

Teaching Reed-Muller Techniques in Introductory Classes on Logic Design

Svetlana N. Yanushkevich and Vladimir P. Shmerko

Abstract: Reed-Muller techniques are not traditionally included in the textbooks for introductory one-semester courses on logic design. Two exclusions are the textbooks by D. Green (1986) and T. Sasao (1999). Based on our experience of developing and instructing logic design courses, we introduce our approach to teaching the Reed-Muller techniques for undergraduate students.

Keywords: Reed-Miller techniques, logic design, sum-of-product, Boolean techniques, minterms, polynomial form.

1 Introduction

An introductory course on logic design of discrete devices is a fundament for the study of many fields that constitute the ever-expanding discipline of computer engineering, computer science, and electrical engineering. While it serves as a prerequisite for additional coursework in the study of theory of communication, signal processing, digital system design, and neural networks, it is frequently encountered in the second year of the undergraduate programs and assumes no background on the part of the reader. The course usually includes five main topics: number system, Boolean algebra, sum-of-product (SOP) based manipulations, and techniques for combinational and sequential logic network design. The SOP techniques are the focus of the introductory course based on the textbooks such as [1, 2]. Additional courses are then provided that expand on the one-semester course by including a more detailed treatment of digital system design, focusing, in particular, on simulation using hardware description languages.

Manuscript received August 4, 2007.

Department of Electrical and Computer Engineering University of Calgary, CANADA, (e-mail: [syanshk, vshmerko]@ucalgary.ca).

Reed-Muller techniques are not being usually included in introductory one-semester courses on logic design, and only two exclusive efforts are known: the textbook by D. Green [3] and the textbook by T. Sasao [4]. While SOP-based forms are the primary focus of undergraduate curricular, Reed-Muller techniques are often taught in classes on the advanced techniques [5, 6]. The main reason is that Reed-Muller techniques are more complicated than SOP-based techniques.

The motivation for incorporating Reed-Muller into the introductory course on logic design are as follows:

- (a) Reed-Muller techniques are an important part of the contemporary logic design, and
- (b) Reed-Muller techniques are found useful for predictable technologies.

This viewpoint is reflected in our recent textbook [7]. In this textbook, we propose several approaches for teaching the polynomial forms of Boolean functions. Our motivation to study polynomial forms of Boolean functions is as follows:

- Polynomial expressions provide additional flexibility in terms of choice of implementation technology. This property is efficiently utilized in logic design, especially in design for specific-area applications; in particular, encoding and encryption of information.
- There are various physical and molecular effects in predictable technology which can be interpreted as EXOR operations. Nanocomputing devices based on these effects can be used in logic network design and implementation.
- Polynomial forms are well-suited to a logic with more than two values, in particular, to the multi-valued logic. This fact is utilized in the design of some contemporary and next-generation devices.

2 Similarities Between SOP and Polynomial Forms

This topic is based on the understanding of the SOP-based techniques. Given a complete set of minterms, often in the form of truth vector, for a Boolean function, its standard (canonical) SOP expression is formed using the correspondence of 1's in the truth vector. By analogy, a polynomial form is derived from the correspondence of the polarized minterms and non-zero coefficients of the vector of coefficients.

The standard, or canonical, SOP and polynomials forms are unique given a Boolean function. The number of terms in canonical SOP and polynomial expressions are equal to 2^n . Non-canonical SOP expressions can be derived from canonical SOP forms. Similarly, canonical and non-canonical polynomial expressions

can be derived given a Boolean function. Figure 1 shows the structural similarity of standard SOP expressions and polynomial forms of Boolean functions.

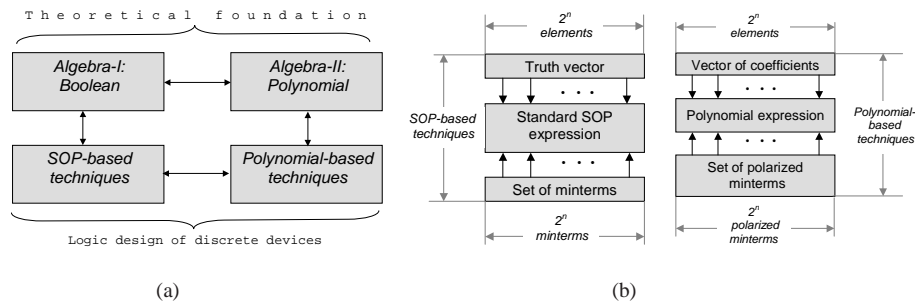


Fig. 1. Techniques for the manipulation of Boolean functions are based on Boolean algebra (Algebra I) and the algebra of polynomial forms (Algebra II) (a) and the structural similarity of standard SOP expressions and polynomial forms (b).

The polynomial form is a representation of a Boolean function derived from the following universal set of operations over Boolean variables: the constant 1, AND operation, and EXOR operation.

Example 1. (Completeness of operations). (a) EXOR of single variables forms, a so-called “linear polynomial”, $x_1 \oplus \dots \oplus x_n$.
 (b) EXOR and AND operations form a so-called “non-linear polynomial”; for example, $x_1 \oplus x_1x_2 \oplus x_1x_2x_3$ is a non-linear polynomial.
 (c) EXOR, and AND operations and the constant 1 are used to implement an arbitrary Boolean function. For example, a complemented variable can be represented using EXOR and the constant 1: $\bar{x} = x \oplus 1$.

Polynomial forms can be represented as follows (Figure 2):

- Algebraic forms,
- Tabulated forms such as *functional tables* and *vectors of coefficients* (similar to truth tables and truth vectors for SOP forms);
- Graphical representations such as *functional maps* and *functional cubes*, as well as *functional decision trees and diagrams* (similar to K-maps, cubes, and decision trees and diagrams for SOP forms); and
- Networks of logic gates or threshold elements.

Operational and functional domains

The relationship between *operational* and *functional* domains is the key to the synthesis and application of the polynomial forms of Boolean functions. In polynomial

algebras, the duality principle exists in the form of *forward* and *inverse* transforms between *operational* and *functional* domains. *Forward* and *inverse* transforms describe the relationship between *operational* and *functional* domains for Boolean data structures:

$$\underbrace{\text{Operational domain}}_{\text{Boolean data structure}} \xleftrightarrow{\text{Conversion}} \underbrace{\text{Functional domain}}_{\text{Boolean data structure}}$$

All satellite Boolean data structures (Figure 2) and the corresponding techniques are aimed at providing for representation, manipulation, optimization, and implementation of Boolean functions in the functional domain; namely:

- (a) Each data structure has particular properties and characteristics, and satisfies the requirements of specific tasks of the logic design cycle. There is no “universal” data structure that can be used in all phases of logic design.
- (b) Each data structure plays a particular role in design, and is efficient only in solving particular tasks.
- (c) Each data structure can be converted into another one. These relationships between data structures are often used to achieve design goals.

3 Algebra of the Polynomial Forms

Boolean algebra is defined as a set of elements, operations, and postulates. This algebraic structure is the formal basis of the SOP representation. The formal basis of the polynomial forms is the *finite fields*. Finite fields are algebraic structures too, but they are characterized by the *elements*, *operations*, and *postulates* of a finite field. The theory of polynomial representations of Boolean functions has been adopted from the related fields, such as digital signal processing.

3.1 Theoretical background

A *finite field* \mathcal{F} is an algebraic structure defined as follows:

- It is a set of elements, together with two binary operations, each having associative, commutative, and distributive properties, closure under addition and multiplication, inverse properties, and a unique element.
- The number of elements in the field is called the **order** of the field. A field with order m exists iff m is a **prime power**, i.e., $m = p^n$ for some integer n and with p a prime integer. In this case, addition and multiplication are defined by a table composed such that the requirements for the field are true.

OPERATIONAL DOMAIN

Algebraic form

$$f = \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$$

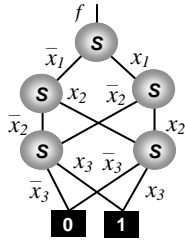
Truth table

Variables			f
x ₁	x ₂	x ₃	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

K-map

		x ₁ x ₂			
		00	01	11	10
x ₃	0		1		1
	1	1		1	

Decision diagram



FUNCTIONAL DOMAIN

Algebraic form

$$f = x_1 \oplus x_2 \oplus x_3$$

Functional table

Coefficients								f
r ₀	r ₁	r ₂	r ₃	r ₄	r ₅	r ₆	r ₇	
0	1	1	0	1	0	0	0	

Functional map

		x ₁ x ₂			
		00	01	11	10
x ₃	0		1		1
	1	1		1	

Functional decision diagram

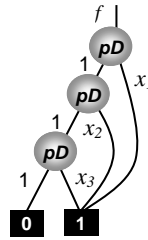


Fig. 2. Data structures for representation of the EXOR function of two variables in the operational domain (SOP form) and the functional domain (polynomial form).

- In any finite field, the number of elements must be a power of a prime, p^k . This field is the Galois field $GF(k)$.
- Every field with p^k elements is isomorphic to every other field with p^k elements. Some of these fields are useful in the representation, manipulation, analysis, and implementation of Boolean functions.

Binary operations are defined as *addition over a field \mathcal{F}* , and *multiplication over a field \mathcal{F}* .

Galois field

An example of a field \mathcal{F} is a *Galois field* denoted as $\text{GF}(q)$:

- It consists of q elements $0, 1, 2, \dots, q$.
- The number of elements in a Galois field must be equal to $p = 2^n$, where p is a *prime* number and n is a positive integer (a natural number $p \geq 2$ is called prime if and only if the only natural numbers which divide p are 1 and p).
- In cases where $p = 2$, all 2^n elements are derived using a polynomial of degree n .
- Operations in $\text{GF}(q)$ are the modulo q sum and modulo q multiplication.

The field of integer numbers modulo a prime number k is a field.

Polynomials

A *polynomial* in the variable x is the representation of a function f as a sum over an algebraic field \mathcal{F}

$$f = \sum_{i=0}^{N-1} a_i x^i \quad \text{over the field } \mathcal{F} \quad (1)$$

The values a_0, a_1, \dots, a_{N-1} are called *coefficients* of the polynomial. Expression 1 means that there exist many polynomials, which are distinguished by the properties of the fields; namely, by the types of operation being addition and multiplication.

Example 2. (Fields for Boolean functions.) *The following fields are used for the representation of Boolean functions:*

- (a) Galois field of order 2, $\text{GF}(2)$. *This field consists of two elements, 0 and 1. In $\text{GF}(2)$, sum and multiplication correspond to EXOR and AND operations, respectively.*
- (b) *A set of integer numbers, or the field of integers, that includes only the elements 0 and 1. In this field, traditional sum and multiplication are used.*

In addition, the sets of rational and complex numbers, together with the arithmetic operations of sum and multiplication, can be used for various representations of Boolean functions.

3.2 Polynomials for Boolean functions

The polynomial in Equation 1 is defined for a single variable x . Boolean algebra operates with a set of variables x_1, x_2, \dots, x_n . To apply this polynomial equation to Boolean functions, some restrictions are needed. These restrictions are based on the fundamental theorem of arithmetic, which states that every integer $i > 2$ can be written in the form $i = i_1 i_2 \dots i_n$ for the unique *primes* $i_1 i_2 \dots i_n$. This means that if any number is completely factored, this expression is unique. Given a Boolean function of n variables x_1, x_2, \dots, x_n ,

$$f = \sum_{i=0}^{2^n-1} a_i \overbrace{(x_1^{i_1} \dots x_n^{i_n})}^{\text{Minterm over } \mathcal{F}} \quad \text{over the field } \mathcal{F}, \quad (2)$$

where a_i is a coefficient, i_j is the j -th bit ($j = 1, 2, \dots, n$) in the binary representation of the index $i = i_1 i_2 \dots i_n$, and the *literal* $x_j^{i_j}$ is defined as

$$x_j^{i_j} = \begin{cases} 1, & \text{if } i_j = 0; \\ x_j, & \text{if } i_j = 1. \end{cases} \quad \text{over the field } \mathcal{F} \quad (3)$$

The group of variables $x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}$ is called a *minterm over the field* \mathcal{F} . While the values of Boolean functions are used in SOP (operational domain), the coefficients a_i are used in polynomial forms of Boolean functions (functional domain).

The polynomial form (Equation 2) is characterized as follows:

- (a) The *operations*, of sum and multiplication are specified by the properties of the field \mathcal{F} ; that is, they are either *logical* or *arithmetic* operations;
- (b) The *coefficients* a_i are computed for each Boolean function using the properties of the field \mathcal{F} ; and
- (c) *Minterms* over the field \mathcal{F} are specified by multiplication of *literals* $x_j^{i_j}$, $j = 1, 2, \dots, n$, over \mathcal{F} (Equation 3).

Example 3. (Polynomials for Boolean functions.) *The following polynomials represent the Boolean function EXOR in the two fields (algebras):*

$$\begin{aligned} f_1 &= x_1 \oplus x_2 \quad \text{over } GF(2) \\ f_2 &= x_1 + x_2 - 2x_1 x_2 \quad \text{over the field of integers} \end{aligned}$$

In the polynomials f_1 and f_2 , logical and arithmetical operations are used, respectively. In f_1 , the coefficients are 0 or 1 because the field $GF(2)$ consists of 0's and

1's only. The coefficients in the polynomial f_2 are integer numbers. Once the values of the Boolean variables x_1 and x_2 are assigned, polynomials f_1 and f_2 assume the values of the initial function EXOR.

The above brief introduction to the basics of finite fields implies that the polynomial forms are more complicated compared than SOP forms, since special techniques are required for computing the coefficients of polynomial forms.

3.3 The functional table

A Boolean function of n variables $f(x_i) \ i = 1, 2, \dots, n$, in the *operational* domain can be described in tabulated form using truth tables. In the *functional* domain, a Boolean function is represented in a polynomial form $f(x_i, a_j) \ i, j \in \{1, 2, \dots, n\}$, which is a function of variables x_i and coefficients a_j , and can be described by a *functional table*.

A functional table is a list of all combinations of 1's and 0's assigned to the binary coefficients $a_0, a_1, \dots, a_{2^n-1}$ and corresponding polynomials. The structural properties of the functional table are given below:

Structural properties of the functional table

- Each row corresponds to a combination of the 2^n coefficients $a_0, a_1, \dots, a_{2^n-1}$ of the polynomial. The number of rows in the table is 2^{2^n} , where n is the number of variables of the Boolean function.
- The columns are the 2^{2^n} polynomial forms of all Boolean functions of n variables. The truth table can be obtained for each polynomial.

Example 4. (Functional table.) For a Boolean function f of two variables x_1 and x_2 ($n = 2$), the functional table is derived as follows. There are $2^n = 2^2 = 4$ coefficients a_0, a_1, a_2 , and a_3 , which specify $2^{2^n} = 2^{2^2} = 16$ polynomial forms of the Boolean function. Hence, the functional table contains 16 rows for all combinations of the coefficients a_0, a_1, a_2, a_3 , and the column labeled f that contains 16 polynomials (Figure 3). For each polynomial in the functional table, a truth table can be derived as shown for four of them. For example, the combination of coefficients $a_0 a_1 a_2 a_3 = 0011$ specifies the polynomial $x_1 \oplus x_1 x_2$. The corresponding function is specified by its values: $x_1 x_2 = 00 \Rightarrow f = 0 \oplus 0 \cdot 0 = 0$; $x_1 x_2 = 01 \Rightarrow f = 0 \oplus 0 \cdot 1 = 0$, etc.

3.4 The functional map

The analog of the *truth vector* and K-map in the functional domain is the *vector of coefficients*. All 2^{2^n} possible vectors of coefficients for a Boolean function of n

Functional table					Functional table (continuation)				
a_0	a_1	a_2	a_3	Polynomial f	a_0	a_1	a_2	a_3	Polynomial f
0	0	0	0	0	1	0	0	0	1
0	0	0	1	x_1x_2	1	0	0	1	$1 \oplus x_1x_2$
0	0	1	0	x_1	1	0	1	0	$1 \oplus x_1$
0	0	1	1	$x_1 \oplus x_1x_2$	1	0	1	1	$1 \oplus x_1 \oplus x_1x_2$
0	1	0	0	x_2	1	1	0	0	$1 \oplus x_2$
0	1	0	1	$x_2 \oplus x_1x_2$	1	1	0	1	$1 \oplus x_2 \oplus x_1x_2$
0	1	1	0	$x_2 \oplus x_1$	1	1	1	0	$1 \oplus x_1 \oplus x_2$
0	1	1	1	$x_2 \oplus x_1 \oplus x_1x_2$	1	1	1	1	$1 \oplus x_2 \oplus x_1 \oplus x_1x_2$

Truth tables for the first four polynomials

$f = 0$			$f = x_1x_2$			$f = x_1$			$f = x_1 \oplus x_1x_2$		
x_1	x_2	f	x_1	x_2	f	x_1	x_2	f	x_1	x_2	f
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	1	0	0	1	0
1	0	0	1	0	0	1	0	1	1	0	1
1	1	0	1	1	1	1	1	1	1	1	0

Fig. 3. Representation of Boolean functions of two variables in the form of a functional table and the corresponding truth tables for the first four polynomials (Example 4).

variables are represented by the *functional map*. Each polynomial in the functional table represents one of 2^{2^n} functions of n variables.

Example 5. (K-map and functional map.) Let the Boolean function of three variables $f = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2x_3$ be given in the form of a K-map (Figure 4). The functional map contains the coefficients of the polynomial in $GF(2)$. The gates used for the implementation of the function in the operational domain are AND and OR, while the gates for polynomial implementation are AND and EXOR.

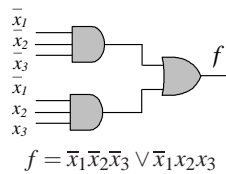
3.5 Polarized minterms

The polynomial form of Equation 2 contains only uncomplemented variables. In order to achieve acceptable flexibility in a network design based on polynomial forms, we need a techniques for manipulation of uncomplemented and complemented variables. This techniques are based on so-called *polarized minterms*. The polarized minterm is the product of *polarized literals*. Note that in an SOP expres-

Operational domain

K - m a p

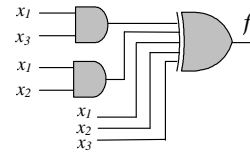
		$x_1 x_2$			
		00	01	11	10
x_3	0	1			
	1		1		



Functional domain

M a p o f
c o e f f i c i e n t s

		$x_1 x_2$			
		00	01	11	10
x_3	1 0	1	1	1	1
	x_3 1	1			1



$$f = x_3 \oplus x_2 \oplus x_1 \oplus x_1 x_3 \oplus x_1 x_2$$

Fig. 4. Representation of the Boolean function $f = \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3$ in the operational and functional domains (Example 5).

sion, DeMorgan's rule provides for manipulation of complemented and uncomplemented variables.

Literals and polarized literals

A literal is the representation of either an uncomplemented or a complemented variable:

$$\text{Literal} = x_j^{i_j} = \begin{cases} \bar{x}_j, & \text{if } i_j = 0; \\ x_j, & \text{if } i_j = 1. \end{cases} \quad (4)$$

Literals in the form of Equation 4 are used in the standard SOP forms. The polarities of variables are specified by the particular Boolean function. The polarity can be changed using DeMorgan's rule.

Example 6. (Literals.) Let $i = 0, 1, 2, 3$ ($i_1 i_2 = 00, 01, 10, 11$ for the representation of two Boolean variables). According to Equation 4, the following literals can be generated:

$$\{x_1^0 x_2^0, x_1^0 x_2^1, x_1^1 x_2^0, x_1^1 x_2^1\} = \{\bar{x}_1 \bar{x}_2, \bar{x}_1 x_2, x_1 \bar{x}_2, x_1 x_2\}.$$

The *polarized* form of a literal provides an approach to *independent* control of the polarity of variables. A *polarized literal* is the representation of either an uncomplemented or a complemented variable specified by the control parameter called *polarity* $c_1, c_2, \dots, c_n, c_j \in \{0, 1\}, j = 1, 2, \dots, n$:

$$\text{Polarized literal} = (x_j \oplus c_j)^{i_j} = \begin{cases} 1, & \text{if } i_j = 0; \\ (x_j \oplus c_j), & \text{if } i_j = 1. \end{cases} \quad \text{over GF}(2) \quad (5)$$

In Equation 5, the parameters i_j for the variable x_j are *separated* from the polarity c_j :

- Parameters i_j only specify the order of the minterms in the polynomial. They are an inherent property of a given form; that is, i_j are dependent parameters.
- Parameter c_j is an *independent* parameter.

The example below shows all possible combinations of the dependent and independent parameters of the literal.

Example 7. (Polarized literals.) For the polarity $c_j \in \{0, 1\}$ and parameter $i_j \in \{0, 1\}$, the complete set of polarized literals is generated as follows:

$$\begin{array}{ll} \overbrace{(x_j \oplus 0)^0}^{c_j=0, i_j=0} = x_j^0 = 1 & \text{over GF}(2); & \overbrace{(x_j \oplus 1)^0}^{c_j=1, i_j=0} = \bar{x}_j^0 = 1 & \text{over GF}(2); \\ \overbrace{(x_j \oplus 0)^1}^{c_j=0, i_j=1} = x_j^1 = x_j & \text{over GF}(2); & \overbrace{(x_j \oplus 1)^1}^{c_j=1, i_j=1} = \bar{x}_j^1 = \bar{x}_j & \text{over GF}(2). \end{array}$$

Minterm structure

The minterm is defined for the assignment i_1, i_2, \dots, i_n of Boolean variables x_1, x_2, \dots, x_n for which a Boolean function is equal to 1; that is, $x_1 = i_1, x_2 = i_2, \dots, x_n = i_n$ if $f = 1$:

$$\text{Minterm} = \overbrace{x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}}^{n \text{ literals}} \quad (6)$$

These minterms are used in the standard SOP expressions. The simplest method for generating the minterms is to use the truth table of the Boolean function.

Polarized minterm structure

A *polarized minterm* is defined by the equation

$$\text{Polarized minterm} = \overbrace{(x_1 \oplus c_1)^{i_1} (x_2 \oplus c_2)^{i_2} \cdots (x_n \oplus c_n)^{i_n}}^{n \text{ polarized literals}} \text{ over GF}(2) \quad (7)$$

where

$$(x_j \oplus c_j)^{i_j} = \begin{cases} 1, & \text{if } i_j = 0; \\ (x_j \oplus c_j), & \text{if } i_j = 1. \end{cases} \text{ over GF}(2)$$

In Equation 7, the polarities of the variables x_1, x_2, \dots, x_n are specified by the polarity parameters c_1, c_2, \dots, c_n , respectively. An arbitrary polarity $c_i \in \{0, 1\}$ can be chosen for each Boolean variable x_i , $i = 1, 2, \dots, n$.

Example 8. (Polarized minterms.) A polarized minterm of two variables is represented as follows:

$$\text{Polarized minterm} = (x_1 \oplus c_1)^{i_1} (x_2 \oplus c_2)^{i_2} \text{ over GF}(2)$$

For example, all four polarized minterms for the polarity $c = 2$ ($c_1 c_2 = 10$), can be in four forms:

$$\begin{aligned} (x_1 \oplus 1)^0 (x_2 \oplus 0)^0 &= 1; & (x_1 \oplus 1)^0 (x_2 \oplus 0)^1 &= x_2; \\ (x_1 \oplus 1)^1 (x_2 \oplus 0)^0 &= \bar{x}_1; & (x_1 \oplus 1)^1 (x_2 \oplus 0)^1 &= \bar{x}_1 x_2. \end{aligned}$$

4 Techniques for Manipulation of Polynomial Forms

The term *polynomial forms* specifies the forms of Boolean functions in which minterms are combined using the EXOR operation. An arbitrary Boolean function can be represented by polynomial expression. The laws of the GF(2) algebra of polynomial forms are given in Table 1. In the example below, techniques for the manipulation of polynomial expressions using laws and identities are introduced.

Example 9. (GF(2) algebra.) The below manipulations illustrate application of the laws of algebra of polynomial forms from Table 1:

$$\begin{aligned} (a) \quad x_1 \oplus x_2 \oplus x_1 x_2 &= x_1 \oplus x_2 (1 \oplus x_1) = x_1 \oplus x_2 \bar{x}_1 \\ (b) \quad 1 \oplus x_1 x_2 \oplus x_1 x_3 &= 1 \oplus x_1 (x_2 \oplus x_3) = \overline{x_1 (x_2 \oplus x_3)} \end{aligned}$$

Table 1. The EXOR algebra and identities for manipulations (fragment).

Laws and identities	Formal notation	Logic network
Associative law	$x_1 \oplus (x_2 \oplus x_3) = (x_1 \oplus x_2) \oplus x_3$ $= x_1 \oplus x_2 \oplus x_3$ $x_1(x_2x_3) = (x_1x_2)x_3$ $= x_1x_2x_3$	
Identities for variables	$x \oplus x = 0, \quad x \cdot x = x$ $x \oplus \bar{x} = 1, \quad x \cdot \bar{x} = 0$	
DeMorgan's rules for polynomials	$x_1 \oplus \bar{x}_2 = \overline{x_1 \oplus x_2}$ $\bar{x}_1 \oplus x_2 = \overline{x_1 \oplus x_2}$ $\bar{x}_1 \oplus \bar{x}_2 = x_1 \oplus x_2$	

4.1 Relationship between standard SOP and polynomial forms

Polynomial form of a Boolean function can be derived directly from the SOP expression of this function using the following rule:

Deriving a polynomial form given a standard SOP expression

Given: The standard SOP expression of a Boolean function

Find: The standard polynomial form

Procedure: Replace the OR operations with EXOR operation

Result: The standard polynomial form of the Boolean function

This standard polynomial form is not an optimal in terms of the implementation cost and representation; additional manipulations are needed for its simplification.

Example 10. (A SOP and polynomial forms.) *The truth table of Boolean function is given in Table 2. A standard SOP of this function is*

$$f = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2x_3$$

The polynomial expression is derived by replacing OR operations by the EXOR operations:

$$f = \bar{x}_1\bar{x}_2\bar{x}_3 \oplus \bar{x}_1\bar{x}_2x_3 \oplus \bar{x}_1x_2x_3 \oplus x_1x_2x_3$$

Note that this polynomial expression consists of a the minterms with variables of different polarities.

Table 2. Relationship between the standard SOP and polynomial forms (Example 10).

Assignment of variables	Value of Boolean function	Minterm
000	1	$\bar{x}_1\bar{x}_2\bar{x}_3$
001	1	$\bar{x}_1\bar{x}_2x_3$
010	0	
011	1	$\bar{x}_1x_2x_3$
100	0	
101	0	
110	0	
111	1	$x_1x_2x_3$

4.2 Local transformations for EXOR expressions

A *local transformation* is defined as a set of rules for the simplification of a data structure. In this section, we consider a local transformations for a logic networks, which consist of various types of logic gates, including EXOR gates. These transformations are based on the theorems of Boolean algebra and polynomial algebra GF2, and are applied locally. The following rules can be applied to logic networks with EXOR gates:

The rules for local transformations

Rule 1: *Replacing an EXOR gate with a constant:*

- Replace an EXOR gate with the corresponding constant using the rules of identities for constants if the inputs of this gates are constants
- Replace an EXOR gate with the corresponding constant using the rules of identities for variables if the inputs of this gates are literals of the same variable

Rule 2: *Replacing an EXOR gate with a variable:* An EXOR gate can be replaced with a variable using the rules of identity for variables and constants if one of the inputs is a constant

The rules for removing the duplicated gates, removing the unused gates, and merging the gates, are similar to the ones for OR and AND gates.

Example 11. (Local transformations.) Figure 5 illustrates two types of the local transformations: Area A: the EXOR gate is replaced with the inverter using the identity rule for variables and constants $x_2 \oplus 1 = \bar{x}_2$; Area B: the inverter and EXOR gates are replaced by the wire using the simplification rule $x_1\bar{x}_2 \oplus x_1 = x_1(\bar{x}_2 \oplus 1) = x_1x_2$.

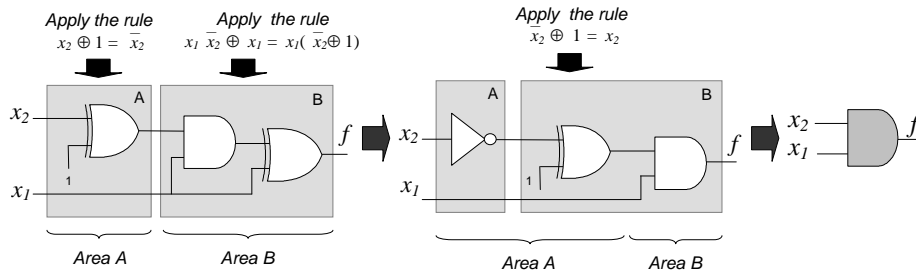


Fig. 5. Local transformations in logic network (Example 11).

4.3 Factoring of polynomials

Factoring of polynomial expressions is used, in particular, when dealing with the fan-in problem, and when a logic network is designed using limited numbers of gate inputs. However, techniques for factoring SOP forms are not acceptable for polynomial forms. Factoring polynomial expressions is based on the laws and identities given in Table 1. Similarly to SOP-based techniques, the factoring of polynomial expressions is based on the designer’s experience, and may be built into CAD tools to a limited extent. The application of various identities does not guarantee satisfactory results from factoring. In particular, an arbitrary Boolean variable can be replaced by its complement as follows:

$$x_1 \oplus x_2 = \overline{\overline{x_1} \oplus x_2} = \overline{x_1 \oplus \overline{x_2}} = (1 \oplus x_1)x_2 \oplus 1 = x_1 \oplus (1 \oplus x_2) \oplus 1$$

Extra variables can be included in the equation using the following properties: $x \oplus x \oplus x = x$ and $x \oplus x = 0$.

Example 12. (Factoring.) *The polynomial expression*

$$f = \underbrace{1 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2 \oplus x_1x_2x_3}_{2 \text{ level logic network}}$$

can be directly implemented by the two-level logic network as shown in Figure 6a. The fan-in of the EXOR gate is equal to 7, which is often unacceptable. The factoring results in the below expression

$$f = \underbrace{1 \oplus x_4 \oplus (x_3 \oplus x_2 \oplus x_2x_3)x_1}_{4 \text{ level logic network}}$$

This polynomial expression is implemented by a four-level logic network (Figure 6b) using 3-input EXOR gates.

I n i t i a l n e t w o r k M o d i f i e d n e t w o r k

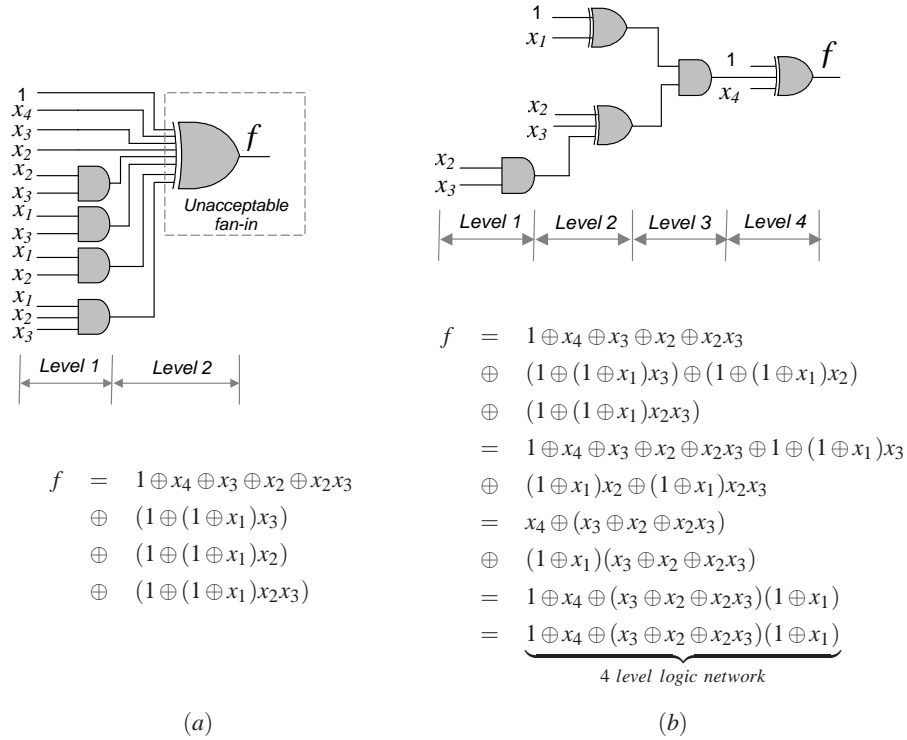


Fig. 6. Two-level logic network implementation of a non-factored polynomial expression (a) and four-level logic network implementation of a factored polynomial expression (b) (Example 12).

5 Fixed and Mixed Polarity Polynomial Forms

In terms of polynomial forms, two types of polarities are distinguished: *fixed* polarity and *mixed* polarity. In a fixed polarity polynomial expression of a Boolean function f , every variable appears either complemented (\bar{x}_i) or uncomplemented (x_i), and never in both forms. There are 2^n fixed polarity forms given a function of n variables. In a mixed polarity form, a variable can appear in one or both polarities. Given a function of n variables, there are 3^n mixed polarity forms.

Example 13. (Fixed and mixed polarity.) In Figure 7, the polynomial expressions are given in a fixed and a mixed polarity forms. In the chosen fixed polarity, the variable x_2 appears uncomplemented; the variables x_1 and x_3 appear complemented only. In the chosen mixed polarity form, the variables x_1, x_2 and x_3 appear in both uncomplemented and complemented forms

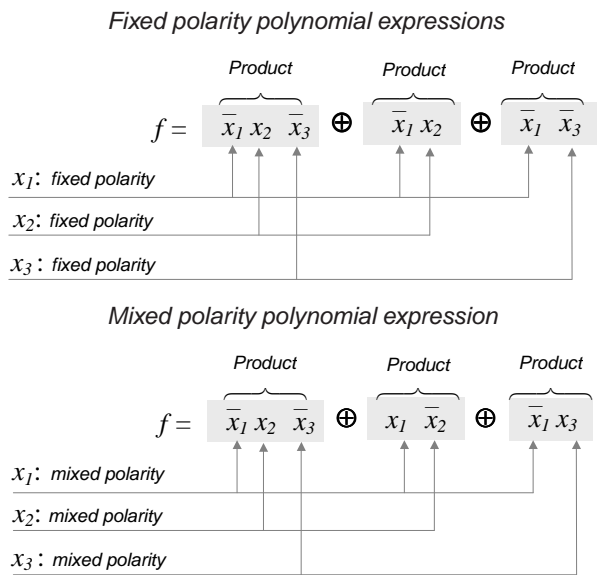


Fig. 7. Illustration of fixed and mixed polarity polynomial forms (Example 13).

A fixed polarity polynomial expression of a Boolean function f of n variables is *unique*; that is, only one representation exists given a polarity c (c_1, c_2, \dots, c_n).

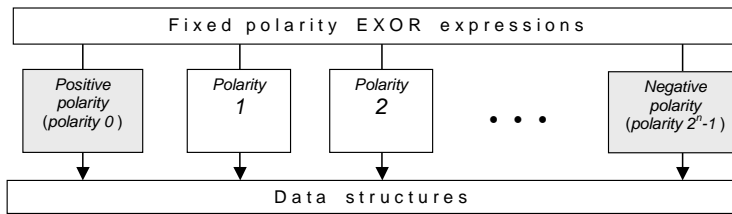


Fig. 8. An arbitrary Boolean function can be represented in a fixed polarity polynomial form.

There are 2^n various polarities, and two boundary cases among them: a *positive polarity* form in which all variables are uncomplemented, and a *negative polarity* form in which all variables are complemented (Figure 8).

Example 14. (Fixed polarity.) *Polynomial expressions in positive polarity $c = 0$ ($c_1 c_2 = 00$) (uncomplemented variables) and the polarity $c = 2$, $c_1 c_2 = 10$ (only the variable x_1 is complemented) are derived as follows:*

$$\begin{aligned}
\text{Polarity } \mathbf{c} = \mathbf{0} \quad (c_1 = 0, c_2 = 0) : \\
f &= r_0(x_1 \oplus c_1)^{i_1}(x_2 \oplus c_2)^{i_2} \oplus r_1(x_1 \oplus c_1)^{i_1}(x_2 \oplus c_2)^{i_2} \\
&\oplus r_2(x_1 \oplus c_1)^{i_1}(x_2 \oplus c_2)^{i_2} \oplus r_3(x_1 \oplus c_1)^{i_1}(x_2 \oplus c_2)^{i_2} \\
&= r_0(x_1^0 x_2^0) \oplus r_1(x_1^0 x_2^1) \oplus r_2(x_1^1 x_2^0) \oplus r_3(x_1^1 x_2^1) \\
&= r_0 \oplus r_1 x_2 \oplus r_2 x_1 \oplus r_3 x_1 x_2 \\
\text{Polarity } \mathbf{c} = \mathbf{2} \quad (c_1 = 1, c_2 = 0) : \\
f &= r_0(x_1 \oplus 1)^0(x_2 \oplus 0)^0 \oplus r_1(x_1 \oplus 1)^0(x_2 \oplus 0)^1 \\
&\oplus r_2(x_1 \oplus 1)^1(x_2 \oplus 0)^0 \oplus r_3(x_1 \oplus 1)^1(x_2 \oplus 0)^1 \\
&= r_0 \oplus r_1 x_2 \oplus r_2 \bar{x}_1 \oplus r_3 \bar{x}_1 x_2.
\end{aligned}$$

Let $f = x \vee y$, then there are four fixed polarity polynomial expressions:

$$\begin{aligned}
0\text{-polarity: } f &= x_1 \oplus x_2 \oplus x_1 x_2, & \text{no complemented variables} \\
1\text{-polarity: } f &= 1 \oplus \bar{x}_2 \oplus x_1 \bar{x}_2, & x_2 \text{ is complemented} \\
2\text{-polarity: } f &= 1 \oplus \bar{x}_1 \oplus \bar{x}_1 x_2, & x_1 \text{ is complemented} \\
3\text{-polarity: } f &= 1 \oplus \bar{x}_1 \bar{x}_2, & x_1 \text{ and } x_2 \text{ are complemented}
\end{aligned}$$

Deriving a fixed polarity polynomial expansion given a Boolean function is a necessary step in the process of implementation of the function given a library of logic gates AND and EXOR together with a constant 1 signal. It forms a *universal* basis of operations for implementing an arbitrary Boolean function.

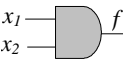
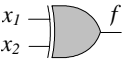
Example 15. (Fixed polarities for gates.) In Table 3, the expressions of the AND and EXOR Boolean functions of two variables are given in fixed polarities. For example, the polynomial in polarity $c = 3$ ($c_1 c_2 = 11$) for the OR function is represented by two non-zero coefficients $f = 1 \oplus \bar{x}_1 \bar{x}_2$. This is an optimal polynomial form of the OR function with respect to the criterion of the minimal number of literals.

5.1 Conversion between polarities

Given one polarity of a polynomial expression of a Boolean function, one can convert it to another polarity expression by algebraic manipulations.

Example 16. (Conversion between polarities.) The mixed polarity polynomial expression $\bar{x}_1 \bar{x}_2 \oplus \bar{x}_1 x_2 \oplus x_1 x_2$ can be transformed into a polynomial form of polarity

Table 3. Polynomial expressions of fixed polarities of elementary Boolean functions (Example 15).

Function (gate)	Coefficients				Polynomial expression
	r_0	r_1	r_2	r_3	
 $f = x_1x_2$	0	0	0	1	x_1x_2
	0	0	1	1	$x_1 \oplus x_1\bar{x}_2$
	0	1	0	1	$x_2 \oplus \bar{x}_1x_2$
	1	1	1	1	$1 \oplus \bar{x}_2 \oplus \bar{x}_1 \oplus \bar{x}_1\bar{x}_2$
 $f = x_1 \oplus x_2$	0	1	1	0	$x_2 \oplus x_1$
	1	1	1	0	$1 \oplus \bar{x}_2 \oplus x_1$
	1	1	1	0	$1 \oplus x_2 \oplus \bar{x}_1$
	0	1	1	0	$\bar{x}_2 \oplus \bar{x}_1$

$c = 2$ ($c_1 = 1, c_2 = 0$):

$$\begin{aligned}
 f &= \overbrace{\bar{x}_1\bar{x}_2 \oplus \bar{x}_1x_2 \oplus x_1x_2}^{\text{Mixed polarity}} \\
 &= \bar{x}_1(1 \oplus x_2) \oplus \bar{x}_1x_2 \oplus (x_1 \oplus 1 \oplus 1)x_2 \\
 &= \bar{x}_1 \oplus \bar{x}_1x_2 \oplus \bar{x}_1x_2 \oplus \bar{x}_1x_2 \oplus x_2 = \overbrace{\bar{x}_1 \oplus \bar{x}_1x_2 \oplus x_2}^{\text{Fixed 1-polarity}}
 \end{aligned}$$

Given the same initial mixed polarity expression, the polynomial form of the fixed polarity $c = 1$ ($c_1 = 0, c_2 = 1$) is obtained as follows:

$$\begin{aligned}
 f &= \bar{x}_1\bar{x}_2 \oplus \bar{x}_1x_2 \oplus x_1x_2 \\
 &= (x_1 \oplus 1)\bar{x}_2 \oplus (x_1 \oplus 1)(\bar{x}_2 \oplus 1) \oplus x_1(\bar{x}_2 \oplus 1) \\
 &= x_1\bar{x}_2 \oplus \bar{x}_2 \oplus x_1\bar{x}_2 \oplus x_1 \oplus \bar{x}_2 \oplus 1 \oplus x_1\bar{x}_2 \oplus x_1 = \overbrace{x_1\bar{x}_2 \oplus 1}^{\text{Fixed 2-polarity}}
 \end{aligned}$$

5.2 Simplification of polynomial expressions

Given a mixed polarity polynomial form, further algebraic transformations can lead to a minimized polynomial representation. Unlike the SOP-based techniques, minimization on a K-map cannot be directly applied to polynomial forms, since the rules for reducing are different in $GF(2)$. Some of these rules are given below: $x \oplus \bar{x} = 1$; $x \oplus x = 0$, $x \oplus 1 = \bar{x}$, and $x \oplus 0 = x$. Since the EXOR operation is commutative and associative, the following rules hold true as well $x_1x_2 \oplus \bar{x}_1x_2 = x_2$, $x_1\bar{x}_2 \oplus x_1 = x_1\bar{x}_2$, and $(x_1 \oplus x_2)x_1 = x_1\bar{x}_2$.

The details of the polynomial minimization are not the subject of the introductory logic design course. Exhaustive techniques, such as generation of all possible mixed polarity forms, can be applied to find the minimal forms. Obviously, such techniques can only be applied to small functions, and for larger functions some heuristic approaches have been developed.

Example 17. (Minimal expressions.) *The expression $\bar{x}_1 \oplus \bar{x}_1 x_2 \oplus x_2$ obtained in Example 16 can be further simplified as follows:*

$$\begin{aligned} f &= \bar{x}_1 \oplus \bar{x}_1 x_2 \oplus x_2 = \bar{x}_1(1 \oplus x_2) \oplus x_2 \\ &= (\bar{x}_1 \oplus x_2)(\bar{x}_2 \oplus x_2) = \bar{x}_1 \oplus x_2 \end{aligned}$$

The resulting expression is a minimal one.

5.3 Polarized minterms in matrix form

The matrix form of a polarized literal is based on the assumption that

- All operations are performed over GF(2);
- Multiplication of the *elementary* transform matrix and the truth vector of the variable x_j results in a vector of coefficients that corresponds to the simplest polynomial expressions, literals $x_j \oplus 0$ or $x_j \oplus 1$.

The elementary $2^1 \times 2^1$ transform matrix is denoted as $\mathbf{R}_{2^1}^{(c_j)}$, where $c_j \in \{0, 1\}$ is the polarity of the variable x_j . The polarized literal corresponds to the elementary transform matrices for $c_j = 0$ and $c_j = 1$ as follows:

$$\text{Polarized literal} = \mathbf{R}_{2^1}^{(c_j)} = \begin{cases} \mathbf{R}_{2^1}^{(0)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, & \text{if } c_j = 0; \\ \mathbf{R}_{2^1}^{(1)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, & \text{if } c_j = 1 \end{cases} \quad \text{over } GF(2) \quad (8)$$

Computing the polarized literal means multiplying the elementary transform matrix $\mathbf{R}_{2^1}^{(c_j)}$ by the truth vector of a single variable x_j , $\mathbf{F} = [0 \ 1]^T$. The polarized literal is computed by the multiplication of the elementary matrix by the truth vector for the variable x_j , $\mathbf{F} = [0 \ 1]^T$, over GF(2):

$$\begin{aligned} \mathbf{R}_{2^1}^{(0)} \mathbf{F} &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \longrightarrow r_0 \oplus r_1 x_j = x_j \\ \mathbf{R}_{2^1}^{(1)} \mathbf{F} &= \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \longrightarrow r_0 \oplus r_1 x_j = 1 \oplus x_j \end{aligned}$$

The polarized minterm is formed using the Kronecker product between n elementary matrices $\mathbf{R}_{2^1}^{(c_j)}$ as follows:

$$\mathbf{R}_{2^n}^{(c)} = \bigotimes_{j=1}^n \mathbf{R}_{2^1}^{(c_j)}, \quad (9)$$

where $\mathbf{R}_{2^1}^{(c_j)}$ is defined by Equation 8. The resulting $2^n \times 2^n$ -matrix $\mathbf{R}_{2^n}^{(c)}$ represents a minterm of polarity $c = c_1 c_2 \dots c_n$.

Example 18. (Kronecker product.) The Kronecker product of matrices $\mathbf{R}_{2^1}^{(0)}$ is computed as shown below:

$$\mathbf{R}_{2^2}^{(0)} = \mathbf{R}_{2^1}^{(0)} \otimes \mathbf{R}_{2^1}^{(0)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \right]$$

Example 19. (Polarized minterms.) A polarized minterm of fifth polarity ($c = 5$, $n = 3$) is constructed in matrix form as shown in Figure 9.

Using Equation 9, the polarized minterms can be generated given the polarity of an EXOR expression. In algebraic form, the EXOR expression is a sum of polarized minterms over GF(2)

$$f = \bigoplus_{i=0}^{2^n-1} (x_1 \oplus c_1)^{i_1} \dots (x_n \oplus c_n)^{i_n} \quad (10)$$

where the polarized literals are formed by Equation 5.

Forward transform

A forward transform is used for the representation of the truth vector of a Boolean function (operational domain) in the form of a vector of coefficients in polynomial form (functional domain):

$$\underbrace{\text{Truth vector}}_{\text{Operational domain}} \xrightarrow[\text{over GF(2)}]{\text{Transform}} \underbrace{\text{Vector of coefficients}}_{\text{Functional domain}}$$

Specifically, given the truth table $\mathbf{F} = [f(0) f(1) \dots f(2^n - 1)]^T$, the vector of coefficients in polarity c , $\mathbf{R}^{(c)} = [r_0^{(c)} r_1^{(c)} \dots r_{2^n-1}^{(c)}]^T$ is derived by the matrix equation

$$\mathbf{R}^{(c)} = \mathbf{R}_{2^n}^{(c)} \cdot \mathbf{F} \text{ over GF(2)}, \quad (11)$$

Deriving a polarized minterm in matrix form

Step 1. Find the corresponding elementary matrix for each literal:

$$\text{Algebraic form} \longrightarrow \underbrace{(x_1 \oplus 1)^{i_1}}_{\mathbf{R}_{2^1}^{(1)}} \underbrace{(x_2 \oplus 0)^{i_2}}_{\mathbf{R}_{2^1}^{(0)}} \underbrace{(x_3 \oplus 1)^{i_3}}_{\mathbf{R}_{2^1}^{(1)}}$$

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

Step 2. Form the $2^3 \times 2^3$ transform matrix $\mathbf{R}_{2^3}^{(5)}$ for the fifth polarity as the Kronecker product of the elementary matrices:

$$\begin{aligned} & \underbrace{\underbrace{\mathbf{R}_{2^1}^{(1)} \otimes \mathbf{R}_{2^1}^{(0)}}_{\text{The 1st step}} \otimes \mathbf{R}_{2^1}^{(1)}}_{\text{The 2nd step}} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}}_{\text{The first step}} \\ & = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \mathbf{R}_{2^3}^{(5)} \end{aligned}$$

Step 3. Use the matrix $\mathbf{R}_{2^3}^{(5)}$ for the matrix transform of the vector \mathbf{F} to a vector of coefficients in the fifth polarity. In Boolean expressions, the variables $x_1, x_2,$ and x_3 are used as $\bar{x}_1, x_2,$ and $\bar{x}_3,$ respectively; that is, the polarities of variables are fixed

Fig. 9. Deriving a polarized minterm in matrix form (Example 19).

where the $2^n \times 2^n$ -matrix $\mathbf{R}_{2^n}^{(c)}$ is generated by the Kronecker product

$$\mathbf{R}_{2^n}^{(c)} = \bigotimes_{j=1}^n \mathbf{R}_{2^1}^{(c_j)}, \quad \mathbf{R}_{2^1}^{(c)} = \begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, & c_j = 0; \\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, & c_j = 1. \end{cases}$$

Example 20. (Forward transform.) Given a Boolean function of two variables in the form of a truth vector $\mathbf{F} = [1011]^T$, the vector of coefficients is computed as follows:

$$\mathbf{R}^{(2)} = \mathbf{R}_{2^2}^{(2)} \cdot \mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \text{ over } GF(2)$$

where the matrix $\mathbf{R}_2^{(2)}$ given $c = 2$ is generated using the Kronecker product

$$\mathbf{R}_2^{(2)} = \mathbf{R}_2^{(1)} \otimes \mathbf{R}_2^{(0)} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

The vector of coefficients $\mathbf{R}^{(2)} = [1 \ 0 \ 1 \ 1]^T$ corresponds to the expression $f = 1 \oplus \bar{x}_1 \oplus \bar{x}_1 x_2$.

Example 21. (Network conversion.) Given a logic network that implements a standard SOP expression (Figure 10), this logic network can be converted into an AND-EXOR network as follows:

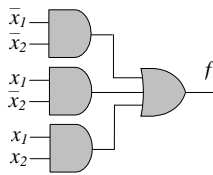
Step 1: Compute the truth vector: $\mathbf{F} = [1 \ 0 \ 1 \ 1]^T$

Step 2: Compute the vector of coefficients using the forward transform with the given positive polarity: $\mathbf{R} = \mathbf{R}_2^3 \cdot \mathbf{F} = [1 \ 1 \ 0 \ 1]^T$.

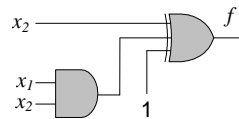
Step 3: Derive the corresponding algebraic form: $f = 1 \oplus x_2 \oplus x_1 x_2$.

Step 4: Design the AND-EXOR network.

Implementation of the standard SOP expression



Implementation of the polynomial expression



$$\mathbf{R} = \mathbf{R}_2^3 \cdot \mathbf{F} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \right] \left[\begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \end{array} \right] = \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \end{array} \right] \quad \text{over GF}(2)$$

$$f = \underbrace{\bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_2 \vee x_1 x_2}_{\text{Operational domain}} = \underbrace{1 \oplus x_2 \oplus x_1 x_2}_{\text{Functional domain}}$$

Fig. 10. Conversion of the AND-OR logic network into the AND-EXOR network using forward transform (Example 21).

Inverse transform

The inverse transform is used for the conversion of a vector of coefficients of the polynomial form of a Boolean function (functional domain) into its truth vector (operational domain):

$$\underbrace{\text{Vector of coefficients}}_{\text{Functional domain}} \xrightarrow[\text{over GF}(2)]{\text{Transform}} \underbrace{\text{Truth vector}}_{\text{Operational domain}}$$

Given a vector of positive polarity polynomial coefficients $\mathbf{R} = [r_0 \ r_1 \ \dots \ r_{2^n-1}]^T$, the truth vector $\mathbf{F} = [f(0) \ f(1) \ \dots \ f(2^n - 1)]^T$ of a Boolean function f is derived as follows:

$$\mathbf{F} = \mathbf{R}_{2^n}^{-1} \cdot \mathbf{R} \text{ over GF}(2), \quad (12)$$

where $\mathbf{R}_{2^n}^{-1} = \mathbf{R}_{2^n}$. Notice that the matrix \mathbf{R}_{2^n} is a self-inverse matrix over AND and polynomial operations.

Example 22. (Inverse transform.) *Given the an AND-EXOR network (Figure 11), this network can be converted into an AND-OR network as follows:*

Step 1: *Compute the vector of coefficients $\mathbf{R} = [0 \ 1 \ 0 \ 1]^T$*

Step 2: *Compute the truth vector. Use the inverse transform given a certain polarity. Let the positive polarity is required: $\mathbf{F} = \mathbf{R}_{2^3}^{(-1)} \cdot \mathbf{R} = [0 \ 1 \ 0 \ 0]^T$. The truth vector corresponds to the standard SOP expression in algebraic form $f = \bar{x}_1 x_2 \vee x_1 \bar{x}_2$.*

Step 3: *Design the AND-OR network.*

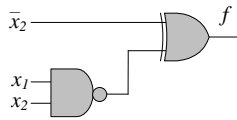
6 Functional Decision Diagrams

Functional decision diagrams can be introduced to the students based on understanding of decision trees and diagrams using Shannon expansion in the nodes for processing of Boolean functions. While the SOP representation is manipulated in the *operational domain*, the EXOR analog of Shannon expansion, known as *Davio expansion*, is manipulated in the *functional domain*. In this domain, Boolean functions are computed using polynomial representations, where a decision tree using Davio expansion is called a *functional decision tree*.

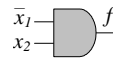
The functional decision tree can be reduced to a *functional decision diagram*. The reduction procedure for functional decision diagrams is different from the one for decision diagrams using Shannon expansion. This is because of different techniques for simplification of SOP and polynomial expressions.

Similarly to decision diagrams based on Shannon expansion, functional decision diagrams are used for various design tasks such as the representation, manipulation, optimization, and implementation of Boolean functions. The difference is that solutions of these tasks are in the functional domain.

Implementation of the polynomial expression



Implementation of the standard SOP expression



$$\mathbf{F} = \mathbf{R}_{2^3}^{(-1)} \cdot \mathbf{R} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \right] \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 1 \end{array} \right] = \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \end{array} \right] \quad \text{over GF}(2)$$

$$f = \bar{x}_2 \oplus \bar{x}_1 \bar{x}_2 = x_2 \oplus 1 \oplus x_1 x_2 \oplus 1 = \underbrace{x_2 \oplus x_1 x_2}_{\text{Functional domain}} = \underbrace{\bar{x}_1 x_2}_{\text{Operational domain}}$$

Fig. 11. Conversion of the AND-EXOR logic network into the AND-OR network using inverse transform (Example 22).

A node in the functional decision tree of a Boolean function f corresponds to the EXOR analog of Shannon expansion with respect to the variable x_i . It is called *Davio* expansion. In decision tree and diagram construction using SOP form, only one type of nodes is used; that is, nodes that implement Shannon expansion. There are two expansions in the functional domain: *positive Davio* expansion and *negative Davio* expansion. This is because the polynomial form of Boolean functions is characterized by polarity. These two expansions provide the construction of polynomial forms as follows:

The EXOR analog of Shannon expansion (Davio expansion)

Positive polarity polynomials: If only the positive Davio expansion is applied with respect to each variable of the function, the resulting polynomial is of zero polarity; that is, all variables in the polynomial are uncomplemented.

Negative polarity polynomials: If only the negative Davio expansion is applied, the resulting polynomial is of $2^n - 1$ polarity; that is, all variables in the polynomial are complemented.

Fixed polarity polynomials: Application of both positive and negative Davio expansion results in fixed polarity; that is, polarity from 1 to $2^n - 2$ of the polynomial.

6.1 Algebraic form of the positive Davio expansions

Given a Boolean function f of n variables $x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n$

$$f = f(x_1, x_2, \dots, x_{i-1}, \boxed{x_i}, x_{i+1}, \dots, x_n)$$

The positive Davio expansion with respect to the variable x_i is defined by the equation:

$$\boxed{f = f_0 \oplus x_i f_2} \quad (13)$$

where $f_2 = f_0 \oplus f_1$. Equation 13 is derived as follows. Shannon expansion of a Boolean function f with respect to the variable x_i results in the expression

$$\begin{aligned} f &= \bar{x}_i f_0 \oplus x_i f_1 = (1 \oplus x_i) f_0 \oplus x_i f_1 \\ &= f_0 \oplus x_i f_0 \oplus x_i f_1 = f_0 \oplus x_i \underbrace{(f_0 \oplus f_1)}_{f_2} \end{aligned}$$

Given $f_2 = f_0 \oplus f_1$, Equation 13 follows straightforwardly.

From Equation 13 follows that an arbitrary Boolean function f of n variables can be represented in expanded form with respect to the i -th variable x_i , $i \in 1, 2, \dots, n$. Hence, positive Davio expansion given Equation 13 is specified by the parameters f_0 , f_1 and f_2 :

Specification of the positive Davio expansion

Factor f_0 : This is the function that is obtained from the function f by replacing the variable x_i by the logic value 0

$$f_0 = f_{x_i=0} = f(x_1, \dots, x_{i-1}, \boxed{x_i = 0}, x_{i+1}, \dots, x_n)$$

Factor f_1 : This is the function that is obtained from the function f by replacing the variable x_i by the logic value 1

$$f_1 = f_{x_i=1} = f(x_1, \dots, x_{i-1}, \boxed{x_i = 1}, x_{i+1}, \dots, x_n)$$

Factor f_2 : This is the function that is obtained by the EXOR sum of factors f_0 and f_1 ; that is:

$$f_2 = f_0 \oplus f_1 = f_{x_i=0} \oplus f_{x_i=1}$$

Factor $x_i f_2$: This is the function that is obtained by the AND multiplication of the variable x_i by the factor f_2 .

Example 23. (Positive Davio expansion.) Let $f = x_1 \oplus x_2 \oplus x_1x_3$, the positive Davio expansion of the Boolean function f with respect to the variable x_2 is defined as follows:

$$\begin{aligned} f_0 &= x_1 \oplus (x_2 = 0) \oplus x_1x_3 = x_1 \oplus x_1x_3 \\ f_1 &= x_1 \oplus (x_2 = 1) \oplus x_1x_3 = 1 \oplus x_1 \oplus x_1x_3 \\ f_2 &= f_0 \oplus f_1 = \underbrace{x_1 \oplus x_1x_3}_{f_0} \oplus \underbrace{1 \oplus x_1 \oplus x_1x_3}_{f_1} = 1 \\ f &= f_0 \oplus x_2f_2 = x_1 \oplus x_1x_3 \oplus x_2 \end{aligned}$$

In terms of the functional decision diagram, the positive Davio expansion is interpreted as follows:

Computing the coefficients using the positive Davio expansion

- The node that implements the positive Davio expansion, denoted by pD, has two outgoing branches:
 - The left branch** corresponds to the factor $1 \cdot f_0$ and
 - The right branch** corresponds to the factor $x_i \cdot f_2$
- Four possible combinations of the outputs f_0 and f_2 can be observed in computing:
 - $\{f_0, f_2\} = \{0, 0\}$: Outputs of the left and right branches are both zero, hence, the input is $f = 0$;
 - $\{f_0, f_2\} = \{0, 1\}$: The output of the right branch is 1, hence, the input is $f = x_i$;
 - $\{f_0, f_2\} = \{1, 0\}$: Outputs of the left and right branches are both 1, hence, the input is $f = 1$;
 - $\{f_0, f_2\} = \{1, 1\}$: The output of the left branch is 1, hence, the input is $f = \bar{x}_i$.

Figure 12 illustrates the computational aspects of the positive Davio expansion.

6.2 Algebraic form of negative Davio expansions

Given a Boolean function f of n variables $x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n$, the negative Davio expansion with respect to the variable x_i is expressed by the equation:

$$\boxed{f = f_1 \oplus \bar{x}_i f_2} \quad (14)$$

By analogy with positive Davio expansion, negative Davio expansion is specified by the factors: f_0 , f_1 , f_2 , and $\bar{x}_i f_2$. Negative Davio expansion (Equation 14) with respect to the variable x_i is defined by analogy with positive Davio expansion:

$$\begin{aligned} f &= \bar{x}_i f_0 \oplus x_i f_1 = \bar{x}_i f_0 \oplus (1 \oplus \bar{x}_i) f_1 \\ &= \bar{x}_i f_0 \oplus f_1 \oplus \bar{x}_i f_1 = f_1 \oplus \bar{x}_i (f_0 \oplus f_1) = f_1 \oplus \bar{x}_i f_2 \end{aligned}$$

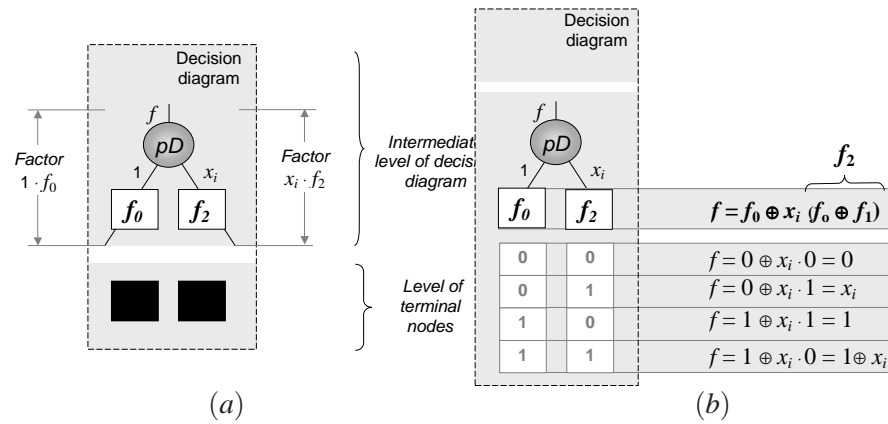


Fig. 12. Nodes in functional decision diagrams and trees which implement the positive Davio expansion pD : function of the node (a) and computing of the terminal node values (b).

Example 24. (Negative Davio expansion.) (Continuation of Example 23.) Negative Davio expansion with respect to the variable x_1 is defined as follows: $f = f_1 \oplus x_2 f_2 = 1 \oplus x_1 \oplus x_1 x_3 \oplus \bar{x}_2$

Figure 13 illustrates the computational aspects of negative Davio expansion.

Computing the coefficients using negative Davio expansion

- The node that implements the negative Davio expansion, denoted by nD , has two outputs:
 - The left branch** corresponds to the factor $1 \cdot f_1$ and
 - The right branch** corresponds to the factor $\bar{x}_i \cdot f_2$
- Four possible combinations of the outputs f_1 and f_2 can be observed in computing:
 - $\{f_1, f_2\} = \{0, 0\}$: Outputs of the left and right branches are both zero, hence, the input is $f = 0$;
 - $\{f_1, f_2\} = \{0, 1\}$: The output of the right branch is 1, hence, the input is $f = \bar{x}_i$;
 - $\{f_1, f_2\} = \{1, 0\}$: Outputs of the left and right branches are both 1, hence, the input is $f = 1$;
 - $\{f_1, f_2\} = \{1, 1\}$: The output of the left branch is 1, hence, the input is $f = x_i$.

6.3 Gate level implementation of Shannon and Davio expansions

Consider the gate level implementation of Shannon and Davio expansions for input x_i , and its complement \bar{x}_i , f_0 and f_1 . The logic networks are given in Figure 14 in

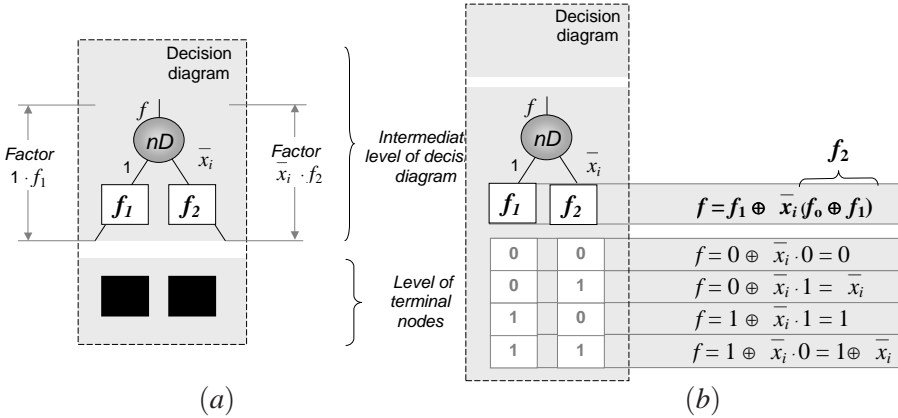


Fig. 13. Nodes in the functional decision diagrams and trees which implement the negative Davio expansion nD : function of the node (a) and computing of the terminal node values (b).

comparison with a network for Shannon expansion.

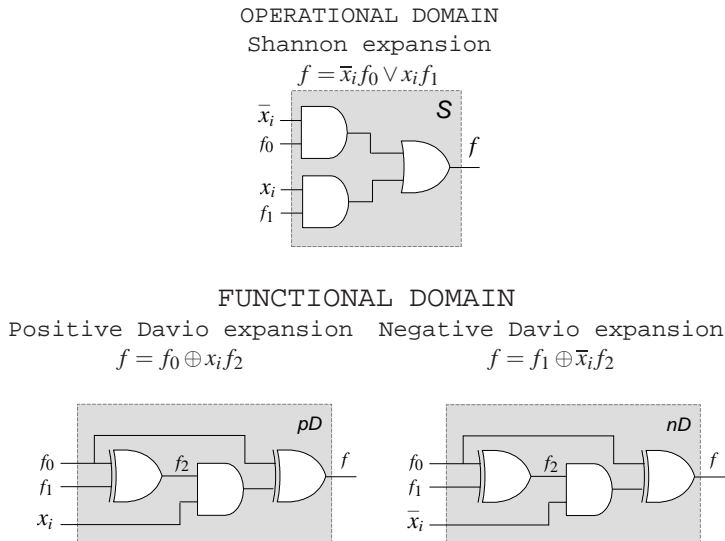


Fig. 14. Gate-level representation of the nodes of decision trees and diagrams using Shannon and Davio expansion.

Example 25. (Implementation of Davio expansion.) Given the Boolean function $f = x_1 \vee x_2$, its positive (pD) and negative (nD) Davio expansions with respect to

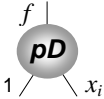
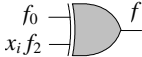
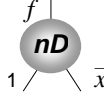

the variable x_1 result in the polynomial expressions

$$\begin{aligned}
 \mathbf{pD}: f &= f_0 \oplus x_1(f_0 \oplus f_1) = \underbrace{x_2}_{f_0} \oplus x_1 \underbrace{(x_2 \oplus 1)}_{f_0 \oplus f_1} = x_2 \oplus x_1 \oplus x_1 x_2 \\
 \mathbf{nD}: f &= f_1 \oplus \bar{x}_1(f_0 \oplus f_1) = \underbrace{x_2}_{f_1} \oplus x_1 \underbrace{(x_2 \oplus 1)}_{f_0 \oplus f_1} = 1 \oplus \bar{x}_1 \oplus \bar{x}_1 x_2
 \end{aligned}$$

The logic networks for Davio expansion given in Figure 14 can be used for computing by specification of the inputs; that is, $f_0 = x_2$ and $f_1 = 1$.

Table 4 summarizes various forms of interpretation of the Davio expansions: the functions of the nodes for positive Davio and negative Davio expansions, labeled as pD and nD respectively. For simplification, realization of the nodes is given using a single EXOR gate. Also, the matrix notation of the nodes for positive Davio and negative Davio expansions are given

Table 4. The nodes of the functional decision tree that implement Davio expansion at the gate-level and their description in algebraic and matrix forms.

Node	Realization	Algebraic	Matrix
Positive Davio node 		$ \begin{aligned} f &= f_0 \oplus x_i f_2 \\ f_0 &= f _{x_i=0} \\ f_2 &= f _{x_i=1} \oplus f _{x_i=0} \end{aligned} $	$ f = [1 \ x_i] \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = 1 \oplus x_i $
Negative Davio node 		$ \begin{aligned} f &= f_1 \oplus \bar{x}_i f_2 \\ f_1 &= f _{x_i=1} \\ f_2 &= f _{x_i=0} \oplus f _{x_i=1} \end{aligned} $	$ f = [1 \ \bar{x}_i] \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = 1 \oplus \bar{x}_i $

7 Techniques for Functional Decision Tree Construction

Techniques for functional decision tree construction consist of:

- Reduction of functional decision trees;
- Matrix-based design;
- Manipulation of pD and nD nodes for conversion between polarities.

7.1 The structure of functional decision trees

The most important structural properties of the functional decision tree with positive Davio nodes are as follows:

Structural properties of functional decision trees

- A Boolean function of n variables is represented by an n -level functional decision tree. The i -th level of the functional decision tree, $i = 1, \dots, n$, includes 2^{i-1} nodes.
- Nodes at the n -th level are connected to 2^n terminal nodes, which take values 0 or 1. The nodes, corresponding to the i -th variable, form the i -th level in the functional decision tree.
- In every path from the root node to a terminal node, the variables appear in a fixed order; the tree is thus said to be ordered.
- The constant nodes are assigned with the values of the coefficients of the polynomial expression for the Boolean function represented.

7.2 Design example: manipulation of pD and nD nodes

This design example introduces techniques for the design of functional decision diagrams for computing polynomial expressions of various polarities. This computing ability is provided by the distribution of pD and nD nodes in the levels of a decision tree. There are 2^n various combinations of the pD and nD nodes in the levels of a decision tree. Each combination corresponds to one polarity of a polynomial. There are two trivial cases in these 2^n combinations:

- (a) The tree consisting of only pD nodes; it computes only the positive polarity polynomial (all variables are non-complemented), and
- (b) The tree consisting of only nD nodes; it computes only the negative polarity polynomial (all variables are complemented).

Example 26. (Manipulation of pD and nD nodes). *Design functional decision trees for computing of all positive fixed polarity polynomial expressions of Boolean functions of two variables. Figure 15 shows all four possible decision trees.*

Example 27. (Tree design in terms of truth-vectors.) *A Boolean function is given by the truth-vector $\mathbf{F} = [0\ 1\ 1\ 0]^T$, construction of its functional decision diagram for the positive polynomial form is shown in Figure 16.*

7.3 Elimination rule

If the outgoing edge of a node labeled with x_i and \bar{x}_i points to the constant zero, then delete the node and connect the edge to the other subgraph directly. The formal basis of this rule is as follows (Figure 17): $\varphi = \varphi_0 \oplus x_i \varphi_2$. If $\varphi_2 = 0$ then $\varphi = \varphi_0$.

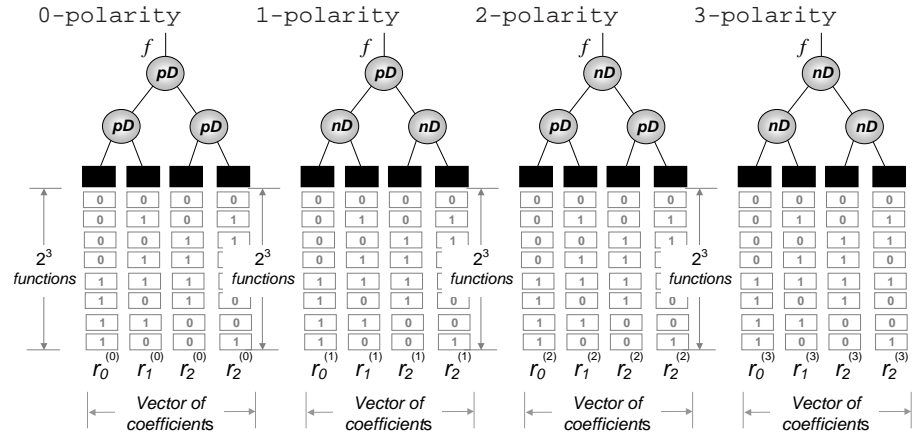
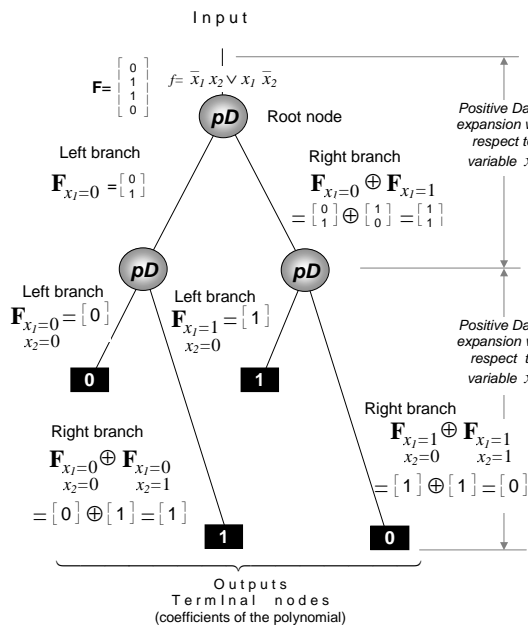


Fig. 15. Functional decision trees for computing polynomial expressions in positive, negative, and fixed polarity Boolean functions of two variables (Example 26).



Step 1: Root *pD* node with input $\mathbf{F} = [0110]^T$. Positive Davio expansion results in:

Left branch:
 $\mathbf{F}_{x_1=0} = [0 \ 1]^T$ and
 Right branch:
 $\mathbf{F}_{x_1=0} \oplus \mathbf{F}_{x_1=1} = [0 \ 1]^T \oplus [1 \ 0]^T = [1 \ 1]^T$

Both outputs results in functions and require further application of Davio expansion.

Step 2: Left node, left branch:
 $\mathbf{F}_{x_1=0, x_2=0} = [0 \ 0]$
 Left node, right branch:
 $\mathbf{F}_{x_1=0, x_2=0} \oplus \mathbf{F}_{x_1=0, x_2=1} = [0 \ 0] \oplus [1 \ 1] = [1 \ 1]$

Step 3: Right node, left branch:
 $\mathbf{F}_{x_1=1, x_2=0} = [1 \ 1]$
 Right node, right branch:
 $\mathbf{F}_{x_1=1, x_2=0} \oplus \mathbf{F}_{x_1=1, x_2=1} = [1 \ 1] \oplus [1 \ 1] = [0 \ 0]$

Fig. 16. Functional decision tree design with positive Davio expansion in the nodes using the truth-vector of the Boolean function $f = \bar{x}_1x_2 \vee x_2\bar{x}_2$ (Example 27).

7.4 Merging rule

Share equivalent subgraphs. In a tree, edges longer than one; i.e., connecting nodes at non-successive levels, can appear. For example, the length of an edge connecting a node at the $(i - 1)$ -th level with a node at the $(i + 1)$ -th level is two.

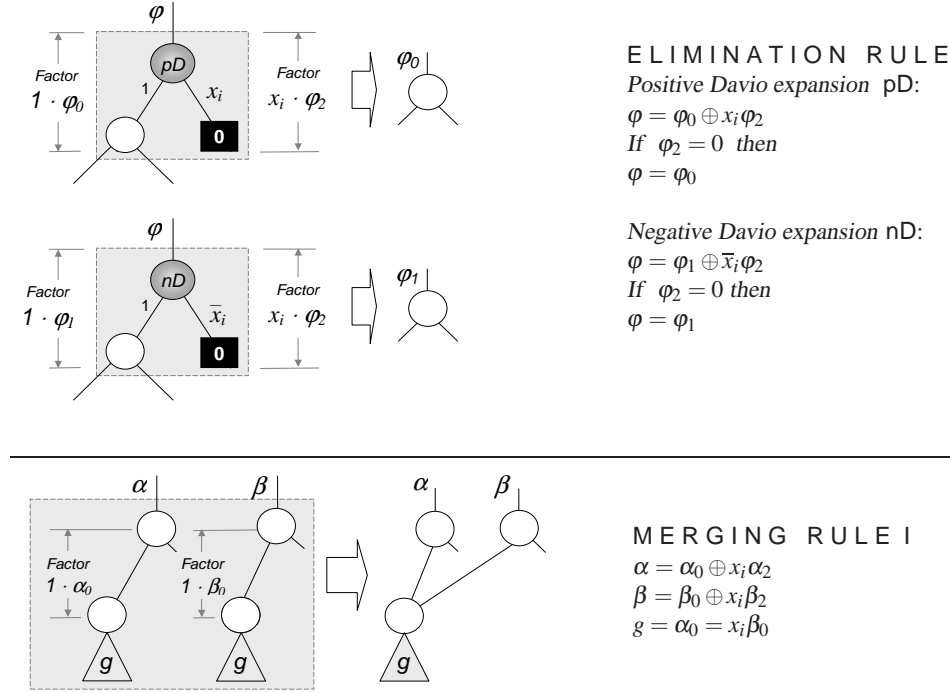


Fig. 17. Reduction rules for functional decision diagram construction.

Example 28. (Reduction rules.) Application of reduction rules to the three-variable NAND function is demonstrated in Figure 18.

8 Conclusion

Although classical in content, our approach is different from other approaches and textbooks on introduction to logic design in emphasizing topics, such as data structures, design and technological requirements. Our approach aims at incorporating the Reed-Muller techniques into the classical textbooks and is characterized as follows:

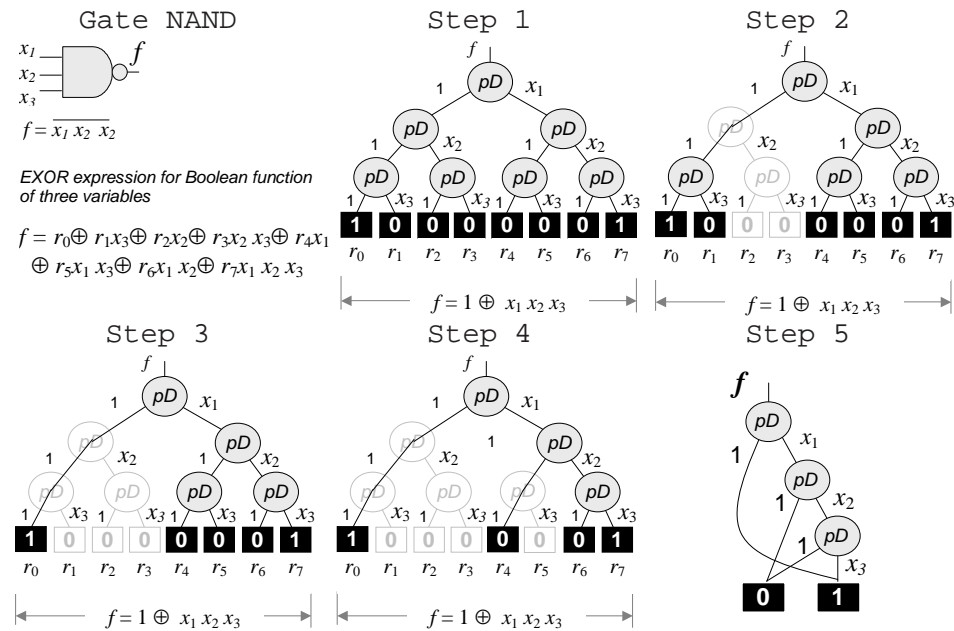


Fig. 18. Functional decision diagram design using pD nodes for the three-variable NAND function (Example 28).

- The new notations such as “polarized literals and minterms”, and “operational and functional domains” are introduced,
- The material is build around the notation of the *data structure*, which allows introducing SOP and polynomial expressions as alternative and interchangeable forms of Boolean functions. The relationships between various data structures and their manipulation through design represent the most important aspect of contemporary logic design.
- Similar to the SOP-based techniques, the local transformations, factorization, and observability for logic networks with EXOR gates are considered.

Moreover, the SOP and Reed-Muller techniques are illustrated using decision diagrams. Reed-Muller techniques are supported by about 200 examples, practice problems with solutions, and problems. The advanced techniques such as spectral approach and related techniques [5, 8] are placed in the “Further study” section.

At the same time, we believe that the new techniques must be introduced bearing in mind that the material for this introductory courses assume no specific prerequisites nor any knowledge of electrical circuits or electronics, in order to satisfy the requirements of interdisciplinary interest in logic design.

Acknowledgments

Prof. R. S. Stanković provided many useful remarks on the polynomial techniques when reading the textbook.

References

- [1] D. D. Givone, *Digital Principles and Design*. New York: McGraw-Hill, 2003.
- [2] M. M. Mano and C. R. Kime, *Logic and Computer Design Fundamentals*, 3rd ed. Prentice Hall, 2004.
- [3] D. H. Green, *Modern Logic Design*. Addison-Wesley, 1986.
- [4] T. Sasao, *Switching Theory for Logic Synthesis*. Dordrecht: Kluwer, 1999.
- [5] R. S. Stanković and J. T. Astola, *Spectral Interpretation of Decision Diagrams*. Springer, 2003.
- [6] S. N. Yanushkevich, V. P. Shmerko, and S. E. Lyshevski, *Logic Design of Nano ICs*. CRC Press, 2005.
- [7] S. N. Yanushkevich and V. P. Shmerko, *Introduction to Logic Design*, 2008.
- [8] S. N. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. S. Stanković, *Decision Diagram Techniques for Micro- and Nanoelectronic Design -Handbook*, 2006.