STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis

Armin Alaghi, *Student Member, IEEE*, and John P. Hayes, *Life Fellow, IEEE*

***Abstract*[1]—Stochastic computing (SC) is an approximate computing technique that processes data in the form of long pseudorandom bit-streams which can be interpreted as probabilities. Its key advantages are low-complexity hardware and high-error tolerance. SC has recently been finding application in several important areas, including image processing, artificial neural networks, and low-density parity check decoding. Despite a long history, SC still lacks a comprehensive design methodology, so existing designs tend to be either *ad hoc* or based on specialized design methods. In this paper, we demonstrate a fundamental relation between stochastic circuits and spectral transforms. Based on this, we propose a general, transform-based approach to the analysis and synthesis of SC circuits. We implemented this approach in a program spectral transform use in stochastic circuit synthesis (STRAUSS), which also includes a method of optimizing stochastic number-generation circuitry. Finally, we show that the area cost of the circuits generated by STRAUSS is significantly smaller than that of previous work.**

***Index Terms*—Design methodology, logic synthesis, probabilistic methods, stochastic circuit optimization, stochastic computing (SC).**

I. Introduction

STOCHASTIC computing (SC) is an unconventional computing technique introduced by Gaines [6], [7] and Poppelbaum *et al.* [21] that processes long digital bit-streams that have well-defined numerical values, but randomly generated bit-patterns. It employs simple logic circuits to perform complex arithmetic operations For instance, multiplication can be implemented by a single AND gate. A bit-stream X containing N_1 1s and N_0 0s denotes the number $p = N_1/(N_1 + N_0)$. Since p lies in the real-number interval [0, 1], it is usually interpreted as the probability of a 1 appearing at a randomly chosen location in X or, equivalently, as the probability that X outputs a 1 at a randomly chosen time.

With suitable interpretation (scaling) of the bit-streams' numerical values, SC can essentially approximate any arithmetic operation [6]. Fig. 1 shows a representative stochastic circuit C which computes the arithmetic function

$$Z = -0.25(X_1 + X_2) \tag{1}$$

involving addition, multiplication, and negation of N-bit numbers. (Why this is so will be explained later.) The accuracy and precision of an SC computation like this depends on the length and randomness of the input bit-streams. Fig. 1 shows $N = 12$, but in practice N is often much longer. Note that, the circuit C is a standard logic circuit defined by Boolean algebra, but its stochastic behavior, as given in (1), is basically analog arithmetic over rational or real numbers. Also note that, the circuit C has three auxiliary inputs (the r_i inputs in Fig. 1) that provide three-independent random numbers of constant probability value 0.5, another typical feature of stochastic circuits.

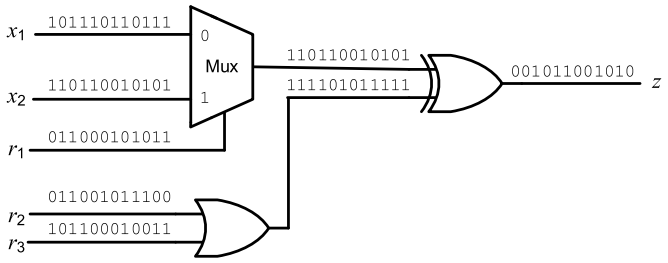


Fig. 1. Stochastic circuit implementing $Z = -(X_1 + X_2)/4$.

As Fig. 1 suggests SC is suitable for applications that require large numbers of relatively low-precision operations. Moreover, SC's probabilistic aspect makes it tolerant of errors of the soft and bit-flip type. Recent technology trends, such as the need for massively parallel processing and the increasing sensitivity of ICs to soft errors, have renewed interest in SC as an attractive alternative to conventional binary computing in a few important applications. For example, Alaghi *et al.* [2] showed that SC can outperform binary computing in some image-processing tasks. A recently discovered use of SC is in low-density parity check (LDPC) decoding [8]; LDPC codes are part of the IEEE WiFi standard [11]. Naderi *et al.* [18] have implemented a stochastic LDPC decoder that has performance comparable to that of conventional designs, but uses less chip area. Another attractive feature of stochastic circuits, reflecting their small size, is their low power requirement. Alaghi and Hayes [1] further discuss the benefits and applications of SC in the contemporary design space.

Although SC has been known for decades, most SC designs have been *ad hoc* in nature and tailored to very specific

Manuscript received August 25, 2014; revised December 18, 2014 and February 21, 2015; accepted April 15, 2015. Date of publication May 12, 2015; date of current version October 16, 2015. This work was supported by the U.S. National Science Foundation under Grant CCF-1318091. This paper was recommended by Associate Editor J. Cortadella.

The authors are with Advanced Computer Architecture Laboratory, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: alaghi@eecs.umich.edu; jhayes@eecs.umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2432138

[1]Parts of this paper are based on "A spectral transform approach to stochastic circuits," which was presented at the International Conference on Computer Design, Oct. 2012 [3].

applications. Some are based on assembling circuits from a fixed and limited library of SC components, e.g., multipliers and adders. A first step toward systematic design of stochastic circuits was taken by Qian and Riedel [22], who developed a method of implementing arithmetic functions based on Bernstein polynomials. Similar methods of synthesis have been proposed targeting combinational [26] and sequential circuits [16], [27].

Sequential circuits can, in general, implement a larger class of stochastic functions, and may be more efficient in implementing nonlinear functions such as *tanh* and *exp* [15]. However, their performance and accuracy are degraded by the auto-correlations that may exist in the input bit-streams, as well as the "warm-up" time needed for the circuits to arrive at their steady-state distributions, two phenomena that do not affect combinational circuits. Furthermore, there are many examples of combinational stochastic circuits that are more efficient than their sequential counterparts. Combinational multipliers, which will be discussed later in this section, do not have a low-cost sequential implementation. Another example is the combinational edge-detection circuit proposed in [2], which is significantly more efficient than the sequential implementation of the same function [14].

Despite the fact that large stochastic systems such as LDPC decoders [18] and even general-purpose machines [20] have been successfully implemented, a comprehensive SC design methodology has yet to appear.

This paper identifies and explores a fundamental relation between SC circuits and spectral transforms like the Fourier transform [10], [13]. Such transforms have many applications in engineering. For instance, consider the time-domain impulse response of an analog filter. While it contains all the information about the filter's behavior, it is not easy to extract the response of the filter to a given input signal. The Fourier transform of the impulse response reveals the "hidden" frequency-domain behavior of the system, from which its response to a given input signal can readily be found.

The spectral transforms of interest in this paper map Boolean functions (BFs) from the logic domain to the domain of real numbers. We show that they lead to a unique multilinear polynomial representation of a given BF, which defines its SC behavior. (A multilinear polynomial is a polynomial in which terms may contain products of variables, but no variable appears with a power of two or higher.) Based on this, we derive methods of analyzing and synthesizing combinational stochastic circuits. We present a synthesis algorithm called spectral transform use in stochastic circuit synthesis (STRAUSS), which applies to general polynomial types and covers a wider range of (positive and negative) stochastic number (SN) formats than previous work. Furthermore, it can synthesize circuits with significantly lower area cost.

The main contributions of this paper are as follows.

1) The use of spectral transforms to link the structure and behavior of stochastic circuits.
2) A general spectral algorithm for designing stochastic circuits to implement arbitrary arithmetic functions.
3) A method of optimizing stochastic designs by sharing SN generators.

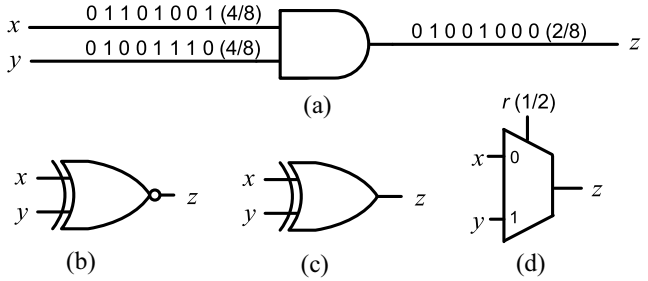


Fig. 2. Stochastic multipliers: (a) UP, (b) BP, and (c) IBP. (d) Stochastic adder for all three formats.

4) A cost comparison of circuits designed with the proposed spectral method and previous nonspectral approaches.

II. Basics of Stochastic Computing

This section reviews the basics of SC, and introduces the terminology and notation used throughout this paper.

In SC, the numerical value associated with a 1-bit logic signal x is usually taken to be the probability of seeing a 1 on x. We denote this value by X and refer to it as an SN that corresponds to signal x. (SNs that are carried by multiple logic signals are not considered in this paper.) This is called the unipolar (UP) format and represents real numbers over the unit interval [0, 1], which is also the probability domain. Table I shows three different but closely related formats for SNs. The bipolar (BP) encoding is often preferred because it represents signed numbers over the $[-1, +1]$ interval in a natural way. We introduce a new SN format called inverted bipolar (IBP) which is the inverse of BP. While, the IBP and BP formats are essentially equivalent, IBP is more convenient to use with spectral transforms, where the Boolean values 0 and 1 are replaced by $+1$ and -1, respectively. This small notational change greatly simplifies the analysis and synthesis of circuits in the spectral domain. To illustrate the various SN formats, consider the bit-stream 0110101101 of length 10 containing six 1s and four 0s. It represents the UP number 0.6, BP number 0.2, and IBP number -0.2. For most of this paper, we will use the IBP format, unless stated otherwise.

Multiplication of UP numbers is implemented by a single AND gate; see Fig. 2(a). The probability of seeing a 1 at the output z is equal to the probability of seeing 1 at input x multiplied by the probability of seeing a 1 at input y, assuming the two probabilities are independent. Hence, $Z = XY$. Fig. 2(b)–(c) shows stochastic multipliers for the BP and IBP formats. Note that, the stochastic behavior of a circuit changes with the SN format used. For example, the XOR gate of Fig. 2(c), i.e., the IBP multiplier, realizes the following UP operation:

$$Z = X(1 - Y) + (1 - X)Y.$$

The circuit of Fig. 1 is designed to operate on BP numbers; its XOR gate performs both multiplication and negation. Once, we know the stochastic behavior of a circuit in one format, we can use the equations given in Table I to derive its behavior in the other formats. In the next section, we will

TABLE I
INTERPRETATIONS OF A BIT-STREAM OF LENGTH N WITH N_1 1S AND N_0 0S

Format	Number value	Number range	Relation to unipolar value X
Unipolar (UP)	N_1/N	[0,1]	X
Bipolar (BP)	$(N_1 - N_0)/N$	[−1,+1]	$2X - 1$
Inverted bipolar (IBP)	$(N_0 - N_1)/N$	[−1,+1]	$1 - 2X$

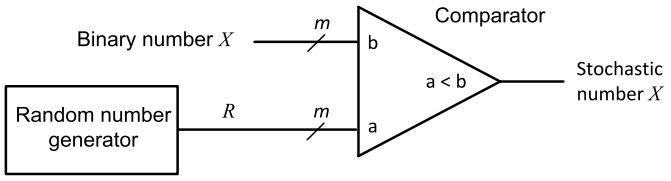


Fig. 3. Stochastic number generator (SNG).

show how spectral transforms can be employed to derive the stochastic behavior of arbitrary logic circuits for any SN format.

Stochastic addition is usually implemented by the multiplexer of Fig. 2(d), which is used with all three SN formats in Table I. Note that, besides the main (data) inputs x and y, an auxiliary signal r is applied to the multiplexer's select input. This consists of a uniform random bit-stream of constant value $R = 0.5$ (in UP format), which serves as a scaling factor; r is typically obtained from a (pseudo) random number generator, such as a linear feedback shift register (LFSR) [12]. Since adding two numbers from the interval [0, 1] (or $[-1, +1]$) can produce a sum in the interval [0, 2] (or $[-2, 2]$), scaling by 0.5 brings the sum back into the original interval. In the BP circuit of Fig. 1, the multiplexer computes $0.5(X_1 + X_2)$, which the XOR gate then multiplies by another factor of 0.5 and negates. The BP constant 0.5 (which is 0.75 in UP format) required by the XOR is generated by the OR gate.

To combine stochastic and conventional logic, interface circuits are needed that convert between (weighted) binary and SN formats. Converting an SN to binary form requires counting the number of 1s in the bit-stream, so a counter suffices to implement it. Converting from binary to SN form requires a more complex circuit. Fig. 3 shows a typical stochastic number generator (SNG) that converts a binary number of value X to an SN of the same value in UP format. At each clock cycle, it compares X with a binary number R that is uniformly distributed in the [0, 1] interval (or equivalently, on m wires, each of which carries an SN of UP value 1/2). The SNG outputs a bit-stream representing X, one bit per clock cycle. As reported in [26], SNGs can consume as much as 80% of an SC circuit's total area, so reducing the number of SNGs is a major design challenge in SC.

The accuracy and precision of SC depends on the length of the bit-streams used and the degree of dependence or correlation among the input bit-streams. For instance, the SNG of Fig. 3, if clocked for 2^m cycles, produces a 2^m-bit SN that has m bits of precision. If the bit-streams applied to the circuit of Fig. 2(a) are correlated, say each has the same bit-pattern of value X, then the output will also have value X instead of X^2. Hence, an AND gate will not perform multiplication accurately if its inputs are correlated. In this paper, we make the usual assumption that the SNs applied to the primary inputs of our stochastic circuits are uncorrelated. Further discussion of accuracy and correlation issues can be found in [1] and [4].

III. SPECTRAL TRANSFORMS

We now introduce the spectral transforms used to analyze and synthesize stochastic circuits. An n-variable BF $f(x_1, \ldots, x_n)$ maps $B^n = \{0, 1\}^n$ to $B = \{0, 1\}$. Here, B^n is seen as a 2^n-dimensional vector space, where each dimension corresponds to a row of f's truth table (TT), or equivalently, to an n-variable minterm. For example, if $n = 2, f(x_1, x_2)$ has the four-dimensional basis vectors $m_0 = (1, 0, 0, 0)$, $m_1 = (0, 1, 0, 0)$, $m_2 = (0, 0, 1, 0)$, and $m_3 = (0, 0, 0, 1)$, and can be written as

$$f(x_1, x_2) = \sum_{i=0}^{3} c_i m_i. \tag{2}$$

This is the familiar sum-of-minterms expansion of f, where the c_i's are 0-1 coefficients that define f.

Example 1: If $f_1(x_1, x_2) = x_1 \vee \bar{x}_2$, then $f_1(x_1, x_2) = m_0 \vee m_2 \vee m_3$, or in the column-vector form that we use later

$$f_1(x_1, x_2) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \vee \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \vee \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \tag{3}$$

The last vector is essentially g's TT. To save space, we also write such vectors in the transposed form $[1\ 0\ 1\ 1]^{\mathrm{T}}$.

As Table I indicates, in the SC context we must deal with real numbers ranging over intervals such as [0, 1] and $[-1, +1]$. Given a BF such as f_1, we are interested in an equivalent real function $\hat{F}_1$ defined on $[-1, +1]^n$ (or some other appropriate domain) that specifies the SC behavior of f_1. This function can be obtained by interpolating the TT values in the real domain via a multilinear polynomial. For example, consider the TT vector in (3). By replacing 0s and 1s with $+1$s and -1s, respectively, we see that f_1 produces the TT vector $[-1\ 1\ -1\ -1]^{\mathrm{T}}$ for inputs $(x_1, x_2) = (1, 1), (1, -1), (-1, 1)$, and $(-1, -1)$, in IBP format. These four discrete "TT points" can be embedded in a continuous real-number function as follows:

$$\begin{aligned} \hat{F}_1(X_1, X_2) = &\left(\frac{1+X_1}{2}\right)\left(\frac{1+X_2}{2}\right)(-1) \\ &+ \left(\frac{1+X_1}{2}\right)\left(\frac{1-X_2}{2}\right)(+1) \\ &+ \left(\frac{1-X_1}{2}\right)\left(\frac{1+X_2}{2}\right)(-1) \\ &+ \left(\frac{1-X_1}{2}\right)\left(\frac{1-X_2}{2}\right)(-1). \end{aligned}$$

Observe that each term of the foregoing expression assumes the correct value 1 or -1 at each TT point. On expanding

this expression, we get

$$\begin{aligned}\hat{F}_1(X_1, X_2) = 0.25\big[&(1 + X_2 + X_1 + X_1X_2)(-1) \\ &+ (1 - X_2 + X_1 - X_1X_2)(+1) \\ &+ (1 + X_2 - X_1 - X_1X_2)(-1) \\ &+ (1 - X_2 - X_1 + X_1X_2)(-1)\big] \\ &= -0.5 - 0.5X_2 + 0.5X_1 - 0.5X_1X_2. \qquad (4)\end{aligned}$$

The polynomial (4) interpolates the TT values in the real numbers. It is linear with respect to variables X_1 and X_2, and is referred to as multilinear. Most importantly as we will see, it represents the stochastic behavior of the BF f_1.

More generally, given a BF $f(x_1, x_2, \ldots, x_n)$, if n independent SNs $X_1, X_2, \ldots, X_n$, defined in an SC format, such as UP, BP, or IBP, are the input arguments of f, the output is another SN that is some function of $X_1, X_2, \ldots, X_n$. We denote this function by $\hat{F}(X_1, X_2, \ldots, X_n)$ and refer to it as the SC behavior of f. We will see that $\hat{F}$ has a unique multilinear form similar to that in (4), and can be determined by means of spectral transforms.

The spectral transforms of interest execute a change of basis from the minterm space of a BF f to a real-valued space. We employ the Fourier transform $\mathcal{F}$ for BFs, which is also known as the discrete Walsh transform in Hadamard ordering [13], [19]. Spectral transforms of this type have been considered previously for a wide variety of logic design and testing tasks [10], [13]. For BFs with large values of n, it may be impractical to deal with 2^n-dimensional spectra, although concise representations of spectra for functions with hundreds of variables are known [5]. This size issue is of much less concern in SC, however, because the values of n tend to be small, e.g., $n = 2$ in Fig. 1.

To compute the Fourier transform of a BF given as a TT vector $\vec{f}$ or equivalent, we first replace 0 and 1 by $+1$ and -1, respectively. The Fourier transform $F = \mathcal{F}(f)$ is specified by

$$\vec{F} = \frac{1}{2^n} H_n \times \vec{f} \qquad (5)$$

where $\vec{F}$ is the vector form of F denoting its spectral coefficients or spectrum and H_n is the Walsh matrix (with natural or Hadamard ordering) of dimension 2^n defined recursively by

$$H_1 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \quad H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}.$$

Equation (5) is evaluated using the rules of linear algebra over real numbers. In the case of f_1 from Example 1, we get

$$\vec{F}_1 = \frac{1}{4} H_2 \times \vec{f}_1 = \frac{1}{4} \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} -1 \\ +1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \\ +0.5 \\ -0.5 \end{bmatrix}.$$

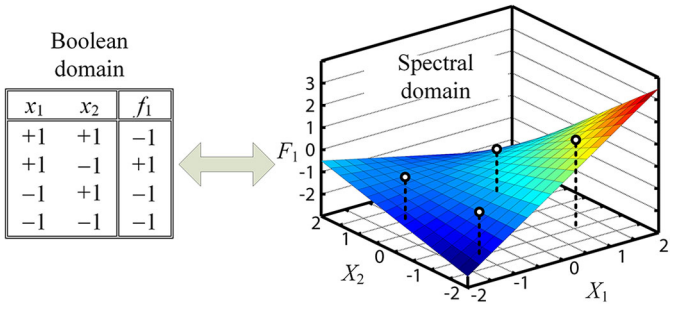


Fig. 4. Illustration of the spectral transformation of the function f_1 of Example 1.

The result is the spectrum of f, and the corresponding basis is

$$[1\ 1\ 1\ 1]^{\mathrm{T}}, [1\ -1\ 1\ -1]^{\mathrm{T}}, [1\ 1, -1, -1]^{\mathrm{T}}, [1\ -1\ -1, 1]^{\mathrm{T}}$$

defined by the rows or columns of H_2. (Note that H_n is symmetric.) These basis vectors resemble digital waveforms, and are analogous to harmonics in the sine–cosine Fourier transform used for time–frequency conversions. More specifically, they correspond to the four linear BFs: 1, x_2, x_1, and $x_1 \oplus x_2$, where $\oplus$ denotes XOR. Recall that in the spectral domain, XOR is replaced by multiplication.

Analogous to the sum-of-minterms expansion (2) for f_1 in the Boolean domain, we write the transformed function as

$$F_1(X_1, X_2) = \sum_{i=0}^{3} C_i S_i \qquad (6)$$

where the S_i's are the basis vectors 1, X_2, X_1, and X_1X_2, in the spectral domain, and the C_i's constitute the spectrum of f_1. Hence, (6) becomes

$$F1(X_1, X_2) = -0.5 - 0.5X_2 + 0.5X_1 - 0.5X_1X_2.$$

This is a multilinear polynomial that interpolates (matches) the original BF f at its four Boolean input coordinates (TT points), namely $(X_1, X_2) = (1, 1), (-1, 1), (1, -1), (-1, -1)$. This transformation is illustrated in Fig. 4. Notice that the last expression above is exactly the same as (4), and the process of arriving at the two expressions is similar. So, we see intuitively that the Fourier transform of a BF defines its unique SC behavior. This leads to the following theorem.

Theorem 1: If $\hat{F}$ denotes the SC behavior of an n-variable BF f in IBP format, and $F = \mathcal{F}(f)$ is f's Fourier transform, then $\hat{F} = F$.

Proof: See the Appendix. ■

Thus, given a combinational circuit or BF, we can determine its IBP behavior by computing its Fourier transform. A simple interval conversion according to Table I is all that is required to determine its behavior in other SN formats. The next example illustrates this.

Example 2: According to Fig. 2, an XOR gate to serves as an IBP multiplier. We can verify this as follows. XOR has the TT vector $\vec{f}_2 = [1\ -1\ -1\ 1]^{\mathrm{T}}$.

Calculating its Fourier transform yields

$$\vec{F}_2 = \frac{1}{4}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

so $F_2(X_1, X_2) = X_1X_2$. According to Theorem 1, the IBP behavior of XOR is

$$\hat{F}_2(X_1, X_2) = F_2(X_1, X_2) = X_1X_2.$$

Similarly, we know that a two-input AND gate acts as a multiplier in UP format. In this case, $\vec{f}_3 = [1\ 1\ 1\ -1]^{\mathrm{T}}$ and

$$\vec{F}_3 = \frac{1}{4}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +1 \\ +1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} +0.5 \\ +0.5 \\ +0.5 \\ -0.5 \end{bmatrix}.$$

Thus

$$F_3(X_1, X_2) = 0.5(1 + X_1 + X_2 - X_1X_2). \tag{7}$$

Since, we want the AND behavior in UP format, we must map (7) from IBP to UP. Let p_1, p_2, and p_3 denote the UP values of X_1, X_2, and F_3, respectively. Table I implies $X_1 = 1 - 2p_1, X_2 = 1 - 2p_2$, and $F_3 = 1 - 2p_3$. Hence

$$1 - 2p_3 = 0.5\big(1 + 1 - 2p_1 + 1 - 2p_2 \\ - 1 + 2p_1 + 2p_2 - 4p_1p_2\big)$$

leading to the desired multiplication $p_3 = p_1p_2$.

Finally, we note that the Fourier transform is invertible and preserves all information about f. We can therefore retrieve the original TT form by applying the inverse Fourier transform $f = \mathcal{F}^{-1}(F)$ to the spectrum. Since H_n is related to its own inverse by a 2^n factor, this may be calculated as follows:

$$\vec{f} = H_n \times \vec{F}. \tag{8}$$

This is the key link from the desired SC behavior F to a logic function f that, with appropriate modifications, implements F.

As mentioned, the elements of $\vec{F}$ correspond to polynomial terms in the spectral domain. For $n = 2$, these terms are 1, X_2, X_1, and X_1, X_2, respectively. In the general case, we assign an n-bit binary number to each element of $\vec{F}$, starting from all 0s. So the elements of $\vec{F}$ are assigned to "000 ... 0," "000 ... 1," ..., "111 ... 0," and "111 ... 1," respectively. Now to find the polynomial term corresponding to each element, we get the assigned binary number and replace each 1 by the corresponding X_i for that position, and replace each 0 by 1. Hence, the terms corresponding to the first, second, and last elements of $\vec{F}$ are 1, X_n, and $X_1X_2 \ldots X_n$, respectively.

IV. Spectral Transform-Based Synthesis

The spectral transforms discussed so far have several useful applications in the SC context. Besides analyzing BFs and extracting their SC behavior, they can be used to systematically design combinational SC circuits. But before discussing our synthesis method (STRAUSS), a few preliminary concepts are needed.

A. Stochastically Implementable Functions

Theorem 1 implies that a combinational circuit can implement a stochastic function $\hat{F}$ given in multilinear polynomial form, i.e., one in which terms may contain products of variables of degree at most one. The idea is then to apply the inverse Fourier transform $\mathcal{F}^{-1}$ to $\hat{F}$ and obtain the corresponding BF $\vec{f}$ in vector TT form, as in (8). However, applying $\mathcal{F}^{-1}$ to an arbitrary target function $\hat{F}$ does not necessarily yield a vector $\vec{f}$ whose elements are the TT values $+1$ and -1. In fact, we can have the following three possible outcomes.

1) All the elements of $\vec{f}$ are $+1$ or -1, in which case $\vec{f}$ is the TT of a BF and is directly implementable by a logic circuit.
2) All elements of $\vec{f}$ lie in the interval $[-1, +1]$, but some have values $\{c_i\}$ other than $+1$ or -1. In that case, the function is still implementable, but requires auxiliary inputs and circuitry to generate the c_i's, as discussed in Section V.
3) Some elements of $\vec{f}$ are larger than $+1$ or less than -1, in which case the function is not implementable. It is still possible to implement a related function $\hat{F}'$ that is an approximate or scaled version of $\hat{F}$.

We say polynomial $\hat{F}$ is SC-implementable if we can synthesize a stochastic circuit whose behavior is defined by $\hat{F}$. SC-implementable functions fall into the first two categories above, and are distinguished by having inverse Fourier transforms. The simple product function X_1X_2 is SC-implementable, whereas the unscaled sum $X_1 + X_2$ is not. We will refer to the latter as SC-unimplementable.

To illustrate the general case, consider the generic two-variable polynomial

$$\hat{F}(X_1, X_2) = a_0 + a_1X_2 + a_2X_1 + a_3X_1X_2.$$

Taking its inverse Fourier transform, we get

$$\vec{f} = H_2 \times \vec{F} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \\ = \begin{bmatrix} a_0 + a_1 + a_2 + a_3 \\ a_0 - a_1 + a_2 - a_3 \\ a_0 + a_1 - a_2 - a_3 \\ a_0 - a_1 - a_2 + a_3 \end{bmatrix}.$$

In order for $\hat{F}$ to be SC-implementable, the elements of $\vec{f}$ should be in the interval $[-1, +1]$. In other words, the following constraints should be satisfied:

$$-1 \le a_0 + a_1 + a_2 + a_3 \le +1$$
$$-1 \le a_0 - a_1 + a_2 - a_3 \le +1$$
$$-1 \le a_0 + a_1 - a_2 - a_3 \le +1$$
$$-1 \le a_0 - a_1 - a_2 + a_3 \le +1.$$

Constraints of this kind can be obtained for polynomials of n variables by applying the inverse Fourier transform to them. Concepts similar to SC-implementable polynomials are discussed by Qian and Riedel [22], [24]. They implement stochastic functions in the form of Bernstein polynomials, and define constraints on the coefficients of Bernstein polynomials in order to distinguish implementable functions. As we will

show later, that approach can also be interpreted in terms of spectral transforms.

B. Target Function Conversion

In order to synthesize a stochastic circuit for an arbitrary target function $\hat{F}$, a few conversion steps are required. For instance, the inverse Fourier transform can only produce suitable BFs when applied to multilinear functions [19]. Suppose $\hat{F}$ is an ordinary polynomial of degree n

$$\hat{F}(X) = a_0 + a_1X + a_2X^2 + \cdots + a_nX^n$$

implying that it has nonlinear terms. We convert it to a multilinear polynomial $\hat{P}(X_1, \ldots, X_n)$ in which the nonlinear terms of $\hat{F}$, such as a_nX^n, are replaced by multilinear terms like $a_nX_1X_2 \ldots X_n$. The new variables $X_1, X_2, \ldots, X_n$ are assumed to be independent copies of the original variable X.

There are many possible ways to select a multilinear polynomial $\hat{P}$ that corresponds to $\hat{F}$. A natural choice is one that is symmetric with respect to all its variables thus

$$\begin{aligned}\hat{P}(X_1, \ldots, X_n) = a_0 &+ \frac{a_1}{n}(X_1 + \ldots + X_n) \\ &+ \frac{a_2}{\binom{n}{2}}(X_1X_2 + X_1X_3 + \ldots + X_{n-1}X_n) \\ &+ \cdots + a_nX_1X_2 \ldots X_n.\end{aligned}$$

However, asymmetric polynomials are also possible. For instance

$$\hat{P}(X_1, \ldots, X_n) = a_0 + a_1X_1 + a_2X_1X_2 + \cdots + a_nX_1 \ldots X_n$$

is one of the possible asymmetric multilinear polynomials that correspond to $\hat{F}$. Existing synthesis methods [3], [22], [26] assume symmetry, but as we will show for the first time in Section VI, asymmetric multilinear polynomials may lead to better implementations.

Finally, to synthesize a function $\hat{F}$ from $[0, 1]^n$ to $[0, 1]$ that is SC-unimplementable, we convert it to an SC-implementable polynomial by approximation. This step is a straightforward polynomial fitting problem and can be easily solved by tools such as MATLAB [17].

C. Synthesis Examples

The main steps of STRAUSS are listed in Fig. 6. We first illustrate them with examples, and then discuss their details.

Example 3: Consider the problem of reverse engineering the IBP multiplier, so the given function is $\hat{F}_2(X_1, X_2) = X_1X_2$. Since this already has the desired multilinear polynomial form, we can skip Step 1 of STRAUSS and proceed to Step 2, where we use the inverse Fourier transform to obtain f_2's TT vector

$$\vec{f}_2 = H_2 \times \vec{F}_2 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

The result $\vec{f}_2$ only has $+1$ and -1 as elements, and is clearly the TT of a two-input XOR gate.

Now consider the same problem for the UP multiplier $\hat{F}_3(X_1, X_2) = X_1X_2$. Note that, this function will be different from $\hat{F}_2$ because of the format change. Mapping $\hat{F}_3$ to the IBP format yields

$$\begin{aligned}\hat{P}_3(X_1, X_2) &= 1 - 2\hat{F}_3\left(\frac{1 - X_1}{2}, \frac{1 - X_2}{2}\right) \\ &= 0.5(1 + X_1 + X_2 - X_1X_2).\end{aligned}$$

Applying the inverse Fourier transform to $\hat{P}_3$ produces the TT $[1\ 1\ 1\ -1]^{\mathrm{T}}$, which defines an AND gate.

Example 4: Suppose, we attempt to synthesize the arithmetic sum function $\hat{F}_4(X_1, X_2) = X_1 + X_2$. This is also in multilinear form, so we proceed with the inverse Fourier transform

$$\vec{f}_4 = H_2 \times \vec{F}_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ -2 \end{bmatrix}.$$

Two elements of $\vec{f}_4$ lie outside the interval $[-1, +1]$, which means that it is SC-unimplementable. The standard solution is the scaled addition which substitutes $s(X_1 + X_2)$ for $X_1 + X_2$, where s is a scale factor that makes the function SC-implementable; in this case $s = 1/2$. The new target function is $\hat{F}_5(X_1, X_2) = 1/2(X_1 + X_2)$, which yields

$$\vec{f}_5 = H_2 \times \vec{F}_5 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

However, the result $\vec{f}_5$ has elements other than $+1$ and -1, namely 0, so a constant number generation step (Step 3) is required. This is discussed in the next section.

V. Constant-Number Generation

The third step of STRAUSS, as seen in Fig. 6, is constant number generation, a challenging problem in itself. We first discuss the intuition behind this step. Then, we formally define the problem and present several solutions.

Continuing Example 4 from the previous section, we derived the TT $\vec{f}_5 = \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix}^{\mathrm{T}}$ for the stochastic add operation. We can interpret this TT as follows. If both inputs are 0, then a constant SN of value $+1$ (in IBP format) is sent out; if both inputs are 1, then a constant SN of value -1 (in IBP format) is sent out; if one input is 1 and the other is 0, a constant SN of value 0 is sent out. Producing an SN with values other than $+1$ or -1 requires additional inputs, i.e., random number

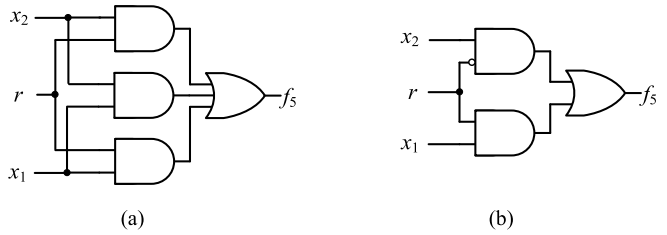


Fig. 5. Two synthesized circuits for SC addition: (a) without and (b) with optimizations.

Procedure *STRAUSS* ($\hat{F}$) -- Returns a stochastic circuit
-- implementing the target function $\hat{F}$

Step 1. Format the target function $\hat{F}$ as a multilinear polynomial $\hat{P}$

Step 2. Compute the inverse Fourier transform $\mathcal{F}^{-1}(\hat{P})$ to obtain $\vec{f}$

Step 3. Generate constant IBP values for elements of $\vec{f}$(Section V)

Step 4. Optimize via standard combinational design procedures

Fig. 6. Main steps of the STRAUSS synthesis method.

sources. For the running example, we can replace the two-input TT $\vec{f}_5$, with the following three-input TT:

$$\vec{f}_5 = \begin{bmatrix} 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$

which is the BF $f_5(x_1, x_2, r) = (x_1 \wedge r) \vee (x_2 \wedge r) \vee (x_1 \wedge x_2)$, where an auxiliary input r has been added to the function. (This happens to be the majority function.) The r input must be fed with the IBP SN 0, i.e., a pure random bit-stream. Fig. 5(a) shows a straightforward AND–OR implementation of f_5.

It is possible to optimize the synthesized circuit further by reordering the elements of $\vec{f}_5$. For example, the following TT has the same stochastic behavior of $\vec{f}_5$ but has a simpler implementation:

$$\vec{f}_5 = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}.$$

This is the BF $f_5(x_1, x_2, r) = (x_1 \wedge r) \vee (x_2 \wedge \bar{r})$, which has the AND–OR implementation of Fig. 5(b). It is obvious that this new circuit is a 2-to-1 multiplexer with r as its select input. This is precisely the standard scaled adder in the SC literature [6].

Finding the best ordering of $+1$s and -1s of a TT is a difficult optimization task. We now formally define the constant SN generation problem, and discuss our solution method for it.

Single SN Generation Problem: Given a constant number $c \in [-1, +1]$ and a desired precision m, we want to find an m-input BF $f(r_1, \ldots, r_m)$ with $k = \lfloor (1-c) \cdot 2^{m-1} \rfloor$ (or $k = \lceil (1-c) \cdot 2^{m-1} \rceil$) minterms that has minimum cost.

When fed with pure random inputs, an m-input BF $f(r_1, \ldots, r_m)$ with $k = (1-c) \cdot 2^{m-1}$ minterms, will output a 1 with a probability of $k/2^m$. Thus, the IBP value generated at the output of f is $1 - 2 \cdot (k/2^m) = c$. Note that, the precision m only refers to the target constant c, and has no implications on the run-time for SN generation.

It should be noted that the number of auxiliary inputs introduced determines the precision of the constant numbers that can be generated. With $m+1$ auxiliary inputs $r_1, \ldots, r_m, r_{m+1}$, we can only generate the following numbers:

$$\left\{ \frac{-2^m}{2^m}, \frac{-2^m+1}{2^m}, \ldots, \frac{-1}{2^m}, \frac{0}{2^m}, \frac{1}{2^m}, \ldots, \frac{2^m-1}{2^m}, \frac{2^m}{2^m} \right\}.$$

Any other number c should be rounded to the closest number from the above set. So for an arbitrary real-valued c, one has to choose a value for m, taking into account that increasing m provides better precision and accuracy but also increases the cost of the circuit.

As the problem suggests, there may be many BFs that generate the same constant number, and our goal is to select one with minimum cost. We use literal count as our cost criterion, because it is easy to use and reflects both transistor and gate count fairly accurately in standard CMOS logic [9]. For example, both the following BFs $f_6(r_1, \ldots, r_m) = r_1$ and $f_7(r_1, \ldots, r_m) = r_1 \oplus \cdots \oplus r_m$ have 2^{m-1} minterms and thus generate the constant $c = 0$. But f_6 has a literal count of 1, while f_7 has a literal count of $2(m-1)$ using a chain or tree of XOR gates. Note that, the cost of an XOR gate is twice the cost of an elementary gate.

Qian and Riedel [23] give a method of synthesizing a minimal two-level circuit that generates a given stochastic constant. Qian *et al.* [25] discussed several other methods that synthesize multilevel circuits. The method of [25] does not directly address the single SN generation problem defined in this paper, so the optimality of that method will not be discussed here. We now present a recursive algorithm called stochastic constant generation (SCG) to obtain a minimal multilevel constant generation circuit. This algorithm takes two parameters, the number of inputs m and the number of minterms k, and returns a TT of length 2^m with k minterms. If $k \leq 2^{m-1}$, i.e., k is at most half the TTs length, then SCG fills the first half of the TT with 0s, and makes a recursive call with parameters $m-1$ and k. This recursion can be interpreted as follows. SCG returns

$$f(r_1, \ldots, r_m) = r_1 \wedge f'(r_2, \ldots, r_m)$$

in which f' is a function with $m-1$ variables and k minterms. If $k > 2^{m-1}$, SCG fills the second half of the TT with 1s, and makes a recursive call with parameters $m-1$ and $k - 2^{m-1}$. Similarly, this recursion step can be interpreted as the algorithm returning

$$f(r_1, \ldots, r_m) = r_1 \vee f'(r_2, \ldots, r_m).$$

After $m-1$ recursion steps, SCG returns a BF that is implemented by a chain of at most $m-1$ AND or OR gates. The steps of the SCG procedure are given in Fig. 7. This algorithm can also be used to solve the multiple constant SN generation problem, as discussed below.

Example 5: Consider finding a seven-input function $f_8(r_1, \ldots, r_7)$ with 77 minterms. We call SCG(7, 77), which returns a TT with 64 ones in the second half, and 13 ones in the first half. The first half is now obtained by calling SCG(6, 13). This recursion step corresponds to

$$f_8(r_1, \ldots, r_7) = r_1 \vee f_8'(r_2, \ldots, r_7)$$

Procedure *SCG* (m, k) -- Returns m-input Boolean function with k -- minterms in truth-table format

Step 1. (terminal cases)
If $k = 0$ then the function is constant 0, so return [0 0 … 0]
If $(k = 2^m)$ then the function is constant 1, so return [1 1 … 1]

Step 2a. If $k \le 2^{m-1}$, fill the first half of the truth-table TT with zeros, and fill the second half using a recursive call to $SCG(m-1, k)$:
$TT = [0\ 0\ \ldots\ 0\ SCG(m-1, k)]$

Step 2b. If $k > 2^{m-1}$ fill the second half of the truth-table TT with ones, and fill the second half using a recursive call to $SCG(m-1, k-2^{m-1})$:
$TT = [SCG(m-1, k-2^{m-1})\ 1\ 1\ \ldots\ 1]$

Step 3. Return TT

Fig. 7. Overview of the stochastic constant generation (SCG) procedure.

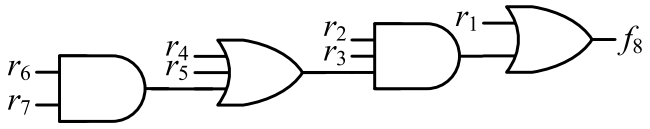


Fig. 8. Optimal circuit implementation for function f_8 of Example 5.

in which f_8' is the result of SCG(6, 13). Continuing down the recursion path we see SCG(5, 13), SCG(4, 13), SCG(3, 5), SCG(2, 1), SCG(1, 1), and SCG(0, 1) which is a terminal case. The resulting function is hence

$$f_8(r_1, \ldots, r_7) = r_1 \vee r_2 r_3 (r_4 \vee r_5 \vee r_6 r_7)$$

which has minimal literal count. The corresponding optimal circuit implementation of f_8 is shown in Fig. 8.

The circuits generated by the SCG procedure are optimal in terms of literal count. The proof is straightforward; each input of the generated BF appears at most once in the final expression. So if m inputs are required to generate a constant, the literal count of the generated circuit will be m, which is the minimum possible literal count for an m-input circuit. The constant number generators synthesized by the method of [23] have near-optimal two-level implementations, but in most cases their literal count is greater than m.

Next, we generalize the problem to multiple constants. Note that, previous work, including the methods discussed in [23] and [25], does not address multiple-constant generation.

Multiple SN Generation Problem: Given a set of constant numbers $\{c_1, \ldots, c_p\}$ where $c_i \in [-1, +1]$, we want to find p minimum-cost m-input BFs $f_1, \ldots, f_p$ with $\lfloor (1 - c_1) \cdot 2^{m-1} \rfloor, \ldots, \lfloor (1 - c_p) \cdot 2^{m-1} \rfloor$ minterms, respectively.

The multiple constant generation problem, which is step 3 of STRAUSS (see Fig. 6), is a difficult one, because there are many possible opportunities for sharing gates between the circuits for the c_i's. Since exhaustively searching among them would be very inefficient, we propose a heuristic algorithm based on the SCG procedure of Fig. 7. To generate all the constants, we call SCG for each c_i, and as SCG progresses, we keep a record of the constant SNs and circuits it has generated so far. The generated circuits are stored in a table and are reused if a previously generated SN is encountered. This approach is demonstrated in the following example which illustrates a complete synthesis computation using STRAUSS.

Example 6: Consider the target function $\hat{F}_9(X) = 0.4375 - 0.25X - 0.5625X^2$. We want to use STRAUSS to synthesize a stochastic circuit that implements $\hat{F}_9$. In Step 1 of Fig. 6, $\hat{F}_9$ is reformulated as a multilinear polynomial $\hat{P}_9$ because it contains a nonlinear term X^2. This term is eliminated by introducing two new inputs X_1 and X_2 to replace X thus

$$\hat{P}_9(X_1, X_2) = 0.4375 - 0.125(X_1 + X_2) - 0.5625 X_1 X_2.$$

This is only one of the many possible multilinear polynomials that are equivalent to the target function $\hat{F}_9$.

For this example, we chose a symmetric multilinear polynomial, but as we will show in the next section, STRAUSS examines many possible polynomials (including asymmetric ones) and chooses one that leads to a lower cost. At Step 2 of STRAUSS, we have

$$\vec{f}_9 = H_2 \times \vec{P}_9 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} +0.4375 \\ -0.1250 \\ -0.1250 \\ -0.5625 \end{bmatrix} = \begin{bmatrix} -0.375 \\ 1 \\ 1 \\ +0.125 \end{bmatrix}.$$

All the elements of $\vec{f}_9$ are in the $[-1, +1]$ interval, implying that the function is SC-implementable. However, there exist elements other than 1 and -1, namely, $c_1 = -0.375$ and $c_2 = -0.125$, which require constant SN generation (Step 3).

To generate c_1, the SCG procedure of Fig. 7 is called with parameters $m = 4$ and $k = 11$. The choice of $m = 4$ stems from the fact that it is the least number of inputs for a BF capable of generating c_1. The parameter k comes from the formula $k = (1 - c_1) \cdot 2^{m-1}$ discussed earlier. Calling SCG(4, 11) leads to the recursive calls: SCG(3, 3), SCG(2, 3), SCG(1, 1), and SCG(0, 1), and the following BF is returned:

$$f_{c_1}(r_1, r_2, r_3, r_4) = r_1 \vee (r_2 . (r_3 \vee r_4)).$$

Next, SCG is called with parameters $m = 4$ and $k = 7$ to generate c_2. Calling SCG(4, 7) entails recursive calls SCG(3, 7) and SCG(2, 3), at which point the recursion stops because the results of the previous calls (during the circuit generation for c_1) are reused. The returned BF is

$$f_{c_2}(r_1, r_2, r_3, r_4) = r_1 (r_2 \vee (r_3 \vee r_4))$$

which shares a gate with f_{c_1}. Step 3 is now done, and we have a stochastic implementation of $\hat{F}_9$, namely

$$f_9(x_1, x_2, r_1, r_2, r_3, r_4) = x_1' x_2' f_{c1} \vee x_1 x_2 f_{c_2}.$$

In the last step, f_9 is optimized via conventional CAD tools. Fig. 9(a) shows the final gate-level implementation of f_9. Notice the shared OR gate $(r_3 \vee r_4)$, which is used in both f_{c1} and f_{c2}. The inputs x_1 and x_2 must be fed with independent

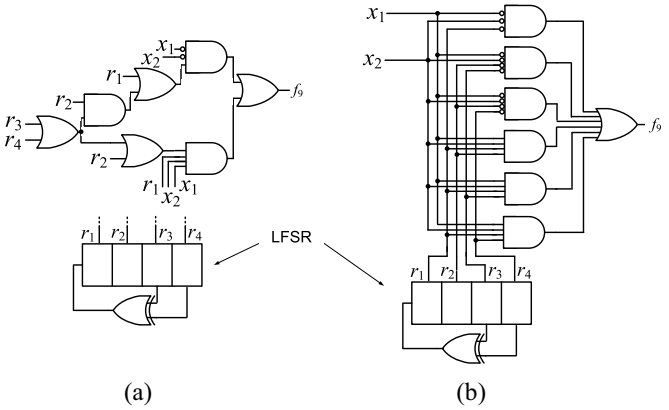


Fig. 9. Stochastic implementation of $\hat{F}_9(X) = 0.4375 - 0.25X - 0.5625X^2$ obtained by (a) STRAUSS and (b) the algorithm of [3].

Procedure *APS*($\hat{F}$) -- Given a polynomial $\hat{F}$, find an asymmetric -- multilinear polynomial of near-optimal cost

Step 1. Convert $\hat{F}$ to a symmetric multilinear polynomial $\hat{P}$

Step 2. Apply the inverse Fourier transform to $\hat{P}$ and obtain a symmetric truth-table $\vec{f}$

Step 3. Pick a group of symmetric elements from $\vec{f}$, and replace them with new elements, all of which, except for at most one, must be +1 or −1

Step 4. Try all the possible orderings of the current group of elements, and select one of lowest cost

Step 5. Repeat Steps 3 and 4 for all elements of $\vec{f}$

Fig. 10. Summary of the asymmetric polynomial selection (APS) procedure.

bit-streams that carry the same SN X. These can be generated by using two-independent SNGs, or just by shifting one bit-stream in time and thus generating an independent copy of it [12]. The auxiliary inputs, on the other hand, must be supplied with pure random bit-streams. As shown in Fig. 9, we connect the auxiliary inputs to a 4-bit LFSR, which generates four independent random bit-streams [12]. Fig. 9(b) shows another SC implementation of $\hat{F}_9$ using the algorithm proposed in [3]. As can be seen, the circuit generated by STRAUSS yields a smaller area based on literal count. In the next section, we discuss further optimizations used in STRAUSS.

VI. Further Optimizations

Step 1 of STRAUSS involves converting the target function to a multilinear polynomial, which can be symmetric or asymmetric. The synthesis method of [3] only uses symmetric polynomials, but it is possible to further optimize the synthesized circuit by considering both symmetric and asymmetric polynomials.

We illustrate this with an example. Consider the target function

$$\hat{F}_{10}(X) = \frac{1}{2}\Big(X^3 + X\Big).$$

Converting $\hat{F}_{10}$ to symmetric multilinear form, we get

$$\hat{P}_{10}(X_1, X_2, X_3) = \frac{1}{6}(X_1 + X_2 + X_3) + \frac{1}{2}X_1X_2X_3$$

which yields the following TT after applying the inverse Fourier transform:

$$\vec{f}_{10} = \big[\,1 \;\; -1/3 \;\; -1/3 \;\; 1/3 \;\; -1/3 \;\; 1/3 \;\; 1/3 \;\; -1\,\big]^{\mathrm{T}}.$$

Since this has elements other than 1 and −1, namely, 1/3 and −1/3, multiple-constant SN generation circuitry is required. However, if we choose the following asymmetric multilinear polynomial in the first step of STRAUSS, then:

$$\hat{P}'_{10}(X_1, X_2, X_3) = \frac{1}{2}(X_1 + X_2 - X_3) + \frac{1}{2}X_1X_2X_3.$$

The inverse Fourier transform produces

$$\vec{f}'_{10} = \big[\,1 \;\; 1 \;\; -1 \;\; 1 \;\; -1 \;\; 1 \;\; -1 \;\; -1\,\big]^{T}$$

which is simply the BF $f'_{10}(x_1, x_2, x_3) = x_1x_2 \vee x_1\bar{x}_3 \vee x_2\bar{x}_3$, and requires no constant generation circuitry. This means that significant cost savings may be possible if asymmetric polynomials are considered.

A given SC-implementable polynomial can be mapped to many different asymmetric multilinear polynomials with the same SC behavior, some of which may be SC-unimplementable. To distinguish between them, we need to apply the inverse Fourier transform (Step 2 of STRAUSS) and check if all the elements of the TT are in the interval $[-1, +1]$. However, applying the transform to all possible polynomials is very time-consuming, making this approach infeasible. So STRAUSS uses a different approach which is discussed below. An overview of this asymmetric polynomial selection (APS) algorithm is given in Fig. 10.

Given a target polynomial, we start with a symmetric multilinear polynomial—there is only one such polynomial—and use the inverse Fourier transform to obtain a symmetric TT (STT). Because of its symmetry, the STT includes repeated elements. We then modify the repeated elements to obtain an asymmetric TT of better cost. If we keep the new elements in the $[-1, +1]$ interval, the newly obtained TT remains SC-implementable. And as long as we keep the average of the new elements the same as that of the old elements, the new TT will have the same SC behavior as the STT.

As an example, consider a generic three-input STT

$$\vec{f}_{11} = \big[\,c_1 \;\; c_2 \;\; c_2 \;\; c_3 \;\; c_2 \;\; c_3 \;\; c_3 \;\; c_4\,\big]^{\mathrm{T}}$$

which has at most four distinct elements $c_1, \ldots, c_4$. We can replace the three c_2's with three new elements, c'_2, c''_2, and c'''_2. If the new elements are within the interval $[-1, +1]$, and if $c'_2 + c''_2 + c'''_2 = 3c_2$, then the new TT

$$\vec{f}'_{11} = \big[\,c_1 \;\; c'_2 \;\; c''_2 \;\; c_3 \;\; c'''_2 \;\; c_3 \;\; c_3 \;\; c_4\,\big]^{\mathrm{T}}$$

will have the same SC behavior as $\vec{f}_{11}$. Since all the inputs of the circuit have the same value X, the probabilities of getting any of the c_2 elements in $\vec{f}_{11}$ will be the same. So by changing the elements to c'_2, c''_2, and c'''_2 in $\vec{f}'_{11}$ and keeping the average the same as before (by assigning $c'_2 + c''_2 + c'''_2 = 3c_2$), the SC behavior of the TT remains unchanged. Similarly, the three c_3's can also be replaced with new elements. The choice of the

new elements directly affects the cost of the new TT. Elements such as $+1$ and -1 are desirable because they can be generated at no cost, while other elements require constant generation circuitry.

For example, consider the following STT:

$$\vec{f}_{10} = \begin{bmatrix} 1 & -1/3 & -1/3 & 1/3 & -1/3 & 1/3 & 1/3 & -1 \end{bmatrix}^{\mathrm{T}}$$

which has three $-1/3$s and three 1/3s. As shown earlier, we can replace these elements with new elements and obtain

$$\vec{f}'_{10} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$

which has the same SC behavior. Notice that, the three $-1/3$s are replaced with two -1s and one $+1$, and the three 1/3s are replaced with two $+1$s and one -1. Another choice of TT with the same behavior is

$$\vec{f}''_{10} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1/3 & 1/3 & 0 & -1 \end{bmatrix}^{\mathrm{T}}$$

but it clearly has a higher cost because it requires several constant generation circuits, while $\vec{f}'_{10}$ requires none. It can be shown that given a set of symmetric elements, we can always find a new set of elements with the same average, all of which, except for at most one, are $+1$s or -1s.

Assume we have a set of k symmetric elements of value c. If $c > 0$, then, we can replace the set with one $+1$ element and $k-1$ new elements c' of value $(kc-1)/(k-1)$. The new set has the same average as the old set

$$+1 + (k-1)c' = 1 + (k-1)\frac{kc-1}{k-1} = kc.$$

Similarly, if $c < 0$, we can replace the set with one -1 and $k-1$ new elements c' of value $(kc+1)/(k-1)$. By repeating this process on the new elements of the set, we obtain a set that has at least $k-1$ elements of value $+1$ or -1.

Another factor affecting cost is the order of the new elements in the TT. For example, the two $+1$s and one -1 replacing $+1/3$ can appear in three possible orders: $[+1+1-1]$, $[+1-1+1]$, and $[-1+1+1]$. Similarly, the new elements replacing $-1/3$s can also be reordered. So the TT $\vec{f}'_{10}$ can have nine different orderings

$$\begin{bmatrix} +1 & +1 & -1 & +1 & -1 & +1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & +1 & -1 & +1 & -1 & -1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & +1 & -1 & -1 & -1 & +1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & +1 & +1 & -1 & -1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & +1 & +1 & -1 & -1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & +1 & -1 & -1 & -1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & -1 & +1 & +1 & +1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & -1 & +1 & +1 & -1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}$$
$$\begin{bmatrix} +1 & -1 & -1 & -1 & +1 & +1 & +1 & -1 \end{bmatrix}^{\mathrm{T}}.$$

The number of different ways to order the TTs grows exponentially with the number of inputs, so searching among all of them is not possible for very large circuits. For such cases, STRAUSS has a greedy search heuristic that usually finds a good ordering. The heuristic starts from the STT and selects a group of repeated elements (say c_2 in $\vec{f}_{11}$) and finds new elements (c'_2, c''_2, and c'''_2) to replace them. Then it tries all the possible orderings of this group, and selects one with the minimum cost. The algorithm proceeds to the next group of repeated elements (c_3 in $\vec{f}_{11}$). In doing so, the algorithm only examines $3+3=6$ orderings out of the $3\times 3=9$ possible orderings. Thus, the search becomes feasible for larger circuits. Like most heuristics, this method does not always find an optimal solution, but our experiments show that a near-optimum polynomial is usually found. We also observed that in most cases, there are multiple optimum and many near-optimum polynomials, which favors the heuristic search.

It is worth noting that employing asymmetric polynomials reduces the precision needed in the constant generation circuit, and hence, can lead to cost savings. A good example is the function $\hat{F}_{10}$ discussed earlier in this section. A symmetric TT for this function is

$$\vec{f}_{10} = \begin{bmatrix} 1 & -1/3 & -1/3 & 1/3 & -1/3 & 1/3 & 1/3 & -1 \end{bmatrix}^{\mathrm{T}}$$

which requires constant generation circuitry for 1/3 and $-1/3$. When applying the SCG procedure (Fig. 7) to $\vec{f}_{10}$, one has to choose a precision m, and round the numbers 1/3 and $-1/3$ to the precision closest to m. However, an asymmetric implementation of $\hat{F}_{10}$, namely

$$\vec{f}'_{10} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}^{\mathrm{T}}$$

needs no constant generation at all. It is as if the constants 1/3 and $-1/3$ are implemented with unlimited precision and at no cost. We end this section by a complete example involving a multivariate target function.

Example 7: Consider the target function $\hat{F}_{11}(X,Y,Z) = 1/64(63 + X^2 + Y^2 + Z^2 - X^2Y^2 - X^2Z^2 - Y^2Z^2 + X^2Y^2Z^2)$. Since the maximum degree of the polynomial is two, we will need two-independent copies of each input. The corresponding symmetric multilinear polynomial of $\hat{F}_{11}$ is

$$\begin{aligned}\hat{P}_{11}&(X_1,X_2,Y_1,Y_2,Z_1,Z_2)\\ &= \frac{1}{64}(63 + X_1X_2 + Y_1Y_2 + Z_1Z_2 - X_1X_2Y_1Y_2\\ &\qquad - X_1X_2Z_1Z_2 - Y_1Y_2Z_1Z_2 + X_1X_2Y_1Y_2Z_1Z_2).\end{aligned}$$

By applying the inverse Fourier transform on $\hat{P}_{11}$ (Step 2 of STRAUSS), we obtain the symmetric TT vector

$$\begin{aligned}\vec{f}_{11} = \Big[&1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \tfrac{7}{8}\ \tfrac{7}{8}\\ &1\ 1 \ldots\ \tfrac{7}{8}\ \tfrac{7}{8}\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \tfrac{7}{8}\ \tfrac{7}{8}\ 1\ 1\ \tfrac{7}{8}\ \tfrac{7}{8}\\ &1\ 1\ 1\ 1\ 1\ 1\ 1 \ldots 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\Big]^{\mathrm{T}}.\end{aligned}$$

Steps 3 and 4 of STRAUSS yield the following BF:

$$\begin{aligned}f_{11}&(x_1,x_2,y_1,y_2,z_1,z_2,r_1,r_2,r_3,r_4)\\ &= ((x_1 \oplus x_2) \vee (y_1 \oplus y_2) \vee (z_1 \oplus z_2))\\ &\quad \wedge (r_1 \wedge r_2 \wedge r_3 \wedge r_4)\end{aligned}$$

which requires four auxiliary inputs. A gate-level implementation of f_{11} is shown in Fig. 11(a).

It is possible to further optimize this circuit by applying the APS procedure to $\hat{F}_{11}$. In the symmetric TT shown above, there are eight symmetric elements with value 7/8.

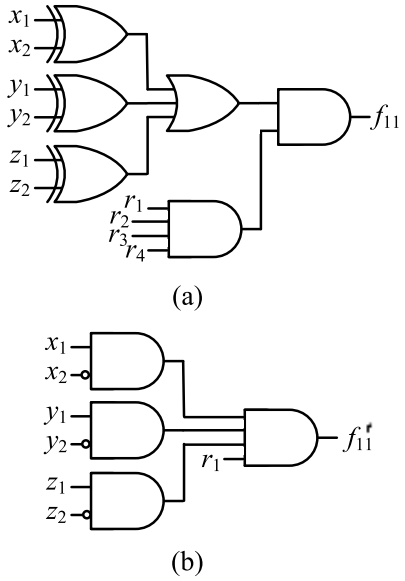


Fig. 11. Stochastic implementations for $\hat{F}_{11}$ of Example 7. (a) Symmetric. (b) Asymmetric.

APS identifies this symmetric group and replaces it with a group of seven elements of value 1 and one element of value 0. After trying different possible orderings, APS returns the following asymmetric TT (which is one of the eight possible optimal TTs):

$$\vec{f}'_{11} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \ldots\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \ldots\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^{\mathrm{T}}.$$

This yields the BF

$$f'_{11}(x_1, x_2, y_1, y_2, z_1, z_2, r_1) = x_1 \wedge \bar{x}_2 \wedge y_1 \wedge \bar{y}_2 \wedge z_1 \wedge \bar{z}_2 \wedge r_1$$

which requires only one auxiliary input. An implementation of f'_{11} is shown in Fig. 11(b). The asymmetric polynomial $\hat{P}'_{11}$ that corresponds to f'_{11} has more terms than the symmetric polynomial $\hat{P}_{11}$ but, surprisingly, it yields a lower-cost implementation. Note that, the APS procedure performs transformations on the TT only and does not deal with different (and in some cases complicated) polynomial terms. The first 16 terms of the asymmetric polynomial $\hat{P}'_{11}$ are shown below for illustration purposes only

$$\begin{aligned}\hat{P}'_{11}&(X_1, X_2, Y_1, Y_2, Z_1, Z_2)\\ &= \frac{1}{64}(63 - X_2 + X_1 + X_1X_2 - Y_2 - X_2Y_2 + X_1Y_2\\ &\quad + X_1X_2Y_2 + Y_1 + X_2Y_1 - X_1Y_1 - X_1X_2Y_1\\ &\quad + Y_1Y_2 + X_2Y_1Y_2 - X_1Y_1Y_2 - X_1X_2Y_1Y_2 + \cdots).\end{aligned}$$

VII. Related Work and Experimental Results

Very little previous research has addressed the systematic design of SC circuits. Qian and Riedel [22] have developed a nonspectral method of implementing a single-variable SC target function $\hat{F}(X)$ defined in UP format. They convert $\hat{F}(X)$ to a Bernstein polynomial of the form

$$\hat{P}(X) = \sum_{i=0}^{n} C_i \binom{n}{i} X^i (1-X)^{n-i} \tag{9}$$

in which the C_i's are constant coefficients. This polynomial is then mapped to a specific style of logic circuit termed reconfigurable SC architecture (ReSC) [26]. It consists of an n-input adder that implements the terms $\binom{n}{i}X^i(1-X)^{n-i}$ (called the Bernstein terms) and a multiplexer that selects the C_i coefficients. Fig. 12(a) shows the ReSC implementation of (9). There are n independent inputs (x_i's) to the adder representing the variable X, and $n+1$ inputs to the multiplexer (c_i's) that represent the coefficients C_i in (9). The probability of a number k at the output of the adder is equal to $\binom{n}{k}X^k(1-X)^{n-k}$, i.e., the kth Bernstein term. Thus, the probability of having a 1 at f is equal to the probability getting a 0 at the adder and a 1 at c_0, plus the probability of getting a 1 at the adder and a 1 at c_1, plus ..., plus the probability of getting n at the adder and a 1 at c_n. These probabilities can be expressed as

$$F(X) = \sum_{i=0}^{n} C_i \binom{n}{i} X^i (1-X)^{n-i}$$

which is the same as (9). Note that, the x_i inputs are independent SNs representing the number X, similar to the x_1 and x_2 inputs of Fig. 9. This structure requires n SNGs to generate the x_i's and $n+1$ SNGs to generate the c_i's, which entails a significant area cost. Similar to STRAUSS, this design approach can be extended to multivariate functions [24], and to the BP format.

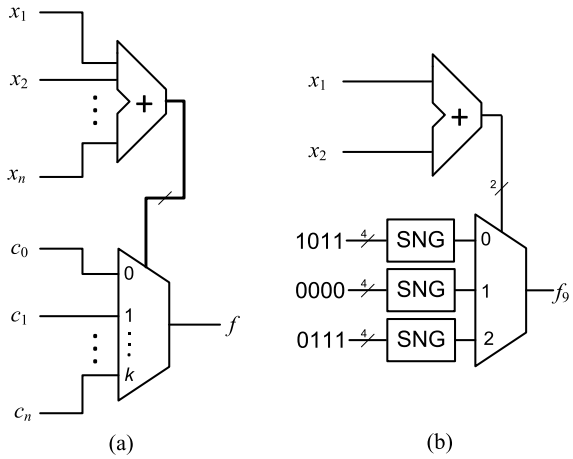


Fig. 12. (a) ReSC architecture proposed in [26]. (b) Implementation of $\hat{F}_9(X)$ from Example 6 using this architecture.

We now show that the method of [26] can be reinterpreted in terms of a spectral transform, a "Bernstein transform" $\mathcal{B}$ with a basis different to that of the Fourier transform $\mathcal{F}$.

Consider the two-variable BF $f(x_1, x_2)$. We define a new spectral basis for it is as follows: $B_0 = 0.25(1+X_1)(1+X_2)$, $B_1 = 0.25(1+X_1)(1-X_2)$, $B_2 = 0.25(1-X_1)(1+X_2)$, and $B_3 = 0.25(1-X_1)(1-X_2)$. (Note the difference between this and the spectral basis of the two-variable Fourier transform,

TABLE II
COMPARISON BETWEEN THE PROPOSED SYNTHESIS METHOD STRAUSS AND THOSE OF [3] AND [26]; AREA IS IN UNIT CELLS FROM A GENERIC LIBRARY AND INCLUDES THE LFSRS AND SNGS USED FOR CONSTANT GENERATION

Target design	Qian et al. [26]	Alaghi and Hayes [3]	STRAUSS
$\hat{F}_9$ from Example 6	113	70	70
$\hat{F}_{10}(X) = \frac{1}{2}(X^3 + X)$	174	112	10
Gamma correction [26]	962	314	276
Average of 10 random target functions	426	151	125

which is 1, X_1, X_2, and X_1X_2.) The resulting transform $\mathcal{B}$ of f can then be written as

$$F(X_1, X_2) = \sum_{i=0}^{3} C_i B_i. \tag{10}$$

This corresponds to the same multilinear expression generated by the Fourier transform. To see this, expand (10) thus

$$\begin{aligned} F(X_1, X_2) = 0.25\big(&C_0(1+X_1)(1+X_2) + C_1(1+X_1)(1-X_2) \\ &+ C_2(1-X_1)(1+X_2) \\ &+ C_3(1-X_1)(1-X_2)\big) \\ = 0.25\big(&C_0 + C_1 + C_2 + C_3 \\ &+ (C_0 - C_1 + C_2 - C_3)X_2 \\ &+ (C_0 + C_1 - C_2 - C_3)X_1 \\ &+ (C_0 - C_1 - C_2 + C_3)X_1X_2\big) \end{aligned}$$

which is the multilinear polynomial produced by the Fourier transform, as the following equation demonstrates:

$$\begin{aligned} \vec{F} &= \frac{1}{4}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \times \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} \\ &= \frac{1}{4}\begin{bmatrix} C_0 + C_1 + C_2 + C_3 \\ C_0 - C_1 + C_2 - C_3 \\ C_0 + C_1 - C_2 - C_3 \\ C_0 - C_1 - C_2 + C_3 \end{bmatrix}. \end{aligned}$$

Despite their similarities, the circuits synthesized by STRAUSS are quite different from those designed via the ReSC architecture of [26]. Most importantly, ReSC does not benefit from asymmetric polynomials or constant input sharing. To illustrate this, we implemented the function $\hat{F}_9$ of Example 6 using ReSC. Fig. 12(b) shows the result (adjusted for the IBP format). Besides its adder and multiplexer, this design contains three SNGs, each consisting of a 4-bit comparator and a 4-bit random number generator (or LFSR), as in Fig. 3. Note, however, that this circuit can be optimized using standard combinational techniques; the middle SNG, for instance produces a 0, and so can be removed from the circuit.

To attempt a fair comparison, we used the Berkeley SIS synthesis tool [28] to optimize this design and those of Fig. 9 and to map them to a generic library of gates. The library includes all the elementary gates and their relative area cost in terms of unit cells in a 0.35μm technology. Table II compares the three implementations, along with several other representative circuits, with area cost reported in terms of unit cells. The run-time of the circuits depend on the desired accuracy of the user. In general, longer run-time leads to better accuracy. The circuits compared in Table II achieve the same level of accuracy for a given run-time. These results show that STRAUSS synthesizes circuits that are significantly smaller than those designed by the techniques of [3] and [26]. The area reported in Table II does not include the conversion circuits that may be required for the primary inputs and outputs.

VIII. CONCLUSION

SC has re-emerged recently as an important technology for certain applications needing low-cost massive parallelism, or related features like very small circuit size or low power. Although SC has been recognized for many years, its underlying theory is not well-developed. We have shown here that well-defined transforms linking the Boolean and the spectral domains exist, which provide fundamental theoretical insights into SC behavior. We have also successfully applied spectral transforms to the design of circuits in a way that naturally accommodates the most useful SN formats. Furthermore, we have presented a novel and general synthesis technique STRAUSS for combinational circuit synthesis. Comparing this paper to the major existing SC design method [26], we found that the results generated by STRAUSS can lead to significant cost savings.

APPENDIX

A. Proof of Theorem 1

We provide a proof by induction on n, the number of variables of f. Suppose $n = 1$, so f is a single-variable BF. Using the Boole–Shannon expansion theorem, we can write $f(x) = c_0\bar{x} \oplus c_1 x$ where $c_0 = f(0)$ and $c_1 = f(1)$ are constants. Now apply a bit-stream X of length N to the input x of f. If this bit-stream contains N_1 1s and N_0 0s, it represents the number $(N_0 - N_1)/N$ in IBP format. The function f, therefore, outputs another bit-stream with N_1c_1's and N_1c_0's representing the IBP number $(N_0C_0 + N_1C_1)/N$, where $C_0 = 1 - 2c_0$ and $C_1 = 1 - 2c_1$. Hence, for an arbitrary SN $X = 1 - 2N_0/N$, the output number is

$$\hat{F}(X) = \frac{C_0 + C_1}{2} + \frac{C_0 - C_1}{2}X \tag{11}$$

which describes the SC behavior of f in the IBP format. Now the TT of f is $\vec{f} = \begin{bmatrix} C_0 \\ C_1 \end{bmatrix}$, so its Fourier transform is

$$\vec{F} = \frac{1}{2}H_1 \times \vec{f} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} C_0 + C_1 \\ C_0 - C_1 \end{bmatrix}.$$

This corresponds to the polynomial

$$F(X) = \frac{1}{2}\big[C_0 + C_1 + (C_0 - C_1)X\big] \tag{12}$$

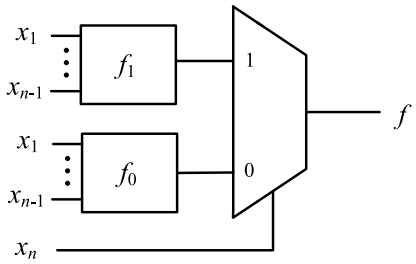


Fig. 13. Circuit illustrating the application of Boole–Shannon expansion to the function f.

which is the same as (11), so the theorem holds for single-variable functions.

Now as the induction hypothesis, assume the theorem holds for all functions of up to $n-1$ variables. We want to show that it also holds for the n-variable function $f(x_1, \ldots, x_n)$. Again using Boole–Shannon expansion, we can write $f(x_1, \ldots, x_n) = f_0 \cdot \bar{x}_n \oplus f_1 \cdot x_n$, where $f_0 = f(x_1, \ldots, x_{n-1}, 0)$ and $f_1 = f(x_1, \ldots, x_{n-1}, 1)$ are functions of $n-1$ or fewer variables. Fig. 13 illustrates how f is decomposed in this way. Thus, f's SC behavior is, in terms of the behavior of a 2-to-1 multiplexer (Fig. 13)

$$\hat{F}(X_1, \ldots, X_n) = \frac{\hat{F}_0 + \hat{F}_1}{2} + \frac{\hat{F}_0 - \hat{F}_1}{2} X_n \qquad (13)$$

where $\hat{F}_0$ and $\hat{F}_1$ denote the SC behavior of f_0 and f_1, respectively.

We can express the TT of f in terms of the TTs of f_0 and f_1 as $\vec{f} = \begin{bmatrix} \vec{f}_0 \\ \vec{f}_1 \end{bmatrix}$. Accordingly, we can decompose the Fourier transform calculation (5) into

$$\vec{F} = \frac{1}{2^n} H_n \times \vec{f} = \frac{1}{2} H_1 \begin{bmatrix} \frac{1}{2^{n-1}} H_{n-1} \times \vec{f}_0 \\ \frac{1}{2^{n-1}} H_{n-1} \times \vec{f}_1 \end{bmatrix} = \frac{1}{2} H_1 \begin{bmatrix} \vec{F}_0 \\ \vec{F}_1 \end{bmatrix}$$

where $\vec{F}_0$ and $\vec{F}_1$ are the Fourier transforms of f_0 and f_1, respectively. The resulting polynomial is

$$F(X_1, \ldots, X_n) = \frac{1}{2}[F_0 + F_1 + (F_0 - F_1) X_n] \qquad (14)$$

which is the same as (13). Hence $\hat{F} = F$, and from the principle of induction we conclude that the theorem holds. ■

REFERENCES

[1] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:12, 2013.

[2] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. Design Autom. Conf.*, Austin, TX, USA, 2013, pp. 1–6.

[3] A. Alaghi and J. P. Hayes, "A spectral transform approach to stochastic circuits," *Proc. Int. Conf. Comput. Design*, Montreal, QC, Canada, 2012, pp. 315–312.

[4] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *Proc. Int. Conf. Comput. Design*, Asheville, NC, USA, 2013, pp. 39–46.

[5] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," in *Proc. Design Autom. Conf.*, Dallas, TX, USA, 1993, pp. 54–60.

[6] B. R. Gaines, "Stochastic computing," in *Proc. Spring Joint Comput. Conf. (AFIPS)*, Atlantic City, NJ, USA, 1967, pp. 149–156.

[7] B. R. Gaines, "Stochastic computing systems," *Advances in Information Systems Science*, vol. 2. New York, NY, USA: Springer, 1969, pp. 37–172.

[8] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Pacific Grove, CA, USA, 2005, pp. 713–717.

[9] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Boston, MA, USA: Kluwer Academic, 1996.

[10] S. M. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques for Digital Logic*. London, U.K.: Academic, 1985.

[11] *Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks*, IEEE Standard 802.11n, 2009. [Online]. Available: http://standards.ieee.org

[12] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Inf. Theory*, vol. 40, no. 3, pp. 716–720, May 1994.

[13] M. G. Karpovsky, R. S. Stankovic, and J. T. Astola, *Spectral Logic and its Applications for the Design of Digital Devices*. Hoboken, NJ, USA: Wiley-Interscience, 2008.

[14] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams: Digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[15] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Trans. Comput.*, vol. 63, no. 6, pp. 1473–1485, Jun. 2014.

[16] P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," in *Proc. Int. Conf. Comput. Design*, Montreal, QC, Canada, 2012, pp. 303–308.

[17] *MATLAB and Statistics Toolbox Release 2012b*, The MathWorks, Inc., Natick, MA, USA, 2012.

[18] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov. 2011.

[19] R. O'Donnell, "Some topics in the analysis of Boolean functions," in *Proc. ACM STOC Conf.*, Victoria, BC, Canada, 2008, pp. 569–578.

[20] W. J. Poppelbaum, "Statistical processors," in *Advances in Computers*, vol. 14, M. Rubinoff and M. C. Yovits, Eds. New York, NY, USA: Academic Press, 1976, pp. 187–230.

[21] W. J. Poppelbaum, C. Afuso, and J. W. Esch, "Stochastic computing elements and systems," in *Proc. Fall Joint Computer Conf. (AFIPS)*, Anaheim, CA, USA, 1967, pp. 635–644.

[22] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proc. DAC*, Anaheim, CA, USA, 2008, pp. 648–653.

[23] W. Qian and M. D. Riedel, "Two-level logic synthesis for probabilistic computation," in *Proc. Int. Workshop Logic Syn.*, Irvine, CA, USA, 2010, pp. 1–9.

[24] W. Qian and M. D. Riedel, "Uniform approximation and Bernstein polynomials with coefficients in the unit interval," *Eur. J. Combin.*, vol. 32, no. 3, pp. 448–463, Apr. 2011.

[25] W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, "Transforming probabilities with combinational logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 9, pp. 1279–1292, Sep. 2011.

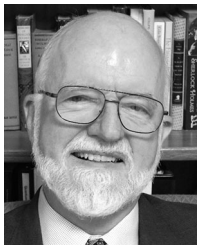
[26] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[27] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, "Stochastic functions using sequential logic," in *Proc. Int. Conf. Comput. Design*, Asheville, NC, USA, 2013, pp. 507–510.

[28] E. M. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Electron. Res. Lab., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/ERL M92/41, 1992.

Armin Alaghi (S'06) received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer architecture from the University of Tehran, Tehran, Iran, in 2006 and 2009, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA.

From 2005 to 2009, he was a Research Assistant with the Field-Programmable Gate-Array (FPGA) Laboratory, University of Tehran, and the Computer-Aided Design Laboratory, University of Tehran, where he was involved in FPGA and network-on-chip testing. Since 2009, he has been with the Advanced Computer Architecture Laboratory, University of Michigan. His current research interests include digital system design, embedded systems, very large-scale integration circuits, computer architecture, and electronic design automation.



John P. Hayes (S'67–M'70–SM'81–F'85–LF'10) received the B.E. degree from the National University of Ireland, Dublin, Ireland, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign, USA, all in electrical engineering.

He participated in the design of the ILLIAC III computer with the University of Illinois. In 1970, he joined the Operations Research Group, Shell Benelux Computing Center, The Hague, The Netherlands, where he was involved in mathematical programming and software development. From 1972 to 1982, he was a Faculty Member with the Departments of Electrical Engineering-Systems and Computer Science, University of Southern California, Los Angeles, CA, USA. Since 1982, he has been with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA, where he holds the Claude E. Shannon Chair in Engineering Science. His current research interests include computer-aided design, verification, and testing, very large-scale integration circuits, computer architecture, and unconventional computing systems. He has authored over 300 technical papers, several patents, and seven books, including *Computer Architecture and Organization* (3rd ed., 1998), *Quantum Circuit Simulation* (with G. F. Viamontes and I. L. Markov, 2009), and *Design, Analysis and Test of Logic Circuits Under Uncertainty* (with S. Krishnaswamy and I. L. Markov, 2012).

Prof. Hayes was a recipient of the University of Michigan's Distinguished Faculty Achievement Award in 1999, the Alexander von Humboldt Foundation's Research Award in 2004, the IEEE Lifetime Contribution Medal for outstanding contributions to test technology in 2013, and the ACM Pioneering Achievement Award for contributions to logic design, fault tolerant computing, and testing in 2014. He was the Founding Director of the Advanced Computer Architecture Laboratory, University of Michigan. He has served as an Editor for various technical journals, such as the *Communications of the ACM* and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He was elected as a fellow of ACM in 2001.