Transforms and Algorithms for Spectral Techniques in Binary and Multiple-Valued Logic

Cicilia Claudia Lozano

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research done by me and has not been submitted for a higher degree to any other University or Institute.

……. …….

Date Cicilia Claudia Lozano

Acknowledgments

First and foremost, I would like to express my deepest gratitude and appreciation for my supervisors Dr. Bogdan Jaroslaw Falkowski and Dr. Susanto Rahardja for the numerous invaluable comments, patient guidance, and kind help that they have given me throughout my study. I have learned so much from them, both on the technical aspect of research and beyond. Their vast knowledge and skills in many areas and their willingness to share them with me have been instrumental in making this research a fruitful and enjoyable journey.

Next, I would like to thank the lab technicians and other postgraduate students of Information System Research Laboratory for making the laboratory a pleasant and conducive environment to do my research. I must also thank my family and friends for their everlasting support and encouragement. Thank you for always being there for me and cheering me up when I was down.

This research is made possible by the financial support from Institute for Infocomm Research. I wish to extend my appreciation to them and to Nanyang Technological University for the various facilities I enjoyed during my research.

Last but not least, I would like to thank God for all the blessings and guidance that He had given me throughout my study.

Summary

Spectral representations of discrete functions have allowed development of powerful tools for many applications in digital logic design and image and signal processing. Through the investigation of their properties, they can be used to effectively solve many problems that are difficult to solve in the original sum of product representation based on the truth table. This thesis focuses on the development of spectral transforms for binary and multiple-valued functions, investigation of their properties, and algorithms for their efficient computation.

Algorithms for calculation and optimization of fixed polarity Reed-Muller expansions for five-valued functions are presented. The algorithms start from different representations of the input functions and due to the calculation steps inside them, the algorithms are suitable for different applications. Computational costs for the algorithms in terms of the required number of arithmetic operations are derived and compared. Computational times for the algorithms are also shown.

New linearly independent transforms for binary functions are introduced. Based on the computational fields, the transforms are classified into linearly independent transforms over Galois field (2) and linearly independent arithmetic transforms. Fast forward and inverse transform matrices for the new transforms are defined. Corresponding butterfly diagram structures for the transforms are also shown. Properties on the structure of the transform matrices and relations between the different matrices are listed and experimental results of the transforms for several MCNC binary benchmark functions are given.

Efficient algorithms for obtaining fixed polarity arithmetic expansions are developed for ternary and quaternary functions. The basic forward and inverse

transforms for the conversion between the truth vector and fixed polarity arithmetic expansions spectra are shown for both cases. Relations between different fixed polarity arithmetic expansions are derived and computational costs for the transformation between them are generated. Based on the relations and computational costs, different algorithms for generation of fixed polarity arithmetic expansions are proposed. The computational steps and computational complexity of each algorithm are found and compared with each other.

Representations of ternary functions with linearly independent basis functions are discussed. Similar to the binary case, the representations are divided into ternary linearly independent expansions and linearly independent ternary expansions depending on whether they use GF(3) arithmetic operations or standard arithmetic operations, respectively. The corresponding new fast forward and inverse transforms are proposed. The properties and relations between the different linearly independent expansions are investigated. The transforms are implemented in MATLAB and the numbers of nonzero spectral coefficients for several ternary test files based on the proposed transforms are shown.

The hardware computations and implementations of some of the discussed transforms are shown. Both the binary and ternary linearly independent transforms possess fast transform properties and so the transformation between the truth vector and their spectra for the input function can be implemented using linear systolic processor array. The linear systolic processor array structures for the conversions are discussed. Some of the linearly independent transforms also have regular structure that allows modular implementation of their resulting polynomial expansions. The basic modules for such hardware implementations are given. The synthesis of the hardware implementation for an n-variable function from the corresponding $n-1$ variable modules is also described.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Review of spectral techniques

In digital logic design, the behavior of a device is formulated as a function of its input variables. Complete definition about the function is traditionally given in the form of truth table, where each entry in it provides the function value for a specific combination of input variables' values. Since it was proven by Shannon [Sha38] that Boolean domain developed originally by Boole [Boo54] provides a precise model for the analysis of switching circuits, the majority of existing methods in digital logic design are concerned with the properties of Boolean functions. Many problems in digital logic design and testing, artificial intelligence, and combinatorics can be expressed as a sequence of operations on Boolean functions.

However, one of the problem with the definition of a function in the Boolean domain (in term of minterms) is that each individual minterm gives information on the behavior of the function at a single point (that is, for a particular combination of the input values) but not about the behavior of the function anywhere else (for any other combination of input values). To get a complete definition of the function the behavior of the function for all possible combination of input values (2^n points for two-valued function) must be known. Spectral representation is an alternative way to represent a function where the information about the function, given in the

form of spectral coefficients, is much more global in nature. In spectral domain, each spectral coefficient of an *n*-variable function contains some information about the behavior of the function at a subset or all of the points, but does not necessarily contain complete information about any of them. The combination of all the values in the spectrum does lead to complete information about the function, but each individual coefficient gives us some global information about the whole function since in general they are affected by the values of the function at more than one points. Thus, spectral domain provides additional insight into the functional structure of combinational circuits. It has been shown that these alternative representations are useful in design and testing of digital devices. Their usefulness stem from the facts that a number of properties, such as decomposability, symmetry, tautology, and satisfiability, can be more easily observed in spectral domain than in operational domain. Similarly, some operations, e.g., equivalence cheking, can be more easily executed in spectral domain [Dam92, DK89, DM00, Edw75, EM86, FR02, Gre86, Hei91, HMM85, Hur78, Hur89, Hur92, Kar76, Kar85, KB81, Llo80, Mal97, Mil87, PM75, Red72, RZ04, SSJ98, TDM01, Tok80, VT88]. These properties allow some problems to be more easily solved when they are transferred from the operational domain to spectral domain.

Conversion from Boolean domain representation to equivalent spectral domain representation is usually done through special transforms called spectral transforms. Spectral transforms redistribute the information contained in the truth table (operational domain) by converting the sum of product representation of the function based on the truth table entries into alternative representations based on some other sets of basis functions that are chosen to more closely resemble the circuit characteristics [AS06, DDT78, HMM85, Hur78, Kar76, Kar85, SSJ98, TDM01]. In this regard, the value of each spectral coefficient indicates the correlation between the function being represented and the basis function that corresponds to the coefficient. Spectral transforms need to have an inverse to ensure that it is possible to obtain the Boolean domain representation back from spectral domain representation without any loss of information. Generally, spectral transforms are orthogonal matrices by which a spectrum can be obtained from the truth vector representation of a Boolean function through matrix multiplication. The spectrum of a function itself may be represented

either in original or reduced spectral domain as spectral vectors, spectral expansions, spectral cubes, spectral trees, or spectral decision diagrams.

Spectral techniques have been playing important roles in digital logic design for many years. In spectral techniques, the original set of data is first transformed into spectral domain, desired properties or operation are extracted or performed on the spectral coefficients, and the results are transformed back into the original domain. The techniques take advantage of the fact that operations in the spectral domain may be simpler than equivalent operations in the original domain. Early works on spectral transforms include those by Rademacher [Rad22] and Walsh [Wal23], followed by study on Hadamard matrices [Wal72] and orthogonal functions [Pal32, Pal33]. Currently there are many spectral transforms that are used in digital logic design. Each of them has its own advantages and disadvantages, and hence its own applications. Some important ones include various versions of Reed-Muller (RM) transform [AAE95, AB96, Abo01, AS06, Alm94, CW83, Dam92, DDT78, DK89, DM00, DTB96, Fal02, FHH+93, FP91, FR95, FR97, FY04, GE78, Gre86, Gre89, Gre90, GT76, Har90, HM92, Hur78, Hur92, JSD02, JSM03, KPH99, KS75, KSR92, KSY91, KSZ90, LM98, PDF90, Pur91, Red72, RF01, RP88, SA03, Sas93, Sas99, SF96, SSJ98, SP92, TDM01, TH79, Tra87, VP01 VT89, VT91, Yan98], arithmetic transform [AS06, Che92, Fal99, Fal02, FC94, FC97, FC99, FP90, FSP92, Hei91, Hur92, Kar85, KB81, KSY91, KSZ90, PM75, RZ04, SA03, SF96, SSJ98, TDM01, VP01, Yan94, Yan98, YSF01], Rademacher-Walsh transform, and Haar transform [AAE95, AS06, Bea84, Cas96, Edw75, Fal02a, FSP92a, HMM85, Hur78, Hur89, Hur92, JSD01, Kar76, Kar85, KSY91, KSZ90, LM98, Mil87, Moh92, SA03, SF96, SSJ98, TDM01, VT88, VT89, Yar85].

Spectral techniques have been applied for various tasks in digital logic design, such as Boolean function classification [Edw75, Hur78, HMM85, MMH82, Muz80, VP01], disjoint decomposition [HMM85, Tok80, Tra87, TV87, VT89], parallel and serial linear decomposition [EM86, Hur78, HMM85, Hur89, Kar76, Kar85, KSA00, Tra87, TV87, VP01, VT89], spectral translation synthesis [EM86, Hur78, Hur89, Kar76, Tra87, TV87], multiplexer synthesis [HMM85, Llo80], prime implicant extraction by spectral summation [HMM85, Kar85, Lec71], threshold logic synthesis

[Edw75, Hur78, Kar76], logic complexity [Kar76, VP01], state assignment [Kar76, VT88], filtering probable matches [Kar85], probabilistic analysis of Boolean expressions [Che92, Hei91, KB81, PM75, VP01], testing and verification [Che92, Dam92, DK89, DM00, Gre86, Hei91, HMM85, Hur89, Hur92, Kar85, KSA00, Mil87, RF99, RZ04], and low power VLSI CAD design [TDM01]. Other than digital logic design field, the spectral transforms also have implementations in the area of signal and image processing [Bea84, Cas96, Fal04, Kar85, RP88, Yar85], communication [LM98, RSL03], and digital filtering [AAE95]. Two design automation systems have incorporated spectral techniques for designing digital circuits. They are SPECSYS [VT89] developed at Drexel University and DIADES [FSP92a] from Portland State University. Relations between the different spectral representations have been investigated [TDM01].

In the beginning, digital logic design is mainly concerned with the properties and operations of binary logic functions related to Boolean functions [VP01]. However as the technology advanced to Ultra Large Scale Integration (ULSI) and the number of gates per integrated circuit (IC) increased, there is an increasing research interest in multiple-valued logic. This interest is motivated by the potential of multiple-valued logic for solving the interconnection problem faced by the current ULSI technology. As the technology matures, the number of gates that can be packed onto an IC grows faster than the rise in the package pin count. This discrepancy leads to the increasingly important problems of pin count limitation and the need for major reduction in interconnections between active devices inside and outside of an IC [Wol98]. One possible way to overcome these problems is by using multiple-valued elements and interconnections [Rin84]. Multiple-valued interconnections are capable of transferring more information than binary interconnections. On the other hand, multiple-valued logic memories and registers can store larger volumes of data than two-valued memory and registers. Thus, fewer interconnections per logic function realization could be achieved when using multiple-valued logic design instead of binary logic. Also, larger logic functions could be implemented for a fixed number of interconnections. As the interconnections are sources of wiring errors and weak connections, the fewer number of interconnections also improves the reliability of the

function realization. New technologies [ZV93], improved Computer-Aided Design (CAD) tools, and process controls have made possible the development of multiple-valued logic.

The research direction in multiple-valued logic is twofold: first is to apply multiple-valued logic to improve the analysis and synthesis of binary circuit; and second is to develop the mathematical foundations and circuit technologies for multiple-valued circuits. Many applications of the multiple-valued logic have been proposed. In [Rin84], hazard elimination in binary network by multiple-valued gates was discussed. Multiple-valued logic has also been applied for optimization of programmable logic arrays (PLAs) [Sas88]. A multiple-valued encoding scheme for low power asynchronous communication has been proposed in [TH04]. A 32×32 bit multiplier using multiple-valued MOS current-mode circuit with smaller size and comparable speed than the binary counterpart has also been reported [KKH+88]. In [MTH04], a common bus architecture employing four-valued encoding has been developed and compared to binary DMA bus architecture. A CAD system for multiple-valued logic design called MV-SIS has also been developed [BGJ+02]. In particular, multiple-valued memory technologies have been proposed and implemented, such as StrataFlash memory technology by Intel.

In the past, although many interesting results, algorithms, and techniques were discussed that are related to the application of spectral methods to digital design problems, their use was greatly limited by the lack of techniques for computing and manipulating the spectra of large functions encountered in practical industrial design situations since the existing methods typically incurred exponential time and space complexities [TDM01]. However, this problem has become less significant with the introduction of Binary Decision Diagram (BDD) [AS06, Bry86, Bry91, SA03, Sas93, Sas99, SF96, SS98, TDM01]. BDD is a graphical representation of a logic function that offers more compact representation than other methods for many practical functions. BDD in which the variable ordering in the expansion is fixed, redundant nodes deleted, and isomorphic subgraphs are shared is called Reduced Ordered Binary Decision Diagram (ROBDD). ROBDDs are canonical and they allow efficient representation and manipulation of many functions encountered in practical

applications. Because of these reasons, ROBDDs are widely used in many applications, such as logic design verification, logic synthesis, test generation, and fault simulation. ROBDDs are very sensitive to the variable ordering, i.e., different choice of variable ordering may lead to vastly different size of the resulting ROBDDs. Thus, there are many researches done on their minimization. Proposed minimization techniques include heuristic variable reordering and sifting.

The wide application of ROBDDs has led to the development of other forms of BDDs and other types of decision diagrams [AS06, Bry86, Bry91, PFC+02, SA03, Sas93, SF96, SKM+90, SS98, SSJ98, Sta95, TDM01, YMS+06]. A class of function that has exponential size when represented by one decision diagram may have linear size when represented by another decision diagram. The developed decision diagrams differ in the decomposition rules applied at each node, the permitted values of the terminal or constant nodes, as well as in the employed reduction rules. Decision diagrams in which the constant nodes are logic values 0 and 1, such as BDD and Functional Decision Diagram (FDD), are called bit level decision diagrams. Decision diagrams in which the constant nodes are element of finite fields, integers, or complex numbers, such as Multi Terminal Binary Decision Diagrams (MTBDDs), are called word level decision diagrams. Classifications of the existing decision diagrams have been presented in [Al-04] and [SS98]. In particular, spectral decision diagrams can be used for computing and representing various spectra of logic functions. They are especially useful for representing and executing operations on large functions.

Similar to the binary case, spectral techniques have also been proven to be useful in multiple-valued logic design. In order to accommodate the multiple-valued nature of the multiple-valued functions, many binary spectral transforms have been generalized and extended to multiple-valued cases [Al-04, AS06, CW83, Dam92, DDT78, DM00, Eps93, FF03, FF04, FF05, FF05a, FF05b, FF05c, FF06, FR01, GE78, Gre86, Gre89, Gre90, GT74, GT76, HM92, Kar85, KPH99, KSZ90, Per92, PSB95, Rin84, SSJ98, Yan94, Yan98]. Especially for RM transform, it has been shown that the advantages in term of area, speed, and easy testability possessed by the binary RM transform are preserved in multiple-valued case [Dam92, DM00, KPH99, Sas93, SF96]. Various efficient algorithms for computing spectra of multiple-valued

functions have been proposed [CW83, FHH+93, FR95, GE78, Gre89, Gre90, HM92, JSD02, JSM02, JSM03, KSZ90, RF01, SA03, SF96, SKM+90, Sta95, TDM01, TH79, Yan94, Yan98]. Different reduced representations of multiple-valued functions and operations on them have also been developed [AS06, SA03, Sas93, SF96, SKM+90, Sta95, TDM01, Yan94, Yan98]. These have increased the practical value and feasibility of multiple-valued spectral techniques. Combined with the increasing importance of multiple-valued logic design and the fact that multiple-valued spectral techniques have potential applications in areas where binary spectral techniques have been applied successfully, further research of spectral techniques in multiple-valued logic design is an important and interesting one. Application of multiple-valued spectral techniques for image processing has been proposed in [Fal04]. Linearization, planarization, optimization, and reduction of interconnections with the help of multiple-valued linear models based on arithmetic transform have also been introduced [YFS01].

1.1.1 Reed-Muller transform

Reed-Muller (RM) is a class of AND-EXOR expressions where the basis switching functions are all conjunctions of the function literals. It directly corresponds to the AND-EXOR two level logic network circuit realizations. In initial RM transform known as complement-free ring-sum [DDT78, Gre86, Zhe27, Zhe28], all of the literals appear in positive (not complimented) form throughout the function polynomial expansion. This type of expansion has since been expanded into Generalized Reed Muller (GRM) expansion, where the literals in the expansion may also appear in complimented form. When all of the literals in the GRM expansion appear consistently in not complimented or complimented form throughout the expansion, the expansion is known as fixed polarity Reed Muller expansion (FPRME). Otherwise, the expansion is called mixed polarity Reed-Muller expansion (MPRME). Expressing a binary function by its MPRMEs may yield smaller circuit realizations. However, as they allow the input variables to be represented by more than one literals, the circuit realizations based on MPRMEs require more inputs than the FPRME circuit realizations. In the literature, the binary RM transform is also known as conjunctive

transform [AAE95] and adding transform [FP90]. In addition, as the elements of the RM matrix can be easily derived using binary Pascal triangle, the matrix is also called Pascal transform [AAE95].

As every variable in FPRME may appear in one of two forms, either complimented or not complimented, there are 2^n possible FPRMEs for a binary function. Each of the expansion is canonical (unique) and is identified by its polarity number ω. The polarity number is obtained by assigning bit 1 or 0 to every variable according to whether it appears in complimented or not complimented form throughout the expansion and taking the decimal equivalent of the resulting binary number. Hence, the complement-free ring-sum expression is none other than the FPRME with polarity number zero. The expansion is also often referred to as the positive polarity RM expansion. FPRMEs with different polarity numbers may have different computational complexity, which is often measured by the number of nonzero spectral coefficients or the number of literals in the FPRME. As the number of nonzero spectral coefficients corresponds to the number of gates needed in the circuit realization while the number of literals corresponds to the number of inputs to the circuit, it is desirable to minimize the numbers. The smaller the numbers, the more efficient is the final hardware implementation based on the expansion. The polarity number for which the number, whichever is used as computational complexity measure, is smallest is called as the optimal polarity. Many researches have been done to find the solution to the problem of finding the optimal polarity for a function.

In earlier technologies, EXOR-based implementation, which is used by RM transform, was not popular because EXOR gates were much larger and slower than AND/NAND/OR/NOR blocks. However, this problem is no longer significant with the introduction of new technologies which allow EXOR-based functions to be implemented efficiently at the same cost as other inclusive gates, such as Signetics LHS501, Xilinx 3000, Actel 1010/1020, Concurrent Logic Cli6000, and other PLDs with EXOR gates [SP92]. This development, together with the findings that RM based implementations are advantageous in terms of area, speed and testability [Red72, Sas93, Sas99, SF96], have resulted in renewed interest in RM transforms. In [Sas99] and [SF96] it was shown that AND-EXOR expressions require on the average fewer

products than the sum of products expression. PLAs with EXOR arrays also often require fewer products than PLAs with OR arrays [SB90].

It is well-known that RM expansions have high testability properties. This stems from the characteristic of the EXOR gate that any change in its input is reflected on the output. It has been found that when a circuit is represented as an RM expansion, if only permanent stuck-at faults occur in either a single AND gate or only a single EXOR gate is in fault, only $(n + 4)$ tests are required for fault-free primary inputs for an arbitrary n-variable Boolean function [Red72]. The testability of binary FPRME networks has also been considered in [DK89], where it was shown that at most n FPRME spectral coefficients need to be verified to detect multiple stuck-at faults and single bridging faults at the input lines of the networks, where n is the number of the inputs. It was noted that the fault detection techniques by verification of RM spectral coefficients have smaller time and hardware complexity compared to those based on Walsh spectral coefficients [DK89]. Many functions can be best realized by their RM expansions, especially those functions that have high content of linear part (EXOR part of the function that does not include any product terms of the primary input variables). Arithmetic functions such as adders, multipliers, counters, and parity checkers are examples of such functions. A linear function can be implemented using only modulo-2 adders with optional scalar or constant multiplier synthesized as an open or close connection (to represent 0 or 1, respectively).

RM transform has been used for image processing [AAE95, Fal04, RP88], fault detection [DK89, Sas99], and coding techniques, especially those concerned with group or block codes for error control [Gre86, Kar85]. It has also been used in developing good and compact models for decision diagrams [AS06, DTB96, KSR92, SA03, SF96, SS98, Sta95]. Many methods exist that can be used to calculate the FPRMEs for a given function from different representation of the function [Abo01, DTB96, Fis74, FP91, FR97, FY04, Har90, Pur91, Sas93, SF96, SP92, SSJ98, Tra87, VT91]. In [Alm94], a map based simplification of FPRME transform has been presented which can be used to convert minterms to FPRME coefficients of binary functions with up to six variables and vice versa, and thus select the optimal FPRME for them. Hardware based translation between the truth vector and FPRME has also

been proposed [AB96]. Other algorithms have also been developed that obtain the optimal FPRME representation from the reduced representations of the input functions, such as graphs [CYP89], trees [Abo01], decision diagrams [DTB96, KSR92, Pur91], and array of cubes [FP91, SP92, VT91]. A non exhaustive approach for finding optimal FPRME of a three-variable binary function from its Walsh-Hadamard spectra has also been presented in [FY04].

The RM transform and its spectral coefficients calculations have been extended to multiple-valued case. Many attentions have especially been given to the ternary case (GF(3)) [FHH+93, Gre86, Gre89, TH79], which is the smallest multiple-valued field, and the RM expansions over GF(2^m) such as quaternary case [FR95, Gre90, JSM03, RF01]. The extension of RM expansions to any prime field GF(q) have also been proposed [DDT78, GE78, Gre86, GT74, GT76, HM92, JSD02, KS75]. Testability of multiple-valued circuit realizing multiple-valued FPRMEs have been investigated in [Dam90] and [DM96], where it has been found that the easy testability property possessed by the binary FPRME networks is maintained in the multiple-valued FPRME. In particular, in the latter it has been deduced that only four tests are required to detect all single stuck-at faults on internal lines in a circuit realizing multiple-valued FPRME with no complimented literals. Multiple-valued functions offer more flexibility and can be more compact when compared with binary functions. However, the number of FPRMEs to be considered increases more rapidly in multiple-valued case than in binary case as the number of input variables rises. As a result, optimization problem of multiple-valued FPRMEs is more complex than its binary counterpart. Many algorithms have been developed for efficient calculation of FPRME spectra from the truth vector representation of multiple-valued functions [FHH+93, FR95, Gre89, Gre90, HM92, RF01]. For larger functions, efficient calculation and manipulation algorithms that work on the reduced representations of multiple-valued functions, such as disjoint cubes, multiple-place decision diagrams, and FDDs also exist [JSD02, JSM03, SA03, Sas96, SKM+90, Sta95]. Universal logic modules for FPRMEs of ternary function have also been proposed [CW83, TH79].

1.1.2 Arithmetic transform

Arithmetic transform, which is also known as probability transform [KB81, PM75], algebraic transform, inverse conjunctive transform [AAE95], and the inverse integer Reed-Muller transform, was initially introduced in [KB81] for probabilistic analysis of Boolean functions. The forward arithmetic transform is obtained by calculating the inverse of the inverse RM transform in arithmetic domain. It has been found that it is often useful to convert a function to standard arithmetic function, especially when probabilistic or stochastic analysis of Boolean function is required [Che92, KB81, Mil87, PM75, TDM01]. Arithmetic expansions are used for efficient representation of multi-output functions and they can be represented by word-level decision diagrams. The binary function operated by the arithmetic transform can be defined with two types of coding: R and S coding. Relations between the arithmetic spectra in different coding can be derived as presented in [FP90].

Similar to the RM transform, initially arithmetic transform expansion does not allow literals to be complimented (called 'polarity zero' expansion). Links between RM and arithmetic transforms when all the literals appear in affirmative form were presented in [KB81, Mil87]. In contrast, generalized arithmetic transform [Fal99, Fal02, FC94, FC99, FP90, FSP92, YMS$^+$06] allows all literals to be both in affirmative and complimented form, which gives rise to 2^n canonical arithmetic expansions for a binary function where each of them may have different number of nonzero spectral coefficients. The expansion with the smallest number of nonzero spectral coefficients are said to be the optimal arithmetic expansion. Value of expansions having fewer nonzero spectral coefficients can be calculated in shorter time. Such property is important in applications where the value of the expansion needs to be computed repeatedly for different assignment of input variables, for example in probabilistic verification of switching functions. Smaller number of nonzero terms also leads to more efficient circuit realizations.

Arithmetic transform has found applications in various tasks. They include synthesis, verification, library matching of CAD, as well as probabilistic Boolean analysis and testing [FR02, Hei91, Hur92, KSY91, KSZ90, Mal97, Mil87, PC75,

RZ04, SF96]. It has also been used in artificial intelligence [Hur75] and reliability of engineering system as well as random testing of digital circuits [Hei91, PM75]. Linearization, planarization, optimization and reduction of interconnections with the help of multiple-valued linear models based on arithmetic transform have also been introduced [YSF01]. The article [Fal99] presented a literature review of arithmetic transform development in Eastern Europe.

Calculation methods of generalized arithmetic transform spectral coefficients had been developed that operate on many different representations of a function. The methods in [FC94, FC97] can be used to calculate generalized arithmetic transform spectral coefficients from various binary decision diagrams that permits manipulation and calculation of functions with large number of variables while the methods presented in [Fal02, FSP92] work on truth vector representation of a function. If the function is given as an array of cubes, then the algorithms to calculate the spectral coefficients from disjoint cubes reduced representation [FC99] are the suitable methods to use. Such methods are also well suited to applications that require calculations of only some spectral coefficients [FSP92a, Hei91, Hur92, Kar76, TDM01].

1.1.3 Linearly independent and linearly independent arithmetic transforms

Binary linearly independent (LI) logic generalizes all possible expansions of binary logic circuits that are realized in GF(2) algebra. The set of basis switching functions in the LI expansion is composed by any set of switching functions as long as the resulting transform matrix is nonsingular. Their basic concepts have been discussed in [Per93, PSB95] and their extension to multiple-valued input case can be found in [Per92, PSB95]. Furthermore, their various properties and special cases have been presented in [FR97a, FR01, PDF90, Per92, Per93, PFC$^+$02, PSB95]. Binary LI logic provides any binary function with canonical representations that can be implemented

as two and multi-level AND/EXOR circuits which are superior in terms of the number of gates, speed, area, and testability. The canonical representations are obtained by repetitive expansions of the logic functions. The well-known RM transform is only a special case of LI transform. Generally, LI logic circuits realizations are smaller and never worse than RM based circuits since they include all AND/EXOR circuits covered by RM transform and they operate in a much larger design space. These properties have allowed development of new types of decision diagrams [PFC+02] and a comprehensive approach to logic synthesis and physical design for two-dimensional logic arrays [SSC+94]. The resulting LI expansion can be readily implemented by using existing 20x8 PAL device or Cypress 330 [FR97a].

There is a large number of all possible LI transforms over GF(2) and calculating all representations of a particular binary function based on them will require extensive resources. Hence, finding efficient ways for generating the expansions is of great importance. In [FR97a], the classes of binary LI transforms that possess fast forward and inverse butterfly diagrams were identified and their recursive definitions were given together with the theory that allows one to find and calculate optimal LI expansions by using fast transforms. Similar classification and fast transform method for multiple-valued input LI logic have also been developed in [FR01]. Even though the fast transform method [FR97a] is able to find single expansion for some polarities of variables, there is still the problem of finding the best expansion in some polarity among all polarities of two-variable expansion. This problem is addressed in [PFC+02] where basic principles of efficient algorithms for selecting the best expansion polarity were given. In addition, multi-level structures of higher regularity for AND/OR/EXOR bases have also been proposed in [PFC+02] which can be mapped to Motorola, ATMEL, and Xilinx fine grain FPGAs. Obviously, different choice of basis functions results in different implementation complexity. Proper selection of the basis functions will greatly increase the efficiency of the final implementation of the circuit, in term of area, speed, and testability.

In addition to the LI transforms identified in [F97a], another four LI transforms which can be computed using fast transform have been introduced in [RF02]. Two of these matrices belong to class A1 whereas the other two are categorized as class A2.

All these four transforms require fewer number of GF(2) additions than all the LI transforms presented in [FR97a], including FPRME transform which was previously known as the fastest transform in GF(2). Hence, they are the fastest and most efficient LI transforms in terms of their GF(2) computational complexity. Properties of polynomial expansions based on two of the fastest LI transforms had been discussed in [FL03a] together with the comparison of their experimental results with those of FPRME transform for several binary benchmark functions. The comparison showed that the fastest LI transforms offer less number of nonzero spectral coefficients than the FPRME transform for some cases. Therefore, it is of interest to identify other LI transforms that have the same computational complexity as the fastest LI transforms and analyze whether they are able to give better representations than the four fastest LI transforms [RF02].

It has been known that for some functions, it is more advantageous to implement the function in arithmetic domain rather than as an EXOR based polynomial expansion. Such an approach is especially useful for testing of properties of binary functions, solving systems of binary equations and analyzing their stochastic properties [Che92, TDM01]. The broadest approach in further generalization of arithmetic expansions is to allow one to formulate such an expansion in terms of standard addition and subtraction operating on an arbitrary logic function or logic expression. It is obvious that, in the special case of the truth vector for the AND function, such an expansion becomes mixed arithmetic where each variable can have an arbitrary polarity throughout all terms in the expansion or generalized arithmetic expansion. In the majority of cases, the new expansion has its own properties and may be advantageous in different applications of arithmetic-type expansions in logic design. The concept of such generalization known as linearly independent arithmetic (LIA) transforms have been introduced [RF99a]. In [RF99a], the classes of binary LIA transforms that possess fast forward and inverse butterfly diagrams are identified and their recursive definitions are given together with the theory that allows one to find and calculate optimal LIA expansions by using fast transforms. Arithmetic transform falls into broad definition of LIA logic. Comparisons among various LIA transforms show that for many classes of logical functions some LIA transforms outperform

arithmetic transform when applied to fault detection [RF99]. In [FR02], another two recursive LIA transforms that possess fast transform properties have been presented that possess smaller computational cost than the arithmetic transform and all other LIA transforms introduced in [RF99a]. These transforms are called fastest LIA transforms and their butterfly diagram and recursive definitions as well as their implementation for Boolean verification are shown in [FR02]. Their properties and experimental results were analyzed and presented in [FL03], where it has been shown that for some binary benchmark functions the pair of LIA transforms is more advantageous than arithmetic transform in terms of number of nonzero spectral coefficients.

The concept of LI and LIA has been extended for multiple-valued cases, i.e., ternary and quaternary. In [FF05] and [FF04], the family of fast LI transforms for representations of ternary functions have been presented that operate over GF(3) and standard arithmetic, respectively. The classes of such transforms that have the smallest computational costs were identified in [FF03], [FF05c], and [FF06]. On the other hand, the articles [FF05a] and [FF05b] presented the fast LI transforms for analysis and synthesis of quaternary functions.

1.2 Objectives and contribution of the thesis

The objectives of this thesis are to develop and analyze new spectral transforms for binary and multiple-valued functions as well as to derive efficient algorithms for obtaining spectral representations of discrete functions.

To that end, several algorithms for efficient generation of FPRMEs over GF(5) are proposed. There are 5^n possible FPRMEs for an n-variable five-valued function. Most often, it is required to find the optimal FPRME to obtain the most efficient realization for the input function, in terms of hardware and storage space. However, for some applications such as testing and fault detection, only certain FPRME spectral coefficients need to be obtained. The proposed algorithms offer different choices of algorithms that are suited for different applications. They start from either the truth

vector representation of the input function or reduced representation of the input function in terms of arbitrary FPRME spectral coefficient vector or array of disjoint cubes. Each algorithm is described and their computational complexities are analyzed. Their computational times for obtaining the optimal FPRME are also compared.

New fastest LI over GF(2) and LIA transforms for binary functions are introduced. They are derived from the existing fastest transforms such that they have the same low computational cost. Fast forward and inverse transforms for the transforms are given so that the butterfly diagrams for their fast software and hardware computation can be quickly obtained. In order to reduce the computational cost of obtaining the spectra of all the fastest transforms, relations between different fastest transforms are presented and the number and locations of the nonzero elements inside the transforms are analyzed. The properties are then used to obtain various arithmetic bounds on the spectra of linearly independent arithmetic transforms. Such properties are useful for testing and verification purposes.

Several algorithms for deriving the spectral coefficients of FPAEs are also shown. The FPAE is closely related to the FPRME as they have the same basis functions in some cases. However they operate over different algebra. The FPAE operates over standard arithmetic algebra and consequently they have the useful property of being able to represent multiple functions by a single expansion. With proper modification, algorithms that have been found efficient for FPRME can be applied for FPAE as well. In this thesis, such algorithms are shown for the FPAEs of ternary and quaternary functions. The fundamental relations used in the algorithms are first analyzed and the steps and computational costs for the introduced algorithms are derived.

The concept of the LI transforms has been extended to represent ternary functions and so in this thesis new spectral transforms for ternary functions are proposed whose basis functions are a set of linearly independent ternary functions over GF(3). The new transforms are chosen such that they have low computational complexity and can be calculated efficiently using fast transform. These requirements ensure that the conversion between the operational and spectral domain based on the transforms can be performed efficiently. This is important as in practice the applicability of a spectral

transform largely depends on the complexity of computation for forward and inverse transformations. Similar to the binary case, the transforms are classified into fastest LI transforms over GF(3), in which all additions and multiplication are performed over GF(3), and LITA transforms, which use decimal arithmetic additions/subtractions and multiplications. Properties of the new transforms are investigated and experimental results for the transforms are shown for different classes of the transforms.

Hardware implementations for the new binary and ternary LI transforms are discussed. Efficient computation of their spectral coefficients from the truth vector is implemented using linear systolic array processor structure whereas their polynomial expansions are realized using a multi level tree modular implementation. It is shown that due to the relations between the fastest LI transforms, a single linear systolic array processor structure or multi level tree modular structure can be reused for many fastest LI transforms which leads to saving in hardware cost.

1.3 Organization of the thesis

The thesis is organized as follows:

Basic concepts and definitions that are used in this thesis are reviewed and presented in Chapter 2. In particular, the RM and arithmetic transforms, which are used for comparison in the subsequent chapters, are given.

Chapter 3 is devoted to the algorithms for optimization of FPRMEs over GF(5). Basic forward and inverse transforms for FPRME over GF(5) are first reviewed and their computational costs are established. They are subsequently used to develop basic relations that are applied by the proposed algorithms. Based on the relations used by the algorithms and the representations of the input function operated by them, the algorithms are differentiated into matrix multiplication, extended dual polarity, cube polarity adjustment, row polarity matrix, and column polarity matrix algorithms. The computational steps for the algorithms are presented and their computational costs are derived. The algorithms are implemented as C++ programs and their computational

costs for several five-valued test files are listed. The generation of the five-valued test files used in the experiment from the binary MCNC benchmark is also described.

New fastest LI transforms over GF(2) and fastest LIA transforms for binary functions are introduced in Chapter 4. Basic definitions for general binary LI and LIA expansions are given followed by the definitions of the forward and inverse matrices for the new transforms. Relations between the existing and new transforms are also shown. Based on the structure of their factorized matrices, the proposed transforms are divided into four types. Several properties on the relations between the forward and inverse transforms as well as between transforms of both same and different types are established. Properties stating number and location of nonzero elements inside the matrices are also found, which are useful for developing computational algorithms for them. Several arithmetic bounds on the fastest LIA spectral coefficients are also derived. Finally, several experimental results for the transforms are shown which compare the number and values of the nonzero terms inside their polynomial expansions.

In Chapter 5, FPAEs for ternary and quaternary functions are discussed. Their forward and inverse transforms are given and their general polynomial expansions are defined. Transformations between the truth vector and FPAEs in different polarities as well as their computational complexities are derived for both ternary and quaternary cases. The transformations are then used to develop different algorithms for FPAEs using similar concepts that have been applied for FPRME. Computational cost for each algorithm is derived and compared with each other. The resulting number of nonzero terms in the optimal FPAE for several quaternary test files are also found and compared with the corresponding number in the optimal FPRME over GF(4).

Chapter 6 focuses on the fastest TLI and LITA transforms for ternary functions. Existing fastest TLI and LITA transforms are first discussed. New TLI and LITA transforms are then generated from the existing ones through permutation. Relations and structures of the new transforms are investigated. The transforms are subsequently further extended into a larger class of transforms with the same computational cost by reordering and permuting the butterfly diagrams of the proposed TLI and LITA

transforms. Experimental results for the transforms are given which show that the new transforms spectra can have smaller number of nonzero elements than the spectra of the existing fastest TLI and LITA transforms.

Based on the butterfly diagrams of the LI transforms over GF(2), LIA, TLI, and LITA transforms, the linear systolic array processor structure for hardware calculation of their spectra can be derived. The derivation process is illustrated in Chapter 7, using LIA transforms as example. The implementation of their polynomial expansions by a multi level tree modular realization is also shown there. In addition, the use of the linear systolic array structures for obtaining the spectra of the FPRMEs and FPAEs using the recursive polarity matrix algorithms are discussed.

Chapter 8 concludes this thesis and comments on the possible future research directions.

Chapter 2

Representations and Transforms for Binary and Multiple-Valued Functions

This chapter reviews a number of mathematical concepts and definitions that are used in this thesis. Among these concepts are the different representations for both binary and multiple-valued functions as well as their spectral transformations. Most of the discussed concepts and definitions and further details on them can be found in [AS06, Bro90, DDT78, Gre86, Gre89, Gre90, HMM85, Jud94, Kar76, MW86, Ros99, SA03, Sas93, Sas99, SSJ98, TDM01, Yan94, Yan98].

2.1 Sets

A **set** is a well-defined finite or infinite collection of objects in which order has no significance. The objects that belong to a set are called its **elements** or **members**.

A set is usually specified either by listing all of its elements inside a pair of braces or by stating the property that determines whether or not an object x belongs to the set. We might write

$$X = \{x_1, x_2, \ldots, x_n\} \tag{2.1}$$

for a set containing elements $x_1, x_2, \ldots, x_n$ or

$$X = \{x \mid x \text{ satisfies } P\} \tag{2.2}$$

if each x in X satisfies a certain property P. We write

$$x_1 \in X$$

to denote that x_1 is the element of set X and

$$x_2 \notin X$$

to say that x_2 is not an element of set X. A set with no elements in it is called an **empty set** and is denoted by $\varnothing$. The number of elements in a finite set A is called the **cardinality** of A, often denoted by $|A|$ or $\# A$.

A set A is called a **subset** of B, written $A \subseteq B$ or $B \supseteq A$, if every element of A is also an element of B. If A and B are unequal, then A is a **proper subset** of B, written as $A \subset B$ or $B \supset A$. Conversely, if A is not a subset of B then we write $A \not\subset B$. When two sets have no elements in common, the sets are said to be **disjoint**. Two sets are **equal**, written $A = B$, iff $A \subseteq B$ and $B \subseteq A$.

We can construct a new set out of existing sets by performing union and intersection operations on the existing sets. The **union** of two sets A and B is defined as

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}. \tag{2.3}$$

The **intersection** of sets A and B is defined by

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}. \tag{2.4}$$

The **Cartesian product** (**direct product**) on n sets $X_1, X_2, \ldots, X_n$ is the set of all possible ordered n-tuples such that the i-th component in the n-tuples is an element of the set X_i $(1 \le i \le n)$. We can write

$$X_1 \times X_2 \times \ldots \times X_n = \{< x_1, x_2, \ldots, x_n > \mid x_1 \in X_1 \text{ and } x_2 \in X_2 \text{ and} \ldots \text{and } x_n \in X_n\}. \tag{2.5}$$

Example 2.1. Let $X_1 = \{2,3\}$ and $X_2 = \{a,b,c\}$. Then $X_1 \times X_2 = \{<2,a>,<2,b>,<2,c>,<3,a>,<3,b>,<3,c>\}$ and $X_2 \times X_1 = \{<a,2>,<a,3>,<b,2>,<b,3>,<c,2>,<c,3>\}$.

2.2 Matrices

A **matrix** is a rectangular array of numbers. An $m \times n$ matrix consists of m rows and n columns. A matrix is said to be **square** if $m = n$ and **rectangular** if $m \neq n$. An $m \times 1$ matrix is called a **column vector** whereas a $1 \times n$ matrix is called a **row vector**.

Let A be an $m \times n$ matrix. Then A has the following general form:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}.$$

The entry of a matrix A that lies in the i-th row and j-th column is often written as $A_{i,j}$. It is often convenient to express the matrix A by $A = [a_{i,j}]$, which indicates that A is the matrix with its (i,j)-th element equal to $a_{i,j}$.

An $n \times n$ **identity matrix**, often denoted by I_n, is a special square matrix that is given explicitly by

$$I_{i,j} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j. \end{cases} \tag{2.6}$$

Matrices can be both added and multiplied. Matrix addition is both commutative and associative. Matrix multiplication is also associative and distributive but, in general, not commutative.

The **inverse** of an $n \times n$ square matrix A, sometimes called reciprocal matrix, is an $n \times n$ matrix A^{-1} such that

$$AA^{-1} = I_n, \tag{2.7}$$

where I_n is the $n \times n$ identity matrix. A matrix for which an inverse can be found is called **nonsingular** or **invertible**.

The **transpose** of an $m \times n$ matrix A, commonly denoted by A^T, is an $n \times m$ matrix that is obtained by exchanging the rows and columns of matrix A such that

$$A^T{}_{i,j} = A_{j,i} \forall i, j \text{ where } i = 0,1,\ldots,n-1 \text{ and } j = 0,1,\ldots,m-1. \tag{2.8}$$

The transpose of the product of matrix multiplication satisfies the following property:

$$(AB)^T = B^T A^T. \tag{2.9}$$

2.3 Kronecker product

Let A be an $m \times n$ matrix and B be a $p \times q$ matrix. Also let C be the result of the **Kronecker product** (direct matrix product, tensor product) $A \otimes B$. Then C is an $mp \times nq$ matrix with elements defined by

$$C_{\alpha,\beta} = A_{i,j} B_{k,l}, \tag{2.10}$$

where

$$\alpha = pi + k \tag{2.11}$$

$$\beta = qj + l. \tag{2.12}$$

Thus,

$$C = A \otimes B$$

$$= \begin{bmatrix} a_{0,0}B & a_{0,1}B & \cdots & a_{0,n-1}B \\ a_{1,0}B & a_{1,1}B & \cdots & a_{1,n-1}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0}B & a_{m-1,1}B & \cdots & a_{m-1,n-1}B \end{bmatrix}$$

$$= \begin{bmatrix} a_{0,0}b_{0,0} & a_{0,0}b_{0,1} & \cdots & a_{0,n-1}b_{0,q-1} \\ a_{0,0}b_{1,0} & a_{0,0}b_{1,1} & \cdots & a_{0,n-1}b_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0}b_{p-1,0} & a_{m-1,0}b_{p-1,1} & \cdots & a_{m-1,n-1}b_{p-1,q-1} \end{bmatrix}. \tag{2.13}$$

The Kronecker product has the following properties:

$$A \otimes (B + C) = A \otimes B + A \otimes C \tag{2.14}$$

$$(B + C) \otimes A = B \otimes A + C \otimes A \tag{2.15}$$

$$k(A \otimes B) = (kA) \otimes B = A \otimes (kB) \tag{2.16}$$

$$(A \otimes B)(C \otimes D) = AC \otimes BD \tag{2.17}$$

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{2.18}$$

$$(A \otimes B)^{T} = A^{T} \otimes B^{T}. \tag{2.19}$$

2.4 Functions

A **function** *f* from a set *A* into a set *B*, written

$$f : A \to B$$

assigns every element $a \in A$ an element $f(a) \in B$. We say that *f* maps *A* to *B*. The set *A* is called the **domain** of *f*; the set *B* is called the **co-domain** of *f*. If $f(a) = b$, *b* is said to be the **image** of *a* and *a* is a **pre-image** of *b*. The **range** of *f* is the set of images of elements of *A* under *f*.

The range of *f* is clearly a subset of its co-domain. If the range of *f* is equal to *B*, we say that the function is **onto** *B*. Also, if $f(a_1) = f(a_2)$ implies that $a_1 = a_2$ for all a_1 and a_2 in the domain of *f*, we say that the function *f* is **one-to-one**. If *f* is a one-to-one correspondence, then we can define an **inverse function** of *f*, denoted by f^{-1}. The

inverse function of f is the function that assigns to an element $b \in B$ the unique element a in A such that $f(a) = b$. Hence, $f^{-1}(b) = a$ when $f(a) = b$.

A function is a specialized relation. Recall that a relation from A to B is a subset of $A \times B$, i.e., a collection of ordered pairs each of which takes its first element from A and its second element from B. Accordingly, we define a function f from A into B as a relation from A to B having the property that each element of A appears as a first element in exactly one of the ordered pairs in f. Thus the formulas "$f(x) = y$" and "$(x, y) \in f$" are equivalent predicates.

An n-variable **binary function** f is a mapping:

$$f : B^n \rightarrow Y,$$

where $B = \{0, 1\}$ and $Y = \{0, 1\}$ or $\{0, 1, -\}$. If $Y = \{0, 1\}$ then the function f is a **completely specified** binary function. Otherwise, f is an **incompletely specified** binary function.

By permitting the domain and co-domain of f to have more than two values, we can obtain a **multiple-valued function**. Let $\vec{X} = \{X_1, X_2, \ldots, X_n\}$ be a set of n multiple-valued variables. Then an n-variable multiple-valued input p-valued output function can be defined as a mapping

$$f\left(\vec{X}\right): R_1 \times R_2 \times \ldots \times R_n \rightarrow P,$$

where X_i, $1 \leq i \leq n$ takes the values from the set $R_i = \{0, 1, \ldots, r_i - 1\}$ and $P = \{0, 1, \ldots, p-1, -\}$ (where – denotes a don't care value). If $R_1 = R_2 = \ldots = R_n = \{0, 1, \ldots, r-1\}$ and $P = \{0, 1, \ldots, p-1\}$ then f is said to be an n-variable r-valued input p-valued output completely specified logic function. When $r = p = 3$, f is a **ternary function** and when $r = p = 4$, f is a **quaternary function**.

A digital function can be represented in many forms where the effectiveness of a representation depends on the underlying problem for which it is used. The simplest representation of a function is a **truth table**, which lists all possible values of the input

variables and the corresponding output value. Fig. 2.1 shows the truth table for a three-variable incompletely specified binary function.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 2.1: Truth table for three-variable binary function.

From the truth table, we can get the **truth vector** for a function, which contains the output values for all possible values of the input variables. For the function in Fig. 2.2, the truth vector is given by $\vec{F} = [1,1,0,1,0,0,1,1]$. The truth vector representation is very useful in obtaining spectral representations by matrix operations. Both the truth table and truth vector represent the operational behavior of the function and therefore describes the operational domain.

In both the truth tables and truth vector, the function values for all possible input conditions are listed. This makes them unsuitable for functions with a large number of variables. The function representation can be simplified by considering the various relationships that exist between the function values. Such representations are called reduced representation. Examples of reduced representations are array of cubes, spectral polynomial expansions, and decision diagrams.

A **literal** is defined to be an input variable or its complement. A **minterm** for an *n*-variable function is a product of *n* literals, with one literal for each variable. Several

minterms can be combined into a **cube** if they share common literals. Instead of the truth vector, a function can be represented by an array of cubes that describe the function. For example, a completely specified binary function can be described by an array of cubes for which the value of the function is either 0 or 1 and it is understood that for minterms that are not covered by the array of cubes the function value is 1 or 0, respectively.

Example 2.2. From the truth table in Fig. 2.1, it can be seen that the two-variable binary function has output value 1 for the minterms 000, 001, 011, 110, and 111 and it has output value 0 for the minterms 010, 100, and 101. Thus, the function can be completely described either by the array of 1 cubes {00–, 0–1, 11–} or by the array of 0 cubes {010, 10–}.

Alternatively, an *n*-variable *p*-valued function can also be represented by **spectral polynomial expansion** in the form of

$$f = \sum_{i=0}^{p^n-1} a_i \phi_i ,$$

where a_i and ϕ_i are called the spectral coefficients and basis functions of the particular polynomial expansion. The set of the basis functions ϕ_i are linearly independent to allow canonical representation for a function and to maintain all the information content from the operational domain. The translation between the truth vector representation and the spectral representation can be performed through matrix operations. Examples of spectral polynomial expansions are Reed-Muller, arithmetic, Walsh, and Haar expansions.

A set of functions $f_1, f_2, \ldots, f_n$ is said to be **linearly independent** if no set of constants $a_1, a_2, \ldots, a_n$, of which at least one is nonzero, exists such that

$$a_1 f_1 + a_2 f_2 + \ldots + a_n f_n = 0$$

Any function from a set of linearly independent functions cannot be reduced to some other function from this set by some simple linear operations over this function and other functions in the set.

2.5 Galois fields

An **algebraic system** is defined by the tuple

$$< A, o_1, \ldots, o_k ; R_1, \ldots, R_m ; c_1, \ldots, c_k >,$$

where A is a non-empty set, o_i is a function $A^{p_i} \to A$, p_i is a positive integer, R_j is a relation on A, and c_i is an element of A.

An algebraic system is a **field** if it has the following properties, where *x*, *y*, and *z* are elements of *A*, $\oplus$ denotes addition and $\cdot$ denotes multiplication:

1. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.
2. For all *x*, a unique 0 element exists such that $x \oplus 0 = 0 \oplus x = x$.
3. For any *x*, an element *y* can be found such that $x \oplus y = y \oplus x = 0$.
4. $x \oplus y = y \oplus x$.
5. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.
6. For any *x*, a unique 1 element exists such that $x \cdot 1 = 1 \cdot x = x$.
7. For any arbitrary nonzero element *x*, an element *y* can be found such that $x \cdot y = y \cdot x = 1$.
8. $x \cdot y = y \cdot x$.
9. $x \cdot (y \oplus z) = x \cdot y \oplus x \cdot z$.
10. $(y \oplus z) \cdot x = y \cdot x \oplus z \cdot x$.

A **finite field (Galois field)** is a field having a finite number of elements. Such a field must have some (finite) prime characteristic p. Hence, its order must be some power $q = p^m$ of the prime p.

When $m = 1$, $q = p$ and so inside GF(q) the elements are the integers modulo q (0, 1, 2, …, $q - 1$) and the arithmetic operations are simply the modulo q arithmetic. For example, inside GF(3) the elements are 0, 1, and 2 whereas the addition and multiplication operations are as shown in Fig. 2.2.

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Figure 2.2: Addition and multiplication operations over GF(3).

A finite field GF(q) is called composite field when $m > 1$. For the composite field we cannot simply use integers modulo q with modulo q arithmetic as the condition 7 above will not be satisfied. For example, when $q = 4 = 2^2$, for element 2 we cannot find any element y under modulo 4 multiplication such that $2 \cdot y = 1$. Fig. 2.3 shows the addition and multiplication operations over GF(4).

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Figure 2.3: Addition and multiplication operations over GF(4).

The addition $f+g$ and the multiplication fg of two Galois functions f and g:

$$f:\{GF(q)\}^n \to GF(q)$$

$$g:\{GF(q)\}^n \to GF(q)$$

are defined to be the component wise extensions of the addition and of the multiplication in GF(q) respectively, i.e.:

$$[(f+g)_e] = [f(e)+g(e)],$$

$$[(fg)_e] = [f(e)g(e)].$$

Let $l \in GF(q)$; the addition $l+f$ and the multiplication lf of the Galois function f and of the field element l is defined by the above qualities when the function g reduces to a constant function l over GF(q).

With these definitions, the set of Galois function $\{GF(q)\}^n \to GF(q)$ is a linear algebra of functions.

2.6 Fixed polarity Reed-Muller expansions

A binary function f can be represented by EXOR sum of products by recursively applying the following expansion:

$$f = f_0 \oplus x_i(f_0 \oplus f_1), \tag{2.20}$$

where $f_0 = f(x_i = 0)$ and $f_1 = f(x_i = 1)$ are the cofactors of f with respect to the variable x_i. The expansion is called the positive Davio expansion and is also known as the Zhegalkin polynomial. It is the earliest Reed-Muller expansion proposed for representing binary function.

In the positive Davio expansion, all the variables appear in positive (not complimented) literals. An equivalent expansion for the function can be derived in

which all the variables appear in negative (complimented) literals. Such expansion is called negative Davio expansion and it is obtained by using the following decomposition rule:

$$f = f_1 \oplus \bar{x}_i (f_0 \oplus f_1). \tag{2.21}$$

By permitting either positive or negative Davio expansion to be performed with respect to a particular input variable x_i, more equivalent Reed-Muller expansions can be obtained for *f*. The resulting expansions are called **fixed polarity Reed-Muller expansions** (FPRMEs). In an FPRME, each variable always appears in the same literal throughout the expansion. Clearly, there are 2^n possible FPRMEs for an *n*-variable binary function, including the positive and negative Davio expansions. Each FPRME is uniquely identified by its polarity numbers, which is defined as the decimal equivalent of the *n*-bit binary number formed by assigning 0 or 1 for each variable according to whether it appear in positive or negative literal, respectively.

The FPRME for an *n*-variable binary function can be written in the following general form:

$$f(\vec{x}) = \sum_{i=0}^{2^n-1} c_i^{\omega} \left[\prod_{l=1}^{n} \hat{x}_l^{k_l} \right], \tag{2.22}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$, $c_i^{\omega} \in \{0,1\}$ is the *i*-th spectral coefficient, ω is the polarity number, $< i >_{10} = < k_1, k_2, \ldots, k_n >_2$ is the spectral coefficient index, $\hat{x}_l$ is the literal of the variable x_l in the FPRME, and $\hat{x}_l^{k_l}$ is the k_l-th power of $\hat{x}_l$. The additions and multiplications are performed over GF(2).

Let $\vec{F}$ be the truth vector of $f(\vec{x})$ and C^{ω} be a column vector that is defined by

$$C^{\omega} = [c_0^{\omega}, c_1^{\omega}, \ldots, c_{2^n-1}^{\omega}]. \tag{2.23}$$

Then, the translation between C^{ω} and $\vec{F}$ can be performed by the following transforms:

$$C^{\omega} = RM_n^{\omega} \cdot \vec{F} \tag{2.24}$$

and

$$\vec{F} = \left(RM_n^{\omega}\right)^{-1} \cdot C^{\omega}, \tag{2.25}$$

where $\left(RM_n^{\omega}\right)^{-1}$ is the inverse of RM_n^{ω} over GF(2) and the multiplication is performed over GF(2).

The matrix RM_n^{ω} can be obtained by the Kronecker product

$$RM_n^{\omega} = \otimes\prod_{l=1}^{n} rm_{\omega_l}, \tag{2.26}$$

where $<\omega>_{10} = <\omega_1, \omega_2, \ldots, \omega_n>_2$, $rm_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and $rm_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$.

Example 2.3. The truth vector for the three-variable binary function in Fig. 2.1 is given by $\vec{F} = [1,1,0,1,0,0,1,1]$. Then by (2.22)–(2.26), the spectral coefficients of the FPRME with polarity number 4 ($<4>_{10} = <1,0,0>_2$) can be calculated as follows:

$$\begin{aligned} C^4 &= RM_3^4 \cdot F^T \\ &= \left(rm_1 \otimes rm_0 \otimes rm_0\right) \cdot F^T \\ &= \begin{bmatrix} 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&1&1&1&1 \\ 1&0&0&0&1&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \end{aligned}$$

Hence, the FPRME in polarity 4 for the function is $f(\vec{x}) = x_2 \oplus \bar{x}_1 \oplus \bar{x}_1 x_2 x_3$.

The binary Reed-Muller can be extended to represent multiple-valued functions in many ways, depending on how the literals and operations are generalized. One possible way is by allowing a p-valued variable x_i to be represented by additive literals $(x_i + \omega_i)$, where ω_i varies from 0 to $p-1$, and by performing all additions and multiplications over GF(p). With such modifications, the FPRME for an n-variable p-valued function is of the form

$$f(\vec{x}) = \sum_{i=0}^{p^n-1} c_i{}^{\omega} \left[\prod_{l=1}^{n} \hat{x}_l{}^{k_l} \right], \tag{2.27}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$, $c_i^{\omega} \in \{0,1,\ldots,p-1\}$, $<\omega>_{10} = <\omega_1, \omega_2, \ldots, \omega_n>_p$, $<i>_{10} = <k_1, k_2, \ldots, k_n>_p$, $\hat{x}_l = (x_l + \omega_l)$ over GF(p) is the literal of the variable x_l, and $\hat{x}_l{}^{k_l}$ is the k_l-th power of $\hat{x}_l$. The additions, multiplications, and power operations are performed over GF(p).

The equations (2.24)–(2.26) for binary FPRME are also valid for FPRME over GF(p), except that for the latter the length of C^{ω} is p^n and $\left(RM_n^{\omega}\right)^{-1}$ is the inverse of RM_n^{ω} over GF(p). There are p basic transforms rm_{ω_l} for FPRME over GF(p), where each is of size $p \times p$. Tables 2.1 and 2.2 show the basic transforms for FPRME over GF(3) and GF(4), respectively.

2.7 Fixed polarity arithmetic expansions

Arithmetic expansions for binary functions have the same basis functions as the binary Reed-Muller expansions. However, the additions and multiplications inside the arithmetic expansion is performed over standard arithmetic algebra instead of over GF(2) as is the case for binary Reed-Muller. Hence, the expansion can be considered to be the integer counterpart of Reed-Muller. Arithmetic transform is also known in the literature as probability transform or inverse integer Reed-Muller transform.

Table 2.1: FPRME transforms over GF(3).

ω_i	rm_{ω_i}
0	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix}$
1	$\begin{bmatrix} 0 & 0 & 1 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 2 & 2 \end{bmatrix}$

Table 2.2: FPRME transforms over GF(4).

ω_i	rm_{ω_i}
0	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
1	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 3 & 2 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
3	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 2 & 3 & 1 & 0 \\ 3 & 2 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Analogously to the FPRME, the **fixed polarity arithmetic expansion** (FPAE) is the term used to refer to the arithmetic expansions in which each variable must appear in the same literal throughout the expansion. In order to identify individual FPAE,

each FPAE is assigned a unique polarity number which is defined the same way as for the FPRME. Inside the FPAE in polarity zero, all variables appear in positive (not complimented) literals. The FPAE in polarity zero can be derived using the following decomposition rule, which is the arithmetic analogue of the positive Davio expansion.

$$f = f_0 + x_i(-f_0 + f_1). \tag{2.28}$$

On the other hand, inside the FPAE in polarity $2^n - 1$, all variables always appear in negative (complimented) form. The expansion can be obtained by recursively applying the arithmetic analogue of the negative Davio expansion rule as follows:

$$f = f_1 + \bar{x}_i(f_0 - f_1). \tag{2.29}$$

The FPAE in other polarities can be derived by applying the expansion rule in (2.28) or (2.29) for the function with respect to every variable x_i, according to whether the variable should appears in positive or negative form in the FPAE, respectively.

The FPAE for an *n*-variable binary function has the following general form:

$$f(\vec{x}) = \sum_{i=0}^{2^n-1} a_i{}^{\omega} \left[\prod_{l=1}^{n} \hat{x}_l{}^{k_l} \right], \tag{2.30}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$, a_i^{ω} is the *i*-th spectral coefficient, ω is the polarity number, $< i >_{10} = < k_1, k_2, \ldots, k_n >_2$ is the spectral coefficient index, $\hat{x}_l$ is the literal of the variable x_l in the FPAE, and $\hat{x}_l{}^{k_l}$ is the k_l-th power of $\hat{x}_l$. The additions and multiplications are performed over standard arithmetic algebra.

The conversion between the truth vector and FPAE spectrum can be performed by matrix multiplications. Let $\vec{F}$ be the truth vector of $f(\vec{x})$ and A^{ω} be the FPAE spectrum for $f(\vec{x})$ in polarity ω such that

$$A^{\omega} = [a_0^{\omega}, a_1^{\omega}, ..., a_{2^n-1}^{\omega}]. \tag{2.31}$$

Then,

$$A^{\omega} = AR_n^{\omega} \cdot \vec{F} \tag{2.32}$$

and

$$\vec{F} = \left(AR_n^{\omega}\right)^{-1} \cdot A^{\omega}, \tag{2.33}$$

where $\left(AR_n^{\omega}\right)^{-1}$ is the inverse of AR_n^{ω} over standard arithmetic algebra.

As the FPRME and FPAE have identical basis functions, the matrix $\left(AR_n^{\omega}\right)^{-1}$ for FPAE is equal to the matrix $\left(RM_n^{\omega}\right)^{-1}$ of the FPRME, whereas the $2^n \times 2^n$ matrix AR_n^{ω} can be constructed by the Kronecker product

$$AR_n^{\omega} = \otimes \prod_{l=1}^{n} ar_{\omega_l}, \tag{2.34}$$

where $< \omega >_{10} = < \omega_1, \omega_2, \ldots, \omega_n >_2$, $ar_0 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$, and $ar_1 = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$.

One important property of arithmetic transform is its ability to represent systems of binary functions. This is done by taking the weighted sum of the binary truth vectors of the functions to obtain an integer truth vector, which is then used to calculate the FPAE. The individual binary truth vectors can be obtained back from the resulting FPAE by applying the arithmetic inverse transform.

Example 2.4. Let $\vec{F}_1$, $\vec{F}_2$, and $\vec{F}_3$ be the truth vectors of three binary functions, where $\vec{F}_1 = [1,0,0,0,1,1,0,0]^T$, $\vec{F}_2 = [0,1,1,0,0,0,1,1]^T$, and $\vec{F}_3 = [0,1,0,1,1,0,1,0]^T$. Then we can derive the FPAE in polarity zero to represent the system of three binary functions as follows.

First, obtain an integer truth vector by taking the weighted sum of the three binary truth vectors:

$$\vec{F}_4 = 2^2 \cdot \vec{F}_1 + 2^1 \cdot \vec{F}_2 + 2^0 \cdot \vec{F}_3$$

$$= 4 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + 2 \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \\ 5 \\ 4 \\ 3 \\ 2 \end{bmatrix}.$$

Subsequently, find the FPAE spectrum in polarity zero for the integer truth vector:

$$A_3^0 = \left(\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \\ 5 \\ 4 \\ 3 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ -1 \\ 2 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Thus, the system of functions can be represented by the FPAE $f(\vec{x}) = 4 - x_3 + 2x_2 + x_1$.

Clearly, if we take arithmetic inverse transform on the spectrum A_3^0 we can get back the integer truth vector $\vec{F}_4$ and hence, the original binary truth vectors $\vec{F}_1$ to $\vec{F}_3$.

Chapter 3

Efficient Algorithms for Calculation of Fixed Polarity Reed-Muller Expansions over GF(5)

The generation of FPRMEs over GF(5) for five-valued functions is considered in this chapter. Basic definitions for FPRME over GF(5) and the notations used throughout this chapter will be first reviewed. Basic forward and inverse FPRME transforms for the conversion between the truth vector and FPRME spectra are given. Mutual relations that exist between the different FPRME representations for five-valued functions are then established, and used to develop several algorithms that can be used to generate the FPRMEs for the input function. The presented algorithms are simple and they start from different representations of the input function. For comparison purpose, experimental results of the algorithms for several five-valued test files are presented at the end of the chapter. Since currently there are no five-valued benchmark functions available, the five-valued test files that are used in the experiments are obtained by modifying the MCNC binary benchmark functions in a regular manner. The rules used to convert the binary benchmark functions to the five-valued test files are given.

3.1 Basic definitions for FPRMEs over GF(5)

Definition 3.1.1. Any five-valued function can be represented by their corresponding FPRMEs over GF(5). Let $^{<0>}x$, $^{<1>}x$, $^{<2>}x$, $^{<3>}x$, and $^{<4>}x$ denote the five literals of a particular five-valued variable x, where $^{<j>}x = x + j$ over GF(5). Then, in an FPRME over GF(5) each variable x_l $(1 \le l \le n)$ must always appear in the form of $^{<j_l>}x_l$ $(0 \le j_l \le 4)$ with the same j_l value throughout the expansion.

Definition 3.1.2. The polarity of an FPRME, denoted by ω, is taken as the decimal number representation of the five-valued number $< j_1, j_2, ..., j_n >$, where $^{<j_l>}x_l$ denotes the literal of the *l*-th variable in the FPRME. That is,

$$< \omega >_{10} = < j_1, j_2, ..., j_n >_5 . \tag{3.1}$$

Definition 3.1.3. The FPRME spectral coefficient vector (spectrum) in polarity ω of an *n*-variable five-valued function is a vector of length 5^n whose elements are all the spectral coefficients of the FPRME in polarity ω for the function. Let C^ω and c_i^ω denote the spectral coefficient vector in polarity ω and the *i*-th spectral coefficient of the FPRME in polarity ω, respectively. Then,

$$C^\omega = [c_0^\omega, c_1^\omega, ..., c_{5^n-1}^\omega]. \tag{3.2}$$

Definition 3.1.4. The FPRME polarity matrix $P[f(\vec{x})]$ is defined as a square matrix of size $5^n \times 5^n$ that contains all the FPRME spectral coefficients of the *n*-variable function $f(\vec{x})$ such that row *i* $(0 \le i \le 5^n - 1)$ of $P[f(\vec{x})]$ corresponds to C^i.

$$P[f(\vec{x})] = \left[C^0, C^1, ..., C^{5^n-1}\right]^T, \tag{3.3}$$

where *T* denotes matrix transpose operator.

Property 3.1.1. Let the element that is located at row *i* and column *l* of $P[f(\vec{x})]$ be

denoted by $P_{i,l}$, where $0 \le i, l \le 5^n-1$. Then by Definition 3.1.4,

$$P_{i,l} = c_l^i .$$

Furthermore, the optimal polarity of $f(\vec{x})$ corresponds to the row index number of the $P[f(\vec{x})]$ row with the largest number of zeros in its columns.

Definition 3.1.5. Let $f(\vec{x})$ be any n-variable five-valued function. Then the FPRME of $f(\vec{x})$ in polarity ω can be written as follows:

$$f(\vec{x}) = \sum_{i=0}^{5^n-1} c_i^{\omega} \left[\prod_{l=1}^{n} \hat{x}_l^{k_l} \right], \tag{3.4}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$, $\hat{x}_l = {}^{<j_l>}x_l$ is the literal of the l-th variable, $c_i^{\omega} \in \{0,1,2,3,4\}$ is the i-th element of C^{ω} for $f(\vec{x})$, and $< i >_{10} = < k_1, k_2, \ldots, k_n >_5$ ($k_l \in \{0, 1, 2, 3, 4\}$) is the spectral coefficient index. All the additions and multiplications in the expression are to be performed over GF(5) as shown in Table 3.1.

Table 3.1: Addition and multiplication operations over GF(5).

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Definition 3.1.6. [JSM03] Let $< a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n >$ and $< b_1, \ldots, b_{i-1}, b_i, b_{i+1}, \ldots, b_n >$ be the n-digit five-valued number representations of any two decimal numbers a and b, respectively, where $0 \le a, b \le 5^n - 1$. Then the polarities a and b are said to be extended dual polarities of each other iff $a_l \ne b_l$ for $l = i$ and $a_l = b_l$ otherwise ($1 \le i, l \le n$).

3.2 FPRME transforms over GF(5), their relations, and their computational costs

The most basic FPRME transforms that allow an FPRME spectrum of a five-valued function to be obtained from either truth vector or another FPRME spectrum are discussed in this section. They are the underlying transforms based on which the algorithms described in this chapter are developed. Here the focus is to derive the computational cost of those fundamental transforms in terms of the required number of additions and multiplication operations. As commonly known, computational cost is one indicator of effectiveness of transforms and/or algorithms. Therefore it would be useful to deduce the computational cost of these fundamental transforms so that a comparison of computational costs can be made with the described algorithms as a measure of the algorithms' effectiveness.

Given the truth vector $\vec{F}$ of an n-variable five-valued function $f(\vec{x})$, the FPRME spectrum of $f(\vec{x})$ in polarity ω can be calculated by

$$\left(C^{\omega}\right)^{T} = S_n^{<\omega>} \cdot \vec{F}, \tag{3.5}$$

where $S_n^{<\omega>}$ denotes the forward FPRME transform matrix of polarity ω. The transform matrix $S_n^{<\omega>}$ is of size $5^n \times 5^n$.

Conversely, the truth vector $\vec{F}$ can be recovered from C^{ω} by

$$\vec{F} = T_n^{<\omega>} \cdot \left(C^{\omega}\right)^{T}, \tag{3.6}$$

where $T_n^{<\omega>}$ is the inverse FPRME transform matrix of polarity ω. The matrix $T_n^{<\omega>}$ is also of size $5^n \times 5^n$ and $T_n^{<\omega>} = \left(S_n^{<\omega>}\right)^{-1}$ over GF(5).

The transform matrices $S_n^{<\omega>}$ and $T_n^{<\omega>}$ can be constructed by taking Kronecker product of n forward and inverse FPRME transform matrices of one-variable input function, respectively as follows:

$$S_n^{<\omega>} = \otimes\prod_{l=1}^{n} S_1^{<j_l>} \tag{3.7}$$

and

$$T_n^{<\omega>} = \otimes\prod_{l=1}^{n} T_1^{<j_l>}, \tag{3.8}$$

where $S_1^{<j>}$ and $T_1^{<j>}$ represent the forward and inverse FPRME transform matrices for one-variable five-valued function in polarity j, respectively (recall that $<\omega>_{10}=<j_1, j_2, \ldots, j_n>_5$). Table 3.2 lists all possible $S_1^{<j>}$ and $T_1^{<j>}$ matrices.

Table 3.2: Transform matrices $S_1^{<j>}$ and $T_1^{<j>}$.

j	$T_1^{<j>}$	$S_1^{<j>}$
0	$\begin{bmatrix}1&0&0&0&0\\1&1&1&1&1\\1&2&4&3&1\\1&3&4&2&1\\1&4&1&4&1\end{bmatrix}$	$\begin{bmatrix}1&0&0&0&0\\0&4&2&3&1\\0&4&1&1&4\\0&4&3&2&1\\4&4&4&4&4\end{bmatrix}$
1	$\begin{bmatrix}1&1&1&1&1\\1&2&4&3&1\\1&3&4&2&1\\1&4&1&4&1\\1&0&0&0&0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1\\4&2&3&1&0\\4&1&1&4&0\\4&3&2&1&0\\4&4&4&4&4\end{bmatrix}$
2	$\begin{bmatrix}1&2&4&3&1\\1&3&4&2&1\\1&4&1&4&1\\1&0&0&0&0\\1&1&1&1&1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1&0\\2&3&1&0&4\\1&1&4&0&4\\3&2&1&0&4\\4&4&4&4&4\end{bmatrix}$
3	$\begin{bmatrix}1&3&4&2&1\\1&4&1&4&1\\1&0&0&0&0\\1&1&1&1&1\\1&2&4&3&1\end{bmatrix}$	$\begin{bmatrix}0&0&1&0&0\\3&1&0&4&2\\1&4&0&4&1\\2&1&0&4&3\\4&4&4&4&4\end{bmatrix}$
4	$\begin{bmatrix}1&4&1&4&1\\1&0&0&0&0\\1&1&1&1&1\\1&2&4&3&1\\1&3&4&2&1\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&0\\1&0&4&2&3\\4&0&4&1&1\\1&0&4&3&2\\4&4&4&4&4\end{bmatrix}$

The algorithms presented in this chapter utilize existing relations between the FPRME spectral coefficient vectors in different polarities. From (3.5) and (3.6), the

relation between the spectral coefficient vector in polarity zero C^0 and the spectral coefficient vector in any other polarity C^ω can be derived. When $n = 1$,

$$\begin{aligned}\left(C^{<j>}\right)^T &= S_1^{<j>} \cdot \vec{F} \\ &= S_1^{<j>} \cdot \left(T_1^{<0>} \cdot \left(C^0\right)^T\right) \\ &= Z_1^{<j>} \cdot \left(C^0\right)^T, \end{aligned} \tag{3.9}$$

where $j \in \{0,1,2,3,4\}$. Table 3.3 shows the basic transform matrices $Z_1^{<j>}$ for all possible values of j.

Generalizing this relation to any n-variable function, the following relation can be obtained:

$$\begin{aligned}\left(C^{<\omega>}\right)^T &= S_n^{<\omega>} \cdot \vec{F} \\ &= \left(\otimes\prod_{l=1}^{n} S_1^{<j_l>}\right) \cdot \left(\otimes\prod_{l=1}^{n} T_1^{<0>}\right) \cdot \left(C^0\right)^T \\ &= \otimes\prod_{l=1}^{n} \left(S_1^{<j_l>} \cdot T_1^{<0>}\right) \cdot \left(C^0\right)^T \\ &= \otimes\prod_{l=1}^{n} Z_1^{<j_l>} \cdot \left(C^0\right)^T \\ &= Z_n^{<\omega>} \cdot \left(C^0\right)^T. \end{aligned} \tag{3.10}$$

It can be observed from Table 3.2 that for $0 \le s, t \le 4$ $Z_1^{<s>} \cdot Z_1^{<t>} = Z_1^{<(s+t) \bmod 5>}$ and $Z_1^{<0>} = I_1$ (identity matrix of size 5×5). Based on these, (3.10) can be further extended to relate a pair of FPRME spectra with any two polarities a and b ($0 \le a, b \le 5^n - 1$) as follows:

$$\begin{aligned}\left(C^a\right)^T &= Z_n^{<a>} \cdot \left(C^0\right)^T \\ &= Z_n^{<a>} \cdot \left(Z_n^{<b>}\right)^{-1} \cdot \left(C^b\right)^T \end{aligned}$$

$$= \otimes\prod_{l=1}^{n} Z_1^{<a_l>} \cdot \otimes\prod_{l=1}^{n} Z_1^{<4b_l>} \cdot \left(C^b\right)^T$$

$$= \otimes\prod_{l=1}^{n} Z_1^{<a_l+4b_l>} \cdot \left(C^b\right)^T$$

$$= Z_n^{<a+4b>} \cdot \left(C^b\right)^T . \qquad (3.11)$$

Note that the transform matrices for (3.10) and (3.11) are exactly the same.

Table 3.3: Transform matrices $Z_1^{<j>}$.

j	$Z_1^{<j>}$
0	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
1	$\begin{bmatrix} 1 & 4 & 1 & 4 & 1 \\ 0 & 1 & 3 & 3 & 1 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
2	$\begin{bmatrix} 1 & 3 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 1 & 4 & 4 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 2 & 4 & 3 & 1 \\ 0 & 1 & 4 & 2 & 2 \\ 0 & 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
4	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

The number of necessary additions to calculate a transform is dependent on the

number of nonzero elements and the number of rows inside it whereas the number of required multiplications is determined only by the number of nonzero elements inside the matrix. Due to the regular structures of $S_n^{<\omega>}$, $T_n^{<\omega>}$, and $Z_n^{<\omega>}$, the number of 0s, 1s, 2s, 3s, and 4s inside the transforms can be obtained. In what follows, the computational costs of the transforms (3.5), (3.6), (3.10), and (3.11) in terms of addition and multiplication numbers are analyzed and derived. For each of the transforms, two cases are considered: when the transform is performed using direct calculation and when the transform is generated using fast transform, which is related to Good's theorem [SSJ98]. In the former case the transform matrix is first generated by appropriate Kronecker product of n basic transform matrices and C^{ω} is then calculated by multiplying the transform matrix with appropriate vector. In the latter case the transform matrix is first factorized into product of n sparse transform matrices and the factorized form is then multiplied with the appropriate vector to obtain C^{ω}. For example, when $n = 3$, C^{ω} can be calculated from $\vec{F}$ either by direct calculation

$$\left(C^{\omega}\right)^T = \left(S_1^{<j_1>} \otimes S_1^{<j_2>} \otimes S_1^{<j_3>}\right) \cdot \vec{F}$$

or by the fast transform

$$\left(C^{\omega}\right)^T = (S_1^{<j_1>} \otimes I \otimes I) \cdot (I \otimes S_1^{<j_2>} \otimes I) \cdot (I \otimes I \otimes S_1^{<j_3>}) \cdot \vec{F},$$

where I denotes the identity matrix of size 3×3.

Let $N_y(R)$ denote the number of entries in matrix R whose values are equal to y ($0 \le y \le 4$). From Table 3.2, it can be seen that a particular basic transform matrix $S_1^{<j>}$ can be obtained from the matrix $S_1^{<j>}$ with any other value of j by simply permuting the columns. As a result, the value of $N_y\left(S_1^{<\omega>}\right)$ for a particular value of y does not change with ω. It follows that $N_y\left(S_n^{<\omega>}\right)$ value is also independent of ω and is only affected by the value of n.

The value of $\sum_{y=1}^{4} N_y\left(S_1^{<\omega>}\right)$ is equal to 18. Since multiplication of two nonzero numbers in GF(5) always produces another nonzero number, it can be easily derived

that there are 18^n nonzero elements in $S_n^{<\omega>}$. In addition, it can also be obtained that the values of $N_1\left(S_n^{<\omega>}\right)$, $N_2\left(S_n^{<\omega>}\right)$, $N_3\left(S_n^{<\omega>}\right)$, and $N_4\left(S_n^{<\omega>}\right)$ are $\frac{1}{4}\left[18^n+2\cdot(-4)^n+10^n\right]$, $\frac{1}{4}\left[18^n-10^n\right]$, $\frac{1}{4}\left[18^n-10^n\right]$, and $\frac{1}{4}\left[18^n-2\cdot(-4)^n+10^n\right]$, respectively. Let $A_1(n)$ and $M_1(n)$ denote the numbers of required additions and multiplications for executing (3.5) by direct calculation. Similarly, let $A_2(n)$ and $M_2(n)$ be the numbers of addition and multiplication operations performed in deriving (3.5) by fast transform. Then their numbers can be easily derived as follows

$$A_1(n)=18^n-5^n \tag{3.12}$$

$$\begin{aligned} M_1(n) &= N_2\left(S_n^{<\omega>}\right)+N_3\left(S_n^{<\omega>}\right)+N_4\left(S_n^{<\omega>}\right) \\ &= \frac{1}{4}\left(3\cdot 18^n-2\cdot(-4)^n-10^n\right) \end{aligned} \tag{3.13}$$

$$\begin{aligned} A_2(n) &= n\cdot\left(18\cdot 5^{n-1}-5^n\right) \\ &= 13n\cdot 5^{n-1} \end{aligned} \tag{3.14}$$

$$\begin{aligned} M_2(n) &= n\cdot\left(13\cdot 5^{n-1}\right) \\ &= A_2(n). \end{aligned} \tag{3.15}$$

In a similar manner, from Table 3.2 and (3.8), it can be derived that there are $\frac{1}{4}\left(21^n+2\cdot 9^n+13^n\right)$ 1s, $\frac{1}{4}\left(21^n-13^n\right)$ 2s, $\frac{1}{4}\left(21^n-13^n\right)$ 3s, and $\frac{1}{4}\left(21^n-2\cdot 9^n+13^n\right)$ 4s in $T_n^{<\omega>}$ with $\sum_{y=1}^{4} N_y\left(T_n^{<\omega>}\right)=21^n$. Therefore, the number of necessary additions and multiplications for performing (3.6) are

$$A_3(n)=21^n-5^n \tag{3.16}$$

$$\begin{aligned} M_3(n) &= N_2\left(T_n^{<\omega>}\right)+N_3\left(T_n^{<\omega>}\right)+N_4\left(T_n^{<\omega>}\right) \\ &= \frac{1}{4}\left(3\cdot 21^n-2\cdot 9^n-13^n\right) \end{aligned} \tag{3.17}$$

for direct calculation and

$$A_4(n) = n \cdot \left(21 \cdot 5^{n-1} - 5^n\right)$$

$$= 16n \cdot 5^{n-1} \tag{3.18}$$

$$M_4(n) = n \cdot \left(8 \cdot 5^{n-1}\right)$$

$$= A_4(n)/2 \tag{3.19}$$

for fast transform calculation.

The computational cost values of (3.5) and (3.6) are listed in Table 3.4 for $n < 6$. As expected, the calculations using the fast transform incur significantly smaller computational costs compared to the direct calculations.

Table 3.4: Computational costs for forward and inverse FPRME transform.

n	Forward				Inverse			
	Direct		Fast		Direct		Fast	
	$A_1(n)$	$M_1(n)$	$A_2(n)$	$M_2(n)$	$A_3(n)$	$M_3(n)$	$A_4(n)$	$M_4(n)$
1	13	13	13	13	16	8	16	8
2	299	210	130	130	416	248	160	80
3	5707	4156	975	975	9136	6032	1200	600
4	104351	76104	6500	6500	193856	135440	8000	4000
5	1886443	1392688	40625	40625	4080976	2940728	50000	25000

The computational costs of (3.10) and (3.11) are the same since they use the same transform matrices. In order to obtain them, let us first divide the basic transform matrices $Z_1^{<j>}$ into three groups based on the distribution of the nonzero elements inside them and analyze each group separately. Let $\alpha_0 = \{0\}$, $\alpha_1 = \{1, 4\}$, and $\alpha_2 = \{2, 3\}$. When $\omega \in \alpha_0$, $Z_1^{<\omega>}$ only has five nonzero elements, all of which have values of one. When $\omega \in \alpha_1$, $N_1\left(Z_1^{<\omega>}\right) = 10$, $N_2\left(Z_1^{<\omega>}\right) = 1$, and $N_3\left(Z_1^{<\omega>}\right) =$

$N_4\left(Z_1^{<\omega>}\right) = 2$. Finally, when $\omega \in \alpha_2$ the values of $N_1\left(Z_1^{<\omega>}\right)$, $N_2\left(Z_1^{<\omega>}\right)$, $N_3\left(Z_1^{<\omega>}\right)$, and $N_4\left(Z_1^{<\omega>}\right)$ are 7, 3, 2, and 3, respectively.

Let Q_m^d be a $5^m \times 5^m$ matrix that is obtained from Kronecker product of m basic matrices whose polarities belong to α_d ($d \in \{0, 1, 2\}$), i.e., $Q_m^d \in \left\{Z_m^{<\omega>} \mid <\omega>_{10} =< j_1, j_2, \ldots, j_m >_5 \text{ and } j_i \in \alpha_d \forall i \in \{1, \ldots, m\}\right\}$. Then when $d = 0$, Q_m^d has 5^m 1s with the rest of the elements zero. When $d = 1$, the numbers of 1s, 2s, 3s, and 4s in the matrix Q_m^d are not zero and they are related by

$$N_1(Q_m^1) = N_3(Q_{m+1}^1) - 8 \cdot N_3(Q_m^1) - \frac{3}{2} \cdot 15^m + \frac{1}{2} \cdot 9^m \tag{3.20}$$

$$N_2(Q_m^1) = \frac{1}{2}\left(15^m - 9^m\right) - N_3(Q_m^1) \tag{3.21}$$

$$N_3(Q_m^1) = 16 \cdot N_3(Q_{m-1}^1) - 65 \cdot N_3(Q_{m-2}^1) + \frac{25}{2} \cdot 15^{m-2} - \frac{1}{2} \cdot 9^{m-2} \tag{3.22}$$

$$N_4(Q_m^1) = \frac{1}{2}\left(15^m + 9^m\right) - N_1(Q_m^1). \tag{3.23}$$

Equation (3.22) is a nonhomogeneous second order linear recurrence equation [Ros99]. Solving all the equations, it is found that there are 15^m nonzero elements in Q_m^1 with $N_1(Q_m^1) = \frac{1}{4}\left(2\sqrt{65}^{m+1} \sin((m+1)\mu) - 16\sqrt{65}^{m} \sin(m\mu) + 15^m + 9^m\right)$, $N_2(Q_m^1) = \frac{1}{4}\left(15^m - 9^m - 2\sqrt{65}^{m} \sin(m\mu)\right)$, $N_3(Q_m^1) = \frac{1}{4}\left(15^m - 9^m + 2\sqrt{65}^{m} \sin(m\mu)\right)$, and $N_4(Q_m^1) = \frac{1}{4}\left(16\sqrt{65}^{m} \sin(m\mu) - 2\sqrt{65}^{m+1} \sin((m+1)\mu) + 15^m + 9^m\right)$, where $\mu = \arctan(0.125)$.

In a similar way, it can be derived that for $d = 2$ the relations between the numbers of 1s, 2s, 3s, and 4s in the matrix Q_m^d are given by

$$N_1(Q_m^2) = 4 \cdot N_3(Q_m^2) - N_3(Q_{m+1}^2) + 3 \cdot 15^m \tag{3.24}$$

$$N_2(Q_m^2) = \frac{1}{2}\left[15^m - 5^m\right] - N_3(Q_m^2) \tag{3.25}$$

$$N_3(Q_m^2) = 8 \cdot N_3(Q_{m-1}^2) - 17 \cdot N_3(Q_{m-2}^2) + \frac{61}{2} \cdot 15^{m-2} - \frac{1}{2} \cdot 5^{m-2} \tag{3.26}$$

$$N_4(Q_m^2) = \frac{1}{2}\left[15^m + 5^m\right] - N_1(Q_m^2). \tag{3.27}$$

Solving the equations gives us the values of $\frac{1}{4}\left(5^m + 15^m - 8\sqrt{17}^m \sin(m\alpha) + 2\sqrt{17}^{m+1} \sin((m+1)\alpha)\right)$, $\frac{1}{4}\left(15^m - 5^m + 2\sqrt{17}^m \sin(m\alpha)\right)$, $\frac{1}{4}\left(15^m - 5^m - 2\sqrt{17}^m \sin(m\alpha)\right)$, and $\frac{1}{4}\left(5^m + 15^m + 8\sqrt{17}^m \sin(m\alpha) - 2\sqrt{17}^{m+1} \sin((m+1)\alpha)\right)$ for $N_1(Q_m^2)$, $N_2(Q_m^2)$, $N_3(Q_m^2)$, and $N_4(Q_m^2)$, respectively with the same total number of 15^m nonzero elements and α = arc tan(0.25).

Based on the numbers given above the computational cost of (3.10) and (3.11) can now be calculated. Let the transform matrix $Z_n^{<\omega>}$ be constructed from Kronecker product of v basic matrices whose polarities belong to α_1, u basic matrices whose polarities belong to α_2, and $n - (u + v)$ $Z_1^{<0>}$. Then the computational cost of the direct calculation of (3.10) and (3.11) is

$$A_5(n) = 5^n(3^w - 1) \tag{3.28}$$

and $$M_5(n) = 5^{n-w}(15^w - (N_1(Q_v^1) \cdot N_1(Q_u^2) + N_4(Q_v^1) \cdot N_4(Q_u^2) + N_2(Q_v^1) \cdot N_3(Q_u^2) + N_3(Q_v^1) \cdot N_2(Q_u^2)))$$

$$= \frac{5^{n-w}}{4}\left(3 \cdot 15^w - 5^u \cdot 9^v + B - (2\sqrt{17}^u \sin(u\alpha) \cdot \sqrt{65}^v \sin(v\mu)\right) \tag{3.29}$$

additions and multiplications, respectively where $w = u + v$ and $B = \left(\sqrt{65}^{v+1} \sin((v+1)\mu) - 8\sqrt{65}^v \sin(v\mu)\right)\left(8\sqrt{17}^u \sin(u\alpha) - 2\sqrt{17}^{u+1} \sin((u+1)\alpha)\right)$.

The corresponding numbers for the calculation of (3.10) and (3.11) using fast transform are

$$A_6(n) = 2w \cdot 5^n \tag{3.30}$$

and

$$M_6(n) = (5w + 3u) \cdot 5^{n-1}, \tag{3.31}$$

respectively

3.3 Matrix multiplication algorithms

Based on the derived computational costs in Section 3.2, it can be seen that it is more computationally efficient to generate all spectral coefficient vectors in some sequence where the current spectral coefficient vector is obtained from the previous polarity spectral coefficient vector by (3.11) rather than calculating all of them from either the spectral coefficient vector in polarity zero by (3.10) or the truth vector by (3.5). In this section, three algorithms that generate all spectral coefficient vectors in such manner are considered. The first algorithm employs lexicographic sequence [Gre89, Gre90] and the direct version of (3.11). The second algorithm also uses lexicographic sequence but in combination with the fast version of (3.11). The last algorithm applies (3.11) along extended dual polarity route, which is an ordering of all polarities such that any two consecutive polarities are extended dual polarities (see Definition 3.1.6).

The first polarity in the lexicographic order is polarity zero. Hence, when all the 5^n polarities spectral coefficient vectors are generated in lexicographic order, the spectral coefficient vector in polarity zero need to be first calculated from the truth vector by (3.5) if the input is the truth vector representation of a five-valued function. After it is obtained, all the other spectral coefficient vectors are then calculated by (3.11). In the calculation of the spectral coefficient vectors in nonzero polarities along the lexicographic order, u is always zero and there are $4 \cdot 5^{n-q}$ instances where the transform matrix $Z_n^{<a+4b>}$ has $v = q$ ($1 \le q \le n$). Substituting this into (3.28)–(3.31), the total calculation cost for generating the spectral coefficient vectors in all the $5^n - 1$ nonzero polarities by the first algorithm is

$$A_7(n) = \sum_{i=1}^{n} 4 \cdot 5^{n-i} \cdot A_5(n)$$

$$= 5^n\left(5^{n+1} - 2 \cdot 3^{n+1} + 1\right) \text{ additions} \tag{3.32}$$

and $$M_7(n) = \sum_{i=1}^{n} 4 \cdot 5^{n-i} \cdot M_5(n)$$

$$= \frac{1}{4640}\begin{pmatrix} 13950 \cdot 25^n - 20880 \cdot 15^n + 2610 \cdot 9^n + \\ 4320\sqrt{65}^{n+1} \sin((n+1)\mu) - 544\sqrt{65}^{n+2} \sin(n\mu) \end{pmatrix} \tag{3.33}$$

multiplications whereas the corresponding computational cost for the second matrix multiplication algorithm is

$$A_8(n) = \frac{5^n}{2}\left(5^{n+1} - 4n - 5\right) \text{ additions} \tag{3.34}$$

and $$M_8(n) = \frac{5^n}{4}\left(5^{n+1} - 4n - 5\right) \text{ multiplications.} \tag{3.35}$$

There is always more than one possible extended dual polarity routes for a given number of variables, which means that there are many possible implementations of the third matrix multiplication algorithm. The chosen extended dual polarity route affects the computational cost of the algorithm, and hence its computational speed. For all the possible extended dual polarity routes, any two consecutive polarities along the routes are always the extended dual polarity of each other, and hence the transform matrix $Z_n^{<a+4b>}$ always has $w = 1$. By (3.30) and (3.31) the number of required additions for the algorithm is always the same regardless of the extended dual polarity route used, whereas the number of the required multiplication is affected by the employed route. The computational cost in terms of multiplication number is minimal when the nonzero bit $<a+4b>$ is always 1 or 4 along the employed route. On the other hand, when the chosen route causes the nonzero bit $<a+4b>$ to be always 2 or 3, the worst-case multiplication number for the algorithm occurs. Procedures that generate all FPRME spectral coefficient vectors by the third algorithm in one possible best-case and worst-case extended dual polarity routes over GF(5) are given in Figs. 3.1 and 3.2, respectively.

```
Void fprm(int truth vector[ ])
{int W[n] = <0, 0, ….,0>, p[ ];
 char Dir[n] = 'aaa…a';
 Calculate C^0 from truth vector using (3.5) and
 store the result in p[ ];
 for(j = 0 to 5^n – 2)
    {cont = true;
     l = n;
     while (cont==true)
      { l = l --;
        cont = false;
        if(Dir[l]=='a')
          { switch (W[l])
             { case 0: W[l] = 1; break;
               case 1: W[l] = 2; break;
               case 2: W[l] = 3; break;
               case 3: W[l] = 4; break;
               case 4: { cont = true; Dir[l] = 'd';}} }
        else
          { switch (W[l])
             { case 4: W[l] = 3; break;
               case 3: W[l] = 2; break;
               case 2: W[l] = 1; break;
               case 1: W[l] = 0; break;
               case 0: { cont=true; Dir[l] = 'a';}} }}
      Calculate C^W from p[ ] and store the result
      back to p[ ];}}
```

Figure 3.1: Procedure *fprm()* with one possible best-case extended dual polarity route.

```
Void fprm(int truth vector[ ])
{int W[n] = <4, 4, …., 4>, p[ ];
 char Dir[n] = 'aaa…a';
 Calculate C^0 from truth vector using (3.5) and
 store the result in p[ ];
 for(j = 0 to 5^n – 2)
    { cont = true;
      l = n;
      while (cont==true)
        { l = l --;
          cont = false;
          if(Dir[l]=='a')
           { switch (W[l])
              { case 0: W[l] = 3; break;
                case 1: { cont = true; Dir[l] = 'd';}
                case 2: W[l] = 0; break;
                case 3: W[l] = 1; break;
                case 4: W[l] = 2; break;} }
          else
           { switch (W[l])
              { case 4: { cont=true; Dir[l] = 'a';}
                case 3: W[l] = 0; break;
                case 2: W[l] = 4; break;
                case 1: W[l] = 3; break;
                case 0: W[l] = 2; break;} }}
          Calculate C^W from p[ ] and store the result
          back to p[ ];}}
```

Figure 3.2: Procedure *fprm()* with one possible worst-case extended dual polarity route.

When one of the possible best-case extended dual polarity routes is used, the value of u in (3.31) is always zero. Hence, the computational cost of generating the spectral coefficient vectors in all the $5^n - 1$ polarities (excluding the starting polarity) by the third algorithm along the best-case extended dual polarity route is

$$A_9(n) = 2 \cdot 5^n (5^n - 1) \text{ additions} \tag{3.36}$$

and

$$M_9(n) = 5^n (5^n - 1) \text{ multiplications,} \tag{3.37}$$

which is lower than the computational cost of the other two algorithms. Similarly, the worst-case computational cost for the third algorithm can be obtained to be

$$A_{10}(n) = 2 \cdot 5^n (5^n - 1) \text{ additions} \tag{3.38}$$

and

$$M_{10}(n) = \frac{8}{5}(25^n - 5^n) \text{ multiplications.} \tag{3.39}$$

The computational costs for generating the FPRME spectra in all the 5^n polarities by the first and second matrix multiplication algorithms and by the procedures given in Figs. 3.1 and 3.2 for $n < 6$ are tabulated and presented in Table 3.5. The numbers include the cost for calculating the starting polarity spectral coefficient vector from truth vector by fast version of (3.5). The table shows that the computational cost of the third algorithm along best-case extended dual polarity route is the smallest among the algorithms.

3.4 Extended dual polarity algorithm

An algorithm that optimizes Kronecker expressions by introducing the term extended dual polarity was presented in [JSM02]. In [JSM03], the algorithm was extended for the optimization of FPRMEs over GF(4). In this section, an algorithm that uses extended dual polarity property for optimizing FPRMEs over GF(5) is introduced. The new algorithm can be implemented with low storage requirement.

Table 3.5: Computational cost of generating all FPRME coefficient vectors (matrix multiplication algorithms).

n	Additions				Multiplications			
	Lexicographic order		Extended dual polarity route		Lexicographic order		Extended dual polarity route	
	Direct	Fast	Best-case	Worst-case	Direct	Fast	Best-case	Worst-case
1	53	53	53	53	33	33	33	45
2	1930	1530	1330	1330	1098	830	730	1090
3	58975	38975	31975	31975	33595	19975	16475	25775
4	1656500	976500	786500	786500	959888	491500	396500	630500
5	44315625	24415625	19565625	19565625	26039025	12228125	9803125	15660625

Definition 3.4.1. [JSM02] For a p-valued n-variable function, extended dual polarity route is an ordering of all p^n polarities in which each two successive polarities are extended dual polarities.

As has been mentioned in Section 3.3, there are always more than one possible extended dual polarity routes for a given function. Two procedures that generate extended dual polarity routes that result in best-case and worst-case computational cost for the shown matrix multiplication algorithm have been given in Figs. 3.1 and 3.2, respectively. Fig. 3.3 gives another procedure that can be used to generate another extended dual polarity route for an n-variable five-valued function. The procedure is obtained by modifying the procedures given in [JSM02] and [JSM03] for Kronecker expressions and FPRMEs over GF(4), respectively. The initial values for the variables *level* and *direction* inside the procedure are zero.

```
Void route(int level, int direction)
{   if( direction == 0)
   { if( level == no_variable)
     {  -- out new polarity vector h }
     else
     {  h[level] = 0;
        route(level+1,0);
        h[level] = 1;
        route(level+1,1);
        h[level] = 2;
        route(level+1,0);
        h[level] = 3;
        route(level+1,1);
        h[level] = 4;
        route(level+1,0);}}
   else //direction == 1
   {if( level == no_variable)
     {  -- out new polarity vector h }
     else
     {  h[level] = 4;
        route(level+1,1);
        h[level] = 3;
        route(level+1,0);
        h[level] = 2;
        route(level+1,1);
        h[level] = 1;
        route(level+1,0);
        h[level] = 0;
        route(level+1,1);}}}
```

Figure 3.3: Procedure *route*.

3.4.1 Generation of FPRME in polarity zero from truth vector

Let us represent each product term $c_i^{\omega}\hat{x}_1^{k_1}\hat{x}_2^{k_2}...\hat{x}_n^{k_n}$ in the FPRME by an (n+1)-digit five-valued string '$k_1k_2\ldots k_n\, c_i^{\omega}$' and call it "term". Based on (3.5), a recursive algorithm that generates the terms of FPRME in polarity zero from the truth vector is constructed. Given a truth vector, the algorithm first replaces each truth vector element f_i ($0 \le i \le 5^n - 1$) by an (n+1)-digit five-valued string, i.e. truth vector term '$m_1m_2\ldots m_nf_i$' where $<m_1, m_2, \ldots, m_n>_5 = <i>_{10}$. The generated truth vector terms are then taken as the input for the first level recursion. The algorithm has n recursion levels. At each level, the output is taken as input for the next recursion. The output of the last level recursion is the terms for the FPRME in polarity zero.

The algorithm uses matrix M which is defined as:

$$M = \begin{bmatrix} 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix},$$

where the element that is located at the row y and column z ($1 \le y \le 3$, $1 \le z \le 4$) of matrix M is denoted by $M_{y,z}$.

The steps of the algorithm are:

Step 1: Generate truth vector terms for the given truth vector.

Step 2: Initialize l to n. Take the truth vector terms as the input.

Step 3: Generate initial terms '$k_1k_2\ldots k_n c_i^{\omega}$' for the output. The initial terms are all the 5^n terms ($0 \le i \le 5^n - 1$) with zero as the last digit.

Step 4: For each nonzero term (the term with nonzero last digit) of the input, generate its contribution to the output terms according to the rule given in Table 3.6. Note that the symbol q in Table 3.6 may represent k or m depending on the input terms.

Step 5: Sum up the initial and all contributed terms. The summation is done by replacing terms with identical first n-digit values with a new term. The first n-digits of the new term are equal to those of replaced terms, while the last digit of the new term is the summation of all the replaced terms' last digits over GF(5). After the summation the output terms are obtained.

Step 6: Decrement l by 1. If l = 0 stop. Otherwise go back to Step 3 with current output as the input.

Table 3.6: Contributed terms rule for generation of polarity zero terms.

Processed term	Contributed terms
$q_1 \ldots q_{l-1}0\ q_{l+1} \ldots\ q_n c$	$q_1 \ldots\ q_{l-1}0 q_{l+1} \ldots\ q_n\ c$
	$q_1 \ldots\ q_{l-1}4 q_{l+1} \ldots\ q_n\ M_{3,c}$
$q_1 \ldots\ q_{l-1}1\ q_{l+1} \ldots\ q_n c$	$q_1 \ldots\ q_{l-1}1 q_{l+1} \ldots\ q_n\ M_{3,c}$
	$q_1 \ldots\ q_{l-1}2 q_{l+1} \ldots\ q_n\ M_{3,c}$
	$q_1 \ldots\ q_{l-1}3 q_{l+1} \ldots\ q_n\ M_{3,c}$
	$q_1 \ldots\ q_{l-1}4 q_{l+1} \ldots\ q_n\ M_{3,c}$
$q_1 \ldots\ q_{l-1}2\ q_{l+1} \ldots\ q_n c$	$q_1 \ldots\ q_{l-1}1 q_{l+1} \ldots\ q_n\ M_{1,c}$
	$q_1 \ldots\ q_{l-1}2 q_{l+1} \ldots\ q_n\ c$
	$q_1 \ldots\ q_{l-1}3 q_{l+1} \ldots\ q_n\ M_{2,c}$
	$q_1 \ldots\ q_{l-1}4 q_{l+1} \ldots\ q_n M_{3,c}$
$q_1 \ldots\ q_{l-1}3\ q_{l+1} \ldots\ q_n c$	$q_1 \ldots\ q_{l-1}1 q_{l+1} \ldots\ q_n\ M_{2,c}$
	$q_1 \ldots\ q_{l-1}2 q_{l+1} \ldots\ q_n\ c$
	$q_1 \ldots\ q_{l-1}3 q_{l+1} \ldots\ q_n\ M_{1,c}$
	$q_1 \ldots\ q_{l-1}4 q_{l+1} \ldots\ q_n\ M_{3,c}$
$q_1 \ldots\ q_{l-1}4\ q_{l+1} \ldots\ q_n c$	$q_1 \ldots\ q_{l-1}1 q_{l+1} \ldots\ q_n\ c$
	$q_1 \ldots\ q_{l-1}2 q_{l+1} \ldots\ q_n\ M_{3,c}$
	$q_1 \ldots\ q_{l-1}3 q_{l+1} \ldots\ q_n\ c$
	$q_1 \ldots\ q_{l-1}4 q_{l+1} \ldots\ q_n\ M_{3,c}$

Example 3.4.1. Let the input function $f(\vec{x})$ be a two-variable five-valued function with the following truth vector: $\vec{F}$ =[0, 0, 2, 1, 0, 3, 0, 4, 0, 0, 0, 0, 2, 1, 0, 3, 0, 4, 0, 0, 0, 0, 0, 0, 0]. Then the truth vector terms of the input function are: {000, 010, 022, 031, 040, 103, 110, 124, 130, 140, 200, 210, 222, 231, 240, 303, 310, 324, 330, 340, 400, 410, 420, 430, 440}.

At the first level recursion (l = 2) the contribution of each nonzero term to the output terms are as follows: {022→014, 022, 031, 043; 031→013, 021, 032, 044; 103→103, 142; 124→113, 124, 132, 141; 222→214, 222, 231, 243; 231→213, 221, 232, 244; 303→303, 342; 324→313, 324, 332, 341}. Summing up the initial and contributed terms, the output terms of the first level recursion are obtained as {000, 012, 023, 033, 042, 103, 113, 124, 132, 143, 200, 212, 223, 233, 242, 303, 313, 324, 332, 343, 400, 410, 420, 430, 440}.

3.4.2 Calculation of FPRMEs in all nonzero polarities

Let a and b be two arbitrary polarity numbers such that $< a >_{10} =< a_1, a_2, \ldots, a_n >_5$ and $< b >_{10} =< b_1, b_2, \ldots, b_n >_5$. Then, according to (3.11), the two FPRME spectral coefficient vectors in polarity numbers a and b are related as follows:

$$\left(C^a\right)^T = \left(\otimes \prod_{l=1}^{n} \left(S_1^{<a_l>} \cdot T_1^{<b_l>} \right) \right) \cdot \left(C^b\right)^T . \tag{3.40}$$

When the two polarity numbers a and b are extended dual polarities, (3.40) is simplified into

$$\left(C^a\right)^T = \left(I_{r-1} \otimes \left(S_1^{<a_r>} \cdot T_1^{<b_r>}\right) \otimes I_{n-r}\right) \cdot \left(C^b\right)^T , \tag{3.41}$$

where I_u denotes an identity matrix of size $5^u \times 5^u$ and $a_r \neq b_r$.

Based on the resulting transform matrix $S_1^{<a_r>} \cdot T_1^{<b_r>}$ for all possible combinations of a_r and b_r values, an algorithm for the generation of all 5^n FPRME spectral coefficient vectors is derived. In the algorithm, each FPRME is represented by its terms. Given the truth vector of a function, the algorithm first generates the terms of the FPRME in polarity zero by making use of the algorithm described in Section 3.4.1. It then generates the FPRME terms in all the nonzero polarities for the function one by one in a sequence determined by the extended dual polarity route. The terms for each FPRME in nonzero polarity are determined based on the terms of the FPRME in

previous polarity according to the relation between the digits of the current polarity a and the previous polarity b.

The steps of the algorithm are as follows:

Step 1: Initialization

- Initialize the polarity vector v to <0, 0, …, 0>.
- Initialize the polarity number ω to 0.
- Initialize the variables ω_{best} and C_{min} to 0.

Step 2: Calculate the terms of the FPRME in polarity zero using the algorithm given in Section 3.4.1. Assign the number of nonzero terms inside the FPRME in polarity zero to C_{min}.

Step 3: Determine the next polarity vector v in the employed extended dual polarity route and set ω to the decimal equivalent of v.

Step 4: Generate the terms of the FPRME in polarity ω.

- List initial terms of polarity ω.
- Find the contribution of each nonzero terms of previous polarity according to the rules given in Tables 3.7 and 3.8.
- Sum up all the terms for the polarity ω.
- If the number of nonzero terms of polarity ω, N is less than current value of C_{min}, set $\omega_{best} = \omega$ and $C_{min} = N$.

Step 5: If the terms for all polarities have been generated exit the algorithm. Otherwise go back to Step 3.

At the end of the algorithm, all 5^n FPRMEs for the given function are obtained. Moreover, the polarity number stored in ω_{best} is the optimal polarity number with C_{min} nonzero spectral coefficients.

It should be noted that when the generation of the FPRMEs is done along the route produced by the procedure in Fig. 3.3, the rules given in Table 3.7 are sufficient. However the algorithm can also be performed along other extended dual polarity routes, for which other combinations of a_r and b_r values may be encountered and so the rest of the rules are given in Table 3.8.

Table 3.7: Contribution of processed term for some combinations of a_r and b_r values.

a_r	b_r	Processed term	Contributed terms
0 1 2 3 4	4 0 1 2 3	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n c$
		$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$
0 1 2 3 4	1 2 3 4 0	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$

Table 3.8: Contribution of processed term for other combinations of a_r and b_r values.

a_r	b_r	Processed term	Contributed terms
0 1 2 3 4	3 4 0 1 2	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$
0 1 2 3 4	2 3 4 0 1	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ c$
		$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$	$k_1 \ldots k_{r-1} 0 k_{r+1} \ldots k_n\ c$
			$k_1 \ldots k_{r-1} 1 k_{r+1} \ldots k_n\ M_{1,c}$
			$k_1 \ldots k_{r-1} 2 k_{r+1} \ldots k_n\ M_{3,c}$
			$k_1 \ldots k_{r-1} 3 k_{r+1} \ldots k_n\ M_{2,c}$
			$k_1 \ldots k_{r-1} 4 k_{r+1} \ldots k_n\ c$

3.5 Cube polarity adjustment algorithm

In this section, an algorithm that converts the disjoint cubes reduced representation of a five-valued function into its FPRME spectral coefficients is presented. Similar to other spectral algorithms that start from the disjoint cubes reduced representation of the input functions [FC99, FSP92a], the main advantage of the presented algorithm is that it can be implemented with reduced memory space compared to other algorithms. This is due to the fact that for many practical functions the number of disjoint cubes is much smaller than the number of minterms [CYP89]. The algorithm also has another advantage in that it may be implemented using parallel programming since inside it each spectral coefficient is calculated independently from other spectral coefficients.

3.5.1 Basic definitions and properties for cube polarity adjustment algorithm

Definition 3.5.1. The FPRME in polarity ω for an n-variable five-valued function $f(\vec{x})$ can also be represented by

$$f(\vec{x}) = \sum_{i=0}^{5^n-1} c_i{}^{\omega} \pi_i{}^{\omega},$$

where each piterm $\pi_i{}^{\omega}$ is equivalent to $\left[\prod_{l=1}^{n} \hat{x}_l{}^{k_l}\right]$ in (3.4).

Property 3.5.1. There are 5^n possible $\pi_i{}^{\omega}$ for a particular polarity number that correspond to assignments of different permutations of k_l.

Property 3.5.2. Each product term $c_i{}^{\omega}\pi_i{}^{\omega}$ in FPRME is characterized by the spectral coefficient c_i^{ω}, the polarity number ω, and the piterm index $< k_1, k_2, \ldots, k_n >$, where $< i >_{10} = < k_1, k_2, \ldots, k_n >_5$.

Definition 3.5.2. If a disjoint cube has zero output value, the cube is an OFF-cube. Otherwise, the cube is an ON-cube.

Definition 3.5.3. The cube ω-adjustment operator is a bit operator that takes two values from the following set: {0, 1, 2, 3, 4, '–'}. It is similar to the GF(5) bit-by-bit addition operator except for the case when one or both of the input(s) is (are) '–' (the symbol for missing literal in cubes). This operation is shown in Table 3.9.

Table 3.9: Cube ω-adjustment operation.

ω-*adj*	0	1	2	3	4	–
0	0	1	2	3	4	–
1	1	2	3	4	0	–
2	2	3	4	0	1	–
3	3	4	0	1	2	–
4	4	0	1	2	3	–
–	–	–	–	–	–	–

Definition 3.5.4. The cube π-adjustment operator is another bit operator that takes two inputs. This operator depends not only on the input values but also on the order of the inputs. The output values of $a_l \pi\text{-}adj k_l$ for all possible values of a_l and k_l are shown in Table 3.10.

Definition 3.5.5. Let m_r, the r-th ON cube be of the form $m_{r1}m_{r2}\ldots m_{rn}$ ($m_{rl} \in$ {0, 1, 2, 3, 4, '– '}, $1 \le l \le n$) with output value $f(r)$ ($1 \le f(r) \le 4$). Then the contribution of m_r to the value of a particular spectral coefficient c_i^{ω} is equal to the result of multiplication of $f(r)$ with the digits of $(m_r \omega\text{-}adj \omega) \pi\text{-}adj k$. Mathematically,

$$h_r^i = g_r^i \cdot f(r), \tag{3.42}$$

where $g_r^i = \left(\prod_{l=1}^{n} a_{rl_{\pi-adj}} k_l \right)$, $a_{rl} = m_{rl\omega\text{-}adj}\omega_l$, and h_r^i denotes the contribution of the r-th ON disjoint cube to the value of a particular spectral coefficient c_i^ω.

Table 3.10: Cube π-adjustment operation.

π-adj	k_l				
a_l	0	1	2	3	4
0	1	0	0	0	4
1	0	4	4	4	4
2	0	2	1	3	4
3	0	3	1	2	4
4	0	1	4	1	4
–	1	0	0	0	0

Definition 3.5.6. The final value of c_i^ω is the summation of all contributions to it over GF(5). Hence,

$$c_i^\omega = \sum_{r=1}^{z} h_r^{\,i}, \tag{3.43}$$

where z is the number of ON disjoint cubes inside the input function.

Property 3.5.3. The number of h_r^i required to obtain all spectral coefficients of the FPRME in the polarity ω is equal to the number of ON disjoint cubes inside the input function multiplied by 5^n.

3.5.2 Algorithm for calculation of FPRME spectral coefficients over GF(5) from disjoint cubes reduced representation

Given disjoint cubes representation of an n-variable five-valued function, the value of

a particular spectral coefficient c_i^{ω} can be calculated by making use of the above mentioned definitions. The steps of the calculation algorithm are

Step 1: Generate the n-digit five-valued representation of ω, $< j_1, j_2, \ldots, j_n >$.

Step 2: Generate the piterm index of c_i^{ω}, $< k_1, k_2, \ldots, k_n >$.

Step 3: For each ON disjoint cube m_r, find out $a_r = m_{r\omega\text{-}adj}\omega$.

Step 4: For each resulting a_r, find its contribution to the value of c_i^{ω}, h_r^i according to Definition 3.5.5.

Step 5: The value of c_i^{ω} is the summation of all h_r^i found in Step 4 over GF(5).

Based on the algorithm above, calculation of a spectral coefficient value at worst-case condition involves $(z-1)$ additions and $n \cdot z$ multiplications, where z is the number of ON disjoint cubes of the input function. The amount of the required multiplications can be reduced by first grouping the disjoint cubes according to their output values as described by the following property.

Property 3.5.4. Let z_1, z_2, z_3, and z_4 be the number of ON disjoint cubes with output values 1, 2, 3, and 4, respectively. Then the computational cost of the algorithm in terms of multiplication number can be reduced by first grouping the ON cubes with the same output values together and then arranging the groups in ascending order with respect to the output values. Such arrangement of ON cubes allows c_i^{ω} to be calculated as

$$c_i^{\omega} = \left(\sum_{r=1}^{z_1} {g_r}^i\right) + \left(\left(\sum_{r=z_1+1}^{z_1+z_2} {g_r}^i\right) \cdot 2\right) + \left(\left(\sum_{r=z_1+z_2+1}^{z_1+z_2+z_3} {g_r}^i\right) \cdot 3\right) + \left(\left(\sum_{r=z_1+z_2+z_3+1}^{z_1+z_2+z_3+z_4} {g_r}^i\right) \cdot 4\right) \quad (3.44)$$

which reduces the worst-case multiplication number to $z \cdot (n-1)+3$.

Example 3.5.1. Let a four-variable five-valued function be represented by the following set of ON cubes: f = {44–2 3, 1322 2, 0432 2, 3–01 3, 123– 4, 0123 1}. Then the computation process of c_{227}^{42} according to the given algorithm and Property 3.5.4 is

- Rearrange the order of the ON cubes $\rightarrow f$ = {0123 1, 1322 2, 0432 2, 44–2 3, 3–01 3, 123– 4}.
- Generate the n-digit five-valued representation of $\omega \rightarrow <\omega>_{10} = <0,1,3,2>_5$.
- Generate the piterm index $\rightarrow <i>_{10} = <1,4,0,2>_5$.
- Calculate a_r for each ON cube $\rightarrow a_r$ = {0200, 1404, 0014, 40–4, 3–33, 131–}.
- Calculate g_r^i for each ON cube $\rightarrow g_r^i$ = {0, 4, 0, 1, 0, 0}.
- $c_{227}^{42} = (0) + ((4 + 0)\cdot 2) + ((1 + 0)\cdot 3) + (0\cdot 4) = 0 + (4\cdot 2) + (1\cdot 3) + 0 = 3 + 3 = 1$.

Hence, for the given function the value of c_{227}^{42} is 1.

3.6 Row polarity matrix algorithm

In this section, another algorithm, namely row polarity matrix algorithm, is presented which generates the complete FPRME polarity matrix of a five-valued function in an efficient and recursive manner based on a developed recursive equation of the polarity matrix. The recursive equation is obtained based on the relationship between any two spectra described by (3.11), the properties of the Kronecker matrix, as well as the addition and multiplication properties over GF(5) shown in Table 3.1. Owing to the efficiency of the recursive equations, the polarity matrix algorithm has relatively low computational costs and potential implementation using parallel programming. The algorithm can also be utilized to derive FPRMEs in specific polarities without first constructing the complete polarity matrix. The fast flow diagrams for computation of both complete and partial polarity matrix are given.

Similar to other polarity matrix algorithms [FHH+93, FR95, RF01], the approach used in generating the row polarity matrix algorithm is to first develop the initial recursive definition of the polarity matrix based on the desired starting point and the appropriate relations between the elements. The initial recursive definition is then optimized into equivalent recursive definition such that the latter incurs less number of arithmetic operations. The row polarity matrix algorithm starts from the first row elements of the polarity matrix (polarity zero spectrum) and the recursive equation is

based on the relations between the elements in different rows of the polarity matrix.

3.6.1 Basic definitions and properties for row polarity matrix algorithm

Definition 3.6.1. Let each FPRME spectral coefficient vector C^{ω} be decomposed into smaller equal size submatrices such that

$$C^{\omega} = [C^{\omega}_{[n-1,<0>]}, C^{\omega}_{[n-1,<1>]}, C^{\omega}_{[n-1,<2>]}, C^{\omega}_{[n-1,<3>]}, C^{\omega}_{[n-1,<4>]}]. \tag{3.45}$$

Property 3.6.1. Let the Definition 3.6.1 be recursive such that every subvector inside C^{ω} can be recursively decomposed into five smaller submatrices of equal length.

$$C^{\omega}_{[\beta,q]} = [C^{\omega}_{[\beta-1,<q,0>]}, C^{\omega}_{[\beta-1,<q,1>]}, C^{\omega}_{[\beta-1,<q,2>]}, C^{\omega}_{[\beta-1,<q,3>]}, C^{\omega}_{[\beta-1,<q,4>]}], \tag{3.46}$$

where $1 \le \beta \le n-1$ and $'<q,r>'(r \in \{0,1,2,3,4\})$ denotes the strings obtained by concatenating r to q.

Property 3.6.2. Since C^{ω} has 5^n elements, it follows from Definition 3.6.1 and Property 3.6.1 that each subvector $C^{\omega}_{[\beta,q]}$ has 5^{β} elements. Moreover, for a particular value of β, q is an $(n-\beta)$-digit five-valued number representation of any decimal number from zero to $5^{n-\beta}-1$.

Property 3.6.3. Let $<w>_{10} = <w_1, w_2, \ldots, w_n>_5$, $q = <q_1, q_2, \ldots, q_{n-\beta}>_5$, and W be a set defined by $W = \{w \mid w_i = q_i \forall i \in \{1,2,\ldots,n-\beta\}\}$ $(0 \le w \le 5^n-1)$. Then the elements of $C^{\omega}_{[\beta,q]}$ correspond to the polarity matrix $P[f(\vec{x})]$ elements with row index numbers ω and column index numbers belonging to W.

Definition 3.6.2. Let $D_{[\beta,q]}$ be defined as a vector of length 5^{β} that can be recursively constructed from five smaller equal length subvectors such that

$$D_{[\beta,q]} = \left[D_{[\beta-1,0]}, D_{[\beta-1,1]}, D_{[\beta-1,2]}, D_{[\beta-1,3]}, D_{[\beta-1,4]}\right], \tag{3.47}$$

where $1 \le \beta \le n$ and $q \in \{0, 1, 2, 3, 4\}$.

Definition 3.6.3. Let $P_\beta\left(D_{[\beta,q]}\right)$ be a square matrix of size $5^\beta \times 5^\beta$ that is given by the following recursive definition

$$P_\beta(D_{[\beta q]}) = \begin{bmatrix}
P_{\beta-1}\left(D_{[\beta-1,0]}\right) & P_{\beta-1}\left(D_{[\beta-1,1]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,0]} + 4D_{[\beta-1,1]} + D_{[\beta-1,2]} + 4D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,1]} + 3D_{[\beta-1,2]} + 3D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,0]} + 3D_{[\beta-1,1]} + 4D_{[\beta-1,2]} + 2D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,1]} + D_{[\beta-1,2]} + 2D_{[\beta-1,3]} + 3D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,0]} + 2D_{[\beta-1,1]} + 4D_{[\beta-1,2]} + 3D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,1]} + 4D_{[\beta-1,2]} + 2D_{[\beta-1,3]} + 2D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,0]} + D_{[\beta-1,1]} + D_{[\beta-1,2]} + D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,1]} + 2D_{[\beta-1,2]} + 3D_{[\beta-1,3]} + 4D_{[\beta-1,4]}\right)
\end{bmatrix}$$

$$\begin{bmatrix}
P_{\beta-1}\left(D_{[\beta-1,2]}\right) & P_{\beta-1}\left(D_{[\beta-1,3]}\right) & P_{\beta-1}\left(D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,2]} + 2D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,2]} + 4D_{[\beta-1,3]} + 4D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,3]} + 2D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,2]} + D_{[\beta-1,3]} + 4D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,3]} + 3D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,4]}\right) \\
P_{\beta-1}\left(D_{[\beta-1,2]} + 3D_{[\beta-1,3]} + D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,3]} + 4D_{[\beta-1,4]}\right) & P_{\beta-1}\left(D_{[\beta-1,4]}\right)
\end{bmatrix}, \tag{3.48}$$

where $D_{[\beta,q]}$ follows Definition 3.6.2.

Property 3.6.4. The first row of the matrix $P_\beta\left(D_{[\beta,q]}\right)$ is simply the vector $D_{[\beta,q]}$.

Property 3.6.5. From Definition 3.6.3 and Property 3.6.4, the matrix $P_\beta\left(D_{[\beta,q]}\right)$ has $5 \times 5 = 25$ elements when $\beta = 1$. Since by (3.48) each $P_1\left(D_{[1,q]}\right)$ is composed from 25 $P_0\left(D_{[0,q]}\right)$, it follows that each matrix $P_0\left(D_{[0,q]}\right)$ contains only a single element $D_{[0,q]}$. Thus, $P_0\left(D_{[0,q]}\right) = D_{[0,q]}$.

Property 3.6.6. When $\beta = n$ and $D_{[\beta,q]} = C^0$, the matrix $P_\beta\left(D_{[\beta,q]}\right)$ is equal to the

polarity matrix, $P[f(\vec{x})] = P_n(C^0)$.

Definition 3.6.4. Let (3.48) be rewritten as follows

$$P_\beta(D_{[\beta,q]}) = \begin{bmatrix} P_{\beta-1}(p_{00}) & P_{\beta-1}(p_{01}) & P_{\beta-1}(p_{02}) & P_{\beta-1}(p_{03}) & P_{\beta-1}(p_{04}) \\ P_{\beta-1}(p_{10}) & P_{\beta-1}(p_{11}) & P_{\beta-1}(p_{12}) & P_{\beta-1}(p_{13}) & P_{\beta-1}(p_{14}) \\ P_{\beta-1}(p_{20}) & P_{\beta-1}(p_{21}) & P_{\beta-1}(p_{22}) & P_{\beta-1}(p_{23}) & P_{\beta-1}(p_{24}) \\ P_{\beta-1}(p_{30}) & P_{\beta-1}(p_{31}) & P_{\beta-1}(p_{32}) & P_{\beta-1}(p_{33}) & P_{\beta-1}(p_{34}) \\ P_{\beta-1}(p_{40}) & P_{\beta-1}(p_{41}) & P_{\beta-1}(p_{42}) & P_{\beta-1}(p_{43}) & P_{\beta-1}(p_{44}) \end{bmatrix}. \quad (3.49)$$

Property 3.6.7. By Properties 3.1.1 and 3.6.4 and Definition 3.6.4, when $D_{[n,q]} = C^0$, $P_{n-1}(p_{0r})(0 \le r \le 5) = P_{n-1}(C^0_{[n-1,<r>]})$.

Definition 3.6.5. The numbers of necessary additions and multiplications for generating the elements of each recursive matrix $P_\beta(D_{[\beta,q]})$ can be further reduced. Let five intermediate vectors $I_{1[\beta-1]}$, $I_{2[\beta-1]}$, $I_{3[\beta-1]}$, $I_{4[\beta-1]}$, and $I_{5[\beta-1]}$ be introduced, where each has $5^{\beta-1}$ elements and $I_{1[\beta-1]} = p_{43} + p_{01}$, $I_{2[\beta-1]} = p_{22} + p_{33}$, $I_{3[\beta-1]} = p_{42} + p_{11}$, $I_{4[\beta-1]} = I_{1[\beta-1]} + p_{00}$, and $I_{5[\beta-1]} = I_{3[\beta-1]} + p_{01}$. Then using these intermediate vectors, (3.49) can be simplified into

$$P_\beta(D_{[\beta q]}) = \begin{bmatrix} P_{\beta-1}(p_{00}) & P_{\beta-1}(p_{01}) & P_{\beta-1}(p_{02}) & P_{\beta-1}(p_{03}) & P_{\beta-1}(p_{04}) \\ P_{\beta-1}(p_{40}+p_{21}+I_{5[\beta-1]}) & P_{\beta-1}(p_{41}+I_{2[\beta-1]}) & P_{\beta-1}(p_{32}+p_{23}) & P_{\beta-1}(p_{03}+p_{04}) & P_{\beta-1}(p_{04}) \\ P_{\beta-1}(I_{4[\beta-1]}+I_{5[\beta-1]}) & P_{\beta-1}(p_{32}+I_{1[\beta-1]}) & P_{\beta-1}(p_{42}+p_{33}) & P_{\beta-1}(p_{13}+p_{04}) & P_{\beta-1}(p_{04}) \\ P_{\beta-1}(p_{31}+I_{4[\beta-1]}) & P_{\beta-1}(p_{03}+I_{3[\beta-1]}) & P_{\beta-1}(p_{02}+p_{43}) & P_{\beta-1}(p_{23}+p_{04}) & P_{\beta-1}(p_{04}) \\ P_{\beta-1}(I_{4[\beta-1]}+I_{2[\beta-1]}) & P_{\beta-1}(p_{21}+p_{02}+p_{13}) & P_{\beta-1}(p_{12}+p_{03}) & P_{\beta-1}(p_{33}+p_{04}) & P_{\beta-1}(p_{04}) \end{bmatrix}. \quad (3.50)$$

3.6.2 Generation of complete polarity matrix

Two recursive definitions for polarity matrix $P[f(\vec{x})]$ have been given in Section 3.6.1. From the definitions, it can be observed that computing all elements of

$P_\beta\left(D_{[\beta,q]}\right)$ by (3.50) gives rise to smaller computational cost than generating them by (3.48). Thus, the row polarity matrix algorithm derives the complete polarity matrix by recursively applying (3.50). The algorithm starts by first calculating C^0 for the input function if it is not known. Once C^0 is obtained, the algorithm then decomposes C^0 into five equal length subvectors according to Definition 3.6.1. It continues by calculating the intermediate vectors and the first row elements of all the submatrices $P_{n-1}(p_{rs})$ $(r \in \{0,1,2,3\}, s \in \{0,1,2,3,4\})$ by (3.50). Once it is done, the algorithm then applies similar steps for each of the recently calculated $P_{n-1}(p_{rs})$ submatrices and repeats it recursively with increasingly smaller size of β until the complete polarity matrix is obtained. Note that as the rightmost column of submatrices are exactly the same, only one of them needs to be further processed. The flow diagram showing the computational sequence that is performed for each P_β matrix at each stage of the recursion is shown in Fig. 3.4. In summary, the steps for deriving the complete polarity matrix from C^0 by the row polarity matrix algorithm are:

Step 1: Initialize β to n.

Step 2: Divide the polarity matrix into P_β matrices. For each P_β matrix with known first row elements, generate the corresponding intermediate vectors. Then, calculate the first row elements of its $P_{\beta-1}(p_{10})$, $P_{\beta-1}(p_{11})$, $P_{\beta-1}(p_{12})$, $P_{\beta-1}(p_{13})$, $P_{\beta-1}(p_{20})$, $P_{\beta-1}(p_{21})$, $P_{\beta-1}(p_{22})$, $P_{\beta-1}(p_{23})$, $P_{\beta-1}(p_{30})$, $P_{\beta-1}(p_{31})$, $P_{\beta-1}(p_{32})$, and $P_{\beta-1}(p_{33})$ by (3.50) as shown by Fig. 3.4.

Step 3: If $\beta > 1$, decrement β by one and repeats Step 2.

Step 4: Complete the polarity matrix by copying all $P_{\beta-1}(p_{14})$, $P_{\beta-1}(p_{24})$, $P_{\beta-1}(p_{34})$, and $P_{\beta-1}(p_{44})$ from the appropriate $P_{\beta-1}(p_{04})$, starting from $\beta = 1$ up to $\beta = n$.

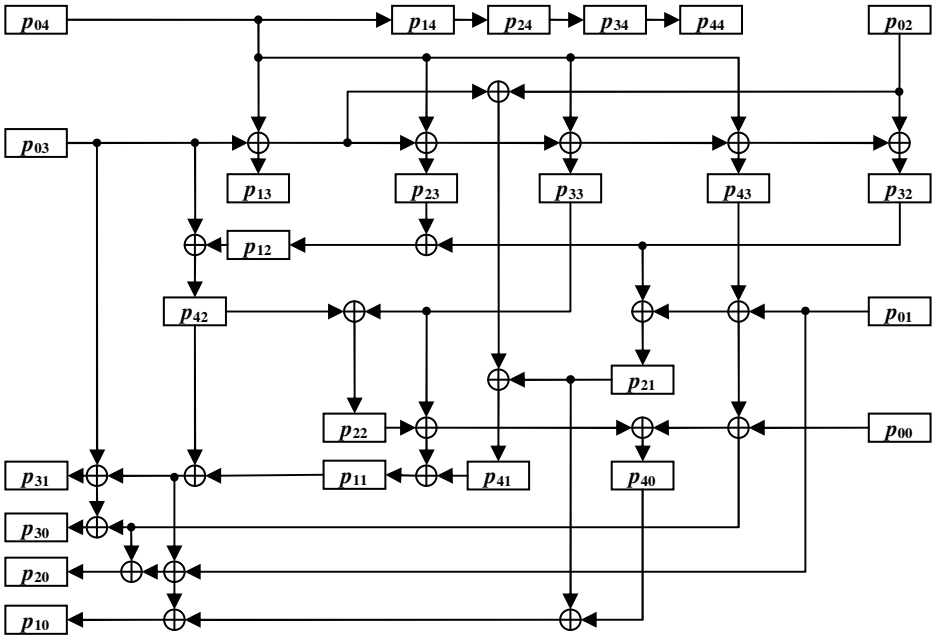


Figure 3.4: Flow diagram for the calculation in a recursion stage of row polarity matrix algorithm.

Property 3.6.8. The computational cost of the row polarity matrix algorithm for generating the complete polarity matrix for an n-variable function is $\frac{23}{16}\left(21^n - 5^n\right)$ additions and zero multiplications.

Proof: At the recursion when $\beta = n$, the first row elements of the 21 P_{n-1} matrices inside the polarity matrix are calculated and based on Fig. 3.4, the calculation requires 23 additions of size 5^{n-1} each. At the next recursion, each of the 21 recently calculated P_{n-1} matrices are divided into their corresponding P_{n-2} submatrices and the first row elements of 21 P_{n-2} submatrices inside each P_{n-1} matrices are calculated. As the generation of the first row elements of the 21 P_{n-2} submatrices inside one P_{n-1} matrix incurs 23 additions of size 5^{n-2}, at the second stage of the recursion there are $21 \cdot 23 \cdot 5^{n-2}$ additions performed.

In general, at the i-th recursion ($2 \le i \le n$) the polarity matrix is divided into 25^{i-1} P_{n+1-i} matrices, where only 21^{i-1} of them are unique and need to be further derived,

i.e., the first row elements of 21 of their P_{n-i} submatrices need to be calculated at the recursion stage. Therefore the computational cost contributed by the i-th recursion is $21^{i-1} \cdot 23 \cdot 5^{n-i}$ additions. The total computational cost to generate the complete polarity matrix of size $5^n \times 5^n$ is obtained as the summation of the computational cost at each stage,

$$\begin{aligned} A_{11}(n) &= \sum_{i=1}^{n} 23 \cdot 21^{i-1} \cdot 5^{n-i} \\ &= \frac{23}{16}\left(21^n - 5^n\right) \text{ additions.} \end{aligned}$$

3.6.3 Generation of selected spectral coefficient vector

Besides generating the complete polarity matrix, based on (3.48) the row polarity matrix algorithm may also be used to calculate a particular spectrum without first deriving the complete polarity matrix. At each recursion stage of the calculation, only the matrices that contain the required spectral coefficients need to be derived. In other words, only a row of $P_{\beta-1}$ matrices inside each P_β matrix need to be calculated as the matrices that are located at other rows do not contribute to the values of the spectral coefficient vector elements. The flow diagrams for the calculation of the first row elements of the rows 1, 2, 3, and 4 $P_{\beta-1}$ matrices from the first row of the corresponding P_β matrix are given in Figs. 3.5, 3.6, 3.7, and 3.8, respectively. In the figures, the symbols '$\oplus$' and '$\otimes$' represent GF(5) adder and multiplier, respectively.

In what follows, the properties of the calculation of a spectral coefficient vector by the row polarity matrix algorithm are established. For the properties, recall that at the i-th recursion ($1 \le i \le n$) only the first row elements of five $P_{\beta-1}$ matrices are derived inside each P_β, where $\beta = n+1-i$ and the five submatrices are located at the same row of P_β.

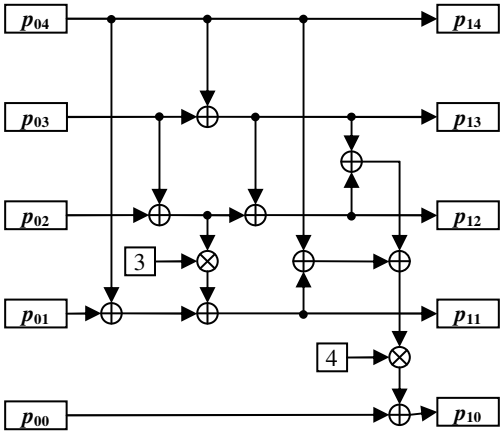


Figure 3.5: Flow diagram for the calculation of row 1 matrices.

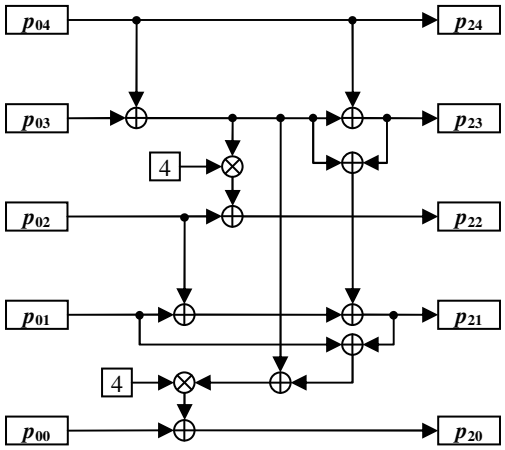


Figure 3.6: Flow diagram for the calculation of row 2 matrices.

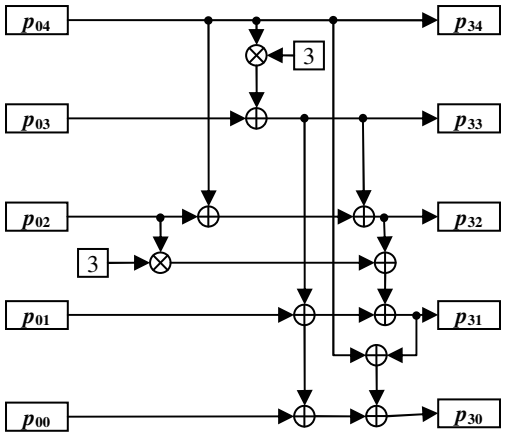


Figure 3.7: Flow diagram for the calculation of row 3 matrices.

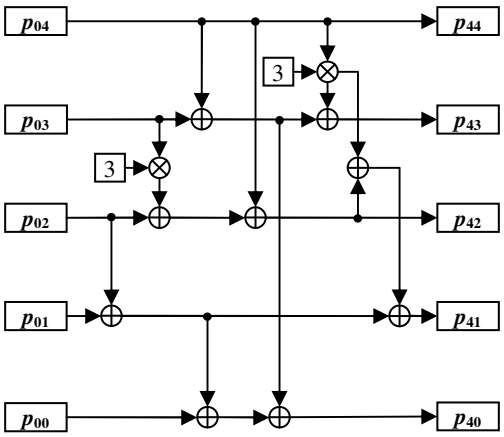


Figure 3.8: Flow diagram for the calculation of row 4 matrices.

Property 3.6.9. The P_β row index number of the matrices to be derived at the i-th recursion ($1 \le i \le n$) is given by j_i ($0 \le j_i \le 4$), where

$$j_i = \text{int}\left(\frac{\omega}{5^{n-i}}\right) \text{for } i = 1, \tag{3.51}$$

$$j_i = \text{int}\left(\frac{\omega - \sum_{l=1}^{i-1} j_l \cdot 5^{n-l}}{5^{n-i}}\right) \text{for } i > 1, \tag{3.52}$$

and the bracket int (rational number) means the integer part of a rational number.

Property 3.6.10. If $[\{(\omega \bmod 5^{n-1}) \bmod 5^{n-2}\} \ldots \bmod 5^{n-i}] = 0$ for some i, $1 \le i \le n$, the total number of the polarity matrix rows that are generated in the process of calculating the spectrum in polarity ω is equal to i.

Property 3.6.11. If $[\{(\omega \bmod 5^{n-1}) \bmod 5^{n-2}\} \ldots \bmod 5^{n-i}] = R$ ($R \in \{1, 2, 3, 4\}$) for some i, $1 \le i \le n$, the total number of the polarity matrix rows that are generated in the process of calculating the spectrum in polarity ω is equal to $i + 1$.

Property 3.6.12. There are $9 \cdot Y_\omega \cdot 5^{n-1}$ additions and $2 \cdot Y_\omega \cdot 5^{n-1}$ multiplications performed in the course of generating the spectrum in polarity ω, where Y_ω is the number of the polarity matrix rows that need to be generated to obtain the spectrum in polarity ω.

Proof: It can be seen from Figs. 3.5–3.8 that the calculation of the first row elements of any row $P_{\beta-1}$ matrices requires $9 \cdot 5^{\beta-1}$ additions and $2 \cdot 5^{\beta-1}$ multiplications. Since each row of the polarity matrix elements is contained in $5^{n-\beta}$ P_β matrices, it follows that the computational cost of calculating each row of polarity matrix is $9 \cdot 5^{n-1}$ additions and $2 \cdot 5^{n-1}$ multiplications. Therefore, if the number of generated polarity matrix rows is Y_ω, then the total computational cost incurred is $9 \cdot Y_\omega \cdot 5^{n-1}$ additions and $2 \cdot Y_\omega \cdot 5^{n-1}$ multiplications.

Property 3.6.13. By properties 3.6.10–3.6.12, it can be easily obtained that the worst-case computational cost for the calculation of a spectral coefficient vector occurs when $Y_\omega = n$ at $9n \cdot 5^{n-1}$ additions and $2n \cdot 5^{n-1}$ multiplications.

3.7 Column polarity matrix algorithm

Another recursive algorithm for obtaining all FPRMEs over GF(5) is presented in this section. Similar to the row polarity matrix algorithm, a recursive definition for the FPRME polarity matrix is first developed based on the Kronecker matrix properties as well as the presented relations between FPRME spectra in different polarities. However, the recursive definition derived for the algorithm presented in this section, called column polarity matrix algorithm, is one that allows the FPRME polarity matrix to be completely generated from the first column of the polarity matrix instead of the first row. The recursive definition is then optimized based on the arithmetic properties over GF(5) such that the new recursive definition requires smaller number of additions and multiplications. The column polarity matrix algorithm operates based on the

shown recursive definitions. Its applications for calculating both complete and partial FPRME polarity matrix is described and the fast flow diagrams for computation inside the algorithm are also given. Computational cost for the algorithm is then derived where it is shown that the new algorithm is advantageous over the other presented algorithms for computing selected FPRME spectrum.

3.7.1 Basic definitions and properties for column polarity matrix algorithm

Definition 3.7.1. Let $g_{[\beta,q]}$ be a column vector of size 5^{β} that can be recursively decomposed into smaller subvectors as follow

$$g_{[\beta,q]} = \left[g_{[\beta-1,0]}, g_{[\beta-1,1]}, g_{[\beta-1,2]}, g_{[\beta-1,3]}, g_{[\beta-1,4]}\right]^T, \tag{3.53}$$

where $q \in \{0, 1, 2, 3, 4\}$ and $\beta \geq 1$.

Definition 3.7.2. Let $H_{\beta}\left(g_{[\beta,q]}\right)$ be a $5^{\beta} \times 5^{\beta}$ matrix that is recursively defined by

$$H_{\beta}\left[g_{[\beta,q]}\right] = \begin{bmatrix}
H_{\beta-1}\left(g_{[\beta-1,0]}\right) & H_{\beta-1}\left(g_{[\beta-1,1]}+3g_{[\beta-1,2]}+2g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,1]}+g_{[\beta-1,2]}+g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(g_{[\beta-1,1]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+g_{[\beta-1,2]}+3g_{[\beta-1,3]}+2g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,2]}+g_{[\beta-1,3]}+g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(g_{[\beta-1,2]}\right) & H_{\beta-1}\left(2g_{[\beta-1,0]}+4g_{[\beta-1,1]}+g_{[\beta-1,3]}+3g_{[\beta-1,4]}\right) & H_{\beta-1}\left(g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,3]}+g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(g_{[\beta-1,3]}\right) & H_{\beta-1}\left(3g_{[\beta-1,0]}+2g_{[\beta-1,1]}+4g_{[\beta-1,2]}+g_{[\beta-1,4]}\right) & H_{\beta-1}\left(g_{[\beta-1,0]}+g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(g_{[\beta-1,4]}\right) & H_{\beta-1}\left(g_{[\beta-1,0]}+3g_{[\beta-1,1]}+2g_{[\beta-1,2]}+4g_{[\beta-1,3]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+g_{[\beta-1,1]}+g_{[\beta-1,2]}+4g_{[\beta-1,3]}\right)
\end{bmatrix}$$

$$\begin{bmatrix}
H_{\beta-1}\left(g_{[\beta-1,1]}+2g_{[\beta-1,2]}+3g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(4g_{[\beta-1,0]}+g_{[\beta-1,2]}+2g_{[\beta-1,3]}+3g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(3g_{[\beta-1,0]}+4g_{[\beta-1,1]}+g_{[\beta-1,3]}+2g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(2g_{[\beta-1,0]}+3g_{[\beta-1,1]}+4g_{[\beta-1,2]}+g_{[\beta-1,4]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right) \\
H_{\beta-1}\left(g_{[\beta-1,0]}+2g_{[\beta-1,1]}+3g_{[\beta-1,2]}+4g_{[\beta-1,3]}\right) & H_{\beta-1}\left(4g_{[\beta-1,0]}+4g_{[\beta-1,1]}+4g_{[\beta-1,2]}+4g_{[\beta-1,3]}+4g_{[\beta-1,4]}\right)
\end{bmatrix}$$

$$= \begin{bmatrix}
H_{\beta-1}(h_{00}) & H_{\beta-1}(h_{01}) & H_{\beta-1}(h_{02}) & H_{\beta-1}(h_{03}) & H_{\beta-1}(h_{04}) \\
H_{\beta-1}(h_{10}) & H_{\beta-1}(h_{11}) & H_{\beta-1}(h_{12}) & H_{\beta-1}(h_{13}) & H_{\beta-1}(h_{14}) \\
H_{\beta-1}(h_{20}) & H_{\beta-1}(h_{21}) & H_{\beta-1}(h_{22}) & H_{\beta-1}(h_{23}) & H_{\beta-1}(h_{24}) \\
H_{\beta-1}(h_{30}) & H_{\beta-1}(h_{31}) & H_{\beta-1}(h_{32}) & H_{\beta-1}(h_{33}) & H_{\beta-1}(h_{34}) \\
H_{\beta-1}(h_{40}) & H_{\beta-1}(h_{41}) & H_{\beta-1}(h_{42}) & H_{\beta-1}(h_{43}) & H_{\beta-1}(h_{44})
\end{bmatrix}, \tag{3.54}$$

where $1 \leq \beta \leq n$.

Property 3.7.1. The first column of $H_\beta\left(g_{[\beta,q]}\right)$ is the column vector $g_{[\beta,q]}$.

Property 3.7.2. By Definition 3.7.2 and Property 3.7.1, $H_0\left(g_{[0,q]}\right)$ has only one element. Thus, $H_0\left(g_{[0,q]}\right) = g_{[0,q]}$.

Definition 3.7.3. Let $\vec{F}$ be the truth vector of an n-variable five-valued function $f(\vec{x})$. Then $Y_\beta^{<v>}\left(\vec{F}\right)$ is defined as a subvector of $\vec{F}$ with 5^β elements that satisfies (3.55) and (3.56).

$$Y_n^{<0>}\left(\vec{F}\right) = \vec{F} \tag{3.55}$$

$$Y_\beta^{<v>}\left(\vec{F}\right) = \left[Y_{\beta-1}^{<v,0>}\left(\vec{F}\right), Y_{\beta-1}^{<v,1>}\left(\vec{F}\right), Y_{\beta-1}^{<v,2>}\left(\vec{F}\right), Y_{\beta-1}^{<v,3>}\left(\vec{F}\right), Y_{\beta-1}^{<v,4>}\left(\vec{F}\right)\right], \tag{3.56}$$

where $1 \le \beta \le n$, v is an $n-\beta+1$ -digits five-valued number, and '$<v,r>$' ($r \in \{0,1,2,3,4\}$) denotes the five-valued number obtained by shifting all digits of v by one place to the left and placing r in the least significant digit position.

Definition 3.7.4. Let $Y_\beta^{<v>}\left(\vec{F}\right)$ be as in Definition 3.7.3. Then $X_\beta^{<v>}\left(\vec{F}\right)$ ($1 \le \beta \le n$) is defined as a vector that is related to $Y_\beta^{<v>}\left(\vec{F}\right)$ such that

$$X_\beta^{<v>}\left(\vec{F}\right)\begin{cases}\left[Y_{\beta-1}^{<v,0>}\left(\vec{F}\right), Y_{\beta-1}^{<v,4>}\left(\vec{F}\right), Y_{\beta-1}^{<v,3>}\left(\vec{F}\right), Y_{\beta-1}^{<v,2>}\left(\vec{F}\right), Y_{\beta-1}^{<v,1>}\left(\vec{F}\right)\right], & \text{if } \beta = 1 \\ \left[X_{\beta-1}^{<v,0>}\left(\vec{F}\right), X_{\beta-1}^{<v,4>}\left(\vec{F}\right), X_{\beta-1}^{<v,3>}\left(\vec{F}\right), X_{\beta-1}^{<v,2>}\left(\vec{F}\right), X_{\beta-1}^{<v,1>}\left(\vec{F}\right)\right], & \text{otherwise.}\end{cases} \tag{3.57}$$

Property 3.7.3. Let $<v>_{10}$ be the decimal number representation of v in the vector $Y_\beta^{<v>}\left(\vec{F}\right)$ or $X_\beta^{<v>}\left(\vec{F}\right)$. Then, $0 \le <v>_{10} < 5^{n-\beta}$.

Property 3.7.4. By Definition 3.1.4, Property 3.1.1, (3.5), and (3.7), it can be obtained that the matrix $H_\beta\left(g_{[\beta,q]}\right)$ is equal to the FPRME polarity matrix of an n-variable function $f(\vec{x})$ when $\beta = n$ and $g_{[\beta,q]} = X_n^{<0>}\left(\vec{F}\right)$.

$$P(f(\vec{x})) = H_\beta\left(X_n^{<0>}\left(\vec{F}\right)\right). \tag{3.58}$$

Example 3.7.1. Let $f(\vec{x})$ be a one-variable five-valued function with truth vector $\vec{F} = [f_0, f_1, f_2, f_3, f_4]^T$. Then by Definitions 3.7.3 and 3.7.4, $Y_1^{<0>}\left(\vec{F}\right) = \vec{F}$ and $X_1^{<0>}\left(\vec{F}\right) = [f_0, f_4, f_3, f_2, f_1]$. Furthermore, by Property 3.7.4

$$P(f(\vec{x})) = H_1\left[X_1^{<0>}\left(\vec{F}\right)\right]$$

$$= \begin{bmatrix}
f_0 & f_4+3f_3+2f_2+4f_1 & 4f_4+f_3+f_2+4f_1 & & \\
f_4 & 4f_0+f_3+3f_2+2f_1 & 4f_0+4f_3+f_2+f_1 & & \\
f_3 & 2f_0+4f_4+f_2+3f_1 & f_0+4f_4+4f_2+f_1 & & \\
f_2 & 3f_0+2f_4+4f_3+f_1 & f_0+f_4+4f_3+4f_1 & & \\
f_1 & f_0+3f_4+2f_3+4f_2 & 4f_0+f_4+f_3+4f_2 & & \\
 & & & f_4+2f_3+3f_2+4f_1 & 4f_0+4f_4+4f_3+4f_2+4f_1 \\
 & & & 4f_0+f_3+2f_2+3f_1 & 4f_0+4f_4+4f_3+4f_2+4f_1 \\
 & & & 3f_0+4f_4+f_2+2f_1 & 4f_0+4f_4+4f_3+4f_2+4f_1 \\
 & & & 2f_0+3f_4+4f_3+f_1 & 4f_0+4f_4+4f_3+4f_2+4f_1 \\
 & & & f_0+2f_4+3f_3+4f_2 & 4f_0+4f_4+4f_3+4f_2+4f_1
\end{bmatrix}.$$

The same result of $P[f(\vec{x})]$ is obtained if the FPRME spectra in all polarities are calculated from truth vector by (3.5) and then rearranged according to Definition 3.1.4.

The recursive definition for $H_\beta\left(g_{[\beta,q]}\right)$, and hence $P[f(\vec{x})]$, in (3.54) can be optimized by examining the relations between the submatrices of $H_\beta\left(g_{[\beta,q]}\right)$ and applying them to generate an equivalent recursive definition such that the complete generation of $P[f(\vec{x})]$ from $X_n^{<0>}\left(\vec{F}\right)$ requires less numbers of arithmetic operations. Such definition that is used by the column polarity matrix algorithm is shown in Definition 3.7.5.

Definition 3.7.5. Let $I_{1[\beta-1]}$, $I_{2[\beta-1]}$, and $I_{3[\beta-1]}$ represent $h_{20}+h_{30}$, $h_{20}+h_{30}+h_{00}$, and $h_{20}+h_{30}+h_{02}+2h_{10}$, respectively. Then the recursive definition for $H_\beta\left(g_{[\beta,q]}\right)$ in (3.54) can be rewritten as follows

$$H_\beta\left(g_{[\beta,q]}\right)=\begin{bmatrix} H_{\beta-1}(h_{00}) & H_{\beta-1}\left(I_{3[\beta-1]}+h_{20}\right) & H_{\beta-1}\left(I_{1[\beta-1]}+I_{2[\beta-1]}+h_{04}\right) & H_{\beta-1}\left(I_{3[\beta-1]}+h_{30}\right) & H_{\beta-1}\left(4\times\left(I_{2[\beta-1]}+h_{10}+h_{40}\right)\right) \\ H_{\beta-1}(h_{10}) & H_{\beta-1}(h_{41}+h_{22}+h_{33}) & H_{\beta-1}(h_{32}+h_{23}) & H_{\beta-1}(h_{03}+h_{04}) & H_{\beta-1}(h_{04}) \\ H_{\beta-1}(h_{20}) & H_{\beta-1}(h_{01}+h_{32}+h_{43}) & H_{\beta-1}(h_{42}+h_{33}) & H_{\beta-1}(h_{13}+h_{04}) & H_{\beta-1}(h_{04}) \\ H_{\beta-1}(h_{30}) & H_{\beta-1}(h_{11}+h_{42}+h_{03}) & H_{\beta-1}(h_{02}+h_{43}) & H_{\beta-1}(h_{23}+h_{04}) & H_{\beta-1}(h_{04}) \\ H_{\beta-1}(h_{40}) & H_{\beta-1}(h_{21}+h_{02}+h_{13}) & H_{\beta-1}(h_{12}+h_{03}) & H_{\beta-1}(h_{33}+h_{04}) & H_{\beta-1}(h_{04}) \end{bmatrix}. \tag{3.59}$$

3.7.2 Computational steps of column polarity matrix algorithm

The column polarity matrix algorithm recursively generates either a spectrum or complete FPRME polarity matrix for a five-valued function based on (3.54) and (3.59), respectively. Below the calculation steps of the column polarity matrix algorithm for the two cases are given.

Steps to calculate the complete polarity matrix

Step 1: Generate $X_n^{<0>}\left(\vec{F}\right)$ based on Definitions 3.7.3 and 3.7.4, where $\vec{F}$ and n are the truth vector and the number of input variables of the input function, respectively.

Step 2: Let $X_n^{<0>}\left(\vec{F}\right)$ be the first column of the polarity matrix, i.e., $P[f(\vec{x})]=H_n\left(X_n^{<0>}\left(\vec{F}\right)\right)$. Set $\beta = n$.

Step 3: For each H_β matrix: if its first column elements have been obtained in the previous steps then divide the matrix into its respective $H_{\beta-1}$ submatrices based on (3.54). Subsequently, calculate the first column elements of the $H_{\beta-1}$ submatrices according to the flow diagram given in Fig. 3.9. In the figure, h_{rs} $(0 \le r, s \le 4)$ represents the first column elements of the submatrix $H_{\beta-1}(h_{rs})$ of the currently processed H_β matrix (refer to (3.54)). Note that in the figure the first column elements are generated only for 21 submatrices instead of 25 since the rest can be obtained by copying which is done at Steps 5 and 6.

Step 4: If $\beta = 1$ go to Step 5. Else, decrement β by 1 and repeat Step 3.

Step 5: For each H_β matrix, divide the matrix into its $H_{\beta-1}$ submatrices. If all elements of the submatrix $H_{\beta-1}(h_{04})$ of the H_β matrix have been obtained in the previous steps, then complete the matrix by copying its submatrix h_{04} to its submatrices $H_{\beta-1}(h_{14})$, $H_{\beta-1}(h_{24})$, $H_{\beta-1}(h_{34})$, and $H_{\beta-1}(h_{44})$.

Step 6: If $\beta = n$ exit the algorithm. Else, increment β by 1 and repeat Step 5.

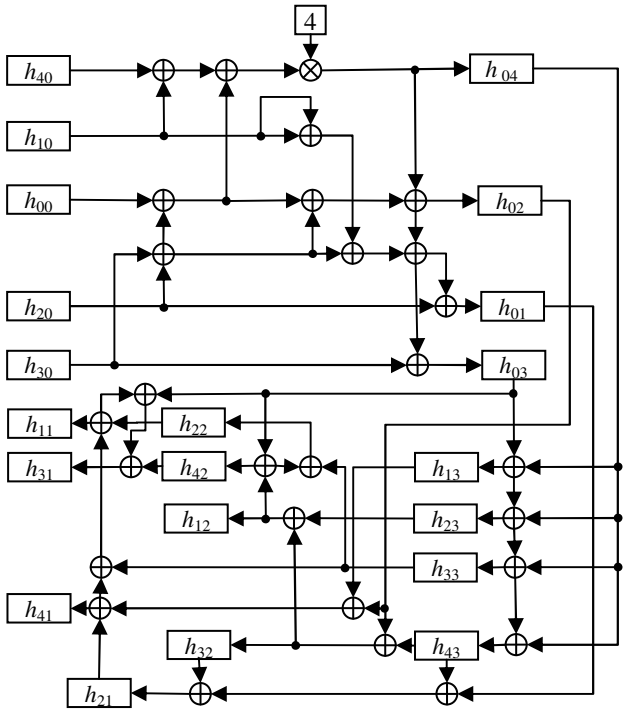


Figure 3.9: Flow diagram for the calculation in a recursion stage of column polarity matrix algorithm.

Steps to calculate FPRME spectral coefficients in polarity ω

Step 1: Generate $X_n^{<0>}(\vec{F})$ based on Definitions 3.7.3 and 3.7.4, where $\vec{F}$ and n are the truth vector and the number of input variables of the input function, respectively.

Step 2: Let $<\omega>_{10} = <j_1, j_2, ..., j_n>_5$. Set β and θ to n and 1, respectively.

Step 3: For each H_β matrix with known first column elements, generate its $H_{\beta-1}$

submatrices that contain the polarity matrix row to be obtained, i.e. $H_{\beta-1}(h_{j_\theta 1})$, $H_{\beta-1}(h_{j_\theta 2})$, $H_{\beta-1}(h_{j_\theta 3})$, and $H_{\beta-1}(h_{j_\theta 4})$. The flow diagram for the calculation of those submatrices are given in Fig. 3.10, where $j_\theta \in \{0,1,2,3,4\}$ and $r_y = j_\theta + y$ over GF(5), respectively. In the figure, h_{rs} $(0 \le r, s \le 4)$ represents the first column elements of the submatrix h_{rs} of the currently processed H_β matrix (refer to (3.54)). Also, the symbols '⊕' and '⊗' represent GF(5) adder and multiplier, respectively.

Step 4: If $\beta = 1$, exit the algorithm. Else, set β and θ to $\beta - 1$ and $\theta + 1$, respectively and repeat Step 3.

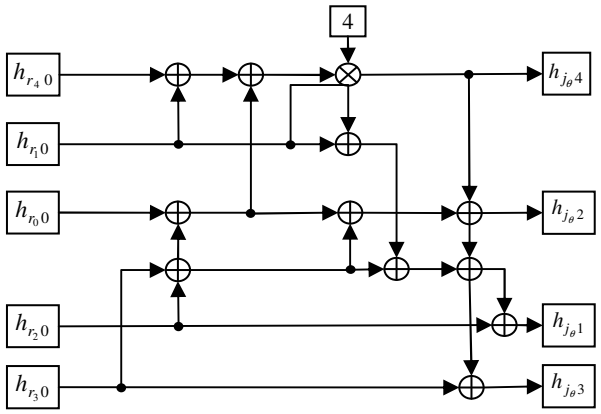


Figure 3.10: Flow diagram for the calculation of row j_θ matrices (column polarity matrix algorithm).

Example 3.7.2. Let $f(\vec{x})$ be a two-variable five-valued function that is described by the truth vector $\vec{F} = [3,4,2,1,0,4,1,2,0,3,2,0,4,1,3,3,2,4,0,1,0,0,0,0,0]$. Then, the steps of obtaining FPRME of $f(\vec{x})$ in polarity seven by the column polarity matrix algorithm are as follows:

- Generate $X_2^{<0>}(\vec{F})$. By Definition 3.7.3, the subvectors $Y_1^{<0,0>}(\vec{F})$, $Y_1^{<0,1>}(\vec{F})$, $Y_1^{<0,2>}(\vec{F})$, $Y_1^{<0,3>}(\vec{F})$, and $Y_1^{<0,4>}(\vec{F})$ are [3,4,2,1,0], [4,1,2,0,3], [2,0,4,1,3], [3,2,4,0,1], and [0,0,0,0,0], respectively. By Definition 3.7.4, the subvectors

$X_1^{<0,0>}(\vec{F})$, $X_1^{<0,1>}(\vec{F})$, $X_1^{<0,2>}(\vec{F})$, $X_1^{<0,3>}(\vec{F})$, and $X_1^{<0,4>}(\vec{F})$ are [3,0,1,2,4], [4,3,0,2,1], [2,3,1,4,0], [3,1,0,4,2], and [0,0,0,0,0], respectively. Subsequently, $X_2^{<0>}(\vec{F})$=[3,0,1,2,4,0,0,0,0,0,3,1,0,4,2,2,3,1,4,0,4,3,0,2,1].

- Divide $P[f(\vec{x})]$ into 25 H_1 matrices. Since $\omega =< 7 >_{10} =< 1,2 >_5$, we need to calculate the first column elements of the submatrices $H_1(h_{11})$, $H_1(h_{12})$, $H_1(h_{13})$, and $H_1(h_{14})$ of $P[f(\vec{x})] = H_2(X_2^{<0>}(\vec{F}))$. By applying Fig. 3.10, it can be obtained that the first column elements of the $H_1(h_{11})$, $H_1(h_{12})$, $H_1(h_{13})$, and $H_1(h_{14})$ submatrices are $[4,1,2,3,0]^T$, $[0,0,0,0,0]^T$, $[1,1,1,1,1]^T$, and $[3,3,3,3,3]^T$, respectively.
- Divide the H_1 matrices that corresponds to $H_1(h_{10})$, $H_1(h_{11})$, $H_1(h_{12})$, $H_1(h_{13})$, and $H_1(h_{14})$ of $P[f(\vec{x})]$ into their respective H_0 submatrices. Since $j_2 = 2$, calculate $H_0(h_{21})$, $H_0(h_{22})$, $H_0(h_{23})$, and $H_0(h_{24})$ submatrices of those H_1 matrices. After the calculation, all elements of C^7 for $f(\vec{x})$ have been obtained:

$$C^7 = [0,0,0,0,0,2,0,0,4,0,0,0,0,0,0,1,0,0,0,0,3,0,0,0,0].$$

- Hence, the FPRME of $f(\vec{x})$ in polarity seven is

$$f(\vec{x}) = 2^{<1>}x_1 + 4^{<1>}x_1^{<2>}x_2^{\,3} + {}^{<1>}x_1^{\,3} + 3^{<1>}x_1^{\,4}$$

which can be implemented by the circuit shown in Fig. 3.11. In the figure, the addition and multiplication gates output the sum and product of the inputs over GF(5), respectively.

Property 3.7.5. The total number of additions and multiplications performed by the algorithm in order to generate the complete polarity matrix from the first column of polarity matrix is $\frac{27}{16}(21^n - 5^n)$ and $\frac{1}{16}(21^n - 5^n)$, respectively.

Proof: Let the number of $H_i(X_n^{<0>}(\vec{F}))$ matrices with unknown first column elements at the recursion stage where $\beta = i$ be $A(i)$. Since there are a total of 25^{n-i} $H_i(X_n^{<0>}(\vec{F}))$ matrices in the polarity matrix, based on the given steps of the

algorithm at the recursion stage there are $\left(25^{n-i} - A(i)\right)$ $H_i\left(X_n^{<0>}\left(\vec{F}\right)\right)$ matrices whose first column elements of its 21 submatrices are calculated. By the flow diagram given in Fig. 3.9, the number of arithmetic operations involved in the calculation of the submatrices of an $H_i\left(X_n^{<0>}\left(\vec{F}\right)\right)$ is $27 \cdot 5^{i-1}$ additions and 5^{i-1} multiplications. Hence, there are a total of $\left(25^{n-i} - A(i)\right) \cdot 27 \cdot 5^{i-1}$ additions and $\left(25^{n-i} - A(i)\right) \cdot 5^{i-1}$ multiplications performed at the recursion stage where $\beta = i$. Since there are n recursion stages, the cost of deriving the complete polarity matrix by this algorithm is

$$A_{12}(n) = \sum_{i=1}^{n} \left(25^{n-i} - A(i)\right) \cdot 27 \cdot 5^{i-1} \text{ additions} \tag{3.60}$$

and

$$M_{12}(n) = \sum_{i=1}^{n} \left(25^{n-i} - A(i)\right) \cdot 5^{i-1} \text{ multiplications.} \tag{3.61}$$

When $i = n$, the value of $A(i)$ is equal to 0. At the next stage when $i = n - 1$, the value of $A(i)$ is equal to $\left(A(n-1) \cdot 25\right) + \left(21 \cdot 4\right)$. The first term is the submatrices number of the $H_{i+1}\left(X_n^{<0>}\left(\vec{F}\right)\right)$ matrices whose first column elements are unknown at the previous recursion stage, whereas the second term is the number of $H_i(h_{14})$, $H_i(h_{24})$, $H_i(h_{34})$, and $H_i(h_{44})$ submatrices of the rest of the $H_{i+1}\left(X_n^{<0>}\left(\vec{F}\right)\right)$ matrices at the previous recursion stage. Continuing this analysis, it is obtained that in general,

$$\begin{aligned} A(i) &= \left(A(i+1) \cdot 25\right) + \left(21^{n-i-1} \cdot 4\right) \text{ for } 1 \le i \le n-1 \\ &= \sum_{l=1}^{n-i} \frac{4}{25} \cdot 21^{(n-i)} \cdot \left(\frac{25}{21}\right)^{l} \\ &= 25^{n-i} - 21^{n-i}. \end{aligned} \tag{3.62}$$

Substituting (3.62) to (3.60) and (3.61), it is obtained that the total number of additions and multiplications in all recursion stages are $\sum_{i=1}^{n} 27 \cdot 21^{n-i} \cdot 5^{i-1} = \frac{27}{16}\left(21^n - 5^n\right)$ and $\sum_{i=1}^{n} 21^{n-i} \cdot 5^{i-1} = \frac{1}{16}\left(21^n - 5^n\right)$, respectively.

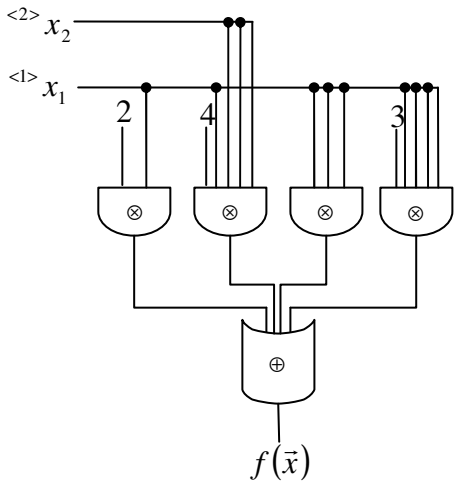


Figure 3.11: Circuit implementation of FPRME in polarity seven.

Property 3.7.6. The number of arithmetic operations required by the algorithm in order to generate the elements of one FPRME spectrum is $11 \cdot n \cdot 5^{n-1}$ additions and $n \cdot 5^{n-1}$ multiplications.

Proof: There are 5^{n-i} $H_i\left(X_n^{<0>}\left(\vec{F}\right)\right)$ that contain the spectral coefficients of the vector being calculated. At the recursion stage where $\beta = i$, the first column elements of four $H_{i-1}\left(Z_n^{<0>}\left(\vec{F}\right)\right)$ submatrices of each of those matrices are generated using the flow diagram in Fig 3.10. Since from the figure it can be obtained that the generation of the first column elements of the four submatrices of a particular $H_i\left(Z_n^{<0>}\left(\vec{F}\right)\right)$ matrix requires $11 \cdot 5^{i-1}$ and 5^{i-1} additions and multiplications, respectively, it follows that at the recursion stage there are a total of $11 \cdot 5^{n-1}$ additions and 5^{n-1} multiplications being performed. The algorithm has n recursion stages. Hence, the total computational cost for calculating a particular spectral coefficient vector using this algorithm is $11 \cdot n \cdot 5^{n-1}$ additions and $n \cdot 5^{n-1}$ multiplications.

The computational cost for deriving the complete polarity matrix by both the row and column polarity matrix algorithms from the truth vector is shown in Table 3.11 for $n < 6$. Since the computational cost derived for the row polarity matrix is based on the

assumption that the first row of the polarity matrix has been known, in Table 3.11 the computational cost for the row polarity matrix is given as the summation of the derived computational cost and the cost of obtaining FPRME spectrum in polarity zero from truth vector by fast version of (3.5). Comparing the computational cost numbers shown in Tables 3.5 and 3.14, it can be concluded that the column polarity matrix algorithm has the smallest computational cost for deriving the complete polarity matrix for $n < 4$ whereas for larger functions the row polarity matrix algorithm has the most efficient computational cost. In addition, Table 3.12 lists the computational cost of deriving an FPRME spectrum from the truth vector by the column polarity matrix algorithm and by the fast version of (3.5). It can be seen that the computational cost for the column polarity matrix algorithm is always smaller than for the fast version of (3.5). Since both the matrix multiplication and row polarity matrix algorithms require the FPRME spectrum in the starting polarity to be known, their computational cost for generating a spectral coefficient vector from the truth vector is always greater than the computational cost of the fast version of (3.5). Therefore, from Table 3.12 it can be concluded that among the algorithms, the column polarity matrix algorithm incurs the least computational cost for deriving a particular spectral coefficient vector from the truth vector.

Table 3.11: Computational cost of polarity matrix algorithms.

n	Column polarity matrix algorithm		Row polarity matrix algorithm	
	Additions	Multiplications	Additions	Multiplications
1	27	1	36	13
2	702	26	728	130
3	15417	571	14108	975
4	327132	12116	285168	6500
5	6886647	255061	5907028	40625

Table 3.12: Computational cost of deriving selected spectral coefficient vector.

n	Additions		Multiplications	
	Fast version of (3.5)	Column polarity matrix algorithm	Fast version of (3.5)	Column polarity matrix algorithm
1	13	11	13	1
2	130	110	130	10
3	975	825	975	75
4	6500	5500	6500	500
5	40625	34375	40625	3125

3.8 Experimental results for FPRMEs over GF(5)

Experimental results for the algorithms presented in Sections 3.2–3.7 in terms of their execution time for obtaining all FPRMEs, and therefore the optimal FPRME, are presented in this section. The generation of the five-valued test files used as inputs to the algorithm are also explained.

3.8.1 Generation of five-valued test files

The algorithms described in this chapter have been implemented as a C program. Since currently no five-valued benchmarks are available, an effort is made to generate five-valued test files from the MCNC binary benchmarks to be used as inputs to the program. The generation is done by converting each entry in the binary benchmark into one or more entries in the corresponding five-valued test file through mapping of every three literals in the binary input cube to a literal in the five-valued input cube. Similarly, to obtain the five-valued outputs, every three outputs in the binary benchmark is mapped into a five-valued output. Since there are eight possible combinations of three binary literals (outputs) while a five-valued literal (output) has only five values, statistics of some binary benchmarks files are investigated in order to

determine the best mapping rules. The binary benchmarks that are taken as samples are: 9sym, apex4, clip, con1, ex1010, ex5, inc, misex1, rd84, squar5, xor5, z5xp1 and z9sym.

The collected data describe how many times each combination of three literal values occurs in the input cubes (input frequency of appearance *ifapp*) and how many times each combination of three output values is encountered in the output cubes (output frequency of appearance *ofapp*). For collecting this data, each binary input cubes with missing literals ('−') are first replaced by their minterms, where the outputs of the minterms are the same as the replaced cube's outputs. When the number of input or output variables is not a multiple of three, one or two appropriate variables are added behind in such a way that the numbers of both input and output variables are multiple of three. The values of the added variables are always zero. The rules for counting *ifapp* and *ofapp* are different. For input cubes, each input cube can only contribute maximum one count to *ifapp* for each combination even when some combinations occur more than once in the cube. Because of this rule, the accurate information can be obtained on how many cubes will become invalid if certain input combination is not used. For output, every time a combination occurs, the *ofapp* for that combination is incremented by one, regardless of whether it occurs more than once in an entry. Table 3.13 lists *ifapp* and *ofapp* of all possible combinations. The given number for a particular combination is the total number that such a combination has been found in all sample benchmarks.

It can be seen that the *ifapp* for each combination are roughly the same so it is decided that none of the input combinations are eliminated (the entries with those combinations are not used). For the output, the numbers vary more. To make the numbers of the nonzero output values in the test files roughly the same, the chosen mapping rule for output is as follows: 000, 101, and 110 are mapped to 0; 001 and 011 are mapped to 1; 010 is mapped to 2; 111 is mapped to 3; and 100 is mapped to 4.

Several input mapping rules have been explored to determine the best mapping rule for input cube conversion. For each input mapping rule, the five-valued test files are generated by performing the following steps:

- If the number of input or output variable is not a multiple of three, add one or two appropriate variables behind such that the numbers of input and output variables are both multiple of three. The values of the added variables are always zero.
- Divide the input cube literals of each entry (line) in the binary benchmark into groups of three consecutive literals. Similarly, divide the output values of the entry into groups of three consecutive output values.
- For those entries that have one or more groups with less than three missing literals, expand the input cube with regards to the scattered '–'s. Place each expansion into separate entry. The output values of the new entries are the same as the output values of the cube. Finally, delete the original cube.
- Convert every entry into an entry in the five-valued test files by mapping every group of three binary literals or outputs into a literal or output in the test file according to the used input and output mapping rules. The used output mapping rule is the one given above.
- Build truth vector for each output in the test file. If a minterm has more than one nonzero output values for an output variable, take the smallest value as the output value for the minterm.

Table 3.13: The values of *ifapp* and *ofapp* for all combinations.

Combination	Input	Output
000	2579	12096
001	2114	1664
010	2112	2110
011	1906	563
100	2786	3409
101	2044	455
110	2157	491
111	1764	2173

Three criteria are used to choose the selected input mapping rule. First, the number of unique cubes in the resulting five-valued test files should be as high as possible. Second, the number of nonzero truth vector elements for the output variables should also be as high as possible. Third, the number of minterms having more than one nonzero output values for an output variable should be minimized. Based on these criteria, the following mapping rule is chosen for the input: −−− is mapped to −; 000 is mapped to 0; 001 is mapped to1; 010 is mapped to 2; 011 is mapped to 0; 100 is mapped to 4; 101 is mapped to 3; 110 is mapped to 3; and 111 is mapped to 4.

Example 3.8.1. An entry in binary benchmark '010−−−11−1 10' is converted into two entries in five-valued test file as follows:

- As the numbers of input and output variables are not multiple of three, add new variables with values zero behind both the input and output cubes ('010−−−11−100 100').
- As the input cube has a group with one missing literal, expand the input cube ('010−−−110100 100'; '010−−−111100 100').
- Convert the entries into entries in five-valued test files according to the mapping rules ('2−34 4'; '2−44 4').

3.8.2 Experimental results for cube polarity adjustment algorithm

A C program that takes the disjoint cubes representation of a five-valued function and generates its whole FPRME polarity matrix using the algorithm presented in Section 3.5 has been written and run on a 500 MHz, 256 MB RAM Pentium III computer. Its execution time for several five-valued test files is shown in Table 3.14. Here the smaller computer speed is chosen so that the effect of the number of disjoint cubes of the input function on the execution time of the algorithm can be seen more clearly.

The generation of the five-valued test files that are used here has been described in the Section 3.8.1. Since the cubes in the resulting five-valued test files may not be disjoint, an algorithm similar to that given in [FSP92a] is used to generate the

equivalent disjoint cubes representations of the five-valued test files before they are used as inputs to the program. The preprocessing program generates separate sets of disjoint cubes for each nonzero value of each output variable. The numbers of input and output variables as well as the total number of disjoint cubes for the five-valued test files are shown in Table 3.15.

Table 3.14: Execution time for several five-valued test files.

Input files	Execution time (in seconds)
xor5	0.42
squar5	1.14
con1	2.26
clip	3.38
rd84	3.50
inc	4.26
misex1	4.79
z5xp1	4.97
apex4	9.62
ex5	21.39
ex1010	118.44
b12	444.83
table3	5105.10

As can be seen from Tables 3.14 and 3.15, the execution time of the program increases as the number of either variables or disjoint cubes rises. Similarly, detailed analysis of the algorithm shows that it has small memory requirement. Hence, it can be concluded that if the minimum calculation time is desired, the algorithm should be used when the input functions are represented by small number of disjoint cubes and when a few spectral coefficients need to be calculated. However, if the memory requirement is more important, this algorithm is useful for calculating the FPRME spectral coefficients for functions with large number of variables, for which other

methods require much more storage space.

Table 3.15: Numbers of input, output, and disjoint cubes for several five-valued test files.

Input files	Number of input variables	Number of output variables	Total number of disjoint cubes
xor5	2	1	10
squar5	2	3	38
con1	3	1	13
clip	3	2	99
rd84	3	2	103
inc	3	3	67
misex1	3	3	180
z5xp1	3	4	74
apex4	3	7	454
ex5	3	21	592
ex1010	4	4	621
b12	5	3	56
table3	5	5	1234

3.8.3 Experimental results for all FPRME algorithms

The generation of the complete FPRME polarity matrix by the algorithms presented in Sections 3.2–3.7 have also been implemented as C++ programs and run on a 2.8 GHz, 512 MB RAM Pentium IV computer. Their execution times for some five-valued test files are presented in Table 3.16. For all the algorithms, the execution times shown in Table 3.16 are the times required by the algorithms to run from the moment the input file is specified until the complete polarity matrix is generated and the polarities with three best nonzero numbers for the input function are obtained. Hence, it includes the time spent for building the truth vector and the required transform matrices as well as

the time needed to determine the next polarity to be calculated for the algorithms that employ the extended dual polarity route. In the experiment, the extended dual polarity route in Fig. 3.1 is used for both the extended dual polarity algorithm and the matrix multiplication algorithm along extended dual polarity route. The five-valued test files that are used as inputs to these programs are actually binary MCNC benchmarks that have been modified in a consistent manner to represent five-valued functions as described in Section 3.8.1.

Table 3.16: Execution times to calculate spectral coefficients of several five-valued test files (in seconds).

Input file name	Matrix multiplication algorithms			Extended dual polarity algorithm	Cube polarity adjustment algorithm	Polarity matrix algorithms	
	Lexicographic order (direct)	Lexicographic order (fast)	Extended dual polarity (best-case)			Row polarity matrix	Column polarity matrix
xor5	0.36	0.27	0.27	0.27	0.05	0.02	0.00
squar5	0.31	0.31	0.30	0.31	0.05	0.00	0.02
con1	0.39	0.34	0.36	0.38	0.11	0.02	0.03
z5xp1	0.75	0.63	0.63	0.64	0.39	0.13	0.13
inc	0.64	0.55	0.55	0.53	0.28	0.09	0.09
rd84	0.52	0.47	0.44	0.44	0.25	0.06	0.06
misex1	0.64	0.55	0.52	0.55	0.39	0.08	0.09
ex5	2.97	2.28	2.23	2.19	2.34	0.67	0.66
9sym	0.39	0.36	0.36	0.34	0.11	0.05	0.02
z9sym	0.38	0.34	0.38	0.34	0.11	0.02	0.03
clip	0.52	0.47	0.45	0.47	0.22	0.08	0.08
apex4	1.13	0.89	0.91	0.91	1.01	0.22	0.23
ex1010	42.69	25.66	20.97	2.53	27.78	0.92	0.91

From the numbers in Table 3.16, it can be seen that among all the algorithms the polarity matrix algorithms have the shortest execution times for all test files whereas the matrix multiplication algorithms have the largest execution time for most of the test files, especially when the number of input and output variables are small. The differences between their execution times are increasing with the number of input and/or outputs variables. From the computational costs derived for the three developed matrix multiplication algorithms, it is expected that among them the calculation time for the dual polarity route algorithm is the shortest followed by lexicographic order (fast) and lexicographic order (direct). Although this is true for most of the test files, for some test files the lexicographic order (fast) is faster than the extended dual polarity route algorithm. This discrepancy is due to the additional time used for determining the polarity of the spectral coefficient vector to be calculated next in the extended dual polarity route algorithm. Similarly, from the derived computational cost, it is found that the column polarity matrix algorithm has smaller computational cost compared to the row polarity matrix algorithm for $n < 4$. However, their execution times as shown in Table 3.16 are not much different. This is due to the time spent for converting the truth vector to the first column of the polarity matrix in the column polarity matrix algorithm.

Chapter 4

New Fastest Linearly Independent Transforms for Binary Functions

New fastest LI transforms for binary functions are introduced in this chapter. The new transforms are grouped into fastest LI transforms over GF(2) and fastest LIA transforms based on their operational fields. The transforms have the lowest computational cost in their respective field in terms of the number of required additions/subtractions. Here the fastest LI transforms over GF(2) are first discussed followed by fastest LIA transforms. For each of them, basic definitions are given followed by their properties and experimental results.

4.1 Basic definitions and operators for fastest LI transforms of binary functions

Definition 4.1.1. Let M_n be a square matrix of size $2^n \times 2^n$ with columns corresponding to truth vectors of 2^n n-variable binary functions. Then M_n is said to be linearly independent if the set of columns in M_n is linearly independent with respect to XOR operations (i.e., columns are bit-by-bit XORed). A linearly independent M_n is

nonsingular and has a unique inverse in both GF(2) and standard arithmetic algebra.

Definition 4.1.2. The truth vector of binary functions can be encoded using either R-coding or S-coding. In R-coding, true minterms are represented by 1, false minterms by 0, and don't care minterms by 0.5. In S-coding, they are represented by –1, 1, and 0, respectively.

Definition 4.1.3. Let $X = < x_n, x_{n-1}, ..., x_1 >$ be an n-bit binary string with x_n as the most significant bit (MSB). Then $L_B(X)$ is defined as the subscript value of the rightmost bit in X whose value is equal to 1.

Example 4.1.1. Let X_1 and X_2 be two binary strings, where X_1 = <0, 0, 0, 1> and X_2 = <1, 1, 0, 1, 0, 0>. Then according to Definition 4.1.3, $L_B(X_1)$ and $L_B(X_2)$ are 1 and 3, respectively.

Definition 4.1.4. The operator R_0 on a column vector is defined as applying dyadic shift [Moh92] that reverses the order of the elements in the vector. If $\vec{b}$ represents a column vector with t elements $\vec{b} = [b_1, b_2, ..., b_{t-1}, b_t]^T$, then $R_0(\vec{b}) = [b_t, b_{t-1}, ..., b_2, b_1]^T$, where T denotes matrix transpose operator.

Definition 4.1.5. The operator $R_{1,r}$ on a $2^n \times 2^n$ matrix M_n is defined as performing 4^{n-r-1} counterclockwise rotations involving 4^{n-r} submatrices each of order 2^r $(0 \le r \le n-1)$.

Example 4.1.2. Let M_2 be a transform matrix of size 4 × 4, where $M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & -1 & 1 \end{bmatrix}$. Then by Definition 4.1.5, $\left(R_{1,1}(M_2)\right) = \left[\begin{array}{cc:cc} 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ \hdashline 1 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 \end{array}\right]$ and

$$\left(R_{1,0}\left(M_2\right)\right) = \left[\begin{array}{cc:cc} 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 \\ 0 & -1 & 1 & -1 \end{array}\right].$$

Definition 4.1.6. The operator R_2 on a $2^n \times 2^n$ matrix M_n is defined as recursively applying operator $R_{1,r}$ on M_n for $r = n - 1, n - 2, \ldots, 1, 0$. The integer power of operator R_2 is specified as $R_2{}^2\left(M_n\right) = R_2\left(R_2\left(M_n\right)\right)$.

Definition 4.1.7. Let M_n be a square matrix of dimension $2^n \times 2^n$. The operator α_1 on M_n is defined as replacing the element of M_n that is located at the bottom left corner with 1. Thus, if $M_n = [m_{i,j}]$ then $\alpha_1(M_n) = [m'_{i,j}]$, where $0 \le i, j \le 2^n - 1$ and

$$m'_{i,j} = \begin{cases} 1, & \text{if } i = 2^n - 1 \text{ and } j = 0 \\ m_{i,j}, & \text{otherwise.} \end{cases} \tag{4.1}$$

Definition 4.1.8. Let M_n be a square matrix of size $2^n \times 2^n$. Then the operator α_2 on M_n is defined as replacing the element of M_n that is located at the bottom left corner with -1. Thus, if $M_n = [m_{i,j}]$ then $\alpha_2(M_n) = [m'_{i,j}]$, where $0 \le i, j \le 2^n - 1$ and

$$m'_{i,j} = \begin{cases} -1, & \text{if } i = 2^n - 1 \text{ and } j = 0 \\ m_{i,j}, & \text{otherwise.} \end{cases} \tag{4.2}$$

Definition 4.1.9. Permutation matrices are matrices that contain exactly one 1 in each row and column. There are 2^n possible unique permutation matrices of dimension $2^n \times 2^n$ that can be derived from Kronecker product of n elementary permutation matrices. They are differentiated from each other by the permutation number p. Let the elementary permutation matrices be denoted by ρ_0 and ρ_1. Then the permutation matrix of size $2^n \times 2^n$ and permutation number p ($0 \le p \le 2^n - 1$) is given by

$$P_n^p = \bigotimes_{j=n}^{1} \rho_{p_j}, \tag{4.3}$$

where $\rho_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\rho_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ [Moh92], $\otimes$ denotes Kronecker product and $< p_n, p_{n-1}, ..., p_1 >$ is the n-bit binary representation of p, i.e., $< p >_{10} =< p_n, p_{n-1}, ..., p_1 >_2$.

4.2 Fastest LI transforms over GF(2)

In this section we seek to derive new fastest LI transforms over GF(2) by reordering the butterfly diagram stages of the existing fastest LI transforms [RF02] and inserting a permutation matrix in front of, behind, or inside the butterfly diagrams. Due to their way of generation, the new fastest LI transforms have the same numbers of computational cost as the existing ones [RF02] and possess regular structure. It also ensures that the fastest LI transforms have fast forward and inverse transforms, which means that they can be calculated quickly and efficiently. The fast forward and inverse matrices for the transforms are defined. Several properties on the matrices structure and relations between different fastest LI transforms are listed. Experimental results are also given which show that the new fastest LI transforms are able to give polynomial expansions with less number of nonzero spectral coefficients compared to those based on the existing ones for the majority of binary benchmark functions. It should be noted that when the polynomial expansions based on spectral transforms are used for spectral synthesis [KSA00], the spectral coefficients for the represented function are stored in memory and the function output is obtained by multiplying the spectral coefficients with the basis functions provided by suitably designed function generator and summing them up. Thus, the complexity of the spectral realization of the given function is proportional to the number of nonzero coefficients that need to be stored in memory. Less number of nonzero spectral coefficients implies smaller storage requirement for the hardware realization of the function.

4.2.1 Basic definitions and properties for fastest LI transforms over GF(2)

Definition 4.2.1. Let $\vec{F} = [f_0, f_1, ..., f_{2^n-1}]^T$ be the truth vector of an *n*-variable binary function $f(\vec{x})$ in natural binary ordering, where *T* denotes matrix transpose operator. Then, for any LI matrix M_n

$$\vec{F} = M_n \vec{A} \ , \tag{4.4}$$

where $\vec{A} = [a_0, a_1, ..., a_{2^n-1}]^T$ is the coefficient column vector for the particular LI transform matrix M_n.

Conversely,

$$\vec{A} = M_n^{-1} \vec{F} \ , \tag{4.5}$$

where M_n^{-1} is the inverse of M_n in GF(2).

Definition 4.2.2. Let g_j represent the *n*-variable binary function whose truth vector is given by column *j* of a particular LI transform matrix M_n $(0 \le j \le 2^n - 1)$. Then, based on M_n any *n*-variable binary function can be expressed as a polynomial expansion over GF(2)

$$f(\vec{x}) = \sum_{j=0}^{2^n-1} a_j g_j \ , \tag{4.6}$$

where $\vec{x} = [x_n, x_{n-1}, \ldots, x_1]$ and a_j $(0 \le j \le 2^n - 1)$ is the *j*-th element of the coefficient column vector $\vec{A}$ for M_n.

Definition 4.2.3. Let $M_n = [m_{i,j}]$ be a matrix of size $2^n \times 2^n$, where $0 \le i, j \le 2^n - 1$. The set $Q_i(M_n)$ is defined as the set of column index numbers of all 1s in the row *i* of M_n. That is, $Q_i(M_n) = \{j \mid m_{i,j} = 1\}$. Similarly, the set $Q^j(M_n)$ is defined as the set of row index numbers of all 1s in the column *j* of M_n, $Q^j(M_n) = \{i \mid m_{i,j} = 1\}$. The

cardinality of $Q_i(M_n)$ and $Q^j(M_n)$ are denoted by $|Q_i(M_n)|$ and $|Q^j(M_n)|$, respectively.

In [RF02], four fastest LI transform matrices over GF(2) which have the most efficient computational complexity were identified. All the fastest transform matrices possess both fast forward and inverse transforms and can be recursively defined in terms of submatrices O_{n-1}, Y_{n-1}, and M_{n-1}, where O_{n-1} is a $2^{n-1}\times 2^{n-1}$ matrix with all its elements 0 and Y_{n-1} is a $2^{n-1}\times 2^{n-1}$ matrix with all its elements 0 except one element located at the corner of each matrix depending on the position of Y_{n-1}.

Definition 4.2.4. Let M_n be a $2^n \times 2^n$ LI transform matrix over GF(2). All the fastest LI transform matrices introduced in [RF02] are constructed from one O_{n-1} submatrix, one Y_{n-1} submatrix, and two M_{n-1} submatrices. Their forward transformations are given by the following equations.

$$M_n = \begin{bmatrix} M_{n-1} O_{n-1} \\ Y_{n-1} M_{n-1} \end{bmatrix} \tag{4.7}$$

$$M_n = \begin{bmatrix} M_{n-1} Y_{n-1} \\ O_{n-1} M_{n-1} \end{bmatrix} \tag{4.8}$$

$$M_n = \begin{bmatrix} Y_{n-1} M_{n-1} \\ M_{n-1} O_{n-1} \end{bmatrix} \tag{4.9}$$

$$M_n = \begin{bmatrix} O_{n-1} M_{n-1} \\ M_{n-1} Y_{n-1} \end{bmatrix}. \tag{4.10}$$

The location of 1 entry inside the Y_{n-1} submatrix is at bottom left corner for (4.7), at top right corner for (4.8), at top left corner for (4.9), and at bottom right corner for (4.10).

In order to differentiate the four fastest LI transform matrices in Definition 4.2.4, the symbols M_n^a, M_n^b, M_n^c, and M_n^d are used to denote fastest LI transform matrices

of dimension $2^n \times 2^n$ that satisfy (4.7), (4.8), (4.9), and (4.10), respectively. In [RF02], M_n^a and M_n^b are grouped into class A1 whereas M_n^c and M_n^d belong to class A2.

Example 4.2.1. Let $f_1(\vec{x}) = \overline{x_3}\left(\overline{x_1} \vee \overline{x_2}\right) \vee \left(x_1 \oplus x_2\right)$ and $f_2(\vec{x}) = \overline{x_3}\,\overline{x_2} \vee \overline{x_3} x_1$ be two three-variable binary functions with truth vectors $\overrightarrow{F_1} = [1,1,1,0,0,1,1,0]^T$ and $\overrightarrow{F_2} = [1,1,0,1,0,0,0,0]^T$, respectively. By (4.7), the matrix M_3^a is given by

$$M_3^a = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

where $g_0 = \overline{x_3}\,\overline{x_2} \vee x_2 x_1$, $g_1 = \overline{x_3}\,\overline{x_2} x_1$, $g_2 = \overline{x_3} x_2$, $g_3 = \overline{x_3} x_2 x_1$, $g_4 = x_3\left(\overline{x_2} \vee x_1\right)$, $g_5 = x_3 \overline{x_2} x_1$, $g_6 = x_3 x_2$, and $g_7 = x_3 x_2 x_1$. By (4.5), the coefficient column vectors for f_1 and f_2 based on M_3^a are $\overrightarrow{A_1} = \left(M_3^a\right)^{-1} \cdot \overrightarrow{F_1} = [1,0,1,0,0,1,1,0]^T$ and $\overrightarrow{A_2} = \left(M_3^a\right)^{-1} \cdot \overrightarrow{F_2} = [1,0,0,0,0,0,0,1]^T$, respectively. Hence, using these sets of basis functions, the switching functions f_1 and f_2 can be represented as LI polynomial expansions over GF(2) $f_1 = g_0 + g_2 + g_5 + g_6$ and $f_2 = g_0 + g_7$, respectively.

Definition 4.2.5. Let M_n^θ be $2^n \times 2^n$ fastest LI transform matrix of type θ, where $\theta \in \{a, b, c, d\}$. Due to its recursive definition, M_n^θ can be represented (factorized) as product of n sparse matrices over GF(2) as follows

$$M_n^\theta = \prod_{j=n-1}^{0} K_j^\theta = K_{n-1}^\theta K_{n-2}^\theta \ldots K_1^\theta K_0^\theta, \qquad (4.11)$$

where K_j^θ $(0 \le j \le n-1)$ represents the $2^n \times 2^n$ sparse matrices in the factorized representation of M_n^θ.

Property 4.2.1. All the sparse matrices K_j^θ contain at most two nonzero elements in

each row and column. Let $\otimes$ and I_n denote Kronecker product and identity matrix of dimension $2^n \times 2^n$, respectively. Then the general formulae for K_j^{θ} are

$$K_j^a = I_{n-j-1} \otimes \alpha_1 \left(I_{j+1} \right) \tag{4.12}$$

$$K_j^b = R_2{}^2 \left(K_j^a \right) \tag{4.13}$$

$$K_j^c = R_{1,j} \left(K_j^b \right) \tag{4.14}$$

$$K_j^d = R_2{}^2 \left(K_j^c \right). \tag{4.15}$$

Example 4.2.2. In Example 4.2.1, the matrix M_3^a has been constructed by using the recursive definition in (4.7). Based on Definition 4.2.5 and Property 4.2.1, M_3^a can also be obtained from the product of three sparse matrices K_2^a, K_1^a, and K_0^a where $K_2^a = I_0 \otimes \alpha_1(I_3)$, $K_1^a = I_1 \otimes \alpha_1(I_2)$, and $K_0^a = I_2 \otimes \alpha_1(I_1)$ such that

$$\begin{aligned} M_3^a &= K_2^a \cdot K_1^a \cdot K_1^a \\ &= \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 1&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&1 \end{bmatrix} \\ &= \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 1&0&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&0&0&0&1&0&1&1 \end{bmatrix}. \end{aligned}$$

Using the factorized representation of fastest LI transform matrices based on (4.12)–(4.15), their butterfly diagrams can be easily constructed. Fig. 4.1 shows the forward butterfly diagrams for all fastest LI transform matrices over GF(2) of order $8 = 2^3$. In Fig. 4.2, the butterfly diagrams that correspond to the factorized

representations of inverse fast binary FPRME transforms in polarities zero and seven are given. All the operations on the butterfly diagrams are performed over GF(2). It can be easily noticed that the butterfly diagram structures of the fast FPRME transforms in polarity zero and polarity seven correspond to the butterfly diagram structures of fastest LI transforms over GF(2) and that the fast FPRME transforms require larger number of additions than the fastest LI transforms.

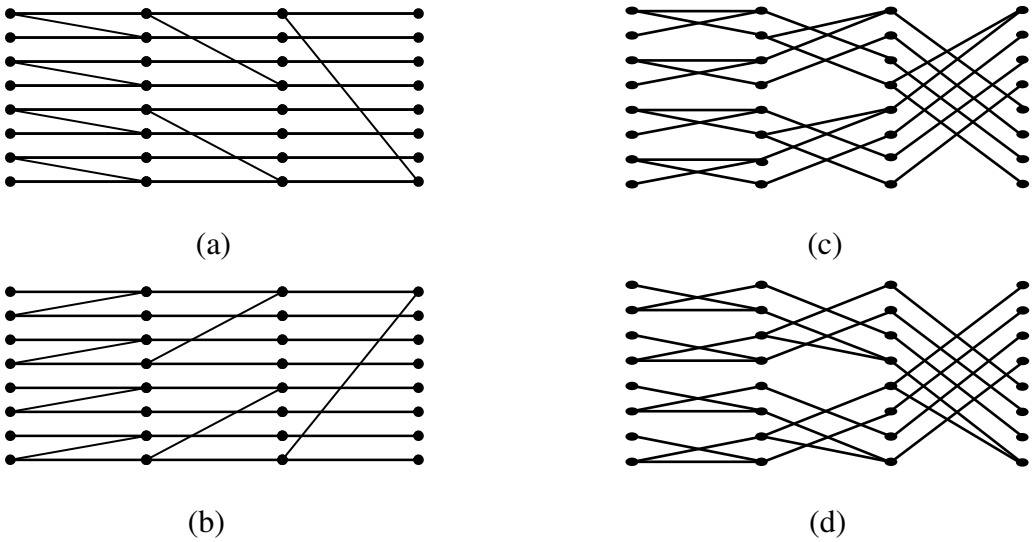


Figure 4.1: Butterly diagrams of all fastest LI transform matrices of order 8:

(a) M_3^a; (b) M_3^b; (c) M_3^c; (d) M_3^d.

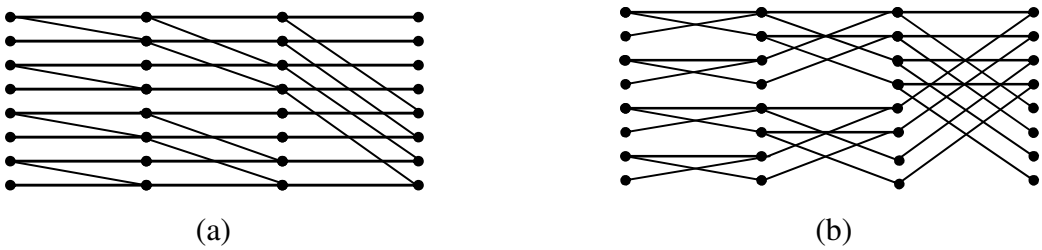


Figure 4.2: Butterly diagrams for fast FPRME transforms:

(a) $\left(RM_3^0\right)^{-1}$; (b) $\left(RM_3^7\right)^{-1}$.

Since there are four different fastest LI transforms, based on Definition 4.2.1 four coefficient column vectors corresponding to each fastest LI transform need to be calculated separately in order to obtain the best representation for a particular binary

function based on them. It means that the calculation requires computation of four different transforms or four different hardware that correspond to each transform. However, due to the various relationships that exist between the fastest LI transforms the required resources can be reduced. In what follows properties describing those useful relationships as well as an example illustrating their usage for reducing the needed resources are presented.

Property 4.2.2. Both M_n^a and M_n^b are self-inverse whereas M_n^c and M_n^d are inverses of each other. That is, $\left(M_n^a\right)^{-1} = M_n^a$, $\left(M_n^b\right)^{-1} = M_n^b$, and $\left(M_n^c\right)^{-1} = M_n^d$.

Property 4.2.3. M_n^a is M_n^c that is flipped vertically. Similarly, M_n^b is vertically-flipped M_n^d. Also, M_n^c can be obtained by flipping M_n^b horizontally and flipping M_n^d horizontally produces M_n^a. As a result, by Definition 4.1.9 class A2 forward LI transform matrices can be obtained from class A1 forward transform matrices through premultiplication or postmultiplication by the permutation matrix $P_n^{2^n-1}$ as given below.

$$M_n^c = P_n^{2^n-1} M_n^a \tag{4.16}$$

$$M_n^d = P_n^{2^n-1} M_n^b \tag{4.17}$$

$$M_n^d = M_n^a P_n^{2^n-1} \tag{4.18}$$

$$M_n^c = M_n^b P_n^{2^n-1}. \tag{4.19}$$

Conversely, by applying matrix inverse operator on (4.16)–(4.19) equations for generating class A1 forward transform matrices from class A2 forward LI transform matrices through premultiplication or postmultiplication by $P_n^{2^n-1}$ can be obtained.

The butterfly diagrams for M_3^c and M_3^d that correspond to (4.16)–(4.19) are shown in Figs. 4.3 and 4.4. It can be seen that the number of necessary additions of the butterfly diagrams in Figs. 4.3 and 4.4 are the same with those in Fig. 4.1 since the

permutation matrix does not incur any addition. However the butterfly diagrams in Figs. 4.3 and 4.4 are simpler and have more regular structures than those in Fig. 4.1.

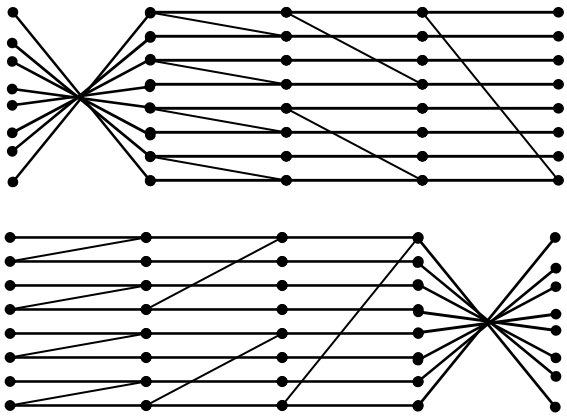


Figure 4.3: Butterfly diagrams with permutations for M_3^d.

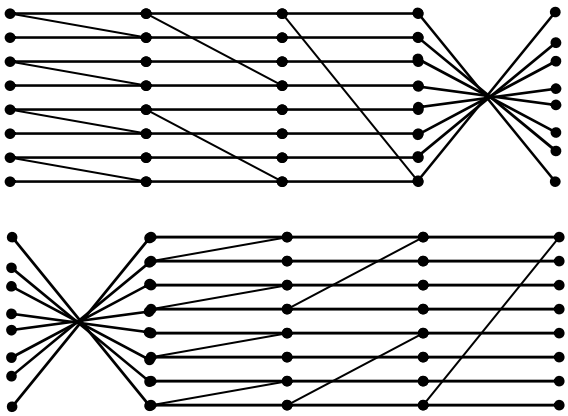


Figure 4.4: Butterfly diagrams with permutations for M_3^c.

Property 4.2.4. Let $\vec{F}$ be the truth vector of a particular *n*-variable binary function $f(\vec{x})$. In addition, let $\vec{A}_{n1}$ and $\vec{A}_{n2}$ denote the column coefficient vectors for $f(\vec{x})$ with respect to M_n^c and M_n^d, respectively. Then,

$$R_0(\vec{A}_{n1}) = M_n^c R_0(\vec{F}) \tag{4.20}$$

$$R_0(\vec{A}_{n2}) = M_n^d R_0(\vec{F}). \tag{4.21}$$

Proof: By Property 4.2.2,

$$\vec{A}_{n1} = M_n^d \vec{F} \tag{4.22}$$

$$\vec{A}_{n2} = M_n^c \vec{F} \,. \tag{4.23}$$

From Fig. 4.1 it can be seen that the butterfly diagrams of M_n^c and M_n^d are vertical flip of each other. Substituting this into (4.22) and (4.23), (4.20) and (4.21) are obtained.

Property 4.2.5. Let $\vec{A}_{n1}$, $\vec{A}_{n2}$, and $\vec{F}$ be vectors as defined in Property 4.2.4 whereas $\vec{A}_{n3}$ and $\vec{A}_{n4}$ denote the coefficient column vectors that are obtained by $\left(M_n^b\right)^{-1}\vec{F}$ and $\left(M_n^a\right)^{-1}\vec{F}$, respectively. Then, by Property 4.2.3

$$\vec{A}_{n1} = R_0\left(\vec{A}_{n3}\right) \tag{4.24}$$

and

$$\vec{A}_{n2} = R_0\left(\vec{A}_{n4}\right). \tag{4.25}$$

Property 4.2.6. The two fastest LI transform matrices that belong to class A1 are transposes of each other. That is

$$M_n^a = \left(M_n^b\right)^T, \tag{4.26}$$

where *T* denotes matrix transpose operator.

Due to the above properties, only one fastest LI matrix needs to be constructed in order to obtain the polynomial expansions for a binary function based on all the four fastest LI transforms. The coefficient column vector that corresponds to the simplest fastest LI transform matrix can first be computed by (4.5). The remaining coefficient column vectors can then be generated by simply applying appropriate permutation matrices or R_0 operator without deriving the other fastest LI transform matrices.

Example 4.2.3. Let us derive all the coefficient column vectors for a binary function

by constructing only M_n^a, where $\vec{A}_{n1}$, $\vec{A}_{n2}$, $\vec{A}_{n3}$, and $\vec{A}_{n4}$ represent the coefficient column vectors for M_n^c, M_n^d, M_n^b, and M_n^a, respectively. Then first M_n^a need to be obtained by using either (4.7) or (4.11). After M_n^a is obtained, $\vec{A}_{n4}$ can then be computed from $\vec{F}$, the truth vector of the binary function by (4.5). Next, $\vec{A}_{n1}$ is calculated either by $\vec{A}_{n1} = M_n^a P_n^{2^n-1} \vec{F}$ or $\vec{A}_{n1} = M_n^a R_0(\vec{F})$. Finally, $\vec{A}_{n2}$ and $\vec{A}_{n3}$ are simply $R_0(\vec{A}_{n4})$ and $R_0(\vec{A}_{n1})$, respectively.

In the following properties, the total number and location of 1s in the fastest LI transform matrices are given. The properties are useful for calculating selected LI spectral coefficients as well as for establishing the properties of the spectra for special classes of input functions.

Property 4.2.7. Let M_n be any fastest LI transform matrix of size $2^n \times 2^n$. Then the total number of 1s in M_n is always equal to $2^{n+1} - 1$.

Property 4.2.8. The value of $\left|Q_i\left(M_n^b\right)\right|$ is equal to $L_B(<i>_2)$, where $<i>_2$ is the binary representation of i $(0 \le i \le 2^n - 1)$. Furthermore, the set $Q_i\left(M_n^b\right) = \{q_k \mid q_k = i + 2^k - 1 \text{ and } 0 \le k < L_B(<i>_2)\}$.

Property 4.2.9. The value of $\left|Q^j\left(M_n^b\right)\right|$ is equal to $L_B(<j + 1>_2)$, where $0 \le j \le 2^n - 1$ and $<j + 1>_2$ is the binary representation of $(j+1)$. Also, the set of row index numbers of all 1s in column j of M_n^b is given by $Q^j\left(M_n^b\right) = \{q_k \mid q_k = j - (2^k - 1) \text{ and } 0 \le k < L_B(<j+1>_2)\}$.

By Property 4.2.6, Properties 4.2.8 and 4.2.9 can be easily modified to obtain the positions of 1s in the rows and columns of M_n^a as follows:

Property 4.2.10. The number of 1s in row i ($0 \le i \le 2^n - 1$) of M_n^a, i.e., $\left|Q_i\left(M_n^a\right)\right|$, is equal to $L_B(<i + 1>_2)$, where $<i + 1>_2$ is the binary representation of $(i+1)$. Their column index numbers are the elements of the set $Q_i\left(M_n^a\right)$, where $Q_i\left(M_n^a\right) = \{q_k \mid q_k = i-(2^k-1) \text{ and } 0 \le k < L_B(<i+1>_2)\}$.

Property 4.2.11. The column j ($0 \le j \le 2^n - 1$) of M_n^a has $L_B(<j>_2)$ 1s, where $<j>_2$ is the binary representation of j. The 1s are located at rows $Q^j\left(M_n^a\right)$, where $Q^j\left(M_n^a\right) = \{q_k \mid q_k = j + 2^k - 1 \text{ and } 0 \le k \le L_B(<j>_2)\}$.

Properties 4.2.8–4.2.11 above can be further extended to obtain the numbers and locations of 1s in M_n^c and M_n^d by making use of Property 4.2.3.

Theorem 4.2.1. [RF02] The number of GF(2) additions required to compute the coefficient vector of a fastest LI transform for an n-variable binary function is $2^n - 1$.

Proof: For a transformation of an n-variable binary function, the fast transforms may be simply derived by repeating the fast transform of an $(n-1)$-variable binary function twice, placing one below another and introducing the n-th stage transform which simply involves one GF(2) addition. As such, the computational cost is

$$\begin{aligned} B_n &= 2B_{n-1} + 1 \\ &= 2^{n-k} B_{n-(n-k)} + \sum_{l=0}^{n-k-1} 2^l, \\ &= 2^n - 1 \end{aligned} \tag{4.27}$$

which is always equal or smaller than $n \cdot 2^{n-1}$, the computational cost of FPRME.

4.2.2 Fastest LI transform matrices over GF(2) with reordering and permutation

The four fastest LI transform matrices presented in Section 4.2.1 are generalized into a larger group of fastest LI transforms introduced in this section. For simplicity, and to

preserve the regularity and low computational complexity of the fastest LI transforms, the new transforms are constructed from the factorized transform matrices defined in (4.12)–(4.15) and one permutation matrix. The notation used for the transforms is defined below.

Definition 4.2.6. A $2^n \times 2^n$ fastest LI transform matrix with reordering and permutation over GF(2) $M_n^{\theta}(\varphi, \sigma, p)$ is defined as

$$M_n^{\theta}(\varphi,\sigma,p) = \begin{cases} \left[\prod_{j=n}^{\sigma} K_{\varphi_j}^{\theta}\right] \times P_n^p \times \left[\prod_{j=\sigma-1}^{1} K_{\varphi_j}^{\theta}\right], & \text{if } \sigma \neq n+1 \\ P_n^p \times \prod_{j=n}^{1} K_{\varphi_j}^{\theta}, & \text{otherwise} \end{cases}, \tag{4.28}$$

where $\theta \in \{a,b,c,d\}$ is the type of the LI transform matrix, φ is the ordering of the n sparse matrices in the factorized representation of $M_n^{\theta}(\varphi, \sigma, p)$, σ $(1 \le \sigma \le n+1)$ is the position where the permutation matrix is added in the butterfly diagram, p is the type of the added permutation matrix, and $< p >_{10} = < p_n, p_{n-1}, ..., p_1 >_2$.

Property 4.2.12. The ordering φ is an n-digit string whose digits take values from 0 to $n-1$ and no two different digits in it have the same values,

$$\varphi = < \varphi_n, \varphi_{n-1}, \ldots, \varphi_1 > \tag{4.29}$$

where $\varphi_i \in \{0, 1, \ldots, n-1\}$ and $\varphi_i = \varphi_j$ iff $i = j\ (1 \le i, j \le n)$.

Example 4.2.4. Let $f(\vec{x})$ be a three-variable binary function with truth vector $\vec{F} = [0,1,1,0,1,1,0,0]^T$. Then by Definitions 4.1.9, 4.2.1, and 4.2.6 the coefficient column vectors of fastest LI transforms M_3^a, $M_3^a(102,2,3)$, and $M_3^b(012,2,2)$ for $f(\vec{x})$ are $\overrightarrow{A_0}$, $\overrightarrow{A_1}$, and $\overrightarrow{A_2}$, respectively where

$$\overrightarrow{A_0} = \left(K_2^a \cdot K_1^a \cdot K_0^a\right)^{-1} \cdot \vec{F},$$

$$\overrightarrow{A_1} = \left(K_1^a \cdot K_0^a \cdot P_3^3 \cdot K_2^a\right)^{-1} \cdot \vec{F},$$

$$\overrightarrow{A_2} = \left(K_0^b \cdot K_1^b \cdot P_3^2 . K_2^b\right)^{-1} \cdot \vec{F}.$$

By (4.12) and Definition 4.1.7,

$$K_2^a = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 1&0&0&0&0&0&0&1 \end{bmatrix}, \quad K_1^a = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 1&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&1 \end{bmatrix}, \text{ and } K_0^a = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1&1 \end{bmatrix}.$$

By (4.13) and Definitions 4.1.5 and 4.1.6,

$$K_2^b = R_2\left(R_2\left(\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 1&0&0&0&0&0&0&1 \end{bmatrix}\right)\right) = \begin{bmatrix} 1&0&0&0&0&0&0&1\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}.$$ Similarly,

$$K_1^b = \begin{bmatrix} 1&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&1\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix} \text{ and } K_0^b = \begin{bmatrix} 1&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&1 \end{bmatrix}.$$

Substituting the factorized matrices to (4.5), it is obtained that

$$\overrightarrow{A_0} = [0,1,1,1,1,0,0,1]^T,$$

$$\overrightarrow{A_1} = [1,1,1,0,1,0,0,0]^T,$$

and

$$\overrightarrow{A_2} = [0,0,1,1,0,0,0,1]^T.$$

Hence, among the three fastest LI transform matrices $M_3^b(012,2,2)$ gives the smallest number of nonzero spectral coefficients for $f\left(\vec{x}\right)$, followed by $M_3^a(102,2,3)$ and M_3^a, respectively.

Property 4.2.13. From (4.28) and Definition 4.2.5, it can be clearly seen that when $\varphi_i = i-1$ for $1 \le i \le n$ and $p = 0$

$$M_n^{\theta}(\varphi, \sigma, p) = M_n^{\theta}, \tag{4.30}$$

where $\theta \in \{a, b, c, d\}$, M_n^{θ} is the $2^n \times 2^n$ existing fastest LI transform matrix of type θ, and $M_n^{\theta}(\varphi, \sigma, p)$ is as in Definition 4.2.6. Hence, the existing fastest LI transform matrices are special cases of fastest LI transform matrices with reordering and permutation.

In Section 4.2.1, some properties on the relationships between the existing fastest LI transform matrices as well as numbers and locations of 1s inside them have been presented and it has been shown that these properties reduce the amount of resources needed to generate polynomial expansions based on all four fastest LI transform matrices. Since the numbers of all possible fastest LI transforms with reordering and permutation are much greater than four, computation of all polynomial expansions based on them requires even more extensive resources. Hence, it is important to obtain similar properties on them to reduce the required resources as much as possible. In what follows, such properties are listed.

Properties 4.2.14–4.2.16 below are valid for class A1 fastest LI transform matrices.

Property 4.2.14. $M_n^{\theta}(\varphi, \sigma, 0)$ always yields the same matrix regardless of φ and σ.

Property 4.2.15. Let $M_n^{\theta}(\varphi_1, \sigma_1, p_1)$ and $M_n^{\theta}(\varphi_2, \sigma_2, p_2)$ be any two fastest LI transform matrices and $\beta(\varphi, \sigma)$ be the set defined as

$$\beta(\varphi, \sigma) = \begin{cases} \{\varphi_j \mid n \le j \le \sigma\}, & \text{if } \sigma \ne 1 \\ \varnothing, & \text{otherwise.} \end{cases} \tag{4.31}$$

Then,

$$M_n^{\theta}(\varphi_1, \sigma_1, p_1) = M_n^{\theta}(\varphi_2, \sigma_2, p_2) \text{ if } \beta(\varphi_1, \sigma_1) = \beta(\varphi_2, \sigma_2) \text{ and } \sigma_1 = \sigma_2 \text{ and } p_1 = p_2 \tag{4.32}$$

$$M_n^{\theta}(\varphi_1, \sigma_1, p_1) = R_2^{\,2}\left(M_n^{\theta}(\varphi_2, \sigma_2, p_2)^T\right) \text{ if } p_1 = p_2 \text{ and } \sigma_2 = n + 2 - \sigma_1 \text{ and}$$
$$(\beta(\varphi_1, \sigma_1) \cup \beta(\varphi_2, \sigma_2) = \{0, 1, \ldots, n-1\}). \tag{4.33}$$

Example 4.2.5. $M_4^b(0123,3,1) = M_4^b(0132,3,1) = M_4^b(1023,3,1) = M_4^b(1032,3,1)$.

Property 4.2.16. For any two class A1 fastest LI transform matrices with the same θ, p, and φ,

$$M_n^{\theta}(\varphi,1, p) = R_2^{\,2}\left(M_n^{\theta}(\varphi, n+1, p)^T\right). \tag{4.34}$$

The following properties are valid for all fastest LI transform matrices with reordering and permutation.

Property 4.2.17. Let $M_n^{\theta}(\varphi, \sigma, p)$ be any $2^n \times 2^n$ fastest LI transform matrix with reordering and permutation over GF(2). Then,

$$\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1} = R_2^{\,2}\left(\left(M_n^{\theta}(\varphi, \sigma, p)\right)^T\right). \tag{4.35}$$

It follows from (4.35) that for a pair of class A1 fastest LI transform matrices $M_n^{\theta}(\varphi_1, \sigma_1, p_1)$ and $M_n^{\theta}(\varphi_2, \sigma_2, p_2)$ which satisfies all the conditions specified in (4.33)

$$\left(M_n^{\theta}(\varphi_1, \sigma_1, p_1)\right)^{-1} = \left(M_n^{\theta}(\varphi_2, \sigma_2, p_2)\right). \tag{4.36}$$

Property 4.2.18. Let $M_n^a(\varphi, \sigma, p)$, $M_n^b(\varphi, \sigma, p)$, $M_n^c(\varphi, \sigma, p)$, and $M_n^d(\varphi, \sigma, p)$ be the four different types of $2^n \times 2^n$ fastest LI transform matrices with reordering and permutation. Then they are related by the following equations.

$$\left(M_n^a(\varphi, \sigma, p)\right)^{-1} = R_2^{\,2}\left(\left(M_n^b(\varphi, \sigma, p)\right)^{-1}\right) \tag{4.37}$$

$$\left(M_n^c(\varphi, \sigma, p)\right)^{-1} = R_2^{\,2}\left(\left(M_n^d(\varphi, \sigma, p)\right)^{-1}\right) \tag{4.38}$$

Due to the regular structure of the butterfly diagrams for the fastest LI transform matrices, the number and location of 1s in a fastest LI transform matrix with reordering and permutation can be determined. In the following properties, some algorithms to calculate them are presented.

Property 4.2.19. By Property 4.2.14, the number and location of 1s in $M_n^{\theta}(\varphi,\sigma,0)$ for $\theta \in \{a,b\}$ are the same as those in M_n^{θ}, which are presented in Properties 4.2.7–4.2.11.

Property 4.2.20. Recall that the total number of 1s in $M_n^c(\varphi,\sigma,p)$ is equal to $\sum_{i=0}^{2^n-1}\left|Q_i\left(M_n^c(\varphi,\sigma,p)\right)\right| = \sum_{j=0}^{2^n-1}\left|Q^j\left(M_n^c(\varphi,\sigma,p)\right)\right|$. When p = 0, its value is given by

$$\sum_{i=0}^{2^n-1}\left|Q_i\left(M_n^c(\varphi,\sigma,0)\right)\right| = \sum_{j=0}^{2^n-1}\left|Q^j\left(M_n^c(\varphi,\sigma,0)\right)\right| = \begin{cases} (5\cdot 2^{n-1})-2, & \text{if } \varphi_n = 0 \\ 2^n + \sum_{l=1}^{n}\left((2+y_l)\cdot 2^{n-\varphi_l-1}\right), & \text{otherwise,} \end{cases} \tag{4.39}$$

where y_l is obtained by the procedure given in Fig. 4.5. In the procedure, z is the subscript value of the digit in φ whose value is equal to 0 such that $\varphi = <\varphi_n, \varphi_{n-1}, ..., \varphi_{z+1}, 0, \varphi_{z-1}, ..., \varphi_1>$.

Property 4.2.21. Let S_p be the set of subscript values of all the bits in $<p_n, p_{n-1}, ..., p_1>$ whose values are equal to 1, $S_p = \{i \mid p_i = 1\}$ (recall that $<p>_{10} = <p_n, p_{n-1}, ..., p_1>_2$). Then the sets $Q_{Row}\left(M_n^a(\varphi,\sigma,p)\right)$, $Q^{Col}\left(M_n^a(\varphi,\sigma,p)\right)$, $Q_{Row}\left(M_n^c(\varphi,\sigma,p)\right)$, and $Q^{Col}\left(M_n^c(\varphi,\sigma,p)\right)$ can be obtained by the procedures shown in Figs. 4.6, 4.7, 4.8, and 4.9, respectively where $0 \le Row, Col \le 2^n - 1$. In the procedures, S_2 is the set of binary strings whose members have one-to-one mapping to the members of the sets to be obtained such that the members of S_2 are the binary representations of the members of the set to be obtained.

Property 4.2.22. Recall that $< p >_{10} =< p_n, p_{n-1}, ..., p_1 >_2$. The total number of 1s in $M_n^a(\varphi, \sigma, p)$ is equal to $2^{n+1} - 1 + U$, where

$$U = \sum_{i=n}^{\sigma} \sum_{j=\sigma-1}^{1} \begin{cases} \left(2^{n-\max(\varphi_i, \varphi_j)-1}\right), & \text{if } p_l = 1 \forall l, 1 \le l \le \min(\varphi_i, \varphi_j) + 1 \\ 0, & \text{otherwise.} \end{cases} . \quad (4.40)$$

By (4.40), the number of 1s in $M_n^a(\varphi, \sigma, p)$ is always $2^{n+1} - 1$ when the least significant bit (LSB) in $<p>_2 = p_1$ is equal to 0.

```
For l = z to n
   y_l = −1;
For l = 1 to z − 1
   y_l = 0;
For loop_var = n to z + 1
  {S_1 = {0, 1, ..., φ_loop_var − 1};
   For loop_var1 = loop_var − 1 to z
    {If φ_loop_var1 < φ_loop_var
      Remove φ_loop_var1 from S_1;}
   For loop_var1 = z − 1 to 1
    {If φ_loop_var1 < φ_loop_var
      {If φ_loop_var1 ∈ S_1
        Remove φ_loop_var1 from S_1;
      Else
        Add φ_loop_var1 to S_1;}
     Else
      {If S_1 = ∅
        y_loop_var1 = y_loop_var1 + 1;}}}
```

Figure 4.5: Procedure to determine y_l.

Property 4.2.23. Let $Q_{Row}\left(M_n^b(\varphi,\sigma,p)\right)$ and $Q_{2^n-1-Row}\left(M_n^a(\varphi,\sigma,p)\right)$ be the sets of column index numbers of 1s in row *Row* of $M_n^b(\varphi,\sigma,p)$ and row $(2^n-1-Row)$ of $M_n^a(\varphi,\sigma,p)$, respectively. If $Q_{2^n-1-Row}\left(M_n^a(\varphi,\sigma,p)\right)=\{q_i \mid 0\le i<\left|Q_{2^n-1-Row}\left(M_n^a(\varphi,\sigma,p)\right)\right|\}$ then $Q_{Row}\left(M_n^b(\varphi,\sigma,p)\right)=\{q'_i \mid q'_i=2^n-1-q_i \text{ and } 0\le i<\left|Q_{2^n-1-Row}\left(M_n^a(\varphi,\sigma,p)\right)\right|\}$. Similarly,

$$Q^{Col}\left(M_n^b(\varphi,\sigma,p)\right)=\{q'_i \mid q'_i=2^n-1-q_i \text{ and } 0\le i<\left|Q^{2^n-1-Col}\left(M_n^a(\varphi,\sigma,p)\right)\right|\}$$

$$\text{if } Q^{2^n-1-Col}\left(M_n^a(\varphi,\sigma,p)\right)=\{q_i \mid 0\le i<\left|Q^{2^n-1-Col}\left(M_n^a(\varphi,\sigma,p)\right)\right|\} \quad (4.41)$$

$$Q_{Row}\left(M_n^d(\varphi,\sigma,p)\right)=\{q'_i \mid q'_i=2^n-1-q_i \text{ and } 0\le i<\left|Q_{2^n-1-Row}\left(M_n^c(\varphi,\sigma,p)\right)\right|\}$$

$$\text{if } Q_{2^n-1-Row}\left(M_n^c(\varphi,\sigma,p)\right)=\{q_i \mid 0\le i<\left|Q_{2^n-1-Row}\left(M_n^c(\varphi,\sigma,p)\right)\right|\} \quad (4.42)$$

$$Q^{Col}\left(M_n^d(\varphi,\sigma,p)\right)=\{q'_i \mid q'_i=2^n-1-q_i \text{ and } 0\le i<\left|Q^{2^n-1-Col}\left(M_n^c(\varphi,\sigma,p)\right)\right|\}$$

$$\text{if } Q^{2^n-1-Col}\left(M_n^c(\varphi,\sigma,p)\right)=\{q_i \mid 0\le i<\left|Q^{2^n-1-Col}\left(M_n^c(\varphi,\sigma,p)\right)\right|\}. \quad (4.43)$$

Example 4.2.6. Let us find the location of 1s in row seven of $M_3^a(201,3,5)$. By Definition 4.2.6,

$$M_3^a(201,3,5)=K_2^aP_3^5K_0^aK_1^a=\begin{bmatrix}0&0&0&0&1&1&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&1&0&1&1\\0&0&0&0&0&0&1&0\\1&1&0&0&0&0&0&0\\1&0&0&0&0&0&0&0\\1&0&1&1&0&0&0&0\\0&0&1&0&1&1&0&0\end{bmatrix}.$$

By Property 4.2.21, the steps for calculating $Q_7\left(M_n^a(201,3,5)\right)$, the set of the column index numbers of 1s in row seven of $M_3^a(201,3,5)$ are

- $Row = 7$, $<Row>_2 = 111$, $S_p = \{1, 3\}$, $S_2 = \{111\}$.
- $loop\_var = 3 \rightarrow S_2 = \{111, 000\}$.
- Invert bit s_p of all elements of $S_2 \rightarrow S_2 = \{010, 101\}$.

Compute $<Row>_2$, *the n-bit binary representation of Row;*

Set $S_2 = \{<Row>_2\}$;

For loop_var = n to σ

{*Compute* $<Row + 1>_2$, *the n-bit binary representation of (Row + 1);*

If $\varphi_{loop\_var} + 1 < L_B(<Row + 1>_2)$

Add the n-bit binary representation of $(Row - 2^{\varphi_{loop\_var}+1} + 1)$ *to* S_2;}

For every member of S_p, s_p

Invert bit s_p *of all elements of* S_2;

For loop_var = $\sigma - 1$ *to 1*

{*For every element of* S_2, s_i

{*If* $\varphi_{loop\_var} + 1 < L_B(s_i + 1)$

Add the n-bit binary representation of $(<s_i>_{10} - 2^{\varphi_{loop\_var}+1} + 1)$ *to* S_2;}}

Figure 4.6: Procedure to obtain $Q_{Row}\left(M_n^a(\varphi, \sigma, p)\right)$.

Compute $<Col>_2$, *the n-bit binary representation of Col;*

Set $S_2 = \{<Col>_2\}$;

For loop_var = $\sigma - 1$ *to 1*

{ *If* $\varphi_{loop\_var} + 1 < L_B(<Col >_2)$

Add the n-bit binary representation of $(Col + 2^{\varphi_{loop\_var}+1} - 1)$ *to* S_2;}

For every member of S_p, s_p

Invert bit s_p *of all elements of* S_2;

For loop_var = n to σ

{*For every element of* S_2, s_i

{*If* $\varphi_{loop\_var} + 1 < L_B(s_i)$

Add the n-bit binary representation of $(<s_i>_{10} + 2^{\varphi_{loop\_var}+1} - 1)$ *to* S_2;}}

Figure 4.7: Procedure to obtain $Q^{Col}\left(M_n^a(\varphi, \sigma, p)\right)$.

- $loop\_var = 2 \rightarrow S_2 = \{010, 101, 100\}$.
- $loop\_var = 1 \rightarrow S_2 = \{010, 101, 100\}$.

Hence, the 1s in row seven of $M_3^a(201,3,5)$ are located at columns 2, 4, and 5 $\rightarrow$ $Q_7\left(M_n^a(201,3,5)\right) = \{2, 4, 5\}$.

Compute <Row>$_2$, the n-bit binary representation of Row;
Set S_2 = {<Row>$_2$};
For loop_var = n to σ
{For every element of S_2, s_i
{Invert bit φ_{loop_var} + 1 of s_i;
If $L_B(s_i) = \varphi_{loop\_var} + 1$
Add $(s_i - 1)$ to S_2;}}
For every member of S_p, s_p
Invert bit s_p of all elements of S_2;
For loop_var = $\sigma - 1$ to 1
{For every element of S_2, s_i
{Invert bit φ_{loop_var} + 1 of s_i;
If $L_B(s_i) = \varphi_{loop\_var} + 1$
Add $(s_i - 1)$ to S_2;}}

Figure 4.8: Procedure to determine $Q_{Row}\left(M_n^c(\varphi, \sigma, p)\right)$.

Property 4.2.24. Let $p_1 + p_2 = p_3$ $(p_2 > p_1,\ 0 \le p_1,\ p_2 \le 2^n - 1)$, $S_3(\varphi) = \{\varphi_i \mid n \ge i \ge \sigma\}$, and $S_4(\varphi) = \{\varphi_i \mid \sigma > i \ge 1\}$. $M_n^a(\varphi, \sigma, p_3)$ can be derived from $M_n^a(\varphi, \sigma, p_1)$ by simply multiplying $M_n^a(\varphi, \sigma, p_1)$ with $P_n^{p_2}$ and doing some simple modification if the following cases occur.

- Case 1: If $p_2 = 2^{n-1}$ then

$$M_n^a(\varphi, \sigma, p_3) = P_n^{p_2} M_n^a(\varphi, \sigma, p_1) \text{ if } (n-1) \in S_4(\varphi) \tag{4.44}$$

$$M_n^a(\varphi, \sigma, p_3) = M_n^a(\varphi, \sigma, p_1) P_n^{p_2} \text{ if } (n-1) \in S_3(\varphi). \tag{4.45}$$

- Case 2: If $L_B(<p_1 + 1>_2) = L_B(<p_3 + 1>_2)$ and $p_2 = 2^j$ ($j \in \{1, 2, \ldots, n-2\}$) then

$$M_n^a(\varphi,\sigma,p_3) = P_n^{p_2} M_n^a(\varphi,\sigma,p_1) \text{ if } S_5 \subset S_4(\varphi) \quad (4.46)$$

$$M_n^a(\varphi,\sigma,p_3) = M_n^a(\varphi,\sigma,p_1) P_n^{p_2} \text{ if } S_5 \subset S_3(\varphi) \quad (4.47)$$

$M_n^a(\varphi,\sigma,p_3)$ = output of the procedure in Fig. 4.10, if $S_5 \not\subset S_4(\varphi)$ and $S_5 \not\subset S_3(\varphi)$ (4.48)

where $S_5 = \{j, j+1, \ldots, n-1\}$ and $M = P_n^{p_2} M_n^a(\varphi,\sigma,p_1)$ is the input of the procedure in Fig. 4.10.

In order to minimize the computational cost of generating all $M_n^a(\varphi,\sigma,p)$ matrices, a fastest LI transform matrix $M_n^a(\varphi_1,\sigma_1,p_4)$ ($p_4 \neq 2^j-1$, $j \in \{0, 1,\ldots, n-1\}$) should be generated from $M_n^a(\varphi_1,\sigma_1,p_4 - 2^{M_B(p_4)-1})$, where $M_B(p_4)$ is the subscript value of the leftmost bit in the binary representation of p_4 whose value is equal to 1.

Compute <Col>$_2$, the n-bit binary representation of Col;
Set S_2 = {<Col>$_2$};
For loop_var = 1 to $\sigma - 1$
  {For every element of S_2, s_i
    {If $L_B(s_i + 1) = \varphi_{loop\_var} + 1$
      Add $(s_i + 1)$ to S_2;}
  Invert bit φ_{loop_var} of all elements of S_2;}
For every member of S_p, s_p
  Invert bit s_p of all elements of S_2;
For loop_var = σ to n
  {For every element of S_2, s_i
    {If $L_B(s_i + 1) = \varphi_{loop\_var} + 1$
      Add $(s_i + 1)$ to S_2;}
  Invert bit φ_{loop_var} of all elements of S_2;}

Figure 4.9: Procedure to determine $Q^{Col}(M_n^c(\varphi,\sigma,p))$.

$S_6 = \{ S_3(\varphi) \cap S_5\};$

For every element of S_6, s

$\{r = 2^{s+1};$

For $l = 0$ *to* $((2^n/r) - 1)$

$\{row1 = ((l + 1) \cdot r) - 1);$

$row2 = row1 - p_2;$

$S_7 = \{t | M_{row2,t} = 1\};$

For every element of S_7, t

$\{If\ (t \geq (l \cdot r))\ and\ (t < ((l \cdot r) + p_2 - 1))$

$\{M_{row2,t} = 0;$

$M_{row1,(t+p2)} = 1;\}\}\}$

$M_n^a(\varphi, \sigma, p_3) = M;$

Figure 4.10: Procedure to obtain $M_n^a(\varphi, \sigma, p_3)$ from $P_n^{p_2} M_n^a(\varphi, \sigma, p_1)$.

Property 4.2.25. Let p_5 be an even number that is smaller than 2^n, $v = L_B(<p_5>_2)$, and $V = \{v - 1, v, v + 1, \ldots, n - 1\}$. Due to Properties 4.2.14 and 4.2.24,

$$M_n^a(\varphi_1, \sigma_1, p_5) = M_n^a(\varphi_2, \sigma_2, p_5) \tag{4.49}$$

when $V \subset S_3(\varphi_1)$ and $V \subset S_3(\varphi_2)$ or $V \cap S_3(\varphi_1) = \varnothing$ and $V \cap S_3(\varphi_2) = \varnothing$, where the definition of $S_3(\varphi)$ follows Property 4.2.24.

Property 4.2.26. Let p_1, p_2, and p_3 be related as in Property 4.2.24. Then if $p_2 = 2^j$ $(0 \leq j \leq n - 1)$

$$M_n^c(\varphi, \sigma, p_3) = P_n^{p_2} M_n^c(\varphi, \sigma, p_1) \text{ if } S_5 \subset S_4(\varphi) \tag{4.50}$$

$$M_n^c(\varphi, \sigma, p_3) = M_n^c(\varphi, \sigma, p_1) P_n^{p_2} \text{ if } S_5 \subset S_3(\varphi) \tag{4.51}$$

$$M_n^d(\varphi, \sigma, p_3) = P_n^{p_2} M_n^d(\varphi, \sigma, p_1) \text{ if } S_5 \subset S_4(\varphi) \tag{4.52}$$

$$M_n^d(\varphi, \sigma, p_3) = M_n^d(\varphi, \sigma, p_1)P_n^{p_2} \text{ if } S_5 \subset S_3(\varphi). \quad (4.53)$$

Note that the $S_3(\varphi)$, $S_4(\varphi)$, and S_5 in (4.50)–(4.53) are the same as those in Property 4.24.

4.2.3 Experimental results for fastest LI transforms over GF(2)

The calculation of spectral coefficients for the existing fastest LI transforms has been implemented in C programs and run on a Pentium III 800 MHz computer with 192 MB RAM for several standard binary benchmark functions. The total numbers of nonzero spectral coefficients for the existing fastest LI transforms are listed in Table 4.1. Due to Property 4.2.5, which states that the coefficient column vector of M_n^a (M_n^c) for a particular binary function can be obtained by reversing the coefficient column vector of M_n^d (M_n^b) for the same binary function, the number of nonzero coefficients of M_n^a and M_n^d are always the same and likewise for M_n^c and M_n^b. Hence, in the Table 4.1 the number of nonzero spectral coefficients for existing fastest LI transforms are combined into two columns.

In addition, the numbers of nonzero spectral coefficients in the fastest LI expansion based on $M_n^\theta(\varphi, \sigma, p)$ with the smallest number of nonzero terms when p is set to zero are also shown in Table 4.1 for the same benchmark functions. Note that due to Property 4.2.15, for $\theta \in \{a, b\}$ the number of nonzero spectral coefficients for $M_n^\theta(\varphi, \sigma, 0)$ is always the same with the corresponding number for M_n^θ regardless of the ordering. Therefore, in the table only the numbers for optimal $M_n^c(\varphi, \sigma, 0)$ and $M_n^d(\varphi, \sigma, 0)$ are given. It can be concluded from the table that reordering the butterfly diagram stages of the existing fastest LI transforms may result in smaller number of nonzero spectral coefficients.

For comparison purpose, the computation of FPRME transforms has also been implemented in C programs and run on the same computer. Execution time (in

milliseconds) for the spectral coefficients calculation of the existing fastest LI transforms as well as polarity zero and all 2^n polarities of FPRME using fast transform are given in Table 4.2. Here the time for obtaining polarity zero FPRME spectral coefficients is taken to represent the time to obtain any polarity FPRME spectral coefficient as they will not differ much. Therefore, from the entries in Table 4.2, it can be concluded that the time required for generating the spectral coefficients of an existing fastest LI transform is generally either the same or shorter than the time required for generating an FPRME spectral coefficient in any polarity. This result is expected since fastest LI transforms require smaller number of additions over GF(2) compared to FPRME and the higher the number of addition operations to be performed the longer is the computation time. It should be noted that to be able to obtain the best polarity for the FPRME transform, all 2^n FPRME spectra have to be calculated whereas there are only four coefficient column vectors that need to be generated to obtain the best representation based on the existing fastest LI transforms.

Table 4.1: Number of nonzero spectral coefficients for several benchmark functions.

Input files	Existing fastest LI transforms		Optimal $M_n^{\theta}(\varphi,\sigma,0)$	
	M_n^a and M_n^d	M_n^b and M_n^c	$M_n^c(\varphi,\sigma,0)$	$M_n^d(\varphi,\sigma,0)$
xor5	19	20	16	16
squar5	30	31	30	29
rd53	28	30	28	26
bw	26	22	22	26
con1	96	91	72	73
5xp1	128	128	128	127
z5xp1	128	128	127	128
rd73	118	121	113	110
inc	88	85	74	74
rd84	240	246	227	224
misex1	99	99	76	76
ex5	240	223	207	207

Table 4.2: Execution time to calculate the spectral coefficients of polarity zero FPRME, all polarities FPRME, and existing fastest LI transforms for several binary benchmark functions (in ms).

Input files	Execution time for polarity zero FPRME	Execution time for M_n^a	Execution time for M_n^b	Execution time for M_n^c	Execution time for M_n^d	Execution time for all polarities FPRME
xor5	10	10	10	10	10	381
squar5	40	10	10	10	10	1001
rd53	30	10	10	10	10	420
bw	150	10	10	10	10	3365
con1	20	10	10	10	10	1272
5xp1	80	20	20	20	20	5178
z5xp1	80	20	10	20	20	5018
rd73	30	10	10	10	10	1743
inc	70	10	10	20	10	4627
rd84	40	20	10	20	10	4657
misex1	70	30	20	20	20	7972
ex5	411	170	170	180	170	70171
9sym	30	10	10	10	10	3475
z9sym	30	10	10	10	10	3445
clip	90	50	50	50	50	14400
apex4	210	110	110	90	100	46998
sao2	120	60	60	50	50	36953
ex1010	220	110	100	90	100	104450

The generation of the coefficient column vectors for the fastest LI transforms with reordering and permutation has also been implemented in C and run on a Pentium 4 2.8 GHz, 512 MB RAM computer for the same binary benchmark functions. For each benchmark function, the coefficient column vectors for all possible $M_n^{\theta}(\varphi, \sigma, p)$ transform matrices are generated and the numbers of their nonzero spectral coefficients are computed. The smallest numbers of nonzero spectral coefficients for each type of the fastest LI transforms for those functions are presented in Table 4.3.

Table 4.3: Smallest nonzero spectral coefficient numbers of $M_n^{\theta}(\varphi, \sigma, p)$.

Input files	Smallest number of nonzero coefficients → One example of fastest LI transform matrices that give such number of nonzero coefficients			
	$M_n^a(\varphi, \sigma, p)$	$M_n^b(\varphi, \sigma, p)$	$M_n^c(\varphi, \sigma, p)$	$M_n^d(\varphi, \sigma, p)$
xor5	14 → M_5^a(21430,4,15)	14 → M_5^b(21403,4,7)	13 → M_5^c(41230,1,3)	13 → M_5^d(24130,1,11)
squar5	28 → M_5^a(43120,3,3)	29 → M_5^b(03214,5,11)	28 → M_5^c(31240,6,22)	27 → M_5^d(10243,5,31)
rd53	25 → M_5^a(43012,3,10)	27 → M_5^b(43012,3,2)	24 → M_5^c(13402,6,9)	24 → M_5^d(10423,3,5)
bw	22 → M_5^a(43210,6,21)	22 → M_5^b(43210,6,30)	22 → M_5^c(03142,5,17)	22 → M_5^d(43201,6,23)
con1	62 → M_7^a(1543260,7,113)	67 → M_7^b(5306214,5,95)	63 → M_7^c(1023456,7,9)	58 → M_7^d(1034562,7,81)
5xp1	121 → M_7^a(6543120,3,5)	127 → M_7^b(6541032,3,73)	126 → M_7^c(4201563,4,120)	119 → M_7^d(6145023,6,1)
z5xp1	120 → M_7^a(1654320,3,53)	120 → M_7^b(6543120,3,125)	119 → M_7^c(6102543,6,81)	111→ M_7^d(5461023,4,1)
rd73	105 → M_7^a(3654012,4,2)	109 → M_7^b(4365210,6,2)	102 → M_7^c(2135460,4,73)	101 → M_7^d(3652410,5,10)
inc	74 → M_7^a(0543216,7,21)	75 → M_7^b(6540213,4,127)	72 → M_7^c(2063451,7,67)	72 → M_7^d(2013546,7,23)

From the numbers in Tables 4.1 and 4.3, it can be seen that generally at least one fastest LI transform with reordering and permutation can be found that provides a particular function with simpler representation than the representations based on the existing fastest LI transforms in terms of the number of nonzero spectral coefficients. These simpler representations with smaller number of nonzero spectral coefficients require less number of two-input XOR gates when directly implemented in hardware and they are easier to be minimized if it is necessary. Hence, the new fastest LI transforms are useful for minimizing hardware implementation cost of a binary function or a system of binary functions.

4.3 Fastest LIA transforms

New generalized fastest LIA transforms that have the same computational complexity as the existing fastest LIA transforms are introduced in this section. Fastest LIA transforms are the integer counterpart of fastest LI transforms over GF(2) where the spectral coefficients are not restricted to only 0 and 1. General definitions, properties, and experimental results for both the existing and new fastest LIA transforms are presented. The experimental results show that for some binary functions, fastest LIA transforms provide representations with smaller number of nonzero spectral coefficients and therefore allow the functions' values to be calculated more quickly for different assignment of input variables.

4.3.1 Basic definitions for fastest LIA transforms

Definition 4.3.1. Let T_n denote an LIA transform of size $2^n \times 2^n$ and $\vec{F} = [f_0, f_1, ..., f_{2^n-1}]^T$ be a column vector defining the truth vector of an n-variable binary function $f(\vec{x})$ in natural binary ordering. Then the coefficient column vector (spectrum) $\vec{C}$ of T_n for $f(\vec{x})$ can be obtained by

$$\vec{C} = T_n \vec{F}, \tag{4.54}$$

where $\vec{x} = [x_n, x_{n-1}, ..., x_1]$, $\vec{C} = [c_0, c_1, ..., c_{2^n-1}]^T$ and T represents matrix transpose operator. Conversely,

$$\vec{F} = T_n^{-1}\vec{C}, \tag{4.55}$$

where T_n^{-1} is a linearly independent matrix which is the inverse of T_n performed in standard arithmetic algebra.

Definition 4.3.2. Let T_n be an LIA transform of size $2^n \times 2^n$. Then, the LIA expansion of a binary function $f(\vec{x})$ based on T_n can be expressed as

$$f(\vec{x}) = \sum_{j=0}^{2^n-1} c_j g_j, \tag{4.56}$$

where c_j $(0 \le j \le 2^n - 1)$ is the j-th spectral coefficient of T_n for $f(\vec{x})$ and g_j is the binary function whose truth vector is represented by column j of T_n^{-1}.

The new fastest LIA transforms that are introduced here are called generalized fastest LIA transforms. They are classified into type *a*, *b*, *c*, and *d* depending on the structure of their factorized transform matrices. The general formulae for their factorized transform matrices are given in Definition 4.3.3. In Definition 4.3.4, the factorized representation of the forward generalized fastest LIA transform matrices are presented. Since this definition covers the existing fastest LIA transform matrices defined in [FR02], the terms generalized fastest LIA transform matrices and fastest LIA transform matrices are used interchangeably in this thesis.

Definition 4.3.3. Let $K_{n,j}^{\theta}$ $(0 \le j \le n-1,\ \theta \in \{a,b,c,d\})$ denote a factorized transform matrix for $2^n \times 2^n$ fastest LIA transforms of type θ. Then $K_{n,j}^{\theta}$ is a $2^n \times 2^n$ square matrix that contains at most two nonzero elements in each row and column and follows the following general formulae, where I_n denotes the identity matrix of dimension $2^n \times 2^n$.

$$K_{n,j}^{a} = I_{n-j-1} \otimes \alpha_2 \left(I_{j+1} \right) \tag{4.57}$$

$$K_{n,j}^{b} = R_2{}^2 \left(K_{n,j}^{a} \right) \tag{4.58}$$

$$K_{n,j}^{c} = R_{1,j} \left(K_{n,j}^{b} \right) \tag{4.59}$$

$$K_{n,j}^{d} = R_2{}^2 \left(K_{n,j}^{c} \right) \tag{4.60}$$

Definition 4.3.4. Let $T_n^{\theta}(\varphi, \sigma, p)$ denote a $2^n \times 2^n$ generalized fastest LIA transform matrix. Then $T_n^{\theta}(\varphi, \sigma, p)$ is defined as

$$T_n^{\theta}(\varphi, \sigma, p) = \begin{cases} \left[\prod_{j=n}^{\sigma} K_{n,\varphi_j}^{\theta} \right] \cdot P_n^{p} \cdot \left[\prod_{j=\sigma-1}^{1} K_{n,\varphi_j}^{\theta} \right], & \text{if } \sigma \neq n+1 \\ P_n^{p} \cdot \prod_{j=n}^{1} K_{n,\varphi_j}^{\theta}, & \text{otherwise,} \end{cases} \tag{4.61}$$

where $\theta \in \{a, b, c, d\}$ is the type of the fastest LIA transform matrix, φ is the ordering of the n sparse matrices in the factorized representation of $T_n^{\theta}(\varphi, \sigma, p)$, σ $(1 \leq \sigma \leq n+1)$ is the position of the permutation matrix, and p $(0 \leq p \leq 2^n - 1)$ is the permutation number of the added permutation matrix. The ordering φ is an n-digit string in which every digit takes values from 0 to $n-1$ and no two different digits in it are allowed to have the same values,

$$\varphi = < \varphi_n, \varphi_{n-1}, \ldots, \varphi_1 > , \tag{4.62}$$

where $\varphi_i \in \{0, 1, ..., n-1\}$ and $\varphi_i = \varphi_j$ iff $i = j \, (1 \leq i, j \leq n)$.

Example 4.3.1. Based on Definitions 4.1.9, 4.3.3, and 4.3.4, the generalized fastest LIA transform matrix $T_3^{a}(021,2,1)$ is given by

$$T_3^a(021,2,1) = K_{3,0}^a \cdot K_{3,2}^a \cdot P_3^1 \cdot K_{3,1}^a$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & -1 \end{bmatrix}.$$

Definition 4.3.5. Let $T_n^\theta(\varphi,\sigma,p)$ be a $2^n \times 2^n$ generalized fastest LIA transform matrix with ordering φ. Then the sets φ_{front} and φ_{back} of $T_n^\theta(\varphi,\sigma,p)$ are defined as the subscript j values of $K_{n,j}^\theta$ matrices that are located in the left hand side and the right hand side of the permutation matrix in the factorized representation of $T_n^\theta(\varphi,\sigma,p)$, respectively.

$$\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right) = \{\varphi_i \mid \sigma \le i \le n\} \tag{4.63}$$

$$\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right) = \{\varphi_i \mid 1 \le i \le \sigma - 1\} \tag{4.64}$$

4.3.2 Properties of fastest LIA transforms and their spectra

In this section, some properties for fastest LIA transforms and their spectra are given.

In Properties 4.3.1 and 4.3.2, special cases of generalized fastest LIA transforms and the relationships between them are presented. In Property 4.3.3, the computational cost of fastest LIA transforms is shown. More general relations between different fastest LIA transforms are presented in Properties 4.3.4–4.3.7 that allow some fastest LIA transforms to be obtained from one another without incurring any additions or multiplications. Clearly, these properties are useful for reducing the total computational cost when the best generalized fastest LIA expansion is being searched.

Property 4.3.1. Let φ_a^n be an n-digit string $<\varphi_{a_n},\varphi_{a_{n-1}},\ldots,\varphi_{a1}>$ where $\varphi_{a_j} = j$ $(1 \le j \le n)$. Then, when $p = 0$ and $\varphi = \varphi_a^n$, $T_n^{\theta}(\varphi,\sigma,p)$ matrices are recursive and can be defined as follows:

$$T_n^a(\varphi_a^n,\sigma,0) = \begin{bmatrix} T_{n-1}^a(\varphi_a^{n-1},\sigma,0) & O_{n-1} \\ Z_{n-1} & T_{n-1}^a(\varphi_a^{n-1},\sigma,0) \end{bmatrix} \tag{4.65}$$

$$T_n^b(\varphi_a^n,\sigma,0) = \begin{bmatrix} T_{n-1}^b(\varphi_a^{n-1},\sigma,0) & Z_{n-1} \\ O_{n-1} & T_{n-1}^b(\varphi_a^{n-1},\sigma,0) \end{bmatrix} \tag{4.66}$$

$$T_n^c(\varphi_a^n,\sigma,0) = \begin{bmatrix} Z_{n-1} & T_{n-1}^c(\varphi_a^{n-1},\sigma,0) \\ T_{n-1}^c(\varphi_a^{n-1},\sigma,0) & O_{n-1} \end{bmatrix} \tag{4.67}$$

$$T_n^d(\varphi_a^n,\sigma,0) = \begin{bmatrix} O_{n-1} & T_{n-1}^d(\varphi_a^{n-1},\sigma,0) \\ T_{n-1}^d(\varphi_a^{n-1},\sigma,0) & Z_{n-1} \end{bmatrix}, \tag{4.68}$$

where O_{n-1} is a $2^{n-1} \times 2^{n-1}$ matrix with all its elements zero and Z_{n-1} is a $2^{n-1} \times 2^{n-1}$ matrix with all its elements zero except one element located at one corner of the matrix that is equal to -1. The location of -1 element inside Z_{n-1} is at bottom left corner, top right corner, top left corner, and bottom right corner for (4.65), (4.66), (4.67), and (4.68), respectively, where (4.65) and (4.66) correspond to the recursive definitions of the fastest LIA transform matrices presented in [FR02].

Property 4.3.2. $T_n^a(\varphi_a^n,\sigma,0)$ is simply $T_n^c(\varphi_a^n,\sigma,0)$ that is flipped vertically. Similarly, $T_n^b(\varphi_a^n,\sigma,0)$ can be obtained by vertically flipping $T_n^d(\varphi_a^n,\sigma,0)$.

Furthermore, flipping $T_n^a(\varphi_a^n,\sigma,0)$ horizontally results in $T_n^d(\varphi_a^n,\sigma,0)$. These can be clearly seen from (4.65)–(4.68) and from their corresponding butterfly diagrams for $n = 3$ given in Fig. 4.11. As a result, the coefficient column vector of $T_n^c(\varphi_a^n,\sigma,0)$ ($T_n^d(\varphi_a^n,\sigma,0)$) for a particular n-variable binary function $f(\vec{x})$ is simply the coefficient column vector of $T_n^a(\varphi_a^n,\sigma,0)$ ($T_n^b(\varphi_a^n,\sigma,0)$) for the same function with the order of the coefficients being reversed.

Property 4.3.3. [RF02] The number of subtractions required to compute the coefficient column vector of a generalized fastest LIA transform for an n-variable binary function is $2^n - 1$.

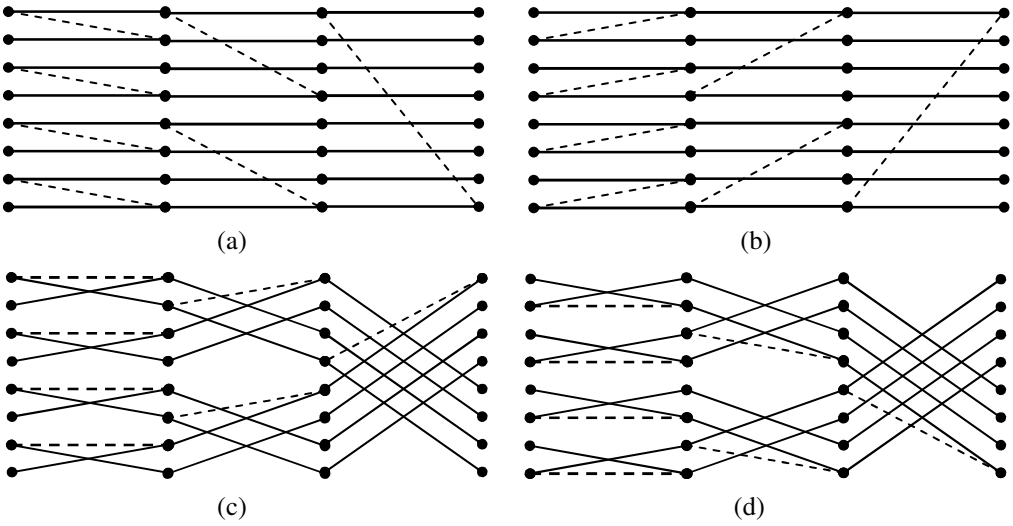


Figure 4.11: Butterfly diagrams of forward fastest LIA transform matrices: (a) $T_3^a(210,1,0)$; (b) $T_3^b(210,1,0)$; (c) $T_3^c(210,1,0)$; (d) $T_3^d(210,1,0)$.

Proof: From Definition 4.3.4, any generalized fastest LIA transform matrix of size $2^n \times 2^n$ can be factorized into product of a permutation matrix and n sparse matrices $K_{n,j}^{\theta}$ ($j = \{0, 1, \ldots, n-1\}$), where each factorized matrix corresponds to one stage in

the flow graph of the LIA transform matrix. From (4.3), the flow graph of a permutation matrix does not involve any arithmetic operations whereas from (4.57)–(4.60) the flow graph of each $K_{n,j}^{\theta}$ incurs 2^{n-j-1} subtractions. Hence, the total computational cost for a generalized fastest LIA transform matrix is

$$\begin{aligned} D_n &= \sum_{j=0}^{n-1} 2^{n-j-1} \\ &= \sum_{j=0}^{n-1} 2^{j} \\ &= 2^n - 1, \end{aligned}$$

all of which are subtractions. As the computational cost of arithmetic transform for an n-variable binary functions is $n \cdot 2^{n-1}$, generalized fastest LIA transform matrix always has smaller computational cost than arithmetic transform when $n > 1$.

Property 4.3.4. The product of the same subset of factorized matrices for $T_n^{\theta}(\theta \in \{a, b\})$ always produces the same results regardless of the order in which they are multiplied. As a result, any two generalized fastest LIA transform matrices $T_n^{\theta_1}(\varphi_1, \sigma_1, P_1)$ and $T_n^{\theta_2}(\varphi_2, \sigma_2, P_2)$ are identical if $\theta_1 = \theta_2 \in \{a, b\}$, $P_1 = P_2$, and $\varphi_{front}\left(T_n^{\theta_1}(\varphi_1, \sigma_1, P_1)\right) = \varphi_{front}\left(T_n^{\theta_2}(\varphi_2, \sigma_2, P_2)\right)$. Furthermore, $T_n^a(\varphi, \sigma, 0) = T_n^a(\varphi_a^n, \sigma, 0)$ and $T_n^b(\varphi, \sigma, 0) = T_n^b(\varphi_a^n, \sigma, 0)$ for any values of φ and σ, where φ_a^n and $\varphi_{front}\left(T_n^{\theta}(\varphi, \sigma, p)\right)$ were described in Property 4.3.1 and Definition 4.3.5, respectively.

Property 4.3.5. Let $T_n^a(\varphi, \sigma, p)$, $T_n^b(\varphi, \sigma, p)$, $T_n^c(\varphi, \sigma, p)$, and $T_n^d(\varphi, \sigma, p)$ be four generalized fastest LIA transform matrices of different type with the same values of φ, σ, and p. Then,

$$T_n^a(\varphi, \sigma, p) = {R_2}^2\left(T_n^b(\varphi, \sigma, p)\right) \tag{4.69}$$

and

$$T_n^c(\varphi, \sigma, p) = {R_2}^2\left(T_n^d(\varphi, \sigma, p)\right). \tag{4.70}$$

Property 4.3.6. Let $T_n^{\theta}(\varphi,\sigma,p)$ be any $2^n \times 2^n$ generalized fastest LIA transform matrix. Then, the inverse of $T_n^{\theta}(\varphi,\sigma,p)$ can be obtained by simply applying operation $R_2{}^2$ on $\left(T_n^{\theta}(\varphi,\sigma,p)\right)^T$ and replacing all elements of the resulting matrix by their absolute values,

$$\left(T_n^{\theta}(\varphi,\sigma,p)\right)^{-1} = \left| R_2{}^2\left((T_n^{\theta}(\varphi,\sigma,p))^T\right)\right|. \tag{4.71}$$

Example 4.3.2. In Example 4.3.1, it has been obtained that

$$T_3^a(021,2,1) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & -1 \end{bmatrix}.$$

Using Property 4.3.5, the matrix $T_3^b(021,2,1)$ can be calculated by

$$T_3^b(021,2,1) = R_2{}^2\left(T_3^a(021,2,1)\right)$$
$$= \begin{bmatrix} -1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Furthermore, by Property 4.3.6,

$$\left(T_3^b(021,2,1)\right)^{-1} = \left| R_2{}^2\left(T_3^b(021,2,1)^T\right)\right|$$
$$= \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Property 4.3.7. Let $T_n^{\theta}(\varphi,\sigma,P_1)$ and $T_n^{\theta}(\varphi,\sigma,P_2)$ be two $2^n \times 2^n$ generalized fastest LIA transform matrices of the same type, ordering, and permutation positions but different permutation numbers P_1 and P_2, respectively ($0 \le P_1, P_2 \le 2^n - 1, P_2 > P_1$). If $P_3 = (P_2 - P_1) > P_1$ and $P_3 = 2^j$ ($0 \le j \le n-1$), then $T_n^{\theta}(\varphi,\sigma,P_1)$ and $T_n^{\theta}(\varphi,\sigma,P_2)$ are related as follows

$$T_n^{\theta}(\varphi,\sigma,P_2) = P_n^{P_3} T_n^{\theta}(\varphi,\sigma,P_1) \quad \text{if } \{j, j+1, \ldots, n-1\} \subseteq \varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,P_1)\right) \tag{4.72}$$

$$T_n^{\theta}(\varphi,\sigma,P_2) = T_n^{\theta}(\varphi,\sigma,P_1) P_n^{P_3} \quad \text{if } \{j, j+1, \ldots, n-1\} \subseteq \varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,P_1)\right) \tag{4.73}$$

$$T_n^{\theta}(\varphi,\sigma,P_2) = M \quad \text{if } \{j, j+1, \ldots, n-1\} \not\subset \varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,P_1)\right) \text{ and}$$

$\{j, j+1, \ldots, n-1\} \not\subset \varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,P_1)\right)$ and $L_B(<P_1 + 1>_2) = L_B(<P_2 + 1>_2)$ and

$\theta \in \{a,b\}$, (4.74)

where M is the result of the procedure given in Fig. 4.12 and $P_n^{P_3}$ follows Definition 4.1.9.

We are interested in analyzing several arithmetic properties of the generalized fastest LIA transforms spectra that may be useful for testing and fault detection applications, such as various bounds on the spectra. In order to derive these various bounds, properties on the numbers and locations of nonzero elements inside the transforms must first be known. Hence, in what follows properties related to the generalized fastest LIA transform matrix elements are first presented. Various properties on the spectra of generalized fastest LIA transform matrices $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$ are then listed. Note that similar properties for the spectra of $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$ can be derived in a similar way.

Property 4.3.8. The locations of nonzero elements in $T_n^a(\varphi,\sigma,p)$ and $T_n^c(\varphi,\sigma,p)$ can be determined by the procedures shown in Figs. 4.13 and 4.14. Given an integer number *index* ($0 \le index \le 2^n - 1$) and θ, φ, p, and σ of a generalized fastest LIA transform matrix, *Procedure1* calculates the column (row) index numbers of the nonzero elements in row (column) *index* of $T_n^a(\varphi,\sigma,p)$ ($T_n^c(\varphi,\sigma,p)$). Conversely,

Procedure3 calculates the row (column) index numbers of the nonzero elements in column (row) *index* of $T_n^a(\varphi,\sigma,p)$ ($T_n^c(\varphi,\sigma,p)$). In both procedures, the sets E_3 and E_4 contain n-bit binary strings and other than in the initialization their numbers of elements are modified by sub procedures *Procedure2* and *Procedure4* inside *Procedure1* and *Procedure3*, respectively. At the end of the procedures the elements in the set E_3 are the binary representations of the appropriate indexes of 1s whereas the elements in the set E_4 are the binary representations of the appropriate indexes of −1s. Note that due to Property 4.3.5, the nonzero elements locations in $T_n^b(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$ can be easily determined once the locations of nonzero elements in $T_n^a(\varphi,\sigma,p)$ and $T_n^c(\varphi,\sigma,p)$ are known.

$M = P_n^{P_3} T_n^{\theta}(\varphi,\sigma,P_1)$;
$E_1 = \{j, j+1, \ldots, n-1\} \cap \varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,P_1)\right)$;
for every element e of E_1
  {$k = 2^{e+1}$;
  for ($l = 0$ *to* $((2^n/k) - 1)$)
    {*if* ($\theta = a$)
      {$row1 = ((l+1)\cdot k) - 1$;
       $row2 = row1 - P_3$;}
     else
      {$row1 = (l \cdot k)$;
       $row2 = row1 + P_3$;}
     $E_2 = \{i | M_{row2,i} \neq 0\}$;
     for every element i of E_2
      {*if* ($\theta = a$)
        {*if* (($i \geq (l \cdot k)$) *and* ($i < ((l \cdot k) + P_3 - 1)$))
          {$M_{row1,(i+P_3)} = M_{row2,i}$;
           $M_{row2,i} = 0$;}}
       else
        {*if* (($i \geq (((l+1) \cdot k) - P_3)$) *and* ($i < ((l+1) \cdot k)$))
          {$M_{row1,(i-P_3)} = M_{row2,i}$;
           $M_{row2,i} = 0$;}}}}}

Figure 4.12: Procedure to obtain $T_n^{\theta}(\varphi,\sigma,P_2)$.

```
Procedure1(index, θ, φ, p, σ)
{if (θ = c)
   {for (loop_var = 1 to n)
      φ_loop_var = φ_n+1−loop_var ;
    σ = n − σ + 2;}
 set E3 ={<index>2} and E4 = {};
 for (loop_var = n to σ)
   Procedure2(loop_var);
 for every element e in E3 and E4
   replace e with (e ⊕ <p>2), where ⊕ denotes bitwise XOR operation;
 for (loop_var = σ − 1 to 1)
   Procedure2(loop_var);}

Procedure2(loop_var)
{ set E5 = { } and E6 = { };
 for every element e3 of E3
   { if (θ = c)
      invert bit (φ_loop_var + 1) of e3;
    if (LB(e3 + 1) ≥ φ_loop_var + 2)
      {e5 = e3 with bits 1 to (φ_loop_var + 1) inverted;
       add e5 to E5;}}
 for every element e4 of E4
   { if (θ = c)
      invert bit (φ_loop_var + 1) of e4;
    if (LB(e4 + 1) ≥ φ_loop_var + 2)
      { e6 = e4 with bits 1 to (φ_loop_var + 1) inverted;
       add e6 to E6;}}
 add all elements of E5 to E4;
 add all elements of E6 to E3;}
```

Figure 4.13: *Procedure1(index, θ, φ, p, σ)* and *Procedure2(loop_var)*.

```
Procedure3(index, θ, φ, p, σ)
{ if (θ = c)
   { for (loop_var = 1 to n)
       φ_loop_var = φ_n+1−loop_var ;
       σ = n − σ + 2;}
  set E3 ={<index>2} and E4 = {};
  for (loop_var = 1 to σ − 1)
    Procedure4(loop_var);
  for every element e in E3 and E4
    replace e with (e ⊕ <p>2), where ⊕ denotes bitwise XOR operation;
  for (loop_var = σ to n)
    Procedure4(loop_var);}

Procedure4(loop_var)
{ set E5 = { } and E6 = { };
  for every element e3 of E3
    { if (LB(e3 ) ≥ φ_loop_var + 2)
        { e5 = e3 with bits 1 to (φ_loop_var + 1) inverted;
          add e5 to E5;}}
  for every element e4 of E4
    { if (LB(e4 ) ≥ φ_loop_var + 2)
        { e6 = e4 with bits 1 to (φ_loop_var + 1) inverted;
          add e6 to E6;}}
  add all elements of E5 to E4;
  add all elements of E6 to E3;}
  if (θ = c)
    invert bit (φ_loop_var + 1) of all elements of E3 and E4;}
```

Figure 4.14: *Procedure3(index, θ, φ, p, σ)* and *Procedure4(loop_var)*.

Example 4.3.3. Let us find the column index numbers of the nonzero elements in row seven of $T_3^a(021,2,1)$. By Property 4.3.8, the calculation can be performed by *Procedure1*. The states of the sets E_3 and E_4 inside the procedure are as follows:

- Initialization: $E_3 = \{111\}$; $E_4 = \{\}$
- *loop_var* = 3 -> After *Procedure2(loop_var)* : $E_3 = \{111\}$; $E_4 = \{110\}$
- *loop_var* = 2 -> After *Procedure2(loop_var)* : $E_3 = \{111\}$; $E_4 = \{110, 000\}$
- After replacing e with $(e \oplus <p>_2)$: $E_3 = \{110\}$; $E_4 = \{111, 001\}$
- *loop_var* = 1 -> After *Procedure2(loop_var)* : $E_3 = \{110, 100\}$; $E_4 = \{111, 001\}$

Hence, it has been obtained that there are four nonzero elements inside row seven of $T_3^a(021,2,1)$, which consists of two 1s at columns four and six and two −1s at columns one and seven.

Property 4.3.9. Based on Properties 4.3.5 and 4.3.8, the total number of nonzero elements in $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$ are equal to $\left(2^{n+1}-1\right)+h$, where

$$h=\sum_{i=n}^{\sigma}\sum_{j=\sigma-1}^{1}\begin{cases}2^{n-(\max(\varphi_i,\varphi_j)+1)}, & \text{if } L_B(<p+1>_2)\geq(\min(\varphi_i,\varphi_j)+2)\\ 0, & \text{otherwise.}\end{cases} \tag{4.75}$$

The nonzero elements consist of (2^n-1) −1s and (2^n+h) 1s.

Property 4.3.10. The number of nonzero elements in $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$ can also be derived based on Properties 4.3.5 and 4.3.8. In order to simplify the analysis, the matrices are divided into several groups based on the values of σ, p, and z, where z is the subscript value of the ordering digit whose value is equal to zero, i.e., $\varphi_z = 0$. Let X_1, X_2, X_3, X_4, X_5, and X_6 be the binary strings given by (4.76)–(4.81), where $\oplus$ and $\&$ denote bitwise operators XOR and AND. Then the number of 1s and −1s for each group are as given in Tables 4.4 and 4.5, respectively. It can be seen that the number of −1s in $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$ are always equal to 2^n-1 when p is an even number. On the other hand, the number of 1s, and hence the total number of nonzero elements, changes with z and p.

$$X_1(i,j) = < x_n, x_{n-1}, \cdots, x_1 > \text{ where } x_k = 1 \text{ if } k \in \{\varphi_l + 1 \mid i \le l \le j\} \text{ and } x_k = 0, \text{otherwise} \tag{4.76}$$

$$X_2(i,j) = (< 2^{\varphi_i} - 1 >_2 \oplus X_1(j+1, i-1)) \& < 2^{\min(\varphi_i, \varphi_j)+1} - 1 >_2 \tag{4.77}$$

$$X_3(i,j) = (< 2^{\varphi_i} - 1 >_2 \oplus X_1(j+1, i-1) \oplus < p >_2) \& < 2^{\min(\varphi_i, \varphi_j)+1} - 1 >_2 \tag{4.78}$$

$$X_4(i,j) = (< 2^{\varphi_i} - 1 >_2 \oplus X_1(j+1, i-1) \oplus < p >_2 \oplus < 1 >_2) \& < 2^{\min(\varphi_i, \varphi_j)+1} - 1 >_2 \tag{4.79}$$

$$X_5(i,j,k) = \begin{cases} (< 2^{\varphi_j} - 1 >_2 \oplus < p >_2 \oplus X_1(k+1, j-1)) \& < 2^{\min(\varphi_j, \varphi_k)+1} - 1 >_2, & \text{if } \varphi_i < \varphi_j \\ (< 2^{\varphi_i} - 2^{\varphi_j} >_2 \oplus < p >_2 \oplus X_1(k+1, i-1)) \& < 2^{\min(\varphi_i, \varphi_k)+1} - 1 >_2, & \text{otherwise} \end{cases} \tag{4.80}$$

$$X_6(i,j,k) = \begin{cases} (< 2^{\varphi_j} - 1 >_2 \oplus X_1(k+1, j-1)) \& < 2^{\min(\varphi_j, \varphi_k)+1} - 1 >_2, & \text{if } \varphi_i < \varphi_j \\ (< 2^{\varphi_i} - 2^{\varphi_j} >_2 \oplus < p >_2 \oplus X_1(j+1, i-1) \oplus X_1(k+1, j-1)) \& < 2^{\min(\varphi_i, \varphi_k)+1} - 1 >_2, & \text{otherwise.} \end{cases} \tag{4.81}$$

The upper bounds presented in Properties 4.3.11–4.3.17 below are derived based on the distribution of the nonzero elements inside $T_n^a(\varphi, \sigma, p)$ and $T_n^b(\varphi, \sigma, p)$ matrices. Since by (4.69) the nonzero elements distribution in the rows and columns of $T_n^b(\varphi, \sigma, p)$ are the same as that of $T_n^a(\varphi, \sigma, p)$, the proofs for the properties are given only for $T_n^a(\varphi, \sigma, p)$ but the resulting bounds are also true for $T_n^b(\varphi, \sigma, p)$.

Property 4.3.11. The absolute value of any spectral coefficient of $T_n^a(\varphi, \sigma, p)$ or $T_n^b(\varphi, \sigma, p)$ is always less than or equal to $\frac{1}{4}(n^2 + 4)$ for R-coding and less than or equal to $\frac{1}{4}(n^2 + 2n + 5)$ for S-coding.

Proof: By Property 4.3.8, when p is even the odd rows of $T_n^a(\varphi, \sigma, p)$ have one 1 and between zero to $n - 1$s whereas the even rows of $T_n^a(\varphi, \sigma, p)$ have only one 1.

When p is odd, the even rows of $T_n^a(\varphi,\sigma,p)$ contain one 1 and between zero to $(\sigma-1)$ -1s whereas the odd rows have between one and $((n+1-\sigma)(\sigma-1)+1)$ 1s and between zero to $(n+1-\sigma)$ -1s. By Property 4.3.3, $c_i = \sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j$, where $T_{i,j}$ denotes the element that is located at row i and column j of $T_n^a(\varphi,\sigma,p)$. Thus, for any i, $|c_i| \le$ max (no.of 1s, no.of -1s) in row i for R-coding and $|c_i| \le$ number of nonzero elements in row i for S-coding.

Let us first consider R-coding. According to the above, the maximum absolute value of spectral coefficients for $T_n^a(\varphi,\sigma,p)$ = the largest value of max (no.of 1s, no. of -1s) in any row of $T_n^a(\varphi,\sigma,p)$ = max (n, maximum value of $((n+1-\sigma)(\sigma-1)+1)$). Let $Y = ((n+1-\sigma)(\sigma-1)+1)$. Then, the peak value of Y occurs when $\frac{dY}{d\sigma} = -2\sigma+n+2 = 0 \rightarrow$ the peak of Y occurs at $\sigma = (n+2)/2$ with value of $\frac{1}{4}(n^2+4)$. As $\frac{d^2Y}{d\sigma} = -2 < 0$, the peak value is the maximum value of Y. Since max (n, $\frac{1}{4}(n^2+4)$) = $\frac{1}{4}(n^2+4)$, it is obtained that for R-coding the maximum absolute value of C_i for $T_n^a(\varphi,\sigma,p)$ = $\frac{1}{4}(n^2+4)$.

Next, let us consider S-coding. For S-coding, the maximum absolute value of spectral coefficients for $T_n^a(\varphi,\sigma,p)$ = maximum number of nonzero elements in any row of $T_n^a(\varphi,\sigma,p)$ = max $(((n+1-\sigma)(\sigma-1)+1) + (n+1-\sigma))$. Let $Y =$ $((n+1-\sigma)(\sigma-1)+1) + (n+1-\sigma)$. Then similarly, it can be derived that the maximum value of Y, and hence the maximum absolute value of c_i, is equal to $\frac{1}{4}(n^2+2n+5)$ for $\sigma = (n+1)/2$.

Table 4.4: Number of 1s for $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$.

Cases	No. of 1s	
	p = even	p = odd
$\sigma = 1$; $\sigma = n+1$	$2^n + \sum_{i=1}^{z-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{n}\sum_{j=1}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_2(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$	
$2 \le \sigma \le n$ and $z = n$	$3\cdot 2^{n-1} - 1$	$2^{n-1} + \sum_{i=\sigma}^{n} 2^{n-\varphi_i-1} + \sum_{i=\sigma}^{n-1}\sum_{j=1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$
$2 \le \sigma \le n$ and $z = 1$	2^n	$2^n + \sum_{i=\sigma}^{n}\sum_{j=1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$
$2 \le \sigma \le n$ and $\sigma < z < n$	$2^n + \sum_{i=1}^{z-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{n}\sum_{j=\sigma}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_2(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases} + \sum_{i=z+1}^{n}\sum_{j=1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$	$2^n + \sum_{i=\sigma}^{z-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{n}\sum_{j=\sigma}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_2(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases} + \sum_{i=\sigma}^{z-1}\sum_{j=1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$
$2 \le \sigma \le n$ and $z = \sigma$	$2^n + \sum_{i=1}^{\sigma-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{n}\sum_{j=1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$	2^n
$2 \le \sigma \le n$ and $z = \sigma - 1$	$2^n + \sum_{i=i}^{z-1} 2^{n-\varphi_i-1} + \sum_{i=\sigma}^{n}\sum_{j=1}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$	$3\cdot 2^{n-1} - 1$
$2 \le \sigma \le n$ and $1 < z < \sigma - 1$	$2^n + \sum_{i=1}^{z-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{\sigma-1}\sum_{j=1}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_2(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases} + \sum_{i=\sigma}^{n}\sum_{j=1}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$	$3\cdot 2^{n-1} - 1 - \sum_{i=z+1}^{\sigma-1} 2^{n-\varphi_i-1} + \sum_{i=z+1}^{\sigma-1}\sum_{j=1}^{z-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_2(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases} + \sum_{i=\sigma}^{n}\sum_{j=z+1}^{\sigma-1} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j) =<0>_2 \\ 0, & \text{otherwise} \end{cases}$

Table 4.5: Number of −1s for $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$.

Cases	No. of −1s	
	p = even	p = odd
$\sigma = 1$; $\sigma = n+1$	2^n-1	2^n-1
$2 \le \sigma \le n$ and $z = n$	2^n-1	$2^n-1+\sum_{i=\sigma}^{n-1}\sum_{j=1}^{\sigma-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j)=<0>_2\\ 0, & \text{otherwise}\end{cases}$
$2 \le \sigma \le n$ and $z = 1$	2^n-1	2^n-1
$2 \le \sigma \le n$ and $\sigma < z < n$	2^n-1	$2^n-1+\sum_{i=\sigma}^{z-1}\sum_{j=1}^{\sigma-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_3(i,j)=<0>_2\\ 0, & \text{otherwise}\end{cases}+\sum_{i=z+1}^{n}\sum_{j=\sigma}^{z-1}\sum_{k=1}^{\sigma-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j,\varphi_k)-1}, & \text{if } X_2(i,j)=<0>_2 \text{ and } X_5(i,j,k)=<0>_2\\ 0, & \text{otherwise}\end{cases}$
$2 \le \sigma \le n$ and $z = \sigma$	2^n-1	2^n-1
$2 \le \sigma \le n$ and $z = \sigma-1$	2^n-1	$2^n-1+\sum_{i=\sigma}^{n}\sum_{j=1}^{z-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_4(i,j)=<0>_2\\ 0, & \text{otherwise}\end{cases}$
$2 \le \sigma \le n$ and $1 < z < \sigma-1$	2^n-1	$2^n-1+\sum_{i=\sigma}^{n}\sum_{j=1}^{z-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j)-1}, & \text{if } X_4(i,j)=<0>_2\\ 0, & \text{otherwise}\end{cases}+\sum_{i=\sigma}^{n}\sum_{j=z+1}^{\sigma-1}\sum_{k=1}^{z-1}\begin{cases}2^{n-\max(\varphi_i,\varphi_j,\varphi_k)-1}, & \text{if } X_3(i,j)=<0>_2 \text{ and } X_6(i,j,k)=<0>_2\\ 0, & \text{otherwise}\end{cases}$

Property 4.3.12. Let $S_1\left(T_n^\theta(\varphi,\sigma,p)\right)$ be the sum of the LIA matrix $T_n^\theta(\varphi,\sigma,p)$ spectral coefficients for a particular n-variable binary function $f\left(\vec{x}\right)$. If $\vec{F}$ is the truth vector of $f\left(\vec{x}\right)$ then there exist a subset of $\vec{F}$, F_1 whose values do not affect the value

of $S_1\left(T_n^{\theta}(\varphi,\sigma,p)\right)$. For $\theta\in\{a,b\}$ the number of elements inside F_1, denoted by $\left|F_1\right|$, is

$$\left|F_1\right|=\begin{cases}1, & \text{if } n=1 \text{ and } p \bmod 4=0\\ 2^{n-2}, & \text{if } n>1 \text{ and } p \bmod 4=0 \text{ or } \sigma=1 \text{ or } \sigma=n+1\\ 2^{n-2}-2^{n-u_1-1}, & \text{if } 1\in\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right) \text{and } p \bmod 4=2\\ 2^{n-2}-2^{n-u_2-1}, & \text{if } 1\in\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right) \text{and } p \bmod 4=2\\ 2^{n-1}-2^{n-u_1-1}, & \text{if } 0\in\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right) \text{and } p \bmod 4=1 \text{ or } 3\\ \text{at least } 2^{n-2}+2^{n-u_1-1}-2^{n-u_3-1}, & \text{if } 0\in\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right) \text{and } p \bmod 4=1\\ \text{at least } 2^{n-1}-2^{n-u_2-1}-2^{n-u_3-1}, & \text{if } 0\in\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right) \text{and } p \bmod 4=3,\end{cases} \tag{4.82}$$

where u_1, u_2, and u_3 are $\min\left(\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)-\{0\}\right)$, $\min\left(\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)-\{0\}\right)$, and $\min\left(\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)-\{0,u_1\}\right)$, respectively.

Proof: $S_1\left(T_n^{\theta}(\varphi,\sigma,p)\right)=\sum_{i=0}^{2^n-1}c_i=\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^n-1}T_{i,j}\cdot f_j=\sum_{j=0}^{2^n-1}\sum_{i=0}^{2^n-1}T_{i,j}\cdot f_j=\sum_{j=0}^{2^n-1}f_j\left(\sum_{i=0}^{2^n-1}T_{i,j}\right)$.

Thus, a particular truth vector element f_j does not contribute to the value of $S_1\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ if the sum of elements in column j of $T_n^{\theta}(\varphi,\sigma,p)$ is equal to zero. Hence, the number of elements in F_1 = the number of columns in $T_n^{\theta}(\varphi,\sigma,p)$ for which the sum of its elements is equal to zero. Table 4.6 summarizes the distribution of nonzero elements in the columns of $T_n^{a}(\varphi,\sigma,p)$ based on Property 4.3.8. Summing up the number of columns with equal number of 1s and −1s, (4.82) is obtained.

For the Properties 4.3.13–4.3.15, it is assumed that $1 < \sigma < n + 1$. The same properties of $T_n^{a}(\varphi,\sigma,p)$ and $T_n^{b}(\varphi,\sigma,p)$ when $\sigma = 1$ or $\sigma = n + 1$ are given in Properties 4.3.16 and 4.3.17.

Table 4.6: Distribution of nonzero elements in the columns of $T_n^a(\varphi, \sigma, p)$.

p	For columns with column indexes $4 \cdot k$ $(0 \le k \le 2^{n-2}-1)$	For columns with column indexes $4 \cdot k+1$ $(0 \le k \le 2^{n-2}-1)$	For columns with column indexes $4 \cdot k+2$ $(0 \le k \le 2^{n-2}-1)$	For columns with column indexes $4 \cdot k+3$ $(0 \le k \le 2^{n-2}-1)$
p mod 4 = 0	The columns have one 1 and two or more −1s	The columns have one 1 and zero −1	The columns have one 1 and one −1	The columns have one 1 and zero −1
p mod 4 = 1	None of the columns have the same number of 1s and −1s when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$. However, some of them may possibly have the same number of 1s and −1s when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	From all the columns, $2^{n-2}-2^{n-u_1-1}$ of them have one 1 and one −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ whereas $2^{n-u_1-1}-2^{n-u_3-1}$ of them have one 1 and one −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	The columns have one 1 and zero −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ and one 1 and one −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	The columns have one 1 and one −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ and one 1 and zero −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$
p mod 4 = 2	The columns have one 1 and at least two −1s when $1 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$ whereas $2^{n-2}-2^{n-u_2-1}$ of them have one 1 and one −1 when $1 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$	The columns have one 1 and zero −1	The columns have one 1 and at least two −1s when $1 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ whereas $2^{n-2}-2^{n-u_1-1}$ of them have one 1 and one −1 when $1 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	The columns have one 1 and zero −1
p mod 4 = 3	None of the columns have the same number of 1s and −1s when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$. When $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$, at least $2^{n-2}-2^{n-u_2-1}$ of them have the same number of 1s and −1s	The columns have one 1 and one −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ and one 1 and zero −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	The columns have one 1 and zero −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ whereas $2^{n-2}-2^{n-u_1-1}$ of them have one 1 and one −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$	From all the columns, $2^{n-2}-2^{n-u_1-1}$ of them have one 1 and one −1 when $0 \in \varphi_{front}\left(T_n^\theta(\varphi, \sigma, p)\right)$ whereas $2^{n-u_1-1}-2^{n-u_3-1}$ of them have one 1 and one −1 when $0 \in \varphi_{back}\left(T_n^\theta(\varphi, \sigma, p)\right)$

Property 4.3.13. Let $S_2\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ be the sum of absolute values of all spectral coefficients for $T_n^{a}(\varphi,\sigma,p)$ and $T_n^{b}(\varphi,\sigma,p)$. Then $S_2\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ is always less than or equal to $\left(2^{n+1}-1\right)+h$ when S-coding is used, where h has been given in (4.75). For R-coding the maximum value of $S_2\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ is $(3\cdot 2^{n-1})-1$ when p is even and $2^n - 2^{n-\min\left(\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right)-1} - 2^{n-\min\left(\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right)-1} + \sum_{i=1}^{\sigma-1} 2^{n-\varphi_i-1} + \sum_{i=1}^{\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|} \gamma_i$ when p is odd, where $\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|$ denotes the number of elements in $\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ and γ_i is obtained by the procedure given below.

Procedure to obtain γ_i of $T_n^{a}(\varphi,\sigma,p)$ and $T_n^{b}(\varphi,\sigma,p)$

Step 1: Initialization

Let $A=\left\{A_1,A_2,\ldots,A_{\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|}\right\}$ be $\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ that is sorted such that $A_i > A_l$ if $i < l$ $(1 \le i, l \le \left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|)$;

Let $B=\left\{B_1,B_2,\ldots,B_{\left|\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|}\right\}$ be $\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)$ that is sorted such that $B_i < B_l$ if $i < l$ $(1 \le i, l \le \left|\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|)$;

temp=1; *temp1*=1; $s = 0$;

$k_1 = \left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right| + 1 - i$; // k_1 is the number of −1s in the processed rows

$k_2 = 2^{n-A_i-1}$;

if (i != 1)

$k_2 = k_2 - 2^{n-A_{i-1}-1}$; // k_2 is the number of processed rows

$k_3 = k_2$; // k_3 is the current maximum number of 1s in the processed rows

$j = 0$;

Step 2: $j = j + 1$;

//calculate the maximum number of 1s among the processed rows at the end of *Procedure2*($\sigma - j$) of *Procedure1* shown in Fig. 4.13 and the number of processed rows with such number of 1s at the end of *Procedure2*($\sigma - j$).

if ($j \le \left|\varphi_{back}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|$)

{if ($L_B(<p+1>_2) \ge (\min(A_i, B_j)+2)$) go to Step 3; //

else go to Step 4;}

else go to Step 6;

Step 3: $temp = temp + 1$;

if ($A_i > B_j$)

{for ($l = i+1$ to $\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|$)

{if ($A_l > B_j$)

$temp = temp + 1$;}

$k_3 = k_2$;}

else

{$k_3 = 2^{n-B_j-1}$;

for ($l = i-1$ to 1)

{if ($L_B(<p+1>_2) < (\min(A_l, B_j)+2)$)

{if (bits (A_i+2) to $(\min(A_l, B_j)+1)$ of $<p>_2 = 0$)

{$k_3 = k_3 - 2^{n-\max(A_l,B_j)-1}$;

break;}}}}

go to Step 5;

Step 4: for ($l = i+1$ to $\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|$)

{if ($L_B(<p+1>_2) \ge (\min(A_l, B_j)+2)$)

{if (bits (A_l+2) to $(\min(A_i, B_j)+1)$ of $<p>_2 = 0$)

{$temp = temp + 1$; $k_4 = k_3$;

$k_3 = 2^{n-\max(A_l,B_j)-1}$;

if ($k_3 > k_2$)

$k_3 = k_2$;

else if (i != 1)

{if (($A_{i-1} > B_j$) and (bits (A_i+2) to (B_j+1) of $<p>_2 = 0$))

$k_3 = k_3 - 2^{n-A_{i-1}-1}$;

if (($A_{i-1} < B_j$) and (bits (A_i+2) to $(A_{i-1}+1)$ of $<p>_2 = 0$))

{$temp = temp - 1$;

$k_3 = k_4;\}\}$

break;$\}\}\}$

go to Step 5;

Step 5: //Update value of s

if ($temp > k_1$) // there exist rows with more 1s than −1s among the processed rows

{if ($temp1 > k_1$) // this is not the first j for which the maximum number of 1s among the processed rows at the end of *Procedure2*($\sigma - j$) is greater than k_1

$s = s + (k_3 \cdot (temp - temp1))$;

else

$s = (k_1 \cdot (k_2 - k_3)) + (k_3 \cdot temp)$;$\}$

$temp1 = temp$;

go to Step 2;

Step 6: //get the value of γ_i

if ($temp \le k_1$) //none of the processed rows have more number of 1s than −1s

$\gamma_i = k_2 \cdot k_1$;

else // all or some of the processed rows have more number of 1s than −1s

$\gamma_i = s$;

exit the procedure;

Proof: $S_2\left(T_n^{\theta}(\varphi, \sigma, p)\right) = \sum_{i=0}^{2^n-1} |c_i| = \sum_{i=0}^{2^n-1} \left| \sum_{j=0}^{2^n-1} T_{i,j} \cdot f_j \right| \le \sum_{i=0}^{2^n-1} \max \left| \sum_{j=0}^{2^n-1} T_{i,j} \cdot f_j \right|$. For S-coding, $\sum_{i=0}^{2^n-1} \max \left| \sum_{j=0}^{2^n-1} T_{i,j} \cdot f_j \right|$ is simply equal to $\sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} |T_{i,j}|$, the total number of nonzero elements in the matrix. Hence, by Property 4.3.9 the upper bound for S-coding is $\left(2^{n+1} - 1\right) + h$.

For R-coding, $\sum_{i=0}^{2^n-1} \max \left| \sum_{j=0}^{2^n-1} T_{i,j} \cdot f_j \right| = \sum_{i=0}^{2^n-1} \max(\text{no. of 1s}, \text{no. of } -1\text{s}) \text{ in row } i =$

$\sum_{i=even} \max(\text{no. of 1s}, \text{no. of } -1\text{s}) \text{ in row } i + \sum_{i=odd} \max(\text{no. of 1s}, \text{no. of } -1\text{s}) \text{ in row } i$. When p is

even, the even rows of $T_n^a(\varphi,\sigma,p)$ only have one 1 and zero -1, whereas the odd rows of $T_n^a(\varphi,\sigma,p)$ have one 1 and between one to n -1s. Hence, max (no. of 1s, no. of -1s) in the even rows is equal to 1 whereas for odd rows it is equal to the number of -1s in the row. By Property 4.3.9, it follows that $\sum_{i=even}\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in row }i + \sum_{i=odd}\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in row }i$ when p is even $= 2^{n-1}+2^n-1 = (3\cdot 2^{n-1})-1$.

When p is odd, the even rows of $T_n^a(\varphi,\sigma,p)$ have one 1 and zero or more -1s. By Property 4.3.8, the number of even rows with zero -1s is equal to $2^{n-1}-2^{n-\min\left(\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}$ whereas the total number of -1s in the even rows $= \sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}$. Therefore, it follows that $\sum_{i=even}\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in row }i = 2^{n-1}-2^{n-\min\left(\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}$. The odd rows of $T_n^a(\varphi,\sigma,p)$, on the other hand, have between zero to $\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|$ -1s and one or more 1s. As γ_i $(1\le i\le \left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|)$ = sum of max (no. of 1s, no. of -1s) in rows with $(\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|+1-i)$ -1s, $\sum_{i=odd}\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in row }i = \sum_{i=1}^{\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|}\gamma_i$ + $\sum\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in odd rows with zero}-1\text{s}$ = $2^{n-1}-2^{n-\min\left(\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|}\gamma_i$. Hence, $\sum_{i=0}^{2^n-1}\max(\text{no. of 1s},\text{no. of}-1\text{s})\text{ in row }i = 2^{n-1}-2^{n-\min\left(\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}+2^{n-1}-2^{n-\min\left(\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|}\gamma_i$ $= 2^n-2^{n-\min\left(\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}-2^{n-\min\left(\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}+\sum_{i=1}^{\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|}\gamma_i$.

It should be mentioned that the procedure for obtaining γ_i given above is

developed based on *Procedure1* for $T_n^a(\varphi', \sigma, p)$, where $\varphi' = \left\langle A_1, A_2, ..., A_{\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|}, B_1, B_2, ..., B_{\left|\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|} \right\rangle$. However, since according to Property 4.3.4 $T_n^a(\varphi', \sigma, p) = T_n^a(\varphi, \sigma, p)$, this modification does not affect the result of the procedure.

Property 4.3.14. Let $S_3\left(T_n^\theta(\varphi, \sigma, p)\right)$ be the sum of the squares of spectral coefficients for $T_n^a(\varphi, \sigma, p)$ and $T_n^b(\varphi, \sigma, p)$, A and B be as given in Property 4.3.13, $A' = A \cup \{0\}$, and $B' = B - \{0\}$. Then $S_3\left(T_n^\theta(\varphi, \sigma, p)\right)$ is always less than or equal to $2^{n-1} + \sum_{i=1}^{|A'|} \varepsilon_i$ when p is even. When p is odd, $S_3\left(T_n^\theta(\varphi, \sigma, p)\right) \le 2^n + \left(\sum_{i=2}^{|B|} 2^{n-B_i-1} \cdot (2i-1)\right) - 2^{n-A_{|A|}-1} + \sum_{i=1}^{|A|} \gamma_i^2$ for R-coding and $S_3\left(T_n^\theta(\varphi, \sigma, p)\right) \le 2^n + 3 \cdot 2^{n-B_1-1} + \left(\sum_{i=2}^{|B|} 2^{n-B_i-1} \cdot (1+2i)\right) - 2^{n-A_{|A|}-1} + \sum_{i=1}^{|A|} \gamma_i^2$ for S-coding, where γ_i^2 and ε_i are obtained by the procedures given below.

Procedure to obtain γ_i^2 of $T_n^a(\varphi, \sigma, p)$ and $T_n^b(\varphi, \sigma, p)$

Steps 1 to 4: same as Steps 1 to 4 in the procedure to obtain γ_i of $T_n^a(\varphi, \sigma, p)$ and $T_n^b(\varphi, \sigma, p)$

Step 5: //Update value of *s*

if (j = 1)

$k_5 = k_2$; //initialize the value of k_5

if (S-coding is used)

$s = s + (k_5 - k_3)(k_1 + temp1)^2$;

else

{if ($temp > k_1$) // there exist rows with more number of 1s than −1s among the processed rows

{if ($temp1 > k_1$) // this is not the first j for which the maximum number of

1s among the processed rows at the end of *Procedure2*($\sigma - j$) is greater than k_1

$s = s + (k_5 - k_3)(temp1)^2$;

else

$s = (k_2 - k_3)(k_1)^2$;}}

$temp1 = temp$;

$k_5 = k_3$;

go to Step 2;

Step 6: //get the value of γ_i^2

if (R-coding is used)

{$s = s + (k_3 \cdot temp^2)$;

if ($temp \le k_1$) //all of the processed rows have more number of −1s than 1s

$\gamma_i^2 = k_2 \cdot (k_1)^2$;

else // all or some of the processed rows have more number of 1s than −1s

$\gamma_i^2 = s$;}

else //S-coding is used

{$s = s + k_3 \cdot (k_1 + temp)^2$;

$\gamma_i^2 = s$;}

exit the procedure;

Procedure to obtain ε_i of $T_n^a(\varphi, \sigma, p)$ and $T_n^b(\varphi, \sigma, p)$

Step 1: Initialization

$k_1 = |A'| + 1 - i$; // k_1 is the number of −1s in the processed rows just after *Procedure2*(σ)

$temp = k_1$; $temp1 = k_1$; $\varepsilon_i = 0$;

$k_2 = 2^{n-A'_i-1}$;

if (i != 1)

$k_2 = k_2 - 2^{n-A'_{i-1}-1}$; // k_2 is the number processed rows

$k_3 = k_2$; // k_3 is the current maximum number of 1s in the processed rows

$k_4 = k_2$;

$j = 0$;

Step 2: $j = j + 1$;

//Calculate the maximum number of −1s among the processed rows at the end of *Procedure2*($\sigma - j$) of *Procedure1* shown in Fig. 4.13 and the number of processed rows with such number of −1s at the end of *Procedure2*($\sigma - j$).

if ($j \le |B'|$)

{if ($L_B(<p>_2) \ge (\min(A_i', B_j') + 2)$) go to Step 3; //

else go to Step 4;}

else go to Step 5;

Step 3:

if ($A_i' > B_j'$)

$temp = temp + 1$;}

else

{if ((i !=1) and (bits ($A_i' + 2$) to ($\min(A_{i-1}', B_j') + 1$) of $<p>_2 = 0$))

{ $k_3 = 2^{n-B_j'-1} - 2^{n-\max(A_{i-1}', B_j')-1}$;

if (k_3 != 0)

$temp = temp + 1$;

else

$k_3 = k_4$;}

else

{$temp = temp + 1$;

$k_3 = 2^{n-B_j'-1}$;}}

go to Step 4;

Step 4: //Update value of ε_i

if (S-coding is used)

$\varepsilon_i = \varepsilon_i + (k_4 - k_3)(temp1 + 1)^2$;

else //R-coding is used

$\varepsilon_i = \varepsilon_i + (k_4 - k_3)(temp1)^2$;

$temp1 = temp$;

$k_4 = k_3$;

go to Step 2;

Step 5: //Get the final value of ε_i

if (R-coding is used)

$\varepsilon_i = \varepsilon_i + k_3 \cdot (temp)^2$;

else // S-coding is used

$\varepsilon_i = \varepsilon_i + k_3 \cdot (temp + 1)^2$;

exit the procedure;

Proof: $S_3\left(T_n^{\theta}(\varphi,\sigma,p)\right) = \sum_{i=0}^{2^n-1} c_i^{\,2} = \sum_{i=0}^{2^n-1}\left(\sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j\right)^2 \le \sum_{i=0}^{2^n-1}\left(\max\left|\sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j\right|\right)^2$. It has been shown in the proof of Property 4.3.13 that when p is even the even rows of $T_n^a(\varphi,\sigma,p)$ only have one 1 and zero -1 and the odd rows of $T_n^a(\varphi,\sigma,p)$ have one 1 and between one to n -1s. Also, when p is odd the even rows of $T_n^a(\varphi,\sigma,p)$ have one 1 and zero or more -1s and the odd rows of $T_n^a(\varphi,\sigma,p)$ have between zero to $\left|\varphi_{front}\left(T_n^{\theta}(\varphi,\sigma,p)\right)\right|$ -1s and one or more 1s, where the odd rows with zero -1s have only one 1.Therefore, it can be written that

$$\sum_{i=0}^{2^n-1}\left(\max\left|\sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j\right|\right)^2 = \sum_{i=odd}\left(\max\left|\sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j\right|\right)^2 + \sum_{i=even}\left(\max\left|\sum_{j=0}^{2^n-1} T_{i,j}\cdot f_j\right|\right)^2$$

$$= \sum_{i=odd}(\text{no.of}-1\text{s in row } i)^2 + \sum_{i=even} 1 \text{ when } p \text{ is even and R - coding is used}$$

$$= \sum_{i=odd}(1+\text{no.of}-1\text{s in row } i)^2 + \sum_{i=even} 1 \text{ when } p \text{ is even and S - coding is used}$$

$$= \sum_{i=odd}(\max(\text{no.of 1s, no.of}-1\text{s}) \text{ in row } i)^2 + \sum_{\substack{i=\text{even rows}\\ \text{with zero } -1\text{s}}} 1 +$$

$$\sum_{\substack{i=\text{even rows}\\ \text{with one or more } -1\text{s}}} (\text{no.of}-1\text{s in row } i)^2 \text{ when } p \text{ is odd and R - coding is used}$$

$$= \sum_{i=odd}(\text{no.of 1s} + \text{no.of}-1\text{s in row } i)^2 + \sum_{\substack{i=\text{even rows}\\ \text{with zero } -1\text{s}}} 1 +$$

$$\sum_{\substack{i=\text{even rows}\\ \text{with one or more } -1\text{s}}} (1+\text{no.of}-1\text{s in row } i)^2 \text{ when } p \text{ is odd and S - coding is used.}$$

Let us first examine the cases when p is even. By observing the given procedure, it can be seen that for even p $\sum_{i=1}^{|A'|}\varepsilon_i = \sum_{i=odd}(\text{no.of } -1\text{s in row } i)^2$ for R-coding whereas for S-coding $\sum_{i=1}^{|A'|}\varepsilon_i = \sum_{i=odd}(1+\text{no.of } -1\text{s in row } i)^2$. Hence, $\sum_{i=0}^{2^n-1}\left(\max\left|\sum_{j=0}^{2^n-1}T_{ij}\cdot f_j\right|\right)^2 = 2^{n-1}+\sum_{i=1}^{|A'|}\varepsilon_i$ when p is even.

When p is odd, the numbers of even rows in $T_n^a(\varphi,\sigma,p)$ with zero, j, and $|B|$ -1s are $2^{n-1}-2^{n-B_1-1}$, $2^{n-B_j-1}-2^{n-B_{j+1}-1}$, and $2^{n-B_{|B|}-1}$, respectively. Substituting this numbers, it can be easily obtained that $\sum_{\substack{i=\text{even rows}\\ \text{with zero } -1\text{s}}}1 + \sum_{\substack{i=\text{even rows}\\ \text{with one or more } -1\text{s}}}(\text{no. of } -1\text{s in row } i)^2 = 2^{n-1}+\left(\sum_{i=2}^{|B|}2^{n-B_i-1}\cdot(2i-1)\right)$ and $\sum_{\substack{i=\text{even rows}\\ \text{with zero } -1s}}1 + \sum_{\substack{i=\text{even rows}\\ \text{with one or more } -1s}}(1+\text{no. of } -1\text{s in row } i)^2 = 2^{n-1}+3\cdot 2^{n-B_1-1}+\left(\sum_{i=2}^{|B|}2^{n-B_i-1}\cdot(1+2i)\right)$.

Steps 1 to 4 of the procedure to obtain γ_i of $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$ calculates the numbers of 1s in the rows with $(\left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|+1-i)$ -1s $(1\le i\le \left|\varphi_{front}\left(T_n^\theta(\varphi,\sigma,p)\right)\right|)$ and the corresponding numbers of such rows with such number of 1s. Hence, by modifying Steps 5 and 6 appropriately, the procedure can be used to calculate $\sum_{\substack{j=\text{odd rows}\\ \text{with } i\ -1s}}(\max(\text{no.of 1s, no. of } -1\text{s}) \text{ in row } j)^2$ and $\sum_{\substack{j=\text{odd rows}\\ \text{with } i\ -1s}}(\text{no.of 1s} + \text{no. of } -1\text{s in row } j)^2$ for R-coding and S-coding, respectively. As $\sum_{j=0}^{|A|}(\text{number of odd rows with } j-1\text{s}) = \text{number of odd rows}$, $\sum_{i=odd}\left(\max\left|\sum_{j=0}^{2^n-1}T_{i,j}\cdot f_j\right|\right)^2 = \text{no.of odd rows with zero} -1\text{s} + \sum_{i=1}^{|A|}\gamma_i^2 = 2^{n-1}-2^{n-A_{|A|}-1}+\sum_{i=1}^{|A|}\gamma_i^2$. Combining the

above, it is obtained that when p is odd, $\sum_{i=0}^{2^n-1}\left(\max\left|\sum_{j=0}^{2^n-1}T_{i,j}\cdot f_j\right|\right)^2$ $= 2^n+\left(\sum_{i=2}^{|B|}2^{n-B_i-1}\cdot(2i-1)\right)-2^{n-A_{|A|}-1}+\sum_{i=1}^{|A|}\gamma_i^2$ for R-coding and $2^n+3\cdot 2^{n-B_1-1}+\left(\sum_{i=2}^{|B|}2^{n-B_i-1}\cdot(1+2i)\right)-2^{n-A_{|A|}-1}+\sum_{i=1}^{|A|}\gamma_i^2$ for S-coding.

Property 4.3.15. Let z be the subscript value of the ordering digit whose value is equal to zero. Then when $1 \le z \le \sigma - 1$, for $\theta \in \{a,b\}$ $S_2\left(T_n^\theta(\varphi,\sigma,p)\right)$ for R-coding and odd value of p is always less than or equal to $2^{n-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}+h$, where

$$h=\sum_{i=n}^{\sigma}\sum_{j=\sigma-1}^{1}\begin{cases}2^{n-(\max(\varphi_i,\varphi_j)+1)}, & \text{if } L_B(<p+1>_2)\ge(\min(\varphi_i,\varphi_j)+2)\\ 0, & \text{otherwise.}\end{cases}$$

Proof: By Property 4.3.8, when $1 \le z \le \sigma - 1$ and p is odd, the number of 1s in odd rows of $T_n^a(\varphi,\sigma,p)$ is always greater than or equal to the number of -1s. Hence, it follows that $\sum_{i=odd}\max(\text{no.of 1s},\text{no.of}-1\text{s})$ in row i = total no. of 1s in $T_n^\theta(\varphi,\sigma,p)$ − the total no. of 1s in even rows. By Properties 4.3.8 and 4.3.9, $\sum_{i=odd}\max(\text{no.of 1s},\text{no.of}-1\text{s})$ in row i $= 2^n+h-2^{n-1} = 2^{n-1}+h$. Since in the proof of Property 4.3.13 it has been shown that $\sum_{i=even}\max(\text{no.of 1s},\text{no.of}-1\text{s})$ in row i $= 2^{n-1}-2^{n-\min\left(\varphi_{back}\left(T_n^\theta(\varphi,\sigma,p)\right)\right)-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}$, $\sum_{i=0}^{2^n-1}\max(\text{no.of 1s},\text{no.of}-1\text{s})$ in row i $= 2^{n-1}+\sum_{i=1}^{\sigma-1}2^{n-\varphi_i-1}+h$.

Property 4.3.16. For $\sigma = 1$ or $\sigma = n+1$, the maximum value of $S_2\left(T_n^\theta(\varphi,\sigma,p)\right)$ is $2^{n+1}-1$ when S-coding is used and $3\cdot 2^{n-1}-1$ when R-coding is used.

Property 4.3.17. For $\sigma = 1$ or $\sigma = n + 1$, $S_3\left(T_n^{\theta}(\varphi,\sigma,p)\right) \le 2^n + \sum_{i=2}^{n} 2^{n-i} \cdot (2i-1)$ when R-coding is used and $S_3\left(T_n^{\theta}(\varphi,\sigma,p)\right) \le 5 \cdot 2^{n-1} + \sum_{i=2}^{n} 2^{n-i} \cdot (2i+1)$ when S-coding is used.

4.3.3 Experimental results for fastest LIA transforms

The calculation of fastest LIA transforms spectral coefficients as well as binary FPAE transforms spectra have been implemented in C programs and run on a Pentium IV 2.8 GHz computer with 512 MB RAM for several standard binary benchmark functions. For each binary benchmark function, its coefficient column vectors for all fastest LIA transforms as well as FPAE transforms in all polarities are generated and the number of nonzero spectral coefficients and the largest absolute spectral coefficient value in each spectrum are computed. The results are shown in Tables 4.7 – 4.12, where the numbers for fastest LIA transforms that are less than or equal to their arithmetic counterparts are written in italic.

In Tables 4.7 and 4.8, the minimum numbers of nonzero spectral coefficients in the spectra of fastest LIA transforms with polarity number zero for R and S coding are presented. The numbers of nonzero spectral coefficients inside the FPAE spectrum in polarity zero as well as the optimal polarity are also shown. In the tables, the numbers for the fastest LIA transforms that are not larger than the corresponding numbers of the polarity zero FPAE transform are written in italic. Table 4.9 lists the minimum numbers of nonzero spectral coefficients in the spectra of all fastest LIA transforms of types *a* and *b* whereas the numbers for polarities *c* and *d* are shown in Table 4.10. In the tables, the numbers that are written in italic are those which are smaller than or equal to the corresponding numbers for the optimal FPAE transform. From the numbers in Tables 4.7 and 4.8, it can be seen that the minimum number of nonzero spectral coefficients for $T_n^c(\varphi,\sigma,0)$ and $T_n^d(\varphi,\sigma,0)$ transforms are generally smaller than the number of nonzero spectral coefficients in $T_n^a(\varphi,\sigma,0)$ and $T_n^b(\varphi,\sigma,0)$ spectra.

Table 4.7: Minimum number of nonzero spectral coefficients (R-coding).

Input files	$T_n^a(\varphi,\sigma,0)$	$T_n^b(\varphi,\sigma,0)$	$T_n^c(\varphi,\sigma,0)$	$T_n^d(\varphi,\sigma,0)$	FPAE in polarity zero	Optimal FPAE transform
xor5	*21*	*21*	*15*	*15*	31	31
squar5	30	31	30	31	23	23
rd53	*28*	32	*23*	*22*	31	31
bw	*26*	*22*	*26*	*22*	32	22
con1	104	94	76	74	21	18
5xp1	128	128	128	128	62	62
z5xp1	128	128	128	128	62	62
rd73	*122*	128	*92*	*93*	127	127
inc	*92*	*89*	*81*	*81*	92	49
rd84	*249*	256	*182*	*187*	255	255
misex1	105	104	66	64	60	20
ex5	*240*	*223*	*208*	*191*	256	113
9sym	*344*	*344*	*184*	*184*	465	352
z9sym	*344*	*344*	*184*	*184*	465	352
clip	494	493	481	478	264	255
apex4	495	493	462	460	445	445

By Properties 4.3.2 and 4.3.4, the number of nonzero spectral coefficients of $T_n^c(\varphi_a^n,\sigma,0)$ and $T_n^d(\varphi_a^n,\sigma,0)$ are equal to that of any $T_n^a(\varphi,\sigma,0)$ and $T_n^b(\varphi,\sigma,0)$ transforms, respectively. Therefore, it can also be concluded from Tables 4.7 and 4.8 that reordering the factorized matrices, i.e., changing the ordering φ, helps to obtain new fastest LIA polynomial expansions with smaller number of polynomial terms. Similarly, by comparing the numbers for fastest LIA transforms in Tables 4.7 – 4.10, it can be clearly seen that adding a permutation matrix in the factorized representation of fastest LIA transforms are useful as some of the fastest LIA transforms built from

both factorized matrices and permutation matrix can provide binary functions with representations that have smaller number of nonzero terms compared to the representations based on the fastest LIA transforms built from the factorized matrices only.

Table 4.8: Minimum number of nonzero spectral coefficients (S-coding).

Input files	$T_n^a(\varphi,\sigma,0)$	$T_n^b(\varphi,\sigma,0)$	$T_n^c(\varphi,\sigma,0)$	$T_n^d(\varphi,\sigma,0)$	FPAE in polarity zero	Optimal FPAE transform
xor5	*32*	*32*	*22*	*22*	32	32
squar5	31	32	31	32	24	24
rd53	*32*	*32*	*24*	*24*	32	32
bw	*30*	*30*	*28*	*28*	32	23
con1	102	103	76	78	21	18
5xp1	128	128	128	128	62	62
z5xp1	128	128	128	128	62	62
rd73	*128*	*128*	*94*	*94*	128	128
inc	108	107	*87*	*88*	92	50
rd84	*256*	*256*	*187*	*187*	256	256
misex1	198	198	99	99	60	21
ex5	*240*	*223*	*208*	*191*	256	113
9sym	*398*	*398*	*215*	*215*	466	352
z9sym	*398*	*398*	*215*	*215*	466	352
clip	502	502	484	484	265	255
apex4	495	493	462	460	445	445

Looking at the numbers in Tables 4.7 – 4.10, it can be noticed that the fastest LIA transforms can give smaller number of nonzero spectral coefficients than the optimal FPAE transform for many of the binary benchmark functions, such as rd73. As the values of a function for different assignment of variables can be calculated more quickly when the function is represented by a polynomial expansion with smaller number of terms, this advantage of the fastest LIA transforms over best polarity arithmetic expansion can be useful in analysis of logic circuits using signal

probabilities. Furthermore, if we compare the numbers in Table 4.9 with the numbers in Table 4.10, it can be seen that the numbers in Table 4.10 are generally smaller than or equal to the corresponding numbers in Table 4.9. This shows that among all fastest LIA transforms, the one that gives the minimum number of nonzero spectral coefficients is in most cases the types *c* and *d* of fastest LIA transforms.

Tables 4.11 and 4.12 compare the largest absolute spectral coefficient value of fastest LIA and FPAE transforms. In Table 4.11, the largest absolute spectral coefficient value of the polarity zero arithmetic transform spectra and the fastest LIA transforms with zero polarity number spectra are given whereas Table 4.12 lists the largest absolute spectral coefficient value among all fastest LIA transforms and all polarities FPAE transforms spectral coefficients. It can be clearly seen from the tables that the fastest LIA transforms often have smaller value of maximum absolute spectral coefficients than FPAE transforms. This is important since the largest absolute spectral coefficient value may affect the stability of the system. In addition, larger spectral coefficient requires more storage space when they are stored in memory. Note that all the numbers in the Tables 4.11 and 4.12 for fastest LIA transforms are less than or equal to the theoretical upper bounds for maximum absolute value of spectral coefficients given in Property 4.3.11.

Table 4.9: Minimum number of nonzero spectral coefficients for $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$.

Input files	$T_n^a(\varphi,\sigma,p)$		$T_n^b(\varphi,\sigma,p)$	
	R-coding	S-coding	R-coding	S-coding
xor5	*16*	*24*	*16*	*24*
squar5	30	31	30	31
rd53	*27*	*31*	*28*	*31*
bw	*22*	28	*22*	28
con1	70	77	70	75
5xp1	128	128	128	128
z5xp1	128	128	128	128
rd73	*118*	*124*	*122*	*124*
inc	75	87	75	87
rd84	*240*	*251*	*249*	*254*
misex1	76	130	76	130
ex5	207	207	196	196
9sym	*218*	*300*	*218*	*300*
z9sym	*218*	*300*	*218*	*300*

Table 4.10: Minimum number of nonzero spectral coefficients for $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$.

Input files	$T_n^c(\varphi,\sigma,p)$		$T_n^d(\varphi,\sigma,p)$	
	R-coding	S-coding	R-coding	S-coding
xor5	*14*	*21*	*14*	*21*
squar5	30	31	30	31
rd53	*21*	*22*	*21*	*22*
bw	*22*	28	*22*	28
con1	67	69	66	69
5xp1	120	120	124	124
z5xp1	128	128	128	128
rd73	*85*	*86*	*85*	*86*
inc	75	84	75	84

Table 4.11: Largest absolute spectral coefficient value.

Input files	$T_n^a(\varphi,\sigma,0)$		$T_n^b(\varphi,\sigma,0)$		$T_n^c(\varphi,\sigma,0)$		$T_n^d(\varphi,\sigma,0)$		FPAE in polarity zero	
	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding
xor5	*2*	*2*	*3*	*2*	*3*	*4*	*3*	*4*	16	32
squar5	*3*	*5*	*2*	*4*	*4*	*6*	*4*	*5*	4	8
rd53	*2*	*5*	*3*	*3*	*3*	*5*	*3*	*4*	16	32
bw	*4*	*5*	*2*	*6*	*4*	*5*	*4*	*6*	4	8
con1	5	6	4	*4*	5	6	6	7	2	4
5xp1	*4*	*6*	*6*	*6*	*6*	*8*	*6*	*8*	6	12
z5xp1	7	*8*	*6*	*8*	7	*8*	7	*8*	6	12
rd73	*3*	*5*	*4*	*5*	*5*	*6*	*4*	*6*	64	128
inc	*4*	*7*	*4*	*6*	5	*7*	5	*7*	4	8
rd84	*4*	*17*	*4*	*17*	*5*	*9*	*5*	*7*	128	256
misex1	5	7	5	7	6	7	5	7	3	6
ex5	7	7	7	9	7	8	7	9	3	6
9sym	*4*	*5*	*4*	*5*	*6*	*7*	*6*	*7*	14	28
z9sym	*4*	*5*	*4*	*5*	*6*	*7*	*6*	*7*	14	28
clip	*7*	*6*	*7*	*7*	*8*	*10*	*7*	*10*	12	24
apex4	*6*	*8*	*7*	*8*	*6*	*9*	*7*	*9*	20	40

Table 4.12: Largest absolute spectral coefficient value of all spectra.

Input files	$T_n^a(\varphi,\sigma,p)$		$T_n^b(\varphi,\sigma,p)$		$T_n^c(\varphi,\sigma,p)$		$T_n^d(\varphi,\sigma,p)$		FPAE transform in all polarities	
	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding	R-coding	S-coding
xor5	*3*	*5*	*4*	*5*	*4*	*5*	*4*	*5*	16	32
squar5	*4*	10	5	9	*4*	10	5	10	4	8
rd53	*5*	*8*	*5*	*7*	*5*	*8*	*5*	*6*	16	32
bw	*5*	*8*	*5*	*8*	*5*	*8*	*5*	*9*	5	10
con1	10	11	9	9	10	12	9	11	3	6
5xp1	10	17	13	17	12	17	13	17	6	12
z5xp1	13	17	10	17	13	17	12	17	6	12
rd73	*9*	*14*	*11*	*14*	*11*	*14*	*11*	*14*	64	128
inc	7.5	13	10	13	10	14	10	13	5	10

Chapter 5

Algorithms for Efficient Computation of Ternary and Quaternary Fixed Polarity Arithmetic Expansions

FPAE, which can be thought of as the integer counterpart of FPRME, have been reviewed in Chapters 1 and 2 for binary functions. Similar to the FPRME, the binary FPAEs can also be extended to multiple-valued cases. This chapter is concerned with the extension of FPAEs for ternary and quaternary functions and development of efficient algorithms for their calculation. The basic definitions for FPAEs of any p-valued functions, where $p > 2$, are first presented. Two special cases of the FPAEs are then discussed for the case where $p = 3$ (ternary) and $p = 4$ (quaternary). For each of the cases, the corresponding basic FPAE transforms are shown and their computational costs are derived. Algorithms for computation of their FPAE spectra are subsequently described.

5.1 FPAEs for multiple-valued functions

Definition 5.1.1. Let $f(\vec{x})$ be an n-variable p-valued function. Then the general form

of the FPAE for $f(\vec{x})$ in polarity ω is

$$f(\vec{x}) = \sum_{i=0}^{p^n-1} c_i^{\omega} \left[\prod_{l=1}^{n} \hat{x}_l^{\,k_l} \right], \tag{5.1}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]$, c_i^{ω} is the i-th spectral coefficient inside FPAE in polarity ω, $\hat{x}_l$ is the literal of the l-th variable, i is the decimal number representation of n-digit p-valued number $< k_1, k_2, \ldots, k_n >$, i.e., $< i >_{10} = < k_1, k_2, \ldots, k_n >_p$, and $\hat{x}_l^{k_l} \in \{0, 1, \ldots, p-1\}$ is the k_l-th power of $\hat{x}_l$ mod p. All the additions and multiplication are performed in standard arithmetic algebra.

Definition 5.1.2. In an FPAE, each input variable x_i $(1 \le i \le n)$ must appear in the same literal throughout the expansion. Let the possible literals for an p-valued variable x_i be denoted by ${}^{<j_i>}x_i$, where ${}^{<j_i>}x_i = (x_i + j_i) \bmod p$ and $0 \le j_i \le p-1$. Then the polarity of an FPAE is taken as the decimal number representation of the n-digit p-valued number $< j_1, j_2, \ldots, j_n >$, i.e., $< \omega >_{10} = < j_1, j_2, \ldots, j_n >_p$ if the variable x_i $(1 \le i \le n)$ always appears as ${}^{<j_i>}x_i$ inside the FPAE.

Definition 5.1.3. The FPAE spectral coefficient vector in polarity ω for the function $f(\vec{x})$ is denoted by C^{ω} where

$$C^{\omega} = [c_0^{\omega}, c_1^{\omega}, \ldots, c_{p^n-1}^{\omega}], \tag{5.2}$$

$0 \le \omega \le p^n - 1$, and c_i^{ω} represents the i-th spectral coefficient in the FPAE of $f(\vec{x})$ in polarity ω.

Definition 5.1.4. The FPAE polarity matrix of an n-variable p-valued function $f(\vec{x})$ is denoted by $P[f(\vec{x})]$. It is a $p^n \times p^n$ square matrix whose rows correspond to the FPAE spectral coefficient vectors C^{ω} such that

$$P[f(\vec{x})] = \left[C^0, C^1, \ldots, C^{p^n-1} \right]^T, \tag{5.3}$$

where T denotes matrix transpose operator.

Definition 5.1.5. The column coefficient vector of a function $f(\vec{x})$ in polarity ω is denoted by B^{ω}, where

$$B^{\omega} = [b_0^{\omega}, b_1^{\omega}, \ ..., b_{p^n-1}^{\omega}]^T . \tag{5.4}$$

The vector B^{ω} corresponds to column ω $(0 \le \omega \le p^n - 1)$ of $P[f(\vec{x})]$.

Property 5.1.1. Let the element that is located at row g and column h of $P[f(\vec{x})]$ be denoted by $P_{g,h}$ $(0 \le g, h \le p^n - 1)$. Then by Definitions 5.1.3–5.1.5, $P_{g,h} = c_h^g = b_g^h$.

5.2 FPAEs for ternary functions

Let $\vec{F} = [F_0, F_1, ..., F_{3^n-1}]^T$ be the truth vector of an n-variable ternary function $f(\vec{x})$, where $\vec{x} = [x_1, x_2, \ldots, x_n]$, $F_0 = f(0,0,\ldots,0)$, $F_1 = f(0,0,\ldots,1)$, and $F_{3^n-1} = f(2,2,\ldots,2)$. Then C^{ω} for $f(\vec{x})$ can be calculated by the transform

$$C^{\omega} = S_n^{<\omega>} \cdot \vec{F} , \tag{5.5}$$

where $S_n^{<\omega>}$ is the FPAE forward transform matrix for ternary functions. It is a square matrix of size $3^n \times 3^n$ that can be constructed from the n-times Kronecker product

$$S_n^{<\omega>} = \otimes \prod_{l=1}^{n} S_1^{<j_l>} , \tag{5.6}$$

where $<\omega>_{10} = <j_1, j_2, \ldots, j_n>_3$, $\otimes$ denotes Kronecker product, and $S_1^{<j_l>}$ denotes the FPAE forward transform matrix for one-variable ternary function in polarity j_l ($j_l \in \{0, 1, 2\}$). The basic transforms $S_1^{<j_l>}$ are given below.

$$S_1^{<0>} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$S_1^{<1>} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 1 & 0 \\ 2 & -1 & -1 \end{bmatrix}$$

$$S_1^{<2>} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Consequently, if we denote the inverse of $S_1^{<j_l>}$ over standard arithmetic algebra by $T_1^{<j_l>}$ then we can build the FPAE inverse transform matrix $T_n^{<\omega>}$ by

$$T_n^{<\omega>} = \otimes\prod_{l=1}^{n} T_1^{<j_l>}, \tag{5.7}$$

where

$$\vec{F} = T_n^{<\omega>} \cdot C^{\omega}. \tag{5.8}$$

Note that the matrices $T_1^{<j_l>}$ ($j_l \in \{0, 1, 2\}$) are exactly the same as the basic inverse transform for FPRME over GF(3) [Gre89]. Their values are

$$T_1^{<0>} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

$$T_1^{<1>} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$T_1^{<2>} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

Example 5.2.1. Let $f(\vec{x})$ be a two-variable ternary function with truth vector $\vec{F} = [0,1,2,1,2,1,0,0,0]^T$. Then, C^3 for $f(\vec{x})$ can be calculated as follows

$$C^3 = S_n^{<3>} \cdot \vec{F} = (S_1^{<1>} \otimes S_1^{<0>}) \cdot \vec{F}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & -1 & 2 & -1 & 0 & 0 & 0 \\ 2 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -2 & 2 & 0 & 1 & -1 & 0 & 1 & -1 \\ -2 & 4 & -2 & 1 & -2 & 1 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -2 \\ 2 \\ -1 \\ 3 \\ -2 \end{bmatrix}.$$

So, by Definitions 5.1.1 and 5.1.2 the FPAE for $f(\vec{x})$ in polarity three is

$$f(\vec{x}) = \dot{x}_1 - 2\dot{x}_1 x_2 + 2\dot{x}_1 x_2^{\,2} - \dot{x}_1^{\,2} + 3\dot{x}_1^{\,2} x_2 - 2\dot{x}_1^{\,2} x_2^{\,2},$$

where $\dot{x}_1$ and x_2 are equivalent to the literals ${}^{<1>}x_1$ and ${}^{<0>}x_2$ in Definition 5.1.2.

Given the FPAE, we can get back the output value of $f(\vec{x})$ for a pair of x_1 and x_2 values by evaluating the expression in standard arithmetic algebra, except for the power operations which are performed modulo 3. Hence, for $x_1 = 1$ and $x_2 = 1$,

$$\begin{aligned} f(\vec{x}) &= 2 - 2 \cdot 2 \cdot 1 + 2 \cdot 2 \cdot 1 - 1 + 3 \cdot 1 \cdot 1 - 2 \cdot 1 \cdot 1 \\ &= 2 - 4 + 4 - 1 + 3 - 2 = 2. \end{aligned}$$

Let ω_1 and ω_2 be any two polarity numbers such that $< \omega_1 >_{10} =< \omega_{11}, \omega_{12}, \ldots, \omega_{1n} >_3$ and $< \omega_2 >_{10} =< \omega_{21}, \omega_{22}, \ldots, \omega_{2n} >_3$ where $0 \le \omega_1, \omega_2 \le 3^n - 1$. Then, by (5.5)–(5.8)

$$\begin{aligned} C^{\omega_1} &= S_n^{<\omega_1>} \cdot \vec{F} \\ &= \left(\otimes \prod_{l=1}^{n} S_1^{<\omega_{1l}>} \cdot T_1^{<0>} \right) \cdot C^0 \\ &= \left(\otimes \prod_{l=1}^{n} Z_1^{<\omega_{1l}>} \right) \cdot C^0, \end{aligned} \tag{5.9}$$

where $Z_1^{<0>} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $Z_1^{<1>} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & -3 & -2 \end{bmatrix}$, and $Z_1^{<2>} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & -1 \\ 0 & 3 & 1 \end{bmatrix}$.

Furthermore, as $Z_1^{<1>} = \left(Z_1^{<2>} \right)^{-1}$

$$C^{\omega_1} = \left(\otimes \prod_{l=1}^{n} Z_1^{<\omega_{1l}>} \right) \cdot \left(\otimes \prod_{l=1}^{n} \left(Z_1^{<\omega_{2l}>} \right)^{-1} \right) \cdot C^{\omega_2}$$

$$= \left(\otimes \prod_{l=1}^{n} Z_1^{<\omega_{1l}>} \cdot Z_1^{<2\omega_{2l}>} \right) \cdot C^{\omega_2}$$

$$= \left(\otimes \prod_{l=1}^{n} Z_1^{<(\omega_{1l}+2\omega_{2l}) \bmod 3>} \right) \cdot C^{\omega_2}$$

$$= Z_n^{<\omega_1+2\omega_2>} \cdot C^{\omega_2}. \tag{5.10}$$

Each matrix $S_1^{<j_l>}$ has six nonzero elements with one element has an absolute value greater than one. By the property of the Kronecker product, it can be easily obtained that there are 6^n nonzero elements inside $S_n^{<\omega>}$, where 5^n of them have absolute values of one. As such, the numbers of additions/subtractions and multiplications required to perform direct transform of (5.5) are

$$A_1(n) = 6^n - 3^n \tag{5.11}$$

and

$$M_1(n) = 6^n - 5^n, \tag{5.12}$$

respectively whereas the computational cost for performing fast transform are

$$A_2(n) = n \cdot 3^n \text{ additions/subtractions} \tag{5.13}$$

and

$$M_2(n) = n \cdot 3^{n-1} \text{ multiplications.} \tag{5.14}$$

Similarly, it can be obtained that the Kronecker product of n $Z_1^{<1>}$ produces a matrix with 7^n nonzero elements where 4^n of them have absolute values of one whereas Kronecker product of n $Z_1^{<2>}$ have the same number of nonzero elements with 5^n of them are either 1 or -1. Hence, it follows that the computational cost of performing direct transform of (5.10) is

$$A_3(n) = 3^n \left(\left(\frac{7}{3} \right)^{u+v} - 1 \right) \text{ additions/subtractions} \tag{5.15}$$

and

$$M_3(n) = 3^{n-(u+v)} \left(7^{u+v} - 4^u \cdot 5^v \right) \text{ multiplications,} \tag{5.16}$$

where u and v are the number of 1s and 2s in the ternary number $<\omega_1 + 2\omega_2>$, respectively. Performing fast transform reduces the computational cost of (5.10) to

$$A_4(n) = 4 \cdot (u+v) \cdot 3^{n-1} \text{ additions/subtractions} \tag{5.17}$$

and

$$M_4(n) = (3u+2v) \cdot 3^{n-1} \text{ multiplications.} \tag{5.18}$$

Similar to the case for FPRME over GF(5), the relationships between the truth vector and FPAEs in different polarities as shown in (5.5)–(5.10) can be used to generate algorithms for calculating FPAEs for ternary functions. The algorithms are described in the following sections.

5.2.1 Matrix multiplication algorithms for ternary FPAEs

In the matrix multiplication algorithm, the generation of all FPAEs for the input function is performed by first choosing an arbitrary ordering of all the 3^n possible polarity numbers. It starts by calculating the FPAE spectral coefficient vector in the arbitrary starting polarity from the truth vector by (5.5). After it has been obtained, the FPAEs in the remaining polarities are then calculated one at a time by (5.10) such that the FPAE spectrum in a particular polarity is computed from the FPAE spectrum in the previous polarity number in the chosen ordering. For example, if lexicographic ordering is chosen then for two-variable functions the polarity numbers are ordered as follows: 0-1-2-3-4-5-6-7-8 and so C^{ω_1} is calculated from $C^{\omega_1 - 1}$ for all ω_1, $1 \le \omega_1 \le 8$.

In Section 5.2, it has been shown that the computational cost of (5.10) changes with the values of ω_1 and ω_2. As a result, the computational cost of the matrix multiplication algorithms is dependent on the chosen ordering and whether fast transform is used. In what follows, the computational costs for three different orderings are derived.

When the polarity numbers are ordered in lexicographic order, in the generation of the FPAE spectra in all the $3^n - 1$ nonstarting polarities by (5.10), the nonzero digits in $<\omega_1 + 2\omega_2>$ are always one. Furthermore, there are $2 \cdot 3^{n-i}$ instances where the

number of nonzero digits inside $<\omega_1 + 2\omega_2>$ are i ($1 \le i \le n$). As such, the computational cost of generating all FPAEs in lexicographic order using fast transform is

$$A_5(n) = A_2(n) + \sum_{i=1}^{n} 2 \cdot 3^{n-i}(4 \cdot i \cdot 3^{n-1})$$

$$= 3^{n-1}\left(2 \cdot 3^{n+1} - n - 6\right) \text{ additions/subtractions} \qquad (5.19)$$

and

$$M_5(n) = M_2(n) + \sum_{i=1}^{n} 2 \cdot 3^{n-i}(3 \cdot i \cdot 3^{n-1})$$

$$= \frac{3^{n-1}}{2}\left(3^{n+2} - 4n - 9\right) \text{ multiplications.} \qquad (5.20)$$

On the other hand, when all the polarities are ordered in the reverse lexicographic order (from $3^n - 1$ down to zero), the number of instances where the number of nonzero digits inside $<\omega_1 + 2\omega_2>$ are i ($1 \le i \le n$) are the same as in lexicographic order. However, now the nonzero digits are always two instead of one. From (5.15)–(5.18), it can be deduced that compared to the lexicographic order, the reverse lexicographic order lead to the same number of additions/subtractions but smaller number of multiplications. The computational cost for the reverse lexicographic order with fast transform is

$$A_6(n) = 3^{n-1}\left(2 \cdot 3^{n+1} - n - 6\right) \text{ additions/subtractions} \qquad (5.21)$$

and

$$M_6(n) = M_2(n) + \sum_{i=1}^{n} 2 \cdot 3^{n-i}(2 \cdot i \cdot 3^{n-1})$$

$$= 3^{n-1}\left(3^{n+1} - n - 3\right) \text{ multiplications.} \qquad (5.22)$$

Alternatively, the computational cost can be minimized by choosing an ordering such that the ternary number $<\omega_1 + 2\omega_2>$ in (5.10) always has only one nonzero digit, i.e., the ordering is an extended dual polarity route (see Definition 3.4.1). As $Z_1^{<1>}$ has more nonzero elements than $Z_1^{<2>}$, the best case computational cost occurs when the extended dual polarity route is one such that the nonzero digit in

$< \omega_1 + 2\omega_2 >$ is always two. The matrix multiplication algorithm with such ordering is given in Fig. 5.1. The computational cost of the algorithm is

$$A_7(n) = 3^{n-1}\left(4 \cdot 3^n + 3n - 4\right) \text{ additions/subtractions} \tag{5.23}$$

and

$$M_7(n) = 3^{n-1}\left(2 \cdot 3^n + n - 2\right) \text{ multiplications.} \tag{5.24}$$

The computational costs for the matrix multiplication algorithms with fast transform and the three described orderings are shown in Table 5.1 for $n < 7$.

5.2.2 Cube polarity adjustment algorithm for ternary FPAEs

In Chapter 3, an algorithm called cube polarity adjustment has been introduced to calculate the spectral coefficient of FPRMEs over GF(5) from the disjoint cube array representations of the input functions. In this section, the algorithm is extended for generating FPAE spectral coefficients of ternary functions.

Definition 5.2.1. Let the literal of a ternary variable in a cube be denoted by $X_i^{S_i}$, where $S_i \subseteq \{0,1,2\}$ is said to be the true set of literal X_i and $X_i^{S_i} = \begin{cases} 1 \text{ if } X_i \in S_i \\ 0 \text{ if } X_i \notin S_i \end{cases}$. Then a ternary cube is simply a product of ternary literals $X_1^{S_1} X_2^{S_2} \ldots X_i^{S_i}$, $i \le n$.

Definition 5.2.2. A cube intersection operation is defined as

$$A \cap B = \begin{cases} X_1^{S_1 A \cap S_1 B} X_2^{S_2 A \cap S_2 B} \cdots X_n^{S_n A \cap S_n B}, & \text{if } S_i A \cap S_i B \neq \varnothing \text{ for all } i \\ \varnothing, & \text{otherwise} \end{cases}, \tag{5.25}$$

where A and B are cubes and S_iA and S_iB are true sets of literal X_i in A and B, respectively.

Definition 5.2.3. Two cubes A and B are disjoint if $A \cap B = \varnothing$. Otherwise, A and B are nondisjoint (overlapped).

```
Void tfpae(int truth vector[ ])
{int W[n] = <0, 0, …., 0>, p[ ];
 char Dir[n] = 'aaa…a';
 Calculate C^0 from truth vector using (5.5) and
  store the result in p[ ];
 For(j = 0 to 3^n – 2)
  { cont = true;
    l = n;
    while (cont==true)
      { l = l —;
        cont = false;
        if(Dir[l]=='a')
        { switch (W[l])
          { case 0: W[l] = 2; break;
            case 1: { cont = true; Dir[l] = 'b';}
            case 2: W[l] = 1; break; }}
        else if(Dir[l]=='b')
        { switch (W[l])
          { case 0: W[l] = 2; break;
            case 1: W[l] = 0; break;
            case 2: { cont = true; Dir[l] = 'c';}} }
        else
        { switch (W[l])
          { case 0: { cont=true; Dir[l] = 'a';}
            case 2: W[l] = 1; break;
            case 1: W[l] = 0; break; } }}
    Calculate C^W from p[ ] and store the result
      back to p[ ];}}
```

Figure 5.1: Matrix multiplication algorithm for ternary FPAEs.

Table 5.1: Computational cost of matrix multiplication algorithms for ternary FPAEs.

n	Additions/subtractions			Multiplications		
	Lexicographic order	Reverse lexicographic order	Algorithm in Fig. 5.1	Lexicographic order	Reverse lexicographic order	Algorithm in Fig. 5.1
1	11	11	11	7	5	5
2	138	138	114	96	66	54
3	1377	1377	1017	999	675	495
4	12852	12852	8964	9504	6372	4428
5	117207	117207	79623	87399	58401	39609
6	1059966	1059966	711990	793152	529254	355266

Definition 5.2.4. A minterm is a cube that includes literals for all function variables $X_1, X_2, ..., X_n$ in which every set S_i has only one element.

Example 5.2.2. Let $X_1^{\{1,2\}} X_2^{\{0,1\}} X_3^{\{2\}}$ be a ternary cube with three input variables. Then the minterms covered by the cube are $X_1^{\{1\}} X_2^{\{0\}} X_3^{\{2\}}$, $X_1^{\{1\}} X_2^{\{1\}} X_3^{\{2\}}$, $X_1^{\{2\}} X_2^{\{0\}} X_3^{\{2\}}$, and $X_1^{\{2\}} X_2^{\{1\}} X_3^{\{2\}}$ or 102, 112, 202, and 212, respectively.

Definition 5.2.5. A cube can be represented by its positional notation such that each literal is represented by *r* (number of input values) binary digits, with the (*i* + 1)-th digit is 1 when *i* is a member of the true set of the literal and 0 otherwise.

Example 5.2.3. The ternary cube $X_1^{\{1,2\}} X_2^{\{0,1\}} X_3^{\{2\}}$ given in Example 5.2.2 can be represented in positional notation as 011–110–001.

The cube polarity adjustment algorithm calculates each of the spectral coefficient values by computing the contribution of each disjoint cube to the value of the spectral coefficient and summing them up. The computation process is exactly the same for all

spectral coefficients and it allows the required spectral coefficients to be calculated in any order. In what follows, the basic definitions and properties related to the algorithm is given. The steps of the algorithm are also described. The input to the algorithm is the reduced representation of a ternary function in the form of disjoint cubes array, where each cube can be represented as an integer coded cube or by its positional notation. Therefore, the algorithm is given for both cases.

Definition 5.2.6. The canonical FPAE of $f(\vec{x})$ in polarity ω for ternary function can also be expressed by

$$f(\vec{x}) = \sum_{i=0}^{3^n-1} c_i^{\omega} \pi_i^{\omega} , \tag{5.26}$$

where each piterm π_i^{ω} is equivalent to the product term $\left[\prod_{l=1}^{n} \hat{x}_l^{\,k_l}\right]$. In other word, π_i^{ω} is the product term in FPAE that corresponds to the spectral coefficient c_i^{ω}.

Property 5.2.1. There are 3^n different possible π_i^{ω} for a particular polarity number that corresponds to the different permutations of k_l.

Property 5.2.2. The *n*-variable ternary FPAE can be fully described by the set of all spectral coefficients c_i^{ω}, where $c_i^{\omega} \in \{0,1,2\}$.

Definition 5.2.7. The subscript *i* in both π_i^{ω} and c_i^{ω} is called the spectral coefficient index. From Definition 5.1.1, $< i >_{10} =< k_1, k_2, \ldots, k_n >_3$.

Definition 5.2.8. The cube of degree *d* is a cube that has *d* defined literals. If the symbol *m* denotes the number of missing literals ('–') in the cube and *n* denotes the total number of input variables for the function then $d = n - m$.

Property 5.2.3. The missing literal in a cube corresponds to the literal $X_i^{\{0,1,2\}}$ as well as '111' in positional notation.

Definition 5.2.9. The ω-adjustment operator is an operator that takes a polarity digit and a ternary cube literal as its inputs. The cube can be represented either by integer coded cube or positional notation. Tables 5.2 and 5.3 show the ω-adjustment operation when the cube is an integer coded cube and in positional notation, respectively.

Table 5.2: Cube ω-adjustment operation (ternary integer coded cube).

ω-adj	Polarity digit		
Cube digit	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1
–	–	–	–

Table 5.3: Cube ω-adjustment operation (ternary positional notation).

ω-adj	Polarity digit		
Variable positional notation	0	1	2
001	001	100	010
010	010	001	100
011	011	101	110
100	100	010	001
101	101	110	011
110	110	011	101
111	111	111	111

Definition 5.2.10. The π-adjustment operator is defined as an operation between a spectral coefficient index digit and a ternary cube literal. Similar to the ω-adjustment operator, the cube input for the π-adjustment operation can be in the form of either integer coded cube or positional notation. Tables 5.4 and 5.5 show the

outputs of the π-adjustment operation for all possible inputs.

Table 5.4: Cube π-adjustment operation (ternary integer coded cube).

π-adj	Spectral coefficient index digit		
Cube digit	0	1	2
0	1	0	1
1	0	1	2
2	0	1	1

Table 5.5: Cube π-adjustment operation (ternary positional notation).

π-adj	Spectral coefficient index digit		
Variable positional notation	0	1	2
001	0	1	1
010	0	1	2
011	0	0	1
100	1	0	1
101	1	1	2
110	1	1	1

Definition 5.2.11. Let h_q^i denotes the contribution of the q-th disjoint cube to the value of the i-th spectral coefficient c_i^{ω}. Then the final value of c_i^{ω} is a summation of all contributions to it in standard arithmetic algebra. If z is the number of ON disjoint cubes inside the input function, then

$$c_i^{\omega} = \sum_{q=1}^{z} h_q^i \,. \tag{5.27}$$

In the algorithm, the contribution of each disjoint cube to the value of each spectral coefficient is calculated separately using the same procedure. Let the q-th ON disjoint

cube of an *n*-variable ternary function be given by $q = q_1 q_2 \cdots q_n$. Then the value of h_q^i as described in Definition 5.2.11 can be calculated by the following steps:

Step 1: Generate the *n*-digit ternary representation of the spectral coefficient index i, $< k_1, k_2, \ldots, k_n >$ and polarity number ω, $< j_1, j_2, \ldots, j_n >$.

Step 2: Perform bitwise operation on q and ω:

$$q' = q_{\omega - adj}\, \omega .$$

Step 3: Set $i' = i$. If the degree of the cube $q' = n$, go to Step 4. Otherwise perform the following:

- If the cube is an integer coded cube:

  For every digit of the cube q'_l $(1 \le l \le n)$, if q'_l ='–' then set $q'_l = i'_l$ and $i'_l = 0$.

- If the cube is represented by its positional notation:

  For every literal in the cube q'_l $(1 \le l \le n)$, if $q'_l = 111$ then set q'_l = 100, 010, or 001 according to whether i'_l = 0, 1, or 2, respectively. Subsequently set $i'_l = 0$.

Step 4: Perform bitwise operation on q' and i':

$$q'' = q'_{\pi - adj}\, i' .$$

Step 5: Set $M = \prod_{l=1}^{n} q''_l$.

Step 6: Determine the sign of h_q^i by the following:

- If the cube is an integer coded cube:

  Perform bitwise modulo 3 addition, $temp = q' \oplus i'$. If the number of nonzero digit in *temp* is even, $sign = 1$. Else, $sign = -1$.

- If the cube is represented by its positional notation:

  Set *temp* = 0. For every nonzero digit in i', i'_l $(1 \le l \le n)$: if ($i'_l = 1$ and the rightmost digit in the positional notation of the l-th input variable in q' = 0) or ($i'_l = 2$ and the middle digit in the positional notation of the

l-th input variable in q' = 0) then set *temp* = *temp* + 1. If the final value of *temp* is even, $sign = 1$. Else, $sign = -1$.

Step 7: Let $f(q)$ be the output value of the disjoint cube q. Then,

$$h_q^i = sign \times M \times f(q).$$

Example 5.2.4. Let $f(\vec{x})$ be a three-variable ternary function with the following disjoint cubes array: {00– 1, 01– 1, 2–0 1, 222 1, 020 2, 022 2, 100 2, 102 2}. Then, according to the given procedure the computation process of the spectral coefficient c_{24}^5 ($<5>_{10}=<0,1,2>_3$; $<24>_{10}=<2,2,0>_3$) is as illustrated in Table 5.6.

Table 5.6: Computation example for ternary integer coded cubes.

q	$f(q)$	q' (after Step 2)	q' (after Step 3)	i'	q''	M	*temp*	*sign*	h_q^i
00–	1	01–	010	220	121	2	200	–1	–2
01–	1	02–	020	220	111	1	210	1	1
2–0	1	2–2	222	200	100	0	122	–1	0
222	1	201	201	220	110	0	121	–1	0
020	2	002	002	220	110	0	222	–1	0
022	2	001	001	220	110	0	221	–1	0
100	2	112	112	220	220	0	002	–1	0
102	2	111	111	220	220	0	001	–1	0
								c_{24}^5	–1

Example 5.2.5. Let $f(\vec{x})$ be the three-variable ternary function given in Example 5.2.4. Then $f(\vec{x})$ can also be represented by the following disjoint cubes in positional notation: {100–110–111 1; 001–001–001 1; 001–111–100 1, 010–100–101 2; 100–001–101 2}. Table 5.7 shows the process of calculating the value of c_{24}^5 from the disjoint cubes by the presented algorithm. As the functions in Examples 5.2.4 and 5.2.5 are the same, the values of c_{24}^5 are also the same for both of them.

Table 5.7: Computation example for ternary positional notation.

q	$f(q)$	q' (after Step 2)	q' (after Step 3)	i'	q''	M	$temp$	$sign$	h_q^i
100–110–111	1	100–011–111	100–011–100	220	111	1	1	−1	−1
001–001–001	1	001–100–010	001–100–010	220	110	0	2	1	0
001–111–100	1	001–111–001	001–001–001	200	100	0	1	−1	0
010–100–101	2	010–010–011	010–010–011	220	220	0	0	1	0
100–001–101	2	100–100–011	100–100–011	220	110	0	2	1	0
								c_{24}^5	−1

5.2.3 Row polarity matrix algorithm for ternary FPAEs

The row polarity matrix algorithm generates the required FPAE spectral coefficients based on the recursive definition of the polarity matrix that has been modified using its special properties such that the resulting computational costs are reduced. In what follows, the definitions and properties of the recursive definitions used by the algorithm are presented.

Definition 5.2.12. Let each spectral coefficient vector C^{ω} be divided into three equal size subvectors as follows:

$$C^{\omega} = [c_{[n-1,0]}^{\omega}, c_{[n-1,1]}^{\omega}, c_{[n-1,2]}^{\omega}]. \tag{5.28}$$

Since there are 3^n elements in C^{ω}, each of the subvectors $c_{[n-1,q]}^{\omega}$ $(q \in \{0,1,2\})$ contains 3^{n-1} elements.

Property 5.2.4. The definition above is recursive and each subvector of C^{ω} can be further divided into three subvectors of equal size. Generally for $1 \le p \le n$,

$$c_{[p,q]}^{\omega} = [c_{[p-1,q0]}^{\omega}, c_{[p-1,q1]}^{\omega}, c_{[p-1,q2]}^{\omega}], \tag{5.29}$$

where q is the $(n-p)$-digit ternary representation of any decimal number from zero to

$3^{n-p}-1$.

Property 5.2.5. Let $< q >_{10}$ be the decimal number representation of q in $c^{\omega}_{[p,q]}$. Then the elements of $c^{\omega}_{[p,q]}$ correspond to the polarity matrix $P[f(\vec{x})]$ elements with row index numbers ω and column index numbers j, where $\left(< q >_{10} \cdot 3^{p}\right) \le j < \left(< q >_{10} +1\right)\cdot 3^{p}$.

Property 5.2.6. Each subvector $c^{\omega}_{[p,q]}$ has 3^p elements. When $p=n$, $c^{\omega}_{[p,q]}$ has 3^n elements, and therefore $c^{\omega}_{[n,0]} = C^{\omega}$.

Definition 5.2.13. Let T_p be a recursive square matrix which is defined by (5.30). The subscript p is allowed to take the value from 1 to n.

$$T_p(c^{\omega}_{[p,q]}) = \left[T_{p0}(c^{\omega}_{[p,q]}), T_{p1}(c^{\omega}_{[p,q]}), T_{p2}(c^{\omega}_{[p,q]})\right], \tag{5.30}$$

where

$$T_{p0}(c^{\omega}_{[p,q]}) = \begin{bmatrix} T_{p-1}(c^{\omega}_{[p-1,q0]}) \\ T_{p-1}(c^{\omega}_{[p-1,q0]} + 2c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]}) \\ T_{p-1}(c^{\omega}_{[p-1,q0]} + c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]}) \end{bmatrix},$$

$$T_{p1}(c^{\omega}_{[p,q]}) = \begin{bmatrix} T_{p-1}(c^{\omega}_{[p-1,q1]}) \\ T_{p-1}(c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]}) \\ T_{p-1}(-2c^{\omega}_{[p-1,q1]} - c^{\omega}_{[p-1,q2]}) \end{bmatrix},$$

and

$$T_{p2}(c^{\omega}_{[p,q]}) = \begin{bmatrix} T_{p-1}(c^{\omega}_{[p-1,q2]}) \\ T_{p-1}(-3c^{\omega}_{[p-1,q1]} - 2c^{\omega}_{[p-1,q2]}) \\ T_{p-1}(3c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]}) \end{bmatrix}.$$

Property 5.2.7. Each matrix $T_p(c^{\omega}_{[p,q]})$ is of size $3^p \times 3^p$.

Property 5.2.8. By Property 5.2.7, $T_1(c^{\omega}_{[1,q]})$ only has 3×3 elements. Therefore

$T_0(c^{\omega}_{[0,s]}) = c^{\omega}_{[0,s]} = c^{\omega}_{s}$.

Property 5.2.9. When $\omega = 0$ and $p = n$, the matrix $T_p(c^{\omega}_{[p,q]})$ is the polarity matrix, $P[f(\vec{x})] = T_n(c^{0}_{[n,0]})$. Furthermore, by Property 5.2.6, $c^{0}_{[n,0]} = C^0$ and $P[f(\vec{x})] = T_n(C^0)$.

Definition 5.2.14. By analyzing the relations between the elements of polarity matrix $P[f(\vec{x})]$, (5.30) can be rewritten as:

$$T_p(c^{\omega}_{[p,q]}) = \begin{bmatrix} T_{p-1}(t_{00}) & T_{p-1}(t_{01}) & T_{p-1}(t_{02}) \\ T_{p-1}(t_{10}) & T_{p-1}(t_{11}) & T_{p-1}(t_{12}) \\ T_{p-1}(t_{20}) & T_{p-1}(t_{21}) & T_{p-1}(t_{22}) \end{bmatrix}$$

$$= \begin{bmatrix} T_{p-1}(t_{00}) & T_{p-1}(t_{01}) & T_{p-1}(t_{02}) \\ T_{p-1}(t_{01}+t_{20}) & T_{p-1}(t_{01}+t_{02}) & T_{p-1}(t_{21}-t_{11}) \\ T_{p-1}(t_{00}+t_{11}) & T_{p-1}(t_{00}-t_{10}) & T_{p-1}(t_{01}-t_{21}) \end{bmatrix}. \quad (5.31)$$

It can be seen that the revised recursive equation of $T_p(c^{\omega}_{[p,q]})$ in (5.31) is more efficient than (5.30) for generating the complete polarity matrix.

The row polarity matrix algorithm calculates all elements of the polarity matrix by recursively applying (5.30) on all $T_p(c^{\omega}_{[p,q]})$ matrices in each recursion stage. The steps performed by the algorithm are as follows:

Step 1: Generate polarity zero spectral coefficient vector by (5.5) if it is not known. Otherwise go directly to Step 2.

Step 2: Set $p = n$.

Step 3: Divide $P[f(\vec{x})]$ into $3^{n-p} \times 3^{n-p}$ T_p matrices. For each of the T_p matrices, divide it into its T_{p-1} matrices and calculate the first row elements of all the T_{p-1} matrices from the first row elements of the T_p matrix by (5.31). Fig. 5.2 shows the flow diagram for this calculation for each T_p matrix. Note that in the figure, the dashed line ---- indicates that the sign of the signal should be inverted before being added.

Step 4: Decrement p by 1. If $p = 0$, all elements of the polarity matrix have been derived. Exit the algorithm. Otherwise go back to Step 3.

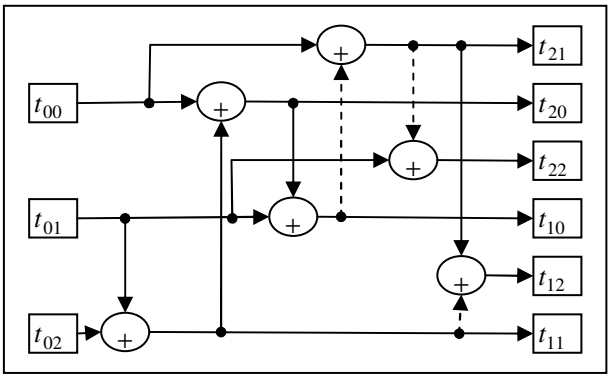


Figure 5.2: Flow diagram of the calculation of rows 1 and 2 matrices.

Property 5.2.10. The computational cost of deriving the complete polarity matrix of an n-variable ternary function from the FPAE spectral coefficient vector in polarity zero by the algorithm is $3^n\left(3^n-1\right)$ additions/subtractions.

Proof: By the given steps, there are n recursion stages in the algorithm. At the i-th recursion ($1 \le i \le n$), there are $9^{i-1}\ T_{n+1-i}$ matrices being processed, each requires three additions and three subtractions of size 3^{n-i}. Therefore, the total computational cost (additions/subtractions) for the algorithm is

$$A_8(n) = \sum_{i=1}^{n} 9^{i-1} \cdot 6 \cdot 3^{n-i} = 2 \cdot 3^{n-1} \cdot \sum_{i=1}^{n} 3^i = 3^n\left(3^n-1\right).$$

The computational costs of obtaining all spectral coefficient vectors by the row polarity matrix algorithm are given in Table 5.8 for $n < 7$. Note that the computational cost of calculating polarity zero spectrum by fast transform of (5.5) has been added to the cost given in Property 5.2.10. Comparing Tables 5.1 and 5.8, it can be seen that the row polarity matrix algorithm requires smaller number of additions and multiplications than all the matrix multiplication algorithms.

Table 5.8: Computational costs of row polarity matrix algorithm (ternary FPAEs).

n	Additions/subtractions	Multiplications
1	9	1
2	90	6
3	783	27
4	6804	108
5	60021	405
6	535086	1458

Besides deriving the complete polarity matrix, the algorithm can also be used to generate a spectral coefficient vector only without the necessity of obtaining the whole polarity matrix first. The steps of the algorithm in this case is similar to the steps given above, except that only 3^{n-p} T_p matrices which contain the selected spectral coefficient vector are processed at each recursion stage and that only one row of T_{p-1} matrices need to be derived for each of the T_p matrices. Figs. 5.3 and 5.4 show the flow diagrams of the calculation of row 1 and row 2 T_{p-1} matrices from the first row elements of the T_p matrix, respectively. The algorithm is finished when the selected spectral coefficient vectors elements have all been obtained.

The properties of the calculation of a spectral coefficient vector by the row polarity matrix algorithm are given in the following. For the properties, let ω be the polarity number of the selected spectral coefficient vector, where $<\omega>_{10} = < j_1, j_2, \ldots, j_n >_3$.

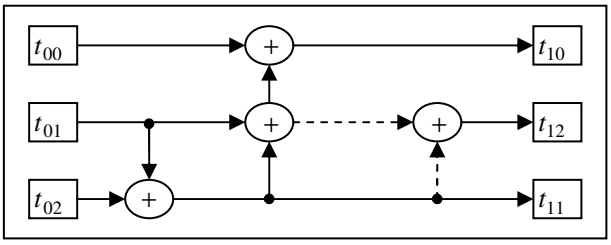


Figure 5.3: Flow diagram of the calculation of row 1 matrices.

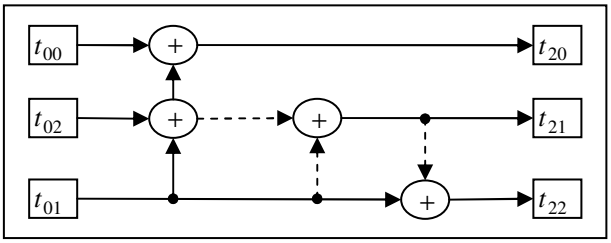


Figure 5.4: Flow diagram of the calculation of row 2 matrices.

Property 5.2.11. The T_p row index number of the matrices to be derived at the i-th recursion ($1 \le i \le n$) is given by r_i ($0 \le r_i \le 2$), where

$$r_i = \text{int}\left(\frac{\omega}{3^{n-i}}\right) \text{for } i = 1, \tag{5.32}$$

$$r_i = \text{int}\left(\frac{\omega - \sum_{l=1}^{i-1} r_l \cdot 3^{n-l}}{3^{n-i}}\right) \text{for } i > 1, \tag{5.33}$$

and the bracket int (rational number) means the integer part of a rational number.

Property 5.2.12. If $[\{(\omega \bmod 3^{n-1}) \bmod 3^{n-2}\} \ldots \bmod 3^{n-i}] = 0$ for some i, $1 \le i \le n$, the total number of the polarity matrix rows that are generated in the process of calculating the FPAE spectrum in polarity ω is equal to i.

Property 5.2.13. If $[\{(\omega \bmod 3^{n-1}) \bmod 3^{n-2}\} \ldots \bmod 3^{n-i}] = R$ ($R \in \{1, 2\}$) for some i, $1 \le i \le n$, the total number of the polarity matrix rows that are generated in the process of calculating the FPAE spectrum in polarity ω is equal to $i + 1$.

Property 5.2.14. Let the set $Y = \{y_1, y_2, \cdots, y_{|Y|}\}$ be defined as $Y = \{y \mid j_y \neq 0\}$. Then the polarity numbers of the generated spectral coefficient vectors are given by the decimal number representations of ternary numbers $< j_1, j_2, ..., j_{y_i}, 0, ..., 0 >$, where the value of index i varies from 1 to $|Y|$.

Property 5.2.15. The computational cost of deriving the FPAE spectra in polarity ω by the row polarity matrix algorithm is $4\cdot|Y|\cdot 3^{n-1}$ additions/subtractions.

5.2.4 Column polarity matrix algorithm for ternary FPAEs

The column polarity matrix algorithm calculates the required FPAE spectra based on an optimized recursive definition of the ternary FPAE polarity matrix. The optimized recursive definition is obtained by examining the relations between the polarity matrix columns and minimizing the required number of arithmetic additions/subtractions and multiplications. Similar approach has been presented for FPRMEs over GF(5).

Let us first examine the FPAE transform polarity matrix for one-variable ternary function with truth vector $\vec{F}=[F_0,F_1,F_2]^T$. By (5.5) and Definition 5.1.4, the spectral coefficient vectors for the function are

$$C^0=[F_0,F_2-F_1,2F_1-F_0-F_2]$$

$$C^1=[F_2,F_1-F_0,2F_0-F_2-F_1]$$

$$C^2=[F_1,F_0-F_2,2F_2-F_1-F_0]$$

and the polarity matrix for the function is

$$P[f(\vec{x})]=\begin{bmatrix} F_0 & F_2-F_1 & 2F_1-F_0-F_2 \\ F_2 & F_1-F_0 & 2F_0-F_2-F_1 \\ F_1 & F_0-F_2 & 2F_2-F_1-F_0 \end{bmatrix}.$$

It can be seen that the first column of the polarity matrix consists of truth vector elements and as a result, the elements in the second and third columns can be obtained from the first column elements. Generalizing this analysis for larger ternary functions, it can be derived that for any number of input variables the first column of the polarity matrix (B^0) is simply the truth vector of the input function that has been reordered in a regular manner. Once the first column elements are obtained, other column elements can then be recursively calculated from them by utilizing the relations between different column coefficient vectors. The general recursive definition for the ternary

FPAE polarity matrix that enables such calculation is given below.

Definition 5.2.15. Let $T_\beta^{<p,q>}(\vec{g})$ be a recursive matrix that is defined as

$$T_\beta^{<p,q>}(\vec{g}) = \begin{bmatrix} T_{\beta-1}^{<p0,q0>}(\vec{g}) & T_{\beta-1}^{<p0,q1>}(\vec{g}) & T_{\beta-1}^{<p0,q2>}(\vec{g}) \\ T_{\beta-1}^{<p1,q0>}(\vec{g}) & T_{\beta-1}^{<p1,q1>}(\vec{g}) & T_{\beta-1}^{<p1,q2>}(\vec{g}) \\ T_{\beta-1}^{<p2,q0>}(\vec{g}) & T_{\beta-1}^{<p2,q1>}(\vec{g}) & T_{\beta-1}^{<p2,q2>}(\vec{g}) \end{bmatrix}$$

$$= \begin{bmatrix} t_{00} & t_{01} & t_{02} \\ t_{10} & t_{11} & t_{12} \\ t_{20} & t_{21} & t_{22} \end{bmatrix}, \tag{5.34}$$

where $\vec{g}$ is a column vector of size 3^n, $T_d^{<p,q>}(\vec{g})$ $(d \in \{\beta, \beta-1\})$ is a square matrix of size $3^d \times 3^d$, p and q are ternary strings, $1 \le \beta \le n$, and the columns are related such that $\begin{bmatrix} t_{01} \\ t_{11} \\ t_{21} \end{bmatrix} = \begin{bmatrix} t_{10} - t_{20} \\ t_{20} - t_{00} \\ t_{00} - t_{10} \end{bmatrix}$ and $\begin{bmatrix} t_{02} \\ t_{12} \\ t_{22} \end{bmatrix} = \begin{bmatrix} 2t_{20} - t_{00} - t_{10} \\ 2t_{00} - t_{10} - t_{20} \\ 2t_{10} - t_{20} - t_{00} \end{bmatrix}$.

Definition 5.2.16. Let $\vec{F}$ be the truth vector of an n-variable ternary function $f(\vec{x})$. Then $Y_\beta^{<v>}(f(\vec{x}))$ is defined as a subvector of $\vec{F}$ that satisfies (5.35) and (5.36).

$$Y_n^{<0>}(f(\vec{x})) = \vec{F} \tag{5.35}$$

$$Y_\beta^{<v>}(f(\vec{x})) = \left[Y_{\beta-1}^{<v,0>}(f(\vec{x})), Y_{\beta-1}^{<v,1>}(f(\vec{x})), Y_{\beta-1}^{<v,2>}(f(\vec{x}))\right] \tag{5.36}$$

There are 3^β elements inside $Y_\beta^{<v>}(f(\vec{x}))$, where $1 \le \beta \le n$ and v is an $(n-\beta+1)$-digits ternary number.

Definition 5.2.17. Let $Y_\beta^{<v>}(f(\vec{x}))$ be as in Definition 5.2.16. Then $X_\beta^{<v>}(f(\vec{x}))$ $(1 \le \beta \le n)$ is defined as a subvector of size 3^β that is recursively built from $Y_\beta^{<v>}(f(\vec{x}))$ by (5.37) and (5.38).

$$X_1^{<v>}(f(\vec{x})) = \left[Y_0^{<v,0>}(f(\vec{x})), Y_0^{<v,2>}(f(\vec{x})), Y_0^{<v,1>}(f(\vec{x}))\right] \tag{5.37}$$

$$X_\beta^{<v>}(f(\vec{x})) = \left[X_{\beta-1}^{<v,0>}(f(\vec{x})), X_{\beta-1}^{<v,2>}(f(\vec{x})), X_{\beta-1}^{<v,1>}(f(\vec{x}))\right], (\beta > 1) \tag{5.38}$$

Property 5.2.16. $T_{\beta}^{<p,q>}(\vec{g})$ is the FPAE polarity matrix for an n-variable function $f(\vec{x})$ when $p = q = 0$ and $\vec{g} = X_n^{<0>}(f(\vec{x}))$,

$$P(f(\vec{x})) = T_n^{<0,0>}\left(X_n^{<0>}(f(\vec{x}))\right). \tag{5.39}$$

In the recursive definition of $T_{\beta}^{<p,q>}(\vec{g})$ given in Definition 5.2.15, the generation of submatrices inside the second and third columns from the first column submatrices requires nine matrix subtractions and three matrix scalar multiplications. This computational cost can be reduced by considering the relations between the second and third column submatrices such as given in Definition 5.2.18.

Definition 5.2.18. The recursive matrix $T_{\beta}^{<p,q>}(\vec{g})$ in Definition 5.2.15 can also be written as

$$T_{\beta}^{<p,q>}(\vec{g}) = \begin{bmatrix} t_{00} & t_{10} & -t_{20} & t_{11} & -t_{01} \\ t_{10} & t_{20} & -t_{00} & t_{21} & -t_{11} \\ t_{20} & t_{00} & -t_{10} & t_{01} & -t_{21} \end{bmatrix}. \tag{5.40}$$

It can be seen that the new recursive equation in (5.40) requires smaller number of arithmetic operations than the recursive equation in (5.34).

The column polarity matrix algorithm can be used to calculate all or selected polarities spectral coefficient vectors. When the complete polarity matrix is to be generated, the algorithm performs the calculation based on (5.40) as it incurs smaller computational cost in terms of number of arithmetic operations. On the other hand, when only selected spectral coefficient vectors are required, the computation is performed based on (5.34) to reduce the number of intermediate spectra that needs to be calculated in the process. The steps of the algorithm for both cases are given below.

Steps for generation of complete ternary FPAE polarity matrix by column polarity matrix algorithm

Step 1: Find $X_n^{<0>}(f(\vec{x}))$ from $\vec{F}$, the truth vector of the input function $f(\vec{x})$, by (5.35)–(5.38). $X_n^{<0>}(f(\vec{x}))$ is B^0 of $f(\vec{x})$. Set $\beta = n$.

Step 2: Divide the polarity matrix $P[f(\vec{x})]$ into $9^{n-\beta}$ $T_{\beta}^{<p,q>}$ matrices. For each $T_{\beta}^{<p,q>}$ matrices, identify its t_{00}, t_{10}, t_{20}, t_{01}, t_{11}, t_{21}, t_{02}, t_{12}, and t_{22} submatrices. Calculate the first column elements of the t_{01}, t_{11}, t_{21}, t_{02}, t_{12}, and t_{22} submatrices from the known first column elements of t_{00}, t_{10}, and t_{20} submatrices by (5.40). Fig. 5.5 shows the flow diagram of the calculation inside each $T_{\beta}^{<p,q>}$ matrices, where the dashed line ---- indicates that the sign of the signal should be inverted before being added.

Step 3: Decrement β by 1. If $\beta = 0$, the complete polarity matrix has been generated and the algorithm is finished. Otherwise repeat Steps 2 and 3.

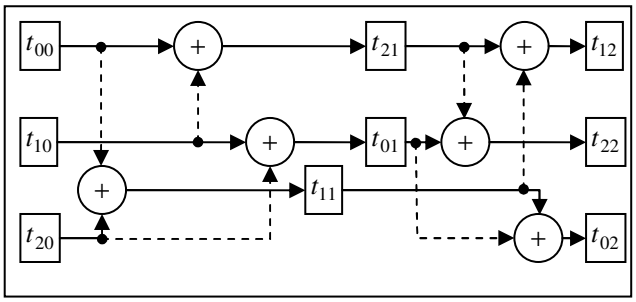


Figure 5.5: Flow diagram for each recursion stage (complete polarity matrix generation).

Steps for generating ternary FPAE spectrum in polarity ω by column polarity matrix algorithm

Step 1: Find $< j_1, j_2, \ldots, j_n >$, the ternary representation of ω ($< \omega >_{10} = < j_1, j_2, \ldots, j_n >_3$).

Step 2: Find $X_n^{<0>}(f(\vec{x}))$ from $\vec{F}$, the truth vector of the input function $f(\vec{x})$, by (5.35)–(5.38). $X_n^{<0>}(f(\vec{x}))$ is B^0 of $f(\vec{x})$. Set $\beta = n$ and $p = 0$.

Step 3: For each $T_{\beta}^{<p,q>}$ inside the polarity matrix, calculate the first columns of its t_{r1} and t_{r2} submatrices from the known first columns of its t_{00}, t_{10}, and t_{20}

submatrices based on (5.34), where $r = j_{n+1-\beta}$. The flow diagram for the computation inside each $T_\beta^{<p,q>}$ matrix is shown in Fig. 5.6, where r_i ($i \in \{0,1,2\}$) represents $(r+i) \bmod 3$.

Step 4: Decrement β by 1. If $\beta = 0$, the required spectral coefficient vector has been obtained and the algorithm is finished. Otherwise, set $p = pr$ (append r at the end of string p) and repeat Steps 3 and 4.

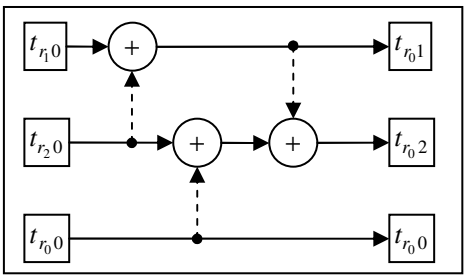


Figure 5.6: Flow diagram for each recursion stage (spectral coefficient vector generation).

Based on the given steps of the algorithm, the computational costs of the column polarity matrix algorithm can be derived.

Property 5.2.17. The generation of the complete polarity matrix of an n-variable ternary function by the algorithm incurs $3^n\left(3^n-1\right)$ subtractions and no multiplications.

Proof: From Fig. 5.5, it can be seen that the computational cost of Step 2 for each $T_\beta^{<p,q>}$ matrix is $6\cdot 3^{\beta-1}$ subtractions. Since there are $9^{n-\beta}$ $T_\beta^{<p,q>}$ matrices inside the polarity matrix, it follows that the total computational cost of Step 2 is $9^{n-\beta}\cdot 6\cdot 3^{\beta-1} = 6\cdot 3^{2n-\beta-1}$ subtractions. Step 2 is performed recursively for β = 1 to n. Therefore, the total computational cost of the algorithm for generating complete polarity matrix is

$$A_9(n) = 6\sum_{\beta=1}^{n} 3^{2n-\beta-1}$$

$$= 6\cdot 3^{n-2}\sum_{\beta=1}^{n} 3^{\beta}$$

$$= 6 \cdot 3^{n-2} \cdot \frac{3\left(1-3^{n}\right)}{1-3}$$

$$= 3^{n}(3^{n}-1) \text{ subtractions.}$$

Property 5.2.18. The computational cost of the algorithm for generating selected spectral coefficient vector of an *n*-variable ternary function is $n \cdot 3^{n}$ subtractions.

Proof: It can be seen from Fig. 5.6 that the computational cost of Step 3 for each $T_{\beta}^{<p,q>}$ matrix is three subtractions, where each is of size $3^{\beta-1}$. Due to the limitation on the value of p imposed in Step 4, when $\beta = i$ there are only 3^{n-i} $T_{\beta}^{<p,q>}$ matrices processed at Step 3. Thus, the total computational cost of Step 3 when $\beta = i$ is $3 \cdot 3^{i-1} \cdot 3^{n-i} = 3^{n}$ subtractions. Step 3 is iterated *n* times, corresponding to the changing value of β from *n* to 1. Therefore, the total computational cost for the algorithm for calculating a spectral coefficient vector is simply $n \cdot 3^{n}$ subtractions.

In Section 5.2, it has been derived that the cost of performing (5.5) by fast transform is $n \cdot 3^{n}$ additions/subtractions and $n \cdot 3^{n-1}$ multiplications. Since both the matrix multiplication and row polarity matrix algorithms require the FPAE spectrum in the starting polarity to be first calculated from the truth vector by (5.5) if it is not known, the computational cost of generating the FPAE spectra in selected polarity by those algorithms are always greater than or equal to $n \cdot 3^{n}$ additions/subtractions and $n \cdot 3^{n-1}$ multiplications. As such, from Property 5.2.18, it can be concluded that the column polarity matrix algorithm has the smallest computational cost for generating a spectral coefficient vector. Table 5.9 lists the computational cost of the column polarity matrix algorithm for $n < 7$ when it is used to generate the complete polarity matrix. From Tables 5.1, 5.8, and 5.9, it can be seen that among them the column polarity matrix algorithm also requires the least number of arithmetic operations for calculating all spectral coefficients.

Table 5.9: Computational costs of column polarity matrix algorithm (ternary FPAEs).

n	Addition/subtraction	Multiplication
1	6	0
2	72	0
3	702	0
4	6480	0
5	58806	0
6	530712	0

5.3 FPAEs for quaternary functions

Let $\vec{F} = [F_0, F_1, ..., F_{4^n-1}]^T$ be the truth vector of a quaternary function $f(\vec{x})$, where $\vec{x} = [x_1, x_2, \ldots, x_n]$, $F_0 = f(0,0,\ldots,0)$, $F_1 = f(0,0,\ldots,1)$, and $F_{4^n-1} = f(3,3,\ldots,3)$. Then C^ω for $f(\vec{x})$ can be calculated from $\vec{F}$ by

$$C^\omega = \frac{1}{2^n} \cdot S_n^{<\omega>} \cdot \vec{F}$$

$$= \frac{1}{2^n} \cdot \left(\otimes \prod_{l=1}^{n} S_1^{<j_l>} \right) \cdot \vec{F} . \tag{5.41}$$

Conversely, $\vec{F}$ can be obtained back from C^ω by

$$F = T_n^{<\omega>} \cdot C^\omega$$

$$= \left(\otimes \prod_{l=1}^{n} T_1^{<j_l>} \right) \cdot C^\omega , \tag{5.42}$$

where $\otimes$ denotes Kronecker product and $<\omega>_{10} = <j_1, j_2, \ldots, j_n>_4$. The basic transform matrices for FPAE of quaternary functions $T_1^{<j_l>}$ and $S_1^{<j_l>}$ are

$$T_1^{<0>}=\begin{bmatrix}1&0&0&0\\1&1&1&1\\1&2&0&0\\1&3&1&3\end{bmatrix} \qquad T_1^{<1>}=\begin{bmatrix}1&1&1&1\\1&2&0&0\\1&3&1&3\\1&0&0&0\end{bmatrix}$$

$$T_1^{<2>}=\begin{bmatrix}1&2&0&0\\1&3&1&3\\1&0&0&0\\1&1&1&1\end{bmatrix} \qquad T_1^{<3>}=\begin{bmatrix}1&3&1&3\\1&0&0&0\\1&1&1&1\\1&2&0&0\end{bmatrix}$$

$$S_1^{<0>}=\begin{bmatrix}2&0&0&0\\-1&0&1&0\\-2&3&0&-1\\1&-1&-1&1\end{bmatrix} \qquad S_1^{<1>}=\begin{bmatrix}0&0&0&2\\0&1&0&-1\\3&0&-1&-2\\-1&-1&1&1\end{bmatrix}$$

$$S_1^{<2>}=\begin{bmatrix}0&0&2&0\\1&0&-1&0\\0&-1&-2&3\\-1&1&1&-1\end{bmatrix} \qquad S_1^{<3>}=\begin{bmatrix}0&2&0&0\\0&-1&0&1\\-1&-2&3&0\\1&1&-1&-1\end{bmatrix}$$

Similar to the case for ternary FPAE, the relations between any two quaternary FPAE coefficient vectors for the same function can be derived from (5.41) and (5.42). Let us consider two coefficient vectors with polarity numbers 0 and ω_1 $(0\le\omega_1\le 4^n-1)$, respectively where $<\omega_1>_{10}=<\omega_{11},\omega_{12},\ldots,\omega_{1n}>_4$. Then, it can be derived that

$$\begin{aligned}C^{\omega_1}&=\frac{1}{2^n}\cdot S_n^{<\omega_1>}\cdot\vec{F}=\frac{1}{2^n}\cdot S_n^{<\omega_1>}\cdot T_n^0\cdot C^0\\&=\frac{1}{2^n}\cdot\left(\otimes\prod_{l=1}^{n}S_1^{<\omega_{1l}>}\right)\cdot\left(\otimes\prod_{l=1}^{n}T_1^{<0>}\right)\cdot C^0\\&=\left(\otimes\prod_{l=1}^{n}\left(\frac{1}{2}\cdot S_1^{<\omega_{1l}>}\cdot T_1^{<0>}\right)\right)\cdot C^0\\&=\left(\otimes\prod_{l=1}^{n}Z_1^{<\omega_{1l}>}\right)\cdot C^0=Z_n^{<\omega_1>}\cdot C^0.\end{aligned}\tag{5.43}$$

The transform matrices $Z_1^{<j_l>}$ for $j_l\in\{0,1,2,3\}$ are

$$Z_1^{<0>}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$Z_1^{<1>} = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 0 & -1 & 0 & -1 \\ 0 & -4 & -1 & -3 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

$$Z_1^{<2>} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$Z_1^{<3>} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & -1 & -1 \\ 0 & -2 & 0 & -1 \end{bmatrix}.$$

Given (5.43), the following equation can now be derived that relates quaternary FPAE coefficient vectors in any two polarities ω_1 and ω_2 $(0 \le \omega_1, \omega_2 \le 4^n - 1)$:

$$\begin{aligned} C^{\omega_1} &= Z_n^{<\omega_1>} \cdot C^0 = Z_n^{<\omega_1>} \cdot \left(Z_n^{<\omega_2>}\right)^{-1} \cdot C^{\omega_2} \\ &= \left(\otimes \prod_{l=1}^{n} \left(Z_1^{<\omega_{1l}>} \cdot \left(Z_1^{<\omega_{2l}>}\right)^{-1} \right) \right) \cdot C^{\omega_2}. \end{aligned}$$

Since $\left(Z_1^{<j_l>}\right)^{-1} = \left(Z_1^{<(3\cdot j_l) \bmod 4>}\right)$, the derivation can be continued as follows:

$$\begin{aligned} C^{\omega_1} &= \left(\otimes \prod_{l=1}^{n} \left(Z_1^{<\omega_{1l}>} \cdot \left(Z_1^{<(3\cdot\omega_{2l}) \bmod 4>}\right) \right) \right) \cdot C^{\omega_2} \\ &= Z_n^{<\omega_1 + 3\omega_2>} \cdot C^b \end{aligned} \tag{5.44}$$

Each basic transform matrix $S_1^{<j_l>}$ has ten nonzero elements and three of them have absolute values greater than one. Due to the property of Kronecker product and (5.41), it follows that there are 10^n nonzero elements inside each $S_n^{<\omega>}$, where $10^n - 7^n$ of them have absolute values greater than 1. Therefore, the computational cost of direct application of (5.41) is

$$A_{10}(n) = 10^n - 4^n \text{ additions/subtractions} \tag{5.45}$$

and

$$M_{10}(n) = 10^n - 7^n \text{ multiplications.} \tag{5.46}$$

whereas the computational cost of fast transform of (5.41) is

$$A_{11}(n) = 6n \cdot 4^{n-1} \text{ additions/subtractions} \tag{5.47}$$

and $$M_{11}(n) = 3n \cdot 4^{n-1} \text{ multiplications.} \tag{5.48}$$

In deriving the computational costs above, the division by 2^n is assumed to be performed by bit shifting operation, and therefore it does not involve any multiplications.

On the other hand, the number of nonzero elements inside the matrices $Z_1^{<j_l>}$ are not the same for different value of j_l. Let $N(y)$ ($y \in \{0,1,2,3\}$) be the total number of y in the quaternary representation of $<\omega_1 + 3\omega_2>$. Then the cost of performing the fast transform of (5.44) are

$$A_{12}(n) = 4^{n-1}\left(7 \cdot N(1) + 3 \cdot N(2) + 7 \cdot N(3)\right) \text{ additions/subtractions} \tag{5.49}$$

and $$M_{12}(n) = 4^{n-1}\left(5 \cdot N(1) + 3 \cdot N(2) + 2 \cdot N(3)\right) \text{ multiplications.} \tag{5.50}$$

The shown relations between the truth vector and quaternary FPAE spectra as well as their computational costs can be used to develop more efficient algorithms to generate selected spectral coefficients, selected FPAE spectra, or complete FPAE polarity matrix. In what follows, the algorithms given in Section 5.2 for ternary FPAE are extended for quaternary FPAE based on (5.41)–(5.44).

5.3.1 Matrix multiplication algorithms for quaternary FPAEs

In matrix multiplication algorithms, all the 4^n quaternary FPAE coefficient vectors are sorted in a particular order, where the first coefficient vector in the order is calculated from the truth vector by (5.41), and each of the other vectors are calculated from the previous coefficient vector in the order by (5.44). Clearly, by (5.49) and (5.50), the resulting total computational cost of such algorithm is dependent on the ordering used. When the coefficient vectors are ordered in lexicographical order based on their polarity numbers, the total numbers of $N(1)$ in all the $4^n - 1$ $Z_n^{<\omega_1+3\omega_2>}$ used in the computation is $\left(4^{n+1} - 3n - 4\right)/3$ whereas $N(2) = N(3) = 0$. Thus, by (5.47)–(5.50), it can be derived that the computational cost of the matrix multiplication algorithm with

fast transform and lexicographical order is

$$A_{13}(n) = 4^{n-1}\left(28\cdot\left(4^n-1\right)-3n\right)/3 \text{ additions/subtractions} \tag{5.51}$$

and

$$M_{13}(n) = 4^{n-1}\left(20\cdot\left(4^n-1\right)-6n\right)/3 \text{ multiplications.} \tag{5.52}$$

Alternatively, reverse lexicographic order can be used where the polarity numbers are ordered from 4^n-1 down to zero. When such ordering is used, the total number of $N(1)$ and $N(2)$ are zero whereas the total number of $N(3)$ is $\left(4^{n+1}-3n-4\right)/3$ and so the computational cost of matrix multiplication algorithm with reverse lexicographic order is

$$A_{14}(n) = 4^{n-1}\left(28\cdot\left(4^n-1\right)-3n\right)/3 \text{ additions/subtractions} \tag{5.53}$$

and

$$M_{14}(n) = 4^{n-1}\left(8\cdot\left(4^n-1\right)+3n\right)/3 \text{ multiplications.} \tag{5.54}$$

On the other hand, when the same ordering shown in the algorithm in Fig. 5.7 is used, the total numbers of $N(1)=0$, $N(2)=\frac{1}{3}\left(4^n+2\right)$, and $N(3)=\frac{1}{3}\left(2\cdot 4^n-5\right)$. So, the total computational cost for the matrix multiplication algorithm is

$$A_{15}(n) = 4^{n-1}\left(17\cdot 4^n+18n-29\right)/3 \text{ additions/subtractions} \tag{5.55}$$

and

$$M_{15}(n) = 4^{n-1}\left(7\cdot 4^n+9n-4\right)/3 \text{ multiplications,} \tag{5.56}$$

which are smaller than the computational cost of both the lexicographic and reverse lexicographic order.

Table 5.10 lists the computational cost values for the three matrix multiplication algorithms for $n \le 7$. From the table, it can be seen that the algorithm in Fig 5.7. is the most efficient among them.

5.3.2 Cube polarity adjustment algorithm for quaternary FPAEs

Definition 5.3.1. Let the literal of a quaternary variable in a cube be denoted by $X_i^{S_i}$,

where $S_i \subseteq \{0,1,2,3)$ is said to be the true set of literal X_i. Then a quaternary cube is simply a product of quaternary literals $X_1^{S_1} X_2^{S_2} \ldots X_i^{S_i}$, $i \le n$.

```
Void qfpae(int truth vector[ ])
{   int W[n] = <0, 0, …., 0>, p[ ];
    char Dir[n] = 'aaa…a';
    Calculate C^0 from truth vector by (5.41) and
       store the result in p[ ];
    For(j = 0 to 4^n – 2)
    { cont = true;
      a = n;
      while (cont==true)
      { a = a––;
        cont = false;
        if(Dir[a]=='a')
        { switch (W[a])
          { case 0: W[a] = 2; break;
            case 1: W[a] = 3; break;
            case 2: W[a] = 1; break;
            case 3: { cont = true; Dir[a] = 'd';}} }
        else
        { switch (W[a])
          { case 3: W[a] = 2; break;
            case 2: W[a] = 1; break;
            case 1: W[a] = 0; break;
            case 0: { cont=true; Dir[a] = 'a';}} }}
      Calculate C^W from p[ ] by (5.44); p[ ] = C^W ;}}
```

Figure 5.7: Matrix multiplication algorithm for quaternary FPAEs.

Table 5.10: Computational cost of matrix multiplication algorithms for quaternary FPAEs.

n	Additions/subtractions			Multiplications		
	Lexicographic order	Reverse lexicographic order	Algorithm in Fig. 5.7	Lexicographic order	Reverse lexicographic order	Algorithm in Fig. 5.7
1	27	27	19	18	9	11
2	552	552	372	384	168	168
3	9360	9360	5936	6624	2736	2512
4	152064	152064	93760	108288	43776	38912
5	2443008	2443008	1490688	1743360	699648	615168
6	39131136	39131136	23794688	27942912	11188224	9803776

Similar to the case for ternary functions, the cube polarity adjustment algorithm for FPAEs of quaternary functions takes an array of disjoint cubes as its input, where the cubes can be represented as integer coded cube or by its positional notation.

Example 5.3.1. Let $X_1^{\{1,2\}}X_2^{\{0,1,3\}}X_3^{\{3\}}$ be a quaternary cube with three input variables. Then the quaternary minterms covered by the cube are $X_1^{\{1\}}X_2^{\{0\}}X_3^{\{3\}}$, $X_1^{\{2\}}X_2^{\{0\}}X_3^{\{3\}}$, $X_1^{\{1\}}X_2^{\{1\}}X_3^{\{3\}}$, $X_1^{\{2\}}X_2^{\{1\}}X_3^{\{3\}}$, $X_1^{\{1\}}X_2^{\{3\}}X_3^{\{3\}}$, and $X_1^{\{2\}}X_2^{\{3\}}X_3^{\{3\}}$ or 103, 203, 113, 213, 133, and 233, respectively. Also, the positional notation of the cube is 0110−1101−0001.

Definition 5.3.2. The canonical FPAE of $f(\vec{x})$ in polarity ω for quaternary function can also be expressed by

$$f(\vec{x}) = \sum_{i=0}^{4^n-1} c_i^{\omega} \pi_i^{\omega}, \tag{5.57}$$

where each piterm π_i^{ω} is equivalent to the product term $\left[\prod_{l=1}^{n} \hat{x}_l^{k_l}\right]$.

Property 5.3.1. There are 4^n different possible π_i^{ω} for a particular polarity number that correspond to the different permutations of k_l.

Property 5.3.2. The *n*-variable quaternary FPAE can be fully described by the set of all spectral coefficients c_i^{ω}, where $c_i^{\omega} \in \{0,1,2,3\}$.

Property 5.3.3. The missing literal in a quaternary cube corresponds to the literal $X_i^{\{0,1,2,3\}}$ as well as '1111' in positional notation.

Definition 5.3.3. The ω-adjustment operator is an operator that takes a polarity digit and a quaternary cube literal as its inputs. The cube can be represented either by integer coded cube or positional notation. Tables 5.11 and 5.12 show the ω-adjustment operation when the cube is an integer coded cube and in positional notation, respectively.

Definition 5.3.4. The π-adjustment operator is defined as an operation between a spectral coefficient index digit and a quaternary cube literal. Similar to the ω-adjustment operator, the cube input for the π-adjustment operation can be in the form of either integer coded cube or positional notation. Tables 5.13 and 5.14 show the outputs of the π-adjustment operation for all possible inputs.

Table 5.11: Cube ω-adjustment operation (quaternary integer coded cube).

ω-adj	Polarity digit			
Cube digit	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2
–	–	–	–	–

Table 5.12: Cube ω-adjustment operation (quaternary positional notation).

ω-*adj*	Polarity digit			
Variable positional notation	0	1	2	3
0001	0001	1000	0100	0010
0010	0010	0001	1000	0100
0011	0011	1001	1100	0110
0100	0100	0010	0001	1000
0101	0101	1010	0101	1010
0110	0110	0011	1001	1100
0111	0111	1011	1101	1110
1000	1000	0100	0010	0001
1001	1001	1100	0110	0011
1010	1010	0101	1010	0101
1011	1011	1101	1110	0111
1100	1100	0110	0011	1001
1101	1101	1110	0111	1011
1110	1110	0111	1011	1101
1111	1111	1111	1111	1111

Table 5.13: Cube π-adjustment operation (quaternary integer coded cube).

π-*adj*	Spectral coefficient index digit			
Cube digit	0	1	2	3
0	2	1	2	1
1	0	0	3	1
2	0	1	0	1
3	0	0	1	1

Table 5.14: Cube π-adjustment operation (quaternary positional notation).

π-adj	Spectral coefficient index digit			
Variable positional notation	0	1	2	3
0001	0	0	1	1
0010	0	1	0	1
0011	0	1	1	0
0100	0	0	3	1
0101	0	0	2	0
0110	0	1	3	2
0111	0	1	2	1
1000	2	1	2	1
1001	2	1	3	2
1010	2	0	2	0
1011	2	0	3	1
1100	2	1	1	0
1101	2	0	0	1
1110	2	0	1	1

Definition 5.3.5. Let h_q^i denotes the contribution of the q-th ON disjoint cube to the value of the i-th spectral coefficient c_i^ω. Then the final value of c_i^ω is a summation of all contributions to it in standard arithmetic algebra. If z is the number of ON disjoint cubes inside the quaternary input function, then

$$c_i^\omega = \sum_{q=1}^{z} h_q^i \,. \tag{5.58}$$

Let the q-th ON disjoint cube of an n-variable quaternary function be given by $q = q_1 q_2 \ldots q_n$. Then the value of h_q^i as described in Definition 5.3.5 can be generated through the following steps:

Step 1: Generate the n-digit quaternary representation of the spectral coefficient index i, $< k_1, k_2, \ldots, k_n >$ and polarity number ω, $< j_1, j_2, \ldots, j_n >$.

Step 2: Perform bitwise operation on q and ω:

$$q' = q_{\omega-adj}\,\omega .$$

Step 3: Set $i' = i$. If the degree of the cube $q' = n$, go to Step 4. Otherwise perform the following:

- If the cube is an integer coded cube:

  For every digit of the cube q'_l $(1 \le l \le n)$, if $q'_l =$'–' then set $q'_l = i'_l$ and $i'_l = 0$.

- If the cube is represented by its positional notation:

  For every literal in the cube q'_l $(1 \le l \le n)$, if $q'_l = 1111$ then set q'_l = 1000, 0100, 0010 or 0001 according to whether i'_l = 0, 1, 2 or 3, respectively. Subsequently set $i'_l = 0$.

Step 4: Perform bitwise operation on q' and i':

$$q'' = q'_{\pi-adj}\, i' .$$

Step 5: Set $M = \prod_{l=1}^{n} q''_l$.

Step 6: Determine the sign of h_q^i by the following:

- If the cube is an integer coded cube:

  Perform bitwise modulo 3 addition, $temp = q' \oplus i'$. If the number of nonzero digit in *temp* is even, $sign = 1$. Else, $sign = -1$.

- If the cube is represented by its positional notation:

  Set *temp* = 0. For every nonzero digit in i', i'_l $(1 \le l \le n)$: if ($i'_l = 1$ and the leftmost digit in the positional notation of the l-th input variable in q' = 1) or ($i'_l = 2$ and the second leftmost digit in the positional notation of the l-th input variable in q' = 0) or ($i'_l = 3$ and both the rightmost and leftmost digits in the positional notation of the l-th input variable in q' = 0) or ($i'_l = 3$ and both the middle digits in the positional notation of

the l-th input variable in $q' = 1$) then set *temp* = *temp* + 1. If the final value of *temp* is even, $sign = 1$. Else, $sign = -1$.

Step 7: Let $f(q)$ be the output value of the disjoint cube q. Then,

$$h_q^i = \frac{1}{2^n} \times sign \times M \times f(q).$$

5.3.3 Row polarity matrix algorithm for quaternary FPAEs

Definition 5.3.6. Let each coefficient vector C^ω be decomposed into four equal size subvectors such that

$$C^\omega = [c^\omega_{[n-1,0]}, c^\omega_{[n-1,1]}, c^\omega_{[n-1,2]}, c^\omega_{[n-1,3]}]. \tag{5.59}$$

Since there are 4^n elements in C^ω, each of the subvectors $c^\omega_{[n-1,q]}$ $(q \in \{0,1,2,3\})$ contains 4^{n-1} elements.

Property 5.3.4. The definition above is recursive and each subvector of C^ω can be further decomposed into four smaller subvectors of equal size. Generally for $1 \le p \le n$,

$$c^\omega_{[p,q]} = [c^\omega_{[p-1,q0]}, c^\omega_{[p-1,q1]}, c^\omega_{[p-1,q2]}, c^\omega_{[p-1,q3]}], \tag{5.60}$$

where q is the $(n-p)$-digit quaternary representation of any decimal number from 0 to $4^{n-p} - 1$.

Property 5.3.5. Let $< q >_{10}$ be the decimal number representation of q in $c^\omega_{[p,q]}$. Then the elements of the subvector $c^\omega_{[p,q]}$ correspond to the polarity matrix $P[f(\vec{x})]$ elements with row index numbers ω and column index numbers j, where $(< q >_{10} \cdot 4^p) \le j < (< q >_{10} + 1) \cdot 4^p$.

Property 5.3.6. Each subvector $c^\omega_{[p,q]}$ has 4^p elements. When $p = n$, $c^\omega_{[p,q]}$ has 4^n

elements, and therefore $c^{\omega}_{[n,0]}$ is simply C^{ω}.

Definition 5.3.7. Let T_p be a $4^p \times 4^p$ matrix that is recursively defined by

$$T_p(c^{\omega}_{[p,q]}) = \left[T_{p0}(c^{\omega}_{[p,q]}), T_{p1}(c^{\omega}_{[p,q]}), T_{p2}(c^{\omega}_{[p,q]}), T_{p3}(c^{\omega}_{[p,q]})\right] \tag{5.61}$$

where

$$T_{p0}\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}\left(c^{\omega}_{[p-1,q0]}\right) \\ T_{p-1}\left(c^{\omega}_{[p-1,q0]} + 3c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]} + 3c^{\omega}_{[p-1,q3]}\right) \\ T_{p-1}\left(c^{\omega}_{[p-1,q0]} + 2c^{\omega}_{[p-1,q1]}\right) \\ T_{p-1}\left(c^{\omega}_{[p-1,q0]} + c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]} + c^{\omega}_{[p-1,q3]}\right) \end{bmatrix},$$

$$T_{p1}\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}\left(c^{\omega}_{[p-1,q1]}\right) \\ -T_{p-1}\left(c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q3]}\right) \\ -T_{p-1}\left(c^{\omega}_{[p-1,q1]}\right) \\ T_{p-1}\left(c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q3]}\right) \end{bmatrix},$$

$$T_{p2}\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}\left(c^{\omega}_{[p-1,q2]}\right) \\ -T_{p-1}\left(4c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]} + 3c^{\omega}_{[p-1,q3]}\right) \\ T_{p-1}\left(2c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q2]} + 4c^{\omega}_{[p-1,q3]}\right) \\ T_{p-1}\left(2c^{\omega}_{[p-1,q1]} - c^{\omega}_{[p-1,q2]} - c^{\omega}_{[p-1,q3]}\right) \end{bmatrix},$$

and

$$T_{p3}\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}\left(c^{\omega}_{[p-1,q3]}\right) \\ T_{p-1}\left(2c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q3]}\right) \\ -T_{p-1}\left(c^{\omega}_{[p-1,q3]}\right) \\ -T_{p-1}\left(2c^{\omega}_{[p-1,q1]} + c^{\omega}_{[p-1,q3]}\right) \end{bmatrix}.$$

Property 5.3.7. By Definition 5.3.7 and Property 5.3.4, the first row of matrix $T_p(c^{\omega}_{[p,q]})$ is simply the vector $c^{\omega}_{[p,q]}$.

Property 5.3.8. By Definition 5.3.7, $T_1(c^{\omega}_{[1,q]})$ only has $4 \times 4 = 16$ elements. Therefore $T_0(c^{\omega}_{[0,s]}) = c^{\omega}_{[0,s]} = c^{\omega}_{s}$.

Property 5.3.9. When $\omega = 0$ and $p = n$, the matrix $T_p(c^{\omega}_{[p,q]})$ yields the FPAE polarity matrix for quaternary functions, $P[f(\vec{x})] = T_n(c^{0}_{[n,0]})$. Furthermore, by

Property 5.3.8, $c^{0}_{[n,0]} = C^{0}$ and $P[f(\vec{x})] = T_n(C^{0})$.

Property 5.3.10. Due to the linear property of the $T_p(c^{\omega}_{[p,q]})$ matrices, the recursive definition of $T_p(c^{\omega}_{[p,q]})$ in Definition 5.3.7 can be rewritten as follows:

$$T_p\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}(t_{00}) & T_{p-1}(t_{01}) & T_{p-1}(t_{02}) & T_{p-1}(t_{03}) \\ T_{p-1}(t_{10}) & T_{p-1}(t_{11}) & T_{p-1}(t_{12}) & T_{p-1}(t_{13}) \\ T_{p-1}(t_{20}) & T_{p-1}(t_{21}) & T_{p-1}(t_{22}) & T_{p-1}(t_{23}) \\ T_{p-1}(t_{30}) & T_{p-1}(t_{31}) & T_{p-1}(t_{32}) & T_{p-1}(t_{33}) \end{bmatrix}$$

$$= \begin{bmatrix} T_{p-1}(t_{00}) & T_{p-1}(t_{01}) & T_{p-1}(t_{02}) & T_{p-1}(t_{03}) \\ T_{p-1}(t_{30}) + T_{p-1}(t_{03}) + T_{p-1}(t_{13}) & -T_{p-1}(t_{01}) - T_{p-1}(t_{03}) & T_{p-1}(t_{00}) - T_{p-1}(t_{10}) - T_{p-1}(t_{01}) & 2T_{p-1}(t_{01}) + T_{p-1}(t_{03}) \\ T_{p-1}(t_{00}) + 2T_{p-1}(t_{01}) & -T_{p-1}(t_{01}) & T_{p-1}(t_{03}) + 2T_{p-1}(t_{01}) - T_{p-1}(t_{12}) & -T_{p-1}(t_{03}) \\ T_{p-1}(t_{00}) + T_{p-1}(t_{01}) + T_{p-1}(t_{02}) + T_{p-1}(t_{03}) & -T_{p-1}(t_{11}) & 2T_{p-1}(t_{01}) - T_{p-1}(t_{02}) - T_{p-1}(t_{03}) & -T_{p-1}(t_{13}) \end{bmatrix}. \quad (5.62)$$

Definition 5.3.8. Let $R_{1[p-1]}$, $R_{2[p-1]}$, and $R_{3[p-1]}$ be three vectors of length 4^{p-1} that are defined as $R_{1[p-1]} = T_{p-1}(t_{01}) + T_{p-1}(t_{03})$, $R_{2[p-1]} = T_{p-1}(t_{00}) + T_{p-1}(t_{02})$, and $R_{3[p-1]} = 2T_{p-1}(t_{01}) - T_{p-1}(t_{03})$, respectively. Then (5.62) can be further simplified into

$$T_p\left(c^{\omega}_{[p,q]}\right) = \begin{bmatrix} T_{p-1}(t_{00}) & T_{p-1}(t_{01}) & T_{p-1}(t_{02}) & T_{p-1}(t_{03}) \\ T_{p-1}(t_{30}) + T_{p-1}(t_{03}) + T_{p-1}(t_{13}) & -R_{1[p-1]} & T_{p-1}(t_{00}) - T_{p-1}(t_{10}) - T_{p-1}(t_{01}) & 2T_{p-1}(t_{01}) + T_{p-1}(t_{03}) \\ T_{p-1}(t_{00}) + 2T_{p-1}(t_{01}) & -T_{p-1}(t_{01}) & -T_{p-1}(t_{12}) - R_{3[p-1]} & -T_{p-1}(t_{03}) \\ R_{1[p-1]} + R_{2[p-1]} & -T_{p-1}(t_{11}) & R_{3[p-1]} - T_{p-1}(t_{02}) & -T_{p-1}(t_{13}) \end{bmatrix}. \quad (5.63)$$

In order to generate the complete polarity matrix, the row polarity matrix algorithm requires C^{0}, the coefficient vector of the input function in polarity zero, to be known. Therefore, if the input function is given in the form of truth vector, the C^{0}

need to be first generated from the truth vector by (5.41). Once the C^0 is obtained, the algorithm then generates the remaining elements of the polarity matrix by performing the following steps:

Step 1: Let $P[f(\vec{x})]$ be $T_n(c^0_{[n,0]})$ with C^0 as the first row of the matrix. Set $p = n$.

Step 2: Divide $T_n(c^0_{[n,0]})$ into $4^{n-p} \times 4^{n-p}$ T_p matrices. For each of the T_p matrices, identify its $T_{p-1}(t_{gh})$ submatrices ($0 \le g, h \le 3$) according to (5.62).

Step 3: For each T_p matrices whose t_{00}, t_{01}, t_{02}, and t_{03} are known, calculate the unknown first row elements of its T_{p-1} matrices, i.e., t_{gh} ($1 \le g \le 3$, $0 \le h \le 3$) except t_{11}, t_{21}, t_{23}, and t_{33} from t_{00}, t_{01}, t_{02}, and t_{03} according to the flow diagram shown in Fig. 5.8. In the figure, the dashed line ---- indicates that the sign of the signal should be inverted. The flow diagram is developed based on (5.63) as it requires smaller number of arithmetic operations compared to (5.61).

Step 4: Decrement p by 1. If $p = 0$, go to Step 5. Otherwise go back to Step 2.

Step 5: Complete the polarity matrix by copying all $T_p(t_{31})$, $T_p(t_{01})$, $T_p(t_{03})$, and $T_p(t_{13})$ matrices to corresponding $T_p(t_{11})$, $T_p(t_{21})$, $T_p(t_{23})$, and $T_p(t_{33})$ matrices with the signs reversed, where p changes from 0 to $n-1$.

Alternatively, the algorithm can also be used to generate the elements of a selected coefficient vector only. When this is the case, the algorithm calculates the required spectral coefficients based on the recursive definition given in (5.61) rather than (5.63) so that they can be obtained without first deriving the complete polarity matrix. The steps of the algorithm for generating coefficient vector in polarity ω (recall that $<\omega>_{10} = < j_1, j_2, \ldots, j_n >_4$) from C^0 are as follows:

Step 1: Let $P[f(\vec{x})]$ be $T_n(c^0_{[n,0]})$ with C^0 as the first row of the matrix. Set $p = 1$.

Step 2: Divide $T_n(c^0_{[n,0]})$ into $4^{p-1} \times 4^{p-1}$ T_{n+1-p} matrices. For each of the T_{n+1-p} matrices whose first row elements are known, identify its $T_{p-1}(t_{gh})$ submatrices ($0 \le g, h \le 3$) according to (5.62).

Step 3: Let $r = j_p$. If $r = 0$, go to Step 4. Otherwise, for each of the T_{n+1-p} matrices whose first row elements are generated in the previous recursion stage, calculate its t_{r0}, t_{r1}, t_{r2}, and t_{r3} by the flow diagrams in Figs. 5.9–5.11, according to whether r = 1, 2, or 3, respectively. As in Fig. 5.8, the dashed line ---- in the figures indicates that the sign of the signal should be inverted.

Step 4: Increment p by 1. If $p = n + 1$, the required spectral coefficient vector has been obtained and the algorithm is finished. Otherwise go back to Step 2.

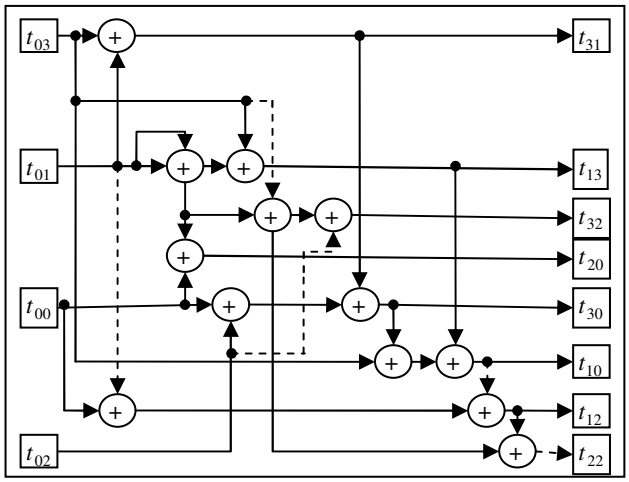


Figure 5.8: Flow diagram for each recursion stage (complete polarity matrix generation).

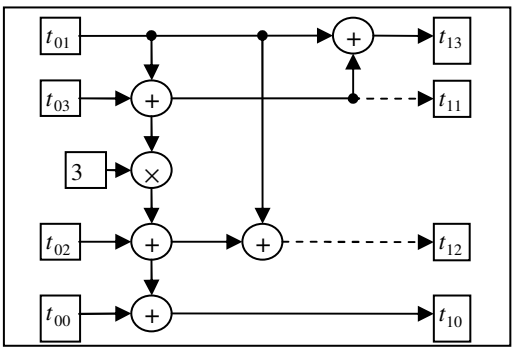


Figure 5.9: Flow diagram in a recursion stage of coefficient vector generation when $r = 1$.

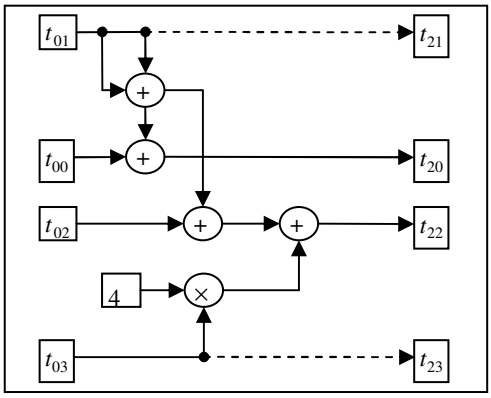


Figure 5.10: Flow diagram in a recursion stage of coefficient vector generation when $r = 2$.

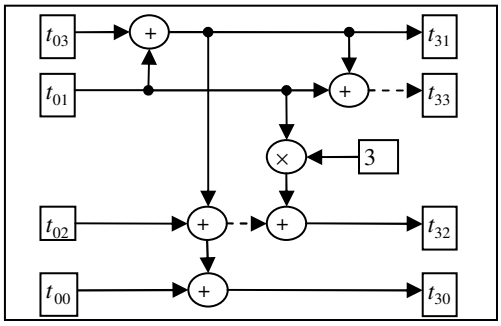


Figure 5.11: Flow diagram in a recursion stage of coefficient vector generation when $r = 3$.

Based on the steps of the algorithm given above, the following properties on the computational cost of the row polarity matrix algorithm can be derived.

Property 5.3.11. For an n-variable quaternary function, the computational cost of generating the complete quaternary FPAE polarity matrix from the coefficient vector in polarity zero using the algorithm is $13 \cdot (12^n - 4^n)/8$ additions/subtractions.

Property 5.3.12. The computational cost of generating coefficient vector in polarity ω of an n-variable quaternary function by the new algorithm is $(5\alpha + 4\beta) \cdot 4^{n-1}$ additions/subtractions and $(\alpha + \beta) \cdot 4^{n-1}$ multiplications, where α is the number of 1s

and 3s in the quaternary number representation of ω while β is the number of 2s in the quaternary number representation of ω.

Property 5.3.13. In the process of generating coefficient vector in polarity ω for an n-variable quaternary function, the new algorithm also derives $j-1$ ($0 \le j \le n-1$) other coefficient vectors for the function (excluding C^{ω}), where j is the total number of nonzero digits in the quaternary representation of ω.

The computational costs of deriving the complete polarity matrix by the row polarity matrix algorithm and by fast transform of (5.41) from truth vector when $n < 7$ are listed in Table 5.15, where the cost of obtaining C^0 from truth vector by fast transform of (5.41) has been added to the cost of the algorithm given in Property 5.3.11. From Tables 5.10 and 5.15, it can be seen that the row polarity matrix algorithm is more efficient than the matrix multiplication algorithms.

Table 5.15: Computational costs of row polarity matrix algorithm (quaternary FPAEs).

n	Additions/subtractions	Multiplications
1	19	3
2	256	24
3	2992	144
4	34816	768
5	410368	3840
6	4882432	18432

5.3.4 Column polarity matrix algorithm for quaternary FPAEs

Definition 5.3.9. Let $T_{\beta}^{<p,q>}(\vec{g})$ be a recursive matrix defined by (5.64), where $\vec{g}$ is a

column vector of size 4^n, $T_d^{<p,q>}(\vec{g})$ ($d \in \{\beta, \beta-1\}$) is a square matrix of size $4^d \times 4^d$, p and q are quaternary numbers, and $1 \le \beta \le n$.

$$T_\beta^{<p,q>}(\vec{g}) = \begin{bmatrix} T_{\beta-1}^{<p0,q0>}(\vec{g}) & T_{\beta-1}^{<p0,q1>}(\vec{g}) & T_{\beta-1}^{<p0,q2>}(\vec{g}) & T_{\beta-1}^{<p0,q3>}(\vec{g}) \\ T_{\beta-1}^{<p1,q0>}(\vec{g}) & T_{\beta-1}^{<p1,q1>}(\vec{g}) & T_{\beta-1}^{<p1,q2>}(\vec{g}) & T_{\beta-1}^{<p1,q3>}(\vec{g}) \\ T_{\beta-1}^{<p2,q0>}(\vec{g}) & T_{\beta-1}^{<p2,q1>}(\vec{g}) & T_{\beta-1}^{<p2,q2>}(\vec{g}) & T_{\beta-1}^{<p2,q3>}(\vec{g}) \\ T_{\beta-1}^{<p3,q0>}(\vec{g}) & T_{\beta-1}^{<p3,q1>}(\vec{g}) & T_{\beta-1}^{<p3,q2>}(\vec{g}) & T_{\beta-1}^{<p3,q3>}(\vec{g}) \end{bmatrix}$$

$$= \begin{bmatrix} t_{00} & t_{01} & t_{02} & t_{03} \\ t_{10} & t_{11} & t_{12} & t_{13} \\ t_{20} & t_{21} & t_{22} & t_{23} \\ t_{30} & t_{31} & t_{32} & t_{33} \end{bmatrix}, \tag{5.64}$$

where $\begin{bmatrix} t_{01} \\ t_{11} \\ t_{21} \\ t_{31} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -t_{00}+t_{20} \\ -t_{10}+t_{30} \\ t_{00}-t_{20} \\ t_{10}-t_{30} \end{bmatrix}$, $\begin{bmatrix} t_{02} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -2t_{00}-t_{10}+3t_{30} \\ 3t_{00}-2t_{10}-t_{20} \\ 3t_{10}-2t_{20}-t_{30} \\ -t_{00}+3t_{20}-2t_{30} \end{bmatrix}$, and

$\begin{bmatrix} t_{03} \\ t_{13} \\ t_{23} \\ t_{33} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} t_{00}+t_{10}-t_{20}-t_{30} \\ -t_{00}+t_{10}+t_{20}-t_{30} \\ -t_{00}-t_{10}+t_{20}+t_{30} \\ t_{00}-t_{10}-t_{20}+t_{30} \end{bmatrix}$.

Definition 5.3.10. Let $\vec{F}$ be the truth vector of an n-variable quaternary function $f(\vec{x})$. Then $Y_\beta^{<v>}(f(\vec{x}))$ is defined as a subvector of $\vec{F}$ with 4^β elements that satisfies (5.65) and (5.66).

$$Y_n^{<0>}(f(\vec{x})) = \vec{F} \tag{5.65}$$

$$Y_\beta^{<v>}(f(\vec{x})) = [Y_{\beta-1}^{<v,0>}(f(\vec{x})), Y_{\beta-1}^{<v,1>}(f(\vec{x})), Y_{\beta-1}^{<v,2>}(f(\vec{x})), Y_{\beta-1}^{<v,3>}(f(\vec{x}))], \tag{5.66}$$

where $1 \le \beta \le n$ and v is an $(n-\beta+1)$-digits quaternary number.

Definition 5.3.11. Let $Y_\beta^{<v>}(f(\vec{x}))$ be as in Definition 5.3.10. Then $X_\beta^{<v>}(f(\vec{x}))$ is defined as a subvector that is recursively built from $Y_\beta^{<v>}(f(\vec{x}))$ by

$$X_\beta^{<v>}(f(\vec{x})) = \begin{cases} [Y_{\beta-1}^{<v,0>}(f(\vec{x})), Y_{\beta-1}^{<v,3>}(f(\vec{x})), Y_{\beta-1}^{<v,2>}(f(\vec{x})), Y_{\beta-1}^{<v,1>}(f(\vec{x}))], & \text{if } \beta = 1 \\ [X_{\beta-1}^{<v,0>}(f(\vec{x})), X_{\beta-1}^{<v,3>}(f(\vec{x})), X_{\beta-1}^{<v,2>}(f(\vec{x})), X_{\beta-1}^{<v,1>}(f(\vec{x}))] & \text{if } 2 \le \beta \le n. \end{cases} \tag{5.67}$$

Therefore, $X_n^{<0>}(f(\vec{x}))$ is simply $\vec{F}$ that has been reordered recursively.

Property 5.3.14. The FPAE polarity matrix of an *n*-variable quaternary function $f(\vec{x})$ is simply $T_n^{<0,0>}(\vec{g})$ when $\vec{g} = X_n^{<0>}(f(\vec{x}))$,

$$P(f(\vec{x})) = T_n^{<0,0>}\left(X_n^{<0>}(f(\vec{x}))\right). \tag{5.68}$$

Definition 5.3.12. The recursive matrix $T_\beta^{<p,q>}(\vec{g})$ in Definition 5.3.9 can be rewritten into

$$T_\beta^{<p,q>}(\vec{g}) = \begin{bmatrix} t_{00} & \frac{1}{2}(t_{20}-t_{00}) & -t_{00}+t_{30}+t_{11} & -t_{01}-t_{11} \\ t_{10} & \frac{1}{2}(t_{30}-t_{10}) & t_{00}-t_{10}-t_{01} & t_{01}-t_{11} \\ t_{20} & -t_{01} & t_{10}-t_{11}-t_{20} & -t_{03} \\ t_{30} & -t_{11} & t_{01}+t_{20}-t_{30} & -t_{13} \end{bmatrix}. \tag{5.69}$$

The complete polarity matrix for a given input function can be generated by (5.69). The computation starts by generation of $X_n^{<0>}(f(\vec{x}))$ (first column elements of $P[f(\vec{x})]$) from the truth vector and continues with the recursive generation of the rest of the $P[f(\vec{x})]$ elements in *n* recursion stages, where at the *i*-th recursion ($1 \le i \le n$), $P[f(\vec{x})]$ is decomposed into $T_{n+1-i}^{<p,q>}(\vec{g})$ matrices. At the *i*-th recursion stage, the first column elements of each $T_{n+1-i}^{<p,q>}(\vec{g})$ matrix are already known and therefore the first column elements of all $T_{n-i}^{<p,q>}(\vec{g})$ matrices are derived at the stage based on (5.69). Fig. 5.12 shows the flow diagram for the generation. Note that in the figure, the dashed line ---- indicates that the sign of the signal should be inverted before being added. In summary, the steps to construct the complete polarity matrix for an *n*-variable quaternary function $f(\vec{x})$ from its truth vector by this algorithm are:

Step 1: Obtain the first column of the polarity matrix $X_n^{<0>}(f(\vec{x}))$ from truth vector according to Definitions 5.3.10 and 5.3.11.

Step 2: Recursively calculate t_{01}, t_{11}, t_{02}, t_{12}, t_{22}, t_{32}, t_{03}, and t_{13} by (5.69).

Step 3: Complete the whole polarity matrix by recursively copying t_{01}, t_{03}, t_{11}, and t_{13} to t_{21}, t_{23}, t_{31}, and t_{33}, respectively with their signs reversed.

Besides generating the complete polarity matrix, the given recursive equations can also be used to generate selected coefficient vectors without first deriving the complete polarity matrix. The computation process for this case is similar to the one described above for generating the complete polarity matrix. However, now in each recursion stage only those $T_{n-i}^{<p,q>}(\vec{g})$ submatrices that contain the spectral coefficients to be generated need to be derived. Fig. 5.13 shows the flow diagram to calculate row r ($r \in \{0,1,2,3\}$) matrices in a recursion stage. In the figure, r_i ($i \in \{0,1,2,3\}$) represents $(r+i) \bmod 4$.

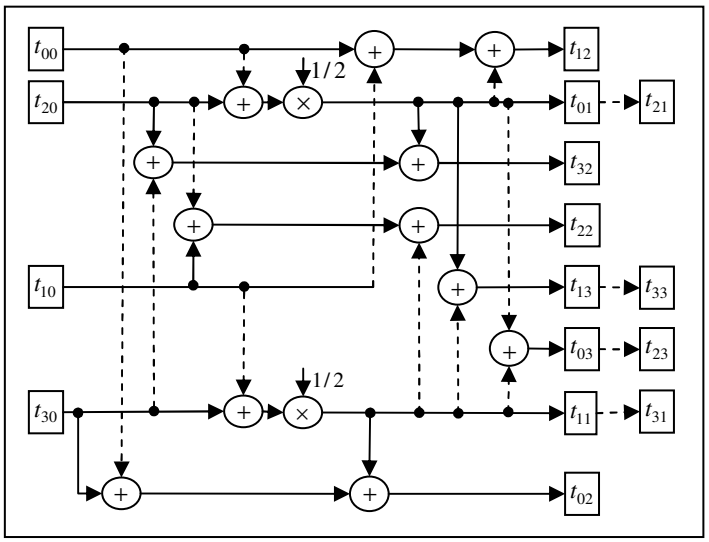


Figure 5.12: Flow diagram for each recursion stage (complete polarity matrix generation).

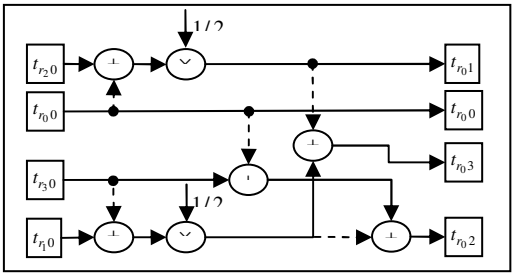


Figure 5.13: Flow diagram for each recursion stage (coefficient vector generation).

Due to the recursive nature of the computation, the computational cost for generating both complete and partial polarity matrix by the column polarity matrix algorithm can be easily deduced. They are given in Properties 5.3.15 and 5.3.16. Note that similar to the case for ternary function, the column polarity matrix algorithm has smaller computational cost than both matrix multiplication and row polarity matrix algorithms for generating selected FPAE spectrum.

Property 5.3.15. The total number of additions/subtractions and multiplications performed by the algorithm in order to generate the complete polarity matrix of size $4^n \times 4^n$ is $\frac{3}{2}\left(12^n - 4^n\right)$ and $\frac{1}{4}\left(12^n - 4^n\right)$, respectively.

Property 5.3.16. The computational cost of the algorithm for generating one selected spectral coefficient vector of an n-variable quaternary function is $5n \cdot 4^{n-1}$ addition/subtractions and $2n \cdot 4^{n-1}$ multiplications.

The computational cost values of deriving the complete polarity matrix by the column polarity matrix algorithm when $n < 7$ are listed in Table 5.16. Comparing the numbers in the table with those in Table 5.15, it can be observed that the row polarity matrix algorithm has smaller number of multiplications whereas the column polarity matrix algorithm has smaller number of additions.

Table 5.16: Computational costs of column polarity matrix algorithm (quaternary FPAEs).

n	Additions/subtractions	Multiplications
1	12	2
2	192	32
3	2496	416
4	30720	5120
5	371712	61952
6	4472832	745472

5.3.5 Experimental results for quaternary FPAEs

The computations of all FPAEs for quaternary functions have been implemented as C++ programs and run for several quaternary test files. The quaternary test files are the MCNC binary benchmarks that had been modified to represent quaternary functions instead of the original binary benchmark functions. The translation from binary to quaternary cases has been done by changing every two input (output) bits in binary files to an input (output) symbol in multiple-valued files. If the number of input and/or output variables is odd, then a zero bit is added behind the binary cubes to make it even. For both input and output, −− is taken as −, 00 is taken as 0, 01 is taken as 1, 10 is taken as 2, and 11 is taken as 3. With these conversions, the binary benchmark files become an array of quaternary cubes.

The number of nonzero spectral coefficients inside the optimal FPAE has been recorded for each of the test files and they are listed in Tables 5.17. For comparison purpose, the number of nonzero spectral coefficients inside the optimal FPRME over GF(4) are also given in the tables. It can be seen that the number of nonzero spectral coefficients inside the optimal FPAE are smaller than that inside the optimal FPRME over GF(4) for many of the test files.

Table 5.17: Number of nonzero terms in optimal FPRME over GF(4) and quaternary FPAE .

Input file	FPRME over GF(4)	Quaternary FPAE
9sym	302	370
ex1010	1017	1016
ex5	200	187
inc	129	71
rd84	112	202
squar5	42	27
xor5	11	18
z5xp1	143	60
z9sym	302	370

Chapter 6

New Fastest Linearly Independent Transforms for Ternary Functions

In this chapter the fastest LI transforms for binary functions presented in Chapter 4 are extended for ternary functions. Similar to the binary fastest LI transforms, the fastest LI transforms for ternary functions are also classified into two transforms: fastest ternary LI (TLI) transforms, which operate over GF(3), and fastest linearly independent ternary arithmetic (LITA) transforms that operate in normal standard arithmetic algebra. These transforms have lower computational costs than FPRME over GF(3) and FPAE for ternary functions, respectively, in terms of the number of required additions/subtractions. The existing fastest TLI and LITA transforms are discussed. Basic definitions for the new transforms are then given. Several properties for the transforms are also listed and their experimental results shown.

6.1 Basic definitions and operators for fastest LI transforms of ternary functions

Definition 6.1.1. Let M_n be an $N \times N$ ($N = 3^n$) matrix with rows corresponding to minterms and columns corresponding to some ternary switching functions of n

variables. If the sets of columns are linearly independent over GF(3), then M_n has only one inverse in GF(3) and is said to be linearly independent.

Definition 6.1.2. The operator R_0 on a column vector is defined as reversing the order of the elements in the vector. If $\vec{b}$ represents a column vector with t elements $\vec{b} = [b_t, b_{t-1}, ..., b_2, b_1]^T$, then $R_0(\vec{b}) = [b_1, b_2, ..., b_{t-1}, b_t]^T$.

Definition 6.1.3. The operator $R_{1,r}$ on an $N \times N$ matrix M_n is defined as performing 9^{n-r-1} counterclockwise rotations twice involving $8 \cdot 9^{n-r-1}$ submatrices each of order 3^r $(0 \le r \le n-1)$.

Example 6.1.1. Let M_2 be a transform matrix of size 9×9, where

$$M_2 = \begin{bmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&1&0&1 \end{bmatrix}.$$

Then by Definition 6.1.3,

$$(R_{1,0}(M_2)) = \left[\begin{array}{c|c|c} 0\,0\,1 & 0\,0\,0 & 0\,0\,0 \\ \hline 0\,1\,0 & 0\,0\,0 & 0\,0\,0 \\ \hline 1\,0\,1 & 0\,0\,0 & 0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,1 & 0\,0\,0 \\ \hline 0\,0\,0 & 0\,1\,0 & 0\,0\,0 \\ \hline 0\,0\,0 & 1\,0\,0 & 0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0 & 0\,0\,1 \\ \hline 0\,0\,0 & 0\,0\,0 & 0\,1\,0 \\ \hline 0\,0\,1 & 0\,0\,0 & 1\,0\,1 \end{array}\right]$$

and

$$(R_{1,1}(M_2)) = \left[\begin{array}{c|c|c} 0\,0\,0 & 0\,0\,0 & 1\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,0 & 0\,1\,0 \\ \hline 0\,0\,0 & 0\,0\,0 & 1\,0\,1 \\ \hline 0\,0\,0 & 1\,0\,0 & 0\,0\,0 \\ \hline 0\,0\,0 & 0\,1\,0 & 0\,0\,0 \\ \hline 0\,0\,0 & 0\,0\,1 & 0\,0\,0 \\ \hline 1\,0\,0 & 0\,0\,0 & 0\,0\,0 \\ \hline 0\,1\,0 & 0\,0\,0 & 0\,0\,0 \\ \hline 1\,0\,1 & 0\,0\,0 & 1\,0\,0 \end{array}\right].$$

Definition 6.1.4. The operator R_2 on an $N \times N$ matrix M_n is defined as recursively applying operator $R_{1,r}$ on M_n for $r = n - 1, n - 2, \ldots, 1, 0$. The square of operator R_2 is specified as $R_2{}^2(M_n) = R_2(R_2(M_n))$.

Definition 6.1.5. Let $X = < x_n, x_{n-1}, ..., x_1 >$ be an n-digit ternary number with x_n as the most significant bit (MSB). Then $L_b(X)$ is defined as the subscript value of the rightmost bit in X whose value is equal to 1.

Definition 6.1.6. Let X be an n-digit ternary number as in Definition 6.1.5. Then $L_1(X)$ is defined as the number of digit in X whose value is equal to 1.

Definition 6.1.7. Let $X = < x_n, x_{n-1}, ..., x_1 >$ be an n-digit ternary number. Then $M_1(X)$ is defined as the subscript value of the leftmost digit in X whose value is equal to 1.

Definition 6.1.8. $L_{b2}(X)$ is defined as the subscript value of the rightmost digit in X whose value is not equal to 0, where $X = < x_n, x_{n-1}, ..., x_1 >$ is an n-digit ternary number.

Example 6.1.2. Let X_1 and X_2 be two five-digit ternary numbers, where X_1 = <0, 1, 2, 1, 0> and X_2 = <0, 0, 2, 0, 2>. Then according to Definitions 6.1.5–6.1.8, $L_b(X_1) = 2$, $L_b(X_2) = 6$, $L_1(X_1) = 2$, $L_1(X_2) = 0$, $M_1(X_1) = 4$, $M_1(X_2) = 6$, $L_{b2}(X_1) = 2$, and $L_{b2}(X_2) = 1$.

Permutation matrices are matrices that contain exactly one '1' in each row and column. As such there are six possible permutation matrices of size 3×3 and 6^n permutation matrices of size $3^n \times 3^n$ that can be derived from the Kronecker product of the 3×3 permutation matrices.

Definition 6.1.9. Let the six 3×3 permutation matrices be referred to as elementary

permutation matrices and be denoted by ρ_0, ρ_1, ρ_2, ρ_3, ρ_4, and ρ_5. Then P_n^p is defined as the permutation matrix of size $3^n \times 3^n$ with permutation number p $\left(0 \le p \le 6^n - 1\right)$ that is calculated by

$$P_n^p = \overset{1}{\underset{j=n}{\otimes}} \rho_{p_j}, \tag{6.1}$$

where $< p_n, p_{n-1}, \ldots, p_1 >$ is the n-digit six-valued representation of p, $\rho_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\rho_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, $\rho_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\rho_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$, $\rho_4 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, $\rho_5 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, and $\otimes$ denotes Kronecker product.

Clearly, the inverse of P_n^p, denoted by $\left(P_n^p\right)^{-1}$, is simply

$$\left(P_n^p\right)^{-1} = \overset{1}{\underset{j=n}{\otimes}} \left(\rho_{p_j}\right)^{-1}, \tag{6.2}$$

where ρ_0, ρ_1, ρ_2, and ρ_5 are self inverse and ρ_3 and ρ_4 are inverses of each other.

Property 6.1.1. From Definition 6.1.9, it can be derived that

$$\left(P_n^p\right)^{-1} = (P_n^p)^T. \tag{6.3}$$

6.2 Fastest TLI transforms

In this section, new fastest TLI transforms that are generated from the existing fastest TLI transforms [FF05c] are introduced. The existing fastest TLI transforms are reviewed and their extensions to obtain new fastest TLI transforms by permutation are shown. Properties on the conversion between different fastest TLI transforms and structure of the transforms as well as their experimental results are also established.

6.2.1 Basic definitions for fastest TLI transforms

Definition 6.2.1. Let M_n be an LI matrix of order $N = 3^n$ as specified in Definition 6.1.1 and $\vec{F} = [f_0, f_1, \ldots, f_{N-1}]^T$ be the truth column vector of an n-variable ternary switching function $f(\vec{x})$ in a natural ternary ordering. Then,

$$\vec{A} = M_n^{-1}\vec{F} \tag{6.4}$$

and

$$\vec{F} = M_n\vec{A}, \tag{6.5}$$

where $\vec{A} = [a_0, a_1, \ldots, a_{N-1}]^T$ is the spectrum of $f(\vec{x})$ based on M_n, M_n^{-1} is the inverse of M_n over GF(3), T denotes transpose operator, and all the additions and multiplications are performed over GF(3).

Definition 6.2.2. Let $f(\vec{x})$ be an n-variable ternary switching function. Then the LI expansion of $f(\vec{x})$ based on a ternary LI transform M_n is

$$f(\vec{x}) = \sum_{j=0}^{3^n-1} a_j g_j, \tag{6.6}$$

where g_j $(0 \le j \le 3^n - 1)$ denotes the ternary switching function whose truth vector is given by column j of M_n, a_j denotes the j-th spectral coefficient in the spectrum of $f(\vec{x})$ obtained by $M_n\vec{A}$, and all operations are performed over GF(3).

Groups of ternary LI transforms with lower computational cost than FPRME transform over GF(3) have been presented in [FF05c] and are classified into classes Y1, Y2, Z1, and Z2. There are six TLI transforms in each class, where four of them have the same computational cost, which is lower than the computational cost of the other two transforms in the class. Here, those fastest TLI transforms with the lowest computational cost are referred as existing ternary fastest LI transforms.

Definition 6.2.3. All the existing ternary fastest LI transforms presented in [FF05c] have the following general form:

$$M_n = \begin{bmatrix} M_{n-1}^{(1)} & O_{n-1} & M_{n-1}^{(2)} \\ O_{n-1} & M_{n-1}^{(5)} & O_{n-1} \\ M_{n-1}^{(3)} & O_{n-1} & M_{n-1}^{(4)} \end{bmatrix}, \tag{6.7}$$

where each submatrix $M_{n-1}^{(j)}$, $j = \{1,2,3,4,5\}$, has a dimension of $3^{n-1} \times 3^{n-1}$ and contains one recursive equation which is either $O_{n-1}, X_{n-1}, Y_{n-1}$ or M_{n-1}, where O_{n-1} is a $3^{n-1} \times 3^{n-1}$ submatrix with all its elements being zero, $X_{n-1} = I_{n-1}$ or J_{n-1}, and Y_{n-1} is a $3^{n-1} \times 3^{n-1}$ submatrix with all its elements being zero except for one element located at one corner of each matrix, depending on the location of Y_{n-1}. I_{n-1} is a $3^{n-1} \times 3^{n-1}$ identity submatrix whereas J_{n-1} is a reverse identity matrix of dimension $3^{n-1} \times 3^{n-1}$. Table 6.1 lists the recursive equations for the existing fastest TLI transforms that belong to class Y1 and Y2.

Definition 6.2.4. The rotation operator *Rot* on the square recursive matrix M_n is recursively defined as clockwise rotation on all $3^{n-1} \times 3^{n-1}$ submatrices of the matrix.

Definition 6.2.5. The inverse rotation operator Rot^{-1} on the square recursive matrix M_n is recursively defined as counterclockwise rotation on all $3^{n-1} \times 3^{n-1}$ submatrices of the matrix.

Definition 6.2.6. The class Z1 and Z2 matrices are derived from class Y1 and Y2 matrices by the following equations:

$$M_{(Z1.x)n} = Rot\left(M_{(Y1.x)n}\right) \tag{6.8}$$

$$M_{(Z2.x)n} = Rot^{-1}\left(M_{(Y2.x)n}\right) \tag{6.9}$$

$$M^{-1}_{(Z1.x)n} = Rot^{-1}\left(M^{-1}_{(Y1.x)n}\right) \tag{6.10}$$

$$M^{-1}_{(Z2.x)n} = Rot\left(M^{-1}_{(Y2.x)n}\right). \tag{6.11}$$

Table 6.1: Class Y1 and Y2 matrices.

Matrix name	M_n	M_n^{-1}
Y1.1	$\begin{bmatrix} M_{n-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ Y_{n-1} & O_{n-1} & M_{n-1} \end{bmatrix}$	$\begin{bmatrix} M_{n-1}^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ 2Y_{n-1} & O_{n-1} & M_{n-1}^{-1} \end{bmatrix}$
Y1.2	$\begin{bmatrix} M_{n-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ Y_{n-1} & O_{n-1} & M_{n-1} \end{bmatrix}$	$\begin{bmatrix} M_{n-1}^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ 2Y_{n-1} & O_{n-1} & M_{n-1}^{-1} \end{bmatrix}$
Y1.4	$\begin{bmatrix} M_{n-1} & O_{n-1} & Y_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & M_{n-1} \end{bmatrix}$	$\begin{bmatrix} M_{n-1}^{-1} & O_{n-1} & 2Y_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & M_{n-1}^{-1} \end{bmatrix}$
Y1.5	$\begin{bmatrix} M_{n-1} & O_{n-1} & Y_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & M_{n-1} \end{bmatrix}$	$\begin{bmatrix} M_{n-1}^{-1} & O_{n-1} & 2Y_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & M_{n-1}^{-1} \end{bmatrix}$
Y2.1	$\begin{bmatrix} Y_{n-1} & O_{n-1} & M_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ M_{n-1} & O_{n-1} & O_{n-1} \end{bmatrix}$	$\begin{bmatrix} O_{n-1} & O_{n-1} & M_{n-1}^{-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ M_{n-1}^{-1} & O_{n-1} & 2Y_{n-1} \end{bmatrix}$
Y2.2	$\begin{bmatrix} Y_{n-1} & O_{n-1} & M_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ M_{n-1} & O_{n-1} & O_{n-1} \end{bmatrix}$	$\begin{bmatrix} O_{n-1} & O_{n-1} & M_{n-1}^{-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ M_{n-1}^{-1} & O_{n-1} & 2Y_{n-1} \end{bmatrix}$
Y2.4	$\begin{bmatrix} O_{n-1} & O_{n-1} & M_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ M_{n-1} & O_{n-1} & Y_{n-1} \end{bmatrix}$	$\begin{bmatrix} 2Y_{n-1} & O_{n-1} & M_{n-1}^{-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ M_{n-1}^{-1} & O_{n-1} & O_{n-1} \end{bmatrix}$
Y2.5	$\begin{bmatrix} O_{n-1} & O_{n-1} & M_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ M_{n-1} & O_{n-1} & Y_{n-1} \end{bmatrix}$	$\begin{bmatrix} 2Y_{n-1} & O_{n-1} & M_{n-1}^{-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ M_{n-1}^{-1} & O_{n-1} & O_{n-1} \end{bmatrix}$

Example 6.2.1. Let us derive the fastest TLI transform matrices $M_{(Y1.1)2}$ and $M_{(Z1.1)2}$ as well as their inverses using Table 6.1 and Definition 6.2.6. Then, we can obtain that

$$M_{(Y1.1)2}=\begin{bmatrix}1&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0\\1&0&1&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&1&0&1\end{bmatrix},\qquad M^{-1}_{(Y1.1)2}=\begin{bmatrix}1&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0\\2&0&1&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&1&0\\2&0&0&0&0&0&2&0&1\end{bmatrix},$$

$$M_{(Z1.1)2}=\begin{bmatrix}1&0&1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0\\1&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0\\1&0&0&0&0&0&1&0&1\\0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&1&0&0\end{bmatrix},\quad\text{and}\quad M^{-1}_{(Z1.1)2}=\begin{bmatrix}0&0&1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0\\1&0&2&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&1&0\\0&0&2&0&0&0&1&0&2\end{bmatrix}.$$

Property 6.2.1. The existing fastest TLI matrices are related such that they provide only four unique fastest TLI polynomial expansions for the given input function. For example, polynomial expansions based on $M_{(Y1.1)n}$, $M_{(Y1.4)n}$, $M_{(Z1.1)n}$, and $M_{(Z1.4)n}$.

Definition 6.2.7. Let $K^{\theta}_{n,j}$ and $\left(K^{\theta}_{n,j}\right)^{-1}$ denote the j-th factorized transform matrix of the forward and inverse transforms of the fastest TLI transform $M_{(Y1.\theta)n}$, respectively ($1\le j\le n$, $\theta\in\{1,2,4,5\}$). Then, $M_{(Y1.\theta)n}$ and $\left(M_{(Y1.\theta)n}\right)^{-1}$ can be represented in the factorized form

$$M_{(Y1.\theta)n}=\prod_{j=n}^{1}K^{\theta}_{n,j} \tag{6.12}$$

and

$$\left(M_{(Y1.\theta)n}\right)^{-1}=\prod_{j=n}^{1}\left(K^{\theta}_{n,j}\right)^{-1}, \tag{6.13}$$

where the general formulae for $K^{\theta}_{n,j}$ and $\left(K^{\theta}_{n,j}\right)^{-1}$ are given in Property 6.2.2.

Property 6.2.2. The factorized transform matrices of $M_{(Y1.\theta)n}$ and $\left(M_{(Y1.\theta)n}\right)^{-1}$ ($\theta\in\{1,2,4,5\}$) can be derived by using (6.14)–(6.21) as follows:

$$K^1_{n,j} = \begin{bmatrix} K^1_{n-1,j} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & K^1_{n-1,j} \end{bmatrix} \tag{6.14}$$

$$\left(K^1_{n,j}\right)^{-1} = \begin{bmatrix} \left(K^1_{n-1,j}\right)^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & \left(K^1_{n-1,j}\right)^{-1} \end{bmatrix} \tag{6.15}$$

$$K^2_{n,j} = \begin{bmatrix} K^2_{n-1,j} & O_{n-1} & O_{n-1} \\ O_{n-1} & X_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & K^2_{n-1,j} \end{bmatrix} \tag{6.16}$$

$$\left(K^2_{n,j}\right)^{-1} = \begin{bmatrix} \left(K^2_{n-1,j}\right)^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & X_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & \left(K^2_{n-1,j}\right)^{-1} \end{bmatrix} \tag{6.17}$$

$$K^4_{n,j} = R_2{}^2\left(K^1_{n,j}\right) \tag{6.18}$$

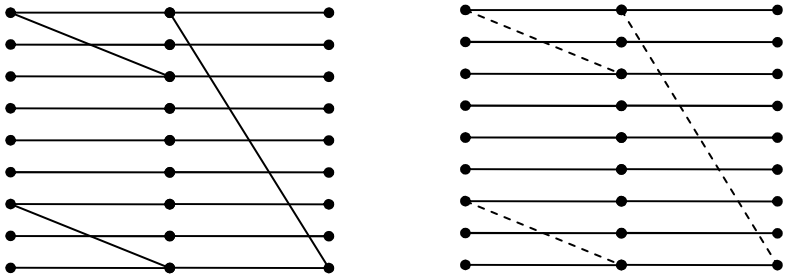
$$\left(K^4_{n,j}\right)^{-1} = R_2{}^2\left(\left(K^1_{n,j}\right)^{-1}\right) \tag{6.19}$$

$$K^5_{n,j} = R_2{}^2\left(K^2_{n,j}\right) \tag{6.20}$$

$$\left(K^5_{n,j}\right)^{-1} = R_2{}^2\left(\left(K^2_{n,j}\right)^{-1}\right), \tag{6.21}$$

where $K^\theta_{j,j}$ is identity matrix of size $3^j \times 3^j$ with bottom left element replaced by 1, $\left(K^\theta_{j,j}\right)^{-1}$ is identity matrix of size $3^j \times 3^j$ with bottom left element replaced by 2 ($\theta \in \{1,2\}$), and $X_{n-1} = \begin{cases} J_{n-1}, & \text{if } n = j+1 \\ I_{n-1}, & \text{otherwise.} \end{cases}$

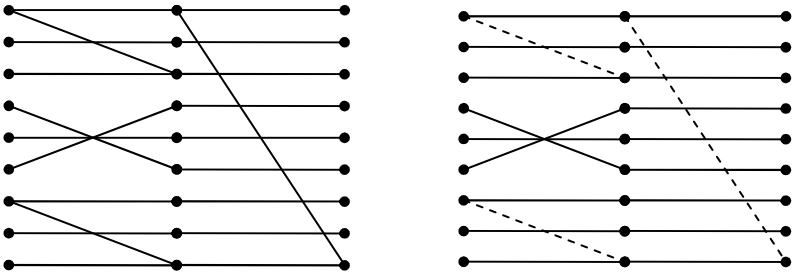
By Definition 6.2.7 and Property 6.2.2, the butterfly diagrams for $M_{(Y1.\theta)_n}$ can be easily constructed. The forward and inverse butterfly diagrams for $M_{(Y1.1)2}$ are shown in Fig 6.1. In Fig 6.2 the forward and inverse butterfly diagrams for $M_{(Y1.2)_2}$ are given. In both figures, the solid and dashed line represents the values 1 and 2, respectively.



(a) (b)

Figure 6.1: Butterfly diagrams of $M_{(Y1.1)2}$:

(a) Forward transform; (b) Inverse transform.



(a) (b)

Figure 6.2: Butterfly diagrams of $M_{(Y1.2)2}$:

(a) Forward transform; (b) Inverse transform.

Property 6.2.3. [FF05c] The number of additions required to compute the spectra of fastest TLI transforms is $2^n - 1$.

6.2.2 Fastest TLI transforms with permutation

The recursive definitions for existing fastest TLI transforms have been given in Section 6.2.1. In this chapter we are concerned with identifying new fastest TLI transforms that have the same computational cost as the existing fastest TLI transforms and can also be calculated efficiently by fast transforms while offering the

possibility of more compact polynomial representations, i.e., have the smaller number of nonzero terms. One of the simplest ways to do that is by permuting the existing fastest TLI transforms. Such class of fastest TLI transforms is defined in this section. Due to the relations between the existing fastest TLI transforms, the LI expansions based on all the fastest TLI transforms with permutation cover the LI expansions based on all the existing fastest TLI transforms. As such, the minimum number of nonzero spectral coefficients in the spectra of fastest TLI transforms with permutation is always smaller than or equal to the minimum nonzero spectral coefficient number in the spectra of existing fastest TLI transforms.

Definition 6.2.8. Let $M_n(p)$ denote the fastest TLI transform matrix of size $3^n \times 3^n$ with permutation number p $\left(0 \le p \le 6^n - 1\right)$. Then $M_n(p)$ and its inverse transform matrix $\left(M_n(p)\right)^{-1}$ are defined as

$$M_n(p) = P_n^p \cdot \prod_{j=n}^{1} K_{n,j}^1 \tag{6.22}$$

and

$$\left(M_n(p)\right)^{-1} = \left(\prod_{j=n}^{1} \left(K_{n,j}^1\right)^{-1}\right) \cdot \left(P_n^p\right)^{-1}, \tag{6.23}$$

respectively.

Property 6.2.4. There are $3^n + 2^n - 1$ nonzero elements inside both $M_n(p)$ and $\left(M_n(p)\right)^{-1}$. All the nonzero elements in $M_n(p)$ are 1s whereas inside $\left(M_n(p)\right)^{-1}$ 3^n of the nonzero elements are 1s and the rest are 2s.

Property 6.2.5. From Definitions 6.2.6 and 6.2.8, the existing fastest TLI transforms and fastest TLI transforms with permutation are related as follows:

$$M_{(Y1.1)n} = M_n(0) \tag{6.24}$$

$$M_{(Y1.4)n} = M_n(6^n - 1) \cdot P_n^{6^n - 1} \tag{6.25}$$

$$M_{(Z1.1)n} = M_n(6^{n-1} - 1) \tag{6.26}$$

$$M_{(Z1.4)n} = M_n(5 \cdot 6^{n-1}) \cdot P_n^{6^n-1}. \quad (6.27)$$

Property 6.2.6. Let $f(\vec{x})$ be an n-variable ternary switching function with the truth vector $\vec{F}$. Then there are $3^n - 2^{n-1}$ spectral coefficients in the spectrum of $f(\vec{x})$ based on any fastest TLI transform with permutation $M_n(p)$ whose values are equal to the values of the truth vector elements, i.e., their values can be directly obtained from $\vec{F}$ without any additions or multiplications. Furthermore, if $\overrightarrow{F_1}$ is defined as the subset of the truth vector elements whose values affect the values of the 2^{n-1} spectral coefficients that need to be calculated, $\overrightarrow{F_1}$ has 2^n elements.

Property 6.2.7. All possible fastest TLI matrix with permutation can be divided into 3^n groups of size 2^n such that if $S(p)$ is defined as the set of truth vector elements that are directly forwarded to the spectral coefficients of $M_n(p)$, then all $M_n(p)$ in the same group have identical set $S(p)$. Therefore, the number of elements inside $S(p)$ that have nonzero values give the minimum number of nonzero elements for the corresponding group of $M_n(p)$.

Let the matrix R be defined as $R = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \\ 2 & 1 & 0 \end{bmatrix}$, where the row and column index numbers start from zero and $R_{i,j}$ denotes the element that is located at row i and column j of R. Then any two fastest TLI matrices with permutation $M_n(p_a)$ and $M_n(p_b)$ belong to the same group if

$$R_{p_{a_l},1} = R_{p_{b_l},1} \text{ for } l = 2,3,\ldots,n$$

and

$$R_{p_{a_l},2} = R_{p_{b_l},2} \text{ for } l = 1, \quad (6.28)$$

where $< p_a >_{10} =< p_{a_n}, p_{a_{n-1}}, \ldots, p_{a_1} >_6$ and $< p_b >_{10} =< p_{b_n}, p_{b_{n-1}}, \ldots, p_{b_1} >_6$.

Example 6.2.2. In Example 6.2.1, the transform matrix $\left(M_{(Y1.1)2}\right)^{-1}$ has been derived. From the matrix, it can be obtained that the set $S(p)$ for $M_{(Y1.1)2} = M_2(0)$ is $S(p) = \{f_0, f_1, f_3, f_4, f_5, f_6, f_7\}$. By Property 6.2.7, the TLI matrices that have the same $S(p)$ as $M_2(0)$ are $M_2(<0,2>)$, $M_2(<5,0>)$, and $M_2(<5,2>) = M_2(2)$, $M_2(30)$, and $M_2(32)$, respectively.

The calculation of the spectra for ternary functions based on all $M_n(p)$ have been implemented in MATLAB and run for several binary benchmark functions that have been modified to represent ternary functions. The translation from binary to ternary cases has been done by changing every two input (output) bits in binary files to an input (output) symbol in ternary files. If the number of input and/or output variables is odd, then a zero bit is first added behind the binary cubes to make it even. For input (output), −− is converted to −, 00 is converted to 0, 01 is converted to 1, 10 is converted to 2, and 11 is ignored (converted to 0). The resulting numbers of nonzero spectral coefficients inside the spectra of each ternary input function based on $M_n(0)$, $M_n(6^n - 1)$, $M_n(6^{n-1} - 1)$, and $M_n(5 \cdot 6^{n-1})$ are listed in Table 6.2. Recall that those fastest TLI transforms with permutation correspond to the existing fastest TLI transforms. In addition, the number of nonzero spectral coefficients for each input function based on all $M_n(p)$ are compared and the minimum number is shown in the rightmost column of Table 6.2. Based on the numbers in Table 6.2, it can be seen that for some ternary functions $M_n(p)$ reduces the number of terms required to represent them, which leads to faster calculation of the output value, for example for con1, rd84, 9sym, and alu4 in Table 6.2.

6.2.3 Generalized fastest TLI transforms

The fastest TLI transforms with permutation defined in Section 6.2.2 can be further extended into a wider set of fastest TLI transforms by allowing the permutation to be located either in one side of the butterfly diagrams or between the butterfly diagram

stages and by allowing the butterfly diagram stages to be reordered such as it has been done for binary fastest LI transforms in Chapter 4. The resulting TLI transforms are called generalized fastest TLI transforms. As reordering and permutation do not incur any additional cost, the computational cost of the generalized ternary fastest LI transforms are the same as the existing fastest TLI transforms.

Table 6.2: Number of nonzero spectral coefficients for $M_n(p)$.

Input filename	Number of nonzero spectral coefficients				
	$M_n(0)$	$M_n(6^n-1)$	$M_n(6^{n-1}-1)$	$M_n(5\cdot 6^{n-1})$	Optimal $M_n(p)$
xor5	10	10	10	10	9
con1	45	46	45	47	42
squar5	16	17	17	16	16
z5xp1	53	53	53	53	53
inc	52	51	51	52	51
rd84	49	50	49	50	43
misex1	52	53	53	52	51
ex5	81	81	81	81	77
9sym	116	123	125	116	107
clip	153	157	156	153	150
apex4	162	161	161	162	157
ex1010	178	179	179	178	174
alu4	2179	2179	2179	2179	2153
misex3	2156	2161	2161	2156	2150

Definition 6.2.9. Let $M_n^{\theta}(\varphi,\sigma,p)$ denote a generalized fastest TLI transform of dimension $3^n\times 3^n$ with ordering φ, permutation position σ $(1\le\sigma\le n+1)$, and permutation number p $(0\le p\le 6^n-1)$. Then $M_n^{\theta}(\varphi,\sigma,p)$ is defined as

$$M_n^{\theta}(\varphi,\sigma,p)=\begin{cases}\left(\prod_{j=n}^{\sigma}K_{n,\varphi_j}^{\theta}\right)\cdot P_n^p\cdot\left(\prod_{j=\sigma-1}^{1}K_{n,\varphi_j}^{\theta}\right), & \text{if } \sigma\neq n+1\\ P_n^p\cdot\prod_{j=n}^{1}K_{n,\varphi_j}^{\theta}, & \text{otherwise,}\end{cases} \tag{6.29}$$

where $\theta\in\{1,2,4,5\}$ and $K_{n,j}^{\theta}$ and P_n^p have been defined in Property 6.2.2 and Definition 6.1.9, respectively.

Property 6.2.8. The ordering φ is an n-digit string in which every digit takes values from 1 to n and no two different digits in it are allowed to have the same values,

$$\varphi=<\varphi_n,\varphi_{n-1},\ldots,\varphi_1>\ , \tag{6.30}$$

where $\varphi_i\in\{1,2,\ ...,n\}$ and $\varphi_i=\varphi_j$ iff $i=j\ (1\le i,j\le n)$.

Property 6.2.9. Clearly, the inverse of $M_n^{\theta}(\varphi,\sigma,p)$ is simply

$$\left(M_n^{\theta}(\varphi,\sigma,p)\right)^{-1}=\begin{cases}\left(\prod_{j=1}^{\sigma-1}\left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right)\cdot\left(P_n^p\right)^{-1}\cdot\left(\prod_{j=\sigma}^{n}\left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right), & \text{if } \sigma\neq n+1\\ \left(\prod_{j=1}^{n}\left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right)\cdot\left(P_n^p\right)^{-1}, & \text{otherwise.}\end{cases} \tag{6.31}$$

Property 6.2.10. Any two generalized fastest TLI matrices $M_n^{\theta_1}(\varphi_1,\sigma_1,p_1)$ and $M_n^{\theta_2}(\varphi_2,\sigma_2,p_2)$ are identical when $\theta_1=\theta_2$, $\sigma_1=\sigma_2$, $p_1=p_2$, and $S_1=\{\varphi_{1l}\mid 1\le l\le\sigma_1-1\}$ = $S_2=\{\varphi_{2l}\mid 1\le l\le\sigma_2-1\}$.

Example 6.2.3. According to Property 6.2.10, the fastest TLI matrix $M_4^1(3201,3,3)$ = $M_4^1(2301,3,3)$ = $M_4^1(3210,3,3)$ = $M_4^1(2301,3,3)$.

Property 6.2.11. By Definitions 6.1.9 and 6.2.9 and Property 6.2.2, it can be derived that

$$M_n^4(\varphi,\sigma,p)=R_2{}^2\left(M_n^1(\varphi,\sigma,p')\right) \tag{6.32}$$

and

$$M_n^5(\varphi,\sigma,p)=R_2{}^2\left(M_n^2(\varphi,\sigma,p')\right), \tag{6.33}$$

where $< p >_{10} =< p_n, p_{n-1}, \ldots, p_1 >_6$, $p'_j = 0, 2, 1, 4, 3,$ and 5 if $p_j = 0, 1, 2, 3, 4,$ and 5, respectively, and $< p' >_{10} =< p'_n, p'_{n-1}, \ldots, p'_1 >_6$.

Property 6.2.12. Let the matrix $Z = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$ and the matrix R be as given in Property 6.2.7. Then the column index numbers of nonzero elements in row *index* ($0 \le index \le 3^n - 1$) of $M_n^{\theta}(\varphi, \sigma, p)$ and $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$ ($\theta \in \{1,2\}$) can be calculated by *Procedure1* shown in Fig. 6.3. Conversely, the row index numbers of nonzero elements in column *index* ($0 \le index \le 3^n - 1$) of $M_n^{\theta}(\varphi, \sigma, p)$ and $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$ can be calculated by *Procedure3* shown in Fig. 6.4. The two procedures call *Procedure2* and *Procedure4* that are given in Fig. 6.5. The value of the argument *inv* for the procedures is equal to zero for $M_n^{\theta}(\varphi, \sigma, p)$ and one for $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$. At the end of the procedures, if *inv* = 1 then the sets E_1 and E_2 contain the *n*-digit ternary representation of the appropriate indexes of 1s and 2s inside $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$. On the other hand, if *inv* = 0 then the union of the sets E_1 and E_2 gives the calculated indexes of 1s inside $M_n^{\theta}(\varphi, \sigma, p)$.

Note that due to Property 6.2.11, the locations of the nonzero elements inside row (column) *index* of $M_n^{\theta}(\varphi, \sigma, p)$ and $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$ for $\theta \in \{4,5\}$ can also be calculated using the procedures in Figs. 6.3–6.5 with appropriate adjustment.

Property 6.2.13. Let $\vec{A}_1$ and $\vec{A}_2$ be the spectra of an *n*-variable ternary function $f(\vec{x})$ with truth vector $\vec{F}$ based on $M_n^4(\varphi, \sigma, p)$ and $M_n^5(\varphi, \sigma, p)$. Then, it follows from Property 6.2.11 that

$$R_0(\vec{A}_1) = \left(M_n^1(\varphi, \sigma, p')\right)^{-1} \cdot R_0(\vec{F}) \tag{6.34}$$

and

$$R_0(\vec{A}_2) = \left(M_n^2(\varphi, \sigma, p')\right)^{-1} \cdot R_0(\vec{F}). \tag{6.35}$$

```
Procedure1(index, θ, φ, p, σ, inv, n)
{  set E1 = {<index>3} and E2 = { };
   if (inv = 0)
      {var1 = σ; var2 = n;}
   else
      {var1 = 1; var 2 = σ − 1;}
   e1 = element of E1;
   if (L1(e1) ≠ 0)
      {if (θ = 2)
         {for (loop_var = var1 to var2)
            {if (φ_loop_var ≠ n)
               {if ((digit (φ_loop_var + 1) of e1 = 1) and (M1(e1) = φ_loop_var + 1))
                  {for (k = 1 to φ_loop_var)
                     {Replace digit e1k by (2 − e1k);
                      break;}}}}}
   else
      {for (loop_var = var1 to var2)
         Procedure2(loop_var);}
   for every element h of E1 and E2
      {if (inv = 0)
         Replace each digit hk by Z_pk,hk ;
       else
         Replace each digit hk by R_pk,hk ;}
   if (inv = 0)
      {var1 = 1; var2 = σ − 1;}
   else
      {var1 = σ; var2 = n;}
   if (θ = 2)
      {for every element e1 of E1
         {if (L1(e1) ≠ 0)
            {for (loop_var = var1 to var2)
               {if (φ_loop_var ≠ n)
```

```
                    {if ((digit ( φ_loop_var +1 ) of e_1 = 1) and (M_1(e_1) = φ_loop_var +1))
                        {for (k = 1 to φ_loop_var )
                            {Replace digit e_1k by (2 − e_1k);
                             break;}}}}}}
            for every element e_2 of E_2
                {if (L_1(e_2) ≠ 0)
                    {for (loop_var = var1 to var2)
                        {if ( φ_loop_var ≠ n )
                            {if ((digit ( φ_loop_var +1 ) of e_2 = 1) and (M_1(e_2) = φ_loop_var +1))
                                {for (k = 1 to φ_loop_var )
                                    {Replace digit e_2k by (2 − e_2k);
                                     break;}}}}}}
            for (loop_var = var1 to var2)
                Procedure2(loop_var);}
```

Figure 6.3: *Procedure1*(*index*, θ, φ, *p*, σ, *inv*, *n*).

```
    Procedure3(index, θ, φ, p, σ, inv, n)
{   set E_1 = {<index>_3} and E_2 = { };
    if (inv = 0)
        {var1 = 1; var2 = σ − 1;}
    else
        {var1 = σ ; var 2 = n;}
    e_1 = element of E_1;
    if (L_1(e_1) ≠ 0)
        {if ( θ = 2)
            {for (loop_var = var1 to var2)
                {if ( φ_loop_var ≠ n )
                    {if ((digit ( φ_loop_var +1 ) of e_1 = 1) and (M_1(e_1) = φ_loop_var +1))
                        {for (k = 1 to φ_loop_var )
                            {Replace digit e_1k by (2 − e_1k);
                             break;}}}}}}
    else
```

```
    {for (loop_var = var1 to var2)
        Procedure4(loop_var);}
for every element h of E1 and E2
    {if (inv = 0)
        Replace each digit h_k by R_{p_k,h_k} ;
     else
        Replace each digit h_k by Z_{p_k,h_k} ;}
if (inv = 0)
    {var1 = σ ; var2 =n;}
else
    {var1 = 1; var2 = σ − 1;}
if ( θ = 2)
    {for every element e1 of E1
        {if (L1(e1) ≠ 0)
            {for (loop_var = var1 to var2)
                {if ( φ_{loop_var} ≠ n )
                    {if ((digit ( φ_{loop_var} +1) of e1 = 1) and (M1(e1) = φ_{loop_var} +1))
                        {for (k = 1 to φ_{loop_var} )
                            {Replace digit e1k by (2 − e1k);
                             break;}}}}}}
    for every element e2 of E2
        {if (L1(e2) ≠ 0)
            {for (loop_var = var1 to var2)
                {if ( φ_{loop_var} ≠ n )
                    {if ((digit ( φ_{loop_var} +1) of e2 = 1) and (M1(e2) = φ_{loop_var} +1))
                        {for (k = 1 to φ_{loop_var} )
                            {Replace digit e2k by (2 − e2k);
                             break;}}}}}}
for (loop_var = var1 to var2)
    Procedure4(loop_var);}
```

Figure 6.4: *Procedure3*(*index*, θ, φ, p, σ, *inv*, *n*).

Procedure2(loop_var)
{ set E_3 = { } and E_4 = { };
for every element e_1 of E_1
{ if (($L_b(e_1 + 1) > \varphi_{loop\_var}$) and ($L_1(e_1) = 0$))
{$e_3 = e_1$ with bits 1 to φ_{loop_var} replaced by 0;
add e_3 to E_3;}}
for every element e_2 of E_2
{ if (($L_b(e_2 + 1) > \varphi_{loop\_var}$) and ($L_1(e_2) = 0$))
{$e_4 = e_2$ with digits 1 to φ_{loop_var} replaced by 0;
add e_4 to E_4;}}
add all elements of E_3 to E_2;
add all elements of E_4 to E_1;}

Procedure4(loop_var)
{ set E_3 = { } and E_4 = { };
for every element e_1 of E_1
{ if (($L_{b2}(e_1) > \varphi_{loop\_var}$) and ($L_1(e_1) = 0$))
{$e_3 = e_1$ with bits 1 to φ_{loop_var} replaced by 2;
add e_3 to E_3;}}
for every element e_2 of E_2
{ if (($L_{b2}(e_2) > \varphi_{loop\_var}$) and ($L_1(e_2) = 0$))
{$e_4 = e_2$ with digits 1 to φ_{loop_var} replaced by 2;
add e_4 to E_4;}}
add all elements of E_3 to E_2;
add all elements of E_4 to E_1;}

Figure 6.5: *Procedure2(loop_var)* and *Procedure4(loop_var)*.

Example 6.2.4. By Definition 6.2.9,

$$M_2^1(12,2,16) = K_{2,1}^1 \cdot P_2^{16} \cdot K_{2,2}^1$$

$$= \begin{bmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0&1 \end{bmatrix} \begin{bmatrix} 0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&1 \end{bmatrix}$$

$$= \begin{bmatrix} 0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1&1 \end{bmatrix}.$$

By Property 6.2.11, $M_2^4(12,2,9) = R_2{}^2(M_2^1(12,2,16))$

$$= \begin{bmatrix} 1&1&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \end{bmatrix}.$$

Also, by Property 6.2.12, the steps for calculating the set of the column index numbers of 1s in row eight of $M_2^1(12,2,16)$ are

- *index* = 8, <*index*>$_3$ = 22, n = 2, inv = 0, $E_1 = \{22\}$, $E_2 = \{\ \}$.
- *loop_var* = 2 $\rightarrow$ $E_1 = \{22\}$, $E_2 = \{20\}$.
- Replace each digit h_k by $Z_{p_k,h_k} \rightarrow E_1 = \{21\}$, $E_2 = \{22\}$.
- *loop_var* = 1 $\rightarrow$ $E_1 = \{21, 00\}$, $E_2 = \{22\}$.

Hence, the 1s in row eight of $M_2^1(12,2,16)$ are located at columns 0, 7, and 8.

The calculation of the spectra based on all generalized fastest TLI transforms

$M_n^{\theta}(\varphi,\sigma,p)$ has also been implemented in MATLAB and run for the ternary functions that are obtained from binary benchmark functions in the manner described in Section 6.2.2. The resulting minimum number of nonzero spectral coefficients that can be obtained by each type of $M_n^{\theta}(\varphi,\sigma,p)$ are shown in Table 6.3. Comparing the numbers in Tables 6.2 and 6.3, it can be observed that for some ternary functions, $M_n^{\theta}(\varphi,\sigma,p)$ can give more compact representations that those based on $M_n(p)$ in terms of smaller number of nonzero spectral coefficients. Since $M_n(p)$ is a special case of $M_n^{\theta}(\varphi,\sigma,p)$, the minimum number of spectral coefficients based on all $M_n^{\theta}(\varphi,\sigma,p)$ is never larger than that based on $M_n(p)$.

Table 6.3: Minimum number of nonzero spectral coefficients for $M_n^{\theta}(\varphi,\sigma,p)$.

Input filename	$M_n^1(\varphi,\sigma,p)$	$M_n^2(\varphi,\sigma,p)$	$M_n^4(\varphi,\sigma,p)$	$M_n^5(\varphi,\sigma,p)$
xor5	8	8	8	8
con1	41	41	38	39
squar5	15	15	16	15
z5xp1	52	52	51	51
inc	47	47	45	46
rd84	42	43	41	43
misex1	47	46	47	45
ex5	76	77	75	75
9sym	103	103	107	107
clip	144	144	145	145
apex4	155	155	153	153
ex1010	174	174	174	174

6.3 Fastest LITA transforms

New fastest LITA transforms are presented in this section. Similar to the case for fastest TLI transforms, the new transforms are built from the factorized transform matrices of the existing fastest LITA transforms [FF03, FF06] and their definitions cover those existing fastest LITA transforms. Basic definitions for the fastest LITA transforms and the existing fastest LITA transforms are first given. The new fastest LITA transforms are then defined. Relations between the new and existing fastest LITA transforms as well as several other properties and experimental results for the new fastest LITA transforms are then derived.

6.3.1 Basic definitions for fastest LITA transforms

Definition 6.3.1. Any *n*-variable ternary function $f(\vec{x})$ can be represented by its LITA polynomial expansion as follows:

$$f(\vec{x}) = \sum_{j=0}^{3^n-1} c_j g_j , \tag{6.36}$$

where g_j ($0 \le j \le 3^n - 1$) is a ternary switching function whose truth vector corresponds to the *j*-th column of a particular TLI transform M_n and c_j is the *j*-th LITA spectral coefficient for $f(\vec{x})$ based on T_n. The addition is performed over standard arithmetic algebra.

Definition 6.3.2. Let $\vec{F} = [f_0, f_1, ..., f_{3^n-1}]^T$ be a column truth vector of a ternary function $f(\vec{x})$ in natural ternary ordering and $\vec{C} = [c_0, c_1, ..., c_{3^n-1}]^T$ be the coefficient column vector (spectrum) of $f(\vec{x})$ for T_n. Then,

$$\vec{F} = T_n \cdot \vec{C} , \tag{6.37}$$

and

$$\vec{C} = T_n^{-1} \cdot \vec{F} , \tag{6.38}$$

where T_n^{-1} denotes the inverse of T_n over standard arithmetic algebra, T represents matrix transpose operator, and c_j is as given in Definition 6.3.1.

In [FF03], sixteen fastest LITA transforms have been identified. They are classified into class Z1 and Z2. Each class has eight transforms, where four of them have only 0s and 1s whereas the other four contain 2s as well. Here we are concerned with only those fastest LITA transforms without 2 elements as their spectral coefficients can be calculated without any multiplications and therefore are simpler. The recursive definitions for the forward and inverse transforms of such fastest LITA transforms are listed in Table 6.4, where O_{n-1} is a $3^{n-1}\times 3^{n-1}$ matrix with all its elements equal to zero, I_{n-1} is an identity matrix of size $3^{n-1}\times 3^{n-1}$, J_{n-1} is a reverse identity matrix of size $3^{n-1}\times 3^{n-1}$, Y_{n-1} is a $3^{n-1}\times 3^{n-1}$ matrix with all its elements equal to zero except one element at the corner that is equal to 1, and Z_{n-1} is a $3^{n-1}\times 3^{n-1}$ matrix with all its elements equal to zero except one element at the corner that is equal to -1.

Other fastest LITA transforms have also been introduced in [FF06] where the new transforms are obtained by applying *Rot* and Rot^{-1} operators (see Definitions 6.2.4 and 6.2.5) on some of the class Z1 and Z2 matrices.

Definition 6.3.3. [FF06] The class Z*1 and Z*2 matrices are derived from class Z1 and Z2 matrices by the following equations, where $x, y \in \{1,2\}$.

$$T_{(Z^*1.x.y)n} = Rot\left(T_{(Z1.x.y)n}\right) \tag{6.39}$$

$$T_{(Z^*2.x.y)n} = Rot^{-1}\left(T_{(Z2.x.y)n}\right) \tag{6.40}$$

$$T^{-1}_{(Z^*1.x.y)n} = Rot^{-1}\left(T^{-1}_{(Z1.x.y)n}\right) \tag{6.41}$$

$$T^{-1}_{(Z^*2.x.y)n} = Rot\left(T^{-1}_{(Z2.x.y)n}\right). \tag{6.42}$$

Property 6.3.1. From Tables 6.1 and 6.4 and Definitions 6.2.6 and 6.3.3, the forward

transforms of the class Z1, Z2, Z*1, and Z*2 fastest LITA transforms coincide with the forward transforms of the class Y1, Y2, Z1, and Z2 fastest TLI transforms, respectively.

Table 6.4: Class Z1 and Z2 matrices.

Matrix name	T_n	T_n^{-1}
Z1.1.1	$\begin{bmatrix} T_{n-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ Y_{n-1} & O_{n-1} & T_{n-1} \end{bmatrix}$	$\begin{bmatrix} T_{n-1}^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ Z_{n-1} & O_{n-1} & T_{n-1}^{-1} \end{bmatrix}$
Z1.1.2	$\begin{bmatrix} T_{n-1} & O_{n-1} & Y_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & T_{n-1} \end{bmatrix}$	$\begin{bmatrix} T_{n-1}^{-1} & O_{n-1} & Z_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & T_{n-1}^{-1} \end{bmatrix}$
Z1.2.1	$\begin{bmatrix} T_{n-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ Y_{n-1} & O_{n-1} & T_{n-1} \end{bmatrix}$	$\begin{bmatrix} T_{n-1}^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ Z_{n-1} & O_{n-1} & T_{n-1}^{-1} \end{bmatrix}$
Z1.2.2	$\begin{bmatrix} T_{n-1} & O_{n-1} & Y_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & T_{n-1} \end{bmatrix}$	$\begin{bmatrix} T_{n-1}^{-1} & O_{n-1} & Z_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & T_{n-1}^{-1} \end{bmatrix}$
Z2.1.1	$\begin{bmatrix} Y_{n-1} & O_{n-1} & T_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ T_{n-1} & O_{n-1} & O_{n-1} \end{bmatrix}$	$\begin{bmatrix} O_{n-1} & O_{n-1} & T_{n-1}^{-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ T_{n-1}^{-1} & O_{n-1} & Z_{n-1} \end{bmatrix}$
Z2.1.2	$\begin{bmatrix} O_{n-1} & O_{n-1} & T_{n-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ T_{n-1} & O_{n-1} & Y_{n-1} \end{bmatrix}$	$\begin{bmatrix} Z_{n-1} & O_{n-1} & T_{n-1}^{-1} \\ O_{n-1} & J_{n-1} & O_{n-1} \\ T_{n-1}^{-1} & O_{n-1} & O_{n-1} \end{bmatrix}$
Z2.2.1	$\begin{bmatrix} Y_{n-1} & O_{n-1} & T_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ T_{n-1} & O_{n-1} & O_{n-1} \end{bmatrix}$	$\begin{bmatrix} O_{n-1} & O_{n-1} & T_{n-1}^{-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ T_{n-1}^{-1} & O_{n-1} & Z_{n-1} \end{bmatrix}$
Z2.2.2	$\begin{bmatrix} O_{n-1} & O_{n-1} & T_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ T_{n-1} & O_{n-1} & Y_{n-1} \end{bmatrix}$	$\begin{bmatrix} Z_{n-1} & O_{n-1} & T_{n-1}^{-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ T_{n-1}^{-1} & O_{n-1} & O_{n-1} \end{bmatrix}$

Property 6.3.2. It follows from Properties 6.2.1 and 6.3.1 that only four unique fastest

LITA polynomial expansions can be obtained from the set of all class Z1, Z2, Z*1, and Z*2 fastest LITA transforms for a ternary input function.

Due to Property 6.3.1, the fastest TLI transforms and fastest LITA transforms are closely related and so for compatibility with the fastest TLI transforms, from here onwards the fastest LITA transform matrices $T_{(Z1.1.1)n}$, $T_{(Z1.2.1)n}$, $T_{(Z1.1.2)n}$ and $T_{(Z1.2.2)n}$ will also be referred as T_n^1, T_n^2, T_n^4, and T_n^5, respectively.

Definition 6.3.4. Let $K_{n,j}^{\theta}$ and $\left(K_{n,j}^{\theta}\right)^{-1}$ denote the j-th factorized transform matrix of the forward and inverse transforms of the fastest LITA transforms T_n^{θ}, respectively ($1 \le j \le n, \theta \in \{1,2,4,5\}$). Then, T_n^{θ} and $\left(T_n^{\theta}\right)^{-1}$ can be represented in the factorized form

$$T_n^{\theta} = \prod_{j=n}^{1} K_{n,j}^{\theta} \tag{6.43}$$

and

$$\left(T_n^{\theta}\right)^{-1} = \prod_{j=n}^{1} \left(K_{n,j}^{\theta}\right)^{-1}, \tag{6.44}$$

where the general formulae for $\left(K_{n,j}^{\theta}\right)^{-1}$ are given in Property 6.3.3 and the formulae for $K_{n,j}^{\theta}$ follows (6.14), (6.16), (6.18) and (6.20) for θ = 1, 2, 4, and 5, respectively.

Property 6.3.3. The factorized transform matrices of $\left(T_n^{\theta}\right)^{-1}$ ($\theta \in \{1,2,4,5\}$) can be derived by using (6.45)–(6.48) as follows:

$$\left(K_{n,j}^{1}\right)^{-1} = \begin{bmatrix} \left(K_{n-1,j}^{1}\right)^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & I_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & \left(K_{n-1,j}^{1}\right)^{-1} \end{bmatrix} \tag{6.45}$$

$$\left(K_{n,j}^{2}\right)^{-1} = \begin{bmatrix} \left(K_{n-1,j}^{2}\right)^{-1} & O_{n-1} & O_{n-1} \\ O_{n-1} & X_{n-1} & O_{n-1} \\ O_{n-1} & O_{n-1} & \left(K_{n-1,j}^{2}\right)^{-1} \end{bmatrix} \tag{6.46}$$

$$\left(K_{n,j}^{4}\right)^{-1} = R_2{}^{2}\left(\left(K_{n,j}^{1}\right)^{-1}\right) \tag{6.47}$$

$$\left(K_{n,j}^{5}\right)^{-1} = R_2{}^{2}\left(\left(K_{n,j}^{2}\right)^{-1}\right), \tag{6.48}$$

where $\left(K_{j,j}^{\theta}\right)^{-1}$ is identity matrix of size $3^j \times 3^j$ with bottom left element replaced by -1 ($\theta \in \{1,2\}$), and $X_{n-1} = \begin{cases} J_{n-1}, & \text{if } n = j+1 \\ I_{n-1}, & \text{otherwise.} \end{cases}$

The butterfly diagrams for the inverse fastest LITA transforms $\left(T_n^{\theta}\right)^{-1}$ are shown in Fig. 6.6 for $n = 2$, where the solid and dashed lines represent the values 1 and -1, respectively.

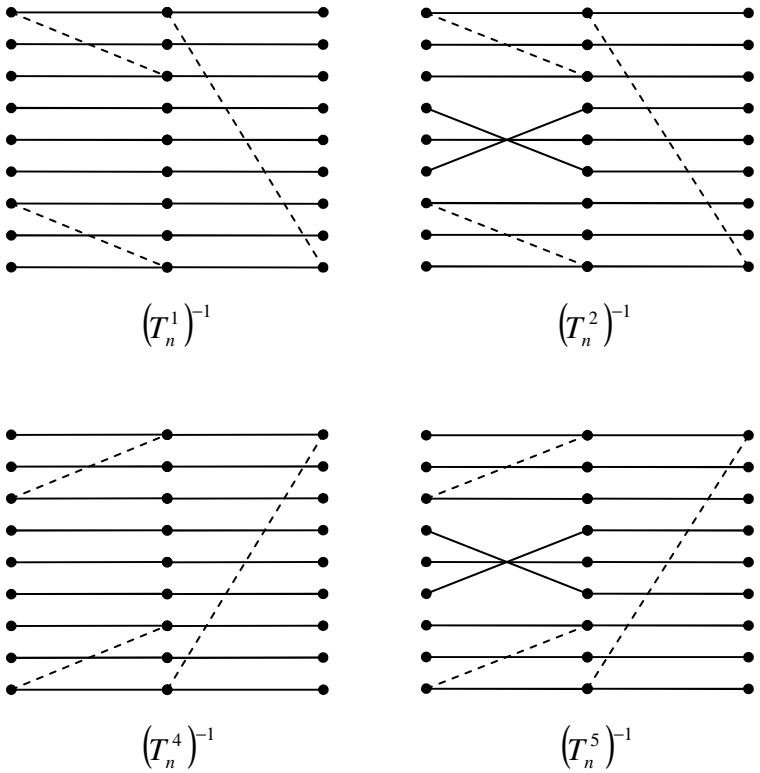


Figure 6.6: Butterfly diagrams of $\left(T_n^{\theta}\right)^{-1}$.

Example 6.3.1. Let $f(\vec{x})$ be a two-variable ternary function with truth vector

$\vec{F} = [1,1,1,2,1,2,2,1,2]^T$. Then by Definitions 6.3.2 and 6.3.4 and Property 6.3.3, the coefficient column vector for $f(\vec{x})$ based on T_2^2 is

$$\begin{aligned}\vec{C} &= \left(T_2^2\right)^{-1} \cdot \vec{F} \\ &= \left(K_{2,2}^2\right)^{-1} \cdot \left(K_{2,1}^2\right)^{-1} \cdot \vec{F} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \end{bmatrix} \\ &= [1\ 1\ 0\ 2\ 1\ 2\ 2\ 1\ -1]^T. \end{aligned}$$

Thus, the LITA polynomial expansion for $f(\vec{x})$ based on T_2^2 is

$$f(\vec{x}) = g_0 + g_1 + 2g_3 + g_4 + 2g_5 + 2g_6 + g_7 - g_8,$$

where $g_j (0 \le j \le 8)$ denotes the ternary switching function whose truth vector is given by column j of T_2^2.

6.3.2 Fastest LITA transforms with permutation

In this section, the existing fastest LITA transforms given in Section 6.3.1 is extended to a larger group of LITA transforms with the same computational cost by means of permutation matrix. Definitions and properties for the new fastest LITA transforms are presented and the number of nonzero terms for the new fastest LITA transforms are shown and compared with that of the existing fastest LITA transforms for several ternary test files.

Definition 6.3.5. Let $T_n(p)$ denote a $3^n \times 3^n$ fastest LITA transform matrix with permutation number p ($0 \le p \le 3^n - 1$). Then, $T_n(p)$ and its inverse transform $(T_n(p))^{-1}$ are defined as

$$T_n(p) = P_n^p \cdot \prod_{j=1}^{n} K_{n,j}^1 \tag{6.49}$$

and

$$\left(T_n(p)\right)^{-1} = \left(\prod_{j=1}^{n} \left(K_{n,j}^1\right)^{-1}\right) \cdot \left(P_n^p\right)^{-1}, \tag{6.50}$$

where $K_{n,j}^1$ and $\left(K_{n,j}^1\right)^{-1}$ $(1 \le j \le n)$ are the j-th factorized transform matrix of T_n^1 and $\left(T_n^1\right)^{-1}$, respectively and the permutation matrices P_n^p and $\left(P_n^p\right)^{-1}$ have been given in Definition 6.1.9.

Property 6.3.4. Due to the Definition 6.3.5 and Properties 6.2.5 and 6.3.1, the following relations between the existing fastest LITA transforms and fastest LITA transforms with permutation hold.

$$\left(T_n(0)\right)^{-1} = \left(T_n^1\right)^{-1} \tag{6.51}$$

$$\left(T_n(6^n - 1)\right)^{-1} = P_n^{6^n-1} \cdot \left(T_n^4\right)^{-1} \tag{6.52}$$

$$\left(T_n(6^{n-1} - 1)\right)^{-1} = \left(T_{(Z^*1.1.1)n}\right)^{-1} \tag{6.53}$$

$$\left(T_n(5 \cdot 6^{n-1})\right)^{-1} = P_n^{6^n-1} \cdot \left(T_{(Z^*1.1.2)n}\right)^{-1}. \tag{6.54}$$

In what follows, several properties on the number and location of nonzero elements in $\left(T_n(p)\right)^{-1}$ are presented. These properties can be applied to speed up computation when only selected LITA spectral coefficients need to be computed. Furthermore, they are also needed for analyzing various properties of the fastest LITA spectra that are useful for testing and fault detection application.

Property 6.3.5. There are $3^n + 2^n - 1$ nonzero elements inside $\left(T_n(p)\right)^{-1}$, where 3^n of them are 1s and the rest are -1s.

For the following properties, let the matrices R and Z be as defined in Properties

6.2.7 and 6.2.12, respectively, where $R_{s,t}$ ($Z_{s,t}$) ($0 \le s \le 5$; $0 \le t \le 2$) represents the element that is located at row s and column t of matrix R (Z).

Property 6.3.6. The column index numbers of the nonzero elements in row i ($0 \le i \le 3^n - 1$) of $(T_n(p))^{-1}$ can be determined by the following steps:

Step 1: Let $< p_n, p_{n-1}, ..., p_1 >$ and $< i >_3 = < i_n, i_{n-1}, ..., i_1 >$ be the n-digit ternary representations of p and i, respectively. Set $l = 0$ and $< h >_3 = < i >_3$.

Step 2: Set $E_5 = \{ < i >_3 \}$ and $E_6 = \{ \}$. If the number of 1s in $< i >_3$ is not equal to zero: go to Step 5. Else: go to Step 3.

Step 3: Increment l by 1. If $l > n$: go to Step 5. Else go to Step 4.

Step 4: If $i_l = 2$: add $< h' >_3$ to E_6, where $< h' >_3 = < h >_3$ with h_l set to 0; set $< h >_3 = < h' >_3$; go back to Step 3. Else: go to Step 5.

Step 5: For each n-digit ternary number $< h >_3$ in E_5 and E_6: Replace each digit h_l ($1 \le l \le n$) in $< h >_3$ with R_{p_l, h_l}.

After Step 5, the decimal number representations of the E_5 (E_6) elements are the column index numbers of 1s (−1s) in row i.

Property 6.3.7. The row index numbers of the nonzero elements in column i ($0 \le i \le 3^n - 1$) of $(T_n(p))^{-1}$ can be determined by the following steps:

Step 1: Let $< p_n, p_{n-1}, ..., p_1 >$ and $< i >_3 = < i_n, i_{n-1}, ..., i_1 >$ be the n-digit ternary representations of p and i, respectively.

Step 2: Replace each digit i_l ($1 \le l \le n$) in $< i >_3$ with Z_{p_l, i_l}. Set $l = 0$ and $< h >_3 = < i >_3$.

Step 3: Set $E_7 = \{ < i >_3 \}$ and $E_8 = \{ \}$. If the number of 1s in $< i >_3$ is not equal to zero: exit. Else: go to Step 4.

Step 4: Increment l by 1. If $l > n$: exit. Else: go to Step 5.

Step 5: If $i_l = 0$: add $< h' >_3$ to E_8, where $< h' >_3 = < h >_3$ with h_l set to 2; set $< h >_3 = < h' >_3$; go back to Step 4. Else: exit.

At the end of the steps above, the decimal number representations of the E_7 (E_8) elements are the row index numbers of 1s (−1s) in column i of $(T_n(p))^{-1}$.

Property 6.3.8. The crossing lines in the butterfly diagrams of $(T_n(p))^{-1}$ coincide with the crossing lines in the butterfly diagrams of $(M_n(p))^{-1}$ when their permutation numbers p are the same. Thus, Properties 6.2.6 and 6.2.7 apply to $T_n(p)$ as well.

The resulting numbers of nonzero spectral coefficients for T_n^1 and T_n^4 for several ternary test files are shown in Table 6.5 together with the smallest number of nonzero spectral coefficients based on all $T_n(p)$. It can be seen that for most of the ternary test files, the fastest LITA transforms with permutation are able to provide smaller number of nonzero spectral coefficients than the existing fastest LITA transforms, which corresponds to smaller and simpler hardware implementation.

Table 6.5: Minimum number of nonzero spectral coefficients for $T_n(p)$.

Input files	T_n^1	T_n^4	Min. nonzero number for all $T_n(p)$
xor5	10	10	9
con1	47	46	44
squar5	16	17	16
z5xp1	53	53	53
inc	52	51	51
rd84	51	50	44
misex1	52	53	51
ex5	81	81	77
9sym	116	126	108
clip	153	157	152
apex4	162	161	158
ex1010	178	179	176
alu4	2179	2179	2155
misex3	2159	2162	2155

6.3.3 Generalized fastest LITA transforms

By not restricting the position of factorized transform matrices in the factorized representation of the fastest LITA transforms and allowing the permutation matrix to be located between them, in this section the fastest LITA transforms with permutation presented in previous section is extended to an even wider set of fastest LITA transforms. Such transforms are called generalized fastest LITA transforms.

Definition 6.3.6. Let $T_n^{\theta}(\varphi,\sigma,p)$ denote a generalized fastest LITA transform of dimension $3^n \times 3^n$ with ordering φ, permutation position σ $(1 \le \sigma \le n+1)$, and permutation number p $(0 \le p \le 6^n - 1)$. Then $T_n^{\theta}(\varphi,\sigma,p)$ is defined as

$$T_n^{\theta}(\varphi,\sigma,p) = \begin{cases} \left(\prod_{j=n}^{\sigma} K_{n,\varphi_j}^{\theta}\right) \cdot P_n^p \cdot \left(\prod_{j=\sigma-1}^{1} K_{n,\varphi_j}^{\theta}\right), & \text{if } \sigma \neq n+1 \\ P_n^p \cdot \prod_{j=n}^{1} K_{n,\varphi_j}^{\theta}, & \text{otherwise,} \end{cases} \tag{6.55}$$

where $\theta \in \{1,2,4,5\}$ and $K_{n,j}^{\theta}$ and P_n^p have been defined in Property 6.2.2 and Definition 6.1.9, respectively.

Property 6.3.9. The ordering φ is an n-digit string in which every digit takes values from 1 to n and no two different digits in it are allowed to have the same values,

$$\varphi = < \varphi_n, \varphi_{n-1}, \ldots, \varphi_1 > , \tag{6.56}$$

where $\varphi_i \in \{1,2, \ldots, n\}$ and $\varphi_i = \varphi_j$ iff $i = j$ $(1 \le i, j \le n)$.

Property 6.3.10. By Definitions 6.1.9 and 6.3.6 and Property 6.3.3, the inverse of $T_n^{\theta}(\varphi,\sigma,p)$ is simply

$$\left(T_n^{\theta}(\varphi,\sigma,p)\right)^{-1} = \begin{cases} \left(\prod_{j=1}^{\sigma-1} \left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right) \cdot \left(P_n^p\right)^{-1} \cdot \left(\prod_{j=\sigma}^{n} \left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right), & \text{if } \sigma \neq n+1 \\ \left(\prod_{j=1}^{n} \left(K_{n,\varphi_j}^{\theta}\right)^{-1}\right) \cdot \left(P_n^p\right)^{-1}, & \text{otherwise.} \end{cases} \tag{6.57}$$

Example 6.3.2. By Definition 6.3.6, the matrix $T_2^5(21,2,9)$ can be derived as follows:

$$T_2^5(21,2,9) = K_{2,2}^5 \cdot P_2^9 \cdot K_{2,1}^5$$

$$= \begin{bmatrix} 1&0&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{bmatrix}$$

$$= \begin{bmatrix} 0&1&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \end{bmatrix}.$$

Furthermore, by Property 6.3.10, the inverse matrix $\left(T_2^5(21,2,9)\right)^{-1}$ is

$$\left(T_2^5(21,2,9)\right)^{-1} = \left(K_{2,1}^5\right)^{-1} \cdot \left(P_2^9\right)^{-1} \cdot \left(K_{2,2}^5\right)^{-1}$$

$$= \begin{bmatrix} 1&0&-1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&-1 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0&-1 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{bmatrix}$$

$$= \begin{bmatrix} 0&-1&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&-1 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&-1&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \end{bmatrix}.$$

Fig. 6.7 shows the butterfly diagrams for $T_2^5(21,2,9)$ and $\left(T_2^5(21,2,9)\right)^{-1}$.

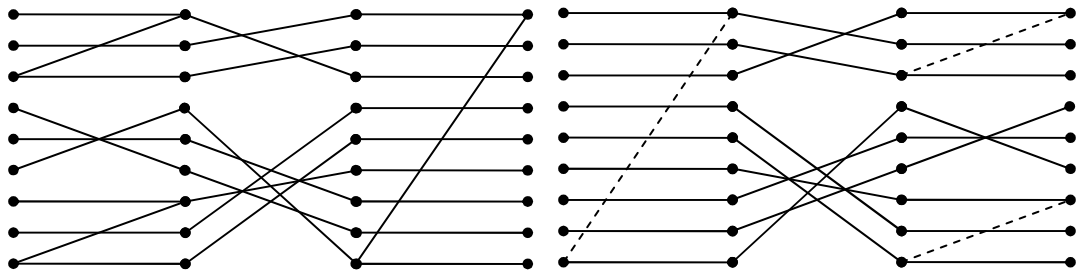


Figure 6.7: Butterfly diagrams for $T_2^5(21,2,9)$:

(a) Forward transform; (b) Inverse transform.

Property 6.3.11. Any two generalized fastest LITA matrices $T_n^{\theta_1}(\varphi_1,\sigma_1,p_1)$ and $T_n^{\theta_2}(\varphi_2,\sigma_2,p_2)$ are identical when $\theta_1=\theta_2$, $\sigma_1=\sigma_2$, $p_1=p_2$, and $S_1=\{\varphi_{1l}\,|\,1\le l\le\sigma_1-1\}$ = $S_2=\{\varphi_{2l}\,|\,1\le l\le\sigma_2-1\}$. Thus, only one of them needs to be computed to obtain a unique LITA polynomial expansion for the input function.

Property 6.3.12. The generalized fastest LITA transforms of different types are related as follows

$$\left(T_n^4(\varphi,\sigma,p)\right)^{-1}=R_2{}^2\left(T_n^1(\varphi,\sigma,p')\right)^{-1} \tag{6.58}$$

and

$$\left(T_n^5(\varphi,\sigma,p)\right)^{-1}=R_2{}^2\left(T_n^2(\varphi,\sigma,p')\right)^{-1}, \tag{6.59}$$

where $<p>_{10}=<p_n,p_{n-1},\ldots,p_1>_6$, p'_j = 0, 2, 1, 4, 3, and 5 if p_j = 0, 1, 2, 3, 4, and 5, respectively, and $<p'>_{10}=<p'_n,p'_{n-1},\ldots,p'_1>_6$.

Also,

$$T_n^1(\varphi,\sigma,p)=T_n^4(\varphi',\sigma',p'')^T \tag{6.60}$$

and

$$T_n^2(\varphi,\sigma,p)=T_n^5(\varphi',\sigma',p'')^T, \tag{6.61}$$

where $\varphi=<\varphi_n,\varphi_{n-1},\ldots,\varphi_1>$, $\varphi'=<\varphi_1,\varphi_2,\ldots,\varphi_n>$, $<p''>_{10}=<p''_n,p''_{n-1},\ldots,p''_1>_6$,

$\sigma' = n + 2 - \sigma$, and p''_j = 0, 1, 2, 4, 3, and 5 if p_j = 0, 1, 2, 3, 4, and 5, respectively.

Property 6.3.13. Similarly, it can be found that

$$T_n^{\theta}(\varphi, \sigma, p) = \left| T_n^{\theta}(\varphi', \sigma', p'')^{-1} \right|, \tag{6.62}$$

where φ', σ', and p'' have been defined in Property 6.3.12.

Property 6.3.14. As the forward fastest LITA transforms $T_n^{\theta}(\varphi, \sigma, p)$ are simply the forward fastest TLI transforms $M_n^{\theta}(\varphi, \sigma, p)$, the location of nonzero elements inside the matrices can be obtained by *Procedure1* and *Procedure3* as specified in Property 6.2.12. In addition, the location of the nonzero elements inside the inverse fastest LITA transforms $\left(T_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$ can also be obtained by the same procedure for $\left(M_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$. However, for $\left(T_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$ the elements of sets E_1 and E_2 should be taken as the n-digit ternary representation of the appropriate indexes of 1s and −1s, respectively.

Property 6.3.15. The computational cost of obtaining one fastest LITA transform spectrum from the truth vector is $2^n - 1$ additions/subtractions.

Property 6.3.16. Let $A = \{\varphi_\sigma, \varphi_{\sigma+1}, \ldots, \varphi_n\} - \{n\}$, $B = \{\varphi_1, \varphi_2, \ldots, \varphi_{\sigma-1}\}$, and $A' = \{A'_1, A'_2, \ldots, A'_{|A|}\}$ be A that is reordered such that $A_i < A_j$ if $i < j$. Also, let $\vec{C}_1$, $\vec{C}_2$, $\vec{C}_3$, and $\vec{C}_4$ be the spectra of $T_n^1(\varphi, \sigma, p)$, $T_n^2(\varphi, \sigma, p)$, $T_n^4(\varphi, \sigma, p)$, and $T_n^5(\varphi, \sigma, p)$, respectively for the same input function. Then $\vec{C}_1$ ($\vec{C}_3$) are simply $\vec{C}_2$ ($\vec{C}_4$) that are reordered if either one of the conditions in (6.63)–(6.66) is satisfied.

$$\sigma = n + 1 \tag{6.63}$$

$$\sigma = n \text{ and } \varphi_n = n \tag{6.64}$$

$$Z_{p_{A'_j+1},1} = 1 \forall j, 1 \le j \le |A| \tag{6.65}$$

$$\begin{aligned}\exists t \text{ such that } (Z_{p_{A'_j+1},1} \neq 1 \forall j, 1 \le j \le t) \text{ and } (Z_{p_{A'_j+1},1} = 1 \forall j, t < j \le |A|) \\ \text{and } (A'_t < \min(B)) \text{ and } (Z_{p_1,1} = Z_{p_2,1} = \cdots = Z_{p_{A'_t+1},1}).\end{aligned} \tag{6.66}$$

Property 6.3.17. For $\theta \in \{1,2\}$ there are $(2^n - 1)$ -1s and $(3^n + h)$ 1s inside $\left(T_n^{\theta}(\varphi, \sigma, p)\right)^{-1}$, where

$$h = \sum_{i=1}^{\sigma-1} \sum_{j=\sigma}^{n} \begin{cases} 2^{n-\max(\varphi_i,\varphi_j)-|D|}, & \text{if } R_{p_k,0} = 2 \forall k, 1 \le k \le \min(\varphi_i, \varphi_j) \text{ and } \varepsilon(i,j) = 1 \\ 0, & \text{otherwise,} \end{cases} \tag{6.67}$$

$$\varepsilon(i,j) = \begin{cases} 1, & \text{if } \left((\varphi_i > \varphi_j) \text{ and } (R_{p_k,0} \neq 1 \forall k, \varphi_j < k \le \varphi_i)\right) \text{or} \\ & \left((\varphi_i < \varphi_j) \text{ and } (R_{p_k,1} \neq 2 \forall k, \varphi_i < k \le \varphi_j)\right) \\ 0, & \text{otherwise,} \end{cases} \tag{6.68}$$

and

$$D = \{k \mid R_{p_k,1} \neq 1, \max(\varphi_i, \varphi_j) < k \le n\}. \tag{6.69}$$

When $\sigma = 1$ or $(n+1)$, $h = 0$ and so $T_n^{\theta}(\varphi, \sigma, p)$ has $3^n + 2^n - 1$ nonzero elements. This property can be easily adjusted for $\theta \in \{4,5\}$ by Property 6.3.12.

The calculation of the spectra based on all generalized fastest LITA transforms $T_n^{\theta}(\varphi, \sigma, p)$ have also been implemented in MATLAB and run for the ternary test files. The resulting minimum numbers of nonzero spectral coefficients that can be obtained by each type of $T_n^{\theta}(\varphi, \sigma, p)$ are shown in Table 6.6. Since $T_n^{\theta}(\varphi, \sigma, p)$ covers both the existing fastest LITA transforms and $T_n(p)$, the minimum number of spectral coefficients based on all $T_n^{\theta}(\varphi, \sigma, p)$ is never larger than that based on $T_n(p)$. Comparing the numbers in Table 6.6 with the number of nonzero spectral coefficients of the optimal FPRME over GF(3) for the same set of ternary test files, it was found that the former are smaller than the latter for some of the ternary test files. Examples of such cases are clip and ex1010, for which the best number of nonzero FPRME spectral coefficients is 153 and 236, respectively.

Table 6.6: Minimum number of nonzero spectral coefficients for $T_n^{\theta}(\varphi, \sigma, p)$.

Input filename	$T_n^1(\varphi, \sigma, p)$	$T_n^2(\varphi, \sigma, p)$	$T_n^4(\varphi, \sigma, p)$	$T_n^5(\varphi, \sigma, p)$
xor5	8	8	8	8
con1	42	41	38	39
squar5	15	15	16	16
z5xp1	53	53	52	53
inc	47	47	45	46
rd84	43	44	41	43
misex1	47	46	47	45
ex5	76	77	75	75
9sym	103	103	108	108
clip	144	144	145	145
apex4	155	155	153	153
ex1010	175	174	176	174

Chapter 7

Hardware Implementations

Hardware calculation and implementation of the polynomial expansions based on the transforms introduced in Chapters 4 and 6 are presented in this chapter. First, the conversion between the truth vector and spectral coefficients of the respective transforms are shown using linear systolic array processors. The linear systolic array processor structures are derived from the butterfly diagrams of the respective transforms. As example, the steps of deriving the linear systolic array processors for fastest LIA transforms are described. The multi level tree implementations of the resulting fastest LI polynomial expansions are then shown for both binary and ternary cases. Afterward, the use of the linear systolic array structures for obtaining the spectra of the FPRMEs and FPAEs using the recursive polarity matrix algorithms are discussed.

7.1 Linear systolic array structure for fastest LI transforms

The calculation of fastest LIA transform spectra as well as the spectra of the fastest LI transforms over GF(2), fastest LITA transforms, and fastest TLI transforms can be efficiently performed using a linear systolic array processor [KSY91, KSZ90, Yan94, Yan98]. A linear systolic array processor is constructed from a number of uniform

processors that are interconnected linearly such that the output of a processor is connected to the input of the next processor. Each of the processors has two basic elements, namely computing cell and storage cell. Generally, the computing cell has several computational modules for performing data manipulation whereas the storage cells are composed of first-in first-out (FIFO) registers. The number and type of the computational modules vary depending on the characteristic of the transform being implemented on the systolic array processor. At each iteration, a computing cell performs the specified operations on the data and the result is sent to the computing cell of the next processor as well as to the corresponding storage cell.

The linear systolic array processor structure for calculating the spectral coefficients of a particular LI transform can be obtained from the butterfly diagrams of the transform itself. In the following text, the derivation of the linear systolic array processor structure for fastest LIA transforms from their butterfly diagrams is described. The linear systolic array processor structure for the fastest LI transforms over GF(2), fastest LITA transforms, and fastest TLI transforms can be derived from their respective butterfly diagrams using very similar derivation steps.

In general, the complete spectrum for a particular fastest LIA transform is obtained by the systolic array processor after 2^n iterations, where one truth vector element is inputted and one spectral coefficient is produced at each iteration. The basic steps for the generation of the linear systolic array processor structure from the butterfly diagram are given below, followed by explanation for each of the steps and several examples. The steps are:

Step 1: Obtain the $(n + 1)$-stage butterfly diagrams of the corresponding fastest LIA transform based on the factorized representation for the LIA transform given in Definitions 4.3.3 and 4.3.4.

Step 2: Transform the $(n + 1)$-stage butterfly diagram into an equivalent n-stage butterfly diagram by eliminating the stage that corresponds to the permutation matrix.

Step 3: Combine some stages of the butterfly diagrams in Step 2, if it is possible.

Step 4: Generate the systolic array processor structure based on the butterfly diagram obtained in Step 3.

Step 1 is quite straightforward. From the factorized representation of the fastest LIA transform given in Definition 4.3.4, an $(n + 1)$-stage butterfly diagram for the fastest LIA transform can be directly obtained. One stage in the butterfly diagram represents the permutation matrix P_n^p whereas the remaining stages represent the fast factorized transform matrices $K_{n,j}^{\theta}$. From Fig. 4.11, it can be observed that for $\theta \in \{a, b\}$ all the solid lines in the butterfly diagrams of $K_{n,j}^{\theta}$ are horizontal whereas for $\theta \in \{c, d\}$ none of them are horizontal. The former case is desirable as it leads to simpler mapping to the systolic array processor. Therefore, for $\theta \in \{a, b\}$ the $(n + 1)$-stage butterfly diagram can be directly used as input to Step 2 whereas for $\theta \in \{c, d\}$ the butterfly diagram need to be first modified before continuing to Step 2. The modification involves reordering the input and output such that all the solid lines in the butterfly diagram stages representing $K_{n,j}^{\theta}$ are horizontal.

In Step 2, the $(n + 1)$-stage butterfly diagram produced in Step 1 is transformed into an equivalent n-stage butterfly diagram by removing the stage that corresponds to P_n^p and adjusting the other stages such that the input-output relation is preserved. There maybe more than one possible equivalent n-stage butterfly diagrams for the input butterfly diagram. The chosen butterfly diagram should fulfill the following two requirements. First, all the dotted lines in the butterfly diagram must have the same direction, i.e., all dotted lines are from bottom left to top right or from top left to bottom right. Second, all dotted lines in the same stage must have the same length, i.e., the number of lines separating the edges of all the dotted lines in a stage must be exactly the same.

The n-stage butterfly diagram obtained in Step 2 is examined in Step 3 to check whether some of the stages can be combined. That is, whether the number of stages in the butterfly diagram can be reduced. Two consecutive stages can be combined if the lengths of the dotted lines in the stages are the same and there is no precedence constraints between them (the stages can be exchanged without affecting the output vector).

Finally, in Step 4 the systolic array processor structure is generated based on the

butterfly diagram produced in Step 3. All the information required for the systolic array processor can be found from the butterfly diagram. The number of required processors in systolic array is equal to the number of the butterfly diagram stages. The register length in the storage cell is one more than the number of lines separating the edges of dotted lines in the corresponding stage. The location of subtraction operations in the systolic array corresponds to the location of subtractions in the butterfly diagram. Also, the ordering of the inputs (truth vector elements) and outputs (spectral coefficients) fed to and produced by the systolic array are either the same as in the final butterfly diagram (when the direction of the dotted lines are from top left to bottom right) or in the reverse ordering.

The general steps above apply for any fastest LIA transform $T_n^{\theta}(\varphi,\sigma,p)$. However, for some fastest LIA transforms several of the steps may be skipped due to their inherent structure. Below the generation process for several different cases of fastest LIA transforms are discussed. Their resulting systolic array processor structures are also analyzed.

First, let us examine the case when the permutation number is equal to zero. Clearly, as P_n^0 is simply the identity matrix, the permutation matrix inside the factorized representation of $T_n^{\theta}(\varphi,\sigma,0)$ can be ignored without affecting the fastest LIA transform. As a result, when $p = 0$ Step 2 can be performed by simply removing the stage that corresponds to the permutation matrix. No further adjustment or changes is needed to offset the effect of removing it. For further analysis of $T_n^{\theta}(\varphi,\sigma,0)$, in the following $T_n^{\theta}(\varphi,\sigma,0)$ is differentiated into two cases: when $\theta\in\{a,b\}$ and when $\theta\in\{c,d\}$. Each case is discussed separately.

Let us start with the case when $\theta\in\{a,b\}$. According to the given steps above, the systolic array derivation process for $T_n^{a}(\varphi,\sigma,0)$ and $T_n^{b}(\varphi,\sigma,0)$ starts with the generation of the $(n + 1)$-stage butterfly diagram which would be passed on to Step 2. In Step 2, the $(n + 1)$-stage butterfly diagram is transformed into an n-stage butterfly diagram by simply removing the stage that corresponds to the permutation matrix. It can be obtained from (4.5.7) and (4.5.8) that the resulting n-stage butterfly diagram

satisfies the two requirements stated in Step 2. In addition, it can also be found that the lengths of the dotted lines in different stages of the butterfly diagrams are not the same. As a result, none of the stages in the butterfly diagram can be combined in Step 3. That is, the output of the Step 3 is equal to the output of Step 2, which in turns is simply the butterfly diagram of T_n^{θ} ($\theta = a$ or b) with same or different ordering. As Property 4.3.4 states that the reordering of the stages inside T_n^{θ} does not affect the output, it can be concluded from these that for p = 0 and $\theta \in \{a, b\}$ we can derive the systolic array processor for $T_n^{\theta}(\varphi, \sigma, 0)$ directly from the butterfly diagram of T_n^{θ}, i.e., Steps 1 to 3 can be skipped. The resulting systolic array contains n processors and has a total register length of $2^{n+1} - n - 2$. As an example, the butterfly diagram and systolic array processor structure for $T_3^a(210,1,0)$ is shown in Fig. 7.1.

In Figs. 7.1(a) and 7.1(b), the three-stage butterfly diagram for $T_3^a(210,1,0)$ and the corresponding linear systolic array processor structure as well as its corresponding storage cells contents for calculating the coefficient vector of $T_3^a(210,1,0)$ are shown. As it can be seen from the figure, the resulting systolic array processor consists of three stages, where the truth vector elements are fed to the first stage computing cell and the spectral coefficients are obtained from the output of the third stage computing cell. At the first iteration, the first element of the truth vector f_0 is supplied to the input of the first stage computing cell and the resulting contents of the storage cells are as shown in the top row of Fig. 7.1(b). The output of the third computing cell is the first spectral coefficient, c_0. In the subsequent iterations, the other elements of the truth vector are supplied one by one in the sequence of f_1, f_2, …, f_7 and the spectral coefficients are obtained in the sequence of c_1, c_2, …, and c_7 as shown in the lower rows of Fig. 7.1(b). It can be seen that the length of the registers in the first, second, and third stages of the systolic array processor are 1, 3, and 7, respectively with total register length of $1+3+7 = 2^4-3-2 = 11$.

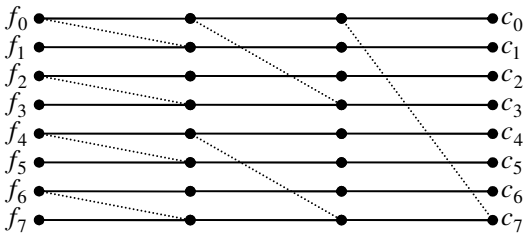
Unlike for $\theta \in \{a, b\}$, when $\theta \in \{c, d\}$ the systolic array structures for $T_n^{\theta}(\varphi, \sigma, 0)$ are dependent on the ordering (φ) used. Different orderings may lead to different

numbers of required processors and register lengths. Generally, the numbers are smaller than or equal to the corresponding numbers for $T_n^a(\varphi,\sigma,0)$ and $T_n^b(\varphi,\sigma,0)$. As Step 2 is performed by simply removing the permutation stage, care must be taken so that the butterfly diagram produced in Step 1 satisfy the two requirements given in Step 2. An example of that is presented in Fig. 7.2 for $T_3^d(012,1,0)$. In Fig. 7.2(a) the original three-stage butterfly diagram for $T_3^d(012,1,0)$ is shown which is then modified into more regular equivalent butterfly diagram given in Fig. 7.2(b) and compressed into two-stage butterfly diagram in Fig. 7.2(c). The butterfly diagram is then mapped into the systolic array processor structure shown in Fig. 7.2(d). Note that the linear systolic array processor has only two stages and it requires only two registers. Fewer numbers of required processors and storage cells contribute to lower cost of hardware implementation by systolic array processor.

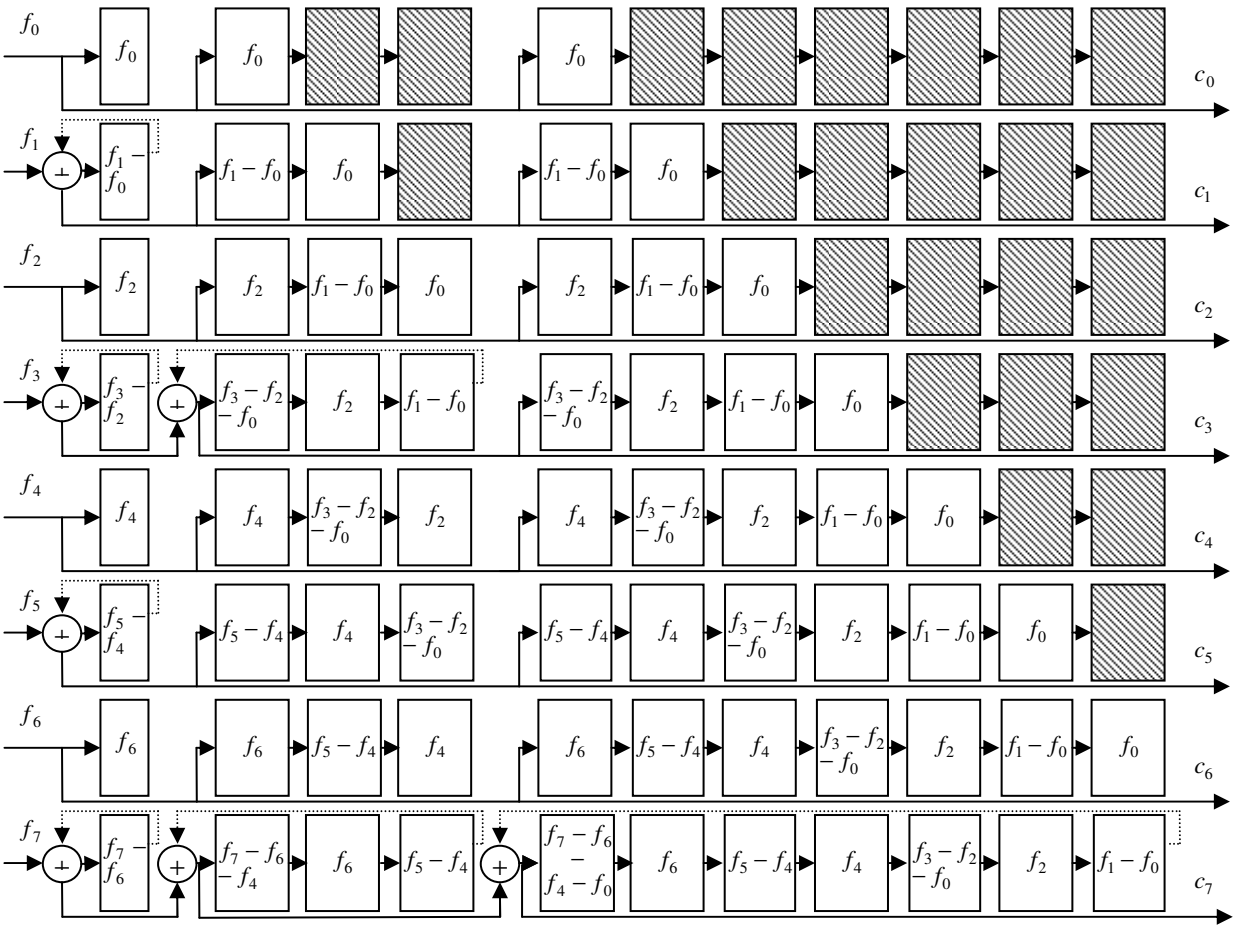
Next, let us consider the more general case when the permutation number is not equal to zero. When $p \neq 0$, the permutation matrix is no longer equal to the identity matrix. Therefore, Step 2 can no longer be executed by simply removing the butterfly diagram stage that corresponds to the permutation matrix. Modification in the other butterfly diagram stages and/or reordering of input or output vectors need to be further performed to offset the effect of removing the permutation stage. In general, generating the systolic array processor structure for $T_n^\theta(\varphi,\sigma,p)$ when $\theta \in \{a,b\}$ is still simpler then for $\theta \in \{c,d\}$. This is mainly due to Step 1, where $T_n^c(\varphi,\sigma,p)$ and $T_n^d(\varphi,\sigma,p)$ need an extra processing step compared to $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$. However, unlike for the case when $p = 0$, the length of the required storage in the systolic array processor for $T_n^a(\varphi,\sigma,p)$ and $T_n^b(\varphi,\sigma,p)$ is no longer fixed. An example of this is shown in Fig. 7.3, where the initial four-stage butterfly diagram for the forward transform of $T_3^a(012,2,2)$ is shown in Fig. 7.3(a). In Fig. 7.3(b), the equivalent three-stage butterfly diagram for $T_3^a(012,2,2)$ chosen in Step 2 is given. Note that the permutation stage inside the butterfly diagram shown in Fig. 7.3(a) (the second leftmost stage) has been removed. Also, the leftmost stage of the butterfly diagram as well as the ordering of the input vector have been rearranged in order to

preserve the input-output relationship. As the first and second leftmost stages of Fig. 7.3(b) have no input-output precedence and all the dotted lines inside them have the same length and direction, they can be compressed in Step 3. Figs. 7.3(c) and 7.3(d) show the compressed butterfly diagram and the corresponding linear systolic array structure for $T_3^a(012,2,2)$. It can be seen that the final systolic array requires only two processors and four registers. These numbers are fewer than three processors and 11 registers required for $T_3^a(210,1,0)$.

If the spectra of more than one fastest LIA transforms need to be generated, it is very useful to investigate whether the fastest LIA transforms are related such that they can be implemented with fewer number of systolic array processors. That is, whether the same systolic array processor can be reused for different fastest LIA transforms so that the implementation cost is reduced. Properties 4.3.2, 4.3.4, 4.3.5, and 4.3.7 have identified groups of fastest LIA transforms that have such relations. As a result, the total number of systolic array processors required for generating spectra of all fastest LIA transforms is much less than the number of all fastest LIA transforms. For example, Property 4.3.2 states that the four fastest LIA transforms $T_n^a(\varphi,\sigma,0)$, $T_n^b(\varphi,\sigma,0)$, $T_n^c(\varphi_n^a,\sigma,0)$, and $T_n^d(\varphi_n^a,\sigma,0)$ can be derived from each other by rearrangement of rows and/or columns. This implies that the spectra calculations of those four fastest LIA transforms can all be performed on the same systolic array processor structure. Suppose that the systolic array processor structure for obtaining the spectrum of $T_n^a(\varphi,\sigma,0)$ has been derived. Then, the spectrum of $T_n^d(\varphi_n^a,\sigma,0)$ is obtained when the input to the systolic processor is applied in reverse ordering. Interpreting the output of the systolic processor in reverse ordering produces the spectrum of $T_n^c(\varphi_n^a,\sigma,0)$. Finally, doing both of them will produce the spectrum of $T_n^b(\varphi,\sigma,0)$. Thus, only one systolic array processor structure is needed to obtain the four spectra instead of four. In addition, Property 4.3.5 implies that the systolic array structure derived for $T_n^a(\varphi,\sigma,n)$ ($T_n^c(\varphi,\sigma,n)$) can also be used to perform spectrum generation for $T_n^b(\varphi,\sigma,n)$ ($T_n^d(\varphi,\sigma,n)$) by simply replacing every f_i and c_i in the derived systolic processor by f_{2^n-1-i} and c_{2^n-1-i}, respectively ($0 \le i \le 2^n-1$).



(a)



Note: ····· indicates that the output of the register is multiplied by −1 prior to being added.

(b)

Figure 7.1: Calculation of $T_3^a(210,1,0)$ spectrum:

(a) Butterfly diagram; (b) Systolic array processor structure.

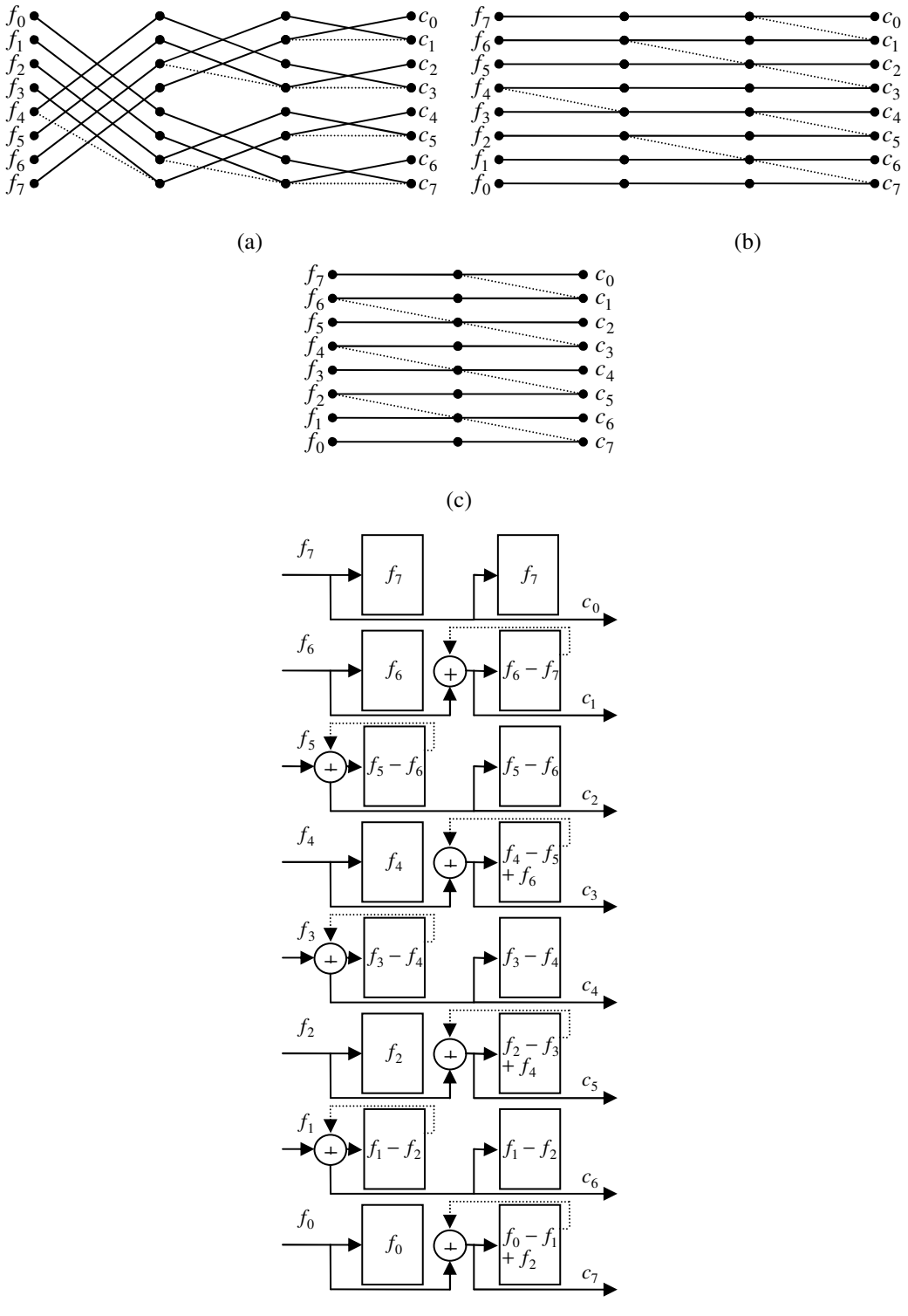


Figure 7.2: Calculation of $T_3^d(012,1,0)$ spectrum:

(a) Butterly diagram (initial); (b) Butterly diagram (three-stage equivalent);

(c) Butterly diagram (two-stage equivalent); (d) Systolic array processor structure.

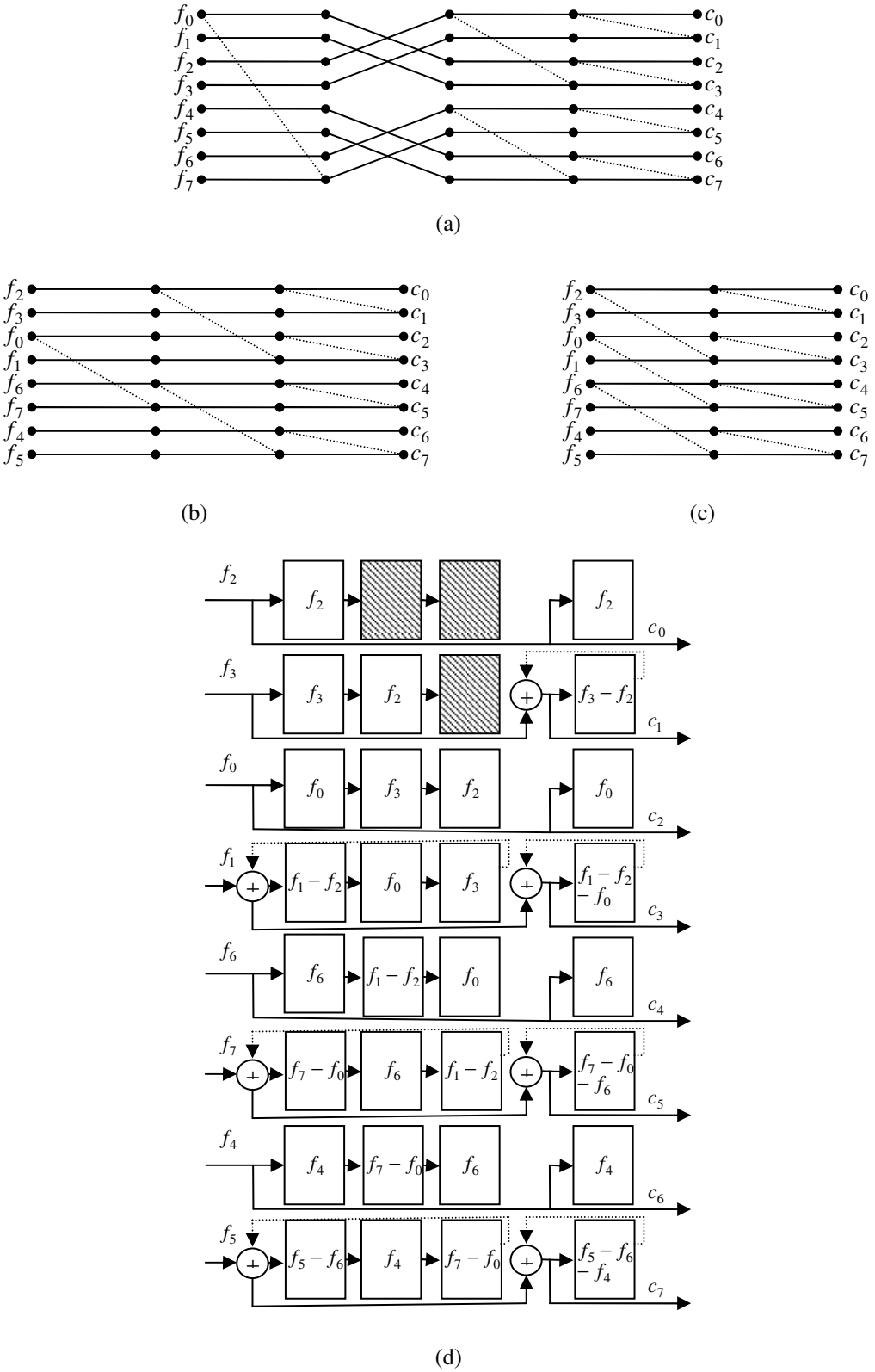
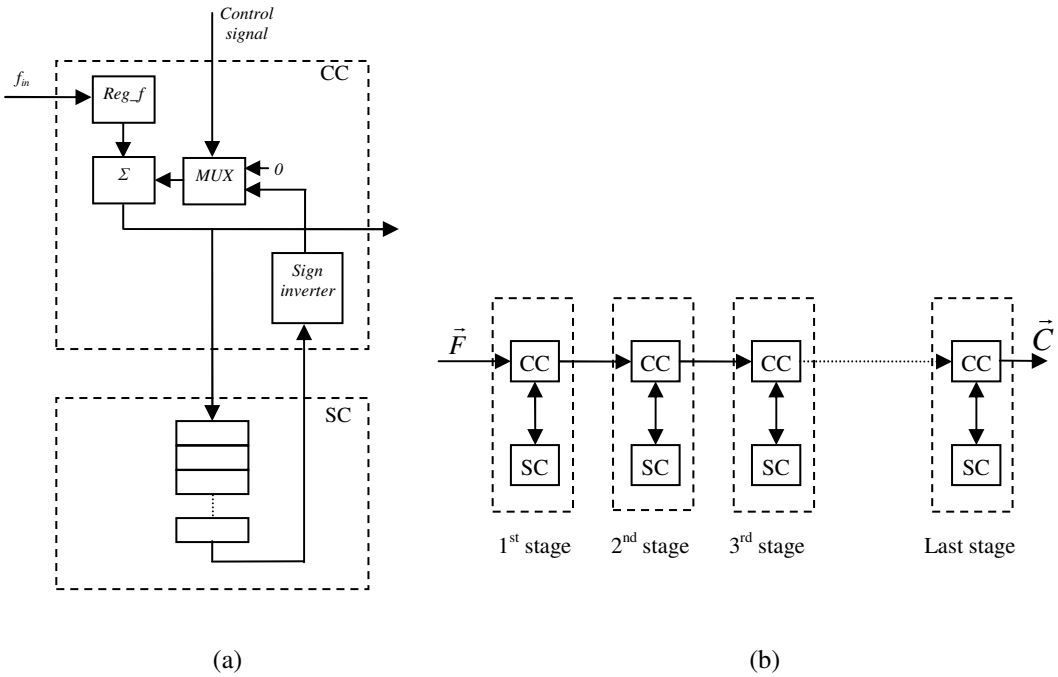


Figure 7.3: Calculation of $T_3^a(012,2,2)$ spectrum: (a) Initial four-stage butterfly diagram; (b) Equivalent three-stage butterfly diagram; (c) Equivalent two-stage butterfly diagram; (d) Systolic array processor structure.

One possible linear systolic array processor structure for calculation of fastest LIA transform spectra is shown in Fig. 7.4. Fig. 7.4(a) shows the configuration of the computing cell (CC) and storage cell (SC) for each stage in the linear systolic array. The computing cell contains an input register, adder, sign inverter and multiplexer. Each computing cell has two inputs: either the input vector (for the first stage) or the output of the previous CC and the oldest content of the SC. The former input is stored into an internal register *Reg_f* whereas the latter input is fed into the sign inverter. The multiplexer passes either the output of the sign inverter or zero based on the control signal. The adder sums up the output of the multiplexer and the content of *Reg_f*. The result of the adder is both passed to the next CC and stored into the corresponding SC. The connection between the stages of the linear systolic array processor is shown in Fig. 7.4(b).



(a) (b)

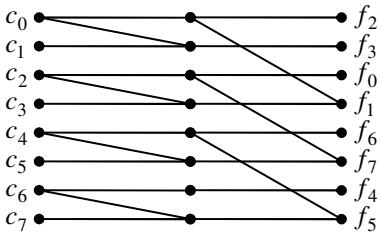
Figure 7.4: Linear systolic array processor structure:
(a) CC and SC in a stage; (b) Inter stage connections.

Besides the relations between the different fastest LIA transforms, the relation between the forward and inverse fast transforms for any fastest LIA transforms has been presented in Property 4.3.6. It follows from this property that the butterfly

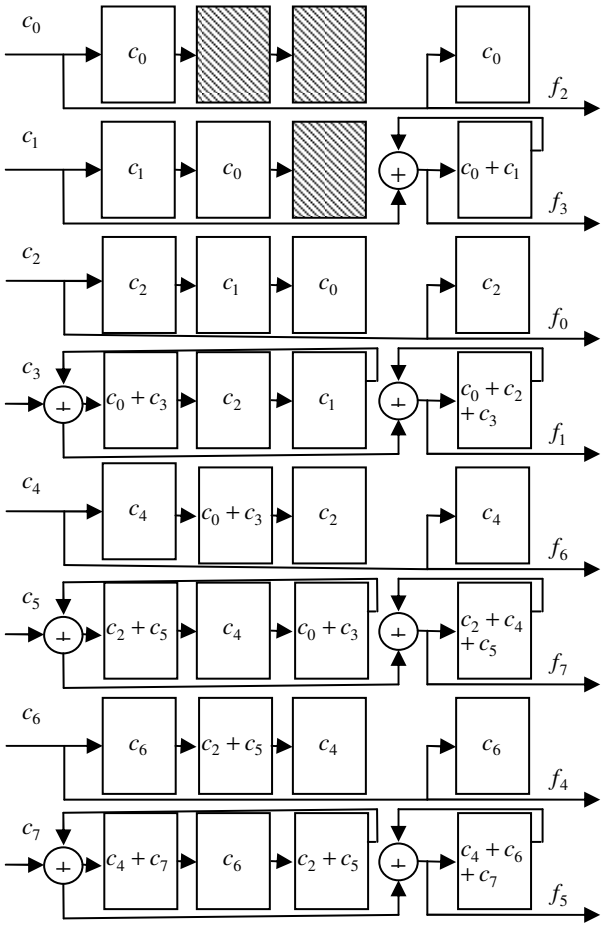
diagram for an inverse fastest LIA transform can be obtained by performing the following operations to the butterfly diagram of the corresponding forward transform. In order to ensure that the resulting butterfly diagram directly corresponds to the systolic array processor implementation of the inverse transform, these operations should be performed on the final forward butterfly diagram (the one that was obtained after Step 3):

- Flip the butterfly diagram structure both horizontally and vertically.
- Exchange the input and output vectors while maintaining their respective sequences.
- Replaced all dotted lines with solid lines.

Clearly, a special case occurs when each stage in the final butterfly diagram for a forward fastest LIA transform remains the same when it is rotated 180°. In such cases, the resulting inverse butterfly diagram is simply the forward butterfly diagram with input and output vectors exchanged, all dotted lines replaced with solid lines and the ordering of the stages reversed. Thus, the systolic array processor structure for implementing the inverse fastest LIA transform is simply the one for the forward fastest LIA transform with the ordering of its stages reversed. It should be noted that if there is no input-output precedence between the different stages in the butterfly diagram, then the systolic processor structure for both forward and inverse transforms are exactly the same. An example of such case is given in Fig. 7.5 for $T_3^a(012,2,2)$. Fig. 7.5(a) shows the butterfly diagram for the inverse transform of $T_3^a(012,2,2)$. Note that the figure is obtained by flipping the butterfly diagram in Fig. 7.3(c) both horizontally and vertically. The linear systolic array processor structure for the inverse of $T_3^a(012,2,2)$ is shown in Fig. 7.5(b). shows the corresponding linear systolic array processor structure. It can be seen that the linear systolic array processor structures in Fig. 7.3(d) for the forward transform and Fig. 7.5(b) for the inverse transform are the same. So, both the forward and inverse transformations of $T_3^a(012,2,2)$ can be implemented on the same systolic array processor.



(a)



(b)

Figure 7.5: Calculation of truth vector from $T_3^a(012,2,2)$ spectrum:

(a) Fast inverse butterfly diagram; (b) Systolic array processor structure.

It should also be noticed that due to the close relation between fastest LI transforms over GF(2) and fastest LIA transforms given by Definitions 4.2.1, 4.2.6,

4.2.17, 4.3.1, and 4.3.4, it follows that for $\theta \in \{a, b\}$ the linear systolic array structure for a particular fastest LI transform over GF(2) $M_n^{\theta}(\varphi, \sigma, p)$ is exactly the same as the linear systolic array structure for fastest LIA transform $T_n^{\theta^*}(\varphi^*, \sigma^*, p)$ except that now all the additions/subtractions are performed over GF(2), where $\theta^* = \theta$, $\sigma^* = n + 2 - \sigma$, and $\varphi^* = <\varphi_1, \varphi_2, \ldots, \varphi_n>$.

7.2 Multi level tree structure implementation

In [GE78], a synthesis procedure for FPRME over a finite field has been proposed which starts from the FPRME coefficients to produce the most economical multi level tree for the expansion. The tree is composed of only two-input gates to avoid the necessity of providing a wide range of fan-in versions for each member of the circuit element family. The synthesis procedure makes use of the general modular tree structures for FPRME that are presented in the paper. Given the FPRME coefficients, circuit realization for the corresponding input function can be obtained directly, although it may not be minimized. The circuit for an *n*-variable function is synthesized from the circuit realization of the (*n*−1)-variable functions. In this section, similar multi level tree structure implementations for the polynomial expansions based on the binary and ternary fastest LI transforms are given.

7.2.1 Binary polynomial expansions based on fastest LIA transforms and fastest LI transforms over GF(2)

Due to the recursive structure of the fastest LIA transforms and fastest LI transforms over GF(2) with permutation number zero, the multi level tree structure for the circuit realization of their polynomial expansions can be easily derived. The modules for the fastest LI transform $M_n^a(\varphi, \sigma, 0)$ with one, two, and three variables are shown in Fig. 7.6. In the figure, the symbols $\oplus, \wedge$, and $\vee$ denote XOR, AND, and OR operations, respectively. It can be observed that the *n*-variable module for $M_n^a(\varphi, \sigma, 0)$ can be built

from two $n-1$ variable modules with one of them being slightly modified. Let $f(\vec{x}_{n-1})$ ($\vec{x}_{n-1}=[x_{n-1},x_{n-2},\ldots,x_1]$) be the $n-1$ variable function realized by the $n-1$ variable module, $f_0(\vec{x}_{n-1})$ be the term that corresponds to a_0 in $f(\vec{x}_{n-1})$, and $f^*(\vec{x}_{n-1})$ be the function similar to $f(\vec{x}_{n-1})$ but with different set of coefficients. Then for $n>3$,

$$f(\vec{x}_n)=f^*(\vec{x}_{n-1})x_n\oplus(f(\vec{x}_{n-1})-a_0f_0(\vec{x}_{n-1}))\bar{x}_n\oplus a_0(f_0(\vec{x}_{n-1})\bar{x}_n\vee x_{n-1}x_{n-2}\ldots x_1). \quad (7.1)$$

Similar to the case for systolic processor, the multi level tree shown in Fig. 7.6 can also be used for other fastest LI transforms over GF(2) $M_n^b(\varphi,\sigma,0)$, $M_n^c(\varphi_a^n,\sigma,0)$, and $M_n^d(\varphi_a^n,\sigma,0)$ by simply reversing the order of the coefficients and/or complimenting the variables as appropriate, where φ_a^n has been defined in Property 4.3.1. In Fig. 7.7 the two-variable module for $M_n^b(\varphi,\sigma,0)$ is shown, which is derived from the two-variable module in Fig. 7.6 by changing the polarities of the input variables and reversing the order of the coefficients. Furthermore, due to Definitions 4.2.2 and 4.2.6, the circuit realization for fastest LI transforms $M_n^\theta(\varphi,\sigma,0)$ can also be easily adjusted to realize the polynomial expansion of $M_n^\theta(\varphi,n+1,p)$ ($1\le p\le 2^n-1$) by simply changing the polarities of the input variables according to the polarity numbers. Let $<p_n,p_{n-1},...,p_1>$ be the n-bit binary representation of the polarity number p. Then in order to obtain the circuit realization for $M_n^\theta(\varphi,n+1,p)$ from that of $M_n^\theta(\varphi,\sigma,0)$, simply change the polarities of the input variables x_i, where $1\le i\le n$ and $p_i=1$. For example, in Fig. 7.8, the module for $M_2^b(\varphi,n+1,2)$ is shown. Comparing the Figs. 7.7 and 7.8, it can be seen that the two figures are identical except for the polarities of the input variable x_2. It should be noted that due to Property 4.2.14, for $\theta\in\{a,b\}$ the described relation between the modules for $M_n^\theta(\varphi,\sigma,0)$ and $M_n^\theta(\varphi,n+1,p)$ always applies as long as their types are of the same, whereas for $\theta\in\{c,d\}$ the ordering φ of the two transforms also have to be the same.

The multi level tree implementations for fastest LIA transforms $T_n^\theta(\varphi,\sigma,p)$ can be derived in the same manner. As in the case for the linear systolic array processor

structure, for $\theta \in \{a,b\}$ the circuit realization for the fastest LIA transform $T_n^{\theta^*}(\varphi^*,\sigma^*,p)$ is simply the module for fastest LI transform over GF(2) $M_n^{\theta}(\varphi,\sigma,p)$ with all XOR operations replaced by arithmetic additions, where the definitions for θ^*, σ^*, and φ^* follow Section 7.1.

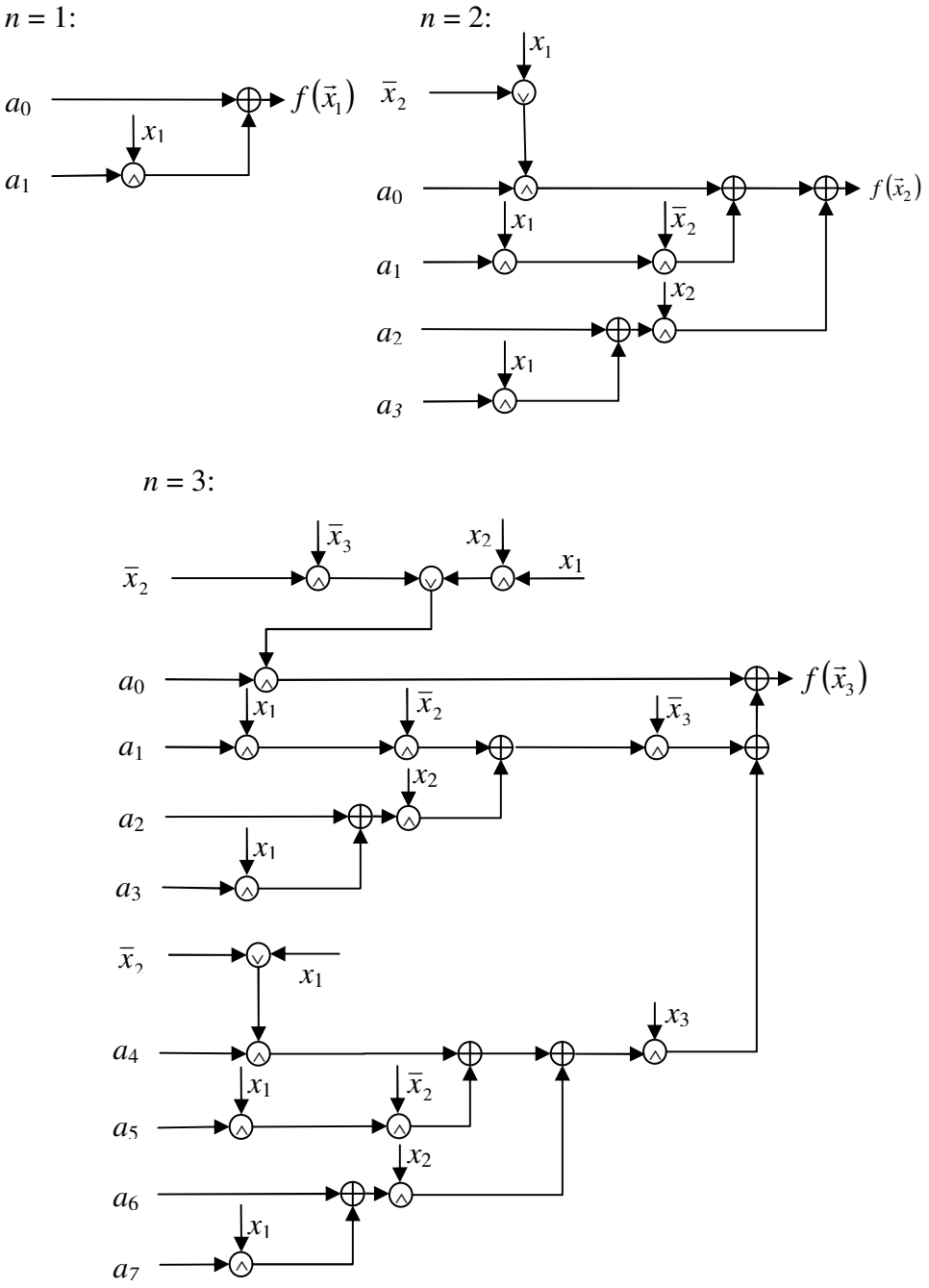


Figure 7.6: Multi level tree for fastest LI transform $M_n^{a}(\varphi,\sigma,0)$.

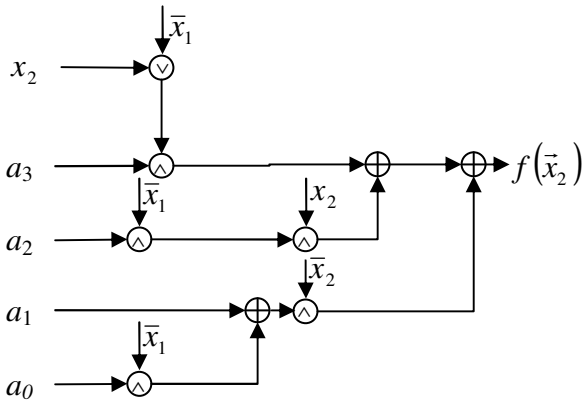


Figure 7.7: Multi level tree for two-variable fastest LI transform $M_2^b(\varphi,\sigma,0)$.

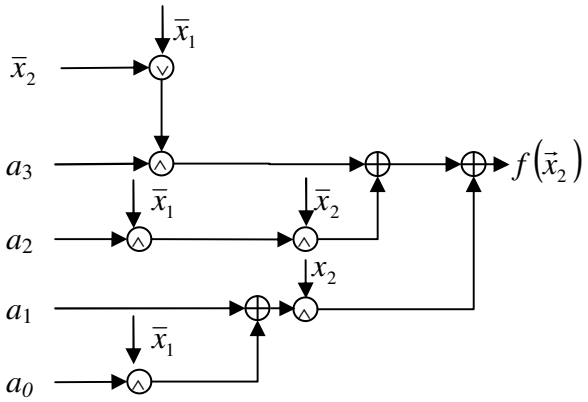


Figure 7.8: Multi level tree for two-variable fastest LI transform $M_2^b(\varphi,n+1,2)$.

7.2.2 Ternary polynomial expansions based on fastest LITA transforms and fastest TLI transforms

In a similar manner as for the binary functions, the modular multi level tree structures that directly correspond to the hardware implementation of polynomial expansion based on the fastest LITA transforms and fastest TLI transforms can be derived. In Fig. 7.9, the modules for implementation of LITA polynomial expansion based on $T_n(0)$ is shown for n = 1 and 2, where $\oplus$ and $\otimes$ represent two-input arithmetic adder

and multiplier, respectively and $X_i^{S_i}$ is a ternary literal as given in Definition 5.2.1.

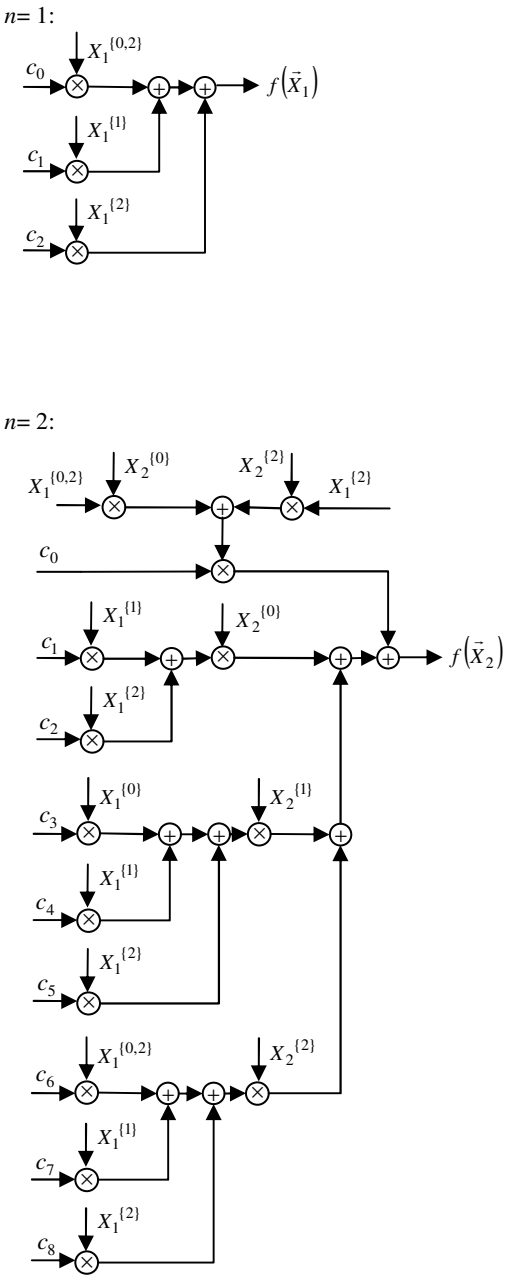


Figure 7.9: Multi level tree for fastest LITA transform $T_n(0)$.

It can be seen from Fig. 7.9 that the two-variable module for $T_n(0)$ can be built from two one-variable modules for $T_n(0)$ with one of them being slightly modified and another additional module that corresponds to the spectral coefficients $c_{3^{n-1}}$ to $c_{(2\cdot 3^{n-1})-1}$. This relation between the one-variable and two-variable modules for $T_n(0)$ also applies for higher number of input variables. Let $f(\vec{X}_n)$ and $f(\vec{X}_{n-1})$ denote the n-variable and $(n-1)$-variable modules for $T_n(0)$, respectively ($n \geq 2$) while $f_0(\vec{X}_{n-1})$ denotes the term that corresponds to c_0 in $f(\vec{X}_{n-1})$. Then, the circuit realization for $f(\vec{X}_n)$ is synthesized by the following principle:

$$f(\vec{X}_n) = f^0(\vec{X}_n) + g(\vec{X}_n) + f^2(\vec{X}_{n-1})X_n^{\{2\}}, \tag{7.2}$$

where :

- $f^0(\vec{X}_n) = \left(f(\vec{X}_{n-1}) - c_0 f_0(\vec{X}_{n-1})\right)X_n^{\{0\}} + c_0\left(f_0(\vec{X}_{n-1})X_n^{\{0\}} + X_n^{\{2\}}X_{n-1}^{\{2\}}...X_1^{\{2\}}\right)$,
- $f^2(\vec{X}_{n-1})$ is simply $f(\vec{X}_{n-1})$ with all spectral coefficients c_i $(0 \leq i \leq 3^{n-1}-1)$ replaced with $c_{i+2\cdot 3^{n-1}}$,
- $g(\vec{X}_n)$ is a multi level submodule that is built from smaller submodules $g(\vec{X}_{n-1})$ by

$$g(\vec{X}_n) = X_n^{\{1\}}\left(g^0(\vec{X}_{n-1}) + g^1(\vec{X}_{n-1}) + g^2(\vec{X}_{n-1})\right). \tag{7.3}$$

In (7.3), $g^u(\vec{X}_{n-1})$, $u \in \{0,1,2\}$ represents $g(\vec{X}_{n-1})$ with $X_{n-1}^{\{1\}}$ replaced by $X_{n-1}^{\{u\}}$ and spectral coefficients c_i $(3^{n-2} \leq i \leq 2\cdot 3^{n-2}-1)$ replaced with $c_{i+(2+u)\cdot 3^{n-2}}$. From Fig. 7.9, it can be seen that $g(\vec{X}_1) = X_1^{\{1\}}$.

The modular multilevel tree structure for $T_n(0)$ given above can also be used to implement LITA polynomial expansion based on any other fastest LITA transform $T_n(p)$ with only rearrangement of the input variables. Let matrix R be as defined in Property 6.2.7. Then the n-variable module for $T_n(p)$ can be obtained from the n-variable module for $T_n(0)$ by replacing every $X_i^{S_i}$ $(1 \leq i \leq n)$ in the $T_n(0)$ module with $X_i^{S_i'}$, where S_i' is S_i with each element s inside it replaced with $R_{p_i,s}$. Recall

that $< p_n, p_{n-1}, ..., p_1 >$ and $R_{p_i,s}$ denote the n-digit ternary representation of p and the element that is located at row p_i $(0 \leq p_i \leq 5)$ and column s $(0 \leq s \leq 2)$ of matrix R, respectively. Fig. 7.10 shows the two-variable module for $T_n(p)$ when $p = 9 = <1, 3>_6$ which is derived from the module in Fig. 7.9 for $n = 2$.

As the fastest LITA transforms $T_n(p)$ and fastest TLI transforms $M_n(p)$ have the same forward transforms, and therefore basis functions, the multi level tree implementation for $M_n(p)$ has the same structure as the multi level tree implementation for $T_n(p)$. The difference is only that for $M_n(p)$ the additions and multiplications are performed over GF(3).

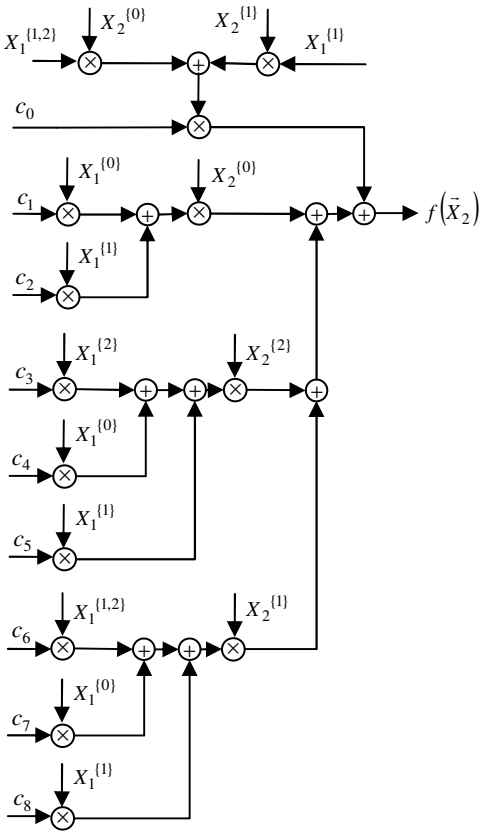


Figure 7.10: Multi level tree for fastest LITA transform $T_2(9)$.

7.3 Linear systolic array structure for row and column polarity matrix algorithms

In Chapters 3 and 5, the recursive polarity matrix algorithms called row and column polarity matrix have been presented for the computation of FPRMEs over GF(5) as well as FPAEs for ternary and quaternary functions. All the algorithms generate the complete polarity matrix or selected spectral coefficient vector in n recursion stages, where in each recursion stage the same arithmetic operations (additions/subtractions and multiplications) are performed for different submatrices of the polarity matrix of the input functions. Flow graphs for the computations inside each recursion stages of the different algorithms have been given. Alternatively, the operations inside each recursion stage can also be performed by linear systolic array similar to that presented in [Yan94]. Here the linear systolic array structure for the calculation of selected FPRME spectral coefficient vector over GF(5) by column polarity matrix algorithm is shown as an example. The linear systolic array for other algorithms will be based on the same principle. Only the number of stages, input and output connections, and stored coefficients need to be adjusted accordingly.

The linear systolic array processor structure for obtaining the selected FPRME over GF(5) spectrum by column polarity matrix algorithm consists of four stages, where each stage has one computing location (CL) and two storage locations (SL1 and SL2). The first storage location (SL1) is used to store the appropriate multiplication coefficients whereas the second storage location (SL2) is a FIFO register that stores the intermediate and final computation results. In each recursion stage of the algorithm, the inputs that are fed to the first stage of the linear systolic array processor structure are the first column elements of the currently processed H_β matrix, i.e., first column elements of h_{00}, h_{10}, h_{20}, h_{30}, and h_{40} (refer to (3.54)) whereas the SL2 of the first, second, third, and fourth stage stores the intermediate and final results for $h_{j_\theta 1}$, $h_{j_\theta 2}$, $h_{j_\theta 3}$, and $h_{j_\theta 4}$, respectively, where j_θ is the row index number of the required submatrices. Figs. 7.11 and 7.12 show the structure of the linear systolic array processor and the CL, respectively [Yan94]. It can be seen from Figs. 7.11 and 7.12

that every h_{in} input fed to a CL is both copied to internal register (*Reg_h*) inside of CL and passed to the CL of the next stage. The copied value of the input is then multiplied by a multiplication coefficient fed from SL1 and added to the content of the *Reg_Σ*, which is obtained from SL2. The result is then stored back into SL2.

Due to the structure of the polarity matrix, the stored multiplier coefficients inside SL1 can be the same for any value of j_θ. Only the ordering of the inputs need to be adjusted such that the inputs are fed in the order of $h_{r_0 0}$, $h_{r_1 0}$, $h_{r_2 0}$, $h_{r_3 0}$, and $h_{r_4 0}$ (recall that $r_y = j_\theta + y$ over GF(5)). Since the size of first column elements of h_{yz} ($0 \le y, z \le 4$) of H_β matrix is $5^{\beta-1}$, based on the given steps to calculate FPRME spectral coefficients in polarity ω by the column polarity matrix algorithm, the SL2s of the systolic processor should have 5^{i-1} registers in the recursion stage where $\beta = i$. Furthermore, the multiplication coefficient values from SL1 applied for $h_{r_0 0}$, $h_{r_1 0}$, $h_{r_2 0}$, $h_{r_3 0}$, and $h_{r_4 0}$ elements should be 0, 1, 3, 2, and 4, respectively for the first stage and 0, 4, 1, 1, 4, respectively for the second stage. For the third and fourth stages the corresponding multiplication coefficient values should be 0, 1, 2, 3, 4 and 4, 4, 4, 4, 4, respectively

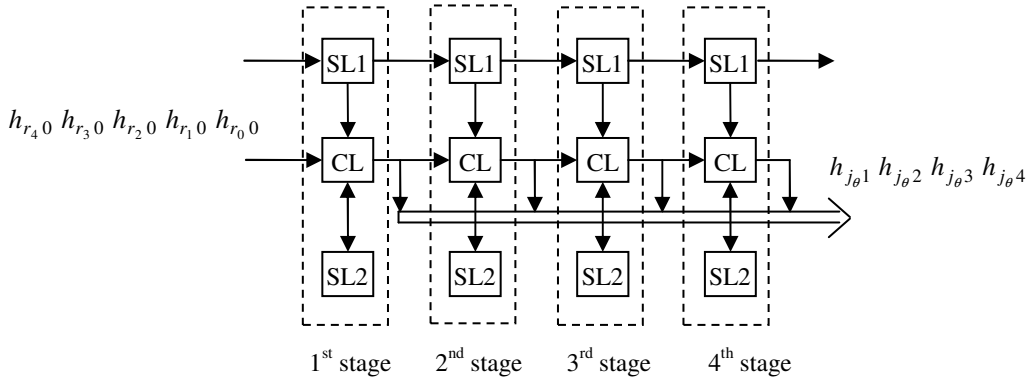


Figure 7.11: Systolic array structure for the calculation of row j_θ matrices.

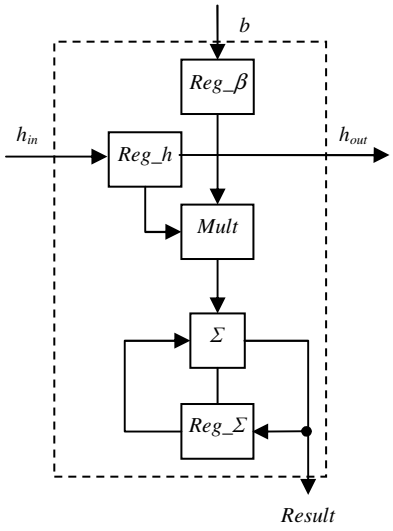


Figure 7.12: CL structure.

.

Chapter 8

Conclusion and Recommendations

8.1 Conclusion

This thesis is concerned with novel spectral representations for discrete functions as well as their useful properties and efficient computation. The focus has been on the spectral transforms for binary, ternary, quaternary, and five-valued functions, which form the smallest fields of multiple-valued discrete functions and also the most common. In particular, the work done concentrated on the transforms derived based on the concept of fast linearly independent transforms as well as the generalization of arithmetic and RM transforms through Galois field and standard arithmetic operators. It has been shown that the proposed transforms provide compact representations for discrete functions and the presented properties and algorithms contribute to their fast and efficient computation.

Several algorithms for the generation of FPRMEs over GF(5) have been proposed. Computational costs for these algorithms in terms of the number of necessary additions and multiplications over GF(5) have also been derived. It should be noted that it is important to find the computational cost of a given method since such a cost is one of the indicators of the method efficiency. All the algorithms utilize the derived relations between the truth vector representation of a five-valued function and its FPRME representation in different polarities. As a result, other than the

disjoint cube polarity algorithm, they are all generating the complete polarity matrix by first calculating certain elements of the polarity matrix from the input and then deriving the rest of the elements in a certain order from the known elements. Among the algorithms that take the truth vector or arbitrary FPRME spectrum as an input, the matrix multiplication algorithm is the simplest. However, at every steps of the calculation it needs to store two full FPRME vectors. In comparison, the extended dual polarity algorithm has similar operating principle as the matrix multiplication algorithms, but by only storing the information about the nonzero terms, it may require smaller amount of storage space. On the other hand, it has been found that from the computational cost and time viewpoints, the row and column polarity matrix algorithms are the most efficient for deriving both the complete polarity matrix and the FPRME spectrum in selected polarity. Due to the recursive nature of the polarity matrix algorithms, their execution time can be further improved by implementing them using parallel programming.

Unlike the rest of the algorithms, the cube polarity adjustment algorithm takes disjoint array of cubes representation of the input function and it does not involve calculation of spectral coefficients from other spectral coefficients. Instead it obtains the spectral coefficients by determining the contributions of each input minterms or cubes to the spectral coefficients values and adding them up. Hence, this algorithm can also be run using parallel programming as the value of each spectral coefficient can be calculated independently of other spectral coefficients. This together with the fact that for most cases the number of disjoint cubes is less than number of minterms gives rise to smaller memory requirement for this algorithm which implies that this algorithm can be used to calculate spectral coefficients of functions with larger number of input variables than what the other algorithms can handle. Since the execution time of the algorithm depends heavily on the number of disjoint cubes and rises rapidly with the increase in the disjoint cubes number, the algorithm is especially useful when only selected spectral coefficients are required, as is the case for testing and decomposition [Hei91] and when the input function is represented by only few disjoint cubes. In short, the algorithms differ in their computational costs, storage requirements, and possibility of parallel implementation. Hence the final choice of the most suitable

algorithm to use depends on the function, the available resources, and the required execution time.

New polynomial expansions over GF(2) that have the fastest and most efficient logic transformations have been introduced and identified. The new LI transforms are generated by reordering the butterfly diagram stages of the existing fastest LI transforms and inserting a permutation matrix in front of, behind or inside the butterfly diagrams. Due to their way of generation, the newly introduced fastest LI transforms have the same numbers of computational cost as the existing ones and possess regular structure. The given experimental results show that the fastest LI transforms are able to give polynomial expansions with less number of nonzero spectral coefficients than those based on the existing ones for the majority of binary benchmark functions. Relations between different fastest LI transform matrices have also been examined and formulae for determining the number and location of 1s in a fastest LI transform matrix have been given. Due to the presence of these properties, only some of the fastest LI transform matrices need to be actually derived by matrix multiplications. The rest can be generated from those matrices by simple rotation or permutation. Groups of fastest LI matrices that are identical to each other have also been identified, which further reduces the computational cost involved in generating all unique representations of a binary function based on them. The polynomial expansions over GF(2) based on the fastest LI matrices combined with other techniques described in [PFC$^+$02, SSC$^+$94] may be efficiently used in mapping to fine grain and cellular automata types of FPGAs. They can also be easily implemented in terms of reversible logic gates [Al-04]. The design of the next generation of fine grain architectures should also be influenced by the efficient bases of LI logic transformations introduced here, as the required hardware implementation can be calculated efficiently by fast transforms and with the combination of methods proposed in [PFC$^+$02, SSC$^+$94], it can use minimal number of basic cells with a compact routing. As a result, it also has the advantages of high speed and area minimization. The choice of a particular transform matrix and expansion is determined by the properties of the particular problem and functions. This eventually reduces to a problem of selecting an optimal transform basis that will give a polynomial expansion that has minimal number of nonzero

coefficients. With the introduction of the classes of fastest transforms, it would open more choices in the design of logical circuits while taking into consideration the tradeoff between computational efficiency, final implementation, and minimal number of nonzero coefficients. The advantage in terms of fastest computation can be also used in other applications of introduced LI transforms such as image processing of standard and biomedical images and nonlinear digital filtering where RM transform has already been used [AAE95, Fal04].

Fastest LIA transforms for binary functions, which are the extension of fastest LI transforms to arithmetic domain, have also been dealt with. New fastest LIA transforms are introduced that can be created efficiently and their matrix can be extended further to higher dimensions. Relations and properties between different matrices are also discussed and the possibility of generating the spectrum of a fastest LIA transform directly from the known spectrum of some other fastest LIA transforms has been indicated. As there exist a huge number of different LIA expansions, there is a great interest in finding such LIA expansions that have fast transforms and equations. Thus, the discussion in this thesis has been restricted only to those fastest LIA transforms that possess fast forward and inverse fastest LIA transforms. The fast transforms formulae are provided, allowing the transform matrices to be extended to higher dimensions, giving the best sets of basis functions for such cases as well. Besides the relations between fastest LIA transforms, other properties on the number and location of nonzero elements in the fastest LIA transforms have also been investigated, which have then been applied to obtain several bounds of the fastest LIA transform spectra. The derived bounds can be used to consider the suitability of the fastest LIA transforms for various fault detection and function verification applications as well as for development of decision diagrams in a similar manner as arithmetic transform has already been used [Che92, Hei91, KB81, KSZ90, KSY91, Mal97, PM75, RZ04, SA03, SF96, YSF01]. Due to the importance of arithmetic transform for functional verification of circuits under the error modeling of their spectra, the presented transform properties and advantages of spectra for fastest LIA transforms can be very useful in these applications.

Manipulations and calculations of discrete functions, for example, analysis of

satisfiability, tautology, equivalence, classification, is a fundamental task in Computer Aided Design. They range from synthesis, verification, testing, library matching and they can be efficiently performed when original logic domain is converted to standard arithmetic operations. FPAE is one such representation that operates in standard arithmetic algebra. In this thesis, algorithms that can be used to efficiently calculate the FPAEs have been presented for ternary and quaternary functions. The algorithms are classified into matrix multiplication, cube polarity adjustment, and row and column polarity matrix algorithms. The matrix multiplication algorithms need to be performed sequentially whereas the cube polarity adjustment and polarity matrix algorithms can be implemented by parallel programming. The computational costs for the algorithms have been derived and it has been found that the polarity matrix algorithms, which can be used to generate the spectra of either selected or all FPAEs, have the smallest computational costs. For ternary FPAE, the column polarity matrix algorithm is the most efficient for finding both partial and complete polarity matrix whereas for quaternary FPAE, the column polarity matrix is more efficient for generating selected spectral coefficient vector but it requires more multiplication operations than row polarity matrix algorithm for obtaining the full polarity matrix. As the algorithms operate in similar manner to the corresponding algorithms for FPRME, the advantages mentioned for the FPRME algorithms also apply for FPAE. The resulting FPAEs can be used as the mathematical apparatus to analyze the stability of finite automata giving more flexibility than the known results for binary dynamic systems as well as the bases of new ternary and quaternary word decision diagrams in a manner similar to the ones developed in [SA03].

The concept of generalized fastest LI transforms through operations on the existing fastest LI transforms has been extended for ternary functions. Thus, new fastest TLI and LITA transforms are introduced. Similar to the binary case, the transforms have regular structure. As a result, properties relating to the relations that exist between them as well as the number and location of nonzero elements inside the matrices can be derived. Such properties simplify the computation of the optimal expansion based on them using software in addition to be helpful in analyzing the correlation between their spectra and the properties of the underlying circuit realization. The transforms

can also be easily calculated by fast transform and the calculation have been shown to incur smaller number of arithmetic operations than FPRME over GF(3) and ternary FPAE, respectively. Similar to other polynomial expansions based on binary and multiple-valued logic, the new transforms can have applications in spectral representations of ternary logic functions, computation of the stochastic behavior, and as bases of new ternary spectral decision diagrams.

The applicability of a transform depends not only on the property of the transform, but also on the availability of efficient methods for computation and manipulation of their spectra. Thus, hardware calculation of the introduced binary and ternary LI transforms spectra have been shown using linear systolic array processor, which is a modern hardware structure with high degree of efficiency and computational speed. The systolic processor structure follows closely the butterfly diagram of the respective transform and therefore the relations between the transforms presented in the preceding chapters can be easily applied to map the spectra calculation of a set of different fastest transforms to the same systolic structures. It is also shown that in some cases the reordering and permutation in the fastest LI matrices reduce the amount of storage cells required to calculate their spectra by systolic processor, which results in hardware cost saving. The realizations of their resulting polynomial expansions using a multi level tree modular implementation have also been discussed and the presented relations between the different transforms have also been proved useful to allow a circuit realization to be shared between related transforms with only reordering of inputs/outputs.

8.2 Recommendations for further research

A number of new open research problem arise from the developments presented in this thesis. Some of the possible future research directions are listed below.

1. Exploring the possibility of applying the same concept employed by the row and column polarity matrix algorithm for other spectral transforms and analyzing their

effectiveness for different transforms.

2. Application of the proposed fastest LIA and LITA transforms for function verification, testing, and fault detection.

3. Generation of decision diagrams based on the transforms discussed in this thesis and analysis of their efficiency.

4. Development of new fastest LI transforms for quaternary functions and their properties in a similar manner as the binary and ternary fastest LI transforms.

5. Analysis of the testability property of the FPAE transforms and their applications in the areas where arithmetic transforms have been used.

Author's Publications

Journals

[1] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Calculation of best fixed polarity Reed-Muller transform over GF(5). *IEICE Electronics Express*, 1(5): 95–97, June 2004.

[2] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Algorithm to generate fixed polarity Reed-Muller GF(5) spectra from disjoint cubes. *IEICE Electronics Express*, 1(6):131–136, June 2004.

[3] B. J. Falkowski and C. C. Lozano. Quaternary fixed polarity Reed-Muller expansion computation through operations on disjoint cubes and its comparison with other methods. *Computers and Electrical Engineering, An International Journal*, 31(2):112–131, March 2005.

[4] S. Rahardja, B. J. Falkowski, and C. C. Lozano. Fastest linearly independent transforms over GF(2) and their properties. *IEEE Transactions on Circuits and Systems – I*, 52(9):1832–1844, September 2005.

[5] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Column polarity matrix algorithm for ternary fixed polarity Reed-Muller expansions. *Journal of Circuits, Systems, and Computers*, 15(2):243–262, April 2006.

[6] C. C. Lozano and B. J. Falkowski. Fastest linearly independent arithmetic transforms and their calculation on systolic array processors. *IET Circuits, Devices, and Systems*, 1(2):181–192, April 2007.

[7] C. C. Lozano, B. J. Falkowski, and S. Rahardja. Properties and relations for fast linearly independent arithmetic transforms. *IET Signal Processing*, 1(2):82–95, June 2007.

[8] B. J. Falkowski, C. C. Lozano, and T. Łuba. Family of fastest linearly independent transforms over GF(3): generation, relations, and hardware implementation. *Journal of Multiple-Valued Logic and Soft Computing*. Accepted for publication.

Conferences

[1] B. J. Falkowski and C. C. Lozano. Properties of fastest LIA transform matrices and their spectra. In *Proceedings of the 36th IEEE International Symposium on Circuits and Systems*, pages 556–559, Bangkok, Thailand, May 2003.

[2] B. J. Falkowski and C. C. Lozano. Generation and properties of fastest transform matrices over GF(2). In *Proceedings of the 36th IEEE International Symposium on Circuits and Systems*, pages 740–743, Bangkok, Thailand, May 2003.

[3] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Fast calculation of ternary fixed-polarity Reed-Muller transform using column polarity matrix. In *Proceedings of the 16th European Conference on Circuit Theory and Design*, pages 357–360, Krakow, Poland, September 2003.

[4] B. J. Falkowski and C. C. Lozano. Calculation of quaternary fixed-polarity Reed-Muller expansion from disjoint cubes. In *Proceedings of the 4th South Eastern Europe Workshop on Computational Intelligence and Information Technologies*, pages 99–102, Nis, Serbia, October 2003.

[5] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Generation of disjoint cubes for multiple-valued functions. In *Proceedings of the 37th IEEE International Symposium on Circuits and Systems*, pages 133–136, Vancouver, Canada, May 2004.

[6] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Fast optimization of fixed-polarity Reed-Muller expansions over GF(5). In *Proceedings of the 34th IEEE International Symposium on Multiple-Valued Logic*, pages 162–167, Toronto, Canada, May 2004.

[7] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Spectra generation for fixed-polarity Reed-Muller transform over GF(5). In *Proceedings of the 34th IEEE International Symposium on Multiple-Valued Logic*, pages 177–183, Toronto, Canada, May 2004.

[8] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Calculation of fixed-polarity Reed-Muller spectra over GF(5) from disjoint cubes of five-valued functions. In *Proceedings of the 13th International Workshop on Post Binary Ultra-Large Scale Integration Systems*, pages 3–7, Toronto, Canada, May 2004.

[9] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Generation of fixed-polarity Reed-Muller transform over GF(5) from disjoint cubes. In *Proceedings of the 47th IEEE Midwest Symposium on Circuits and Systems*, pages 217–220, Hiroshima, Japan, July 2004.

[10] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Efficient calculation of fixed-polarity Reed-Muller expansions over GF(5) using extended dual polarity

property. In *Proceedings of the 47th IEEE Midwest Symposium on Circuits and Systems*, pages 221–224, Hiroshima, Japan, July 2004.

[11] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Algorithms for calculation of fixed polarity Reed-Muller expansions over GF(5). In *Proceedings of the 4th TICSP International Workshop on Spectral Methods and Multirate Signal Processing*, pages 101–105, Vienna, Austria, September 2004.

[12] B. J. Falkowski and C. C. Lozano. Efficient calculation of fixed polarity Reed-Muller transform over GF(4) using disjoint cubes. In *Proceedings of the 10th International Symposium on IC Technology, Systems and Applications*, CD publication, Singapore, September 2004.

[13] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Generalized fastest linearly independent arithmetic transforms. In *Proceedings of the 38th IEEE International Symposium on Circuits and Systems*, pages 480–484, Kobe, Japan, May 2005.

[14] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Generation and properties of new fastest linearly independent transforms over GF(2) with reordering. In *Proceedings of the 38th IEEE International Symposium on Circuits and Systems*, pages 4705–4708, Kobe, Japan, May 2005.

[15] C. C. Lozano, B. J. Falkowski, and S. Rahardja. Binary polynomial expansions based on new fastest linearly independent transforms. In *Proceedings of the 14th International Workshop on Post Binary Ultra-Large Scale Integration Systems*, pages 18–21, Calgary, Canada, May 2005.

[16] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Recursive algorithm for generation of fixed polarity Reed-Muller expansions over GF(5). In *Proceedings of the 48th IEEE Midwest Symposium on Circuits and Systems*, pages 191–194, Cincinnati, OH, USA, August 2005.

[17] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Hardware implementation for calculation of fastest linearly independent spectra over GF(2). In *Proceedings of the 48th IEEE Midwest Symposium on Circuits and Systems*, pages 1027–1030, Cincinnati, OH, USA, August 2005.

[18] C. C. Lozano, B. J. Falkowski, and S. Rahardja. Algorithms for generation of quaternary fixed polarity arithmetic spectra. In *Proceedings of the 39th IEEE International Symposium on Circuits and Systems*, pages 803–806, Kos, Greece, May 2006.

[19] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Efficient computation of fixed polarity arithmetic expansions for ternary functions. In *Proceedings of the 39th IEEE International Symposium on Circuits and Systems*, pages 2409–2412, Kos, Greece, May 2006.

[20] C. C. Lozano, B. J. Falkowski, and S. Rahardja. Properties and hardware implementation of new fastest linearly independent ternary arithmetic transforms. In *Proceedings of the 15th International Workshop on Post Binary Ultra-Large Scale Integration Systems*, pages 7–10, Singapore, May 2006.

[21] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Generalized fastest LIA transform spectra calculation by systolic processor. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2017–2021, Seattle, WA, USA, July 2006.

[22] C. C. Lozano and B. J. Falkowski. Generation of fixed polarity arithmetic spectra for ternary functions. In *Proceedings of the 8th IEEE Asia Pacific Conference on Circuits and Systems*, pages 1332–1335, Singapore, December 2006.

[23] C. C. Lozano and B. J. Falkowski. Fixed polarity arithmetic expansions calculation from disjoint cubes representation of ternary functions. In *Proceedings of the 8th IEEE Asia Pacific Conference on Circuits and Systems*, pages 1625–1628, Singapore, December 2006.

[24] B. J. Falkowski, C. C. Lozano, and S. Rahardja. Disjoint cubes generation algorithm for multiple-valued functions. In *Proceedings of the 8th IEEE Asia Pacific Conference on Circuits and Systems*, pages 1629–1632, Singapore, December 2006.

[25] B. J. Falkowski, C. C. Lozano, and T. Łuba. Efficient algorithm for calculation of quaternary fixed polarity arithmetic expansions. In *Proceedings of the 37th IEEE International Symposium on Multiple-Valued Logic*, CD Publication, Oslo, Norway, May 2007.

[26] B. J. Falkowski, C. C. Lozano, and T. Łuba. New fastest linearly independent transforms over GF(3). In *Proceedings of the 37th IEEE International Symposium on Multiple-Valued Logic*, CD Publication, Oslo, Norway, May 2007.

[27] B. J. Falkowski, C. C. Lozano, and T. Łuba. Properties and relations of new fastest linearly independent arithmetic transforms for ternary functions. In *Proceedings of the 15th European Signal Processing Conference*, Poznan, Poland, September 2007. Accepted for publication.

Bibliography

[AAE95] S. Agaian, J. Astola, and K. Egiazarian. *Binary Polynomial Transforms and Nonlinear Digital Filters*. New York: Marcel Dekker, 1995.

[AB96] A. E. A. Almaini and K. Burnside. Generalized Reed-Muller ASIC converter. In *Proceedings of the 2nd International Conference on ASIC*, pages 73–76, Shanghai, China, October 1996.

[Abo01] S. Aborhey. Reed-Muller tree-based minimisation of fixed polarity Reed-Muller expansions. *IEE Proceedings-Computers and Digital Techniques*, 148(2):63–70, March 2001.

[Al-04] A. N. Al-Rabadi. *Reversible Logic Synthesis: From Fundamentals to Quantum Computing*. New York: Springer-Verlag, 2004.

[Alm94] A. E. A. Almaini. *Electronic Logic Systems*. New York: Prentice Hall, 1994.

[AS06] J. T. Astola and R. S. Stankovic. *Fundamentals of Switching Theory and Logic Design: A Hands on Approach*. Dordrecht: Springer, 2006.

[Bea84] K. G. Beauchamp. *Applications of Walsh and Related Functions*. London: Academic Press, 1984.

[BGJ+02] R. K. Brayton, M. Gao, J.-H. R. Jiang, Y. Jiang, Y. Li, A. Mishchenko, S. Sinha, and T. Villa. Optimization of multi-valued multi-level networks. In *Proceedings of the 32nd IEEE International Symposium on Multiple-Valued Logic*, pages 168–177, Boston, MA, USA, May 2002.

[Boo54] G. Boole. *The Laws of Thought*. London: Macmillan, 1854.

[Bro90] F. M. Brown. *Boolean Reasoning*. Boston: Kluwer Academic, 1990.

[Bry86] R. E. Bryant. Graph-based algorithm for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[Bry91] R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):206–213, February 1991.

[Cas96] K. R. Castleman. *Digital Image Processing*. Englewood Cliffs: Prentice-Hall, 1996.

[Che92] T. H. Chen. *Fault Diagnosis and Fault Tolerance: Systematic Approach to Special Topics*. Berlin: Springer-Verlag, 1992.

[CW83] X. Chen and X. Wu. The synthesis of ternary functions under fixed polarities and ternary I^2L circuits. In *Proceedings of the 13th IEEE International Symposium on Multiple-Valued Logic*, pages 424–429, Kyoto, Japan, May 1983.

[CYP89] M. J. Ciesielski, S. Yang, and M. A. Perkowski. Multiple-valued minimization based on graph coloring. In *Proceedings of IEEE International Conference on Computer Design*, pages 262–265, Cambridge, MA, USA, October 1989.

[Dam92] T. Damarla. Generalized transforms for multiple valued circuits and their fault detection. *IEEE Transactions on Computers*, 41(9):1101–1109, September 1992.

[DDT78] M. Davio, J. P. Deschamps, and A. Thayse. *Discrete and Switching Functions.* New York: Georgi and McGraw-Hill, 1978.

[DK89] T. R. Damarla and M. Karpovsky. Fault detection in combinational networks by Reed-Muller transforms. *IEEE Transactions on Computers*, 38(6):788–797, June 1989.

[DM00] E. V. Dubrova and J. C. Muzio. Easily testable multiple-valued logic circuits derived from Reed-Muller circuits. *IEEE Transactions on Computers*, 49(11):1285–1289, November 2000.

[DTB96] R. Drechsler, M. Theobald, and B. Becker. Fast OFDD-based minimization of fixed polarity Reed-Muller expressions. *IEEE Transactions on Computers*, 45(11):1294–1299, November 1996.

[Edw75] C. R. Edwards. The application of Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis. *IEEE Transactions on Computers*, 24(1):48–62, January 1975.

[EM86] E. Eris and J. C. Muzio. Spectral testing of circuit realizations based on linearizations. *IEE Proceedings-Computers and Digital Techniques*, 133(2):73–78, March 1986.

[Eps93] G. Epstein. *Multiple-Valued Logic Design: An Introduction*. Bristol: Institute of Physics Publishing, 1993.

[Fal99] B. J. Falkowski. A note on the polynomial form of Boolean functions and related topics. *IEEE Transactions on Computers*, 48(8):860–864, August 1999.

[Fal02] B. J. Falkowski. Algorithms for fast arithmetic transform. In *Proceedings of the 35th International Symposium on Circuits and Systems,* pages 753–756, Scottsdale, AZ, USA, May 2002.

[Fal02a] B. J. Falkowski. Calculation of Walsh transform on MASPAR computer. In *Proceedings of the 35th IEEE International Symposium on Circuits and Systems*, pages 771–774, Scottsdale, AZ, USA, May 2002.

[Fal04] B. J. Falkowski. Compact representations of logic functions for lossless compression of Grey scale images. *IEE Proceedings-Computers and Digital Techniques*, 151(3):221–230, May 2004.

[FC94] B. J. Falkowski and C. H. Chang. Efficient algorithms for the calculation of arithmetic spectrum from OBDD and synthesis of OBDD from arithmetic spectrum for incompletely specified Boolean functions. In *Proceedings of the 27th IEEE International Symposium on Circuits and Systems*, pages 197–200, London, United Kingdom, May 1994.

[FC97] B. J. Falkowski and C. H. Chang. Calculation of arithmetic spectra from free binary decision diagrams. In *Proceedings of the 30th International Symposium on Circuits and Systems*, pages 1764–1767, Hong Kong, June 1997.

[FC99] B. J. Falkowski and C. H. Chang. An efficient algorithm for the calculation of generalized arithmetic and adding transforms from disjoint cubes of Boolean functions. *VLSI Design*, 9(2):135–146, April 1999.

[FF03] B. J. Falkowski and C. Fu. Fastest linearly independent arithmetic transforms over GF(3). In *Proceedings of the 28th IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 669–672, Hong Kong, April 2003.

[FF04] B. J. Falkowski and C. Fu. Family of fast linearly independent ternary arithmetic transforms. *IEEE Transactions on Circuits and Systems – I,* 51(6):1132–1147, June 2004.

[FF05] B. J. Falkowski and C. Fu. Classification of new linearly independent transforms over GF(3). *Journal of Circuits, Systems, and Computers*, 14(2):395–421, April 2005.

[FF05a] B. J. Falkowski and C. Fu. Classes of fastest quaternary linearly independent transformations. In *Proceedings of the 35th IEEE International Symposium on Multiple-Valued Logic*, pages 206–211, Calgary, Canada, May 2005.

[FF05b] B. J. Falkowski and C. Fu. Generation of linearly independent transforms over GF(4). In *Proceedings of the 38th IEEE International Symposium on Circuits and Systems*, pages 488–491, Kobe, Japan, May 2005.

[FF05c] B. J. Falkowski and C. Fu. Fastest classes of linearly independent transforms over GF(3) and their properties. *IEE Proceedings-Computers and Digital Techniques*, 152(5):567–576, September 2005.

[FF06] B. J. Falkowski and C. Fu. Properties and experimental results of fastest linearly independent ternary arithmetic transforms. *IEEE Transactions on Circuits and Systems – I*, 53(4):858–866, April 2006.

[FHH+93] B. Fei, Q. Hong, W. Haomin, M. A. Perkowski, and N. Zhuang. Efficient computation for ternary Reed-Muller expansions under fixed polarities. *International Journal of Electronics,* 75(4):685–688, October 1993.

[Fis74] L. T. Fisher. Unateness properties of AND-EXCLUSIVE-OR logic circuits. *IEEE Transactions on Computers*, 23(2):166–172, February 1974.

[FL03] B. J. Falkowski and C. C. Lozano. Properties of fastest LIA transform matrices and their spectra. In *Proceedings of the 36th IEEE International Symposium on Circuits and Systems*, pages 556–559, Bangkok, Thailand, May 2003.

[FL03a] B. J. Falkowski and C. C. Lozano. Generation and properties of fastest transform matrices over GF(2). In *Proceedings of the 36th IEEE International Symposium on Circuits and Systems*, pages 740–743, Bangkok, Thailand, May 2003.

[FP90] B. J. Falkowski and M. A. Perkowski. A family of all essential radix-2 addition/subtraction multi-polarity transforms: algorithms and interpretations in Boolean domain. In *Proceedings of the 23rd IEEE International Symposium on Circuits and Systems*, pages 2913–2916, New Orleans, LA, USA, May 1990.

[FP91] B. J. Falkowski and M. A. Perkowski. One more way to calculate generalized Reed-Muller expansions of Boolean functions. *International Journal of Electronics,* 71(3):385–396, September 1991.

[FR95] B. J. Falkowski and S. Rahardja. Efficient computation of quaternary fixed polarity Reed-Muller expansions. *IEE Proceedings-Computers and Digital Techniques*, 142(5):345–352, September 1995.

[FR97] B. J. Falkowski and S. Rahardja. Algorithms for fast Reed-Muller transform. In *Proceedings of the 30th IEEE International Symposium on Circuits and Systems*, pages 2669–2672, Hong Kong, June 1997.

[FR97a] B. J. Falkowski and S. Rahardja. Classification and properties of fast linearly independent logic transformations. *IEEE Transactions on Circuits and Systems - II*, 44(8):646–655, August 1997.

[FR01] B. J. Falkowski and S. Rahardja. PLI logic for multiple-valued functions. *IEE Proceedings on Computers and Digital Techniques*, 148(1):7–14, January 2001.

[FR02] B. J. Falkowski and S. Rahardja. Boolean verification with fastest LIA transforms. In *Proceedings of the 35th IEEE International Symposium on Circuits and Systems*, pages 321–324, Phoenix, AZ, USA, May 2002.

[FSP92] B. J. Falkowski, I. Schäfer, and M. A. Perkowski. Generation of adding and arithmetic multi-polarity transforms for incompletely specified Boolean functions. *International Journal of Electronics*, 73(2):321–331, August 1992.

[FSP92a] B. J. Falkowski, I. Schäfer, and M. A. Perkowski. Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions. *IEEE Transactions on Computer-Aided Design*, 11(10):1207–1226, October 1992.

[FY04] B. J. Falkowski and S. Yan. Walsh-Hadamard spectral minimization of fixed polarity Reed-Muller expansions. In *Proceedings of the 47th IEEE Midwest Symposium on Circuits and Systems*, pages 509–512, Hiroshima, Japan, July 2004.

[GE78] D. H. Green and M. Edkins. Synthesis procedures for switching circuits represented in generalized Reed-Muller form over a finite field. *IEE Proceedings-Computers and Digital Techniques*, 1(1):27–35, January 1978.

[Gre86] D. H. Green. *Modern Logic Design*. Wokingham: Addison-Wesley, 1986.

[Gre89] D. H. Green. Ternary Reed-Muller switching functions with fixed and mixed polarities. *International Journal of Electronics*, 67(5):761–775, November 1989.

[Gre90] D. H. Green. Reed-Muller expansions with fixed and mixed polarities over GF(4). *IEE Proceedings-Computers and Digital Techniques*, 137(5):380–388, September 1990.

[GT74] D. H. Green and I. S. Taylor. Modular representation of multiple-valued logic systems. *IEE Proceedings-Computers and Digital Techniques*, 121(2):166–172, February 1974.

[GT76] D. H. Green and I. S. Taylor. Multiple-valued switching circuit design by means of generalized Reed-Muller expansions. *Digital Processes*, 2(1):63–81, 1976.

[Har90] B. Harking. Efficient algorithm for canonical Reed-Muller expansions of Boolean functions. *IEE Proceedings-Computers and Digital Techniques,* 137(5):366–370, September 1990.

[Hei91] K. D. Heidtmann. Arithmetic spectrum applied to fault detection for combinational networks. *IEEE Transactions on Computers*, 40(3):320–324, March 1991.

[HM92] B. Harking and C. Moraga. Efficient derivation of Reed-Muller expansions in multiple-valued logic systems. In *Proceedings of the 22nd IEEE International Symposium on Multiple-Valued Logic*, pages 436–441, Sendai, Japan, May 1992.

[HMM85] S. L. Hurst, D. M. Miller, and J. C. Muzio. *Spectral Techniques in Digital Logic*. London: Academic Press, 1985.

[Hur78] S. L. Hurst. *The Logical Processing of Digital Signals*. New York: Crane-Russak, 1978.

[Hur89] S. L. Hurst. Use of linearisation and spectral techniques in input and output compaction testing of digital networks. *IEE Proceedings-Computers and Digital Techniques*, 136(1):48–56, January 1989.

[Hur92] S. L. Hurst. *Custom VLSI Microelectronics*. Hertfordshire: Prentice-Hall, 1992.

[JSD01] D. Jankovic, R. S. Stankovic, and R. Drechsler. Decision diagram method for calculation of pruned Walsh transform. *IEEE Transactions on Computers*, 50(2):147–157, February 2001.

[JSD02] D. Jankovic, R. S. Stankovic, and R. Drechsler. Efficient calculation of fixed-polarity polynomial expressions for multiple-valued logic functions. In *Proceedings of the 32nd IEEE International Symposium on Multiple-Valued Logic*, pages 76–82, Boston, MA, USA, May 2002.

[JSM02] D. Jankovic, R. S. Stankovic, and C. Moraga. Optimization of Kronecker expressions using the extended dual polarity property. In *Proceedings of the XXXVII International Scientific Conference on Information, Communication, and Energy System and Technologies*, pages 749–752, Nis, Yugoslavia, October 2002.

[JSM03] D. Jankovic, R. S. Stankovic, and C. Moraga. Optimization of GF(4) expressions using the extended dual polarity property. In *Proceedings of the 33rd IEEE International Symposium on Multiple-Valued Logic*, pages 50–55, Tokyo, Japan, May 2003.

[Jud94] T. W. Judson. *Abstract Algebra: Theory and Applications*. Boston: PWS Publishing Company, 1994.

[Kar76] M. G. Karpovsky. *Finite Orthogonal Series in Design of Digital Devices*. New York: Wiley, 1976.

[Kar85] M. G. Karpovsky (ed.). *Spectral Techniques and Fault Detection*. Orlando: Academic Press, 1985.

[KB81] S. K. Kumar and M. A. Breuer. Probabilistic aspects of Boolean switching functions via a new transform. *Journal of the ACM*, 28(3):502–520, July 1981.

[KKH+88] S. Kawahito, M. Kameyama, T. Higuchi, and H. Yamada. A 32×32-bit multiplier using multiple-valued MOS current-mode circuits. *IEEE Journal of Solid-State Circuits*, 23(1):124–132, February 1988.

[KPH99] U. Kalay, M. A. Perkowski, and D. V. Hall. Highly testable Boolean ring logic circuits. In *Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic*, pages 268–274, Freiburg, Germany, May 1999.

[KS75] K. L. Kodandapani and R. V. Setlur. Reed-Muller canonical forms in multivalued logic. *IEEE Transactions on Computers*, 24(6):628–636, June 1975.

[KSA00] M. G. Karpovsky, R. S. Stankovic, and J. T. Astola. Spectral techniques for design and testing of computer hardware. In *Proceedings of the 1st International Workshop on Spectral Techniques and Logical Design for Future Digital Systems*, pages 9–43, Tampere, Finland, June 2000.

[KSR92] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Proceedings of the 3rd IEEE European Conference on Design Automation,* pages 43–47, Brussels, Belgium, March 1992.

[KSY91] G. A. Kukharev, V. P. Shmerko, and S. N. Yanushkevich. *Technique of Binary Data Parallel Processing for VLSI.* Minsk: University Press, 1991 (in Russian).

[KSZ90] G. A. Kukharev, V. P. Shmerko, and E. N. Zaitseva. *Multiple-Valued Data Processing Algorithms and Systolic Processors*. Minsk: Science and Engineering, 1990 (in Russian).

[Lec71] R. J. Lechner. *Harmonic Analysis of Switching Functions*. New York: Academic Press, 1971.

[Llo80] A. M. Lloyd. Design of multiplexer universal logic module networks using spectral techniques. *IEE Proceedings-Computers and Digital Techniques,* 127(1):31–36, January 1980.

[LM98] J. S. Lee and L. E. Miller. *CDMA Systems Engineering Handbook.* Boston: Artech House, 1998.

[Mal97] V. D. Malyugin. *Parallel Calculations by Means of Arithmetical Polynomials.* Moscow: Physical and Mathematical Publishing Company, Russian Academy of Sciences, 1997 (in Russian).

[Mil87] D. M. Miller (ed.). *Developments in Integrated Circuit Testing*. London: Academic Press, 1987.

[MMH82] J. C. Muzio, D. M. Miller, and S. L. Hurst. Number of spectral coefficients necessary to identify a class of Boolean functions. *Electronic Letters*, 18(13):577–579, 1982.

[Moh92] P. S. Moharir. *Pattern-Recognition Transforms*. New York: Wiley, 1992.

[MTH04] A. Mochizuki, T. Takeuchi, and T. Hanyu. Intra-chip address-presetting data-transfer scheme using four-valued encoding. In *Proceedings of the 34th International Symposium on Multiple-Valued Logic*, pages 192–197, Toronto, Canada, May 2004.

[Muz80] J. C. Muzio. Composite spectra and the analysis of switching circuits. *IEEE Transactions on Computers*, 29(8):750–753, August 1980.

[Pal32] R. E. A. C. Paley. A remarkable series of orthogonal functions. *Proceedings of London Mathematical Society*, 34(2):241–279, 1932.

[Pal33] R. E. A. C. Paley. On orthogonal matrices. *Journal of Mathematics and Physics*, 12:311–320, 1933.

[PDF90] M. A. Perkowski, P. Dysko, and B. J. Falkowski. Two learning methods for a tree-search combinatorial optimizer. In *Proceedings of the 9th IEEE International Conference on Computers and Communication,* pages 606–613, Phoenix, AZ, USA, March 1990.

[Per92] M. A. Perkowski. The generalized orthonormal expansion of functions with multiple-valued inputs and some of its applications. In *Proceedings of the 22nd IEEE International Symposium on Multiple-Valued Logic*, pages 442–450, Sendai, Japan, May 1992.

[Per93] M. A. Perkowski. A fundamental theorem for EXOR circuits. In *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 52–60, Hamburg, Germany, September 1993.

[PFC+02] M. A. Perkowski, B. J. Falkowski, M. Chrzanowska-Jeske, and R. Drechsler. Efficient algorithms for creation of linearly-independent decision diagrams and their mapping to regular layouts. *VLSI Design*, 14(1):35–52, February 2002.

[PM75] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers*, 24(6):668–670, June 1975.

[PSB95] M. A. Perkowski, A. Sarabi, and F. R. Beyl. Fundamental theorems and families of forms for binary and multiple-valued linearly independent logic. In *Proceedings of the IFIP WG 10.5 1st Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 288–299, Makuhari, Chiba, Japan, August 1995.

[Pur91] S. Purwar. An efficient method of computing generalized Reed-Muller expansions from binary decision diagram. *IEEE Transactions on Computers*, 40(11):1298–1301, November 1991.

[Rad22] H. Rademacher. Einige Sätze von allgemeinen Orthogonalfunktionen. *Math. Annalen*, 87:122–138, 1922.

[Red72] S. M. Reddy. Easily testable realizations for logic functions. *IEEE Transactions on Computers,* 21(11):1183–1188, November 1972.

[RF99] S. Rahardja and B. J. Falkowski. Application of linearly independent arithmetic transform in testing of digital circuits. *Electronics Letters*, 35(5):363–364, March 1999.

[RF99a] S. Rahardja and B. J. Falkowski. Fast linearly independent arithmetic expansions. *IEEE Transactions on Computers*, 48(9):991–999, September 1999.

[RF01] S. Rahardja and B. J. Falkowski. Efficient algorithm to calculate Reed-Muller expansions over GF(4). *IEE Proceedings-Circuits, Devices, and Systems*, 148(6):289–295, December 2001.

[RF02] S. Rahardja and B. J. Falkowski. Polynomial expansions over GF(2) based on fastest transformation. In *Proceedings of the 35th IEEE International Symposium on Circuits and Systems*, pages 377–380, Scottsdale, AZ, USA, May 2002.

[Rin84] D. C. Rine (ed.). *Computer Science and Multiple-Valued Logic: Theory and Applications.* Amsterdam: Elsevier Science Publishers, 1984.

[Ros99] K. H. Rosen. *Discrete Mathematics and Its Applications.* Boston: McGraw-Hill, 1999.

[RP88] B. K. R. Reddy and A. L. Pai. Reed-Muller transform image coding. *Computer Vision, Graphics and Image Processing*, 42(1):48–61, April 1988.

[RSL03] S. Rahardja, W. Ser, and Z. Lin. UCHT-based complex sequences for asynchronous CDMA system. *IEEE Transactions on Communications*, 51(4):618–626, April 2003.

[RZ04] K. Radecka and Z. Zilic. Design verification by test vectors and arithmetic transform universal test set. *IEEE Transactions on Computers*, 53(5):628–640, May 2004.

[SA03] R. S. Stankovic and J. T. Astola. *Spectral Interpretation of Decision Diagrams*. New York: Springer-Verlag, 2003.

[Sas88] T. Sasao. Multiple-valued logic and optimization of programmable logic arrays. *IEEE Computer*, 21(4):71–80, April 1988.

[Sas93] T. Sasao (ed.). *Logic Synthesis and Optimization*. Boston: Kluwer Academic, 1993.

[Sas99] T. Sasao. *Switching Theory for Logic Synthesis*. Boston: Kluwer Academic, 1999.

[SB90] T. Sasao and P. Besslich. On the complexity of MOD-2 Sum PLA's. *IEEE Transactions on Computers*, 39(2), pp. 262–266, February 1990.

[SF96] T. Sasao and M. Fujita (eds.). *Representations of Discrete Functions*. Boston: Kluwer Academic, 1996.

[Sha38] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of American Institute of Electrical Engineers*, 57:713–723, March 1938.

[SKM+90] A. Srinivasan, T. Kam, S. H. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *Proceedings of the 8th IEEE International Conference on Computer-Aided Design*, pages 92–95, Santa Clara, CA, USA, November 1990.

[SP92] A. Sarabi and M. A. Perkowski. Fast exact and quasy-minimal minimization of highly testable fixed polarity AND/XOR canonical networks. In *Proceedings of the 29th ACM/IEEE Design Automatic Conference*, pages 30–35, Anaheim, CA, USA, June 1992.

[SS98] R. S. Stankovic and T. Sasao. Decision diagrams for discrete functions: classification and unified interpretation. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 439–446, Yokohama, Japan, February 1998.

[SSC+94] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski. A comprehensive approach to logic synthesis and physical design for two-dimensional logic arrays. In *Proceedings of the 31st IEEE/ACM Design Automation Conference*, pages 321–326, San Diego, CA, USA, June 1994.

[SSJ98] R. S. Stankovic, M. Stankovic, and D. Jankovic. *Spectral Transforms in Switching Theory: Definitions and Calculations.* Belgrade: IP Nauka, 1998.

[Sta95] R. S. Stankovic. Functional decision diagrams for multiple-valued functions. In *Proceedings of the 25th IEEE International Symposium on Multiple-Valued Logic*, pages 284–289, Bloomington, IN, USA, May 1995.

[TDM01] M. A. Thornton, R. Drechsler, and D. M. Miller. *Spectral Techniques in VLSI CAD*. Dordrecht: Kluwer Academic, 2001.

[TH79] V. H. Tokmen and S. L. Hurst. A consideration of universal-logic-modules for ternary synthesis based upon Reed-Muller coefficients. In *Proceedings of the 9th IEEE International Symposium on Multiple-Valued Logic*, pages 248–256, Bath, UK, May 1979.

[TH04] T. Takahashi and T. Hanyu. Multiple-valued multiple-rail encoding scheme for low-power asynchronous communication. In *Proceedings of the 34th International Symposium on Multiple-Valued Logic*, pages 20–25, Toronto, Canada, May 2004.

[Tok80] V. H. Tokmen. Disjoint decomposability of multiple valued functions by spectral means. In *Proceedings of the 10th IEEE International Symposium on Multiple-Valued Logic*, pages 88–93, New York, USA, May 1980.

[Tra87] A. Tran. Graphical method for the conversion of minterms to Reed-Muller coefficients and the minimisation of exclusive-OR switching functions. *IEE Proceedings-Computers and Digital Techniques,* 134(2):93–99, March 1987.

[TV87] E. A. Trachtenberg and D. Varma. A design automation system for spectral logic synthesis. In *Proceedings of the International Workshop on Logic Synthesis*, Research Triangle Park, NC, USA, May 1987.

[VP01] C. F. Vinokurov and N. A. Periazev (eds.). *Algorithm and Theory for Boolean Functions*. Moscow: Fizmalit, 2001 (in Russian).

[VT88] D. Varma and E. A. Trachtenberg. A fast algorithm for the optimal state assignment of large finite state machines. In *Proceedings of the 6th IEEE International Conference on Computer-Aided Design*, pages 152–155, Santa Clara, CA, USA, November 1988.

[VT89] D. Varma and E. A. Trachtenberg. Design automation tools for efficient implementation of logic functions by decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):901–916, August 1989.

[VT91] D. Varma and E. A. Trachtenberg. Computation of Reed-Muller expansions of incompletely specified Boolean functions from reduced

representations. *IEE Proceedings-Computers and Digital Techniques*, 138(2):85–92, March 1991.

[Wal23] J. L. Walsh. A closed set of orthogonal functions. *American Journal of Mathematics*, 45:5–24, 1923.

[Wal72] J. S. Wallis. *Hadamard Matrices*. New York: Springer-Verlag, 1972.

[Wol98] W. Wolf. *Modern VLSI Design*, *Systems on Silicon*. Upper Saddle River: Prentice Hall, 1998.

[Yan94] S. Yanushkevich. Systolic synthesis algorithms for arithmetic polynomial forms of *k*-valued functions of Boolean algebra. *Automation and Remote Control*, 55(12):1812–1823, December 1994.

[Yan98] S. N. Yanushkevich. *Logic Differential Calculus in Multi-Valued Logic Design.* Szczecin: Technical University of Szczecin Press, 1998.

[Yar85] L. P. Yaroslavsky. *Digital Picture Processing: An Introduction*. Berlin: Springer-Verlag, 1985.

[YFS01] S. N. Yanushkevich, B. J. Falkowski, and V. P. Shmerko. Spectral linear arithmetic approach for multiple-valued functions: overview of applications in CAD of circuit design. In *Proceedings of the 3rd International Conference on Information, Communications and Signal Processing*, CD publication, Singapore, October 2001.

[YMS+06] S. N. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. S. Stankovic. *Decision Diagram Techniques for Micro-and Nanoelectronic Design Handbook*. Boca Raton: CRC Press, 2006.

[Zhe27] I. I. Zhegalkin. O tekhnyke vychyslenyi predlozhenyi v symbolytscheskoi logykye. *Matematicheskii Sbornik*, 34(1):9–28, 1927 (in Russian).

[Zhe28] I. I. Zhegalkin. Arifmetizatiya symbolytscheskoi logyky. *Matematicheskii Sbornik*, 35(1):311–377, 1928 (in Russian).

[ZV93] Z. Zilic and Z. Vranesic. Current-mode CMOS Galois Field circuits. In *Proceedings of the 23rd IEEE International Symposium on Multiple-Valued Logic*, pages 245–250, Sacramento, CA, USA, May 1993.