# Extended Spectral Techniques for Logic Synthesis

*Ingo Schäfer* [+], *Marek A. Perkowski*

*Portland State University, Department of Electrical Engineering*

*P.O. Box 751, Portland, OR 97207-0751*

## 1. INTRODUCTION

With the introduction of discrete orthogonal transforms to digital logic it was attempted to apply them to unify the logic synthesis [1-5]. Their shortcomings, i.e. the computational complexity, enormous memory requirements and difficult application to incompletely specified Boolean functions, did not allow to use these techniques in design automation. However, the combination of classical logic techniques with spectral techniques, the main contribution of this chapter, overcomes these shortcomings and makes it superior to strict classical and spectral methods.

In logic synthesis the interest is in the realizations of logic functions that lead to minimal circuit realizations with respect to the area and the delay of the circuit. The most common criteria for the realizations to be minimal in the sense of their minimal circuit cost is their literal count [6-8] or the number of terms [9,10]. However the possible realizations for a logic function are so numerous that it is nearly impossible to find the minimal among them. Therefore, there is a strong interest in normal realizations which are unique [11]. They are also called *canonical forms* [11]. The interest is to find a set of *canonical forms*, where one of the forms has a close to minimal circuit realization. Thus, the computational effort to find the minimal circuit realization by investigating all possible forms is reduced to that of finding the minimal form among a set of *canonical forms* of a certain type.

In signal processing and image processing unique forms obtained by discrete orthogonal and unitary

[+] The work presented in this Chapter was partially supported by NSF grant MIP-9110772

transforms are important tools [5,12,13]. The transform pair for unitary/orthogonal transforms is described by

$$X(k) = \sum_{n=0}^{N-1} x(n)\, g_k^{\,*}(n) \tag{1}$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, g_k(n) \tag{2}$$

where * denotes conjugated complex, X(k) is the vector of $N$ values in the transform domain, and x(n) is the set of values describing the original function in the object domain [5]. The $N$ transform functions $g_k(n)$ describe an unitary/orthogonal $N \times N$ transform matrix $G$ where the transform matrix G is given by

$$[\,G\,] = \begin{bmatrix} g_0^{\,*}(0) & \dots & g_0^{\,*}(N-1) \\ & \dots & \\ \dots & g_i^{\,*}(j) & \dots \\ & \dots & \\ g_{N-1}^{\,*}(0) & \dots & g_{N-1}^{\,*}(N-1) \end{bmatrix} \tag{3}$$

The transform pair given by Equation (1) and (2) can also be represented by the following matrix multiplications

$$X = [\,G\,]\, x \tag{4}$$

$$x = [\,G\,]^{-1} X \tag{5}$$

The most popular discrete orthogonal transforms are the Discrete Fourier, Hadamard-Walsh, Sine- and Cosine Transform [5,12]. For instance the set of linearly independent functions for the discrete Fourier transform is given by

$$g_k(n) = e^{-j\frac{2\pi\omega kn}{N}} \tag{6}$$

where ω is the frequency variable.

This chapter discusses some aspects of orthogonal and nonorthogonal discrete transforms in the special area of logic design. In particular, discrete transforms for multiple-valued input, binary output logic are investigated. Therefore, first the concept of multiple-valued input, binary output functions is introduced. The following section introduces the classification of the application of spectral techniques for logic synthesis into three areas. It presents the Switching Function (SF) spectrum which is an extension of known orthogonal spectra to any nonorthogonal transforms described by a set of logic functions. Then, as an illustration of the developed theory, section 4 introduces an approach to logic synthesis using multiplexer gates and the mapping of such circuits to the Field Programmable Gate Array (FPGA) from Actel [14]. Finally, section 5 gives some concluding remarks and the outlook for further research in this area.

## 2. MULTIPLE-VALUED INPUT, BINARY OUTPUT FUNCTIONS

This section introduces the basic concepts of multiple-valued input, incompletely specified binary output functions and their vector notation.

*Definition 1:* A multiple-valued input, incompletely specified binary output function *(mv function* for short) is a mapping $f(X) = f(X_1, X_2, \ldots, X_n): R_1 \times R_2 \times, \ldots, R_n \rightarrow B$, where $X_i$ is a multiple-valued variable that takes the values from the set $R_i = \{0, 1, \ldots, p_i - 1\}$ where $p_i$ is the number of values of the variable, and $B = \{0, 1, -\}$.

Such functions found several practical applications in logic design and are implemented in commercial CAD tools by means of multi-level logic realizations with binary gates[15,16,17,18,19].

*Definition 2:* A *literal* of the multiple-valued input variable $X_i$, denoted by $X_i^{S_i}$, is defined as:

$$X_i^{S_i} = \begin{cases} 1 & \text{if } X_i \in S_i \\ 0 & \text{if } X_i \notin S_i \\ - & \text{if } X_i \text{ is not specified} \end{cases} \tag{7}$$

A literal of an $mv$ variable with a single value will be called a *single-valued literal*. A *product of literals*, $X_1^{S_1} X_2^{S_2} \ldots X_n^{S_n}$, is referred to as a *product term* (also called *term* for short). A product term consisting of single-valued literals of all variables is called a *minterm*.

*Definition 3:* The truth vector representation $d$ of an $mv$ function is a vector of minterms in a straight n-ary order.

Example 1 illustrates the truth vector representation of an $mv$ function.

*Example 1:* The truth vector $d$ for a two variable $mv$ functions where variable $X_1$ is three-valued and variable $X_2$ is a four-valued one, is given by

$$\left[\; X_1^0 X_2^0 ,\, X_1^0 X_2^1 ,\, X_1^0 X_2^2 ,\, X_1^0 X_2^3 ,\, \ldots,\, X_1^2 X_2^0 ,\, X_1^2 X_2^1 ,\, X_1^2 X_2^2 ,\, X_1^2 X_2^3 \;\right]$$

An n-variable *mv* function f(X) can be represented by $N = \prod_{i=0}^{n-1} p_i$ values which determine if the function is true, false, or not specified. It follows for Boolean functions that $N$ is equal to $2^n$. There exist two codings for the vector of truth values of a logic function. They are called $S$ and $R$ coding [4,20] where a <true, false, don’t care> (ON, OFF, DC) minterm of a logic function is represented by the value <1, 0, 0.5> for the $R$ coding and <-1, 1, 0> for the $S$ coding.

## 3. SPECTRAL TECHNIQUES FOR THE LOGIC SYNTHESIS OF *MV* LOGIC

First, let us review the concepts of the application of orthogonal spectra in logic synthesis. The application of spectral techniques in logic synthesis can be categorized into three concepts: decomposition of *mv* functions based on the correlation given by a single spectrum, circuit realizations of *mv* functions based on the repetitive calculation of spectra and circuit realizations obtained by applying Galois Field (2) (GF 2) transforms.

### 3.1. Orthogonal Spectra for *mv* Functions

#### 3.1.1. Application of Orthogonal Spectra

One application of orthogonal transforms in logic synthesis is to obtain a spectrum which gives a correlation of the *mv* function to a certain set of linearly independent *mv* functions $g_k$. For this special application the general orthogonal expansion formula given by Equation (2) is restricted to the transform matrix $G$ given by Equation (3) with *mv* functions $g_k$ being linearly independent *mv* functions. Therefore, if the *mv* functions are described in $R$ coding, the transform matrix $G$ has 1, -1, and 0 entries only. Thus, $g_k$ is real: $g_k = g_k^*$. The restriction of the transform matrix $G$ to have only 1, -1, and 0 as the elements leads to transform matrices such as those considered in [21,22] (which include the Walsh-type transforms, the Arithmetic and Adding transforms [23,24]). In logic synthesis such transforms can be applied to the decomposition of an *mv* function based on the correlation information obtained by the spectrum. One of the most important application is the linearization of Boolean functions [25,26]. In the next section a special computation method for the transforms with singular and nonsingular transform matrices $G$ having only entries 0, 1, and -1 is discussed.

### 3.1.2. Circuit Realization Based on Spectra

Another goal in logic synthesis is to express the $mv$ function f(X) by a set of completely specified $mv$ functions $g_k$ where the circuit realization of the set of $mv$ functions $g_k$ is smaller than the circuit realization of f(X). Thus, the matrix $G$ can have entries 1, -1, 0 only, $g_k(n) \in \{ 0, 1, -1 \}$. Because the value of $X(k)$ determines if $g_k$ is a function of the form or not, $X(k)$ is restricted to $X(k) \in \{ 0, 1 \}$. Of special interest are two circuit realizations of $mv$ functions: with OR ($\bigcup$) and AND ($\bigcap$) gates, the Field ($\bigcup$, $\bigcap$); and with EXOR ($\oplus$) and AND gates, the Field ($\oplus$, $\bigcap$). A canonical expansion of an $mv$ function is the expansion where any minterm of the $mv$ function is obtained in a unique way. Thus, a minterm representation of an $mv$ function is canonical. It follows directly, that for a two-level Sum of Product (SOP) realization of an $mv$ function the only canonical form is the minterm representation. To show this formally we apply Equation (1) and Equation (2) to an $mv$ function f($x_0$, ..., $x_{N-1}$) = f(X) over the field F($\bigcup$, $\bigcap$):

$$X(k) = \bigcup_{n=0}^{N-1} f(n) \bigcap g_k(n) \tag{8}$$

$$f(n) = \bigcup_{k=0}^{N-1} X(k) \bigcap g_k(n) \tag{9}$$

where $X(k) \in \{ 0, 1 \}$ is '1' for $g_k(n)$ being a function of the form. It follows from Equation (9) and the definition of the intersection, that $X(k) \in \{ 0, 1 \}$. Therefore, the transform matrix $G = \{ g_k(n) \}$ has to be the identity matrix $I$ or matrix $G$ obtained by any permutation of rows or columns from the identity matrix $I$. Thus, the minterm representation is the only two-level canonical form over the field F($\bigcup$, $\bigcap$). However, a transform does not have to be applied to all variables of the function at once. The simplest case to obtain a multi-level realization is to apply the expansion at a time only to one variable $x_i$ of the function

$$f(n) = ( x_i \bigcap f(n) ) \bigcup ( \overline{x_i} \bigcap f(n) ) \tag{10}$$

This special case is the well known Shannon expansion [11]

$$f(x_0, ..., x_n) = x_i \ f(x_0, ..., 1, ..., x_n) + \bar{x}_i \ f(x_0, ..., 0,..., x_n) \tag{11}$$

The Shannon expansion has been generalized in [15] to AND-OR type multiple-valued logic. In [27,28] the Shannon expansion has been generalized for Boolean rings. Multi-level circuits can be obtained by the recursive application of such kind of expansions. Other multi-level canonical expansions are also possible such as the one illustrated in Example 2.

*Example 2:* A unique three-level expansion for a four-variable function $f$ is given by the linearly independent functions $g_i$ and $h_i$ illustrated in Figure 1.

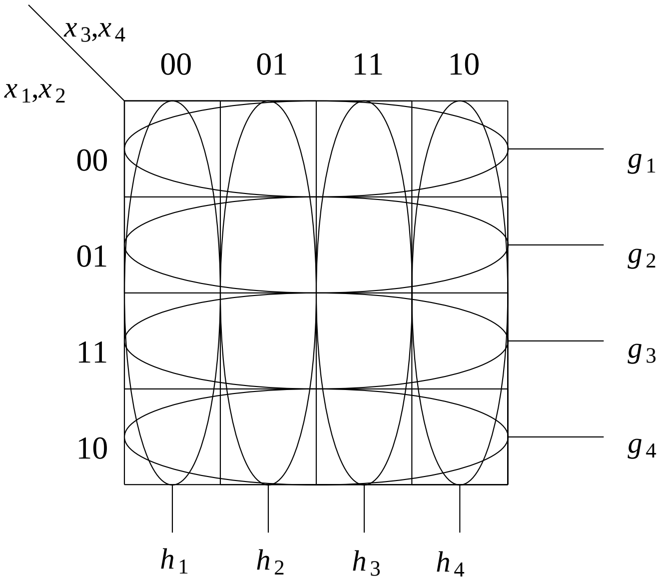



Figure 1. Unique expansion for a four-variable function.

As one can observe from Figure 1, each minterm of the Karnaugh map can be obtained uniquely by the intersection of a function $g_i$ with a function $h_j$

$$f = \bigcup_{i=0}^{3} \bigcup_{j=0}^{3} f \cap g_i \cap h_j$$

One application of such an expansion is the multiplexer synthesis algorithm introduced in section 4

**3.1.3. Circuit Realization Based on GF 2 Transforms.**

The third concept based on spectral methods applied in logic synthesis are transforms over GF 2, such as the canonical Reed-Muller forms [28-31]. The property of such transforms is that they directly describe a two-level AND-EXOR circuit realization of the underlying $mv$ function. The canonical expansions over the GF 2 (F(⊕, $\cap$) are given by

$$X(k) = \bigoplus_{n=0}^{N-1} x(n) \cap g_k(n) \tag{12}$$

It follows from the definition of ⊕ (1 ⊕ 1 = 0, 1 ⊕ 0 = 1) that $X(k) \in \{ 0, 1 \}$ for any function $g_k$. The inverse transform is given by

$$x(n) = \bigoplus_{k=0}^{N-1} X(k) \cap g_k(n) \tag{13}$$

Equations (12) and (13) hold true for any orthogonal matrices over the GF 2 with $g_k(n) \in \{ 0, 1 \}$, where k,n = 0, ..., N-1.

As one can observe, there is only one canonical form for a two-level realization of a function in the Field ($\cap$, $\cup$), while in the Field (⊕ , $\cap$) there are as many canonical forms for two-level realizations as there are orthogonal matrices G = { $g_k(n)$ }. The following Theorem gives the special case $X(k) = f(X)$.

<u>*Theorem 1:*</u> A multiple-valued input, binary output function $f$ can be represented by Equation (14)

$$f = \bigoplus_{k=0}^{N-1} \left[ g_k \cap f \right] \tag{14}$$

where the $N-1$ functions $g_k$ form the orthogonal transform matrix $G$ with the property

$$\bigoplus_{k=0}^{N-1} g_k = 1 \ . \tag{15}$$

*Proof:*

$$f = \bigoplus_{k=0}^{N-1} \left[ g_k \cap f \right] = f \cap \left[ \bigoplus_{k=0}^{N-1} g_k \right] = f \cap 1 = f \qquad \square$$

The expansion obtained by Equation (14) is a particular case of the orthonormal expansions presented in [32].

If the $mv$ functions $g_k$ are mutually disjoint ($g_i \cap g_j = \emptyset$) then the set $G$ of $mv$ functions has the property

$$\bigoplus_{k=0}^{N-1} g_k = \bigcup_{k=0}^{N-1} g_k = 1 \tag{16}$$

where the $mv$ functions $g_k$ are mutually disjoint ($g_i \cap g_j = \emptyset$). Thus, based on the property of set $G$ according to Equation (16) the expansion given by Equation (14) can be expressed by

$$f = \bigcup_{k=0}^{N-1} \left[ g_k \cap f \right] \tag{17}$$

which is an unique AND-OR type Shannon expansion for a multi-level realization, see Example 2.

This section showed the three general concepts of spectral methods that are applied in logic synthesis. A well known application of the first concept is the linearization procedure [25,26] based on the Rademacher-Walsh and the Autocorrelation spectrum. It has been also generalized for incompletely specified multi-output Boolean functions [33]. The two other concepts are based on the Shannon expansion over the Field ($\cap$, $\cup$) and the Davio expansions [27] over the Field ($\oplus$, $\cap$). The Shannon expansion which is an important tool in multi-level logic synthesis, was generalized in this section to $mv$ functions. For the Field ($\cup$, $\cap$) a hierarchical multiplexer synthesis based on the Shannon expansion given by Equation (17) will be presented in the following section. For the third concept: circuit realization based on GF 2 transforms, the $mv$ functions $g_i$ describing the orthogonal transform matrix $G$, Equation (13) can be restricted to single variable $mv$ functions. Thus, Equation (13) describes the three Davio Expansions [27]. Hence, the application of Equation (13) in Boolean domain leads to the canonical Kronecker Reed-Muller (KRM) forms [27-29] having the property given by Equation (15) and (16). For $mv$ logic the restriction to single variable $mv$ functions leads to the Multiple-Valued Input GRM and KRM forms [31,33,34].

### 3.2. ORTHOGONAL AND NONORTHOGONAL SPECTRA FOR *MV* FUNCTIONS

New methods are presented for the calculation of unique and nonunique spectra for *mv* functions based on the disjoint representations as cubes and Decision Diagrams. The difficult incorporation of incompletely specified *mv* functions into the spectral synthesis methods is overcome by the introduction of the *SF* spectrum.

The conversion of a Boolean function, given in a truth table representation $F$, by the multiplication of a nonsingular orthogonal transform matrix will be called spectrum [1,4]. For the sake of brevity the conversion of an *mv* function by singular or nonorthogonal transform matrices is also called spectrum. However, such a spectrum does not have to be unique.

*Definition 4:* The Switching Function (SF) transform of an incompletely specified $n$-variable *mv* function is defined as

$$C = [\ SF\ ] \times F \tag{18}$$

where the row vectors of the transform matrix *SF* describe *mv* functions such that for the special case of Boolean functions *SF* is an non/orthogonal $2^n \times k$ transform matrix [3-5,22], where $k$ is an arbitrary number, and that for *mv* functions *SF* is an orthogonal $\sum_{i=0}^{2^n} p_i \times \sum_{i=0}^{2^n} p_i$ or a nonorthogonal $\sum_{i=0}^{n} p_i \times k$ matrix. $F$ is the vector of truth values for the function f(X), and $C$ is the vector of spectral coefficients.

Such a description of transform matrices allows for the development of special transforms in logic synthesis. One particular application are transforms to compute spectra that give a correlation of the underlying *mv* function to a limited set of *mv* functions, e.g. the realizable functions of a logic block in Field Programmable Gate Arrays (FPGAs) [35]. Examples are the Table-Look-Up (TLU) architectures from Xilinx and AT&T [36] and the Multiplexer Based (MB) architectures from Actel [14]. Section 5 illustrates such a transform for the Actel $ACT^{TM}$ macrocells.

In the area of logic design mostly the Walsh-type transforms have been investigated [1,3-5,23]. Therefore, the calculation of the Rademacher-Walsh spectrum, which is one of the several Walsh-type transforms, is used to illustrate the computation of a spectrum by matrix calculation.

*Example 3:* The Rademacher-Walsh transform matrix and the spectrum of the three-variable function $f(X) = x_1 x_3 + \bar{x}_1 x_2$ is shown in Figure 2. The $S$ coding is used for the vector of truth values $F$ of the function f(X).

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1
\end{bmatrix}
\text{ x }
\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}
=
\begin{array}{cc}
\begin{bmatrix} 0 \\ 0 \\ 4 \\ 4 \\ 4 \\ -4 \\ 0 \\ 0 \end{bmatrix} &
\begin{array}{l} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_{12} \\ s_{13} \\ s_{23} \\ s_{123} \end{array}
\end{array}
$$

Figure 2. Spectrum by matrix calculation.

Recently, transforms for *mv* functions and their applications to logic minimization have been introduced [30,31,34,37,38]. The next example illustrates the possibility of defining a nonorthogonal transform for *mv* functions.

*Example 4:* The *mv* function $f = X^2Y^{01} + X^0Y^{34}$, Figure 3, consists of a three-valued literal $X$ and a five-valued literal $Y$. Thus, the function can be described by a vector of $3 \times 5$ minterms. The example of an nonorthogonal multiple-valued transform matrix for this function is given in Figure 4.

| X \ Y | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | | 1 | 1 |
| 1 | | | | | |
| 2 | 1 | 1 | | | |

Figure 3. Map of function $f = X^2Y^{01} + X^0Y^{34}$.

The calculation of the transform by matrix multiplication is shown in Figure 4.

$$
\begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&1&1&1&1&1&0&0&0&0&1&0&0&0&0\\
1&0&1&1&1&0&1&0&0&0&0&1&0&0&0\\
1&1&0&1&1&0&0&1&0&0&0&0&1&0&0\\
1&1&1&0&1&0&0&0&1&0&0&0&0&1&0\\
1&1&1&1&0&0&0&0&0&1&0&0&0&0&1\\
1&0&0&0&0&0&1&1&1&1&1&0&0&0&0\\
0&1&0&0&0&1&0&1&1&1&0&1&0&0&0\\
0&0&1&0&0&1&1&0&1&1&0&0&1&0&0\\
0&0&0&1&0&1&1&1&0&1&0&0&0&1&0\\
0&0&0&0&1&1&1&1&1&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&0&1&1&1&1\\
0&1&0&0&0&0&1&0&0&0&1&0&1&1&1\\
0&0&1&0&0&0&0&1&0&0&1&1&0&1&1\\
0&0&0&1&0&0&0&0&1&0&1&1&1&0&1\\
0&0&0&0&1&0&0&0&0&1&1&1&1&1&0
\end{bmatrix}
\text{ x }
\begin{bmatrix}
0\\0\\0\\1\\1\\0\\0\\0\\0\\0\\1\\1\\0\\0\\0
\end{bmatrix}
=
\begin{bmatrix}
4\\2\\0\\2\\1\\1\\0\\1\\1\\3\\3\\2\\1\\1\\1\\1\\0\\1\\1\\1\\1\\2\\3\\3
\end{bmatrix}
\begin{array}{l}
1\\ X^0\\ X^1\\ X^2\\ Y^0\\ Y^1\\ Y^2\\ Y^3\\ Y^4\\
X^0 \oplus Y^0\\ X^0 \oplus Y^1\\ X^0 \oplus Y^2\\ X^0 \oplus Y^3\\ X^0 \oplus Y^4\\
X^1 \oplus Y^0\\ X^1 \oplus Y^1\\ X^1 \oplus Y^2\\ X^1 \oplus Y^3\\ X^1 \oplus Y^4\\
X^2 \oplus Y^0\\ X^2 \oplus Y^1\\ X^2 \oplus Y^2\\ X^2 \oplus Y^3\\ X^2 \oplus Y^4
\end{array}
$$

Figure 4. Multiple-valued transform of function $f$ .

The next section presents the meaning of the values of the spectral coefficients by their relation to *mv* functions. Additionally, computation methods are introduced for orthogonal and nonorthogonal transforms having any matrix with {0, 1, -1} as entries. This allows the definition of spectra based on a given set of *mv* logic functions rather than using one of the known discrete transforms.

### 3.2.1. RELATIONS OF SPECTRAL TECHNIQUES TO CLASSICAL LOGIC

The relations of spectral coefficients to classical logic have been shown for the special case of Walsh-type [4,39,40], Adding and Arithmetic [41], and Reed-Muller [42,43] transforms. Each spectral coefficient is calculated by the multiplication of a row vector of the transform matrix $SF$ and the function vector $F$, as in Equation (18). This leads to the development of algorithms to compute the Walsh and Reed-Muller spectrum from an arithmetic cover [26,44], disjoint representations [43,45-48], and the computation of the Arithmetic and Adding spectrum from a minterm representation [24,49]. The methods introduced there are specific for the respective transforms. However, the underlying concept can be generalized to any orthogonal or nonorthogonal transform having {-1, 1, 0} as entries in the transform matrix. To be able to introduce the generalization, first the concept of *standard trivial functions* [20,39] will be extended.

*Definition 5:* The *Positive Standard Trivial Function (PSTF)* $p_I$ is the $mv$ function represented by the minterms defined by the {1} entries of the corresponding row vector $SF[I]$ of the transform matrix $SF$.
The *Negative Standard Trivial Function (NSTF)* $n_I$ is the $mv$ function represented by the minterms defined by the {-1} entries of the corresponding row vector $SF[I]$ of the transform matrix $SF$.

It should be observed that a $PSTF$ or $NSTF$, being an $mv$ function, can be represented by a set of disjoint cubes [33,40,50], Binary Decision Diagrams (BDD) [33,47,48,51], or minterms [33].

*Example 5:* The $PSTF$ for the second row of the transform matrix shown in Figure 2 is given by $p_2 = x_1$ and the corresponding $NSTF$ by $n_2 = \bar{x}_1$.

Definition 6 extends the notation and definitions for the calculation of the values of the spectral coefficients introduced in [20,39] for $PSTF$ and $NSTF$.

*Definition 6:*

$a_I$ number of true minterms of a $mv$ function f(X) covered by the $PSTF$ $p_I$.

$b_I$ number of false minterms of a $mv$ function f(X) covered by the $PSTF$ $p_I$.

$c_I$ number of true minterms of a *mv* function f(X) covered by the *NSTF* $n_I$.

$d_I$ number of false minterms of a *mv* function f(X) covered by the *NSTF* $n_I$.

$e_I$ number of don't care minterms of a *mv* function f(X) covered by the *PSTF* $p_I$.

$f_I$ number of don't care minterms of a *mv* function f(X) covered by the *NSTF* $n_I$.

where $I$ is the row number of the corresponding transform matrix.

One distinguishes between two types of transforms [23,41].

*Definition 7:* A transform with a transform matrix having only {1, -1} as elements is called a *global* transform.

*Definition 8:* A transform with a transform matrix having at least one {0} as element is called a *local* transform.

The spectral coefficients of a *global* transform contain information about the whole *mv* function while the spectral coefficients of a *local* transform contain only information about parts of the *mv* function. The next section presents an application of the *local* transform to multiplexer synthesis.

The value of a spectral coefficient for the Walsh spectrum of an *mv* function in $S$ coding is given by [23]:

$$s_I = (a_I - b_I) - (c_I - d_I) \tag{19}$$

for the $R$ coding by:

$$r_I = (a_I - c_I) + \frac{1}{2}(e_I - f_I) \tag{20}$$

Other formulas derived from Equations (19) and (20) can be found in [23,45]. For a *global transform* the values $b_I$, $d_I$, and $f_I$ can be obtained directly from the total number of true, or don't care minterms of the function, and parameters $a_I$, $c_I$, and $e_I$. In the case of a transform over the Galois Field (2) [28] the Equations (19) and (20) are not valid. There, the value of a spectral coefficient $C_I \in \{0, 1\}$ is determined by $r_I$, or $s_I$ being even or odd.

*Example 6:* The eight *PSTF* s of the Rademacher-Walsh transform matrix in Example 3 are described by the circled areas in Figure 5. Because the Walsh-type transforms are *global* transforms it is not necessary to determine the *NSTF* s for them.

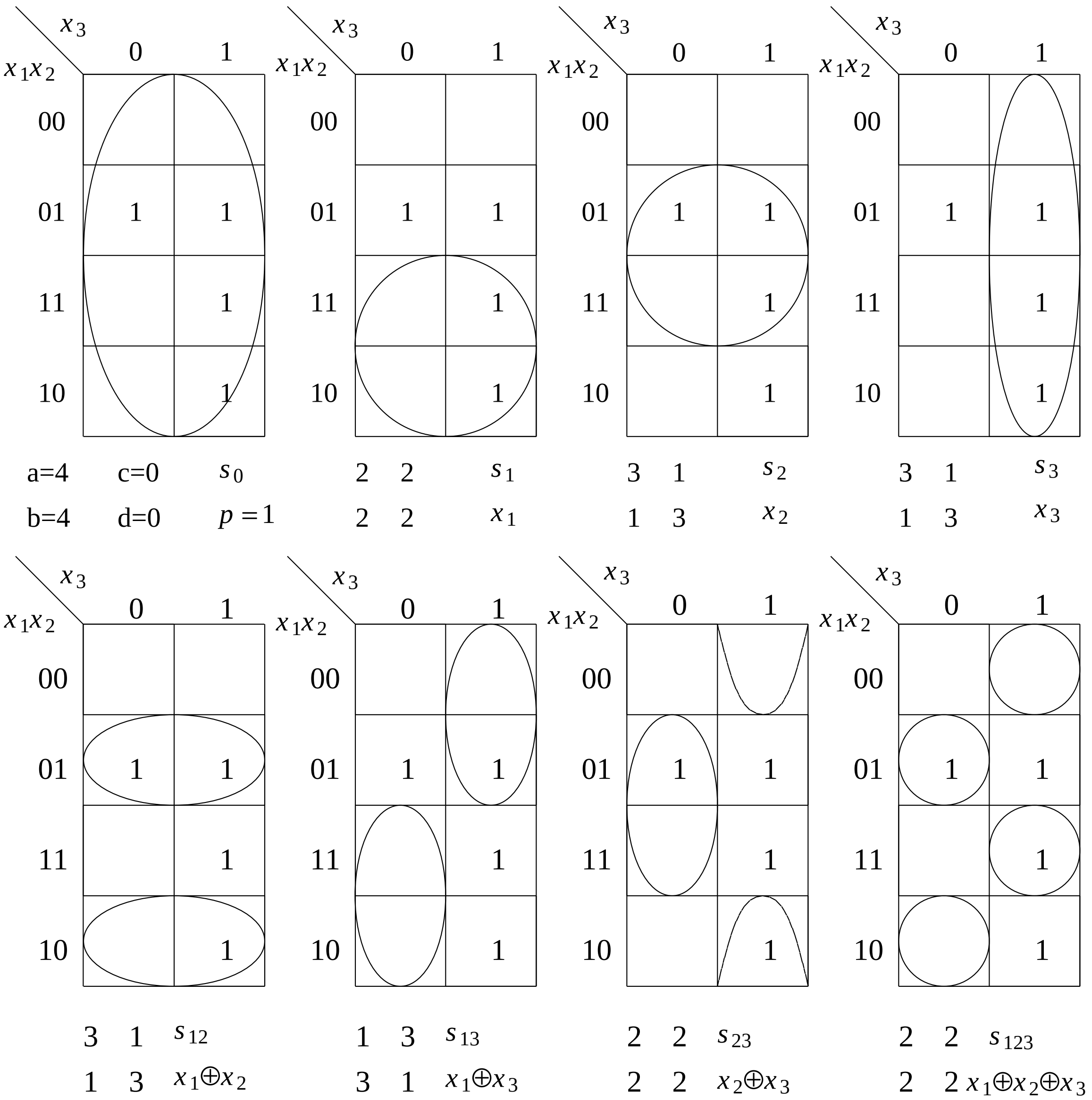



Figure 5. *PSTF* s for the Rademacher-Walsh transform.

The values of the spectral coefficients can be calculated according to Equation (19). The values of $a$, $b$, $c$, and $d$ are given in Figure 5. The same values are obtained as in Example 3 as one can easily verify. The values of the spectral coefficients can either be calculated by the graphical method, counting the number of minterms inside or outside the *standard trivial functions*, or by the corresponding cube calculus method introduced in the previous section. The intersection of the *PSTF* and the *mv* function in a disjoint representation has to be performed. Then the value of $a_I$, $b_I$, $c_I$, $d_I$, $e_I$, $f_I$ can be determined from the intersection. Next the value of the spectral coefficients $s_I$, or $r_I$ can be calculated according to Equation (19) or (20).

### 3.2.2. THE T-CODING

Spectra in the $R$- or $S$ coding for *mv* functions do not allow to determine the direct correlation of the function to a *PSTF* or *NSTF*. Especially in the case of incompletely specified *mv* functions the contribution of the not specified part of the function to the single spectral coefficient can not be determined directly because the information is spread over the complete spectrum. This section introduces the concept of the $T$ coding to overcome disadvantages of the $R$ and $S$ codings.

*Definition 9:* The spectral coefficient $t_I$ of the *SF*-spectrum for a *local* transform of a completely specified *mv* function is given by the pair of values $t_I = <a_I, b_I>$; for an incompletely specified *mv* function by the quadruple of values $t_I = <a_I, b_I, e_I, f_I>$.

The $R$ or $S$ coding for a *local* transform can be obtained from the $T$ coding of the *SF* spectrum by Equations (19) or (20).

*Property 1:* The spectral coefficient $t_I$ in the $T$ coding for a *global* transform of a completely specified *mv* function is given by the value $t_I = <a_I>$; for an incompletely specified *mv* function by the pair of values $t_I = <a_I, e_I>$.

*Property 2:* For a *global* transform of a completely specified *mv* function Equation (19) becomes

$$s_I = 2 \times ( a_I - b_I ) \qquad (21)$$

If we denote the number of minterms covered by the *PSTF* by $pt$ we obtain

$$s_I = 4 \times a_I - 2 \times pt \qquad (22)$$

Thus, Equation (22) gives the relation of the $T$ coding for a *global* transform to the $S$ coding.

*Property 3:* The values $a_f$ and $e_f$ of an incompletely specified *mv* function $f$ are given by the number of ON and DC minterms covered by the function. Thus, with Equation (20) we obtain

$$r_I = 2\,a_I + e_I - a_f - \frac{1}{2}\,e_f \tag{23}$$

which can be calculated directly from the $T$ coding $t_I$.

It can be observed from Property 2 and 3, that the information obtained by the $S$ or $R$ coding of a spectrum can be easily obtained from the $T$ coding. However, the $T$ coding gives additional information about the correlation of an $mv$ function to a *PSTF* or *NSTF*.

*Property 4:* The value of a spectral coefficient $t_I$, an ordered n-tuple, of the $SF$ spectrum for a *global* transform is given by the values $a$ and $e$ of the intersection of the $mv$ function with the $PSTF$. For a *local* transform the spectral coefficient $t_I$ includes additionally the values $c$ and $f$ determined by the intersection of the $mv$ function with the $NSTF$.

*Example 7:* The Walsh spectrum in $T$ coding is determined for the function in Example 3. Only $a_I$ has to be calculated because the function in Example 3 is completely specified, and the Rademacher-Walsh transform is *global*. The value of $a_I$ for each spectral coefficient is given in Figure 5. The value for the $S$ coding of the spectrum given in Figure 2 can be obtained by Equation (22), where $pt = 8$ for $s_0$ and $pt = 4$ for all other coefficients.

The introduced $T$ coding separates the information obtained for the completely specified and the not specified part of the underlying $mv$ function. Thus, the extension of the linearization algorithm [1] introduced in [33], and the multiplexer synthesis algorithm introduced in the next section, taking advantage of the $T$ coding, can directly determine the contribution of the not specified part of the $mv$ function.

The next section presents the computation of a spectrum from the Ordered Multiple-valued Decision Diagram (OMDD) representation [34,52] of the $mv$ function. For the case of Boolean functions Ordered Binary Decision Diagrams (OBDDs) are obtained [51,53-55].

### 3.2.3. CALCULATION OF SPECTRA FROM OMDDS

The method for the calculation of spectral coefficients introduced in the previous section is based on the calculation of the intersection of the *PSTFs* and the *NSTFs* with the given $mv$ function. Therefore, the intersection and the tautology of two functions have to be computed frequently. Because these operations can be performed relatively fast using OMDD representations[55,52], the OMDDs are very well suited for this application. Moreover, in the case of orthogonal and nonorthogonal transforms where no Fast Transforms [5] exist they are even more favorable than the transforms based on matrix multiplication. Nonorthogonal transforms include also the case of the computation of selected spectral coefficients from an orthogonal transform.

Up to now only two algorithms have been introduced for spectral methods based on OBDDs [47,48]. The applied properties, which are similar to the ones used for the calculation from the disjoint cube representation [40,50] are specific to the Walsh-type and Reed-Muller transforms, while the approach shown here is general and can be applied to any $SF$ transform.

To take for the calculation of the $T$ coding of a spectral coefficient efficiently advantage of the OMDDs, we introduce a modification to them. In each node of the OMDD the number of ON minterms ($a$), and DC minterms ($f$), covered by the subsequent nodes is calculated. The next two examples illustrate the calculation of a spectral coefficient from the OBDD representation of a completely and of an incompletely specified Boolean function.

*Example 8:* The Boolean function $f(X)$ given in Example 3, and the $PSTF$ $p = g(X) = x_1 \oplus x_2$ is taken to illustrate the computation of a spectral coefficient. The $PSTF$ is the representation for the spectral coefficient $s_{12}$ of the Rademacher-Walsh transform. Because the function f(X) is completely specified and the Rademacher-Walsh transform is a *global* transform, the values of the spectral coefficients in $T$ coding are fully determined by the number of minterms $a_{12}$ of the OBDD obtained by the intersection of $f(X)$ and $g(X)$ (Property 1). The OBDD of the result of the intersection $f(X) \cap g(X)$ is shown in Figure 6. The spectral coefficient $t_{12}$ is given by the the number of minterms $a_{12}$ of the intersection: $t_{12} = <a_{12} = 3>$. The number of minterms covered by the $PSTF$ is $pt = 4$, Figure 4. Then the value of the spectral coefficient $s_{12}$ is obtained by Equation (22): $s_{12} = 4 \times a_{12} - 2 \times pt = 4$, which has been also obtained in Example 3.

The complete nonorthogonal or orthogonal spectrum can be calculated by performing the above shown operations for every $PSTF$ describing the transform.

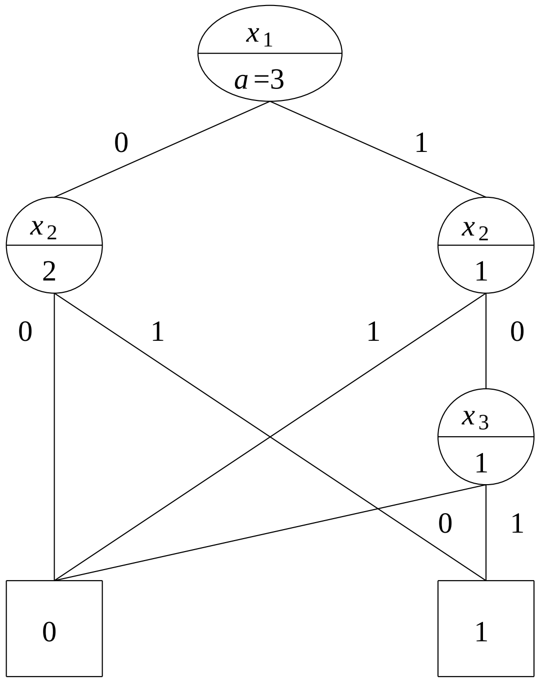



Figure 6. OBDD for the result of intersection between $f(X)$ and $g(X)$: $f(X) \cap g(X)$.

For the case of an incompletely specified $mv$ function, the $T$ coding or the $R$ coding has to be used. Thus, the number of don't care minterms covered by a node in the OBDD has to be calculated as well.

*Example 9:* The OBDD of a Boolean function $f(X)$ with the completely specified part $f_c(X) = x_1x_2\bar{x}_3 + \bar{x}_1x_2$ and the not specified part $f_{in}(X) = x_1x_2x_3$ and the values for $a$ and $e$ are shown in Figure 7. The intersection of $f(X)$ and the $PSTF$ $g(X) = x_1$ has to be performed (Figure 8) to calculate the spectral coefficient $t_1$ or $r_1$. The intersection of the function $f(X)$ and the $PSTF$ $g(X)$ is a complete subtree of the initial function $f(X)$. Thus, the values for $a$ and $c$ do not have to be recalculated. By Property 4, the spectral coefficient $t_1$ is given the values $a$ and $e$ of $f(X) \cap g(X)$: $t_1 := <a, e> = <1, 1>$. The values $a$ and $e$ for $f(X)$ have to be determined to calculate the spectral coefficient $r_1$ from the spectral coefficient $t_1$ by Equation (23). The values $a_f$ and $e_f$ for $f(X)$ are given in Figure 7: $t_1 := <a_f, e_f> = <3, 1>$. Thus,

$$r_1 = 2 \times 1 + 1 - 3 - \frac{1}{2} 1 = -0.5$$

The Walsh-type transforms [4,23] have certain properties which allow the development of special

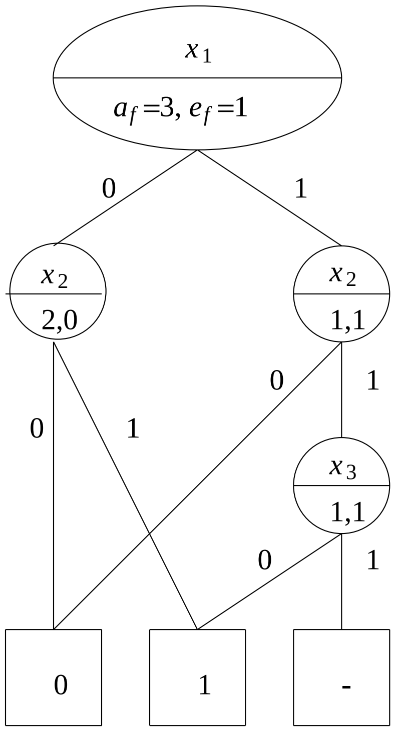



Figure 7. OBDD of an incompletely specified Boolean function, Example 9.

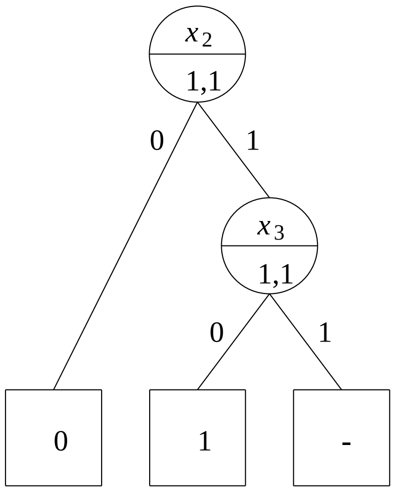



Figure 8. Intersection of the incompletely specified function $f(X)$ and $g(X)$, Example 9.

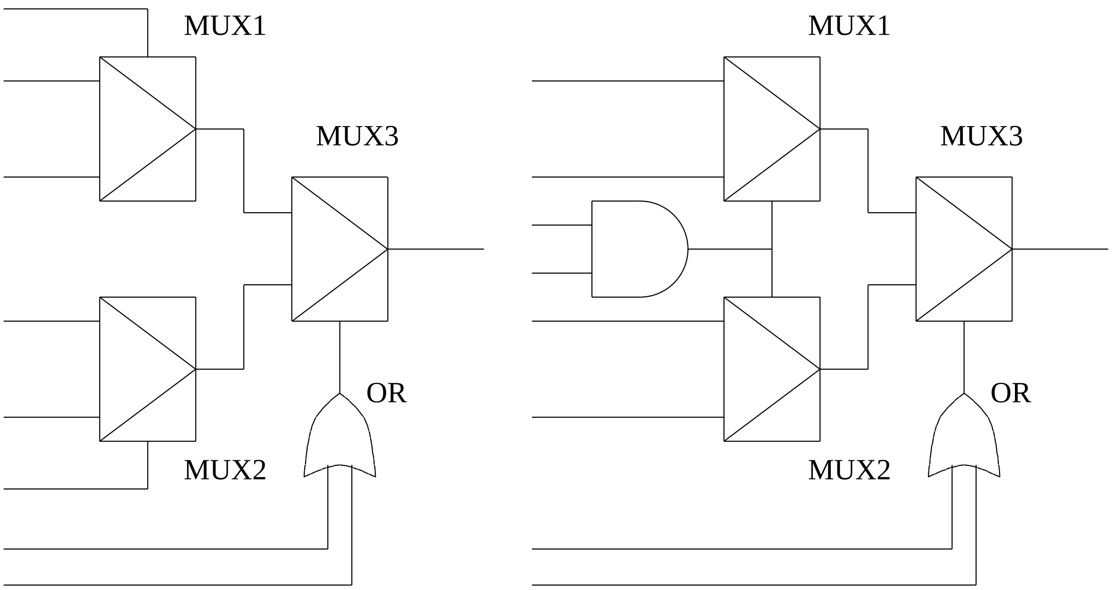
algorithms for their computation from a disjoint representation [33,45].

## 4. SPECTRAL METHODS FOR MULTIPLEXER SYNTHESIS

One approach to the synthesis of logic functions is to take advantage of the powerful multiplexer gate. It has been shown [3,56,57] that multiplexers are universal logic modules, where a multiplexer of $k$ data-select inputs can realize any function of $k+1$ input variables, under the assumption, that the complements of the input variables are also available. Thus, they can be used for the synthesis of multilevel logic networks.

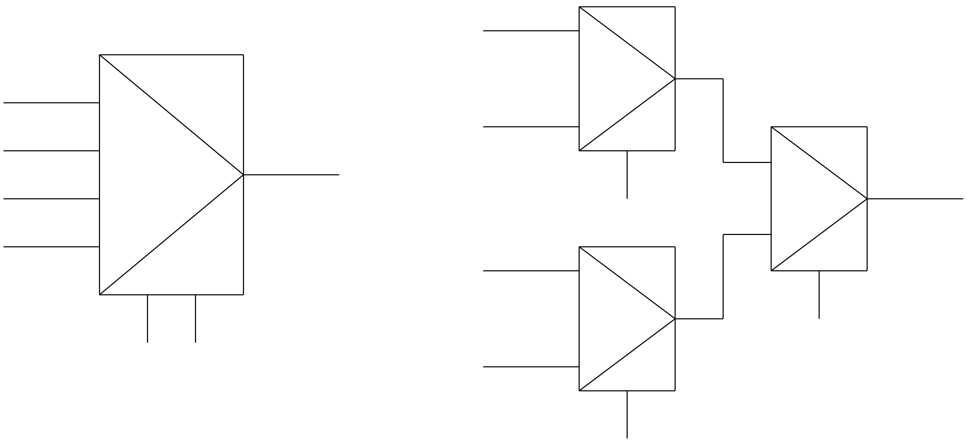
Many synthesis algorithms for different kinds of multiplexer circuits have been developed [58-69]. One synthesis method with multiplexers is to find a single multiplexer, if possible of the minimum size, which realizes the given Boolean function [62-64,68]. Algorithms to find a cascade multiplexer circuit of a Boolean function, if realizable, have been presented in [58,66,69]. The algorithms [63-66] are based on graphical methods which allow only the synthesis for functions with up to six variables. A third realization, which will be further developed in this Chapter, is known as the multiplexer tree circuit [4,59-61,64,65,70,71]. All the previous algorithms operate on minterms, while the method presented here is based on disjoint cubes [45].

One of the motivations to investigate the synthesis for multiplexer tree circuits comes from the introduction of new FPGAs like the $ACT^{TM}$ FPGA family from Actel [14], and Texas Instruments [72] where the basic building block consists of multiplexers, Figure 9. Each basic building block of the $ACT^{TM}$ family allows the implementation of an M(2) multiplexer (Figure 10.a), and in the case of the $ACT$ 1 family also of three hierarchical M(1) multiplexers (Figure 10.b). The $ACT$ 2 family allows only a restricted realization of three hierarchical M(1) multiplexers as can be observed from Figure 9.b. Another motivation for the interest in multiplexer circuits is that a function realized by multiplexers should have less modules than one constructed with conventional logic gates (NAND, NOR) [69]. We will compare the results obtained by technology mappers that support the explicit library of realizable functions of the Actel FPGAs [73,74] and the results of special mapping algorithms which take advantage of the structure of the Actel macrocells [73,75,76] to the results of the mapped multiplexer circuit obtained by the synthesis algorithm presented here.




a. ACT 1 family b. ACT 2 family

Figure 9: Basic building blocks of the $ACT^{TM}$ families.



a. M(2) multiplexer b. three M(1) multiplexers

Figure 10: Multiplexers realizable with the $ACT^{TM}$ family.

In this section we further develop the methods to find redundant multiplexer modules in a tree circuit [58,60,68-71] for the level-by-level top-down (starting from the output) minimization of multiplexer tree circuits [60,70]. The two methods for single-output completely specified Boolean functions, one based on Ratio Parameters for M(1) multiplexer synthesis [58,68,69] and the second based on the

Rademacher-Walsh spectrum for M(2) multiplexer synthesis [4,70] are here uniformly generalized to the M(k) multiplexers synthesis for multi-output incompletely specified Boolean functions. Because computing the complete Walsh spectrum [70] to determine the redundant next level multiplexer modules is complex we introduce the concept of a *local* transform for the data-select variables. This notion is similar to the Ratio Parameter developed for the M(1) synthesis in [58].

### 4.1. General Multiplexer Synthesis

In order to be able to give the general formula for the M(k) multiplexer we first have to introduce the concept of polarity.

<u>*Definition 10:*</u> The *polarity* of a product term $\dot{x}_1\dot{x}_2 \ldots \dot{x}_i \ldots \dot{x}_n$ is defined by the binary string of values being 0 for $\overline{x_i}$ and 1 for $x_i$ respectively.

The general case of a multiplexer M(k), where $k$ is the number of data-select inputs, can be defined as follows.

<u>*Definition 11:*</u> The output function f($x_0$, $x_1$, ..., $x_{n-1}$) = f(X) of a multiplexer M($k$) with $k$ data-select inputs $d_j(X)$ is given by

$$f(X) = \bigcup_{i=0}^{2^k-1} \dot{d}_1(X) \cap \ldots \cap \dot{d}_j(X) \cap \ldots \cap \dot{d}_k(X) \cap f(X) \tag{24}$$

where $\dot{d}_j(X) \in \{d_j(X), \overline{d_j(X)}\}$, being any Boolean function, with the polarity determined by the binary representation of $i$.

For multiplexer synthesis algorithms the data-select functions $d_j(X)$ are commonly restricted to input variables [60,70]. Thus, the form of Equation (25) is obtained

$$f(X) = \bigcup_{i=0}^{2^k-1} \left[ f(X) \right]_{\dot{x}_1 \ldots \dot{x}_k} \tag{25}$$

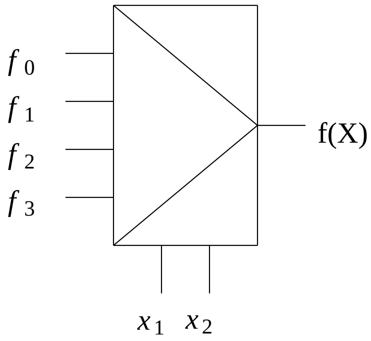
where the cofactor notation follows [18] and the polarity of the data-select variables $\dot{x}_1 \ldots \dot{x}_k$ is determined by the binary representation of $i$. This is the well known Shannon expansion. It follows that the data-input function $f_i(X)$ is given by

$$f_i(X) = \Big[ f(X) \Big]_{\dot{x}_1 \ldots \dot{x}_k} \tag{26}$$

<u>*Example 10:*</u> An $n$ variable Boolean function f(X) with the data-select inputs $x_1$, $x_2$ is decomposed to

$$f(X) = \bar{x}_1 \bar{x}_2 f_0(X) + \bar{x}_1 x_2 f_1(X) + x_1 \bar{x}_2 f_2(X) + x_1 x_2 f_3(X) \tag{27}$$

which is illustrated in Figure 11.




<u>Figure 11:</u> Standard multiplexer M(2) according to Equation (27).

The functions $f_i$ are the data input functions and the variables $x_1$, $x_2$ are the data select variables of the multiplexer. Let us observe that the functions $f_i$ do not depend on $x_1$ and $x_2$.

The general problem in multiplexer synthesis is to find a circuit realization with the minimum number of multiplexers for a given multi-output incompletely specified Boolean function. According to Equation (24) such a circuit can have multiplexers with various numbers of data-select inputs where each data-select input can be any function $d_j(X)$ [56,71]. Because finding the optimal data-select functions being any possible Boolean function is very complex, the multiplexer synthesis algorithms find (quasi)-optimal realizations according to Equation (25), where the data-select variables are input variables [59-62,64-69].

Such a minimal multiplexer circuit can have different permutations of data-select variables in any branch of the circuit [61,71]. To decrease the complexity of the minimization problem, most multiplexer synthesis algorithms assume the same data-select variables in any level of the tree circuit [59-61,70,71], as shown in Figure 12. Such an approach restricts the connectivity to multiplexers in adjacent levels. Thus, the routing of such a circuit mapped to FPGAs with very limited routing resources like the CLi6000 series from Concurrent Logic [77] can be guaranteed. Such a restriction is also advantageous for possible mapping to the $ACT2$ family, Figure 9.b, where the data-select input variables of MUX1 and MUX2 are given by the AND of two variables.

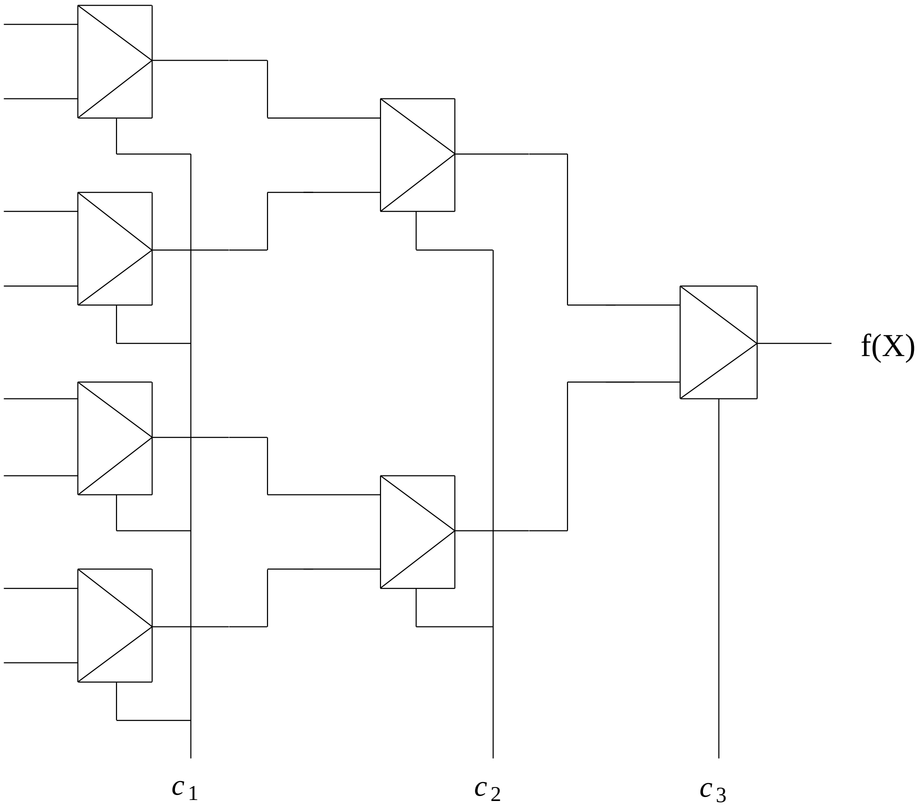



Figure 12: Restricted Multiplexer tree circuit.

As was stated in [71] the minimal upper bound for the levels in such a restricted multiplexer tree circuit is given by

$$L = \frac{n - 1}{k} \tag{28}$$

and the minimal upper bound for the number of multiplexer modules M(k) by

$$M = \sum_{i=0}^{L-1} 2^{ik} \tag{29}$$

One can observe, that the lower bound of levels is also the number of modules necessary for the realization of a linear function. Still an exhaustive search has to be performed to find the optimal permutation of the data-select variables [59,65,71]. Level-by-level minimization algorithms [60,70] decrease the necessary computation and storage requirements for the implicitly exhaustive algorithms to find the optimum tree circuit. It was conjectured [60] that the level-by-level minimization leads to a circuit realization which still has minimal number or close to the minimal number of multiplexer compared to the minimal circuit allowing for an arbitrary data-select variable for each multiplexer. It should be observed that the special case of M(1) multiplexer synthesis is equivalent to finding the minimal SOBDD representation of the Boolean function with negated edges [18,78,79]. Thus, the multiplexer synthesis algorithm for M(1) multiplexers can be applied as a heuristic to find a good variable ordering for the OBDD.

The next section investigates the conditions for which the next level multiplexer modules are redundant.

## 4.2. Redundancy of Multiplexer Modules

The basic principle of the level-by-level minimization algorithm from [60,70] is to find the minimal number of next level modules for a given level. This approach will be adopted here. A similar principle is used for the realization of Boolean functions as cascade multiplexer circuits or single multiplexer circuits, where only one next level module is allowed or no module at all [58,62-64,66,68,69].

There exist three basic conditions for which a next level module is redundant

Condition 1: a data-input function to a multiplexer is trivial

$$f_i = 0$$
$$f_i = 1$$
$$f_i = x_j$$
$$f_i = \overline{x_j}$$

Condition 2: a data-input function to a multiplexer is identical to a data-input function of another multiplexer in the same level of the tree circuit

$$f_i = f_j \qquad\qquad i \neq j$$

Condition 3: a data-input function is the complement of another data-input function to a multiplexer in the same level of the tree circuit

$$f_i = \overline{f_j} \qquad\qquad i \neq j$$

In most algorithms only the first condition is taken into consideration to decrease the number of next level modules [59-61,64,68,71]. The case of a data-input function being the complement of another data-input function has not been taken into consideration in any synthesis algorithm. The advantage of the presented method is that Condition 3 is verified as well. Even if no inverters are available (like in the Actel FPGAs), only one multiplexer is necessary to realize the complemented function instead of a complete subtree. The complemented function can be realized by a control function circuit [71] as shown in Figure 13.

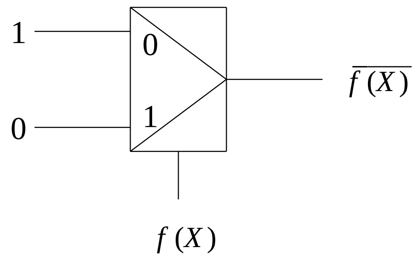



Figure 13. Control Function Circuit for function complementation.

In the case that in a particular level of the multiplexer circuit different combinations of $k$ data-select variables require the same number of next level multiplexer modules, a selection should take into account possible further minimizations in the next lower levels of the multiplexer circuit. Spectral methods are ideally suited for this case, because they give insights into the global structure of the underlying Boolean function [4,58,70]. Therefore, the multiplexer synthesis can be based on the information obtained from the correlation between the Boolean function and the *standard trivial functions* which are determined by the data-select variables.

*Definition 12:* The *positive standard trivial function* $p_i$ represented by the $k$ data-select variables for an M(k) multiplexer is given by

$$p_i = \dot{x}_1 \ldots \dot{x}_j \ldots \dot{x}_k \tag{30}$$

where $i$ is the binary representation of the polarity of the $k$ data-select variables. It follows from Equation (26)

$$f_i(X) = p_i \cap f(X) \tag{31}$$

The number of minterms covered by the *positive standard trivial function* $p_i$ is given by

$$pt = 2^{n-k} \tag{32}$$

Because in this section we only deal with *positive standard trivial functions* we denote them just by *trivial functions*.

The concept of Ratio Parameters [58,68,69] has been developed to determine if data-input functions to an M(1) multiplexer are trivial. The Ratio Parameter method is essentially a spectral method like the method for M(2) multiplexer synthesis [4,70]. Both methods are based on determining the number of minterms covered by the *trivial functions* $p_i$ given by the data-select variables, as shown in the Example in Figure 14.

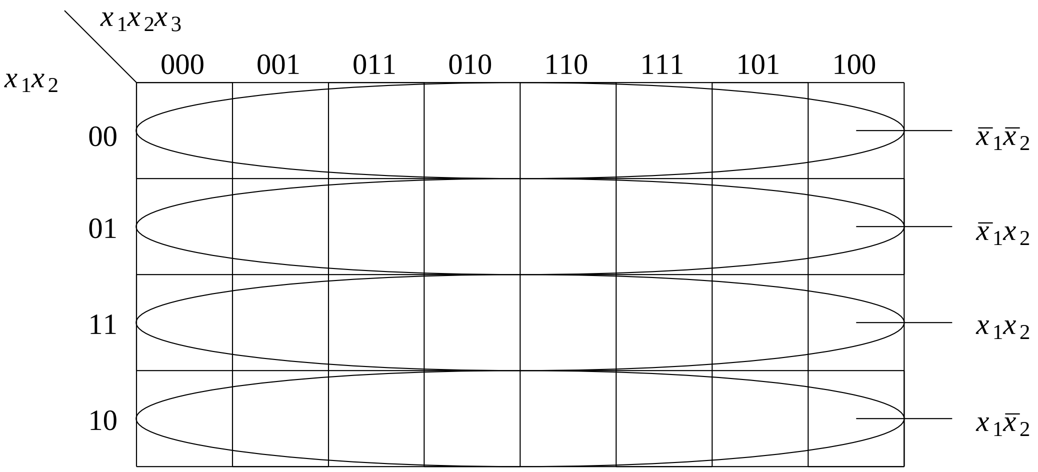



Figure 14. Trivial functions for chosen data-select variables $x_1$ and $x_2$.

The *trivial functions* given in Figure 14 can be taken to define the row vectors of a $SF$ transform for an M(2) multiplexer spectrum.

The spectral method and the Ratio Parameter method can be uniformly generalized for the synthesis with M(k) multiplexers by the introduction of a *local* transform. To be able to incorporate incompletely specified Boolean functions we apply the $T$ coding introduced in the previous section.

For M(2) multiplexer synthesis [70] the Rademacher-Walsh spectrum was adopted to find the correlation of the Boolean function to the *trivial functions*. A further summation [4,70] of subsets of spectral coefficients of the Rademacher-Walsh spectrum is necessary to obtain the final spectrum describing the correlation to the *trivial functions* $p_i$. This summation is realized by a $4 \times 4$ Rademacher-Walsh transforms on subsets of the initial Rademacher-Walsh spectrum. In the general case of an M(k) multiplexer the summation can be realized by a $2^k \times 2^k$ Rademacher-Walsh transform on the respective subset of the initial Rademacher-Walsh spectrum for each possible set of $k$ data-select variables.

The *local SF* transform for multiplexers performs in one step both the initial calculation of the complete Rademacher-Walsh spectrum of the function and the following summation of a subset of coefficients. Thus, one avoids the complexity of computing the whole Rademacher-Walsh spectrum for a given function. An M(k) multiplexer for an n-variable Boolean function can have $\binom{n}{k}$ different possible combinations of $k$ data select variables. Table I compares the number of coefficients that have to be calculated for an M(2) multiplexer synthesis by using the Rademacher-Walsh transform and the *local SF* transform.

TABLE I

COMPARISON OF THE NUMBER OF COEFFICIENTS

| variables | Rademacher-Walsh coefficients | spectral coefficients for $\binom{n}{2}$ pairs of data-select variables |
|---|---|---|
| $n$ | $2^n$ | $\binom{n}{2} \times 4$ |
| 5 | 32 | $10 \times 4$ |
| 10 | 1024 | $45 \times 4$ |
| 20 | 1048576 | $190 \times 4$ |
| 30 | $10^9$ | $435 \times 4$ |

The first column of Table I gives the number of input variables of a Boolean function. The second column gives the number of spectral coefficients of the initial Rademacher-Walsh spectrum. Finally, the last column gives the number of spectral coefficients for all possible pairs of data-select variables, where a spectrum for a data-select pair consists of four coefficients. The formulae for the calculation of the number of coefficients are given in the second row.

As one can observe from Table I, the computation of spectra for functions with more than 20 variables is not feasible with general Fast Transforms. The Rademacher-Walsh spectrum can be calculated directly from a disjoint representation [26,45,47], but still the complexity of calculating all $2^n$ coefficients can be prohibitive. For the decomposition to multiplexer modules given by Equation (25) we are only interested in the correlation between the *trivial functions* and the data-input functions to the multiplexer. Thus, the direct calculation of only the necessary spectral coefficients for the combinations of $k$ data-select variables by the *local SF* transform is much more efficient.

The following property gives the relation of the spectral coefficients $t_i$ obtained by the *local SF* transform for completely specified Boolean functions to the spectral coefficients $s_i$ used in [70].

Property 5: The spectral coefficients $s_i$ of the spectrum $S_{x_1 \cdots x_k}$ for $k$ data-select variables $x_1 \cdots x_k$ are computed as follows

$$s_i = (pt - 2 \times a_i) \times 4 \tag{33}$$

where $pt$ is the number of minterms covered by the *trivial function* $p_i$, and $t_i = \langle a_i \rangle$.

For incompletely specified Boolean functions the value of $t_i = \langle a_i \rangle$ or $t_i = \langle pt - b_i \rangle$ is chosen whichever leads to a heigher absolute value of $s_i$.

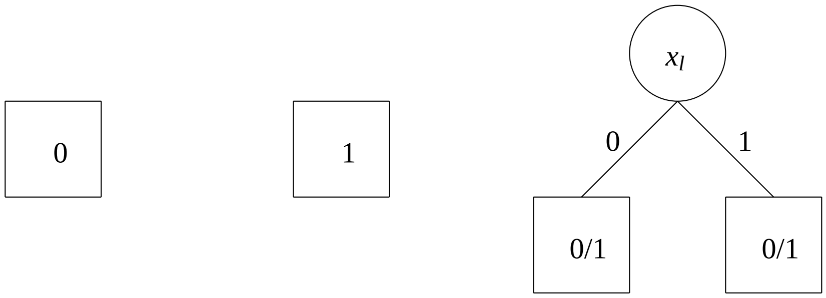
A criterion has been introduced in [70] for the selection of one combination of $k$ data-select variables for the case that more than one combination leads to the same minimal number of next level modules. The criterion is based on the information given by the spectral coefficients. It was stated in [70] that a high value of the sum of absolute values of the spectral coefficients,

$$sum = \sum_{i=0}^{2^n-1} |s_i| = |S_{\dot{x}_1 \ldots \dot{x}_k}| \tag{34}$$

gives a higher possibility for further minimization in the following levels. This is based on the assumption, that a larger number of either false minterms or true minterms in the data-input functions allows the further minimization in the next levels. Therefore, we can formulate an additional Condition 4 for multilevel realizations

Condition 4: for different combinations of $k$ data-select variables with the same minimal number of next level modules the one with the highest value of *sum* is chosen. If several combinations have the same highest value of *sum*, one of them is selected randomly.

With the above Definitions and Properties we are able to formulate the conditions for redundancy of next level modules with spectral conditions and Boolean conditions. For the verification of some of the conditions for the redundant next level modules, the OBDD representation of a Boolean function is very useful.




a. trivial function 0 b. trivial function 1 c. trivial function $x_l$

Figure 15. Trivial functions in OBDD form.

Figure 15 shows that it is easy to verify directly from the OBDD representation that a completely specified Boolean function is trivial. If the OBDD consists of only terminal nodes or a single node, then the function is trivial. However, the identification of *trivial functions* for incompletely specified Boolean functions is more complicated.

Verification of Condition 1: The trivial incompletely specified data-input functions $f_i$ have the following properties in the spectral domain, for $f_i = 0$:

$$a_i = 0 \tag{35}$$

and for $f_i = 1$:

$$b_i = 0 \tag{36}$$

Equation (35) can be verified in Boolean domain for the computation with a cube representation by

$$f_i = f \cap p_i = f_{dc} \tag{37}$$

where $f_{dc}$ consists only of not specified terms. Equation (36) can be similarly verified by

$$f_i = f_{\rightarrow 1} \cap p_i = p_i \tag{38}$$

where the notation $\rightarrow 1$ denotes the specification of the not specified part of the incompletely specified Boolean function $f$ to true terms. In the cube representation only the respective symbols in the output part of the cube have to be changed from don't cares to ones to obtain this transformation. For OBDDs this requires a complete restructuring to obtain an OBDD for a completely specified Boolean functions.

A necessary condition for a completely specified function being dependent on only one variable $x_l$ is

$$s_i = 0 \tag{39}$$

which is equivalent to

$$a_i = 2^{n-k-1} \tag{40}$$

For an incompletely specified Boolean function it is necessary that the not specified terms can be specified in such a way that Equation (39) is fulfilled. This can be verified in Boolean domain by concurrently checking of the two conditions

$$f_{\rightarrow 1}(X) \cap x_l = x_l \tag{41}$$

and

$$f(X) \cap \overline{x_l} = f_{dc} \tag{42}$$

where $f_{dc}$ consists only of not specified terms.

Verification of Condition 2: The assignment of the not specified minterms to true or false ones to obtain identical data-input functions ($f_i = f_j$), if possible, can be easily verified in spectral domain for completely specified Boolean functions by:

$$a_i = a_j \tag{43}$$

However, if Equation (43) is true, still the identity of the two functions has to be checked. With OBDDs in Boolean domain the tautology can be easily verified by a simple pointer comparison. For incompletely specified Boolean functions it has to be computed whether the not specified parts of the functions $f_i$ and $f_j$ can be specified in such a way that $f_i = f_j$. This can be verified by the complement of the intersection $\overline{f}_{in}$ of the two incompletely specified functions

$$f_{in} = f_i \cap f_j \tag{44}$$

having no common minterms with the completely specified part of function $f_i$

$$f_i \cap \overline{f_{in}} = f_i \# f_{in} = f_{dc_1} \tag{45}$$

and no common minterms with the completely specified part of function $f_j$

$$f_j \cap \overline{f_{in}} = f_j \# f_{in} = f_{dc_2} \quad (46)$$

where $\#$ denotes the sharp operation [80] and $f_{dc_1}$ and $f_{dc_2}$ consist only of not specified terms. If the Equations (45) and (46) are true, the not specified part of $f_i$ and $f_j$ can be specified in such a way that $f_i = f_j$.

Verification of Condition 3: For a data-input function $f_i$ being the complement of a completely specified data-input function $f_j$ the necessary condition is as follows

$$b_i + b_j = a_i + a_j = 2^{n-k-1} \quad (47)$$

To find a possible specification of the not specified part of the incompletely specified Boolean function such that two data-input functions are complemented, Equations (48) and (49) have to be true

$$F_i \cap f_j = f_{dc} \quad (48)$$

to verify that the specified parts of both functions have no common minterms. The equivalent of Equation (47) in Boolean domain is given by

$$f_{i \to 1} + f_{j \to 1} = p_i \quad (49)$$

If both formulae are fulfilled, the not specified part of the data-input functions $f_i$ and $f_j$ can be chosen in such a way that $f_i = \overline{f_j}$.

The above conditions have to be computed for all possible combinations of data-select variables. In case when there is more than one set of data-select variables that have a minimum number of next level modules, the one according to Condition 4 is chosen.

The verification of the four conditions allows the determination of the data-select select variables for a single level of multiplexer modules. The extension of the method to multi-output incompletely specified Boolean functions is trivial. In such a case the primary output level of the function (which consists of more than one multiplexer) is treated like any other level of the multiplexer tree circuit.

The overall structure of the multi-level synthesis algorithm is given by Algorithm 6.

The developed methods are illustrated in the next section with a complete example.

Algorithm 6: Multiplexer Synthesis algorithm

```
//   k                : number of data-select variables
//  output_functions : number of output functions for given Boolean function
//  input_variables  : number of input variables for given Boolean function

mux_synthesis(k)
{       min_level_modules = output_functions * 2^k ;
        level = 0;      // determines current level in circuit
        while ( min_level_modules != 0 ){
                // computation of minimal number of next level modules for given level
                for ( i = 0 ; i < ⌈(input_variables − level * k) / k⌉ ; i++ ) {
                        // all permutations of data-select variables
                        data_select_variables = generate_data_select(k,i);
                        // compute number of next level modules for given data_select_variables
                        level_modules = compute_modules(data_select_variables);
                        sum = Σ_{n=0}^{level_modules} sum_n ;  // sum for all multiplexers
                        if ( min_level_modules > level_modules ){
                                // current data_select_variables lead to less next next level modules
                                max_sum = sum;
                                min_level_modules = level_modules;
                                best_select = data_select_variables;
                        }else if ( min_level_modules == level_modules ){
                                if ( sum > max_sum ){
                                        max_sum = sum;
                                        best_select = data_select_variables;
                        }
                }
        }
}
```

### 4.3. Synthesis Example

The synthesis steps from the previous section will be illustrated on the example from [70] :

$$f(X) = \bar{x}_1 x_4 \bar{x}_5 + x_1 x_2 x_4 + \bar{x}_1 x_2 x_3 \bar{x}_4 + \bar{x}_3 \bar{x}_4 x_5 + x_1 \bar{x}_4 x_5$$

For the initial verification of the redundancy of next level modules the *local* SF-spectra for each combination of data-select variables, and the *sum* parameter are calculated. The calculation of the initial spectra $SF_{x_i x_j}$ will be illustrated for the spectrum of the data-select pair $x_1, x_2$. The value of the spectral coefficients of the spectrum $S_{x_i x_j}$ [70] can be obtained by Equation (33). The *trivial functions* $p_i$ for the *local* SF-transform of the chosen data-select pair are: $p_0 = \bar{x}_1 \bar{x}_2$, $p_1 = \bar{x}_1 x_2$, $p_2 = x_1 \bar{x}_2$, $p_3 = x_1 x_2$, where each *trivial function* covers $pt = 2^{5-2} = 8$ minterms, Equation (32).

| $x_1x_2 \backslash x_3x_4x_5$ | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
|---|---|---|---|---|---|---|---|---|
| 00 | | 1 | | 1 | 1 | | | |
| 01 | | 1 | | 1 | 1 | | 1 | 1 |
| 11 | | 1 | 1 | 1 | 1 | 1 | 1 | |
| 10 | | 1 | | | | | 1 | |

Figure 16. Karnaugh-map of synthesis example.

First the intersection of the function f(X) with each *trivial function* has to be calculated to obtain the *local* SF-spectrum for the chosen data-select pair, e.g.

$$f_0 = f(X) \cap p_0 = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1\bar{x}_2x_4\bar{x}_5$$

However, the data-input functions are now independent from the data-select variables $x_1$ and $x_2$. Thus, they can be removed:

$$f_0 = \bar{x}_3\bar{x}_4x_5 + x_4\bar{x}_5$$

Because the function f(X) is completely specified, the spectral coefficients of the *local* SF-spectrum are given by $t_i = \langle a_i \rangle$. They can be directly computed from the disjoint cube representation of the data-input functions $f_i$: $a_0 = 3$, $a_1 = 5$, $a_2 = 2$, and $a_3 = 6$. No coefficient $t_i = \langle a_i \rangle$ fulfills Equations (35), (36) or (40) which verify the possibility of trivial data-input functions. Similarly, the criteria for the identity of two data-input functions, Equations (43), is not satisfied. However, the necessary condition for complemented data-input functions is true for $a_2 + a_3 = 8$. Therefore, this case has to be verified further. Applying Equation (48), we obtain $f_2 \cap f_3 = x_4 \neq \emptyset$. Because the intersection is not empty, $f_2$ can not be the complement of $f_3$. No data-input of the multiplexer is redundant. Therefore, the number of next level modules *Mod* for the data-select variables $x_1$ and $x_2$ is $Mod = 4$. To be able to check later the Condition 4, the $sum = 48$ is computed by applying Equation (33) and (34).

The spectra for the other data-select pairs are calculated the same way. The results are listed in Table II. The exact minimum number of next level modules *Mod* can be directly calculated because all possible cases of redundant next level modules can be verified by Conditions (1)-(3), while the method by Lloyd [70] determines only the upper and the lower bound, *maxMod* and *minMod*.

TABLE II

FIRST LEVEL SPECTRA

| row | spectrum | $t_0$ | $t_1$ | $t_2$ | $t_3$ | sum | Mod |
|---|---|---|---|---|---|---|---|
| 1 | $SF_{x_1x_2}$ | 3 | 5 | 2 | 6 | 48 | 4 |
| 2 | $SF_{x_1x_3}$ | 4 | 4 | 4 | 4 | 0 | 4 |
| 3 | $SF_{x_1x_4}$ | 4 | 4* | 4* | 4* | 0 | 1 |
| 4 | $SF_{x_1x_5}$ | 5 | 3 | 2 | 6 | 48 | 4 |
| 5 | $SF_{x_2x_3}$ | 3 | 2 | 5 | 6 | 48 | 4 |
| 6 | $SF_{x_2x_4}$ | 3 | 2 | 5 | 6 | 48 | 4 |
| 7 | $SF_{x_2x_5}$ | 2 | 3 | 5 | 6 | 48 | 4 |
| 8 | $SF_{x_3x_4}$ | 4* | 4 | 4 | 4 | 0 | 3 |
| 9 | $SF_{x_3x_5}$ | 3 | 5 | 4 | 4 | 16 | 4 |
| 10 | $SF_{x_4x_5}$ | 1 | 7 | 6 | 2 | 80 | 4 |

In Table II the symbol * indicates that the respective data-input function is either identical or complemented to another one, or a trivial one. Because $SF_{x_1x_4}$ has the lowest number of next level modules, all other data-select pairs can be discarded. Thus, the best pair of data-select variables has been found in the first step, while for the pure spectral method [70] three further verification steps are necessary.

The non-trivial data-input functions to the multiplexer module have to be decomposed further for the calculation of the next level of the hierarchical multiplexer circuit, The four data-input functions have been already obtained by the intersection of the *trivial functions* $p_i$ for the data-select pair $x_1$, $x_2$ with the initial function for the calculation of the spectral coefficients: $f_0 = \bar{x}_3 x_5 + x_2 x_3$, $f_1 = \bar{x}_5$, $f_2 = x_5$, and $f_3 = x_2$. The only non-trivial function is $f_0$. Therefore, $f_0$ has to be decomposed further. Because the function $f_0$ depends on only three variables it can be realized by one M(2) multiplexer and the choice of the data-select variables does not matter [69].

However, to illustrate the general synthesis procedure, the three spectra $SF_{x_ix_j}$ for $f_0$ are calculated

(Table III).

TABLE III

SECOND LEVEL SPECTRA FOR $f_0$

| row | spectrum | $t_1$ | $t_2$ | $t_3$ | $t_4$ | sum | Mod |
|---|---|---|---|---|---|---|---|
| 1 | $SF_{x_2 x_3}$ | 4* | 5* | 4* | 3* | 16 | 0 |
| 2 | $SF_{x_2 x_5}$ | 5* | 4* | 4* | 3* | 16 | 0 |
| 3 | $SF_{x_3 x_5}$ | 5* | 3* | 4* | 4* | 16 | 0 |

The property, that no further multiplexer is necessary for any data-select pair is verified by the result given in Table II (Mod = 0 for each data-select pair). Therefore, each selection is equally good. Therefore, randomly the data-select pair $x_2$, $x_3$ is chosen for the realization. Thus, the final hierarchical multiplexer circuit has the form shown in Figure 17.

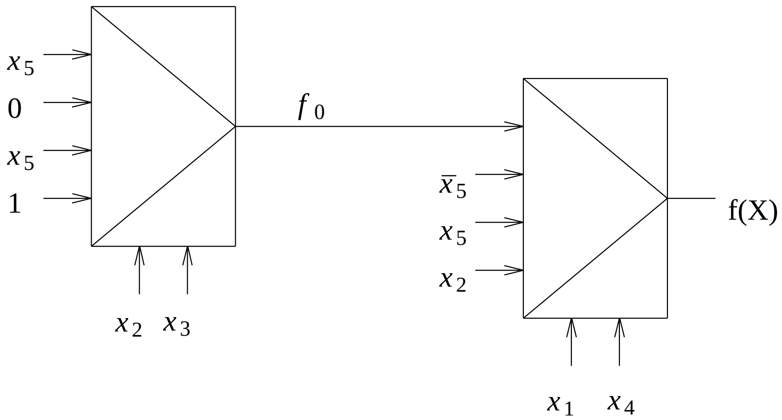



Figure 17. Final multiplexer realization.

### 4.4. Mapping of a Multiplexer Tree Circuit to the ACT1 Macrocells

While the hierarchical multiplexer circuits obtained for M(1) multiplexer can be mapped directly to FPGAs like the CLi6000 of Concurrent Logic [77] a special mapping algorithm is necessary for the $ACT^{TM}$ family of Actel. The macrocells provided by the $ACT^{TM}$ FPGA family from Actel, Figure 9, have a restricted connectivity of M(1) multiplexers. An M(2) multiplexer can be realized with one macrocell both of the $ACT1$ and $ACT2$ family. Therefore, a multiplexer circuit based on M(2) multiplexers can be directly mapped to the Actel FPGAs. However, because of the lack of inverters, for each complemented data-input function an additional macrocell is necessary.

For our implementation we chose the $ACT\,1$ macrocell. The mapping of an M(1) multiplexer circuit to the $ACT\,1$ macrocell is restricted by the internal connectivity of the macrocells. The output functions of the two multiplexers at the input of the macrocell, MUX1 and MUX2, are not available as outputs of the macrocell. Therefore, multiplexers with fan-out greater than 1 have to be realized with the output level multiplexer MUX3 of a macrocell, or the respective multiplexer has to be duplicated. As a heuristic we first map a multiplexer with fan-out greater than 1 to the MUX3 multiplexer of a macrocell. Additionally, input multiplexers to this multiplexer with fan-out equal to 1 are mapped to the same macrocell.

As a second step the multiplexers that have a function $g$ and its complement $\bar{g}$ as inputs are taken into consideration. Because of the restricted internal connectivity of a macrocell, the complementation of the data-input function realized according to Figure 9, has only one minimal mapping shown in Figure 18.

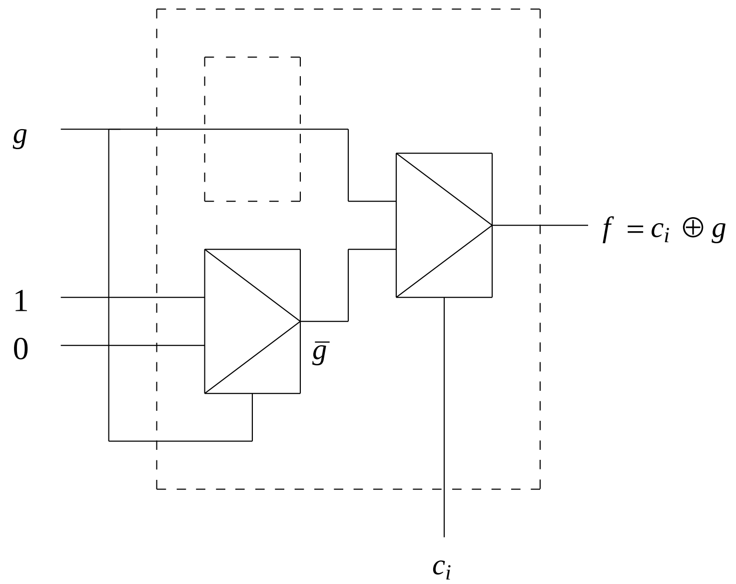



Figure 18. Mapping of complemented data-input functions.

Finally the remaining multiplexers are mapped level-by-level to the macrocells.

The next section gives the results obtained by the implementation of our multiplexer synthesis algorithm followed by the mapping to the *ACT* 1 macrocell.

## 4.5. Benchmark Results

As stated in the previous section, the mapping of M(2) multiplexer circuits to the $ACT^{TM}$ FPGAs from Actel can be done directly. However, an M(1) multiplexer circuit has to be matched to the macrocells.

Table IV lists in column *mux−map* the results of our multiplexer synthesis algorithm for several MCNC benchmarks. For comparison we give the results of *misII* [73,81] with the Actel library of realizable functions as representative for general technology mappers, and the result of Actel specific mappers *mis−pga* [73,81] and *Prosperine* [76,82].

TABLE IV

BENCHMARK RESULTS FOR MULTIPLEXER SYNTHESIS

| | k = 2 | | k = 1 | | mux-map | misII | mis-pga | Amap | Prosperine | |
|---|---|---|---|---|---|---|---|---|---|---|
| function | mod | depth | mod | depth | cells (ti.) | cells (ti.) | cells (ti.) | cells (ti.) | act0 | act1 |
| bw | 61 | 2 | 106 | 4 | 68 (5.1) | 81 (7.1) | 60 (11.7) | 83 (3.6) | 87 | 63 |
| clip | 48 | 4 | 91 | 8 | 49 (21.5) | 57 (4.9) | 48 (25.1) | 60 (2.5) | 86 | 68 |
| con1 | 8 | 3 | 24 | 6 | 9 (0.2) | | | | | |
| f51m | 27 | 4 | 55 | 7 | 27 (4.2) | 52 (5.2) | 44 (14.4) | 56 (2.2) | 83 | 59 |
| inc | 35 | 3 | 94 | 6 | 60 (2.3) | | | | | |
| misex1 | 17 | 4 | 33 | 5 | 21 (0.5) | 22 (1.9) | 18 (6.0) | 25 (1.1) | 30 | 24 |
| misex2 | 90 | 12 | 171 | 22 | 65 (0.6) | 46 (4.3) | 40 (3.5) | 47 (1.5) | 52 | 42 |
| rd53 | 7 | 2 | 15 | 4 | 11 (0.2) | | | | | |
| rd73 | 14 | 3 | 29 | 6 | 23 (5.4) | 30 (12.1) | 32 (2.7) | 32 (1.6) | | |
| rd84 | 21 | 4 | 40 | 7 | 32 (24.2) | 62 (6.5) | 50 (30.7) | 62 (2.6) | 87 | 65 |
| sao2 | 41 | 5 | 83 | 9 | 51 (3.1) | 52 (8.0) | 51 (24.4) | 56 (2.0) | | |
| squar5 | 19 | 2 | 34 | 4 | 19 (0.3) | | | | | |
| xor5 | 2 | 2 | 4 | 4 | 4 (0.0) | | | | | |
| 5xp1 | 28 | 3 | 54 | 6 | 31 (2.7) | 51 (4.4) | 40 (10.4) | 42 (1.7) | 65 | 50 |
| 9sym | 11 | 4 | 23 | 8 | 19 (3.6) | 99 (14.1) | 26 (55.8) | 106 (4.1) | | |
| sum | 292 | | 550 | | 293 (58.8) | 371 (34.3) | 300 (101.8) | 375 (15.2) | 490 | 371 |

The results of the multiplexer synthesis given in the columns k=1 and k=2 assume that there are inverters available at each multiplexer input to realize complemented data input functions. The columns *mod* give the total number of necessary M(1) or M(2) modules, and the columns *depth* gives the longest path in the circuit. The columns *cells* give the numbers of necessary macrocells. The computation times (*ti.*) are given in seconds. The times given for $mux-map$ include the multiplexer synthesis and the mapping step. The bottom row $\sum$ lists the summation of the numbers of modules and the computation times for which the results af all programs were available. The results were obtained on a Sparc 4/370 (12.5 mips). For $mis-pga$ (new) and Amap the time was obtained on a DEC 5500 (28 mips).

It should be stressed, that $mux-map$ performs the synthesis and mapping for the $act0$ type combination of M(1) multiplexers (Figure 10.b). Additionally, the data-select inputs of the multiplexers are restricted to input variables. Nevertheless, better results have been obtained than by the previous published algorithms which take advantage of the $act2$ macrocell possibilities.

In this section the concept of the *local* SF-transform has been applied to the synthesis of multiplexer circuits for incompletely specified multi-output Boolean functions. The results show that that the multiplexer synthesis with following mapping to the $ACT1$ family needs usually less macrocells and is generally faster than the conventional multi-level minimization followed by mapping [73-76,81-83]. The obtained results can be even further improved by using a more sophisticated mapping algorithm. The obtained multiplexer circuit can be easily converted to a mixed EXOR / multiplexer circuit, where multiplexers as in Figure 18 are replaced by EXORs. Another possible extension is to expand the mapping algorithm to the $ACT2$ family. Finally, to obtain a significantly improved execution time and smaller memory requirement the algorithm can be implemented using OBDDs.

## 5. CONCLUSION

This chapter extended the known discrete transforms applied in logic synthesis to any transform describing a set of switching functions. This approach allows for the computation of the correlation of a given $mv$ function with any set of $mv$ functions. Such sets of functions are for example the realizable functions of the macrocells in available FPGA architectures. Based on such spectra, synthesis algorithms can be developed that operate in the functional domain rather than in the algebraic one [18].

Moreover, the introduction of the $T$ coding allows also for the efficient incorporation of incompletely specified functions into the synthesis process.

**References**

1. M. F. Karpovsky, in *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley, 1976.
2. R. J. Lechner, ‘‘Harmonic Analysis of Switching Functions,’’ in *Recent Developments in Switching Theory*, ed. A. Mukhopadhyay, pp. 121-228, Academic, 1971.
3. S. L. Hurst, in *The Logical Processing of Digital Signals*, Crane Russak, 1978.
4. S. L. Hurst, D. M. Miller, and J. C. Muzio, in *Spectral Techniques in Digital Logic*, Academic Press, 1985.
5. Ph. W. Besslich and T. Lu, in *Diskrete Orthogonaltransformationen*, Springer Verlag, 1990.
6. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, in *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
7. R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, ‘‘MIS: Multi-Level Interactive Logic Optimization System,’’ *IEEE Trans. on CAD*, vol. 6, no. 6, pp. 1062-1082, November 1989.
8. Bo-Gwan Kim and D. L. Dietmeyer, ‘‘Multilevel Logic Synthesis with Extended Arrays,’’ *IEEE Trans. on Comput.*, vol. 11, no. 2, pp. 142-157, February 1992.
9. J. M. Saul, ‘‘An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representations,’’ *Int. Conf. on Comput. Design: VLSI in Comput. and Processors*, pp. 372-375, September 1990.
10. Ph. W. Besslich and M. W. Riege, ‘‘An Efficient Program for Logic Synthesis of Mod-2 Sum Expressions,’’ *Euro ASIC*, pp. 136-141, Paris, 1991.
11. W. G. Schneeweiss, in *Boolean Functions with Engineering Applications and Computer Programs*, Springer-Verlag, 1989.
12. A. K. Jain, in *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.
13. A. V. Oppenheim and R. W. Schafer, in *Discrete-Time Signal Processing*, Prentice Hall, 1990.
14. Actel, in *ACT Family Field Programmable Gate Array Data Book*, March 1991.
15. R. Rudell, ‘‘Multiple-Valued Logic Minimization for PLA Synthesis,’’ *Master of Science Thesis, University of California*, *Berkeley*, June 1986.
16. T. Sasao, ‘‘Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays,’’ *IEEE Trans. on Comput.*, vol. 30, no. 9, pp. 636-643, September 1981.
17. T. Sasao, ‘‘Multiple-Valued Logic and Optimization of Programmable Logic Arrays,’’ *IEEE Trans. on Comput.*, vol. 21, no. 4, pp. 71-80, April 1988.
18. R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, ‘‘Multilevel Logic Synthesis,’’ *Proc of the IEEE.*, vol. 78, no. 2, pp. 264-300, February 1990.
19. L. Lavagno, S. Malik, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘MIS-MV: Optimization of Multi-Level Logic with Multiple-Valued Inputs,’’ *Proc. IEEE Int. Conf. on CAD*, pp. 560-563, Santa Clara, CA, November 1990.
20. S. L. Hurst, ‘‘The Application of Chow Parameters and Rademacher-Walsh Matrices in the Synthesis of Binary Functions,’’ *Comput. J.*, vol. 16, no. 2, 1973.
21. A. M. Lloyd, ‘‘A Consideration of Orthogonal Matrices, Other Than the Rademacher-Walsh types for the Synthesis of Digital Networks,’’ *Int. J. Electronics*, vol. 47, no. 3, pp. 205-212, 1975.

22. J. C. Muzio, ''Nonorthogonal Transforms for Logical Design,'' *Technical Report CS77006-R, Virginia Polytechnic Institute and State University*, June 1977.

23. B. J. Falkowski, ''Spectral Methods for Boolean and Multiple-Valued Input Logic Functions,'' *Ph.D. dissertation, Portland State University*, May 1991.

24. I. Schäfer, B. J. Falkowski, and M. A. Perkowski, ''A Fast Computer Implementation of Adding and Arithmetic Multi Polarity Transforms for Logic Design,'' *IEEE 34th Midwest Symp. on Circuit & Systems*, Monterey, CA, May 1991.

25. E. Eris and J. C. Muzio, ''Spectral Testing of Circuit Realization Based on Linearizations,'' *IEE Proc. Pt. E*, vol. 133, pp. 73-78, March 1986.

26. D. Varma and E. A. Trachtenberg, ''Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition,'' *IEEE Trans. on CAD*, vol. 8, pp. 901-916, August 1989.

27. M. Davio, J. P. Deschamps, and A. Thayse, in *Discrete and Switching Functions*, McGraw-Hill, 1978.

28. D. H. Green, ''Families of Reed-Muller Canonical Forms,'' *Int. J. of Electronics*, vol. 63, no. 2, pp. 259-280, January 1991.

29. D. H. Green, ''Reed-Muller Canonical Forms With Mixed Polarity and Their Manipulations,'' *Proc. of IEE Pt. E*, vol. 137, no. 1, January 1990.

30. I. Schäfer and M. A. Perkowski, ''Multiple Valued Generalized Reed-Muller Forms,'' *Proc IEEE 21st Int. Symp. on Multiple-Valued Logic*, pp. 40-48, May 1991.

31. I. Schäfer and M. A. Perkowski, ''Multiple-Valued Input Generalized Reed-Muller Forms,'' *accepted for the publication in Proc. of IEE Pt. E*, May 1992.

32. M. A. Perkowski, ''The Generalized Orthonormal Expansions of Functions with Multiple-Valued Inputs and Some of its Applications,'' *Proc. 22nd Int. Symp. on Multiple-Valued Logic*, pp. 442-450, Sendai, Japan, May, 1992.

33. I. Schäfer, ''Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis of Nearly Linear Functions and Their Mapping to FPGAs,'' *Ph.D. Dissertation, Portland State University*, June, 1992.

34. T. Sasao, ''Optimization of Multiple-Valued AND-EXOR Expressions using Multiple-Place Decision Diagrams,'' *Proc. IEEE 22nd Int. Symp. on Multiple-Valued Logic*, Sendai, Japan, May 1992.

35. A. Auer, in *PLD Handbuch: Tabellen und Daten*, Hüthig Buch Verlag, 1990.

36. Xilinx, in *The Programmable Gate Array Data Book*, 1989.

37. I. Schäfer, B. J. Falkowski, and M. A. Perkowski, ''An Efficient Computer Algorithm for the Calculation of Walsh Transform for Completely and Incompletely Specified Multiple-Valued Input Binary Functions,'' *IEEE 34th Midwest Symp. on Circuits & Systems*, Monterey, CA, May 1991.

38. I. Schäfer and M. A. Perkowski, ''On the Minimal Multiple-Valued Input Kronecker Reed-Muller Form for Incompletely Specified Multiple-Valued Input Functions,'' *submitted to IEEE Trans. on Computers*, April 1992.

39. B. J. Falkowski and M. A. Perkowski, ''Essential Relations Between Classical and Spectral Approaches to Analysis, Synthesis, and Testing of Completely and Incompletely Specified Boolean Functions,'' *Proc. of Int. Symp. Circ. & Systems (ISCAS)*, pp. 1656-1659, New Orleans, May 1990.

40. B. J. Falkowski and M. A. Perkowski, ''Algorithms for the Calculation of Hadamard-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions,'' *Proc. of Int. Conf. on Comput. and Comm.*, pp. 868-869, Scottsdale, Arizona, March 1990.

41. 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,'' *Proc. of Int. Symp. Circ. & Systems (ISCAS)*, pp. 2913-2916, New Orleans, May 1990.

42. D. H. Green, in *Modern Logic Design*, Electronic Systems Engineering Series, 1986.

43. B. J. Falkowski and M. A. Perkowski, ''On the Calculation of Generalized Reed-Muller Canonical Expansions from Disjoint Cube Representation of Boolean Functions,'' *IEEE 33rd Midwest Symp. on Circuits & Systems*, pp. 1131-1133, August 1990.

44. D. Varma and E. A. Trachtenberg, ''Computation of Reed-Muller Expansions of Incompletely Specified Boolean Functions From Reduced Representations,'' *IEE Proc. Pt. E.*, vol. 138, no. 2, pp. 85-92, March 1991.

45. B. J. Falkowski, I. Schäfer, and M. A. Perkowski, ''Effective Computer Methods for the Calculation of the Rademacher-Walsh Spectrum for Boolean Functions,'' *to appear in IEEE Trans. on CAD.*, July 1992.

46. B. J. Falkowski and M. A. Perkowski, ''One More Way to Calculate the Hadamard-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions,'' *Int. J. Electronics*, vol. 69, no. 5, pp. 595-602, 1990.

47. S. Purwar and A. K. Susskind, ''Computation of Walsh Spectrum from Binary Decision Diagram and Binary Decision Diagram from Walsh Spectrum,'' *Computers & Elect. Engng.*, vol. 15, no. 2, pp. 59-65, 1989.

48. S. Purwar, ''An Efficient Method of Computing Generalized Reed-Muller Expansions from Binary Decision Diagram,'' *IEEE Trans. on Comput.*, vol. 40, no. 11, pp. 1298-1301, November 1991.

49. I. Schäfer, B. J. Falkowski, and M. A. Perkowski, ''Generation of Adding and Arithmetic Multi-Polarity Transforms for Incompletely Specified Boolean Functions,'' *accepted for the publication in the Int. J. of Electronics*, February 1992.

50. B. J. Falkowski and M. A. Perkowski, ''One more Way to Calculate Generalized Reed-Muller Expansions of Boolean Functions,'' *Int. J. Electronics*, vol. 71, no. 3, pp. 383-396, 1991.

51. R. E. Bryant, ''Graph-Based Algorithms for Boolean Function Manipulation,'' *IEEE Trans. on Comput.*, vol. 35, no. 8, pp. 667-691, August 1986.

52. A. Srinivasan, T. Kam, S. Malik, and R.K. Brayton, ''Algorithms for Discrete Function Manipulation,'' *IEEE Proc. of Int. Conf. on CAD*, pp. 92-95, 1990.

53. S. B. Akers, ''Binary Decision Diagrams,'' *IEEE Trans. on Comput.*, vol. 27, no. 6, pp. 509-516, June 1978.

54. B. M. E. Moret, ''Decision Trees and Diagrams,'' *Computing Surveys*, vol. 14, no. 4, pp. 594-623, December 1982.

55. K. S. Brace, R. L. Rudell, and R. E. Bryant, ''Efficient Implementation of a BDD Package,'' *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 40-45, June 1990.

56. S. S. Yau and C. K. Tang, ''Universal Logic Modules and Their Applications,'' *IEEE Trans. on Comp.*, vol. 19, no. 2, pp. 141-149, February 1970.

57. X. Chen and S. L. Hurst, ''A Consideration of the Minimum Number of Input Terminals on Universal Logic Gates and Their Realization,'' *Int. J. Electronics*, vol. 50, pp. 1-13, January 1981.

58. S. Bandyopadhyay, A. Pal, and A. K. Choudhury, ''Characterization of Unate Cascade Realizability Using Parameters,'' *IEEE Trans. on Comput.*, vol. 24, no. 2, pp. 218-219, February 1975.

59. D. G. Whitehead, ''Algorithm for Logic-Circuit Synthesis by Using Multiplexers,'' *Electronic Letters*, vol. 1977, no. 12, pp. 355-356, June 1977.

60. A. E. A. Almaini and M. E. Woodward, ''An Approach to the Control Variable Selection Problem for Universal Logic Modules,'' *Digital Processes*, vol. 3, pp. 189-206, 1977.

61. R.P. Voith, ''ULM Implicants for Minimization of Universal Logic Module Circuits,'' *IEEE Trans. on Comput.*, vol. 26, no. 5, pp. 417-424, May 1977.

62. B. Dormido and D. Canto, ''Systematic Synthesis of Combinational Circuits Using Multiplexers,'' *Electronic Letters*, vol. 14, no. 18, pp. 588-590, August 1978.

63. G. G. Langdon, ''A Decomposition Chart Technique to Aid in Realizations with Multiplexers,'' *IEEE Trans. on Comput.*, vol. 27, no. 2, pp. 157-159, February 1978.

64. L. A. M. Bennett, ‘‘The Application of Map-Entered Variables to the Use of Multiplexers in the Synthesis of Logic Functions,’’ *Int. J. Electronics*, vol. 45, no. 4, pp. 373-379, 1978.

65. D. Mange and E. Sanchez, ‘‘Synthese des Fonctions Logiques avec des Multiplexeurs,’’ *Digital Processes*, vol. 4, pp. 29-44, 1978.

66. A. J. Tosser and D. Aoulad-Syad, ‘‘Cascade Networks of Logic Functions Built in Multiplexer Units,’’ *IEE Proc. Pt. E*, vol. 127, no. 2, pp. 64-68, March 1980.

67. D. H. Green and M. A. Chughtai, ‘‘Use of Multiplexers in Direct Synthesis of ASM-based Designs,’’ *IEE Proc. Pt. E*, vol. 133, no. 4, pp. 194-200, July 1986.

68. A. Pal, ‘‘An Algorithm for Optimal Logic Design Using Multiplexers,’’ *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 755-757, August 1986.

69. R. K. Gorai and A. Pal, ‘‘Automated Synthesis of Combinational Circuits by Cascade Networks of Multiplexers,’’ *IEE Proc. Pt.E*, vol. 137, no. 2, pp. 164-170, March 1990.

70. A. M. Lloyd, ‘‘Design of Multiplexer Universal-Logic-Module Networks Using Spectral Techniques,’’ *IEE Proc. Pt. E.*, vol. 127, no. 1, pp. 31-36, January 1980.

71. T.F. Tabloski and F.J. Mowle, ‘‘A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits,’’ *IEEE Trans. on Comput.*, vol. 25, no. 7, pp. 684-702, July 1976.

72. Texas Instruments, *TPC10 Series 1.2-um Field-Programmable Gate Arrays*, July 1991.

73. R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Logic Synthesis for Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 620-625, 1990.

74. F. Mailhot and G. DeMicheli, ‘‘Technology Mapping Using Boolean Matching and Don’t Care Sets,’’ *IEEE Proc. of European Design Automation Conference*, pp. 212-216, Glasgow, March 1990.

75. K. Karplus, ‘‘Amap: A Technology Mapper for Selector-Based Field-Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 244-247, San Francisco, CA, June 1991.

76. S. Ercolani and G. DeMicheli, ‘‘Technology Mapping for Electrically Programmable Gate Arrays,’’ *Proc. 28th ACM/IEEE Design Automation Conference*, pp. 234-239, San Francisco, CA, June 1991.

77. Concurrent Logic Inc., ‘‘CLi6000 Series Field Programmable Gate Array,’’ *Preliminary Information*, December 1991.

78. M. Fujita, Y. Matsunaga, and T. Kakuda, ‘‘On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis,’’ *IEEE European Design Automation Conference*, pp. 50-54, Amsterdam, Netherland, February 1991.

79. N. Ishiura, H. Sawada, and S. Yajima, ‘‘Minimization of Binary Decision Diagrams Based on Exchanges of Variables,’’ *Proc. IEEE Int. Conf. on CAD*, pp. 472-475, Santa Clara, CA, November 1991.

80. D. L. Dietmayer, in *Logic Design of Digital Systems*, Allyn and Bacon, 1978.

81. R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘An Improved Synthesis Algorithm for Multiplexor-based PGAs,’’ *Proc. 1st ACM Workshop on FPGAs*, pp. 97-102, Berkeley, CA, February 1992.

82. A. Bedarida, S. Ercolani, and G. DeMicheli, ‘‘A New Technology Mapping Algorithm for the Design and Evaluation of Fuse/Antifuse-based Field-Programmable Gate Arrays,’’ *Proc. 1st ACM Workshop on FPGAs*, pp. 103-108, Berkeley, CA, February 1992.

83. K.-C. Chen, ‘‘Logic Minimization of Lookup-Table Based FPGAs,’’ *Proc. 1st ACM Workshop on FPGAs*, pp. 71-76, Berkeley, CA, February 1992.