diagram of choice here is the ZBDD [12], because of its power to represent and manipulate sparse combinatorial problems – particularly "sets of combinations" using the unate cube set algebra framework.

Technical Contributions: We 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 the rudimentary polynomial division algorithms, that iteratively cancel one monomial in every step, can be implemented on ZBDDs under RTTO $>$. Subsequently, *we show that RTTO $>$ imposes a special structure on ZBDDs that allows us to cancel multiple monomials in every step of polynomial division*, thus improving GB-reduction in both space and time! Finally, experiments conducted on finite field arithmetic (crypto-circuits) benchmarks show an order of magnitude improvement using our implementation of GB-reduction.

Relationship to prior work in Boolean Gröbner Basis: The symbolic algebra community has studied properties of Boolean GB [14] [15] [16]. From among these, the work of PolyBori [16] 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 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.

II. Preliminaries: Notation and Background

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 $R = \mathbb{F}_2[x_1,\ldots,x_n]$ denote 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 $\vee, \wedge$ and $\oplus$ to denote Boolean OR, AND and XOR operations, respectively.

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 $c_1,\ldots,c_t$ are coefficients and $X_1,\ldots,X_t$ are monomials, i.e. power products of the type $x_1^{e1} \cdot x_2^{e2} \cdots x_n^{e_n}$, $e_i \in \mathbb{Z}_{\geq 0}$. To systematically manipulate the polynomials, a monomial order $>$ (also called a term order) is imposed on the ring 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. We also denote tail(f) = $f - lt(f) = c_2X_2 + \cdots + c_tX_t$. In this work, we are mostly concerned with terms ordered lexicographically (*lex*).

Definition II.1. *Let $f = c_1X_1 + \cdots + c_tX_t$ be a polynomial in $\mathbb{F}_2[x_1,\ldots,x_n]$ such that the coefficients $c_i \in \{0,1\}$, and monomials $X = x_1^{e1} \cdot x_2^{e2} \cdots x_n^{e_n}, e_i \in \{0,1\}$. 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 $\mathbb{F}_2$:

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

Polynomial division, also called reduction, plays a key role in this work. 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 $r = f - \frac{lt(f)}{lt(g)} \cdot g$. Similarly, f can be *reduced (divided) w.r.t. a set of polynomials $F = \{f_1,\ldots,f_s\}$* to obtain a remainder r. This reduction is denoted $f \xrightarrow{F}_{+} r$, and the remainder r has the property that no term in r is divisible by the leading term of any polynomial f_i in F.

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

Definition II.2. [**Gröbner Basis**] *[5]: For a monomial ordering $>$, a set of non-zero polynomials $G = \{g_1, g_2, \cdots, g_t\}$ contained in an ideal J, is called a Gröbner basis of J iff $\forall f \in J$, $f \neq 0$, there exists $i \in \{1,\cdots,t\}$ such that $lm(g_i)$ divides $lm(f)$; i.e., $G = GB(J) \Leftrightarrow\ \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f)$.*

Definition II.3. [**Gröbner Basis Reduction**] *[5]: 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. Moreover, the remainder r so obtained by GBR of f is a canonical expression modulo G.*

Proposition II.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), s.t. ideal $J = \langle F \rangle$. Let $z_i,\ i = 0,\ldots,k-1$ denote 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.*

This verification requires the computation of a Gröbner basis. Given a set of polynomials, $F = \{f_1,\ldots,f_s\}$, corresponding ideal $J = \langle F \rangle$, Buchberger's algorithm [6] computes a Gröbner basis $G = GB(J) = \{g_1,\ldots,g_t\}$. The algorithm has very high complexity ($2^{(O(n))}$ in our setting). The work of [8] showed that the GB computation can be avoided.

When the leading monomials of all polynomials in the basis $F = \{f_1,\ldots,f_s\}$ are relatively prime, then F is already a Gröbner basis ($F = GB(J)$) [17]. For a combinational circuit C, a term order $>$ can be derived by analyzing the circuit topology which ensures such a property [3] [8]:

Proposition II.2. *(From [8]) Let C be any 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, as $lt(f_i)$ and $lt(f_j)$ for $i \neq j$ are relatively prime. This term order $>$ is called the* **Reverse Topological Term Order (RTTO)**.

Subsequently, by imposing RTTO on the polynomials of the circuit, the explosive GB computation is avoided, and verification is performed by the canonical GB-reduction: $z_i \xrightarrow{G}_+ r_i$.

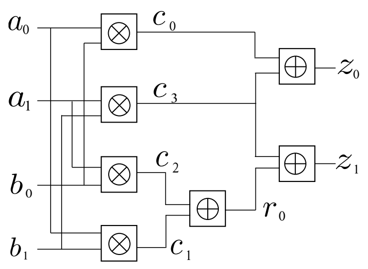

Fig. 1: A 2-bit modulo Multiplier circuit. The gate $\otimes$ is an AND-gate, and $\oplus$ is an XOR-gate; i.e. $\times, +$ modulo 2, respectively.

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

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

Example II.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 $F = \{f_1, \ldots, f_7\}$ forms a GB.*

Then the GBRs $z_1 \xrightarrow{F}_+ a_0 \cdot b_0 + a_1 \cdot b_1$ and $z_0 \xrightarrow{F}_+ a_0 \cdot b_1 + a_1 \cdot b_0 + a_1 \cdot b_1$ are canonical expressions of the output bits.

Unate Cube Sets & Boolean Polynomials: A Boolean *variable* represents a co-ordinate of the Boolean space $\mathbb{B}^n$, a *literal* is an instance of a variable x_i or its complement $\neg x_i$ or x_i'. A *cube* is a product of literals which denotes 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. Each cube in a unate cube set represents a combination, and each literal represents an object chosen in the combination.

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 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 $(a \vee bc)$ represents $(abc) : \{111, 110, 101, 100, 011\}$ 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 algebra over unate cube sets, resembling a classical logic synthesis problem, as shown below. Let us (re)consider the one-step division for Boolean polynomials: $f \xrightarrow{g} r$. This division is implemented as:

$$f \xrightarrow{g} r = f - \frac{lt(f)}{lt(g)} \cdot g = f - \frac{lm(f)}{lm(g)} \cdot g; \ \text{(coeff. 0,1; } lt(f) = lm(f))$$
$$= f + \frac{lm(f)}{lm(g)} \cdot g = f \oplus \frac{lm(f)}{lm(g)} \wedge g$$

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 $\frac{lm(f)}{lm(g)} \cdot g$.

In [18] *Minato* demonstrated that **Zero-Suppressed BDDs (ZBDDs)** [12] are an efficient data-structure for implicit manipulation (algebra) of unate cube sets. 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. For details on the use of unate cube set algebra in classical logic synthesis, and its implementation on ZBDDs, we refer the reader to [12] [18].

III. Theory and Algorithms

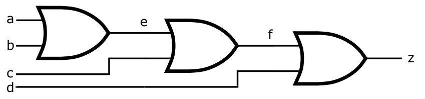

Fig. 4: A chain of OR gates

Consider the circuit in Fig. 4, impose RTTO: *lex* term order with variable ordering as, $z > f > e > d > c > b > a$. The polynomials for the OR gates are: $P_1 = z + f \cdot d + f + d$, $P_2 = f + e \cdot c + e + c$, $P_3 = e + b \cdot a + b + a$. Under RTTO, P_1, P_2, P_3 form a GB. For verification, we have to reduce the output polynomial $F = z$ modulo $G = \{P_1, P_2, P_3\}$ as $F \xrightarrow{G}_+ r$. Using a classical symbolic algebra implementation of the reduction using an explicit representation, the reduction is carried out as:

1) $z \xrightarrow{P_1} fd + f + d$
2) $fd + f + d \xrightarrow{P_2} f + edc + ed + dc + d \xrightarrow{P_2} edc + ed + ec + e + dc + d + c$
3) $edc + ed + ec + e + dc + d + c \xrightarrow{P_3} ed + ec + e + dcba + dcb + dca + dc + d + c \xrightarrow{P_3} ec + e + dcba + dcb + dca + dc + dba + db + da + d + c \xrightarrow{P_3} e + dcba + dcb + dca + dc + dba + db + da + d + cba + cb + ca + c \xrightarrow{P_3} dcba + dcb + dca + dc + dba + db + da + d + cba + cb + ca + c + ba + b + a = r$

In the first step, F is reduced by P_1 just *once* as there is only one term in F that has variable z in its support. In the second step, the result of step one is reduced *twice* by P_2 as the result has two terms that have variable f in them. Similarly, *four* reductions by P_3 are required to reduce the result of step two into an expression containing only primary inputs (which cannot be further reduced).

Observations: i) Notice that the size of the final remainder is corresponds to that of the worst case of a Boolean polynomial: i.e. r contains $2^n - 1$ monomial terms for n variables. ii)

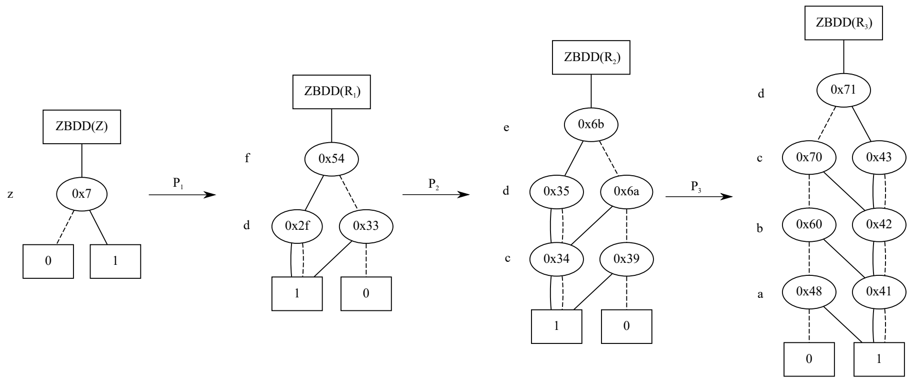

Fig. 3: Reduction of output of the circuit in Figure 4 by P_1, P_2, P_3

Classical division algorithms reduce the polynomials 1-step at a time, where only one monomial is canceled in 1-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 [13], or the ones used by [10], [11]. The F_4-style polynomial reduction of [8], [4] simulates division on a matrix M representing the problem. However, each column of M corresponds to monomial generated in the division process, therefore [8], [4] will also encounter this size explosion.

The use of ZBDDs overcomes this explosion. Fig. 3 shows the same reduction of $F = z$ by P_1, P_2, P_3 using ZBDDs (exact procedure discussed later). The size of the ZBDDs after complete reduction by P_1, P_2, P_3 increases linearly in the number of nodes. It can be shown that:

Proposition III.1. *For the worst case of the Boolean polynomial F with n variables and $2^n - 1$ monomials, the ZBDD representation of F consists of $2 \cdot n - 1$ nodes in the graph.*

ZBDD Representation: The variable ordering imposed on the ZBDDs is the same as the one derived through RTTO. Then every path from the root to terminal **1** represents a monomial, with the 1-edge (resp. 0-edge) denoting the presence (resp. absence) of the variable in the monomial. Traversing only the solid edges from the root node of f to terminal **1** delivers the leading monomial $lm(f)$. Also, the 0-child of the root node f delivers $tail(f)$.

In step 2 of division corresponding to Fig. 4, the polynomial $R_1 = fd + f + d$ needs to be reduced by P_2. The ZBDDs for R_1 and P_2 are shown in Fig 5. The ZBDD for R_1 has three paths (corresponding to fd, f, d) terminating in the node 1 that correspond to its monomials. Checking if $lm(P_2)$ divides $lm(R_1)$ becomes trivial as we just need to compare the indices of top-most nodes of ZBDDs of R_1 and P_2, which in this case are equal.

Division with ZBDDs: Cancel 1 monomial in every step
The algorithm for conventional reduction procedure using ZBDDs is shown in Algorithm 1. The procedure takes F and $POLY\_LIST$ as input arguments. F is the polynomial that we need to reduce *w.r.t.* the polynomials $G \in POLY\_LIST$ corresponding to the gates of the circuit.

Due to RTTO, the circuit polynomials are represented as $P_1 = x_1 + tail(P_1), \dots, P_d = x_d + tail(P_d)$ with variable order $x_1 > \cdots > x_d$ (Prop. II.2). Then the elements in $POLY\_LIST$ are ordered $P_1 > P_2 > \cdots > P_d$. Populating $POLY\_LIST$ in this way avoids the search required to find a polynomial $G \in POLY\_LIST$ that can divide the leading term of F. While iterating over the polynomials $G \in POLY\_LIST$ if a certain polynomial does not divide the leading term of F, it will imply that polynomial is not in the logical cone of F.

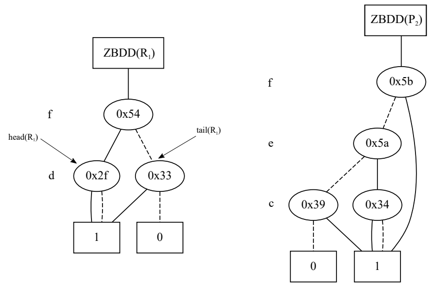

Fig. 5: ZBDD for polynomial R_1 and P_2

Algorithm 1 Reduction: Cancel 1 monomial every iteration

```
 1: procedure $single\_mon\_red(F, POLY\_LIST)$
 2:     for each $G \in POLY\_LIST$ do
 3:         $lead\_G = leading\_term(G)$
 4:         $lead\_F = leading\_term(F)$
 5:         $divide = Cudd\_zddDivide(lead\_F, lead\_G)$
 6:         while $divide \neq zero$ do
 7:             $prod = divide \cdot G$
 8:             $F = F + prod$
 9:             $lead\_F = leading\_term(F)$
10:             $divide = Cudd\_zddDivide(lead\_F, lead\_G)$
11:         end while
12:     end for
13:     return F
14: end procedure
```

The procedure $leading\_term(G)$ returns the leading term of the the ZBDD representation of polynomial G. The procedure $Cudd\_zddDivide(A,B)$ takes ZBDDs of A, B, where B can only be a cube. If B divides A, it returns the quotient of the division, else it returns zero. Line 8 iteratively computes $F = F + \frac{lt(F)}{lt(G)} \cdot G$. The polynomial F is completely reduced *w.r.t.* the polynomial G in the while loop. The $\cdot$ operator computes the product of two ZBDDs and the + operator is a modulo 2 sum: $A+B=A\cup B-A\cap B$.

Improved Reduction: Cancel multiple monomials in 1 step: Next, we will show how $R_1=fd+f+d$ can be reduced by $P_2=f+ec+e+c$ **in one step**. There are two terms in R_1 that contain f, namely fd, f. Both can be canceled by $lt(P_2)=f$ in one step, eliminating the need of the while loop in Algorithm 1.

If we check the 1-branch of node f in R_1 (Figure 3), it represents the polynomial $d+1$. Therefore, the 1-branch of the top-most node of R_1 gives us all the terms that appear with f. So the reduction can be performed by multiplying $d+1$ with P_2 and adding this product to $R_1 \pmod 2$:

$$\begin{aligned}&(fd+f+d)+(d+1)\cdot(f+ec+e+c) \pmod 2\\&=2\cdot(fd+f)+d+(d+1)\cdot(ec+e+c) \pmod 2\\&=d+(d+1)\cdot(ec+e+c) \pmod 2\end{aligned}$$

$head(R_1)$ = 1-branch (THEN) of top-most node of $R_1=d+1$,
$tail(R_1)$ = 0-branch (ELSE) of top-most node of $R_1=d$,
$head(P)$ = 1-branch of top-most node of $P=f$,
$tail(P)$ = 0-branch of top-most node of $P=ec+e+c$.

In the above example, we know that $2\cdot(fd+f) \pmod 2$ is going to be zero. Therefore, in order to reduce number of operations, we directly use the last step as a formula for reduction:

$$\begin{aligned}R_1 \xrightarrow{P}_+ &= d+(d+1)\cdot(ec+e+c)\\&\quad tail(R_1)+head(R_1)\cdot tail(P)\end{aligned}$$

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 $lt(P)$ in one-go, implicitly canceling multiple monomials in one step.*

Algorithm 2 Reduction: Cancel multiple monomials

```
1:  procedure $multi\_mon\_red\_1(F, POLY\_LIST)$
2:      for $i$ from $0,1,...,size(POLY\_LIST)-1$ do
3:          $G_i = POLY\_LIST[i]$
4:          $index_{G_i} = G_i \rightarrow index$
5:          $index_F = F \rightarrow index$
6:          if $index_{G_i} == index_f$ then
7:              $F = ELSE(F) + THEN(F)\cdot ELSE(G_i)$
8:          end if
9:      end for
10:     return F
11: end procedure
```

The algorithm for Multiple Monomial Reduction is shown in Algorithm 2, where the notations, F and $POLY\_LIST$, are same as in Algorithm 1. Unlike in Algorithm 1, however, where we need to find the quotient of $lead\_F/lead\_G$, Algorithm 2 only determines if $lead\_G$ can divide F at all (in this case the quotient is $THEN(F)$). This can be accomplished by just comparing the indices of top-most nodes of F and G. This algorithm significantly reduces the number of iterations, which now exactly equals the size of $POLY\_LIST$. For the example of Figure 4, the number of iterations is 3 using Algorithm 2, whereas 7 iterations are required using Algorithm 1.

We have implemented the above GBR procedures directly using the CUDD package. The circuit under verification is analyzed, RTTO based variable order imposed on the ZBDDs, and the Boolean polynomials of the circuit are represented as unate cube sets. The polynomials of $f_i\in G$ are inserted in $POLY\_LIST$ according to the variable order $x_1>\cdots>x_i>\cdots>x_n$, where $f_i=x_i+tail(f_i)$ (this is due to Prop. II.2). To perform GBR $z_i \xrightarrow{G}_+ r_i$ Algorithm 2 is invoked to obtain the remainder.

IV. Experimental Results

This section presents the results of reducing modulo multiplier circuits used in cryptography using our implementation. The experiments are performed on a 3.5GHz Intel Core^{TM} i7 Quad-Core CPU with 32 GB of RAM. There are two architectures that implement these multipliers, namely Mastrovito and Montgomery. Mastrovito multipliers compute $Z=A\times B \pmod P$ where P is a given primitive polynomial for the datapath size k. Montgomery multipliers perform fast modular multiplication without explicitly performing the reduction $\pmod P$. Fig. 6 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). 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. We have presented results for GBR on both *flattened* and *hierarchical* netlists of these multipliers.

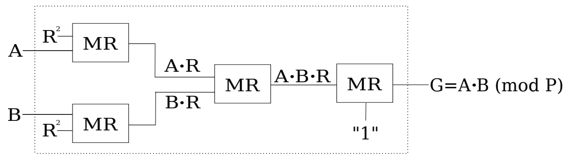

Fig. 6: Montgomery multiplication

Table I provides the statistics on the time, ZBDD size, and savings in the number of monomial cancellations achieved by our approach as compared to classical symbolic algebra methods, for Mastrovito multiplier circuits. The time reported is the total time required to perform the reduction, $z_i \xrightarrow{G}_+ r_i$, for all $i=0,\dots,k-1$. The columns *F4*, *PB*, and *ZR* represent the time taken in seconds by F4 style reduction, PolyBori (version 0.8.3) and our implementation respectively. *MN* represents the maximum size of any ZBDD encountered during reduction of any output-bit, while *MR* represents the maximum size of the ZBDD among all final remainders r_i. *CS* represents the number of iterations saved during reduction due to our implicit method. PolyBori throws a segmentation fault for the multiplier with $k=571$ which is depicted as *CR* in the table. Our approach is orders of magnitude faster than both *F4* and *PB*.

Table II provides the same statistics for *flattened* Montgomery multipliers. The data for *MN* and *CS* columns in case of $k=571$ could not be procured as both these operations require traversal along ZBDDs (using the procedure `Cudd_zddDagSize()` and `Cudd_zddNextPath`). As there is a huge number of large-sized intermediate ZBDDs produced during this reduction, traversing along each path of every ZBDD is infeasible. Also, the reduction times for $k=283$ is inordinate when compared to other datapath sizes, which is purely due to the structure of the circuit.

Table III present the statistics for hierarchical Montgomery multipliers for the blocks A, B, C, and D. The experiment first reduces the outputs of a 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 III shows the time for reduction of each block and the time for reducing the primary outputs across the four levels. The time for reducing the primary outputs across levels in case of F4 implementation is <1 second, and is not explicitly mentioned in the table. The row labeled *Total* presents the sum of time of reduction across levels and the maximum reduction time for each block (as the reductions for the four levels are independent of each other and are done in parallel). The datapath sizes of these circuits correspond to NIST-specification in cryptography with $k = 163, \ldots, 571$ bits.

TABLE I: Mastrovito Multipliers (Time in seconds, timeout = 3 days, K = 10^3, M = 10^6)

k	**# Gates**	**F4**	**PB**	**ZR**	**MN/MR**	**CS**
163	153K	1,443s	70s	**10s**	811/765	153K
233	167K	1,913s	105s	**14s**	772/699	167K
283	399K	11,116s	316s	**45s**	1,413/1,402	400K
409	508K	17,848s	596s	**75s**	1,313/1,227	508K
571	1.6M	192,032s	CR	**616s**	2,849/2,840	1.6M

TABLE II: Montgomery Flat Multipliers (Time in seconds, timout = 3 days, K = 10^3, M = 10^6, B = 10^9)

k	**# of Gates**	**F4**	**PB**	**ZR**	**MN/MR**	**CS**
163	184K	6,897s	**9294s**	9,595	39.4K/765	823M
233	329K	63,805s	1749s	**1,452s**	4.4K/699	186M
283	488K	TO	**127,096s**	247,837s	117K/1.4K	8.7B
409	1.0M	TO	**19,679s**	32,226s	9.5K/1.2K	1.4B
571	1.97M	TO	CR	**128,464s**	TO/2.9K	TO

TABLE III: Montgomery Blocks(Time in seconds, Red. = time for reduction, Coll. = time to reduce across the 4 levels.)

k	**# of Gates**	**Block**	**F4**	**PB**		**ZR**	
				Red.	Coll.	Red.	Coll.
163	33K	Block A	25	12	**21**	1	33
	33K	Block B	25	12		1	
	85K	Block Mid	73	**18**		12	
	32K	Block Out	24	13		1	
	Total		73	**39**		45	
233	55K	Block A	142	34	6	<1	**9**
	55K	Block B	141	36		<1	
	163K	Block Mid	408	39		**13**	
	54K	Block Out	140	33		<1	
	Total		408	45		**22**	
283	82K	Block A	330	98	**35**	28	158
	82K	Block B	329	101		29	
	241K	Block Mid	883	**202**		254	
	81K	Block Out	321	92		30	
	Total		883	**237**		412	
409	168K	Block A	1,322	188	36	<1	**58**
	168K	Block B	1,335	189		<1	
	502K	Block Mid	4,471	200		**117**	
	168K	Block Out	1,338	191		<1	
	Total		4,471	236		**175**	
571	330K	Block A	5,371	1,620	1,566	725	**1,516**
	330K	Block B	5,421	1,843		752	
	980K	Block Mid	37,804	5445		**4,164**	
	328K	Block Out	5,539	1,831		747	
	Total		37,804	7,011		**5,680**	

V. Conclusion

This paper has presented an approach to derive a canonical polynomial representation for each output bit z_i of a circuit, by modeling the gates of the circuit as a set of polynomials G over $\mathbb{F}_2$, and performing the reduction $z_i \xrightarrow{G}_+ r_i$. The unate cube set algebra prowess of ZBDDs is exploited to represent the polynomials implicitly. We take further advantage of this data structure to improve the classical Gröbner basis reduction method that relies on cancelling only 1 monomial in every iteration of division. Our approach cancels multiple monomials in each step of division, thus speeding up the reduction. The efficiency of our approach is demonstrated by completing the reduction for up to 571-bit modulo multipliers in the allotted time, and significant improvement is achieved over the F4-style reduction and PolyBori based techniques. As part of our future work, we will be pursuing word-level implementation of the polynomials reduction over integer arithmetic circuits.

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] 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] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "Singular 3-1-3 — A computer algebra system for polynomial computations," 2011, http://www.singular.uni-kl.de.

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

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

[16] 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.

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

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