A Graph-Based Unified Technique for Computing and Representing Coefficients over Finite Fields

Abusaleh M. Jabir, *Member*, *IEEE*, and Dhiraj K. Pradhan, *Fellow*, *IEEE*

Abstract—This paper presents the generalized theory and an efficient graph-based technique for the calculation and representation of coefficients of multivariate canonic polynomials over arbitrary finite fields in any polarity. The technique presented for computing coefficients is unlike polynomial interpolation or matrix-based techniques and takes into consideration efficient graph-based forms which can be available as an existing resource during synthesis, verification, or simulation of digital systems. Techniques for optimization of the graph-based forms for representing the coefficients are also presented. The efficiency of the algorithm increases for larger fields. As a test case, the proposed technique has been applied to benchmark circuits over $\mathrm{GF}(2^m)$. The experimental results show that the proposed technique can significantly speed up execution time.

Index Terms—Finite or Galois fields, decision diagrams, coefficients, polynomials.

1 Introduction

The motivation of this paper is efficient representation of digital systems as word-level canonic polynomials and their graphs over finite fields for synthesis, verification, and testing, and also error correction. The underlying problem seems to be the determination of the coefficients associated with the multivariate canonic polynomials, which in itself is a very hard problem. Hence, this paper focuses mainly on three aspects:

- the unified theory behind representing functions as canonic polynomials and their graphs in finite fields,
- an efficient graph-based technique for calculation of coefficients in arbitrary finite fields in any polarity, and
- canonic graph-based representation of the coefficients.

1.1 Previous Work

Polynomials over finite fields are crucial to certain public-key crypto systems (for example, the elliptic curve (EC) cryptography) for data protection [31], error control codes (for example, BCH and Reed-Solomon codes) for encountering channel errors [30], [36], and digital signal processing [6]. These are especially significant due to the rapid growth of the mobile and wireless industries, where data security and reliability are very important in devices such as mobile phones, palm-top or pocket computers with wireless capabilities, and so forth. These crypto systems require low-complexity high-performance realizations of these polynomials for fast computations over large finite fields [4], [8]. Given $r \cdot s = m$, the field $\mathrm{GF}(2^m)$ can be represented as an extension of the subfield $\mathrm{GF}(2^r)$ as $\mathrm{GF}((2^r)^s)$, for example, the polynomials over $\mathrm{GF}((2^r)^s)$ can be represented in terms of the coefficients over $\mathrm{GF}(2^r)$, which occurs frequently in the EC cryptosystem [8]. Our technique can inherently handle this scenario. The role of finite fields in error control systems is well established and contributes to many fault-tolerant designs. In the EDA industry multivalued functions, especially in the form of *(M)ultivalued (D)ecision (D)iagrams* or MDDs, play an important role [34]. Multivalued functions can also be represented in finite fields, as shown in [24]. The varied use of finite fields leads to designing high speed, low-complexity systolic very large-scale integrated (VLSI) realizations [10]. Applications of finite fields in Reed-Müller-like canonic polynomial forms have also been proposed for matching algorithms for verification, for example, [9].

Arbitrary combinations of bits or nibbles within a word can be combined together and represented as canonic polynomials over finite fields [3], [11], [28]. Being in canonic form, these polynomials and their graphs allow *error correction* in addition to fast verification—something that is not directly possible by formal methods such as model checkers [3]. Owing to the canonicity, two polynomials in finite fields can be compared by term-by-term polynomial addition/subtraction to find an error polynomial (for example, by *mitering*), which can be added to the erroneous circuit to correct the error.

The underlying problem of representing circuits as word-level canonic polynomials over finite fields is the efficient determination of the coefficients. A number of techniques exist based on interpolation algorithms for small finite fields (for example, $\mathrm{GF}(K)$ and $K \leq 4$), e.g., [1], [15], [38], [39]. Although the technique in [20] seems to be applicable to large fields, it heavily relies on the size of the fields and also, for small fields, a solution only exists if the

- *A.M. Jabir is with the School of Technology, Oxford Brookes University, Oxford OX33 1HX, UK. E-mail: ajabir@brookes.ac.uk.*
- *D.K. Pradhan is with the Department of Computer Science, University of Bristol, Bristol BS8 1UB, UK. E-mail: pradhan@cs.bris.ac.uk.*

Manuscript received 29 Nov. 2005; revised 29 June 2006; accepted 29 Mar. 2007; published online 20 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0423-1105.
Digital Object Identifier no. 10.110/TC.2007.1060.

interpolation points are chosen from sufficiently large extension fields [21]. Most of these techniques (for example, [38], [39]) are suitable for only the 0-polarity (that is, only one polarity). However, most practical circuits have different representations in different polarities in finite fields, for example, certain polarities will require fewer terms or nonzero coefficients, which could result in fewer nodes or paths in a graph than others for the same circuit. In our approach, given a directed acyclic graph (DAG) as the initial representation, word size, polarity number, and *primitive polynomials* (PP), we compute the coefficients of multivariate canonic polynomials in the given polarity over finite fields generated with the PP from the initial DAG and also store the coefficients in another DAG. In this regard, Sasao and Izuhara [35] present a technique for computing the optimal Reed-Müller expressions, that is, the expression under the optimal polarity, over $\mathrm{GF}(2)$ only from minterm-based representations. Due to its exhaustive nature, its application is limited to small circuits. Also, Jankovic et al. [11] present another exhaustive search-based technique for finding optimal expressions over $\mathrm{GF}(4)$ from minterms based on an extension of the dual polarity property over $\mathrm{GF}(2)$. Fourier-like matrix-based algorithms (for example, based on coding theory and the butterfly algorithm) exist for functions in finite fields and also generalized *Reed-Müller* forms [12], [16], [17]. However, the size of the matrices is exponential and grows exponentially with the size of the inputs and field sizes.

1.2 Background

Let $\mathrm{GF}(N)$ denote a set of N elements, where N is a power of a prime number with two special elements 0 and 1 representing the additive and multiplicative identities, respectively, and two operators, addition "+" and multiplication "$\cdot$". $\mathrm{GF}(N)$ defines a finite field, also known as a *Galois Field*, if it forms a *commutative ring* with identity over these two operators in which every element has a multiplicative inverse. Additional properties of finite fields can be found in [24], [36].

Let $I_N = \{0, 1, \ldots, N-1\}$. Let $\delta : I_N \rightarrow GF(N)$ be a one-to-one mapping with $\delta(0) = 0$, $\delta(1) = 1$, and, without loss of generality, $\delta(2) = \alpha$, $\delta(3) = \beta$, and so forth, where α, β, and so forth are elements in $\mathrm{GF}(N)$.

Let the notation $f|_{x_k=y}$ represent the fact that all occurrences of x_k within the function f are replaced with y, that is, $f|_{x_k=y} = f(x_1, x_1, \ldots, x_k = y, \ldots, x_n)$.

Finite fields over $\mathrm{GF}(2^m)$ and $m \geq 2$ can be generated with PP of the form $p(x) = x^m + \sum_{i=0}^{m-1} c_i x^i$, where $c_i \in \mathrm{GF}(2)$ [36]. In this paper, PPs will be considered in their decimal notation. For example, the PP $p(x) = x^3 + x + 1$ for $\mathrm{GF}(2^3)$ can be represented by the bit vector [1, 0, 1, 1], which is 11 in decimal. Note that only one field is possible for $\mathrm{GF}(2)$ and $\mathrm{GF}(4)$, that is, for $m \leq 2$, but multiple fields are possible for $m > 2$ for the same m.

Given a variable, x_i in $\mathrm{GF}(N)$ and $\pi_i \in I_N$ $(1 \leq i \leq n)$, x_i can appear in one of the N *polarities* denoted by $x_{i,\pi_i} = x_i + \delta(\pi_i)$. Certain polarities require more resources to represent a polynomial than others. As an example, let $f(x_1, x_2) = \alpha x_1 x_2$ represent a function in 0-polarity in $\mathrm{GF}(3)$, that is, both x_1 and x_2 are in 0-polarity (note that $x_{i,0} = x_i$). In another polarity, with x_1 in 1-polarity and x_2 in 2-polarity, replacing x_1 with $x_{1,1} - 1$ and x_2 with $x_{2,2} - \delta(2) = x_{2,2} - \alpha$, and, simplifying, we get $f(x_1, x_2) = 1 + x_{2,2} + \alpha x_{1,1} + \alpha x_{1,1} x_{2,2}$, which contains more terms in this case. Finding an optimal polarity is a very hard problem.

The following result will be used in this paper:

Theorem 1 [23]. *Any function $f(x_1, \ldots, x_k, \ldots, x_n)$ in $\mathrm{GF}(N)$ can be expanded as follows:*

$$f(x_1, \ldots, x_k, \ldots, x_n) = \sum_{e=0}^{N-1} g_e(x_k) f|_{x_k=\delta(e)}, \tag{1}$$

where $g_e(x_k) = 1 - [x_k - \delta(e)]^{N-1}$.

1.3 Graph-Based Representation

In Theorem 1, the function $g_e(x_i)$ is called a *literal* in $\mathrm{GF}(N)$ and (1) is called the *literal-based* expansion in $\mathrm{GF}(N)$. Theorem 1 allows MDD-like canonic DAG representation of any function in $\mathrm{GF}(N)$ in its literal form [2], [3]. However, since an MDD is defined in MIN-MAX postalgebra [14], [34], this representation will be called a *(M)ultiple (O)utput (D)ecision (D)iagram* or MODD to distinguish it from an MDD. There are two MODD reduction rules [2]: 1) If all of the N children of a node v point to the same node w, then delete v and connect the incoming edge of v to w. 2) Share equivalent subgraphs. A reduced MODD can be further optimized and normalized based on two additional rules presented in [3]. An MODD which is reduced by all four rules will be called a *(Z)ero suppressed and (N)ormalized MODD* or ZNMODD [3].

This paper is organized as follows: Section 2 presents a generalized canonic polynomial expansion of functions in $\mathrm{GF}(N)$. This section also presents a technique for efficiently representing the coefficients by means of DAGs. Section 3 presents an efficient DAG-based technique for computing the coefficients of any canonic polynomial over $\mathrm{GF}(N)$. Finally, Section 4 presents experimental results that reflect the execution profile of the proposed technique.

2 Polynomial Expansion over $\mathrm{GF}(p^n)$

This section presents a generalized expansion technique for canonic polynomials and their DAG-based representations over $\mathrm{GF}(N)$ in any polarity. Techniques such as [5], [13], [28] [29], [32] are either Reed-Müller expansions or expansions over finite fields of fixed order, for example, $\mathrm{GF}(2)$ or $\mathrm{GF}(4)$, although, theoretically, they can be generalized to higher order fields.

First, the following result is presented from combinatorics, which will be useful later.

Theorem 2. *For any nonzero positive integer n, prime number p, and $0 \leq r < p^n$, $\binom{p^n-1}{r} mod\ p = \binom{p-1}{r \bmod 2}$.*

Proof. Follows by induction on r. □

Theorem 2 gives the following repetitive pattern for $0 \leq r < p^n$:

$$\underbrace{1, p-1, 1, \ldots, 1, p-1, 1}_{p^n \text{times}}.$$

Theorem 2 yields the following for $0 \leq r < 2^n$:

Corollary 2.1. *For any nonzero positive integer n and $0 \le r < 2^n$, $\binom{2^n-1}{r}$ is an odd number.*

We have the following unified polynomial expansion, called the *(P)arameterized (G)alois (E)xpansion* or PGE, in $\mathrm{GF}(N)$.

Theorem 3. *Let $f(x_1, \ldots, x_i \ldots, x_n)$ represent an n-variable function in $\mathrm{GF}(N)$, where $N = p^m$ and p is a prime number and m an integer. Let $0 \le \pi_i < N$ represent the polarity of the variable x_i, that is, $x_{i,\pi_i} = x_i + \delta(\pi_i)$. Then, the PGE of f is*

$$\begin{aligned}\mathcal{PGE}_{\pi_i,i}(f(x_1,\ldots,x_i,\ldots,x_n)) &= \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e) f|_{x_i=\delta(e)} \\ &+ \left(\sum_{r=0}^{N-2} x_{i,\pi_i}^{N-1-r} \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,r}(e) f|_{x_i=\delta(e)}\right),\end{aligned} \tag{2}$$

where $\mathcal{H}_{\pi_i}(e) = 1 - (\delta(\pi_i) + \delta(e))^{N-1}$ and

$$\mathcal{C}_{\pi_i,r}(e) = (-1)^{r+1} \binom{p-1}{r \bmod 2} (\delta(\pi_i) + \delta(e))^r.$$

Proof. (Proof Sketch) From the definition of π_i-polarity, $x_i = x_{i,\pi_i} - \delta(\pi_i)$. After substituting $x_{i,\pi_i} - \delta(\pi_i)$ for x_i in Theorem 1, applying binomial theorem, and rearranging and then factorizing the terms, we have

$$\begin{aligned}f(x_1,\ldots,x_i,\ldots,x_n) &= \sum_{e=0}^{N-1} \left[1 - (\delta(\pi_i) + \delta(e))^{N-1}\right] f|_{x_i=\delta(e)} \\ &+ \left(\sum_{r=0}^{N-2} x_{i,\pi_i}^{N-1-r} \sum_{e=0}^{N-1} (-1)^{r+1} \binom{N-1}{r} (\delta(\pi_i) + \delta(e))^r f|_{x_i=\delta(e)}\right).\end{aligned}$$

From the properties of $\mathrm{GF}(N)$, it follows that

$$\begin{aligned}\forall a \in \mathrm{GF}(N)\ \binom{N-1}{r} \cdot a &= \binom{p^m-1}{r} \cdot a \\ &= \left(\binom{p^m-1}{r} \bmod p\right) \cdot a.\end{aligned}$$

However, from Theorem 2, $\binom{p^m-1}{r} \bmod p = \binom{p-1}{r \bmod 2}$. Hence, the proof follows. □

Special Case over GF(2^m). Finite fields over $\mathrm{GF}(2^m)$ have many applications, especially for verification and simulation of digital systems, error control and data security, DSP, and so forth [2], [4], [6], [9], [18], [19], [36].

Corollary 2.1, together with the fact that $-a = +a$, $\forall a \in \mathrm{GF}(2^m)$, gives the following special case of Theorem 3 in $\mathrm{GF}(2^m)$:

Theorem 4. *Let $f(x_1, \ldots, x_i \ldots, x_n)$ represent an n-variable function in $\mathrm{GF}(N)$, where $N = 2^m$. Let $0 \le \pi_i < N$ represent the polarity of the variable x_i, that is, $x_{i,\pi_i} = x_i + \delta(\pi_i)$. Then, the PGE of f is*

$$\begin{aligned}\mathcal{PGE}_{\pi_i,i}(f(x_1,\ldots,x_i,\ldots,x_n)) &= \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e) f|_{x_i=\delta(e)} \\ &+ \left(\sum_{r=0}^{N-2} x_{i,\pi_i}^{N-1-r} \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,r}(e) f_{x_i=\delta(e)}\right),\end{aligned} \tag{3}$$

where $\mathcal{H}_{\pi_i}(e) = 1 + (\delta(\pi_i) + \delta(e))^{N-1}$ and

$$\mathcal{C}_{\pi_i,r}(e) = (\delta(\pi_i) + \delta(e))^r.$$

Note that, in Theorem 3, depending on the value of e, $\mathcal{H}_{\pi_i}(e) \in \{0, 1\}$, whereas each $\mathcal{C}_{\pi_i,r}(e) \in \mathrm{GF}(N)$.

Example 1. Let us consider $\mathrm{GF}(2)$ and $\pi_i = 1$. From Theorem 4,

$$\begin{aligned}&\mathcal{PGE}_{1,i}(f(x_1,\ldots,x_i,\ldots,x_n)) = \sum_{e=0}^{1} [1 + (\delta(1) + \delta(e))] f|_{x_i=\delta(e)} \\ &+ \left(\sum_{r=0}^{0} x_{i,1}^{1-r} \sum_{e=0}^{1} (\delta(1) + \delta(e))^r f|_{x_i=\delta(e)}\right) \\ &= [1+1+0] f|_{x_i=0} + [1+1+1] f|_{x_i=1} \\ &+ x_{i,1}(1 \cdot f|_{x_i=0} + 1 \cdot f|_{x_i=1}) \\ &= f|_{x_i=1} + x_{i,1}(f|_{x_i=0} + f|_{x_i=1}),\end{aligned}$$

which is clearly the negative Davio expansion of f [35]. Similarly, the positive Davio expansion of f can be obtained from $\mathcal{PGE}_{0,i}(f)$.

Example 2. Let us consider $\mathrm{GF}(4)$ and $\pi_i = 3$. From Theorem 4,

$$\begin{aligned}&\mathcal{PGE}_{3,i}(f(x_1,\ldots,x_i,\ldots,x_n)) \\ &= \sum_{e=0}^{3} [1 + (\delta(3) + \delta(e))^3] f|_{x_i=\delta(e)} \\ &\quad + \sum_{r=0}^{2} x_{i,3}^{3-r} \left(\sum_{e=0}^{3} (\delta(3) + \delta(e))^r f|_{x_i=\delta(e)}\right) \\ &= f|_{x_i=\beta} + \Big(x_{i,3}^3 (f|_{x_i=0} + f|_{x_i=1} + f|_{x_i=\alpha} + f|_{x_i=\beta}) \\ &\quad + x_{i,3}^2 (\beta f|_{x_i=0} + \alpha f|_{x_i=1} + f|_{x_i=\alpha}) \\ &\quad + x_{i,3} (\beta^2 f|_{x_i=0} + \alpha^2 f|_{x_i=1} + f|_{x_i=\alpha})\Big) \\ &= f|_{x_i=\beta} + x_{i,3}^3 (f|_{x_i=0} + f|_{x_i=1} + f|_{x_i=\alpha} + f|_{x_i=\beta}) \\ &\quad + x_{i,3}^2 (\beta f|_{x_i=0} + \alpha f|_{x_i=1} + f|_{x_i=\alpha}) \\ &\quad + x_{i,3} (\alpha f|_{x_i=0} + \beta f|_{x_i=1} + f|_{x_i=\alpha}),\end{aligned}$$

which is the 3-4 negative Davio expansion in [28].

Similarly, the 4-positive, 1-4 negative, and 2-4 negative Davio expansions in [28] can be obtained from Theorem 3 as $\mathcal{PGE}_{0,i}(f)$, $\mathcal{PGE}_{1,i}(f)$, and $\mathcal{PGE}_{2,i}(f)$, respectively.

In this way, any polarity polynomial expansion can be obtained in $\mathrm{GF}(N)$.

Apply Operation. We have the following regarding operations between two functions in $\mathrm{GF}(N)$:

Lemma 1 [2]. *Let $f(x_1, \ldots, x_i, \ldots, x_n)$ and $h(x_1, \ldots, x_i, \ldots, x_n)$ be two functions in $\mathrm{GF}(N)$ and "$\odot$" represent an algebraic operation (for example, addition, multiplication, subtraction, and division) in $\mathrm{GF}(N)$. Then,*

$$f \odot h = \sum_{e=0}^{N-1} [1 - (x_i - \delta(e))^{N-1}](f|_{x_i=\delta(e)} \odot h|_{x_i=\delta(e)}). \quad (4)$$

This can be expressed in terms of two PGEs as follows.

Lemma 2. *Let $f(x_1, \ldots, x_i, \ldots, x_n)$ and $h(x_1, \ldots, x_i, \ldots, x_n)$ be two functions in* GF(N), *and "$\odot$" represent an algebraic operation (for example, addition, multiplication, subtraction, and division) in* GF(N). *Then,*

$$\mathcal{PGE}_{\pi_i,i}(f) \odot \mathcal{PGE}_{\pi_i,i}(h) = \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e)\left(f|_{x_i=\delta(e)} \odot h|_{x_i=\delta(e)}\right) + \left(\sum_{r=0}^{N-2} x_{i,\pi_i}^{N-1-r} \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,r}(e)\left(f|_{x_i=\delta(e)} \odot h|_{x_i=\delta(e)}\right)\right). \quad (5)$$

Proof. Follows from manipulation of Lemma 1 as in the proof of Theorem 3. □

Theorem 3 can be represented in terms of Fourier-like matrix expansions such as those in [26], but this has been left out for brevity.

2.1 Graph-Based Representation

π_i-polarity Expansion. Equation (2) can be recursively expanded with respect to each variable. For example, in (2), let x_j $(j \neq i)$ be another variable. Then, it can be expanded in terms of $\mathcal{PGE}_{\pi_j,j}(f|_{x_i=\delta(e)})$, $\forall e \in I_N$, that is,

$$\mathcal{PGE}_{\pi_i,i}(f) = \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e)\mathcal{PGE}_{\pi_j,j}\left(f|_{x_i=\delta(e)}\right) + \left(\sum_{r=0}^{N-2} x_{i,\pi_i}^{N-1-r} \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,r}(e)\mathcal{PGE}_{\pi_j,j}\left(f|_{x_i=\delta(e)}\right)\right). \quad (6)$$

Equation (2) can also be written as

$$\begin{aligned} &\mathcal{PGE}_{\pi_i,i}(f(x_1, \ldots, x_i, \ldots, x_n)) \\ &= \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e) f|_{x_i=\delta(e)} \\ &+ \left(\sum_{r=1}^{N-1} x_{i,\pi_i}^{r} \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,N-2-(r-1)}(e) f|_{x_i=\delta(e)}\right) \\ &= h_{\pi_i,0} + \sum_{r=1}^{N-1} x_{i,\pi_i}^{r} h_{\pi_i,r}, \end{aligned} \quad (7)$$

where

$$h_{\pi_i,s} = \begin{cases} \sum_{e=0}^{N-1} \mathcal{H}_{\pi_i}(e) f|_{x_i=\delta(e)} & \text{if } s < 1 \\ \sum_{e=0}^{N-1} \mathcal{C}_{\pi_i,N-2-(s-1)}(e) f|_{x_i=\delta(e)} & \text{otherwise} \end{cases}$$

and $s \in I_N$.

Now, (7) can be represented based on an MDD-like structure, as shown in Fig. 1. Here, the internal nodes represent the variables of expansion in π_i-polarity, while the terminal nodes represent the coefficients of expansion. The edges denote the exponents of x_{i,π_i}. To distinguish between representation of MIN-MAX-based postalgebra using an MDD, we call such a diagram the *(G)alois (P)olynomial* DD or GPDD.

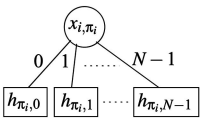


Fig. 1. Decision diagram-based representation of $\mathcal{PGE}_{\pi_i,i}(f)$.

Equation (7) can be recursively expanded as in (6), which will result in a decision diagram.

Reduction. The GPDD which results can be reduced as follows: Let us consider two nodes v and w such that $child_i(v) = w$ and $0 \leq i \leq N-1$.

1. If all of the nonzero edges of w point to the zero terminal node, then reconnect $child_i(v)$ to $child_0(w)$ and delete w.
2. Share equivalent subgraphs.

A linear time GPDD reduction algorithm can be derived based on these rules from the binary decision diagrams (BDD) reduction algorithm in [7]. However, two things need to be considered: 1) The number of children has to be extended to N from 2 and 2) the BDD reduction rule has to be replaced with the first reduction rule above, whereas the sharing rule remains the same. A linear time recursive reduction algorithm has been developed and incorporated into our tool presented in Section 4. This algorithm visits each node exactly once during the reduction process.

As with the ZNMODD, further node reduction can be obtained in a similar manner:

- *Zero Suppression.* Suppress the 0-terminal node, along with all the edges pointing to it.
- *Normalization.* Move the values of the nonzero terminal nodes as weights to the edges and ensure that 1) the weight of a specific valued edge (for example, that with the highest value) is always 1 and 2) assuming P represents the set of all the paths, $\forall z \in P$, the GF(N) product of all the weights along z is equal to the coefficient of the term in the polynomial, which is represented by z.

Although normalization rules have been considered in [22] and [27], those rules are in the integer domain rather than in finite fields and they are also based on the relative primeness of the weights of the edges. It can be argued that these reduction rules will render the diagram canonic and minimal if the weights are assigned in a certain order throughout the graph during normalization. Such a reduced GPDD will be called *(Z)ero suppressed (N)ormalized* GPDD or ZNGPDD.

Example 3. Let $\{0, 1, \alpha\}$ represent the elements in GF(3). Let $f(x_1, x_2) = x_1 + \alpha x_2$. The truth vector of this function is

$$F_f(x_1, x_2) = [0\alpha 110\alpha\alpha 10].$$

Based on this truth-vector, a ZNMODD can be constructed, which appears in Fig. 2a. Here, the lines with zero, one, and two cuts represent 0, 1, and α, respectively.

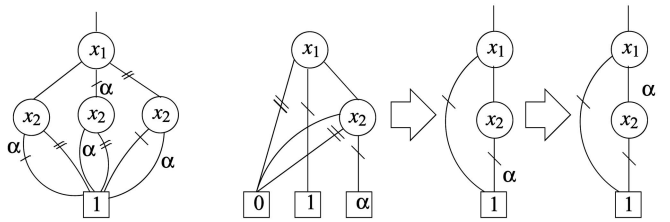


Fig. 2. DDs in Example 3. (a) ZNMODD, (b) GPDD, (c) and (d) gradual reduction to ZNGPDD.

Now, in GF(3),

$$\mathcal{PGE}_{0,i}(f) = f|_{x_i=0} + x_{i,0}(\alpha f|_{x_i=1} + f|_{x_i=\alpha}) - x_{i,0}^2(f|_{x_i=0} + f|_{x_i=1} + f|_{x_i=\alpha}). \quad (8)$$

Equation (8) is the 0-polarity PGE in GF(3). Recursive expansion of (8) will result in a reduced GPDD shown in Fig. 2b. Here, the lines with zero, one, and two cuts represent 0, 1, and 2, respectively. Fig. 2c and Fig. 2d show the gradual conversion to the ZNGPDD. The weights have been moved around and adjusted to ensure the correctness of the underlying function.

Fig. 2d can be used to recursively construct the original polynomial as follows:

$$1 \cdot 1 \cdot x_1^1 + 1 \cdot \alpha \cdot 1 \cdot x_1^0 x_2^1 = x_1 + \alpha x_2,$$

which is $f(x_1, x_2)$.

Again, let $g(x_1, x_2) = x_1 x_2$ in GF(3). The ZNMODD requires three internal nodes for this function, whereas the ZNGPDD requires just two.

As an another example, Fig. 3 shows the reduced MODD and GPDD for a 2-bit integer multiplier in 0-polarity GF(4) (that is, 2-bit word size) as generated by our tool. Here, outputs f_0 and f_1 represent the 2-bit least significant and most significant digits of the multiplier, respectively. The numbers 0-3 represent elements in GF(4).

This example brings to light the following: Given an n input function $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i$ in GF(N), it can be shown by generalizing the number of nodes in a BDD for parity functions that its reduced MODD will require $N(n-1)+1$ internal nodes. Its reduced GPDD will require n internal nodes, each one corresponding to one of the x_is. Assuming node v_i corresponds to x_i $v_{i+1} = child_0(v_i)$ for $i < n$, $\forall i$ $child_1(v_i)$ will point to a nonzero terminal node and the rest of the edges will point to the zero terminal node. Hence, the GPDD will require approximately N times fewer internal nodes. This can be shown to be true irrespective of the variable ordering for these types of functions.

3 Computation of Coefficients

Given a function $f(x_n, x_{n-1}, \ldots, x_1)$ in GF(N) and a polarity number t, it can be shown that recursive application of Theorem 3 with respect to *all* of the variables with their corresponding polarities, as determined by t, will result in a canonic polynomial of the following form [12], [24]:

$$f(x_n, x_{n-1}, \ldots, x_1) = \kappa_{0,t} + \kappa_{1,t}\tilde{x}_1 + \kappa_{N,t}\tilde{x}_2 + \cdots + \kappa_{i,t}\tilde{x}_{j_k}^{i_k}\tilde{x}_{j_{k-1}}^{i_{k-1}} \cdots \tilde{x}_{j_1}^{i_1} + \cdots + \kappa_{N^n-1,t}\tilde{x}_n^{N-1}\tilde{x}_{n-1}^{N-1}\cdots\tilde{x}_1^{N-1}. \quad (9)$$

Here, we have used the notation in [24] as follows: $\kappa_{i,t}$ represents the coefficient of the ith term containing k number of variables in polarity number t, where both i and t are defined in the radix-N number system as $i = i_k N^{j_k} + i_{k-1} N^{j_{k-1}} + \cdots + i_1 N^{j_1}$ and $t = \pi_n N^{n-1} + \pi_{n-1} N^{n-2} + \cdots + \pi_1$.[1]

Further, each π_q represents the polarity of the variable x_q and $1 \le q \le n$, that is, $x_{q,\pi_q} = x_q + \delta(\pi_q)$. Here, $\tilde{x}_l$ represents the fact that x_l is in any one of the polarities $\{0, 1, \ldots, N-1\}$ as determined by t. For example, the term associated with the coefficient $\kappa_{i,t}$ is $\tilde{x}_{j_k}^{i_k}\tilde{x}_{j_{k-1}}^{i_{k-1}}\cdots\tilde{x}_{j_1}^{i_1}$, where the polarity of each variable is determined by t.

Such an expansion will determine all of the coefficients. Unfortunately, this recursive expansion will *always* be exponential. The coefficients can also be computed based on a Fourier-type expansion [12], [17]. However, this approach requires the storage of an $N^n \times N^n$ matrix, which can blow up quite rapidly, especially for large fields. Considering these, an efficient DAG-based technique has been presented in the following for computing the coefficients.

3.1 Graph-Based Coefficient Computation

In many cases, the MODD (or MDD) may be available as a part of the existing resources during a synthesis, verification or simulation process, in which case, the coefficients can be derived from the graphs by means of an efficient *incremental* technique, as shown below.

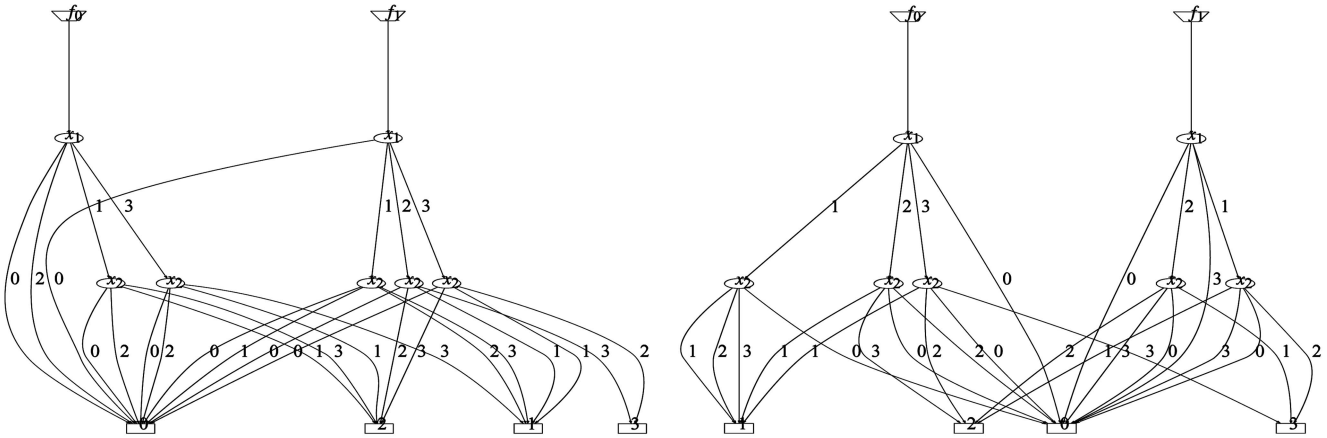
Let the ith term in a canonic polynomial in GF(N) be $\tilde{x}_{j_k}^{i_k}\tilde{x}_{j_{k-1}}^{i_{k-1}}\cdots\tilde{x}_{j_1}^{i_1}$, which is associated with the coefficient $\kappa_{i,t}$. Let us define the function $\theta_{i,t}(\tilde{x}_n, \tilde{x}_{n-1}, \ldots, \tilde{x}_1) = \tilde{x}_{j_k}^{N-1-i_k}\tilde{x}_{j_{k-1}}^{N-1-i_{k-1}}\cdots\tilde{x}_{j_1}^{N-1-i_1}$ as shown in [24]. Here, $\theta_{i,t}$ has the following properties: 1) $\theta_{i,t} = 1$ if $i = 0$ or $i_k = i_{k-1} = \cdots = i_1 = N-1$, 2) only those literals present in $\tilde{x}_{j_k}^{i_k}\tilde{x}_{j_{k-1}}^{i_{k-1}}\cdots\tilde{x}_{j_1}^{i_1}$ appear in $\theta_{i,t}$, and 3) those literals with exponents equal to $N-1$ do not appear in $\theta_{i,t}$ at all.

We have the following:

Theorem 5 [3]. *Let $f(x_n, x_{n-1}, \ldots, x_1)$ represent an n variable function in* GF(N). *Its* ith *coefficient corresponding to a term containing k number of variables in the polarity number t is*

$$\kappa_{i,t} = (-1)^k \sum_{x_{j_k}=\delta(0)}^{\delta(N-1)} \sum_{x_{j_{k-1}}=\delta(0)}^{\delta(N-1)} \cdots \sum_{x_{j_1}=\delta(0)}^{\delta(N-1)} \theta_{i,t}(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) f(x_{j_k}, x_{j_{k-1}}, \ldots, x_{j_1}, -\delta(\pi_l), -\delta(\pi_{l-1}), \ldots, -\delta(\pi_1)), \quad (10)$$

1. Note that each subscript of i and t, uniquely identifying a term in the polynomial, can also be represented in terms of vectors of dimension n. Considering i_k, each element in the vector would represent the exponent of the corresponding variable in the term. If a variable is missing from a term, then a 0 is entered in its position, and so on. Similarly, for t_l, each element in the vector would represent the polarity of the corresponding variable in the term.



(a) (b)

Fig. 3. Graph-based representation of 2-bit integer multiplier. (a) MODD. (b) GPDD.

where $\delta(\pi_l), \delta(\pi_{l-1}), \ldots, \delta(\pi_1)$ are the polarities of the variables $x_l, x_{l-1}, \ldots, x_1$ as determined by t.

In $f(x_{j_k}, x_{j_{k-1}}, \ldots, x_{j_1}, -\delta(\pi_l), -\delta(\pi_{l-1}), \ldots, -\delta(\pi_1))$, variables $x_{j_k}, x_{j_{k-1}}, \ldots, x_{j_1}$ are not assigned any values. Therefore, these variables will be called "unassigned" variables. The rest of the variables, that is, those assigned $-\delta(\pi_l), -\delta(\pi_{l-1}), \ldots, -\delta(\pi_1)$ will be called "assigned" variables.

In a reduced MODD, the zero terminal node does not contribute to a canonic polynomial in $\mathrm{GF}(N)$. Therefore, in the rest of the paper, only the ZNMODD will be considered because they provide with a very efficient representation of functions in finite fields with the zero terminal node suppressed.

Let $TV(x)$ denote the value of the terminal node of path x in an MODD. In a ZNMODD, $TV(z)$ corresponds to the $\mathrm{GF}(N)$ product of all the weights along z. In other words, let E denote the set of all the edges constituting the path z. Then, for a ZNMODD, $TV(z) = \prod_{y \in E} weight(y)$, where $weight(y)$ represents the weight associated with edge y.

Let $contrib(x)$ represent the contribution of path x to $\kappa_{i,t}$ having k variables.

Lemma 3 [3]. *If a path z in a ZNMODD contains all the unassigned variables of $\kappa_{i,t}$ and zero or more assigned variables have their desired values, then its contribution to $\kappa_{i,t}$ is*

$$contrib(z) = TV(z) \cdot \prod_{s=1}^{k} (x_{j_s} + \delta(\pi_{j_s}))^{r_s},$$

where $r_s = N - 1 - i_s$.

Lemma 4 [3]. *If, in a path z of a ZNMODD m $(m \geq 1)$ number of unassigned variables are missing and zero or more assigned variables have their desired values, then its contribution to $\kappa_{i,t}$ is 0, that is, $contrib(z) = 0$.*

A Path Oriented Algorithm. Let P denote the set of all the paths in a ZNMODD. Then, we have $\kappa_{i,t} = (-1)^k \sum_{z \in P} contrib(z)$. A path-oriented algorithm appears in Fig. 4. Here, if z does not satisfy Lemma 3, then $contrib(z) = 0$ because either Lemma 4 is satisfied or z does not constitute a term in

$$f(x_{j_k}, x_{j_{k-1}}, \ldots, x_{j_1}, -\delta(\pi_l), -\delta(\pi_{l-1}), \ldots, -\delta(\pi_1)).$$

Example 4. Let us consider a function $f(x_1, x_2)$ in $\mathrm{GF}(3)$ whose ZNMODD appears in Fig. 5 [3]. It contains four paths: ac, ad, bc, and bd. Let us find its coefficients in polarity number 5, that is, x_1 appears in polarity 1, whereas x_2 appears in polarity 2 since $5 = 1 \times 3 + 2 \times 1$. The canonic polynomial is

$$f(x_1, x_2) = 1 + x_{2,2} + \alpha x_{1,1} + \alpha x_{1,1} x_{2,2}. \tag{11}$$

From Theorem 5, $\kappa_{0,5} = f(-1, -\alpha) = f(\alpha, 1)$. From Fig. 5, this corresponds to path bc. Therefore, $TV(\mathrm{bc}) = \alpha \cdot 1 \cdot \alpha = \alpha^2 = 1$. Hence, $\kappa_{0,5} = 1$.

Again, let us calculate $\kappa_{4,5}$. Since $4 = 1 \times 3^1 + 1 \times 1$, this corresponds to the term $x_{1,1}x_{2,2}$. Therefore, $\theta_{4,5}(x_{1,1}, x_{2,2}) = x_{1,1}x_{2,2}$.

$$\begin{aligned}\kappa_{4,5} &= (-1)^2 \sum_{x_1=0}^{\alpha} \sum_{x_2=0}^{\alpha} (x_1 + 1)(x_2 + \alpha) f(x_1, x_2) \\ &= (-1)^2 (contrib(\mathrm{ac}) + contrib(\mathrm{ad}) + contrib(\mathrm{bc}) \\ &\quad + contrib(\mathrm{bd})).\end{aligned}$$

For path ac, both of the unassigned variables are present. Therefore, from Lemma 3,

$$contrib(\mathrm{ac}) = (1+1)(1+\alpha)TV(\mathrm{ac}) = \alpha \cdot 0 \cdot \alpha^3 = 0.$$

```
1  Algorithm CompByPath;
2  {
3      κ_{i,t} = 0;
4      for all paths z ∈ P{
5          If z satisfies Lemma 3, then
6              κ_{i,t} = κ_{i,t} + contrib(z), where contrib(z) is computed from Lemma 3;
7      }
8      κ_{i,t} = (−1)^k · κ_{i,t};
9  }
```

Fig. 4. An efficient path oriented algorithm.

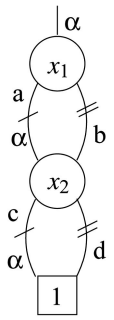


Fig. 5. An example ZNMODD.

Similarly, $contrib(\mathrm{ad}) = \alpha \cdot 1 \cdot \alpha^2 = \alpha$, $contrib(\mathrm{bc}) = 0$, and $contrib(\mathrm{bd}) = 0$. Hence,

$$\kappa_{4,5} = 1 \cdot (0 + \alpha + 0 + 0) = \alpha.$$

Now, let us calculate $\kappa_{1,5}$. This corresponds to the term $x_{2,2}$.

$$\begin{aligned}\kappa_{1,5} &= (-1)^1 \sum_{x_2=0}^{\alpha} (x_2 + \alpha) f(-1, x_2) \\ &= (-1) \sum_{x_2=0}^{\alpha} (x_2 + \alpha) f(\alpha, x_2) \\ &= \alpha(contrib(\mathrm{ac}) + contrib(\mathrm{ad}) + contrib(\mathrm{bc}) \\ &\quad + contrib(\mathrm{bc})).\end{aligned}$$

Paths ac and ad do not contribute. Therefore, their contributions are 0s.

$$contrib(\mathrm{bc}) = 0.\ contrib(\mathrm{bd}) = (\alpha + \alpha) \cdot \alpha = \alpha.$$

Therefore, $\kappa_{1,5} = \alpha(0 + 0 + 0 + \alpha) = 1$.

Again, $\kappa_{3,5}$ is associated with the term $x_{1,1}$.

$$\kappa_{3,5} = (-1)^1 \sum_{x_1=0}^{\alpha} (x_1 + 1) f(x_1, -\alpha) = (-1) \sum_{x_1=0}^{\alpha} (x_1 + 1) f(x_1, 1)$$
$$= \alpha(contrib(\mathrm{ac}) + contrib(\mathrm{ad}) + contrib(\mathrm{bc}) + contrib(\mathrm{bd}))$$
$$= \alpha([(1 + 1)(\alpha \cdot \alpha \cdot \alpha)] + 0 + 0 + 0) = \alpha.$$

It can be shown in a similar manner that the other coefficients equate to 0. Clearly, each of these coefficients verify against (11).

It should be noted that, in polarity number $t = 0$, $f(x_1, x_2) = \alpha x_1 x_2$.

The complexity of Algorithm CompByPath is associated with 1) the number of paths which do not terminate into a 0-terminal node in the MODD or ZNMODD and 2) the number of missing unassigned variables in the path. In the worst case, each path has to be traversed only once, which may render the algorithm exponential in the worst case. However, if there is at least one missing unassigned variable in a path, then that path's contribution is 0. For these types of paths, all of the edges leading to a terminal node need not be traversed. It has been observed that, for most benchmarks, the algorithm found the coefficients quickly. However, for a certain variable ordering, a missing unassigned variable may appear at a level closer to the root than at a different variable ordering of the shared MODD (SMODD). Hence, for a certain variable ordering, a missing unassigned variable can be determined quicker than if a different variable ordering is considered.

This algorithm can be applied in $\mathrm{GF}(2)$ for computing the fixed-polarity Reed-Müller coefficients. In this regard, the technique of [25] seems to be a highly restricted version of the proposed algorithm for $\mathrm{GF}(2)$ only.

Application of this algorithm for determining *all* of the coefficients in higher order fields can render the process slow. A far more efficient technique is proposed in Section 3.2.

3.2 Speeding Up for Large Finite Fields

Let the *partial coefficient* (PC) be defined as

$$\mathcal{PC}_{i,t}(\tilde{x}_{j_k}^{i_k} \tilde{x}_{j_{k-1}}^{i_{k-1}} \cdots \tilde{x}_{j_1}^{i_1}) = \theta_{i,t}(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$$
$$f(x_{j_k}, x_{j_{k-1}}, \ldots, x_{j_1}, -\delta(\pi_l), -\delta(\pi_{l-1}), \ldots, -\delta(\pi_1)).$$

Then, (10) becomes

$$\kappa_{i,t} = (-1)^k \sum_{x_{j_k}=\delta(0)}^{\delta(N-1)} \sum_{x_{j_{k-1}}=\delta(0)}^{\delta(N-1)} \cdots \sum_{x_{j_1}=\delta(0)}^{\delta(N-1)} \mathcal{PC}_{i,t}(\tilde{x}_{j_k}^{i_k} \tilde{x}_{j_{k-1}}^{i_{k-1}} \cdots \tilde{x}_{j_1}^{i_1}). \tag{12}$$

Lemma 5. *Let $\kappa_{i,t}$ and $\kappa_{i',t}$ be the coefficients corresponding to the two terms $\tilde{x}_{j_k}^{i_k} \tilde{x}_{j_{k-1}}^{i_{k-1}} \cdots \tilde{x}_{j_1}^{i_1}$ and $\tilde{x}_{j_k}^{i'_k} \tilde{x}_{j_{k-1}}^{i'_{k-1}} \cdots \tilde{x}_{j_1}^{i'_1}$, respectively. Also, let $i > i'$ and $i_l \geq i'_l$ for $1 \leq l \leq k$. Then, $\kappa_{i',t}$ can be computed from $\kappa_{i,t}$ by multiplying the PCs of $\kappa_{i,t}$ by each value of $\tilde{x}_{j_k}^{i_k - i'_k} \tilde{x}_{j_{k-1}}^{i_{k-1} - i'_{k-1}} \cdots \tilde{x}_{j_1}^{i_1 - i'_1}$ and then adding them together without revisiting the ZNMODD (MODD), that is*

$$\kappa_{i',t} = (-1)^k \sum_{x_{j_k}=\delta(0)}^{\delta(N-1)} \sum_{x_{j_{k-1}}=\delta(0)}^{\delta(N-1)} \cdots \sum_{x_{j_1}=\delta(0)}^{\delta(N-1)} \mathcal{PC}_{i,t}(\tilde{x}_{j_k}^{i_k} \tilde{x}_{j_{k-1}}^{i_{k-1}} \cdots \tilde{x}_{j_1}^{i_1})$$
$$\cdot \tilde{x}_{j_k}^{i_k - i'_k} \tilde{x}_{j_{k-1}}^{i_{k-1} - i'_{k-1}} \cdots \tilde{x}_{j_1}^{i_1 - i'_1}.$$

Proof. This follows from Theorem 5 and the definition of θ.□

Lemma 5 gives us an opportunity to compute coefficients incrementally without having to visit the ZNMODD each time, that is, without resorting to Algorithm CompByPath (Fig. 4) each time.

Let us represent each unassigned variable with 1 and each assigned variable with 0. This results in a set of binary vectors of dimension n. Each position in the binary vector corresponds to whether that particular input variable is assigned or not. The set of all such binary vectors defines a *lattice* over Boolean algebra. The Greatest Lower Bound (GLB) of this lattice is $[00 \cdots 0]$ (that is, all variables assigned), whereas the Least Upper Bound (LUB) is $[11 \cdots 1]$ (that is, all variables unassigned). Clearly, an n-variable function in $\mathrm{GF}(N)$ will have 2^n points in the lattice. For $\mathrm{GF}(2)$, each point in the lattice will correspond to a single coefficient. However, for $\mathrm{GF}(N)$ and $N > 2$, each point will correspond to a range of coefficients. For each 1 in the vector corresponding to a point in the lattice, apart from the GLB, the corresponding variable will have N different exponents in the term ranging from 1 to $N - 1$. Assume that each point in the lattice represents a term having exponents equal to $N - 1$ for all the unassigned variables. If the coefficients of all such terms are known, then the coefficients

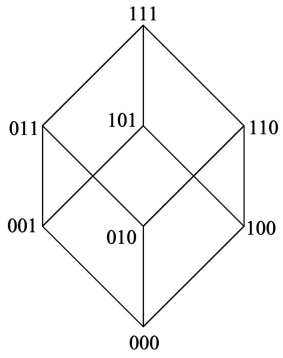


Fig. 6. A Boolean lattice for $n = 3$.

of all of the remaining terms having unassigned variables in the same place can be determined by Lemma 5, without visiting the MODD.

The following example best describes this:

Example 5. Fig. 6 shows an example lattice for a 3-input function in GF(4). The GLB corresponds to the coefficient $\kappa_{0,t}$. The LUB corresponds to the terms $x_{1,\pi_1}x_{2,\pi_2}x_{3,\pi_3}, x_{1,\pi_1}x_{2,\pi_2}x^2_{3,\pi_3}, \ldots, x^3_{1,\pi_1}x^3_{2,\pi_2}x^3_{3,\pi_3}$, that is, the coefficients $\kappa_{21,t}, \kappa_{22,t}, \ldots, \kappa_{64,t}$. Similarly, the point 001 corresponds to the terms x_{3,π_3}, x^2_{3,π_3}, and x^3_{3,π_3}, that is, the coefficients $\kappa_{1,t}$, $\kappa_{2,t}$, and $\kappa_{3,t}$. Considering the point 001, the coefficient for x^2_{3,π_3} can be computed as $\kappa_{2,t} = \sum_{x_3=0}^{\beta}(x_3 + \delta(\pi_3))f(\delta(\pi_1), \delta(\pi_2), x_3)$ (Theorem 5) using Algorithm CompByPath (Fig. 4). Here, $\mathcal{PC}_{2,t} = (x_3 + \delta(\pi_3))f(\delta(\pi_1), \delta(\pi_2), x_3)$. If during this computation $\mathcal{PC}_{2,t}$ is stored for the relevant values of x_3 from the MODD, then the coefficient of x_{3,π_3} can be computed as

$$\kappa_{1,t} = \sum_{x_3=0}^{\beta} \mathcal{PC}_{2,t} \cdot (x_3 + \delta(\pi_3))^{2-1}$$
$$= \sum_{x_3=0}^{\beta} (x_3 + \delta(\pi_3))^2 f(\delta(\pi_1), \delta(\pi_2), x_3)$$

by multiplying the values of $x_3 + \delta(\pi_3)$ by the stored values of $\mathcal{PC}_{2,t}$ and then adding them together without having to revisit the MODD (Lemma 5). In this way, other coefficients with smaller exponents can be determined from those with larger ones by Lemma 5.

A Fast Algorithm. A fast algorithm for computing coefficients can be derived based on Lemma 5. Assume that the exponent of $\kappa_{i,t}$ and that of $\kappa_{i',t}$ differ only in place s and also that $i_s - i'_s = 1$. Then, $\kappa_{i',t}$ can be computed from $\kappa_{i,t}$ by multiplying with the PCs of $\kappa_{i,t}$ the values of $\tilde{x}_{j_s}$ and adding them together if the values of the PCs are stored while computing $\kappa_{i,t}$. Similarly, another coefficient whose exponent differs from $\kappa_{i',t}$ in one place only and by one can be computed, and so on.

An algorithm for computing *all* of the coefficients can be formulated, as shown in Fig. 7. It is assumed that the information about unassigned variables represented by the point of the lattice under consideration is stored while executing Line 10 so that this information can be used to compute the other coefficients in Lines 11-12. One way of implementing Lines 11-12 is shown in Fig. 8.

```
1  Algorithm GfCoeffHs1;
2  {
3      do{
4          Get a point pt in the lattice in ascending order;
5          if(pt is the GLB)
6              Compute by Theorem 5;
7          else{
8              Consider the κ_{i,t} with the highest exponent, i.e.
9              with i_k = i_{k-1} = ··· = i_1 = N − 1 corresponding to the 1s in pt;
10             Compute κ_{i,t} by Algorithm CompByPath;
11             For all i', which differ in one exponent and by one
12                 Compute κ_{i',t} by Lemma 5 from its previous coefficient;
13         }
14     }while(all points in the lattice aren't considered);
15 }
```

Fig. 7. A fast algorithm for large finite fields.

Here, each term is represented by a vector, PosCounter, of exponents and a point on the lattice is stored as another vector, PosVector. However, PosVector only indicates the places in the lattice point, which are nonzeros. HtPc is a hash table, which stores the PCs corresponding to a term. This algorithm generates all the exponents starting from all $N - 1$s to all 1s. However, it also ensures that the exponents are generated in a decreasing order and differing in only one place and by one. Those terms which differ in more than one place are stored in the hash table so that another set of terms can be considered which differ in only one place and by one. It can be argued that the elements are stored in the hash table optimally, that is, it is not possible to store fewer elements on the hash table and still make the algorithm work. The actual coefficients are calculated in Line 21 by Lemma 5. Assuming that the information of each variable, along with the values of the PCs, is stored in Line 10 in Fig. 7, only one pass over the PCs is necessary to compute a new coefficient. Initially, PosCounter has $N - 1$ in those places indicated by PosVector,

```
1  Algorithm CompBy1Diff;
2  {
3      int FirstPos = PosVector[0], branched, i, pos;
4      HtPc[PosCounter] = PartialCoeffs;
5      do{
6          branched = 0;
7          for(i = 0; i < PosVector.size(); i++){
8              pos = PosVector[i];
9              PosCounter[pos]−−;
10             if(PosCounter[pos] < 1){
11                 PosCounter[pos] = radix − 1; // Branch off
12                 branched = 1;
13             }else break; // Continue
14         }
15         if(i < PosVector.size()){
16             if(branched){
17                 PosCounter[pos]++;
18                 PartialCoeffs = HtPc[PosCounter];
19                 PosCounter[pos]−−;
20             }
21             CompByDiff(pos, i);
22             if(PosCounter[FirstPos] == radix − 1)
23                 HtPc[PosCounter] = PartialCoeffs;
24         }
25     }while(i < PosVector.size());
26 }
```

Fig. 8. Incremental computation of coefficients.

for example, for the lattice point 101 in Fig. 6 in $\mathrm{GF}(4)$, $\mathrm{PosVector} = (0, 2)$, and $\mathrm{PosCounter} = (3, 0, 3)$. As the experimental results will show, this algorithm can drastically reduce the computation time.

For the purpose of comparison, another version of the algorithm has been investigated. In this case, instead of considering all of the exponents in such a way that they differ in one place and by one, the coefficients are considered in decreasing order of the exponents. Whenever the least significant exponent is equal to $N - 1$, Algorithm CompByPath is applied to generate it. For the rest of the cases, that is, for the range $N - 2$ to 1 of the least significant exponent, Lemma 5 is applied because the exponents differ in the least significant digit and by one only. This algorithm is called GfCoeffHs. However, it is not presented in detail for brevity as it resembles Algorithm GfCoeffHs1.

Note that techniques such as [39] also use the concept of Boolean lattice for determining coefficients over small fields based on evaluation and decomposition of matrices. However, our technique does not rely on matrices but computes the coefficients using a different approach.

Increased Efficiency for Large Fields. An n-input function in $\mathrm{GF}(2)$ has 2^n lattice points. However, if this same function is encoded in $\mathrm{GF}(2^m)$, that is, at the word level with m-bit word size, then the number of lattice points would come down to $2^{\lceil n/m \rceil}$. This is the total number of times Algorithm CompByPath will be executed to compute all of the coefficients. Hence, with large m, the number of lattice points would be small, implying that the total number of times the Algorithm CompByPath is executed will also be small. However, as m approaches n, the number of terms whose exponents differ by one also increases. This implies that Algorithm CompBy1Diff will also execute more frequently.

4 Experimental Results

The algorithms presented in this paper have been implemented as a part of a tool in C++ (Gnu C++ 3.2.2). This section presents some experimental results carried out on an Intel Celeron machine running at 1 GHz and with 512 Mbytes of RAM. The operating system was RedHat Linux 9 (kernel-2.4.20-30.9). The tool assumes the following PPs by default for generating the fields: (2, 7), (3, 11), (4, 19), (5, 37), (6, 67), (7, 137), and (8, 285). Here, the first number m in the ordered pair (m, d) is the word size and the second number d is the decimal representation of the PP. It is also possible to specify any other word sizes and valid PP as command line options.

Table 1 shows the coefficient count in 0-polarity for small (upper part: $\mathrm{GF}(2)$-$\mathrm{GF}(8)$) and large (lower part: $\mathrm{GF}(16)$-$\mathrm{GF}(64)$) fields, whereas Table 2 shows the corresponding execution times in seconds. Table 3 shows the results in random polarity. Some of the benchmarks are from the International Workshop on Logic Synthesis (IWLS '93) set. Benchmark addm4 is a 4-bit adder/multiplier with a 1-bit multiplexer, whereas gf4adr, gf4mul, and gf4adrmul are adder, multiplier, and adder/multiplier in $\mathrm{GF}(4)$, respectively. $\mathrm{mul}x$ and $\mathrm{adr}x$, for $x = 2 \ldots 6$, are x-bit integer adder and multipliers, respectively.

In Tables 1 and 3, the columns with the label "ncf, scf, and ts" represent the total number of nonzero coefficients (ncf), number of shared coefficients (scf) out of nzcf, and the total number of times they have been shared (ts).

The coefficients are stored as reduced shared GPDD (SGPDD) as they are computed. The complexity of the representations can be reasoned about in terms of the number of nodes and the amount of memory each node takes. Details of the complexity issues can be found in [33]. In this implementation, the graphs were stored as adjacency lists constructed with the help of the vector standard template class library (STL) of C++. Each node has two entries, a flag, and a variable, which stores the reference for deleted nodes. An integer was used to represent the variable number and a vector of integers (called children) of dimension 2^m was used to represent the 2^m children. Here, $\mathrm{children}[i]$ $(0 \leq i < 2^m)$ represents the ith exponent. For example, for mul6 in $\mathrm{GF}(16)$, the SGPDD requires 554 nodes and the SMODD requires 431 nodes or about $(554 + 16) \times (16 + 3) = 10,830$ and $(431 + 16) \times (16 + 3) = 8,493$ words of memory, respectively, assuming that each integer is implemented as a single word. In $\mathrm{GF}(64)$, this figure is 12,797 and 12,462, respectively. Other measures include "com," which is the total number of nodes in the SMODD and SGPDD as a percentage of the total nodes in the multiway decision trees (MDT), where the total nodes in the MDTs corresponding to n-input r-output functions over $\mathrm{GF}(N)$ is $r \cdot \frac{N^{n+1}-1}{N-1}$. "com" indicates the degree of node compression, that is, the lower the value of "com," the higher is the node compression. Another measure is "ratio," which is the ratio of the number of nonterminal to terminal nodes.

Note that the total number of paths in the reduced SGPDD is equal to $\mathrm{ncf} + \mathrm{scf}$ since the shared paths are not counted separately while counting the paths in the reduced SGPDD.

To construct the tables, Algorithms CompByPath (CBP), GfCoeffHs (GFHS), and GfCoeffHs1 (GFHS1) have been applied for each benchmark to compare their performances. In Table 2, their performances appear under the columns CBP, GFHS, and GFHS1. To compare their performances, each execution has been divided into the time spent on computing the coefficient from the MODD (c), by the method of difference incrementally (cbd), amount of time spent on constructing the SGPDD (spdd), and the total time (t) (that is, $\mathrm{t} = \mathrm{c} + \mathrm{cbd} + \mathrm{spdd}$). Note that the symbol "cbd" does not appear under the column CBP because this algorithm relies solely on computing the coefficients from the graph.

The tables are constructed as follows: First, shared MODDs are constructed from the benchmark circuits. Then, the algorithms in Sections 2.1, 3.1, and 3.2 are applied to compute and store all the coefficients exhaustively.

Performance. Tables 1 and 2 provide complete execution profile of the algorithms for the benchmarks. For example, considering benchmark mul5 in $\mathrm{GF}(32)$, the total number of coefficients in 0-polarity is 1,660, out of which 16 of them are shared 16 times. The number of nodes and paths in the SMODD and SGPDD are 63 and 1,760, and 59 and 1,676, respectively. If we use Algorithm CompByPath only, then about 6.11 secs are required, out of which 6.1 secs are spent on actual coefficient computation and 0.01 sec is spent on constructing the reduced SGPDD. However, if we apply Algorithm GfCoeffHs1, then the total execution time reduces to just 0.96 sec, that is, about 6.3 times speed up, and so forth.

TABLE 1
Number of Coefficients in 0-Polarity

Benchmarks	IPOP	GF(2)			GF(4)			GF(8)		
		ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio	ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio	ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio
xor3	3,1	3,0,0	5,4,46,7,2.5	3,3,33,3,1.5						
xor5	5,1	5,0,0	9,16,17,4,4.5	5,5,11,1,2.5						
rd53	5,3	20,0,0	23,35,13,2,11.5	13,20,7,9,6.5						
rd73	7,3	63,0,0	43,147,5,9,21.5	21,63,3,0,10.5						
rd84	8,4	107,0,0	59,294,3,29.5	29,107,1,5,14.5	251,0,0	24,284,4,1,6	39,251,6,3,9.8			
sao2	10,4	1022,696,1376	154,126,1,9,77	158,2396,2,0,79	821,79,79	60,244,2,3,30	116,900,4,4,58			
sqrt8	8,4	128,13,16	42,52,2,2,21	48,144,2,4,24	68,0,0	16,69,2,9,4	28,88,4,7,7			
misex1	8,7	60,46,106	47,37,1,4,23.5	91,166,2,6,45.5	134,45,48	26,53,2,2,6.5	59,182,4,6,14.8			
misex3	14,14	6028,4344,11378	1301,7267,0,3,650.5	1024,17406,0,2,512	20651,10574,15036	878,15109,0,6,219	2823,35687,1,8,705.8			
misex3c	14,14	3116,1794,3239	844,3152,0,2,422	763,6355,0,2,381.5	12484,2402,2641	505,12470,0,3,126.2	1592,15125,1,0,398			
duke2	22,29	7088,5648,17920	976,1632,0,0,488	727,25008,0,0,363.5						
table3	14,14	5509,5320,20300	941,336,0,2,470.5	2272,25809,0,5,1136	27595,15857,24190	743,2743,0,5,185.8	4059,51785,2,7,1014.8			
table5	17,15	74504,74160,277271	873,1708,0,0,436.5	4067,351775,0,1,2033.5						
alu4	14,8	4406,1589,2581	1352,3120,0,5,676	840,6987,0,3,420	18510,3714,4092	1160,23709,1,3,290	2668,22602,3,1,667			
9sym	9,1	210,0,0	33,148,3,4,16.5	27,210,2,8,13.5				249,0,0	11,280,3,2,1.4	32,249,6,8,4
b12	15,9	209,98,120	91,77,0,0,45.5	116,329,0,0,58				4411,107,110	48,3361,0,0,6	251,4521,0,2,31.3
f51m	8,8	56,20,24	70,81,1,8,35	40,80,1,0,20	158,15,15	44,147,3,5,11	49,173,3,9,12.3			
addm4	12,8	1280,320,352	269,314,0,4,134.5	201,1632,0,3,100.5	2018,198,208	146,562,0,7,36.5	210,2226,1,0,52.5			
gf4adr	4,2	4,0,0	6,4,12,7,3.0	4,4,9,8,2.0	2,0,0	5,12,42,9,1.3	2,2,28,6,0.5			
gf4mul	4,2	4,1,1	10,8,19,4.5	7,5,14,5,3.5	1,0,0	4,9,38,1,1	2,1,28,6,0.5			
gf4adrmul	6,2	24,2,2	24,17,10,2,12	13,26,5,9,6.5	7,0,0	14,21,21,2,3.5	5,7,10,6,1.3	41,0,0	9,21,23,3,1.1	7,41,20,5,0.9
mul2	4,4	5,1,1	14,8,12,9,7	10,6,9,7,5	12,0,0	7,12,26,2,3.5	7,12,26,2,3.5			
mul3	6,6	21,6,6	51,43,7,0,25.5	35,27,4,9,17.5				78,2,2	15,77,15,8,3.8	13,80,14,4,3.3
mul4	8,8	97,37,41	157,206,3,9,78.5	114,138,2,8,57	312,28,28	98,346,7,5,24.5	98,340,7,5,24.5			
mul5	10,10	448,182,223	471,917,2,3,232.5	359,671,1,8,179.5	1382,257,267	254,1881,3,8,63.5	404,1649,6,0,101			
mul6	12,12	2046,1052,1491	1348,3943,1,4,674	1228,3538,1,3,614	5831,1106,1175	795,6555,2,4,198.8	954,7006,2,9,238.5	6660,514,520	736,7827,4,0,92	1055,7180,5,7,131.9
adr2	4,3	8,0,0	15,12,18,3,7.5	8,8,10,8,4	13,2,2	8,21,28,6,2	6,15,23,8,1.5			
adr3	6,5	17,0,0	42,38,8,7,21	17,17,3,7,8.5	44,1,1	21,93,14,7,5.3	21,45,14,7,5.3	47,0,0	12,93,13,7,1.5	10,47,12,3,1.3
adr4	8,7	34,0,0	101,108,4,0,50.5	34,34,1,4,17	62,4,4	49,318,5,2,12.3	28,86,3,1,7			
adr5	10,9	67,0,0	224,286,1,8,112	67,67,0,6,33.5	466,11,11	112,1011,2,8,28	140,477,3,5,35			
adr6	12,11	132,0,0	371,1092,0,7,185.5	189,132,0,3,94.5	633,123,123	195,3294,0,9,48.8	211,756,1,0,52.8	1848,0,0	154,4872,1,2,19.3	219,1848,1,6,27.4

Benchmarks	IPOP	GF(16)			GF(32)			GF(64)		
		ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,path,com,ratio	ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio	ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio
rd84	8,4	254,0,0	6,255,8,0,0.1	16,254,11,7,1.0						
sao2	10,4				989,0,0	13,511,4,3,0.4	33,989,6,1,1.0			
sqrt8	8,4	221,0,0	11,165,9,9,0.7	16,221,11,7,1.0						
misex1	8,7	236,1,1	9,121,4,6,0.6	34,237,9,2,2.1						
f51m	8,8	240,0,0	18,255,6,2,1.2	14,240,5,5,0.9						
addm4	12,8	5964,119,119	64,748,0,9,4	84,6083,1,1,5.3				5168,17,17	12,786,0,9,0.2	121,5185,2,2,1.9
mul4	8,8	381,7,7	31,388,8,6,1.9	25,388,7,5,1.6						
mul5	10,10				1660,16,16	63,1760,4,5,1.9	59,1676,4,3,1.8			
mul6	12,12	7483,300,302	431,9900,3,4,26.9	554,7785,4,3,32.6				7038,40,40	127,7536,2,3,2.0	122,7078,2,2,1.9
adr4	8,7	163,2,2	20,381,6,6,1.3	18,165,6,2,1.1						
adr5	10,9				585,0,0	36,1533,3,2,1.1	34,585,3,1,1.0			
adr6	12,11	3181,28,28	67,7800,0,9,4.2	318,3209,3,8,19.9				3881,25,25	129,6048,2,3,2.0	128,3906,2,3,2.0

The experimental results suggest substantial speed up (more than an order of magnitude) for many benchmarks. For example, for benchmarks misex3, misex3c, and so forth, the speed up is close to two orders of magnitude in $GF(4)$. The speedup for that in Table 3 in $GF(4)$ is more than two orders of magnitude. The experimental results also show

TABLE 2
Execution Times in 0-Polarity

Benchmarks	I/P,O/P	GF(2)			GF(4)			GF(8)		
		CBP	GFHS	GFHS1	CBP	GFHS	GFHS1	CBP	GFHS	GFHS1
		c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t	c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t	c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t
xor3	3,1	0,0,0	0,0,0,0	0,0,0,0						
xor5	5,1	0,0,0	0,0,0,0	0,0,0,0						
rd53	5,3	0,0,0	0,0,0,0	0,0,0,0						
rd73	7,3	0,0,0	0,0,0,0	0,0,0,0						
rd84	8,4	0,07,0,01,0,08	0,08,0,0,0,08	0,07,0,0,0,7	0,14,0,02,0,16	0,05,0,01,0,01,0,07	0,0,02,0,01,0,03			
sao2	10,4	0,18,0,08,0,26	0,17,0,0,07,0,24	0,14,0,0,09,0,23	0,6,0,01,0,61	0,22,0,01,0,03,0,26	0,0,02,0,03,0,05			
sqrt8	8,4	0,0,0	0,0,0,0	0,0,0,0	0,03,0,0	0,0,0,0	0,0,0,0			
misex1	8,7	0,01,0,01,0,01	0,03,0,0,02,0,05	0,02,0,0,0,02	0,03,0,0,03	0,0,0,0	0,0,0,0			
misex3	14,14	22,42,1,06,23,48	22,4,0,1,23,4	22,46,0,1,12,23,58	419,17,1,07,420,24	154,73,13,74,1,1,169,57	0,48,22,1,1,09,23,67			
misex3c	14,14	11,7,0,38,12,08	11,68,0,0,36,12,04	11,74,0,0,38,12,12	248,34,0,44,248,78	94,2,8,62,0,46,103,28	0,38,14,43,0,46,15,27			
duke2	22,29	1118,46,2,4,1120,86	1115,87,0,2,6,1118,47	1121,66,0,2,77,1124,43						
table3	14,14	13,92,1,54,15,46	13,78,0,1,63,15,41	14,03,1,49,15,52	128,11,1,56,129,67	48,87,2,48,1,52,52,87	0,24,3,86,1,67,5,77			
table5	17,15	102,48,19,73,122,21	102,52,0,19,72,122,24	102,85,0,19,82,122,67						
alu4	14,8	11,46,0,51,11,97	11,59,0,0,46,12,05	11,47,0,0,5,11,97	513,19,0,63,513,82	194,53,19,7,0,57,214,8	0,56,33,3,0,76,34,62			
9sym	9,1	0,04,0,01,0,05	0,04,0,0,0,01,0,05	0,04,0,0,02,0,06				0,41,0,0,41	0,05,0,04,0,01,0,1	0,0,05,0,0,05
b12	15,9	2,62,0,04,2,66	2,55,0,0,02,2,57	2,75,0,0,06,2,81				108,85,0,12,108,97	21,14,5,39,0,13,26,66	0,05,7,09,0,11,7,25
f51m	8,8	0,01,0,02,0,03	0,02,0,0,0,02	0,01,0,0,03,0,04	0,06,0,01,0,07	0,0,0,02,0,02				
addm4	12,8	0,91,0,1,1,01	0,93,0,0,14,1,07	0,83,0,0,17,1	4,67,0,04,4,71	1,85,0,19,0,06,2,1	0,03,0,29,0,06,0,38			
gf4adr	4,2	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0			
gf4mul	4,2	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0			
gf4adrmul	6,2	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0
adr2	4,3	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0			
adr3	6,5	0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0,0	0,0,0,0			
adr4	8,7	0,02,0,01,0,03	0,02,0,0,0,02	0,01,0,0,0,01	0,17,0,0,17	0,08,0,0,0,08	0,0,01,0,01,0,02	0,0,0	0,0,0,0	0,0,0,0
adr5	10,9	0,16,0,01,0,17	0,16,0,0,0,16	0,16,0,0,01,0,17	1,7,0,02,1,72	0,65,0,12,0,01,0,78	0,65,0,12,0,01,0,78			
adr6	12,11	1,03,0,03,1,06	1,07,0,0,01,1,08	1,03,0,0,02,1,05	19,54,0,03,19,57	6,47,0,9,0,03,7,4	0,07,1,33,0,02,1,42	57,34,0,02,57,36	8,11,5,04,0,05,13,2	0,04,6,11,0,04,6,19
		GF(16)			GF(32)			GF(64)		
Benchmarks	I/P,O/P	CBP	GFHS	GFHS1	CBP	GFHS	GFHS1	CBP	GFHS	GFHS1
		c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t	c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t	c,spdd,t	c,cbd,spdd,t	c,cbd,spdd,t
rd84	8,4	0,18,0,0,18	0,0,05,0,01,0,06	0,0,04,0,04	2,22,0,2,22	0,03,0,24,0,02,0,29	0,0,26,0,01,0,27			
sao2	10,4									
sqrt8	8,4	0,13,0,0,13	0,0,0,0	0,0,0,0						
misex1	8,7	0,11,0,0,11	0,0,02,0,01,0,03	0,0,01,0,0,01						
f51m	8,8	0,18,0,0,18	0,0,02,0,02,0,04	0,0,03,0,01,0,04				12,81,0,05,12,86	0,19,1,62,0,07,1,88	0,1,54,0,09,1,63
addm4	12,8	11,62,0,06,11,68	1,02,1,0,06,2,08	0,01,1,09,0,06,1,16						
mul4	8,8	0,35,0,0,35	0,01,0,06,0,0,07	0,0,05,0,01,0,06						
mul5	10,10				6,1,0,01,6,11	0,16,0,9,0,01,1,07	0,0,94,0,02,0,96			
mul6	12,12	144,04,0,08,144,12	8,82,16,66,0,08,25,56	0,04,17,95,0,1,18,09				112,9,0,11,113,01	1,46,15,97,0,09,17,52	0,02,15,92,0,12,16,06
adr4	8,7	0,31,0,0,31	0,01,0,04,0,02,0,07	0,0,05,0,0,05						
adr5	10,9				5,81,0,01,5,82	0,14,0,82,0,06,1,02	0,0,83,0,01,0,84			
adr6	12,11	113,05,0,03,113,08	7,67,13,49,0,05,21,21	0,02,14,02,0,08,14,12				98,26,0,05,98,31	1,3,12,41,0,09,13,8	0,02,12,7,0,06,12,78

that the time to actually construct the reduced SGPDD is only a fraction of the total execution time and is at most a couple of seconds. As anticipated in Section 3.2, for many benchmarks (for example, misex3, Table 3), the computation time reduces as we go to a higher order field from a lower order field.

TABLE 3
Number of Coefficients in Random Polarity

Benchmarks	IP,OP	GF(2)		GF(4)		GF(8)	
		ncf,scf,ts	SGPDD nodes,paths	ncf,scf,ts	SGPDD nodes,path	ncf,scf,ts	SGPDD nodes,path
xor3	3,1	4,0,0	3,4				
xor5	5,1	5,0,0	5,5				
rd53	5,3	27,3,3	19,30				
rd73	7,3	29,5,5	31,90				
rd84	8,4	53,4,4	51,182	173,0,0	48,173		
sao2	10,4	363,83,106	223,469	805,101,101	126,906		
sqrt8	8,4	55,12,17	46,72	102,0,0	30,102		
misex1	8,7	68,26,49	83,117	133,32,39	59,172		
misex3	14,14	10926,5548,9841	1740,20767	31473,13861,19075	4222,50548		
misex3c	14,14	5758,1502,1973	1194,7731	14312,2266,2448	1664,16760		
duke2	22,29	6770,1179,1901	1014,11591				
table3	14,14	10273,6159,14718	3503,24991	33834,16405,23600	4880,57434		
table5	17,15	35545,13508,22873	3884,58418				
alu4	14,8	6785,1218,1321	1313,8106	18808,3525,3837	2813,22645		
9sym	9,1	173,0,0	32,173			236,0,0	50,236
b12	15,9	148,31,39	117,187			4182,100,100	380,4282
f51m	8,8	88,25,39	62,127	152,17,17	46,169		
addm4	12,8	1225,349,441	276,1666	3072,219,228	474,3300	5464,402,406	597,5870
gf4adr	4,2	3,0,0	4,5	2,0,0	2,2		
gf4mul	4,2	9,2,2	8,11	2,0,0	3,2		
gf4adrmul	6,2	13,2,2	13,15	11,0,0	6,11	37,0,0	7,37
mul2	4,4	12,8,9	13,21	19,1,1	9,20		
mul3	6,6	51,21,31	43,82	74,17,18	38,92	77,13,13	15,90
mul4	8,8	229,138,217	183,446	325,46,47	126,372		
mul5	10,10	904,576,858	574,1762	1546,344,369	441,1915		
mul6	12,12	3765,2702,4508	2296,8273	6571,1742,1927	1248,8498	7342,603,617	1240,7959
adr2	4,3	9,4,4	10,13	15,1,1	7,16		
adr3	6,5	24,9,11	25,35	46,4,4	21,50	50,1,1	10,51
adr4	8,7	79,28,31	59,110	111,4,4	32,115		
adr5	10,9	178,46,48	117,226	513,12,12	157,525		
adr6	12,11	561,26,39	283,600	774,22,23	256,797	1911,45,45	227,1956
Benchmarks	IP,OP	GF(16)		GF(32)		GF(64)	
		ncf,scf,ts	SGPDD nodes,paths	ncf,scf,ts	SGPDD nodes,path	ncf,scf,ts	SGPDD nodes,path
rd84	8,4	246,0,0	17,246				
sao2	10,4			941,0,0	33,941		
sqrt8	8,4	218,0,0	16,218				
misex1	8,7	234,1,1	33,235				
f51m	8,8	241,2,2	14,243				
addm4	12,8	6306,141,141	452,6447			5206,27,27	120,5233
mul4	8,8	398,10,10	30,408				
mul5	10,10			1739,23,23	61,1762		
mul6	12,12	7876,298,301	573,8177			7194,37,37	124,7231
adr4	8,7	185,1,1	19,186				
adr5	10,9			623,0,0	34,623		
adr6	12,11	3360,22,22	326,3382			4068,37,37	129,4105

As compared to Algorithm GfCoeffHs, Algorithm GfCoeffHs1 executes much faster for the majority of the benchmarks. Also, as anticipated, Algorithm GfCoeffHs spends more time on computing from the graph rather than computing incrementally, which is slowing it down.

Word level circuits are constructed from the bit-level circuits as follows: Only those circuits which have non-prime number of inputs are considered in higher order fields. For those circuits which can be considered in higher order fields, the inputs are broken into chunks of k adjacent bit words, where k divides the number of inputs evenly. Hence, parts of the tables are intentionally left blank owing to the way grouping of the bits was done to form words.

Multipliers over GF(2^m). Given α, $\beta \in \mathrm{GF}(2^m)$, if α and β are in their polynomial basis as $\alpha(x) = \sum_{i=0}^{m-1} \alpha_i x^i$ and $\beta(x) = \sum_{i=0}^{m-1} \beta_i x^i$, where $\alpha_i, \beta_i \in \{0, 1\}$, and $0 \leq i < m$, then multiplication over $\mathrm{GF}(2^m)$ can be defined as $w(x) = \alpha(x) \cdot \beta(x) \bmod p(x)$, where $p(x)$ represents the PP used to generate the fields [4], [24], [36]. Multipliers of this form are very important for error control codes and certain public key crypto systems [4]. We have computed the coefficients in 0-polarity for multipliers over the fields $\mathrm{GF}(2^k)$, $2 \leq k \leq 8$, generated with all 51 PPs. The node count in the SMODD grew exponentially with k and, for the majority of the cases, was significantly more than an order of magnitude *higher* than those in the SGPDD. As an example, Table 4 shows the results for 6-bit multipliers over finite fields generated with all six PPs. In $\mathrm{GF}(2)$, only one field is possible, but, in $\mathrm{GF}(2^3)$, there are two PPs possible, $\mathrm{PP} = 11$ and $\mathrm{PP} = 13$, which yield two fields. If the word size is six bits, then only two coefficients are required for each of the six fields over $\mathrm{GF}(2^6)$, which results in an SGPDD with two nodes and one path, whereas the SMODD requires 64 nodes and 3,969 paths. As another example, for the 8-bit multipliers out of the 16 fields, we found minimum values of (nodes, paths) over $\mathrm{GF}(2)$ to be (95, 155) and (95, 152) for the PPs 391 and 451, respectively. This figure is (2,422, 26,240) for the SMODD over $\mathrm{GF}(2)$, that is, well over an order of magnitude higher. The coefficients as (ncf, scf, ts) were (64, 28, 91) and (64, 28, 88), respectively. The maximum (nodes, paths) was observed for the PP 501 to be (120, 174), the coefficient being (64, 28, 110). Again, for the 7-bit multipliers over the 18 fields, the minimum and maximum values of (nodes, paths) were (62, 70) and (89, 131) for the PPs 131 and 191, respectively, over $\mathrm{GF}(2)$. This figure is (1,079, 7,651) for SMODD over $\mathrm{GF}(2)$. The coefficients were (49, 21, 21) and (49, 21, 82), respectively. The nonzero coefficient count was always m^2 over $\mathrm{GF}(2)$. Note that m^2 AND gates are required for bit-parallel realization of these multipliers [4]. Our observation suggests that the minimum node and path counts for the SGPDD were for

TABLE 4
Six-Bit GF Multipliers

PP	GF(2)			GF(8)				
	ncf,scf,ts	SMODD nodes,paths,com,ratio	SGPDD nodes,paths,com,ratio	SMODD nodes,paths,com,ratio	PP = 11		PP = 13	
					ncf,scf,ts	SGPDD nodes,paths,com,ratio	ncf,scf,ts	SGPDD nodes,paths,com,ratio
67	36,15,15	472,2184,1.0,236	47,51,0.1,23.5	564,6860,6.1,70.5	42,12,12	26,54,0.4,3.3	63,0,0	28,63,0.4,3.5
91	36,15,47	473,2184,1.0,236	58,83,0.1,29	563,6860,6.1,70.4	38,0,0	22,38,0.3,2.8	65,2,2	28,67,0.4,3.5
97	36,15,35	474,2184,1.0,236	47,71,0.1,23.5	562,6860,6.1,70.3	64,0,0	28,64,0.4,3.5	48,15,15	26,63,0.4,3.3
103	36,15,38	475,2184,1.0,236	57,74,0.1,28.5	610,6860,6.6,76.3	56,0,0	28,56,0.4,3.5	60,4,4	28,64,0.4,3.5
109	36,15,37	476,2184,1.0,236	59,73,0.1,29.5	564,6860,6.1,70.5	56,6,6	28,62,0.4,3.5	64,5,5	27,69,0.4,3.4
115	36,15,39	477,2184,1.0,236	57,75,0.1,28.5	594,6860,6.4,74.3	59,1,1	28,60,0.4,3.5	51,12,12	26,63,0.4,3.3

TABLE 5
Error Detection in 0-Polarity

Benchmark	GF(4)			GF(8)			GF(16)			GF(32)			GF(64)		
	ncf,scf,ts	n,p	sec	ncf,scf,ts	n,p	sec	ncf,scf,ts	n,p	sec	ncf,scf,ts	n,p	sec	ncf,scf,ts	n,p	sec
mul2*	21,2,2	9,23	0												
mul3*				81,2,2	14,83	0									
mul4*							407,12,12	32,419	0.03						
mul5*										1664,16,16	60,1680	0.94			
mul6*													7795,65,65	129,7860	16.1
adr2*	15,2,2	7,17	0.01												
adr3*				92,5,5	18,97	0.01									
adr4*							378,78,78	28,456	0.06						
adr5*										1349,8,8	50,1357	0.81			
adr6*													6036,37,37	129,6073	13.36

multipliers over $\mathrm{GF}(2^m)$ with primitive trinomials. This figure increased for primitive pentanomials and polynomials with more than five terms and varied according to the positions of the terms within the polynomials. This observation is consistent with the hardware complexity of these multipliers with respect to the number of terms and their positions in the PPs [4]. This information can be very useful for graph-based synthesis of multipliers and exponentiation circuits over $\mathrm{GF}(2^m)$, such as those in [4] and [8].

Verification and Synthesis. Table 5 shows some of the benchmark results from Table 1 in the presence of errors/faults in 0-polarity. The faults have been injected randomly, which include stuck-at-0/1, extra and missing gates (crosspoints), and extra and missing connections. In this table, columns with the label "n, p" represent the number of nodes and paths in the reduced SGPDD, whereas the columns with the label "sec" represent the total time in seconds to compute the coefficients and to construct and minimize the SGPDD. Here, Algorithm GfCoeffHs1 has been used. Clearly, the number of coefficients and the number of nodes/paths in the reduced SGPDD are different from those in Table 1, which indicates errors. The error polynomials can be computed in Tables 1 and 5 by adding/subtracting the polynomials in the designated finite field, which is a simple task once the coefficients and the reduced graphs are constructed. Note that, although the execution time is approximately the same as in Table 1, the program can be terminated early as soon as a mismatch in the coefficient is detected. This can significantly reduce the verification time.

Once the coefficients have been computed, the resulting circuits in the polynomial forms can be synthesized in hardware, for example, based on techniques such as those in [19], [24], [37].

5 Conclusions

This paper presented efficient graph-based techniques for computing and representing coefficients of canonic polynomials over finite fields. Theoretically, the efficiency of the proposed algorithm is associated with the number of paths leading to the nonzero terminal nodes in the graphs. Although this technique can be efficient for computing a single coefficient, it can become time consuming for computing all of the coefficients in large fields. Hence, efficient incremental algorithms have also been presented for large finite fields. The algorithms presented in this paper are implemented as a part of a tool. Experimental results are provided which suggest several orders of magnitude speed gain. These results include multipliers over $\mathrm{GF}(2^k)$, $2 \leq k \leq 8$, generated with all 51 PPs. The results for the multipliers seem to suggest that the complexity of the graphs storing the coefficients is associated with the number of terms and their positions in the PPs with the best cases arising for primitive trinomials.

Acknowledgments

The work of D.K. Pradhan was partially funded by the Engineering and Physical Science Research Council (EPSRC), United Kingdom, under Grant No. GR/S40855/01.

References

[1] A. Dur and J. Grabmeier, "Applying Coding Theory to Sparse Interpolation," *SIAM J. Computing*, vol. 22, no. 4, pp. 695-703, Aug. 1993.

[2] A. Jabir and D. Pradhan, "MODD: A New Decision Diagram and Representation for Multiple Output Binary Functions," *Proc. Design, Automation and Test in Europe (DATE '04)*, pp. 1388-1389, Feb. 2004.

[3] A. Jabir and D. Pradhan, "An Efficient Graph Based Representation of Circuits and Calculation of Their Coefficients in Finite Field," *Proc. Int'l Workshop Logic and Synthesis (IWLS '05)*, pp. 218-225, June 2005.

[4] A. Reyhani-Masoleh and M.A. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $\mathrm{GF}(2^m)$," *IEEE Trans. Computers*, vol. 53, no. 8, pp. 945-959, Aug. 2004.

[5] B. Harking and C. Moraga, "Efficient Derivation of Reed-Müller Expansions in Multiple-Valued Logic Systems," *Proc. 22nd IEEE Int'l Symp. Multiple-Valued Logic (ISMVL '92)*, pp. 436-441, 1992.

[6] R.E. Blahut, *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, 1984.

[7] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers,* vol. 35, no. 8, pp. 677-691, Aug. 1986.
[8] C. Paar, P. Fleischmann, and P. Soria-Rodriquez, "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents," *IEEE Trans. Computers,* vol. 48, no. 10, pp. 1025-1034, Oct. 1999.
[9] C.C. Tsai and M. Marek-Sadowska, "Boolean Matching Using Generalized Reed-Müller Forms," *Proc. Design Automation Conf. (DAC '94),* pp. 339-344, 1994.
[10] C.H. Wu, C.M. Wu, M.D. Sheih, and Y.T. Hwang, "High-Speed, Low-Complexity Systolic Design of Novel Iterative Division Algorithm in $GF(2^m)$," *IEEE Trans. Computers,* vol. 53, no. 3, pp. 375-380, Mar. 2004.
[11] D. Jankovic, R. Stanković, and C. Moraga, "Optimization of $GF(4)$ Expressions Using the Extended Dual Polarity Property," *Proc. Int'l Symp. Multiple Valued Logic (ISMVL '03),* pp. 50-55, May 2003.
[12] D.K. Pradhan and A.M. Patel, "Reed-Müller Like Canonic Forms for Multivalued Functions," *IEEE Trans. Computers,* vol. 24, no. 2, pp. 206-210, Feb. 1975.
[13] D.K. Pradhan, M. Ciesielski, and S. Askar, "Math. Framework for Representing Discrete Functions as Word-Level Polynomials," *Proc. IEEE Int'l High Level Design Validation and Test Workshop (HLDVT '03),* pp. 135-142, Nov. 2003.
[14] D.M. Miller and R. Drechsler, "On the Construction of Multiple-Valued Decision Diagrams," *Proc. 32nd Int'l Symp. Multiple Valued Logic (ISMVL '02),* pp. 245-253, 2002.
[15] D.Y. Grigoriev and M. Karpinski, "Fast Parallel Algorithms for Sparse Multivariate Polynomials over Finite Fields," *SIAM J. Computing,* vol. 19, no. 6, pp. 1059-1063, Dec. 1990.
[16] E.V. Dubrova and J.C. Muzio, "Generalized Reed-Müller Canonical Form for a Multiple-Valued Algebra," *Multiple Valued Logic: An Int'l J.,* pp. 65-84, 1996.
[17] D.H. Green, "Families of Reed-Müller Canonical Forms," *Int'l J. Electronics,* vol. 70, no. 2, pp. 259-279, 1991.
[18] K. Radecka and Z. Zilic, "Design Verification by Test Vectors and Arithmetic Transform Universal Test Set," *IEEE Trans. Computers,* vol. 53, no. 5, pp. 628-640, May 2004.
[19] K.M. Dill, K. Ganguly, R.J. Safranek, and M.A. Perkowski, "A New Zhegalkin Galois Logic," *Proc. Int'l Workshop Applications of the Reed-Müller Expansion in Circuit Design (RM '07),* pp. 247-257, Sept. 1997.
[20] M. Ben-Or and P. Tiwari, "A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation," *Proc. 20th Symp. Theory of Computing,* pp. 301-309, Apr. 1988.
[21] M. Clausen, A. Dress, J. Grebmeier, and M. Karpinski, "On Zero-Testing and Interpolation of k-Sparse Polynomials over Finite Fields," *Theoretical Computer Science,* vol. 84, no. 2, pp. 151-164, Jan. 1991.
[22] M.J. Ciesielski, P. Kalla, Z. Zeng, and B. Rouzeyere, "Taylor Expansion Diagrams: A Compact, Canonical Representation with Applications to Symbolic Verification," *Proc. Design, Automation and Test in Europe,* Mar. 2002.
[23] D.K. Pradhan, "A Multivalued Switching Algebra Based on Finite Fields," *Proc. Int'l Symp. Multiple Valued Logic,* pp. 95-113, May 1974.
[24] D.K. Pradhan, "A Theory of Galois Switching Functions," *IEEE Trans. Computers,* vol. 27, no. 3, pp. 239-249, Mar. 1978.
[25] S. Purwar, "An Efficient Method of Computing Generalized Reed-Müller Expansion from Binary Decision Diagram," *IEEE Trans. Computers,* vol. 40, no. 11, pp. 1298-1301, Nov. 1991.
[26] R. Stanković, T. Sasao, and C. Moraga, "Spectral Transform Decision Diagrams," *Representations of Discrete Functions,* T. Sasao and M. Fujita, eds., pp. 55-92, Kluwer Academic, 1996.
[27] R.E. Bryant and Y.A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," *Proc. Design Automation Conf. (DAC '95),* 1995.
[28] R.S. Stanković and R. Dreschler, "Circuit Design from Kronecker Galois Field Decision Diagrams for Multiple-Valued Functions," *Proc. Int'l Symp. Multiple Valued Logic (ISMVL '97),* pp. 275-280, May 1997.
[29] T. Sasao, "AND-EXOR Expressions and Their Optimization," *Logic Synthesis and Optimization,* T. Sasao, ed., pp. 287-312, Kluwer Academic, 1993.
[30] S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983.
[31] W. Stallings, *Cryptography and Network Security.* Prentice Hall, 1999.
[32] R. Stanković, "Functional Decision Diagrams for Multiple-Valued Functions," *Proc. 25th Int'l Symp. Multiple Valued Logic (ISMVL '95),* pp. 284-289, 1995.
[33] R. Stanković, "Unified Views of Decision Diagrams for Representation of Discrete Functions," *Multiple Valued Logic,* vol. 8, no. 2, pp. 237-282, 2002.
[34] T. Kam, T. Villa, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Multi-Valued Decision Diagrams: Theory and Applications," *Multiple Valued Logic,* vol. 4, nos. 1-2, pp. 9-62, 1998.
[35] T. Sasao and F. Izuhara, "Exact Minimization of FPRMs Using Multi-Terminal EXOR TDDs," *Representations of Discrete Functions,* T. Sasao and M. Fujita, eds., pp. 191-210, Kluwer Academic, 1996.
[36] S.B. Wicker, *Error Control Systems for Digital Communication and Storage.* Prentice Hall, 1995.
[37] Z. Zilic and Z. Vranesic, "Current Mode CMOS Galois Field Circuits," *Proc. 23rd Int'l Symp. Multiple Valued Logic (ISMVL '93),* pp. 245-250, 1993.
[38] Z. Zilic and Z. Vranesic, "A Multiple-Valued Reed-Müller Transform for Incompletely Specified Functions," *IEEE Trans. Computers,* vol. 44, no. 8, pp. 1012-1020, Aug. 1995.
[39] Z. Zilic and Z.G. Vranesic, "A Deterministic Multivariate Interpolation Algorithm for Small Finite Fields," *IEEE Trans. Computers,* vol. 51, no. 9, pp. 1100-1105, Sept. 2002.

Abusaleh M. Jabir received the BSc degree (Honors) in computer science and applied physics and electronics and the MSc degree with distinction, respectively, from the University of Dhaka, Bangladesh, and the DPhil degree in computing from the Computing Laboratory, University of Oxford, United Kingdom, in 2001, where he was with the Hardware Compilation Group, a subdivision of the renowned Programming Research Group. He is currently a senior lecturer in the School of Technology at Oxford Brookes University, United Kingdom. Prior to that, he served as a lecturer in the Department of Computer Science, University of Dhaka, Bangladesh. While completing the DPhil degree, he worked with Celoxica Ltd., United Kingdom, as a senior member of their research staff, prior to starting his career at Oxford Brookes University. His research interests include computer architectures, digital systems design, especially for low-power applications, tests, and verification, efficient hardware design for error control and reliability, and cryptosystems. He is a member of the IEEE.

Dhiraj K. Pradhan is currently a professor in computer science at the University of Bristol, Bristol, United Kingdom. Previously, he was a professor of electrical and computer engineering at Oregon State University, Corvallis, and held the COE endowed chair professorship in computer science at Texas A&M University, College Station, where he also served as the founder of the Laboratory of Computer Systems, and he held a professorship at the University of Massachusetts, Amherst, where he also served as the coordinator of computer engineering. He is also the inventor of two patents, one of which was licensed to Mentor Graphics and Motorola. The recently announced verification tool, Formal Pro, by Mentor Graphics is based on his patent. He has contributed to VLSI computer-aided design and test, as well as to fault-tolerant computing, computer architecture, and parallel processing research, with major publications in journals and conferences, spanning more than 30 years. He is a fellow of the IEEE, the ACM, and the Japan Society of Promotion of Science. He is also the recipient of a Humboldt Prize, Germany. In 1997, he was also awarded the Fulbright-Flad Chair in Computer Science. He received the Best Paper Awards honors including the 1996 *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Ssystem* Best Paper Award, with W. Kunz, on "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems Test, Verification and Optimization." He continues to serve as an editor in prestigious journals, including IEEE transactions. He has also served as the general chair and program chair for various major conferences.