Simulation and Implication Using a Transfer Function Model for Switching Logic

Mitchell A. Thornton, *Senior Member, IEEE*

Abstract—Transfer functions are concise mathematical models representing the input/output behavior of a system and are widely used in many areas of engineering including system theory and signal analysis. We develop a framework for the construction of transfer function models for digital networks and demonstrate their application in simulation and implication. Rather than using a traditional switching theory model, the transfer functions are defined over vector spaces, $\mathbb{H}$. The derivation of the transfer function is provided and it is applied to logic network simulation and implication.

Index Terms—Switching theory, logic design, simulation, modeling, computer-aided design

1 Introduction

Switching theory provides a rich theoretical basis for modeling digital logic circuits [1], [2]. Traditionally, digital logic circuits are modeled using the axioms and postulates of switching theory formulated in terms of a binary-valued Boolean algebra over discrete scalar-valued switching functions. The switching theory framework has led to an extensive set of analysis and synthesis methods that continue to be commonly used in all facets of modern digital circuit design activities.

In the approach discussed here, we reformulate these mathematical models in terms of linear transforms over vector spaces. This result is not intended to serve as a replacement for traditional switching theory, rather as an alternative model that exhibits advantages for certain electronic design automation (EDA) tasks such as symbolic simulation and implication that can be computationally cumbersome within a switching theory framework.

The formulation of a linear algebraic model in place of a switching theory model allows for alternative representations and corresponding analysis and synthesis techniques to be investigated. Here, we focus upon digital logic network modeling and representation within a linear algebraic framework although these results can be applied to any system of first-order propositional logic. The underlying computationally complex problems of logic network representation and manipulation remain within the linear algebraic framework; however, this alternative representation can allow for new heuristic analysis and synthesis methods to be developed that may prove advantageous as compared to corresponding switching theoretical approaches.

Transfer functions describe the input-output behavior of a system of interest and generally have the form of a numerator representing the complete system output behavior and a denominator representing the complete input behavior [3]. The system response with respect to a particular input stimulus can then be obtained through a multiplicative operation among the stimulus and transfer function. The inverse transfer function may be used to determine a corresponding input stimulus given an output response; also through the application of a multiplicative operation. In terms of digital logic network operations, these tasks are commonly referred to as simulation and implication respectively. Because simulation and implication are core operations in many EDA tasks, the use of a transfer function model can prove useful in various applications.

In order for the transfer function approach to be viable, the extraction of the function representation must not be excessively computationally complex and the representation must furthermore not require excessive storage requirements. EDA design tasks involve the transformation of a high-level description of a logic network into a corresponding low-level model while analysis tasks often require the opposite situation; that of transforming a low-level model to a higher one. For this reason, a means to efficiently extract a transfer function from a high-level description as well as a low-level description is required.

Here, we utilize truth tables as a representative high-level description although modern EDA tools utilize more concise representations such as binary decision diagrams (BDD) or cube lists [4], [5]. Although truth table representations are exponentially complex, our techniques are easily extended to the use of these more modern forms of high-level representation and we use truth tables here only for the sake of clarity in explanation of the concepts. Alternatively, we use a netlist or interconnection of basic logic gates as a representative low-level description and we refer to this representation as a 'logic network'.

As opposed to modeling the functionality of a logic network with switching theory [1], [2], we model logic networks as operations over a vector space. This approach is not without precedent and is commonly used in the fields of quantum and reversible logic [6], [7], and the use of this approach for classical logic networks was very briefly

• *The author is with the Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275. E-mail: mitcht@ieee.org.*

Manuscript received 25 July 2012; revised 10 Sept. 2013; accepted 19 Jan. 2015. Date of publication 5 Feb. 2015; date of current version 11 Nov. 2015. Recommended for acceptance by M. Hsiao.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2401034

mentioned in [8] and investigated in terms of spectral responses in [9]. By formulating logic network transfer functions as transformations over vector spaces, the resulting transfer function model of a logic network is easily derived from both high-level and low-level specifications and is of the form of a linear transformation matrix. Hence, we interchangeably refer to the network model as either a 'transfer function' or a 'transfer matrix'.

The use of the transfer function for determination of a network response after stimulation with a test vector is described and extended to provide a means for obtaining multiple responses through a single evaluation of the transfer function. This latter case is an instance of symbolic simulation and has many applications in EDA forming the basis for simulator, formal verification, and test generation tools [10], [11].

The generalized inverse of the transfer function is also presented and applied to the EDA task of implication where a given digital network input stimulus is calculated based upon a given output response and logic network model. The key factors in implementing an implication method are efficient extraction and representation of the logic network inverse model, and efficiency in the actual implication computation given the inverse model and the output response.

2 Switching Theory and Linear Algebra Concepts and Notation

Switching theory is ubiquitous within the logic network design community and basic concepts are well-known by practitioners. An extremely brief review of switching theory is provided for the purpose of specifying the notation used in the development of the alternative vector space characterization of a logic network.

2.1 Switching Theory Algebra and Notation

Switching theory is the application of the principles of binary-valued Boolean algebra for the purpose of modeling switching or digital logic circuits. Because digital circuits operate in two discrete states, it is convenient to model them in terms of switched states. In keeping with practice, we use the term 'Boolean algebra' to imply that that algebra is defined for the set of constants $\{0,1\}$ denoted as $\mathbb{B} = \{0,1\}$ [12].

Boolean algebra can be specified in a variety of ways with one of the most common being that of $\langle \mathbb{B}, +, \cdot, ', 0, 1\rangle$, where $+$ is the additive operation also known as the inclusive-OR, $\cdot$ is the multiplicative operator also known as the two-operand AND, $'$ is the unary multiplicative inverse also known as the NOT, and 0 and 1 are the additive and multiplicative identity elements respectively.

A Boolean algebra variable x can be represented as the expression $x = a \cdot 0 + b \cdot 1$ where $a, b \in \{0,1\}$. In this form, x is written as function that depends upon the particular assignment of constants to coefficients a and b where the particular assignment is considered to be unknown. The four unary functions are the two constant functions, $f(x) = 0$, $f(x) = 1$, the identity, $f(x) = x$, and the inverse, $f(x) = x'$. Application of the unary inverse operation to a variable x yields its complement, or functional inverse, and is denoted as $f(x) = x' = b \cdot 0 + a \cdot 1$. x' is referred to as the 'negative polarity' form of variable x, while x is the 'positive polarity' form. Both x and x' are 'literals' since they represent the positive and negative polarity respectively of variable x.

Switching functions describe relationships between multi-dimensioned finite sets of constants that represent mappings between a domain set $\mathbb{B}^n$ to a range set $\mathbb{B}^m$ where m and n are integers ≥ 1. The multi-dimensioned spaces $\mathbb{B}^n$ and $\mathbb{B}^m$ are formed as Cartesian products of $\mathbb{B}$, $\mathbb{B}^n = \mathbb{B} \times \mathbb{B} \times \cdots \times \mathbb{B} = \Pi_{1\leq i\leq n}(\mathbb{B}_i)$. A switching function f that depends upon n variables is denoted as $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, where m denotes that the function range set contains elements within $\mathbb{B}^m$. A logic network with n inputs and m outputs is modeled as a switching function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$.

2.2 Linear Algebra and Notation

Algebras can be formulated over sets of elements that are not necessarily scalar values. Common non-scalar quantities are one-dimensional arrays of scalars referred to as vectors and two-dimensional arrays called matrices. The notion of a tensor can be used to generalize scalar and non-scalar quantities [13]. Different forms of tensors are denoted by an integer that is their order or degree. The order of a tensor refers to the number of indices required to specify a single value within the tensor and the 'dimension' is the maximum value of the index. In terms of notation, a zeroth-order tensor is a scalar denoted by an italicized lower case character, a, a first-order tensor is a vector denoted by a bold-font lower case character, $\mathbf{a}$, a second-order tensor is a matrix denoted by a bold-font upper case character, $\mathbf{A}$, and a third- or higher ordered-tensor can be visualized as a geometrical cube of scalars. In general, the tensor elements need not be scalars and can actually be other tensors. A tensor is characterized by a set of integral parameters, k_i, referred to as the 'dimension'. The dimension characterizes the maximum number of elements within a given tensor. A 0th-order tensor or scalar has dimension $k = 0$, a first-order tensor or vector has dimension $k = 1$, a matrix, $k = 2$, and so on.

Vectors are expressed as a horizontal array of elements of length k, referred to as 'row vectors' or as a column of elements of length k referred to as 'column vectors.' We denote a column vector as $\mathbf{a}$ and the corresponding row vector as $\mathbf{a}^{\mathrm{T}}$ where the superscript T denotes the transpose operation.

2.2.1 Vector Operations

The operation of addition among two tensors is defined when the tensors have the same order and same dimension. The addition of two vectors $\mathbf{a}$ and $\mathbf{b}$ result in the vector $\mathbf{c} = \mathbf{a} + \mathbf{b}$ where each component of $\mathbf{c}$ is the arithmetic sum of the corresponding components of $\mathbf{a}$ and $\mathbf{b}$. The resulting vector $\mathbf{c}$ has the same dimension as that of $\mathbf{a}$ and $\mathbf{b}$. Another operation is that of 'scaling' which is the operation of multiplying each element of a tensor by a scalar quantity.

Another vector operation is that of norm. Many different definitions of norms are possible and the used here is the euclidean norm denoted as $L_2(\mathbf{a})$ and given in Equation (2.1)

$$L_2(\mathbf{a}) = \|\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^{k} a_i^2}. \tag{2.1}$$

There exist several different definitions for multiplicative operations over vectors including the inner product (dot product) and the outer product (Kronecker or tensor product). The inner product of two k-dimensional vectors **a** and **b** results in a scalar value denoted by $\mathbf{a} \cdot \mathbf{b}$ and is defined in Equation (2.2)

$$\mathbf{a} \bullet \mathbf{b} = \sum_{i=1}^{k} a_i b_i. \tag{2.2}$$

If $\mathbf{a} \cdot \mathbf{b} = 0$, the vectors **a** and **b** are said to be linearly independent. The inner product operator is related to the L_2 norm as $\mathbf{a}^{\mathrm{T}} \cdot \mathbf{a} = [L_2(\mathbf{a})]^2 = [\|\mathbf{a}\|_2]^2$.

The outer product form of multiplication is the tensor, or Kronecker, product operation denoted by $\mathbf{a} \otimes \mathbf{b}$. This form of multiplication can be applied to two tensors of any arbitrary order and if the two argument tensors are of the same order, they need not be of the same dimension. The resultant product is a tensor whose order and dimension is dependent upon those of the multiplicand and multiplier arguments. Unlike the inner product, the tensor product is not commutative, that is, $\mathbf{a} \otimes \mathbf{b} \neq \mathbf{b} \otimes \mathbf{a}$. The tensor product of two column vectors **a** and **b** of dimensions k and h results in a product column vector of dimension kh where each component is formed by scaling **b** with each component of **a** as shown in Equation (2.3)

$$\mathbf{a} \otimes \mathbf{b} = [a_1\mathbf{b}\ a_2\mathbf{b}\ \ldots a_k\mathbf{b}]. \tag{2.3}$$

2.2.2 Matrix Operations

Matrices are denoted as **A**, **B**, **C**, **D**, etc. Two multiplicative operators are the direct product and the tensor or outer product described previously for vectors. Each component of a matrix **A** is denoted by a pair of subscripts *i,j* where i refers to the row location and j the column location. The uppermost element to the left is element a_{11} and the bottommost element to the right is a_{nm} where the matrix dimensions, n and m, are the total number of rows and columns representing **A**. A useful notation for describing the matrix **A** is $\mathbf{A} = [a_{nm}]$ where a_{ij} represents a particular element of **A** in row location $i \leq n$ and column location $j \leq m$. The matrix **A** is said to be 'square' when the number of rows and columns is equal, $n = m$. The transpose of a matrix **A** is written as $\mathbf{A}^{\mathrm{T}}$ and is defined as $\mathbf{A}^{\mathrm{T}} = [a_{ji}]$ where $\mathbf{A} = [a_{ij}]$.

Each column of matrix **A** is referred to as a 'column vector' and the jth column vector is denoted as $\mathbf{a}_j = [a_{1j}\ a_{2j} \ldots a_{nj}]^{\mathrm{T}}$, likewise each row of matrix **A** is a 'row vector' and may be denoted as $\mathbf{a}_i = [a_{i_1} a_{i_2} \ldots a_{i_m}]$. Using the concept of row and column vectors, matrix **A** may be formulated as a row vector whose components are column vectors, $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2 \ldots \mathbf{a}_m]$ or alternatively as a column vector whose components are row vectors, $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2 \ldots \mathbf{a}_n]^{\mathrm{T}}$.

The direct matrix product is denoted by the absence of an operator between the two matrices, **AB**. The direct matrix product is non-commutative, $\mathbf{AB} \neq \mathbf{BA}$. $\mathbf{AB} = [a_{nm}][b_{st}]$ is only defined for the case where $n = t$ and $m = s$. The direct matrix product of **AB** is defined in terms of the inner product of the row vectors of **A**, $\mathbf{a}_i$, and the column vectors of **B**, $\mathbf{b}_j$ as shown in Equation (2.4) or in terms of the outer product of the column vectors of **A** with the row vectors of **B** as given in Equation (2.5).

TABLE 1
Linear Algebra and Bra-Ket Notation

OPERATION	LIN. ALGEBRA	BRA-KET
inner prod.	$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$	$\langle a\|b\rangle = \langle b\|a\rangle$
L_2 norm	$L_2(\mathbf{a}) = (\mathbf{a} \cdot \mathbf{a})^{1/2}$	$\langle a\|a\rangle^{1/2}$
outer prod.	$\mathbf{a} \otimes \mathbf{b}$	$\|a\rangle\langle b\|$
direct prod.	$\mathbf{AB}$	$\mathbf{AB}$
outer prod.	$\mathbf{A} \otimes \mathbf{B}$	$\|\mathbf{A}\rangle\langle\mathbf{B}\|$
vect/matrix prod.	$\mathbf{c} = \mathbf{Ab}$	$\|c\rangle = \mathbf{A}\|b\rangle$
vect/matrix prod.	$\mathbf{c}^{\mathrm{T}} = \mathbf{b}^{\mathrm{T}}\mathbf{A}$	$\langle c\| = \langle b\|\mathbf{A}$

$$\mathbf{AB} = [\mathbf{a}_i^T \bullet \mathbf{b}_j]\Big|_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} \tag{2.4}$$

$$\mathbf{AB} = \sum_{i=1}^{n} \mathbf{a}_i \otimes \mathbf{b}_i. \tag{2.5}$$

The outer product is applicable to matrices and is non-commutative, $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$. The outer product, $\mathbf{C} = \mathbf{A}] \otimes \mathbf{B}$, of two matrices $\mathbf{A} = [a_{nm}]$ and $\mathbf{B} = [b_{st}]$ results in a $ns \times mt$ product matrix $\mathbf{C} = [c_{(ns),(mt)}]$. The outer product can be expressed as a product matrix composed of elements that are scaled matrices as in Equation (2.6)

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}\mathbf{B} & a_{n2}\mathbf{B} & \cdots & a_{nm}\mathbf{B} \end{bmatrix}. \tag{2.6}$$

2.2.3 Dirac Notation

A commonly used vector notation is described in [14]. This notation, called 'bra-ket' notation, was originally formulated as a concise manner for describing the state of a quantum system and to aid in quantum mechanical calculations. Due to the conciseness of bra-ket notation, we use this notation in the formulation of transfer matrices that model digital logic networks in this paper.

A column vector **a** is referred to as 'ket-*a*' and denoted as $|a\rangle$. Likewise, the row vector a^T is referred to as 'bra-*a*' and denoted $\langle a|$. Using bra-ket notation, the multiplicative operations defined above can be expressed as shown in Table 1.

Values within bras and kets may be specified as digit strings enclosed by parentheses and subscripted with the number system radix for clarity. In particular, values specified as binary strings utilize digits $b_i \in \mathbb{B} = \{0, 1\}$ and a corresponding four-bit value may be expressed as $(b_3b_2b_1b_0)_2$ representing the quantity, $VALUE$, where $VALUE$ may be computed using a radix polynomial, $VALUE = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$. As an example, a ket is specified with a four-bit binary string, $(b_3b_2b_1b_0)_2$ is shown in Equation (2.8) and is expanded as an outer product

$$\left|(b_3b_2b_1b_0)_2\right\rangle = |b_3\rangle \otimes |b_2\rangle \otimes |b_1\rangle \otimes |b_0\rangle. \tag{2.8}$$

2.2.4 Vector Spaces

A particular set of first-order tensors that can be added together and also scaled by 0th-order tensors is said to form a 'vector space'. Scaling is the arithmetic multiplication of a given scalar with each element that forms the vector. Because vector addition requires that every first-order tensor in the vector space be composed of the same number of elements, vector spaces are characterized by an integral value called the 'dimension' that specifies the number of elements that comprise a vector.

Definition 1 (Vector Space). *A vector space consists of a set of k-dimensional vectors and the operations of scaling and addition.*

Definition 2 (Hilbert Space). *A vector space defined for an arbitrary dimension k, including an infinite dimension, and that has a norm and inner product associated with it is called a Hilbert space.*

A set of k-dimensional vectors is denoted by $\mathbb{H}^k$. The Hilbert spaces we are interested in here are k-dimensional where k is limited to some finite positive integer and a set of vectors, $\mathbf{v}_i \in \mathbb{H}^k$. Higher ordered Hilbert space spaces of dimension $k \times h$ are formed using the outer product, $\mathbb{H}^{k\times h} = \mathbb{H}^k \otimes \mathbb{H}^h$.

2.2.5 Linear Algebra

A linear algebra is an algebra defined over values that are sets of first-order tensors or vectors with associated operations. Algebras based on elements that are matrices or higher-order tensors are generally referred to as tensor algebras.

In particular, we are interested in the algebra denoted as $\langle \mathbb{H}, \cdot, L_2, \otimes, |0\rangle, |1\rangle\rangle$ with elements $\mathbf{v}_i \in \mathbb{H}^k$ having dimension $k = 2^r$ and $0 \leq r$ where $\mathbb{H}$ represents all vectors contained within an two-dimensional Hilbert space, $\bullet$ is the inner product, L_2 denotes the norm, and the vectors $\{|0\rangle, |1\rangle\} \in \mathbb{H}$ define or 'span' vector space and are defined as

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The basis vectors have particular significance since they, along with the scaling and addition operations allow for any arbitrary vector $\mathbf{v}_i \in \mathbb{H}$ to be formed as shown in Equation (2.9) where α and β are scalars

$$\mathbf{v} = \alpha|0\rangle + \beta|1\rangle. \tag{2.9}$$

Although $\mathbb{H}$ is strictly defined to represent the set of vector elements within a vector space, we shall use $\mathbb{H}$ to also refer to a vector space with the understanding that the set of vector elements have dimension $k = 2$ and that scaling, addition, and a norm is defined. Higher dimensioned vector spaces may be denoted as $\mathbb{H}^i$ where i represents the ith outer product of the vectors within $\mathbb{H}$ as described in Definition 2.

3 Transfer Functions in $\mathbb{H}$ for Digital Networks

Switching circuit models for digital networks are based upon an algebraic framework such as $\langle \mathbb{B}, +, \cdot, ', 0, 1\rangle$. An alternative algebraic framework for modeling digital networks is based upon the algebra $\langle \mathbb{H}, \cdot, L_2, \otimes, |0\rangle, |1\rangle\rangle$. Within the algebra $\langle \mathbb{B}, +, \cdot, ', 0, 1\rangle$, individual logic gates with n inputs and a single output are characterized by switching functions of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}$, here we model logic gates

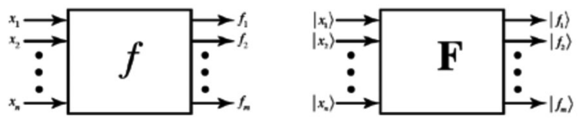


Fig. 1. Digital network models.

as a function $\mathbf{F}:\mathbb{H}^n \rightarrow \mathbb{H}$, where $\mathbf{F}$ is a second order tensor (or matrix) that maps vectors from the domain $\mathbb{H}^n$ to the range $\mathbb{H}$. Depending upon the specified interconnections of the logic gates, an overall transfer function for the network can be derived, $\mathbf{F}_{circ}$. In general, a logic network may depend upon n inputs resulting in m outputs and the overall transfer function model for the network is of the form $\mathbf{F}:\mathbb{H}^n \rightarrow \mathbb{H}^m$. Fig. 1 depicts the switching model and the corresponding transfer function model for a logic network characterized as a function f in the switching theoretical framework and as $\mathbf{F}$ in the transfer function framework.

The circuit inputs at the left side of Fig. 1 are denoted by n Boolean variables, $x_i \in \mathbb{B}$ and the corresponding outputs are denoted by $f_i \in \mathbb{B}$. In the transfer function model at the right of Fig. 1, the inputs are denoted by an n-dimensional vector, $|x_i\rangle \in \mathbb{H}^n$ and the outputs by a vector $|f_i\rangle \in \mathbb{H}^n$. The functional behavior of the circuit is represented by the switching function $f(x_1, \ldots, x_n)$ and the $n \times m$ transformation matrix $\mathbf{F}$ that serves as the specification of the network transfer function.

One of the most common values used to augment $\mathbb{B}$ is that of the unspecified or unknown value usually denoted as **X**. Other common values are **Z**, used to represent a high-impedance or open circuit value. If **X** and **Z** are used, $\mathbb{B}^+ = \{\mathbb{B}, \mathbf{X}, \mathbf{Z}\} = \{0, 1, \mathbf{X}, \mathbf{Z}\}$. **X** is a value introduced for the purpose of modeling and analysis only since it does not represent a model for a physical quantity such as the values 0, 1, and **Z**. In order to ensure interoperability among EDA software produced by different vendors, different definitions of $\mathbb{B}^+$and the electrical interpretation of their elements have been standardized [15], [16].

3.1 Constants Modeled as Elements of $\mathbb{H}$

In the linear algebraic model presented here, we model constants as elements of $\mathbb{H}$. We model the constant element $|0\rangle \in \mathbb{H}$ as $|0\rangle = [1, 0]^{\mathrm{T}}$ and $|1\rangle = [0, 1]^{\mathrm{T}}$. We also amend the elements of $\mathbb{H}$ resulting in $\mathbb{H}^+ = \{\mathbb{H}, \varnothing\rangle, |t\rangle\}$. The additional elements $|\varnothing\rangle$ and $|t$ are not related to $\mathrm{X}, \mathrm{Z} \in \mathbb{B}^+$, rather they are included for the purpose of developing the transfer function model for a logic network and their inclusion is convenient in analysis of the modeled network. Qualitatively, the element $|\varnothing\rangle$ can be considered as the 'absence' of either $|0\rangle$ or $|1\rangle$, while the element $|t\rangle$ represents an element that is simultaneously both $|0\rangle$ and $|1\rangle$, $|t\rangle = |0\rangle + |1\rangle$. In using $\mathbb{H}^+$, a lattice algebra results that can be denoted by a Hasse diagram where values closer to the upper portion of the graph cover or contain values that appear closer to the bottom of the graph [12]. Fig. 2 contains the Hasse diagram for the elements of the set $\mathbb{H}^+$.

3.2 Elements as Transfer Functions Over $\mathbb{H}^+$

Within the switching theory framework, two-place operators are represented by functions that map the domain set of elements $\mathbb{B}^2 = \{00, 01, 10, 11\}$ to a range set $\mathbb{B}$. The particular mapping defines the function and couples each

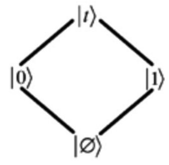


Fig. 2. Hasse diagram for $\mathbb{H}^+$.

unique element of the domain with a corresponding range element. Primitive two-input, single output logic gates are modeled as functions, $f : \mathbb{B}^2 \rightarrow \mathbb{B}$. The same concept is applied for transfer function models of primitive logic gates where $\mathbf{F} : \mathbb{H}^2 \rightarrow \mathbb{H}$ represents the gate whose behavior is specified by the matrix $\mathbf{F}$. To illustrate this concept, Fig. 3 contains two diagrams where the leftmost diagram depicts the mapping function of an exclusive-OR (XOR) gate using elements of $\mathbb{B}$ while that on the right shows the corresponding diagram using elements from $\mathbb{H}$.

The domain space $\mathbb{H}^2$ consists of a collection of four vectors, $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. As an example, consider the case of $|10\rangle$ in the domain space. By application of Equation (2.8), $|10\rangle = |1\rangle \otimes |0\rangle = [0,1]^{\mathrm{T}} \otimes [1,0]^{\mathrm{T}} = [0,0,1,0]^{\mathrm{T}}$.

A transfer function expresses an input-output relationship in a manner such that when it is multiplied by a specific input stimulus, the corresponding output response results. Lemma 1 contains a property that is used in the derivation of the form of a digital logic network transfer function.

Lemma 1. *Vectors $|x_i\rangle \in H^n$ representing logic network input stimuli are linearly independent.*

Proof. Consider the vectors $\mathbf{x}_1 = \left|(b_n b_{n-1} \dots b_2 b_1)_2\right\rangle \in \mathbb{H}^n$ and $\mathbf{x}_2 = \left|(b_n b_{n-1} \dots b_2 b_1)_2\right\rangle \in \mathbb{H}^n$ where $\mathbf{x}_1 \neq \mathbf{x}_2$. From Equation (2.8), $\mathbf{x}_i = |b_n\rangle \otimes |b_n{-}1\rangle \otimes \cdots \otimes |b_1\rangle$ which is of the form of a vector whose components are all zero-valued except for a single element that has value 1 at index $2^i{-}1$. Furthermore, $L_2(\mathbf{x}_i) = 1$. Since $\mathbf{x}_1 \neq \mathbf{x}_2$, each vector must have a unity-valued element at a different index location. These observations indicate that $\langle x_1|x_2\rangle = 0$. By definition, two vectors with a non-zero norm and whose inner product is zero are linearly independent. □

Lemma 2. *The output response of a logic network can be obtained by multiplying the outer product $|x_i\rangle\langle f_i|$ with the transpose of $|x_i\rangle$ where $|x_i\rangle$ represents a specific network stimulus input and $|f_i\rangle$ represents the corresponding network output response.*

Proof.

$$(|x_i\rangle)^{\mathrm{T}}(|x_i\rangle\langle f_i|) = \langle x_i|x_i\rangle\langle f_i|$$

From the proof of Lemma 1, it is observed that $|x_i\rangle$ is of the form of a vector whose components are all zero-valued except for a single element that has value 1 at index 2^{i-1}, thus $\langle x_i|x_i\rangle = 1$.

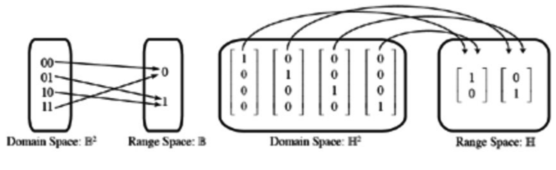

00
01
10
11
0
1
Domain Space: $\mathbb{B}^2$
Range Space: $\mathbb{B}$
Domain Space: $\mathbb{H}^2$
Range Space: $\mathbb{H}$

Fig. 3. Function mappings for XOR operator.

TABLE 2
XOR Truth Table Using Elements of $\mathbb{H}$

a	b	$a \oplus b$	a	b	$a \oplus b$	$\langle ab\| = \langle a\| \otimes \langle b\|$	$a \oplus b$
$\langle 0\|$	$\langle 0\|$	$\langle 0\|$	[1 0]	[1 0]	[1 0]	[1 0 0 0]	[1 0]
$\langle 0\|$	$\langle 1\|$	$\langle 1\|$	[1 0]	[0 1]	[0 1]	[0 1 0 0]	[0 1]
$\langle 1\|$	$\langle 0\|$	$\langle 1\|$	[0 1]	[1 0]	[0 1]	[0 0 1 0]	[0 1]
$\langle 1\|$	$\langle 1\|$	$\langle 0\|$	[0 1]	[0 1]	[1 0]	[0 0 0 1]	[1 0]

$$\langle x_i|x_i\rangle\langle f_i| = (1)\langle f_i|.$$

□

Theorem 1. *The transfer function, $\mathbf{T}$, representing the input-output relationship of a logic network, F, is of the form in Equation (3.3)*

$$\mathbf{T} = \sum_{i=1}^{2^n} |x_i\rangle\langle f_i| \tag{3.3}$$

Proof. From Lemma 1,

$$\langle x_i|x_i\rangle = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

thus, $\langle x_i|\mathbf{T} = \langle f_i|$. □

The form of $\mathbf{T}$ in Equation (3.3) is of an $n \times m$ matrix where $\log_2(n)$ is the number of logic network inputs and $\log_2(m)$ is the number of network outputs. Since the transfer function is in the form of a matrix, we interchangeably use the terms 'transfer function' and 'transfer matrix.' As an example of the application of Theorem 1, we calculate the transfer matrix of the XOR gate modeled as a transfer function $\mathbf{X} : \mathbb{H}^2 \rightarrow \mathbb{H}$.

Example 1.

$$\mathbf{X} = |00\rangle\langle 0| + |01\rangle\langle 1| + |10\rangle\langle 1| + |11\rangle\langle 0|$$

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

The resulting transfer function matrix $\mathbf{X}$ is isomorphic to the truth table representation of an XOR gate as shown in Table 2. This observation implies that commonly used methods for truth table representations of logic functions may be employed to represent the transfer matrix such as BDDs.

Fig. 4 contains common logic gate diagrams and their associated transfer matrices.

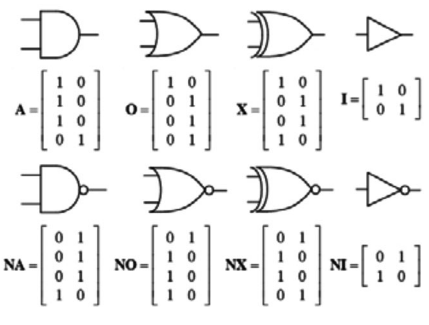


Fig. 4. Common logic gates and transfer matrices.

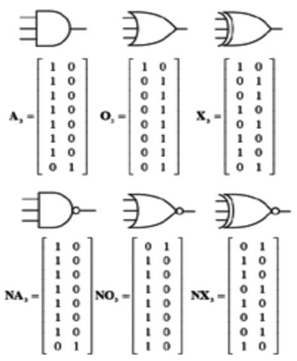


Fig. 5. Common three-input logic gates and transfer matrices.

Negated output logic gates have transfer matrix characterizations that may be directly computed using the relationship in Equation (3.6) where **1** denotes the matrix whose elements are all 1

$$\mathbf{NT} = \mathbf{1} - \mathbf{T}. \tag{3.6}$$

Fig. 5 depicts several common three-input logic gates and their associated transfer matrices.

For the purposes of analysis, it is necessary to represent a conducting path, a fanout node, and a fanin node in terms of transfer matrices. Fig. 6 depicts these circuit structures and their associated transfer matrices.

3.3 Logic Network Transfer Matrices

The construction of a transfer function for a logic network can be accomplished through the application of Equation (3.3) when the input and output responses are all known. This is typically the case for applications such as logic network synthesis where the desired network behavior is known but a detailed low-level network is not yet available.

Analysis tasks generally involve having access to the structure of a logic network but with the overall input/output response unknown. Deriving the transfer matrix using Equation (3.3) is exponentially complex since it involves the determination of 2^n terms through the use of a simulation tool or some other means. Fortunately, the transfer matrix of a given logic network can be determined through the use of transfer matrices of individual network elements and their corresponding interconnections.

3.4 Transfer Matrix from Switching Specification

Consider the example logic network shown in Fig. 7. A switching specification of this circuit is given in Table 3 in the form of a truth table.

The corresponding transfer matrix, **T**, can be computed using the relationship in Equation (3.3). However, as illustrated in Table 2, the transfer function has a structure that consists of row vectors identical to the truth table output vectors expressed as elements of $\mathbb{H}$. Using this observation,

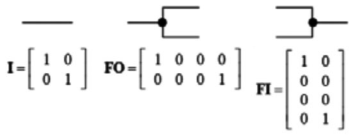


Fig. 6. Common circuit structures and transfer matrices.

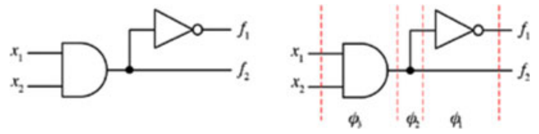


Fig. 7. Example logic network and partitions.

the **T** can be directly written from the truth table specification in Table 3. Example 2 illustrates how **T** can be derived using both approaches

Example 2.

$$\mathbf{T} = \begin{bmatrix} \langle 10| \\ \langle 10| \\ \langle 10| \\ \langle 01| \end{bmatrix} = \begin{bmatrix} \langle 1| \otimes \langle 0| \\ \langle 1| \otimes \langle 0| \\ \langle 1| \otimes \langle 0| \\ \langle 0| \otimes \langle 1| \end{bmatrix} = \begin{bmatrix} [0 \quad 1] \otimes [1 \quad 0] \\ [0 \quad 1] \otimes [1 \quad 0] \\ [0 \quad 1] \otimes [1 \quad 0] \\ [1 \quad 0] \otimes [0 \quad 1] \end{bmatrix}$$
$$= \begin{bmatrix} [0 \quad 0 \quad 1 \quad 0] \\ [0 \quad 0 \quad 1 \quad 0] \\ [0 \quad 0 \quad 1 \quad 0] \\ [0 \quad 1 \quad 0 \quad 0] \end{bmatrix}.$$

$$\begin{aligned} \mathbf{T} &= \sum_{i=0}^{2^n-1} |i\rangle\langle f_i| = \sum_{i=0}^{3} |i\rangle\langle f_i| \\ &= |0\rangle\langle 2| + |1\rangle\langle 2| + |2\rangle\langle 2| + |3\rangle\langle 1| \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{3.11}$$

3.5 Transfer Matrix from Logic Network

The technique for deriving a transfer matrix directly from a logic network involves partitioning the network into a series or cascade of subcircuits, computing the transfer matrix for each partition, and then computing the overall transfer matrix as the direct matrix product of each cascade stage transfer matrix. The individual cascades are determined by forming partition cuts such that all components within the cascade stage are in a parallel arrangement. Fig. 7 shows the example circuit with partitions indicated by dashed lines.

The partitioning process results in three partitions labeled ϕ_1, ϕ_2, and ϕ_3. Fanout and fanin points are treated as network elements since these structures have differing numbers of inputs and outputs. The process of partitioning is easily implemented by levelizing the circuit

The next step is to compute the transfer matrices for each partition. Because each partition is composed of a set of parallel elements, the signals on each parallel line must be

TABLE 3
Truth Table of Example Circuit

x_1	x_2	f_1	f_2
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

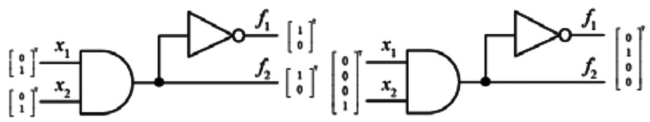


Fig. 8. Example network input and output.

combined into a single element in $\mathbb{H}^w$ where $\log_2(w)$ denotes the number of parallel network signals in a partition. The expansion is performed using the outer product operation among each parallel line.

During the process of partitioning, signals that pass through a stage unmodified are modeled with the identity transformation matrix, $\mathbf{I}_2$. The identity matrix $\mathbf{I}_2$ also serves as the transfer matrix for a non-inverting buffer since both elements pass signals through the stage unaltered.

Partition ϕ_1 consists of two signal paths with the uppermost path through an inverter and the bottom path through a conductor. The transfer matrix, $\mathbf{T}_{\phi_1}$, is computed as shown in Equation (3.12)

$$\mathbf{T}_{\phi_1} = \mathbf{NI} \otimes \mathbf{I} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.12}$$

The transfer matrix for partition ϕ_3 is obtained from Fig. 4 resulting in $\mathbf{T}_{\phi_3} = \mathbf{A}$ and, for partition ϕ_2, $\mathbf{T}_{\phi_2} = \mathbf{FO}$.

The overall logic network transfer matrix, $\mathbf{T}$, is formed through the use of the direct matrix product

$$\begin{aligned}\mathbf{T} &= \mathbf{T}_{\phi_3}\mathbf{T}_{\phi_2}\mathbf{T}_{\phi_1} = (\mathbf{A})(\mathbf{FO})(\mathbf{NI} \otimes \mathbf{I}). \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.\end{aligned} \tag{3.13}$$

Both the outer and direct matrix product operations are non-commutative, thus a convention is followed as to the order of the multiplicands when performing the multiplication operation. Here we use the convention of combining parallel gates in given cascade stages by using the topmost gate matrix as the leftmost operand. For the direct product, we use the stage closest to the circuit input as the leftmost multiplicand.

4 Digital Network Analysis

Simulation and implication are formulated within the context of the linear algebraic circuit model.

4.1 Output Response

The output response may be determined through a vector-matrix direct product as shown in Theorem 2.

Theorem 2. *The output response of a logic network stimulated by input $\langle x_q|$ and modeled by transfer matrix $\mathbf{T}$ is denoted by $\langle f_q|$ and is computed using Equation (4.1)*

$$\langle f_q| = \langle x_q|\mathbf{T}. \tag{4.1}$$

Proof. Multiplying Equation (3.2)-2 by $\langle x_q|$,

$$\langle x_q|\mathbf{T} = \langle x_q| \left. \sum_{i=1}^{2^n} |x_i\rangle\langle f_i| \right)$$

$$\langle x_q|\mathbf{T} = \sum_{i=1}^{2^n} \langle x_q|x_i\rangle\langle f_i|.$$

from Lemma 1,

$$\langle x_i|x_j\rangle = \begin{cases} 0, & i \neq j \\ 1, & i = j. \end{cases}$$

therefore

$$\langle x_q|\mathbf{T} = \langle f_q|.$$

□

The individual input signals specified as vectors $\langle x_i| \in \mathbb{H}$ are combined using the relationship in Equation (2.8) to form the vector $\langle x|$. Fig. 8 depicts the example logic network in Fig. 7 with an input stimulus specified as signals $\langle x_i| \in^2$ on the left and the corresponding network when the input stimulus is specified as the single vector $\langle x| \in \mathbb{H}^2$.

Employing Equation (4.1), the output response $\langle f|$ is calculated in Equation (4.3)

$$\langle f| = \langle x|\mathbf{T} = [\,0 \quad 1 \quad 0 \quad 0\,]. \tag{4.3}$$

The output response can be decomposed to determine individual output elements. This is easily accomplished through expressing the output response in terms of a bra and then converting the value to a binary string. In the case of the network in Fig. 8, the output response is $\langle f| = [\,0 \quad 1 \quad 0 \quad 0\,] = \langle (1)_{10}| = \langle (01)_2| = \langle 0| \otimes \langle 1|$. Thus, in keeping with the convention of combining outer products from top to bottom, we have $\langle f| = \langle f_1| \otimes f_2| = \langle 0| \otimes \langle 1|$ yielding $\langle f_1| = \langle 0|$ and $\langle f_2| = \langle 1|$.

It is advantageous to model the input signals as elements $\langle x_i| \in \mathbb{H}^+ = \{|0\rangle, |1\rangle, |\emptyset\rangle, |t\rangle\}$ rather than restricting them to $\langle x_i| \in \mathbb{H}$ since an output response vector corresponding to more than one stimuli can be obtained through a single multiplication.

As an example, we use the input stimulus vector $\langle x| = \langle tt| = \langle t| \otimes \langle t| = [\,1 \quad 1 \quad 1 \quad 1\,]$. The corresponding output response is calculated in Equation (4.4)

$$\langle f| = \langle tt|\mathbf{T} = [\,0 \quad 1 \quad 3 \quad 0\,]. \tag{4.4}$$

The resulting output response is decomposed as $[\,0\ 1\ 3\ 0\,] = [\,0 \quad 1 \quad 0 \quad 0\,] + 3[\,0 \quad 0 \quad 1 \quad 0\,] = \langle (1)_{10}| + 3\langle (2)_{10}| = \langle (01)_2| + 3\langle (10)_2|$. Thus, for all possible inputs, only two distinct outputs occur; either $\langle f_1| = \langle 0|$ and $\langle f_2| = \langle 1|$ occur once, or, $\langle f_1| = \langle 1|$ and $\langle f_2| = \langle 0|$ occur for three different input values. The results also indicate the case $\langle f_1| = \langle 0|$ and $\langle f_2| = \langle 0|$, or, $\langle f_1| = \langle 1|$ and $\langle f_2| = \langle 1|$ can never occur.

The technique can also be used with only a subset of the inputs set to value $\langle t|$. As an example of determining the output response for all values of input $\langle x_1|$ but restricting $\langle x_2|\langle 0|$ in a single calculation, consider Equation (4.5)

$$\langle f| = \langle t0|\mathbf{T} = [\,0 \quad 0 \quad 2 \quad 0\,]. \tag{4.5}$$

The obtained output response is decomposed as $[0 \;\; 0 \;\; 2 \;\; 0] = 2[0 \;\; 0 \;\; 1 \;\; 0] = 2\langle(2)_{10}| = 2\langle(10)_2|$ indicating that for all possible input values of $\langle x_1|$ while restricting $\langle x_2|\langle 0|$, the only possible output that can occur is $\langle f_1 f_2| = \langle 10|$.

The use of $\langle t|$ as an input value is an advantage of the linear algebraic method of analysis since it allows multiple input-output analysis pairs to be formulated in a single calculation. Applications of this principle for formal verification and simulation are described in [10], [11] where more cumbersome symbolic methods are used.

4.2 Implication

Implication is the inverse problem of simulation. In this case, an output response and the characterization of a logic network are known and it is desired to compute the corresponding input stimuli. To formulate a method for implication within the linear algebraic framework, we solve Equation (4.1) for the input vector $\langle x|$ resulting in Equation (4.6)

$$\langle x| = \langle f|\mathbf{T}^{-1}. \tag{4.6}$$

Equation (4.2)-1 is easily employed when the transfer matrix **T** is square and full rank. When this special case occurs, the logic network is said to be logically 'reversible'. Important classes of logic networks exhibit reversibility including classical logic technologies [17], [18] and emerging technologies such as quantum logic [6].

It is generally the case that the transfer matrices for classical digital logic networks are non-square, and when the matrices are square, they may not necessarily be invertible or of full rank as is the case for the transfer matrix representing the example circuit in Fig. 7. The **T** matrix in this case is square; however, it is not of full rank since the first and fourth column vectors have a norm of zero. This fact does not affect output response calculations but it does have ramifications in employing the relationship in Equation (4.6) for implication computations.

There are numerous techniques available for the general solution of Equation (4.6) when the matrix **T** is non-square or when it is not of full rank based on the use of generalized inverses [19]. In [9], the Moore-Penrose pseudoinverse is described as a means for computing $\mathbf{T}^{-1}$ so that the implication relation in Equation (4.6) may be used. There are three situations for determining $\mathbf{T}^{-1}$.

1) **T** is square and of full rank: A unique solution is possible and the logic network is reversible.
2) **T** has fewer linearly independent row vectors than columns: A unique solution is not possible since the system is under-specified.
3) **T** has more linearly independent row vectors than columns: The system is over-specified and has multiple solutions.

For case 2), the pseudoinverse provides a best-fit solution based upon an L_2 norm for error measurement and for case 3), a solution is provided that has a minimal error in the sense of the L_2 norm. The pseudoinverse is thus unique and exists for either case 2) or 3).

The pseudoinverse of a matrix **T** is denoted as $\mathbf{T}^+$ and may be computed in two ways depending on whether case 2) or case 3) arises. The notation $\mathbf{T}^*$ indicates the Hermitian transpose of **T**. The formulas for computation of the pseudoinverse are provided in Equations (4.7) and (4.8)

$$\mathbf{T}^+ = (\mathbf{T}^*\mathbf{T})^{-1}\mathbf{T}^* \tag{4.7}$$

$$\mathbf{T}^+ = \mathbf{T}^*(\mathbf{T}^*\mathbf{T})^{-1}. \tag{4.8}$$

In the case of digital logic networks, the elements of **T** are always real, thus the Hermitian transpose is simply the transpose of **T**, $\mathbf{T}^* = \mathbf{T}^{\mathrm{T}}$. Substituting this observation into Equations (4.7) and (4.8) yields the following:

$$\mathbf{T}^+ = \left(\mathbf{T}^{\mathrm{T}}\mathbf{T}\right)^{-1}\mathbf{T}^{\mathrm{T}} \tag{4.9}$$

$$\mathbf{T}^+ = \mathbf{T}^{\mathrm{T}}\left(\mathbf{T}^{\mathrm{T}}\mathbf{T}\right)^{-1}. \tag{4.10}$$

Employing Equations (4.9) and (4.10) leads to the following Lemma.

Lemma 3. *The pseudoinverse of the transfer matrix of a single-output logic network is proportional to its transpose with a proportionality constant* ***P*** *where* ***P*** *is a* 2×2 *diagonal matrix.*

Proof. n-input, single-output logic gate transfer matrices where $n \geq 2$ are non-square and are comprised of two column vectors and 2^n row vectors. Furthermore, the two column vectors t_a and t_b are always related as $\mathbf{1} - \mathbf{c}_2 = \mathbf{c}_1$. Using this observation,

$$\mathbf{T} = \begin{bmatrix} t_{a1} & t_{b1} \\ t_{a2} & t_{b2} \\ \vdots & \vdots \\ t_{a2^n} & t_{b2^n} \end{bmatrix} = \begin{bmatrix} 1 - t_{b1} & t_{b1} \\ 1 - t_{b2} & t_{b2} \\ \vdots & \vdots \\ 1 - t_{b2^n} & t_{b2^n} \end{bmatrix}.$$

Computing the term $\mathbf{T}^{\mathrm{T}}\mathbf{T}$ in Equations (4.2)-(4.4) and (4.2)-(4.5) results in,

$$\mathbf{T}^{\mathrm{T}}\mathbf{T} = \mathbf{P} = \begin{bmatrix} 1 - t_{b1} & 1 - t_{b2} & \cdots & 1 - t_{b2^n} \\ t_{b1} & t_{b2} & \cdots & t_{b2^n} \end{bmatrix} \begin{bmatrix} 1 - t_{b1} & t_{b1} \\ 1 - t_{b2} & t_{b2} \\ \vdots & \vdots \\ 1 - t_{b2^n} & t_{b2^n} \end{bmatrix}$$
$$= \begin{bmatrix} 2^n - \langle t_b|t_b\rangle & 0 \\ 0 & \langle t_b|t_b\rangle \end{bmatrix} = \begin{bmatrix} 2^n - N_{\min} & 0 \\ 0 & N_{\min} \end{bmatrix},$$

where $N_{\min}$ denotes the number of minterms for the logic gate. Using this result with Equations (4.2)-(4.4) results in

$$\mathbf{T}^+ = \left(\mathbf{T}^{\mathrm{T}}\mathbf{T}\right)^{-1}\mathbf{T}^{\mathrm{T}} = \begin{bmatrix} \frac{1}{2^n - N_{\min}} & 0 \\ 0 & \frac{1}{N_{\min}} \end{bmatrix}\mathbf{T}^{\mathrm{T}} = \mathbf{P}\mathbf{T}^{\mathrm{T}}.$$

□

The number of minterms $N_{\min}$ is a characterizing constant that is well known for all basic logic gates. Hence, basic logic gates and their pseudoinverses are easily obtained and given in Fig. 9. The buffer and single pass-through conductor both have self-inverses since their transfer matrix is **I**. The fanout and fanin transfer matrices are pseudoinverses of each other, $\mathbf{FO}^+ = \mathbf{FI}$.

As is the case for construction of **T** for a given logic network, the same procedure involving partitioning, outer products within each partition, and direct products of each stage can be employed to compute the overall $\mathbf{T}^+$. To

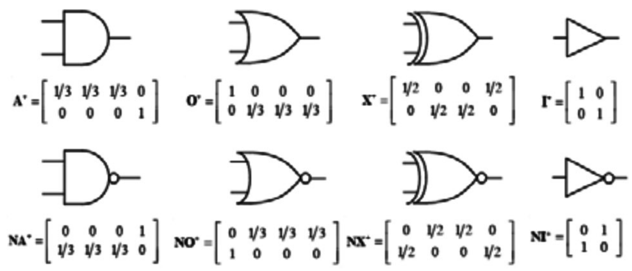

$\mathbf{A}^+ = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$\mathbf{O}^+ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \end{bmatrix}$
$\mathbf{X}^+ = \begin{bmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 1/2 & 0 \end{bmatrix}$
$\mathbf{I}^+ = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$\mathbf{NA}^+ = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$
$\mathbf{NO}^+ = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
$\mathbf{NX}^+ = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 & 1/2 \end{bmatrix}$
$\mathbf{NI}^+ = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Fig. 9. Basic logic gate pseudoinverses.

illustrate this procedure, we compute the implication matrix $\mathbf{T}^+$ for the example logic network in Fig. 7.

Using the construction method in conjunction with the result of Lemma 3, pseudoinverses can be computed for entire logic networks without resorting to the use of more computationally complex numerical linear algebra algorithms.

Example 3. As an example of the use of pseudoinverses, consider the logic network depicted in Fig. 7. Table 4 is used to obtain basic $\mathbf{T}^+$ matrices

$$\mathbf{T}^+_{\phi_1} = \mathbf{A}^+ = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}^+_{\phi_2} = \mathbf{FO}^+ = \mathbf{FI} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{T}^+_{\phi_3} = \mathbf{NI}^+ \otimes \mathbf{I}^+ = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{T}^+ = \mathbf{T}^+_{\phi_3}\mathbf{T}^+_{\phi_2}\mathbf{T}^+_{\phi_1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Equation (4.6) is used to compute the implication of the logic network. The following example contains implication calculations for each of the cases $\langle f_1 f_2| = \langle 00|$, $\langle f_1 f_2| = \langle 01|$, $\langle f_1 f_2| = \langle 10|$, and $\langle f_1 f_2| = \langle 11|$.

Example 4.

$$\langle 00|\mathbf{T}^+ = [1 \quad 0 \quad 0 \quad 0]\mathbf{T}^+ = [0 \quad 0 \quad 0 \quad 0] = \langle \emptyset\emptyset|$$

$$\langle 01|\mathbf{T}^+ = [0 \quad 1 \quad 0 \quad 0]\mathbf{T}^+ = [0 \quad 0 \quad 0 \quad 1] = \langle 11|$$

$$\langle 10|\mathbf{T}^+ = [0 \quad 0 \quad 1 \quad 0]\mathbf{T}^+ = [1/3 \quad 1/3 \quad 1/3 \quad 0]$$

$$\langle 11|\mathbf{T}^+ = [0 \quad 0 \quad 0 \quad 1]\mathbf{T}^+ = [0 \quad 0 \quad 0 \quad 0] = \langle \emptyset\emptyset|.$$

The results of the implication calculations indicate that it is not possible for the example circuit to ever have output values $\langle f_1 f_2| = \langle 00|$ or $\langle f_1 f_2| = \langle 11|$ since the implication is that both input values are $\langle x_1 x_2| = \langle \emptyset\emptyset|$. The qualitative

TABLE 4
Results Using Explicit Matrix Structures

NAME	IN/OUT	STAGES	PARTITION TIME (ms)	MATRIX TIME (ms)
i3	2/3	3	3.00	5.505
test1	3/3	6	7.28	4.794
xor5	5/1	4	1.73	6.882
majority	5/1	6	11.8	17.71
C17	5/2	7	5.00	22.75
rd53	5/3	6	5.32	10.10
squar5	5/8	9	19.5	922.1
con1	7/2	6	7.09	546.1
rd73	7/3	8	5.01	37.33
radd	8/5	11	12.3	1107
x2	10/7	9	11.4	846.2
cm85a	11/3	11	9.78	1,586.2
alu1	12/8	5	8.88	521.9

interpretation of $\langle \emptyset|$ is a signal that is neither $\langle 0|$ nor $\langle 1|$ which is clearly a physical impossibility for conventional digital electronic circuitry. The implication results also indicate that the logic network response $\langle f_1 f_2| = \langle 01|$ can only occur if $\langle x_1 x_2| = \langle 11|$; however $\langle f_1 f_2| = \langle 10|$ can occur whenever $\langle x_1 x_2| = \langle 00|$, $\langle x_1 x_2| = \langle 01|$, or $\langle x_1 x_2| = \langle 10|$.

As is the case with output response calculations, the value $\langle t|$ may be used in implication calculations. Consider the example where we first search for input stimuli such that $\langle f_2| = \langle 1|$ and $\langle f_1|$ is allowed to be either $\langle 0|$ or $\langle 1|$ followed by the case where $\langle f_2| = \langle 0|$ and $\langle f_1|$ is allowed to be either $\langle 0|$ or $\langle 1|$. This condition is imposed by using $\langle f_1| = \langle t|$ in both implication calculations.

Example 5:

$$\begin{aligned}\langle t0|\mathbf{T}^+ &= [1 \quad 0 \quad 1 \quad 0]\mathbf{T}^+ = [1/3 \quad 1/3 \quad 1/3 \quad 0] \\ &= 1/3\langle 00| + 1/3\langle 01| + 1/3\langle 10|\end{aligned}$$

$$\langle t1|\mathbf{T}^+ = [0 \quad 1 \quad 0 \quad 1]\mathbf{T}^+ = [0 \quad 0 \quad 0 \quad 1] = \langle 11|.$$

The pseudoinverse matrices for a logic network have the same form as the transpose of the transfer matrix with an additional row vector constant scaling value included. The scaling value is a function of the number of minterms of the overall logic network. To further ease the computational task in performing implication calculations, we define the 'implication matrix' to be equivalent to the transpose of the transfer matrix.

Definition 6 (Implication Matrix). *The implication matrix,* $\mathbf{T}^{\mathrm{I}}$, *is defined to be the transpose of the transfer matrix,* $\mathbf{T}^{\mathrm{I}} = \mathbf{T}^{\mathrm{T}}$.

The use of $\mathbf{T}^{\mathrm{I}}$ in performing implication analyses allows for determination of $N_{\min}$ to be avoided. Furthermore, after the transfer matrix of a logic network has been obtained, no further preprocessing is required to perform implication analysis when $\mathbf{T}^{\mathrm{I}}$ is used. Consider the implication calculations in Example 4, where $\mathbf{T}^{\mathrm{I}}$ is used in place of the pseudoinverse, $\mathbf{T}^+$.

Example 7:

$$\langle 00|\mathbf{T}^{\mathrm{I}} = [1 \quad 0 \quad 0 \quad 0]\mathbf{T}^{\mathrm{I}} = [0 \quad 0 \quad 0 \quad 0] = \langle \emptyset\emptyset|$$

$$\langle 01|\mathbf{T}^{\mathrm{I}} = [0 \quad 1 \quad 0 \quad 0]\mathbf{T}^{\mathrm{I}} = [0 \quad 0 \quad 0 \quad 1] = \langle 11|$$

$$\langle 10|\mathbf{T}^{\mathrm{I}} = [0 \quad 0 \quad 1 \quad 0]\mathbf{T}^{\mathrm{I}} = [1 \quad 1 \quad 1 \quad 0] \\ = \langle 00| + \langle 01| + \langle 10|$$

$$\langle 11|\mathbf{T}^{\mathrm{I}} = [0 \quad 0 \quad 0 \quad 1]\mathbf{T}^{\mathrm{I}} = [0 \quad 0 \quad 0 \quad 0] = \langle \emptyset\emptyset|.$$

Implication calculations require the same amount of computational complexity as output response determination when the linear algebraic model is used for logic networks.

5 Implementation and Results

To demonstrate the method, a set of benchmark circuits are used. The two-level benchmark netlists are converted into multi-level combinational logic circuits in the form a Verilog structural netlist before the technique is applied to them. This initial conversion process is accomplished by converting the native **.pla** files into corresponding Verilog files using Synopsys Design Compiler. The converted files are then in the form of a set of two-level Boolean equations expressed in Verilog syntax. To convert these into multi-level structural circuits, the Design Compiler tool is next used to synthesize the two-level form with a simple cell library consisting of basic logic gates. The multi-level netlists are then saved as structural Verilog descriptions and used as input to the prototype program that computes the corresponding transfer matrix for each circuit.

The prototype we developed first performs a levelization operation by parsing the topological Verilog descriptions and assigning a level number to each net as is commonly accomplished in netlist simulation algorithms. Serial partitions are identified by finding cuts through all nets that have identical levelization indices. This process requires two passes through the netlist and is thus of temporal complexity $O(N)$ where N is the number of nets. The spatial complexity is also $O(N)$ as the structural Verilog netlist is parsed into an internal graph memory structure where nodes represent gates and primary circuit inputs and outputs. Graph edges correspond to the topological nets in the circuit.

After partitioning has occurred, the overall transfer matrix is computed by forming partition transfer matrices and then multiplying them. To reduce memory requirements for the intermediate matrix computations, we initialize the overall transfer matrix to the transfer matrix for the leftmost partition. Next, the transfer matrix for the second partition is computed and then multiplied with the overall transfer matrix structure. This process repeats by subsequently forming the next intermediate partition transfer matrix and updating the overall matrix structure through a multiplication. In this manner, it is only necessary to utilize memory for two transfers matrices at any instant in time, the overall circuit transfer matrix structure and the intermediate partition transfer matrix that is being formed. In the final step of the process, the transfer matrix for the rightmost partition is formed and multiplied with the overall transfer matrix structure that represents the direct product of all partition matrices except the rightmost one. After this final multiplication, the overall transfer matrix is fully formed.

TABLE 5
Results Using BDDs for Matrix Structures

NAME	IN/OUT	MATRIX SIZE (KB)	MATRIX TIME (ms)
C880	60/26	18513.56	60
C1355	41/32	2876.28	50
C1908	33/25	1193.72	90
C3540	50/22	40409.64	3940
apex7	49/37	26.39	133.1
dalu	75/16	51064.39	563.7
x4	94/71	79.84	8.264
apex5	117/88	42.41	296.5
ex4	128/28	143.72	113.9
frg2	143/139	102.63	306.9
i2	201/1	8.36	4.647

In terms of complexity, each partition transfer matrix requires W outer product operations where W is the width or number of parallel elements within a partition. A total of L direct matrix products are performed where L is the length or number of partitions identified in the levelization procedure. In addition to the storage required for the graphical form of the topological netlist, the method also requires storage for at most two transfer matrices; one for the intermediate partition transfer matrix being processed at the current time and another to hold the partially formed overall circuit transfer matrix. By forming the overall transfer matrix iteratively in processing each partition sequentially, we reduce memory usage by not storing all intermediate partition transfer matrices simultaneously.

Table 5 contains the results of computation of the transfer matrices for a set of benchmark circuits. The data was acquired using a Dell PowerEdge 2950 multi-user server containing dual quad-core Intel Xeon CPUs with a 2.6 GHz core clock speed and hosting a linux operating system. Reported times are actual runtime values as reported through system calls to the linux **time** function. Circuit size is reported with the number of primary inputs and outputs, the number of gates, and the number of partitions given. Actual computation time is reported in milliseconds as the time for partitioning and the time for computing the overall transfer function matrix.

The results in Table 5 illustrate the partitioning and explicit matrix formulation of the approach. Because the explicitly represented partition and overall transfer matrices are exponentially large, it is necessary to utilize a more compact representation. Fortunately, the partition and overall matrices are sparse and consist of elements that are restricted to {0,1}. A variety of sparse matrix structures and associated arithmetic algorithms can be used to greatly reduce computational requirements.

Example 2 shows that the transfer matrix is isomorphic to a function truth table. This observation allows any data structure for representation of a truth table to be used as a transfer matrix representation. Accompanying the structure, there must be a means of manipulating them using linear algebraic operations as well.

In the second set of experimental results, we represent the transfer matrices as BDD structures. In [20] algorithms for linear algebraic operations using BDDs as the underlying data structures are described. Table 5 contains the name of the circuit and its size in terms of the number of primary inputs and outputs. The amount of memory required to store the overall transfer matrix in KB and the time required to build the matrix are also given.

6 Conclusions

An alternative to the traditional switching theory model for digital network representation and manipulation is developed and is based on a linear algebraic underpinning. The principle basis of the approach is the representation of a digital logic network as a transfer function in the form of a matrix that linearly transforms the input stimulus, represented as a vector, to a corresponding output response also represented as a vector. It is shown that the transfer matrix is isomorphic to representations of the switching theory model and a technique for computing the transfer matrix directly from a structural netlist representation is presented. The direct computation of the transfer matrix from a netlist avoids problems that may occur due to excessive memory requirements for other types of switching function representations.

Using the transfer function framework, methods for both digital network simulation and implication are presented. The use of constants $\langle \emptyset |$ and $\langle t |$ allow the methods to produce multiple responses through a single calculation. This approach is a viable alternative to current symbolic simulation and implication approaches based upon switching theory models.

Acknowledgments

The author thanks David Houngninou for generating experimental results and proofreading assistance.

References

[1] G. Boole, *The Mathematical Analysis of Logic—Being an Essay Towards a Calculus of Deductive Reasoning*. Cambridge, George Bell, U.K.: MacMillan, Barclay, & MacMillan, 1847.

[2] C. Shannon, “A symbolic analysis of relay and switching circuits,” MS thesis, Dept. Elect. Eng., Massachusetts Inst. Technol., Cambridge, MA, USA, Aug. 1937.

[3] C.-T. Chen, *Linear System Theory and Design, Holt*. New York, NY, USA: Rinehart and Winston, ISBN 0-03-060289-0, 1984.

[4] R. E. Bryant, “Graph-based algorithms for Boolean functions manipulation,” *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 667–691, Aug. 1986.

[5] T. Sasao, *Switching Theory for Logic Synthesis*. Norwell, MA, USA: Kluwer, 1999.

[6] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[7] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. New York, NY, USA: Springer, 2010.

[8] N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[9] M. A. Thornton, “Spectral analysis of digital logic using circuit netlists,” in *Proc. Int. Proc. Conf. Comput. Aided Syst. Theory*, Feb. 2011, pp. 414–415.

[10] S. A. Szygenda, “The simulation automation system, using automatic program generation for hierarchical digital simulation systems,” in *Proc. Eur. Simul. Conf.*, 1990, pp. 61–65.

[11] C. Segar and R. Bryant, “Formal verification by symbolic evaluation of partially-ordered trajectories,” *Formal Methods Syst. Design*, vol. 6, no. 2, pp. 147–189, 1995.

[12] D. M. Miller and M. A. Thornton, *Multiple-Valued Logic Concepts and Representations*. San Rafael, CA, USA: Morgan & Claypool Publishers, Jan. 2008.

[13] G. Ricci and T. Levi-Civita, “Méthodes de calcul différentiel absolu et leurs applications,” *Mathematische Annalen*, vol. 54, nos. 1/2, pp. 125–201, Mar. 1900.

[14] P. A. M. Dirac, “A new notation for quantum mechanics,” *Proc. Cambridge Philosophical Soc.*, vol. 35, p. 416, 1939.

[15] IEEE Design Automation Standards Committee, *IEEE Standard Multivalue Logic System for VHDL Model Interoperability*, 1993.

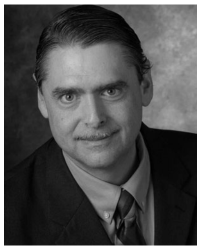
[16] IEEE Design Automation Standards Committee, *1364–2001-IEEE Standard Verilog Hardware Description Language*, 2001.

[17] J. G. Koller and W. C. Athas, “Adiabatic switching, low energy computing, and the physics of storing and erasing information,” in *Proc. Workshop Phys. Comput.*, Oct. 1994, pp. 267–270.

[18] P. Telchmann, *Adiabatic Logic Future Trend and System Level Perspective*. New York, NY, USA: Springer Publishers, 2012.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.

[20] E. M. Clarke, M. Fujita, P. C. McGeer, K. McMillan, J. C.-Y. Yang, and X. Zhao, “Multi-terminal binary decision diagrams: An efficient data structure for matrix representation,” in *Proc. IEEE Int. Workshop Logic Synthesis*, 1993, pp. 1–15.



Mitchell A. Thornton (M’85-SM’99) received the BS degree in electrical engineering from Oklahoma State University in Stillwater, OK, in 1985, the MS degree in electrical engineering from the University of Texas-Arlington in Arlington, Texas, in 1990, the MS degree in computer science from Southern Methodist University in 1993, and the PhD degree in computer engineering from Southern Methodist University in 1995. He was a senior electronic systems engineer at E-Systems, Inc. in Greenville, TX from 1986 through 1991. He was employed as a design engineer at Cyrix Corporation from 1992 through 1993. He has served as a full-time faculty member in the University of Arkansas from 1995 to 1999, Mississippi State University from 1999 to 2002, and is currently the Cecil H. Green chair of engineering and professor at Southern Methodist University. He is a licensed professional engineer in the States of Arkansas, Mississippi, and Texas. Within IEEE, he has served as a chair of several committees. His research interests include electronic design automation algorithms for synthesis and verification, computer arithmetic, disaster tolerant systems and modeling, and computer security hardware. He is a senior member of the IEEE.