

## Spectral Response of Ternary Logic Netlists

Mitchell A. Thornton and Theodore W. Manikas

*Southern Methodist University*

*Dallas, Texas USA 75275-0122*

*Email: mitch@lyle.smu.edu*

### Abstract

*Past methods for computation of the spectrum of a multiple-valued logic network usually rely on first characterizing the network in terms of a switching function, secondly in mapping the function values to complex numbers, and thirdly in performing the computation resulting in the spectrum. More recent approaches use decision diagram (DD) representations but still require initial formation of a DD representing the logic network switching function before the spectrum is computed. A method is described that derives a spectral transfer function directly from a netlist representation. The spectral transfer function can then be used to compute either the logic network spectral response for a specified input, or for computation of the entire Chrestenson spectrum. This method avoids the need for representing the network as a DD before computing the spectrum and can be used to directly compute either a single spectral response, the entire netlist spectrum, or the spectrum of subcircuits contained within a netlist.*

### 1. Introduction

Spectral methods as applied to logic circuit networks have been a topic of interest in the past with applications in both synthesis and analysis of logic netlists [1][2]. Widespread usage of spectral methods has not occurred in part due to the heavy computational burden associated with their computation and storage.

Improvements over use of the explicit transformation matrix have been achieved by representing the switching function response vector in more compact forms and through the application of elementary operations over each element within those more compact forms. Methods that have been devised to represent switching functions in a more compact manner include decision diagrams [3] and cube lists [4]. Unfortunately, these structures continue to have a worst case size of

$O(r^n)$  where  $r$  is the number of logic values and  $n$  is the number of logic network inputs. Furthermore, after the switching function representation is obtained, some means for computing and representing the spectral transformation matrix must also be employed. Computation of the spectrum of a switching function can be accomplished as operations directly over decision diagram structures [5] or through cube lists [4]. Even with these improvements, it is still necessary to extract a switching function representation from a netlist before the computation of the spectrum can occur.

Coupling decision diagram representations with the use of the so-called ‘fast’ algorithmic spectral computation methods is one means for reducing spectral computation complexity. The fast methods are commonly attributed to [6] and require the characterization of a radix- $r$  switching function as an exponentially large  $r^n$  vector that is then applied to  $\log(r^n)$  stages of intermediate computations (butterfly operations). However, this improved approach still requires the logic netlist to be initially modeled as a switching function from which the exponentially-sized  $r^n$  network response vector is formed.

Further savings in the computation of the spectrum can be achieved by representing the network response vector as a decision diagram (DD) and applying smaller butterfly operations of size  $r \times r$  to each DD vertex resulting in the spectrum being represented as a corresponding spectral decision diagram [5] [7] [8]. The DD representations allow for savings since many switching functions and their corresponding spectra can be represented more compactly as a DD rather than vectors of size  $r^n$ . However in the worst case, some switching functions still require exponentially large DDs which in turn require an exponential number of  $r \times r$  butterfly operations. Furthermore, some means for first converting a netlist into a DD structure must first be employed before the spectral transformation can occur.

It is often the case that a logic diagram, or intercon-

nection of logic gates, does not require an exponential number of gates and is much more compact than cube list or DD representations of the corresponding switching functions. Also, in modern design flows using Electronic Design Automation (EDA) software tools, logic networks are first characterized as compact higher-level Register Transfer Language (RTL) descriptions and then automatically synthesized into a logic network expressed in the form of a structural netlist. Thus, a netlist is a more commonly encountered structure in modern design flows as well as being a compact representation.

A netlist is a textual description of the structure of a logic network and may be expressed using a variety of Hardware Description Languages (HDL) [9]. A netlist description at the structural level is isomorphic to a logic network diagram and we represent logic circuit netlists as logic network diagrams here. We assume that the netlist is flattened and does not contain hierarchical subnetlists. Because spectral transformation algorithms model the logic network as a discrete switching function represented as a decision diagram or cube list, spectral methods are inconvenient. For this reason, we are motivated to find a spectral transformation method that operates directly upon the netlist.

Past work in the direct computation of a spectrum from a netlist includes that reported in [10] [11]. In [10] a method was described that allowed for a single spectral coefficient to be obtained but required augmenting the structure of a netlist and then traversing it for each spectral coefficient. In [11] a method was described where spectral coefficients can be obtained directly from a netlist, however the resulting coefficients are based upon a specific variable assignment. Here, a method for computation of the spectrum by traversing a structural netlist representation without modifying the netlist or performing a specific variable assignment is developed. Although any of a class of spectral transforms may be used, we focus upon the Chrestenson transform as described in [1] [12]. Methods similar to the one presented here were developed for the case of binary ( $r = 2$ ) logic networks where individual, partial, or entire Walsh and Reed-Muller spectra were computed [13] [14].

Our approach models each logic network gate with a transfer matrix and each logic value is modeled as an  $r$ -dimensional vector instead of a single integer or complex value. Any logic system that allows for the defining elementary logic operations to be modeled as transformation matrices is applicable to our technique. This representation allows the transfer matrices of each element to be transformed into the spectral domain and a technique for traversing the netlist to compute a

spectral transfer matrix is described.

Furthermore, this technique can be implemented by representing each transfer matrix as a DD. The implementation approach using DDs allows for an average reduction in the computational resource requirements for representing the matrices and vectors. The spectrum may be calculated through the use of DD traversal algorithms as described in [15]. Using DDs in our method provides a savings in memory and computation analogous to those of [5] [7] [8], although these previous methods are quite different from the approach we describe here.

## 2. Background and Notation

Vectors are denoted using “bra-ket” notation [16]. A row vector is denoted by  $\langle x|$  and a column vector is denoted as  $|x\rangle$ . Matrices are denoted as bold font, upper case characters such as  $\mathbf{A}$ . Vectors and matrices may be considered to be tensors of order one and order two respectively. The dimensions of vectors and matrices are described by non-zero integers that have value equivalent to the number of components composing them. A specific tensor is characterized by a number of such integers equivalent to their tensor order. For example, a vector is a tensor of order one and is characterized by a single dimensional value while a matrix is a tensor of order two and is characterized by a pair of dimensions whose values are the number of row and column components.

The inner product of two vectors is denoted as  $\langle x|y\rangle$  and the outer product as  $|x\rangle\langle y|$ . The tensor or outer product can be formed among two tensors regardless of their order or dimension. The outer product is a specific case of the Kronecker product and in the following we use the symbol  $\otimes$  to refer to either of these operations [17]. The Kronecker product may be applied to two matrices as  $\mathbf{A} \otimes \mathbf{B}$ . The form of the tensor operands clarifies which of these two multiplicative operations is being employed.

We use  $\mathbb{H}$  to represent a Hilbert vector space whose elements are three-dimensional row vectors that model ternary logic variable assignments.  $\mathbb{H}^n$  denotes a Hilbert vector space of expanded dimension and defining  $3^n$ -dimensioned vectors. The outer product operation is used to expand the dimensionality of a Hilbert space from  $\mathbb{H}$  to  $\mathbb{H}^n$  as shown in Equation 1 where  $n$  is any natural number,  $n \in \mathbb{N}$ .

$$\mathbb{H}^n = \mathbb{H} \otimes \mathbb{H} \otimes \dots \otimes \mathbb{H} = \bigotimes_{i=1}^n \mathbb{H} \quad (1)$$

Logic networks are mathematically modeled as mappings of vectors within the vector space  $\mathbb{H}^n$  to that of

$\mathbb{H}^m$  or equivalently as  $\mathbf{T} : \mathbb{H}^n \rightarrow \mathbb{H}^m$  where  $\mathbf{T}$  denotes a ‘transfer matrix’ [18].  $\mathbf{T}$  models the functionality of a logic netlist since it is a specific form of the concept of a transfer function [19]. Transfer functions allow a system output response to be calculated when multiplied by an input stimulus and the particular multiplication operation used with  $\mathbf{T}$  is the direct vector-matrix product.

A transfer matrix  $\mathbf{T}$  for a ternary logic network can be derived through use of truth table isomorphism or through netlist traversals and characterizes the input-output behavior in the switching domain. As is the case in other applications of engineering system theory, transfer functions may also be formulated in the spectral domain and used to determine an output response in the form of a spectral coefficient due to a particular input stimulus also expressed in the spectral domain. In this case, the spectral coefficient is referred to as the ‘spectral response’ [19]. In this paper, we derive the spectral transfer matrix  $\mathbf{T}_s$  for a ternary switching function by using the switching domain transfer matrix  $\mathbf{T}$ . Once the spectral transfer function is specified, it may be used to compute the spectral response for a given input stimulus expressed as a vector through a vector-matrix multiplication operation. Because the spectral transfer function is in the form of a transformation matrix, we refer to the transfer function as a ‘spectral transfer matrix.’

## 2.1. Ternary Logic Constant Models

In philosophical ternary logic systems such as those of Łukasiewicz and Kleene/Bochvar, logical outcomes are expressed as states from the sets  $\{true, false, indeterminate\}$  and  $\{true, false, undecidable\}$  respectively [20]. These philosophical logic systems include defined operations among the logic states such as disjunction, conjunction, negation, and others that are typically specified by a ‘truth table.’ In mathematical logic, an abstract algebraic structure is used to describe a logic system where the algebra consists of a set of values, a set of operators, and where the set of values contains members that serve as identities with respect to the various operators. Formally, an algebra must also generally adhere to certain other properties such as closure. The specific algebra chosen in a mathematical logic system is typically one where the defining philosophical logic operations are represented by distinct algebraic operators and where the set of values corresponds to the philosophical logic states. From this point of view, the particular values in the set are arbitrary in the sense that they may be any

type of objects so long as they are consistent with the algebraic operators and properties and that they contain the required identity values.

The application of interest here is that of modeling the functionality of MVL switching circuits as discrete MVL functions. Thus, any algebra can be used so long as it contains operators that allow for modeling all MVL functions and likewise the set of values must be consistent with the defined algebraic operators. From this perspective, the approach described here is not restricted to any particular mathematical logic system as long as the logic gates comprising a logic network may have their functionality expressed in terms of operations from the defining algebraic model. This flexibility justifies our use of a set of vectors instead of the more commonly used sets of scalars or complex values to model the ternary logic values.

Although three-valued switching circuit models commonly use the integers  $\{0, 1, 2\}$  to represent logic values, these values can also be expressed as three distinct complex-valued cube roots of unity denoted as  $a_k$  corresponding to the integral logic value  $k \in \{0, 1, 2\}$ . Equation 2 contains the relationship between the integral ( $k$ ) and complex-valued ( $a_k$ ) logic constants. This complex-valued encoding is convenient for some forms of spectral analysis, in particular the Chrestenson spectrum where logic functions are expressed as complex-weighted sums of the discretized Chrestenson basis functions and where the basis functions are expressed in terms of  $a_k$  as in [1] [12].

$$a_k = e^{i\frac{2k\pi}{3}} \quad (2)$$

Because we model the functionality of a logic network with a linear transformation matrix, an alternative set of ternary logic value encodings is used for consistency with the logic network model. Table 1 contains numerical encodings for ternary logic values that we refer to as ‘scalar’ mappings and the new encodings referred to as ‘vector’ mappings. We note that one of the two scalar mappings in Table 1 contains the three complex roots of unity and that a complex value may be considered to be a two-dimensional vector in the complex plane. We nevertheless refer to this as a ‘scalar complex’ mapping to describe that it is a mapping of the *scalar* switching values to corresponding complex values. Likewise, the new encodings used here are referred to as a ‘vector’ mapping since ternary logic values are expressed as row vectors whose components are either integers or complex values.

Vector switching constants are denoted as row vectors  $\langle i \mid$  where  $i \in \{0, 1, 2\}$  represents a distinct

Table 1. Ternary Logic Constants

Scalar		Vector	
Switching	Complex	Switching	Complex
0	$a_0$	$\langle 0 \rangle$	$\langle c_0 \rangle$
1	$a_1$	$\langle 1 \rangle$	$\langle c_1 \rangle$
2	$a_2$	$\langle 2 \rangle$	$\langle c_2 \rangle$

Vector Constant Definitions			
$\langle 0 \rangle =$	$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$	$\langle c_0 \rangle =$	$\begin{bmatrix} a_0 & a_0 & a_0 \end{bmatrix}$
$\langle 1 \rangle =$	$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$	$\langle c_1 \rangle =$	$\begin{bmatrix} a_0 & a_2 & a_1 \end{bmatrix}$
$\langle 2 \rangle =$	$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$	$\langle c_2 \rangle =$	$\begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix}$

ternary logic valuation. Additionally, we define the ‘null’ vector  $\langle \emptyset \rangle = [0 \ 0 \ 0]$  that represents the absence of any specific logic valuation. The vector complex constants denoted by  $\langle c_0 \rangle$ ,  $\langle c_1 \rangle$ , and  $\langle c_2 \rangle$  are composed of complex-valued components  $a_i$  as defined in Table 1 and can be computed in terms of the corresponding switching vector constants and the Chrestenson transform matrix as  $\langle c_i \rangle = \langle i | \mathbf{C}_1^*$  for  $i \in \{0, 1, 2\}$ .

The vector space expansion operation in Equation 1 allows for vectors in lower-dimensional spaces to be combined into a single vector in a higher dimensional space. This is useful in our approach since it allows for  $n$  different ternary logic values in vector form to be represented by a single  $3^n$ -dimensional vector instead of  $n$  different three-dimensional vectors.

## 2.2. Logic Network Elements

Logic networks are defined as a collection of primitive operators that are depicted in a graphical form and interconnected forming a netlist. Each element is modeled mathematically as a transformation over vectors in  $\mathbb{H}$ . In this work, we utilize the ternary *MIN*, *MAX*, and  $J_k$  (unary literal selection gate) operators as example logic primitives [20]. Figure 1 depicts some corresponding netlist symbols, their truth tables, and their switching transfer matrices.

The output response of a network element,  $\langle f \rangle$ , is obtained by multiplying the corresponding input stimulus vector,  $\langle x \rangle$ , with the element transfer matrix  $\mathbf{T}$  as shown in Equation 3. Input stimulus vectors are obtained through the use of Equation 1 where the vector representation of each individual network input value is combined into a single vector  $\langle x \rangle$ .

$$\langle f \rangle = \langle x | \mathbf{T} \quad (3)$$

## 2.3. Chrestenson Transform

The Chrestenson transform is a discrete orthogonal transform whose basis functions are a set of generalized Walsh functions [21]. Equations 4 and 5 are used







Truth Table	Logic Symbol	Expression	Name	Matrix																														
<table border="1"> <tr><th>X</th><th>Y</th><th>F</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>2</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>2</td><td>1</td></tr> <tr><td>2</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>2</td></tr> </table>	X	Y	F	0	0	0	0	1	0	0	2	0	1	0	0	1	1	1	1	2	1	2	0	0	2	1	1	2	2	2		$F = X \cdot Y$	<i>MIN(X,Y)</i>	$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
X	Y	F																																
0	0	0																																
0	1	0																																
0	2	0																																
1	0	0																																
1	1	1																																
1	2	1																																
2	0	0																																
2	1	1																																
2	2	2																																
<table border="1"> <tr><th>X</th><th>Y</th><th>F</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>2</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>2</td><td>2</td></tr> <tr><td>2</td><td>0</td><td>2</td></tr> <tr><td>2</td><td>1</td><td>2</td></tr> <tr><td>2</td><td>2</td><td>2</td></tr> </table>	X	Y	F	0	0	0	0	1	1	0	2	2	1	0	1	1	1	1	1	2	2	2	0	2	2	1	2	2	2	2		$F = X + Y$	<i>MAX(X,Y)</i>	$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$
X	Y	F																																
0	0	0																																
0	1	1																																
0	2	2																																
1	0	1																																
1	1	1																																
1	2	2																																
2	0	2																																
2	1	2																																
2	2	2																																
<table border="1"> <tr><th>X</th><th>F</th></tr> <tr><td>0</td><td>2</td></tr> <tr><td>1</td><td>0</td></tr> <tr><td>2</td><td>0</td></tr> </table>	X	F	0	2	1	0	2	0		$F = J_0(X) = X^{00}$	$J_0(X)$	$\mathbf{J0} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$																						
X	F																																	
0	2																																	
1	0																																	
2	0																																	
<table border="1"> <tr><th>X</th><th>F</th></tr> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>2</td></tr> <tr><td>2</td><td>0</td></tr> </table>	X	F	0	0	1	2	2	0		$F = J_1(X) = X^{01}$	$J_1(X)$	$\mathbf{J1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$																						
X	F																																	
0	0																																	
1	2																																	
2	0																																	
<table border="1"> <tr><th>X</th><th>F</th></tr> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td></tr> <tr><td>2</td><td>2</td></tr> </table>	X	F	0	0	1	0	2	2		$F = J_2(X) = X^{02}$	$J_2(X)$	$\mathbf{J2} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$																						
X	F																																	
0	0																																	
1	0																																	
2	2																																	
<table border="1"> <tr><th>X</th><th>F1</th><th>F2</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>2</td></tr> </table>	X	F1	F2	0	0	0	1	1	1	2	2	2		$F_1 = X$ $F_2 = X$	<i>FOC(X)</i>	$\mathbf{FO}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$																		
X	F1	F2																																
0	0	0																																
1	1	1																																
2	2	2																																

Figure 1. Ternary Logic Gate Representations

to construct the Chrestenson transform matrix  $\mathbf{C}_n$  for any  $n \in \mathbb{N}$ .

$$\mathbf{C}_1 = \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_1 & a_2 \\ a_0 & a_2 & a_1 \end{bmatrix} \quad (4)$$

$$\mathbf{C}_n = \bigotimes_{i=1}^n \mathbf{C}_1 \quad (5)$$

**2.3.1. Scalar Chrestenson Spectrum of  $f$ .** The scalar complex encoded Chrestenson spectrum of a logic network is a column vector of spectral coefficients  $|s_f\rangle$  and is computed through multiplication of  $\mathbf{C}_m^*$  with a column vector of the complex scalar encoded values of the function  $f$  denoted as  $|f_c\rangle$ . Equation 6 expresses this relationship as used in [1] [12].

$$|s_f\rangle = \mathbf{C}_m^* |f_c\rangle \quad (6)$$

**2.3.2. Vector Chrestenson Spectrum of  $f$ .** The vector Chrestenson spectrum of a switching function  $f$  consists of a column of row vectors resulting in matrix

$\mathbf{S}_f$ . Each row vector in  $\mathbf{S}_f$  is a distinct Chrestenson spectral coefficient. We therefore refer to this form of the Chrestenson spectrum as the ‘vector Chrestenson spectrum.’

The vector Chrestenson spectrum  $\mathbf{S}_f$  is calculated using the relationship in Equation 6 where the column vector of scalar complex conjugate encodings for the function  $|f_c\rangle$  is replaced with matrix  $\mathbf{F}_c$ . Each row of  $\mathbf{F}_c$  is the vector complex encoding of  $f$  values,  $\langle f_c|$ , and  $\mathbf{F}$  is the switching domain transfer matrix. The transfer matrix  $\mathbf{F}$  can be viewed as a single column of row vectors where each row vector is the switching vector encoded truth value of function  $f$ , hence Equation 6 yields  $\mathbf{F}_c$ , the complex vector encoded form of  $\mathbf{F}$ .

$$\mathbf{F}_c = \mathbf{F}\mathbf{C}_n^* \quad (7)$$

Using Equations 6 and 7, the relationship for the computation of the vector Chrestenson spectrum of the function  $f$  is given in Equation 8.

$$\mathbf{S}_f = \mathbf{C}_n^* \mathbf{F}_c \quad (8)$$

*Example 2.1: Scalar Chrestenson Spectrum* Consider the  $J_1$  (literal selection) gate and scalar switching truth table shown in Figure 1. The scalar Chrestenson spectrum for the  $J_1$  gate is computed as:

$$\begin{aligned} |c_{J_1}\rangle &= \mathbf{C}_1^* |J_{1c}\rangle = \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_2 & a_1 \\ a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ a_0 \end{bmatrix} \\ &= \begin{bmatrix} 2a_0 + a_2 \\ a_0 + 2a_1 \\ 2a_0 + a_2 \end{bmatrix} \quad \square \end{aligned}$$

*Example 2.2: Vector Chrestenson Spectrum* To compute the vector Chrestenson spectrum of the  $J_1$  gate, we first formulate the vector complex encoded values of  $J_1$  as the matrix  $\mathbf{J}_{1c}$  through the application of the mapping in Equation 7.

$$\begin{aligned} \mathbf{J}_{1c} &= (\mathbf{J}_1)(\mathbf{C}_1^*) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_2 & a_1 \\ a_0 & a_1 & a_2 \end{bmatrix} \\ &= \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_1 & a_2 \\ a_0 & a_0 & a_0 \end{bmatrix} \end{aligned}$$

The calculation of the spectral matrix  $\mathbf{S}_{J_1}$  is then accomplished using Equation 8.

$$\begin{aligned} \mathbf{S}_{J_1} &= (\mathbf{C}_1^*)(\mathbf{J}_{1c}) \\ &= \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_2 & a_1 \\ a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} a_0 & a_0 & a_0 \\ a_0 & a_1 & a_2 \\ a_0 & a_0 & a_0 \end{bmatrix} \\ &= \begin{bmatrix} (3a_0) & (2a_0 + a_1) & (2a_0 + a_2) \\ (a_0 + a_1 + a_2) & (2a_0 + a_1) & (a_0 + 2a_1) \\ (a_0 + a_1 + a_2) & (a_0 + 2a_2) & (2a_0 + a_2) \end{bmatrix} \quad \square \end{aligned}$$

### Theorem 2.3: Scalar and Vector Spectrum Relation

The scalar Chrestenson spectrum and the rightmost column of the vector Chrestenson spectrum of a function representing the same logic network are identical.

*Proof:* For a given switching function  $f$ , the scalar Chrestenson spectrum is given as  $|s_f\rangle = \mathbf{C}_n^* |f_c\rangle$  and the vector Chrestenson spectrum is given as  $\mathbf{S}_f = \mathbf{C}_n^* \mathbf{F}_c$ . We define a vector  $|v\rangle$  of length  $3^n$  composed of  $3^n - 1$  zero-valued components and a single unity-valued component of the form,  $\langle v| = [0 \ 0 \ \dots \ 0 \ 1]$ . The rightmost column of the vector Chrestenson spectrum can be formed by the product  $\mathbf{S}_f |v\rangle$ , yielding:

$$\mathbf{S}_f |v\rangle = \mathbf{C}_n^* \mathbf{F}_c |v\rangle \quad (9)$$

Subtracting Equation 6 from Equation 9 results in:

$$\begin{aligned} \mathbf{S}_f |v\rangle - |s_f\rangle &= \mathbf{C}_n^* \mathbf{F}_c |v\rangle - \mathbf{C}_n^* |f_c\rangle \\ &= \mathbf{C}_n^* (\mathbf{F}_c |v\rangle - |f_c\rangle) \end{aligned} \quad (10)$$

For a given switching function  $f$ ,  $\mathbf{F}_c$  and  $|f_c\rangle$  result by expressing the components of  $|f\rangle$  using the vector complex and scalar complex encoded values from Table 1. Examination of the encodings specified in Table 1 reveals that the scalar complex and vector complex encodings are identical in that they both have value  $a_i$  corresponding to  $f$  having a scalar switching value of  $i$ . Thus, the Equation 10 term  $(\mathbf{F}_c |v\rangle - |f_c\rangle) = |\emptyset\rangle$ . Substituting this observation into Equation 10 results in  $\mathbf{S}_f |v\rangle - |s_f\rangle = \mathbf{C}_n^* |\emptyset\rangle = |\emptyset\rangle$ . This result can only occur if the rightmost column of  $\mathbf{S}_f$  is equivalent to  $|s_f\rangle$ .  $\square$

## 3. Spectral Transfer Matrix

The spectral transfer matrix  $\mathbf{T}_s$  represents the spectral response of a logic network. When  $\mathbf{T}_s$  is multiplied with the complex vector representation of a logic network input stimulus, the corresponding spectral response results. Figure 2 contains a block diagram where a ternary logic network is represented by a spectral transfer matrix  $\mathbf{T}_s$  with an input stimulus matrix  $\mathbf{X}_c$  and a spectral response matrix  $\mathbf{S}_f$ .

*Theorem 3.1: Spectral Transfer Matrix* The spectral transfer matrix  $\mathbf{T}_s$  for a logic network is related to the switching transfer matrix  $\mathbf{T}$  as given in Equation 11.

$$\mathbf{T}_s = \mathbf{T}\mathbf{C}_m^* \quad (11)$$

*Proof:* Substituting Equation 7 into Equation 8 results in:

$$\mathbf{S}_f = \mathbf{C}_n^* \mathbf{F}_c \mathbf{C}_m^* \quad (12)$$

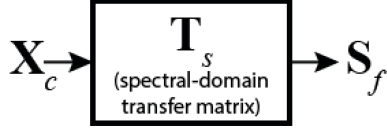


Figure 2. Diagram of Spectral Transfer Matrix

Using the notion of a switching transfer matrix, the switching domain output response,  $\langle f |$ , due to a single switching vector input stimulus  $\langle x |$  is computed as  $\langle f | = \langle x | \mathbf{T}$ . If all possible valuations  $\langle x_i |$  (where  $0 < i < 3^n - 1$ ) are represented as a column of row vectors, a matrix  $\mathbf{X}$  results with each row equivalent to  $\langle x_i |$ . Using  $\mathbf{X}$  to compute the total switching vector response results in  $\mathbf{F} = \mathbf{X}\mathbf{T}$ . Furthermore, due to the switching vector encoding definition given in Table 1, it is observed that  $\mathbf{X} = \mathbf{I}$  where  $\mathbf{I}$  is the identity matrix, hence  $\mathbf{F} = \mathbf{X}\mathbf{T} = \mathbf{I}\mathbf{T} = \mathbf{T}$ .

Substituting this result into Equation 12 results in:

$$\mathbf{S}_f = \mathbf{C}_n^* \mathbf{X} \mathbf{T} \mathbf{C}_m^* = \mathbf{C}_n^* \mathbf{T} \mathbf{C}_m^* \quad (13)$$

It is observed that the leftmost  $\mathbf{C}_n^*$  factor in the expression  $\mathbf{C}_n^* \mathbf{T} \mathbf{C}_m^*$  can be considered to be composed of a single column of row vectors where each row vector represents the vector complex encoded input values  $\mathbf{X}_c$ . From this observation, we have  $\mathbf{S}_f = \mathbf{X}_c \mathbf{T} \mathbf{C}_m^*$ . Hence,  $\mathbf{T}_s = \mathbf{T} \mathbf{C}_m^*$ .  $\square$

**3.0.3. Spectral Response Definition.** In keeping with the terminology of linear systems analysis [19], we refer to the logic network output in switching vector encoded form as the ‘switching response’ of a logic network. Likewise, the ‘total switching response’ is the complete set of all switching responses for all possible valuations of input stimuli  $\langle x |$ . When all valuations of  $\langle x |$  are written as a column of system input stimuli, the matrix  $\mathbf{X} = \mathbf{I}$  results and thus the total switching response  $\mathbf{F}$  is identical to the switching transfer matrix  $\mathbf{T}$  since  $\mathbf{F} = \mathbf{X}\mathbf{T} = \mathbf{T}$ .

Analogous to the switching response of a logic network, the ‘spectral response’ is the vector complex encoded output response due to a specific vector complex encoded input stimulus  $\langle x_c |$ . The ‘total spectral response’ is a column of vector complex encoded spectral coefficients and is the complete spectrum  $\mathbf{S}_f$ . The spectral response of a logic network due to a single valuation of an input stimulus  $\langle x_c |$  is the single spectral coefficient  $\langle s_f |$ . The spectral response of a logic network characterized by  $\mathbf{T}_s$  can be calculated as shown in Equation 14.

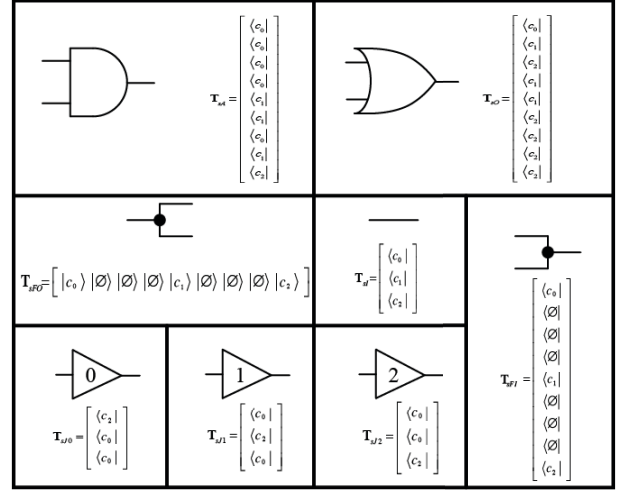


Figure 3. Ternary Network Element Spectral Transfer Matrices

$$\langle s_f | = \langle x_c | \mathbf{T}_s \quad (14)$$

## 4. Spectral Transfer Matrix from Netlist

Applying Theorem 3.1 to the network element transfer matrices given in Figure 1 results in the corresponding spectral transfer matrices as shown in Figure 3.

### 4.1. Spectral Transfer Matrix Procedure

The spectral transfer matrix of a logic network represented as a netlist, can be obtained based on a traversal of the network and knowledge of the individual network element spectral and switching transfer matrices. This technique is summarized below.

- 1) Determine  $k$  logic network partitions  $\phi_j$  where  $1 \leq j \leq k$  by performing cuts such that each partition  $\phi_j$  is comprised of a set of parallel network elements with no serial or cascaded elements and where the first partition,  $\phi_1$ , is closest to the logic network inputs.
- 2) Compute the switching transfer matrices,  $\mathbf{T}_{\phi_j}$  for each partition  $\phi_j$  where  $1 \leq j < k$  by combining the parallel network element switching transfer matrices using the outer product operation.
- 3) Compute the spectral transfer matrix,  $\mathbf{T}_{s\phi_k}$  corresponding to partition  $\phi_k$  by combining the parallel network element spectral transfer matrices using the outer product operation. This step combines the individual steps of using the

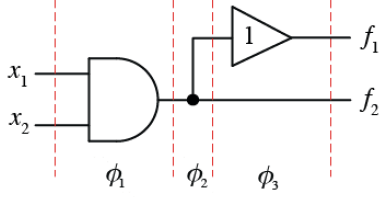


Figure 4. Partitioned Ternary Logic Network

switching transfer matrix for partition  $k$  and then multiplying the overall circuit transfer matrix with  $\mathbf{C}^*$  into a single operation.

- 4) Compute the overall network spectral transfer matrix by combining the spectral transfer matrix for partition  $\phi_k$  with each switching transfer matrix of the remaining partitions using the direct matrix product.

**4.1.1. Example  $\mathbf{T}_s$  Computation.** To illustrate the method for forming the spectral transfer matrix, we depict the example ternary network in Figure 4 with partition cuts labeled  $\phi_1$ ,  $\phi_2$ , and  $\phi_3$ .

Step 1) parses in the netlist file and produces an internal data structure that represents the logic network. During the parsing process, serial partitions are detected. The partitions used here are identical to those described in the unrelated work of [22]. This portion of the procedure has  $O(N)$  complexity where  $N$  is the number of logic gates in the netlist. The netlist parsing operation is commonly implemented in modern EDA software tools.

Step 2) of the procedure determines the switching transfer matrices of the two leftmost partitions. Fanout points are represented by the **FO** switching transfer matrix and are included in the calculation since they expand the dimension of the vector space. These two switching transfer matrices are  $\mathbf{T}_{\phi_1} = \mathbf{A}$  and  $\mathbf{T}_{\phi_2} = \mathbf{FO}$  respectively.

Step 3) of the procedure determines the spectral transfer matrix for partition  $\phi_3$ ,  $\mathbf{T}_{s\phi_3} = \mathbf{T}_{sJ_1} \otimes \mathbf{T}_{sI}$ .

The overall spectral transfer matrix for the network is computed in Step 4) as the direct matrix product of the partition transfer matrices and is given by  $\mathbf{T}_s = (\mathbf{A})(\mathbf{FO})(\mathbf{T}_{sJ_1} \otimes \mathbf{T}_{sI})$ . The explicit spectral transfer matrix for the example network is given in Equation 15.

$$\mathbf{T}_s = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_2 & a_1 & a_1 & a_0 & a_2 & a_2 & a_1 & a_0 & a_0 \\ a_0 & a_2 & a_1 & a_1 & a_0 & a_2 & a_2 & a_1 & a_0 & a_0 \\ a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_2 & a_1 & a_1 & a_0 & a_2 & a_2 & a_1 & a_0 & a_0 \\ a_0 & a_1 & a_2 & a_0 & a_1 & a_2 & a_0 & a_1 & a_2 & a_2 \end{bmatrix} \quad (15)$$

The spectral transfer matrix for each of the outputs  $f_1$  and  $f_2$  can also be separately computed through application of the spectral transfer matrix construction procedure for the portion of the network applicable to each output. The resulting spectral response values for each individual network output are in the form of a row vector consisting of three elements.

Alternatively, the spectral response due to a single input stimulus vector may also be obtained by forming the spectral mapping of a specific input value and traversing the logic network and simultaneously multiplying the complex vector input stimulus of each stage with the corresponding switching transfer matrices followed by multiplying the output response of stage  $k - 1$  with the spectral transfer matrix for stage  $k$ . In this manner, the spectral response of the logic network may be computed through a traversal of the netlist and avoids performing a vector-matrix product after the overall  $\mathbf{T}_s$  matrix is computed.

#### 4.1.2. Spectral Response through Netlist Traversal.

An alternative to computing the entire  $\mathbf{T}_s$  matrix is to compute a spectral response during the traversal of the netlist. This approach avoids the explicit representation of the overall  $\mathbf{T}_s$  matrix and allows for spectral coefficients to be obtained with only a netlist traversal. Initially, the complex vector representing the input stimulus is formed and is multiplied with the transfer matrix of the first partition. The resulting vector is then multiplied by the second partition matrix and the process is repeated for each partition stage. The final resulting complex vector is the spectral response of the netlist corresponding to the initially specified input stimulus vector. This approach avoids computation of the overall  $\mathbf{T}_s$  matrix and a final step of using  $\mathbf{T}_s$  with an input stimulus to determine the spectral response is not required. The complexity of this approach is reduced to storage of the largest partition transfer matrix and requires  $P$  vector-matrix multiplies where  $P$  is the total number of partitions. It is only necessary to store each partition matrix at any instant in time.

## 5. Conclusion

A method for computation of the spectral transfer matrix for a ternary logic network is described. The technique allows the spectral transfer matrix to be computed through a traversal of a netlist representing the logic network. The spectral transfer matrix may be used to compute the entire spectrum of the network or a single or subset of spectral responses (coefficients). Furthermore, the method for computing the spectral transfer matrix may be augmented by pre-multiplying the input switching transfer matrices with a complex vector representation of a specific network input value resulting in computation of the logic network spectral response through a traversal of the netlist. This latter approach avoids the need to perform a vector-matrix product using  $\mathbf{T}_s$  after the partitioning and partition matrix operations are performed.

## References

- [1] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*. Wiley and JUP, 1976.
- [2] S. L. Hurst, J. C. Muzio, and D. M. Miller, *Spectral Techniques in Digital Logic*. Academic Press Publishers, 1985.
- [3] D. M. Miller and R. Drechsler, "Implementing a multiple-valued decision diagram package," in *Proceedings. 1998 28th IEEE International Symposium on Multiple-Valued Logic*, May 1998, pp. 52–57.
- [4] B. J. Falkowski, I. Schaefer, and M. A. Perkowski, "Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 10, pp. 1207–26, 1992.
- [5] D. M. Miller, "Graph algorithms for the manipulation of Boolean functions and their spectra," in *Proceedings. 1987 Congressus Numeratum*, 1987, pp. 177–199.
- [6] J. T. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex fourier series," *Math. Computation*, vol. 19, pp. 297–301, 1965.
- [7] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD*. Kluwer Academic Publishers, 2001.
- [8] R. Stankovic' and J. Astola, *Spectral Interpretation of Decision Diagrams*. Springer-Verlag Publishers, 2003.
- [9] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. McGraw-Hill Publishers, 1994.
- [10] R. Drechsler and M. A. Thornton, "Computation of spectral information from logic netlists," in *Proceedings. 2000 IEEE International Symposium on Multiple-Valued Logic*, May 2000, pp. 53–58.
- [11] R. Krenz, E. Dubrova, and A. Kuehlmann, "Fast algorithm for computing spectral transforms of Boolean and multiple-valued functions on circuit representation," in *Proceedings. 2003 IEEE International Symposium on Multiple-Valued Logic*, May 2003, pp. 334–339.
- [12] C. Moraga, "Complex spectral logic," *Proceedings. IEEE International Symposium on Multiple-Valued Logic*, pp. 149–156, 1978.
- [13] M. A. Thornton, "Spectral analysis of digital logic circuit netlists," in *Proceedings. 2011 International Conference on Computer-Aided Systems Theory*, Feb. 2011, pp. 414–415.
- [14] M. A. Thornton and J. Dworak, "Direct Reed-Muller transform of digital logic netlists," in *Proceedings. 2011 Applications of the Reed-Muller Expansion in Circuit Design*, May 2011, pp. 11–20.
- [15] E. M. Clarke, M. Fujita, P. C. McGeer, K. McMillan, J. C. Yang, and X. Zhao, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," in *Proceedings. IEEE Int. Workshop on Logic Synthesis*, 1993, pp. 1–15.
- [16] P. A. M. Dirac, "A new notation for quantum mechanics," *Proc. of the Cambridge Philosophical Society*, vol. 54, p. 416, 1939.
- [17] H. Yang and G. He, "Some properties of matrix product and its applications in nonnegative tensor decomposition," *Journal of Information and Computing Science*, vol. 3, no. 4, pp. 269–280, 2008.
- [18] M. A. Thornton, "A transfer function model for ternary switching logic circuits," in *Proceedings. 2013 IEEE International Symposium on Multiple-Valued Logic*, May 2013.
- [19] C. T. Chen, *Linear System Theory and Design*. Holt, Rinehart and Winston, 1984.
- [20] D. M. Miller and M. A. Thornton, *Multiple Valued Logic: Concepts and Representations*. Morgan & Claypool Publishers, 2007.
- [21] H. E. Chrestenson, "A class of generalized Walsh functions," *Pacific Journal of Mathematics*, vol. 5, pp. 17–31, 1955.
- [22] V. I. Levin, "Probability analysis of combination systems and their reliability," *Engineering Cybernetics*, no. 6, pp. 78–83, 1964.