A Spectral Transform Approach to Stochastic Circuits

Armin Alaghi and John P. Hayes
Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI, 48109, USA
{alaghi, jhayes}@eecs.umich.edu

***Abstract*—Stochastic computing (SC) processes data in the form of long pseudo-random bit-streams denoting probabilities. Its key advantages are simple computational elements and high soft-error tolerance. Recent technology developments have revealed important new SC applications such as image processing and LDPC decoding. Despite its long history, SC still lacks a comprehensive design methodology; existing methods tend to be ad hoc and limited to a few arithmetic functions. We demonstrate a fundamental relation between stochastic circuits and spectral transforms. Based on this, we propose a transform approach to the analysis and synthesis of SC circuits. We illustrate the approach for a variety of basic combinational SC design problems, and show that the area cost associated with stochastic number generation can be significantly reduced.**

Keywords- Design methodology, logic synthesis, probabilistic methods, stochastic computing;

I. INTRODUCTION

Stochastic computing (SC) is an unconventional computational technique introduced by Gaines [2] and Poppelbaum et al. [14] that processes data in the form of probabilities represented by bit-streams. It employs simple logic circuits to perform complex arithmetic operations. For instance, multiplication can be implemented by a single AND gate. A bit-stream S containing N_1 1's and N_0 0's denotes the number $p = N_1/(N_1 + N_0)$. Since p lies in the real-number interval [0,1], it is interpreted as the probability of a 1 appearing at a randomly chosen location in S or, equivalently, as the probability that S outputs a 1 at a randomly chosen time.

Figure 1 shows a representative stochastic circuit called an *update node*, which is used in decoders for low-density parity check (LDPC) codes [11]. The two inputs p_1 and p_2 represent 2/8 and 6/8, respectively. The update node can be shown to compute

$$p_3 = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} \qquad (1)$$

The accuracy and precision of this computation depend on the length and randomness of the input bit-streams. Observe that the update node is a logic circuit defined by Boolean algebra, but its SC behavior, as given by (1), is basically analog arithmetic over rational or real numbers.

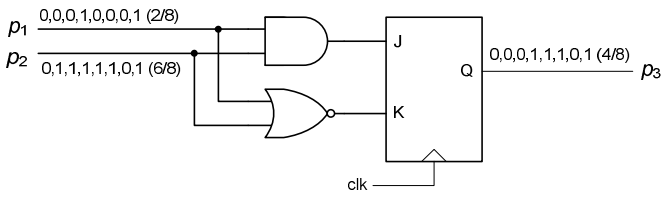

Figure 1. Stochastic update node used in LDPC decoders.

As Figure 1 suggests, SC is suitable for applications that require large numbers of relatively low-precision operations. In addition, SC's probabilistic aspect makes it tolerant of errors of the soft, bit-flip type. Recent technology trends, such as the need for massive 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, Qian et al. [15] show that SC can outperform binary computing in some image processing tasks. A recently discovered use of SC is in LDPC decoding [4]; LDPC codes are part of the IEEE WiFi standard [6]. Naderi et al. [11] have implemented an SC LDPC decoder that has comparable performance to conventional designs, but uses less chip area. 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 are ad hoc in nature and tailored to very specific applications. Some are based on assembling circuits from a fixed and limited library of SC components, e.g., multipliers, adders and multiplexers. Synthesis approaches have been proposed for a few classes of combinational [15] and sequential [9][10] circuits, e.g., arithmetic function generators circuits and in some cases only positive numbers are used. Despite the fact that complete SC systems such as LDPC decoders [11] and even general-purpose machines [13], have been implemented, a comprehensive SC design methodology has yet to appear.

This paper reveals a fundamental relation between SC circuits and spectral transforms like the Fourier transform [5][8]. 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 quickly be found.

The spectral transforms of interest in this work map Boolean functions from the logic domain to the domain of real numbers. We show that they lead to a unique multi-linear representation of a given Boolean function, which defines its SC behavior. Based on this, we derive methods of analyzing and synthesizing combinational stochastic circuits. Our approach applies to both positive and negative stochastic numbers. Furthermore, we show how it can synthesize circuits with lower area than circuits generated by previous methods.

The paper is organized as follows. Section II gives an overview of stochastic numbers and their generation. Section III discusses spectral transforms and their relation to SC. A systematic synthesis method for combinational stochastic cir-

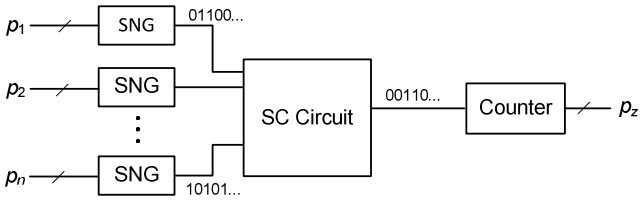

Figure 2. SC circuit surrounded by number conversion blocks.

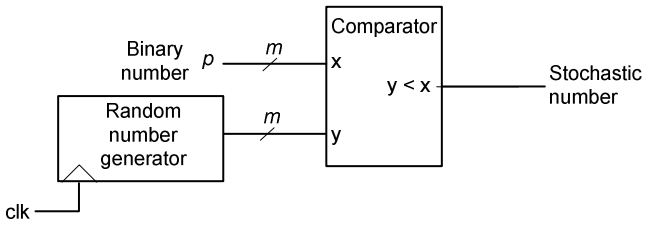

Figure 3. Stochastic number generator (SNG).

cuits is proposed in Section IV. Section V compares our approach to previous work, and Section VI gives conclusions.

II. Stochastic Numbers

This section reviews stochastic numbers (SNs), and introduces the terminology and notation used in the paper.

To combine an SC circuit with conventional logic, peripheral circuits are needed to convert between binary and stochastic number formats, as indicated in Figure 2. Converting an SN to ordinary binary form requires counting the number of 1s and/or 0s in the bit-stream, and hence a counter suffices to implement it. Converting from binary to SN form requires a more complex stochastic number generator (SNG). Figure 3 shows a typical SNG that converts a binary number p to stochastic form. It compares p with a uniform random bit-stream m produced by a pseudo-random number generator, such as a linear feedback shift-register (LFSR) [7]. On being clocked, the SNG outputs a bit-stream representing p, one bit per clock cycle. As reported in [15], SNGs can consume as much as 80% of an SC circuit's total area.

The accuracy of SC depends on the length of the bit-streams used and the amount of dependence or correlation among the input bit-streams. For instance, if the input bit-streams applied to the circuit of Figure 1 are correlated, say one is the inverse of the other, then the output will remain at 0 or 1 depending on its initial state; this implies that the output SN does not follow (1). Formally [7], two bit-streams S_1 and S_2 of length N are *independent* if

$$\frac{1}{N}\sum_{i=1}^{N} S_1(i) \times \frac{1}{N}\sum_{i=1}^{N} S_2(i) = \frac{1}{N}\sum_{i=1}^{N} S_1(i)\, S_2(i)$$

where $S_1(i)$ and $S_2(i)$ are the i-th bits of S_1 and S_2, respectively. The left-hand-side of the equation denotes the product of the probabilities of S_1 and S_2, while the right-hand-side denotes the probability of the bit-stream obtained by multiplying the elements of S_1 and S_2 together. More generally, a set of n bit-streams $S_1, S_2, \ldots, S_n$ is called an *independent set* if for every selection of bit-streams from the set (any two, any three, etc.), the product of their individual probabilities is equal to the probability of the product of the bit-streams. We make the usual assumption that the bit-streams applied to our stochastic circuits form an independent set. A detailed discussion of accuracy and correlation issues can be found in [1].

Table 1. Interpretations of a bit-stream of length N with N_1 1's and N_0 0's.

SN type	Number value	Interval	Relation to unipolar number p
Unipolar (UP)	N_1/N	[0,1]	p
Bipolar (BP)	$(N_1 - N_0)/N$	[–1,1]	$2p - 1$
Inverted bipolar (IBP)	$(N_0 - N_1)/N$	[–1,1]	$1 - 2p$

The SNs discussed so far are positive numbers defined on the real interval [0,1] and are called *unipolar* (UP) [2]. It is possible to interpret a bit-stream in other ways to cover larger intervals. Table 1 shows how a bit-stream of length N with N_1 1's and N_0 0's is interpreted in three different SN domains. *Bipolar* (BP) SNs are defined on [–1,1] and are convenient for representing positive and negative numbers. We introduce a new SN format called *inverted bipolar* (IBP) which is the inverse of BP. The IBP and BP formats are equivalent in that circuits in one format are easily converted to the other. However, IBP is more convenient to use with spectral transforms, where the Boolean 0 and 1 values are normally 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 number formats, consider the bit-stream (0,1,1,0,1,0,1,1,0,1) of length 10 containing six 1's and four 0's. It represents UP number 0.6, BP number 0.2, and IBP number –0.2. SNs can be transformed between intervals using the relations shown in Table 1. Furthermore, SC functions defined over one interval can be mapped to another interval using the same relations. For instance, the function $Y = F(X)$ on the UP interval [0,1], maps to $Y = 1 - 2F((1 - X)/2)$ on the IBP interval [–1,1].

III. Spectral Transforms

An n-variable Boolean function (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 function. For example, if $n = 2$, $f(x_1,x_2)$ has the 4-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

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

This is the familiar sum-of-minterms expansion of f, where the c_i's are 0-1 coefficients that define f. For example, if $f(x_1,x_2) = x_1 \vee x_2'$, then $f(x_1, x_2) = m_0 \vee m_2 \vee m_3$, or in the column-vector form that we use later:

$$f(x_1, x_2) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \vee \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \vee \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}1\\0\\1\\1\end{bmatrix} \qquad (3)$$

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

As Table 1 indicates, in the SC context we must deal with real numbers ranging over intervals such as [0,1] and [−1,1]. So

given a BF f, we are interested in an equivalent real function $\hat{F}$ defined on $[-1,1]^n$ (or some other appropriate domain) that specifies the SC behavior of f. This function can be obtained by interpolating the TT values in the real domain via a multi-linear polynomial. For example, consider the TT vector in (3). By replacing 0's and 1's with +1's and −1's, respectively, we see that f produces the TT vector $[-1 \;\; 1 \;\; -1 \;\; -1]^T$ for inputs (x_1, x_2) = (1, 1), (1, −1), (−1, 1) and (−1, −1), respectively, in the IBP domain. These four discrete "minterm points" can be embedded in a continuous real-number function as follows:

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

Note that each term of the expression assumes the correct TT value 1 or −1 at each of the minterm points. On expanding this expression, we get

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

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 *multi-linear*. It denotes the stochastic behavior of the BF f in the IBP domain.

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

There are several spectral transforms of interest, but all lead to similar results. They perform a change of basis from the minterm space of f to a real-valued space in which +1/−1 replace 0/1. We employ the Fourier transform $\mathcal{F}$ [12] as it is both well-known and convenient. Spectral transforms have been considered previously in logic design and testing [5] [8], but their use has been severely limited by the fact that with typical values of n, it is impractical to deal with 2^n-dimensional spectra. The same limitation does *not* apply to SC, however, because the values of n tend to be small, e.g., $n = 2$ in Figure 1.

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

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

where $\vec{F}$ is the vector form of F denoting its spectral coefficients or *spectrum,* and H_n is the Hadamard matrix defined by

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

Equation (5) is evaluated using the rules of linear algebra over real numbers. In the case of the two-variable BF (3), we get

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

The result is the spectrum of f. The corresponding basis is

$$[1 \;\; 1 \;\; 1 \;\; 1]^T, [1 \;\; -1 \;\; 1 \;\; -1]^T, [1 \;\; 1, \;\; -1, \;\; -1]^T [1 \;\; -1 \;\; -1, \;\; 1]^T$$

which are the rows of H_2. These basis vectors resemble 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_1, x_2, and $x_1 \oplus x_2$, where $\oplus$ denotes XOR. Note that in the spectral domain, XOR is replaced by multiplication.

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

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

where the S_i's are the basis vectors 1, X_1, X, and X_1X_2, in the spectral domain, and the C_i's constitute the spectrum of f. Hence, for the running example, (6) becomes

$$F(X_1, X_2) = -0.5 - 0.5X_1 + 0.5X_2 - 0.5X_1X_2$$

This is a multi-linear polynomial that interpolates (matches) the original BF f at its four Boolean input coordinates (minterm points), namely (X_1,X_2) = (1,1), (−1,1), (1,−1), (−1,−1). 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 SC behavior. This leads to the following theorem.

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

Thus, given a combinational circuit or Boolean function, we can determine its SC behavior in the IBP domain by computing its Fourier transform. An interval conversion according to Table 1 is required to determine its behavior in other domains. Two examples illustrate this.

It is well-known that a 2-input XNOR gate implements multiplication in the SC bipolar domain [3]. So we would expect an XOR gate to serve as a multiplier in the IBP domain. We can verify this as follows. XOR has the TT vector $\vec{f} = [1 \;\; -1 \;\; -1 \;\; 1]^T$. Calculating its Fourier transform yields

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

so $F(X_1, X_2) = X_1X_2$, and according to Theorem 1, the IBP SC behavior of XOR is

$$\hat{F}(X_1, X_2) = F(X_1, X_2) = X_1 X_2$$

Similarly, a 2-input AND gate acts as a multiplier in the SC unipolar domain. In this case, $\vec{f} = [1 \quad 1 \quad 1 \quad -1]^{\mathrm{T}}$ and

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

Thus,

$$F(X_1, X_2) = 0.5(1 + X_1 + X_2 - X_1 X_2) \qquad (7)$$

Since we want the AND behavior in the UP domain, 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, respectively. Table 1 implies $X_1 = 1 - 2p_1$, $X_2 = 1 - 2p_2$, and $F = 1 - 2p_3$. Hence,

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

leading to the desired multiplication $p_3 = p_1 p_2$.

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. Noting that H_n is its own inverse, this may be calculated as follows:

$$\vec{f} = H_n \times \vec{F}$$

IV. Combinational Circuit Synthesis

The spectral transforms discussed so far have several useful applications in the SC context. Besides analyzing Boolean functions and extracting their SC behavior, they can be used to systematically design combinational and (as we will briefly illustrate) sequential SC circuits.

Consider the problem of synthesizing a combinational SC circuit to realize a given arithmetic function. We now present a procedure that accepts a target function $\hat{F}$ in polynomial form and synthesizes its combinational logic implementation. The main steps of the proposed synthesis method are listed in Table 2. We illustrate it with several examples.

Table 2. Proposed combinational synthesis method.

Step	Description
1	Format the target function $\hat{F}$ as a multi-linear polynomial $\hat{P}$.
2	Compute the inverse Fourier transform $\mathcal{F}^{-1}(\hat{P})$ to obtain $\vec{f}$.
3	Scale the elements of $\vec{f}$ to match the IBP interval, yielding $\vec{f}^*$.
4	If $\vec{f}^*$ contains elements besides +1 and -1, modify $\vec{f}^*$ to obtain $\vec{f}^{**}$; see Figure 4 for details.
5	Optimize $\vec{f}^{**}$ via standard combinational design procedures.

Example 1. Consider the problem of reverse engineering the IBP multiplier, so the given function is $\hat{F}(X, Y) = XY$. Since this is already has the desired multi-linear polynomial form, we can skip Step 1 and proceed to Step 2 where we use the inverse Fourier transform to obtain *f's* TT vector.

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

Modify ($\vec{f}^*$, *max_iterations*) **{**
//modify $\vec{f}^*$ to one with 1s and -1s only
repeat for *max_iterations* **{**
if every element t_i^* in $\vec{f}^*$ is 1 or −1 **then return** $\vec{f}^*$
else for each element t_i^* in $\vec{f}^*$ **{**
if $t_i^* < 0$ **then** $t_{2i-1}^{**} \leftarrow -1$, $t_{2i}^{**} \leftarrow 2t_i^* + 1$
else $t_{2i-1}^{**} \leftarrow +1$, $t_{2i}^{**} \leftarrow 2t_i^* - 1$
} //replace t_i^* with two elements t_{2i-1}^{**} and t_{2i}^{**} in $\vec{f}^{**}$
$\vec{f}^* \leftarrow \vec{f}^{**}$
}
for each element t_i^{**} in $\vec{f}^{**}$
if $t_i^{**} < 0$ **then** $t_i^{**} \leftarrow -1$ **else** $t_i^{**} \leftarrow 1$
//make each element of $\vec{f}^{**}$ 1 or -1
reorder elements of $\vec{f}^{**}$ *//optional optimization step*
return $\vec{f}^{**}$ **}**

Figure 4. Pseudo-code for Step 4 of the proposed synthesis method.

The resulting TT only has +1 and −1 as elements, so Steps 3 and 4 are also skipped. $\vec{f}$ is the TT of a 2-input XOR gate.

Consider the same problem for the UP multiplier. In Step 1, we map the target function to the IBP domain thus:

$$\hat{P}(X_1, X_2) = 1 - 2\hat{F}\left(\frac{1 - X_1}{2}, \frac{1 - X_2}{2}\right)$$
$$= 0.5(1 + X_1 + X_2 - X_1 X_2)$$

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

Example 2. Suppose we want to synthesize the IBP add function $\hat{F}(X, Y) = X + Y$. It is already in multi-linear form, so we proceed with the inverse Fourier transform.

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

Two of the elements of $\vec{f}$ are greater than 1 or less than −1, which means that it is impossible to implement the basic add operation directly by an SC circuit The standard solution to this dilemma is "scaled" addition [3], which substitutes $s(X + Y)$ for $X + Y$, where s is an ad hoc scale factor requiring an extra pseudo-random input.

Steps 3 and 4 of the proposed synthesis procedure handle the foregoing scaling problem automatically. For Example 2, Step 3 divides $\vec{f}^* = [2 \quad 0 \quad 0 \quad -2]^{\mathrm{T}}$ by its maximum element 2 to bring it into the interval [−1,1], making $\vec{f}^* = [1 \quad 0 \quad 0 \quad -1]^{\mathrm{T}}$. Since this contains 0 elements, Step 4 is needed to remove them. As described in Figure 4, $\vec{f}^*$ is repeatedly processed until all its elements are converted to 1 or −1. This involves expanding $\vec{f}^*$ and introducing auxiliary inputs. After applying Step 4 (without the optional optimization step) to the $\vec{f}^*$ of Example 2, we get

$$\vec{f}^{**} = [1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1]^{\mathrm{T}}$$

which is the BF $f(x,y,r) = (x \wedge r) \vee (y \wedge r) \vee (x \wedge y)$, where an auxiliary input r has been added to the function. This input must be fed with the IBP stochastic number 0, i.e., a random

bit-stream. Figure 5(a) shows a simple AND-OR implementation of *f*.

It is possible to optimize the synthesized circuit further by reordering the elements of $\vec{f}^{**}$. As noted earlier, Step 4 replaces each element of $\vec{f}^{*}$ with two or more new elements. While the number of 1's and −1's is important during this step, their order does not matter. For example, the 0's in $\vec{f}^{*}$ of the running example are expanded to $[1\ -1]^{T}$ in $\vec{f}^{**}$, but they can also be expanded to $[-1\ 1]^{T}$. Reordering the elements does not alter the SC behavior, but it can lead to smaller circuits. For instance, a reordered $\vec{f}^{**}$ for the running example is

$$\vec{f}^{**} = [1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1]^{T}$$

This is the BF $f(x,y,r) = (x \wedge r) \vee (y \wedge r')$, which has the AND-OR implementation of Figure 5(b). It is obvious that this circuit is a 2-to-1 multiplexer with *r* as its select input. This is precisely the standard scaled adder in the SC literature [2].

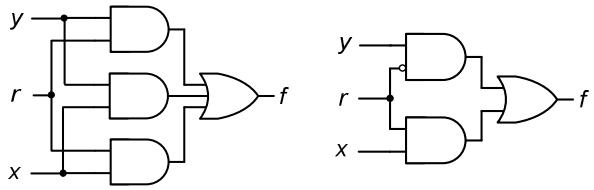

Figure 5. Two synthesized circuits for SC addition: (a) without optimization; (b) with optimization.

Finding the best ordering in Step 4 of the synthesis algorithm is a difficult optimization problem. In this paper, we employ a greedy heuristic that sorts the elements in ascending (or descending) order, but produces acceptable results.

Example 3. Next consider implementing a more general arithmetic function $\hat{F}(X) = 0.4375 - 0.25X - 0.5625X^2$. This time the function has a non-linear X^2 term. Such terms are replaced by multi-linear terms by introducing new inputs. In general, if the target function is a polynomial of degree *k*, we replace its variables with *k* new variables. In the present case, X_1 and X_2 replace *X*.

$$\hat{P}(X_1, X_2) = 0.4375 - 0.125(X_1 + X_2) - 0.5625X_1X_2$$

Note that there may be many multi-linear polynomials that are equivalent to the target function, and considering them all is infeasible. In our approach, we only select multi-linear polynomials that are symmetric with respect to their variables, like the one shown above. At Step 2, we have

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

All the elements of $\vec{f}$ are in the [−1,1] interval, so Step 3 can be skipped. Step 4, is required, and it takes four iterations to arrive at an $\vec{f}^{**}$ with 1's and −1's, which results in the addition of four auxiliary inputs r_1,r_2,r_3,r_4. The final BF is

$f(x_1, x_2, r_1, r_2, r_3, r_4) = (r_1' \wedge x_1' \wedge x_2') \vee (r_2' \wedge r_3' \wedge x_1' \wedge x_2') \vee (r_2' \wedge r_4' \wedge x_1' \wedge x_2') \vee (r_1 \wedge r_2 \wedge x_1 \wedge x_2) \vee (r_1 \wedge r_3 \wedge x_1 \wedge x_2) \vee (r_1 \wedge r_4 \wedge x_1 \wedge x_2)$

Figure 6 shows a straightforward two-level implementation C_1 of *f*. The auxiliary inputs are connected to a 4-bit LFSR, which generates four independent random bit-streams [7]. Notice that x_1 and x_2 are fed with independent bit-streams that carry the same number *X*. They 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 [7].

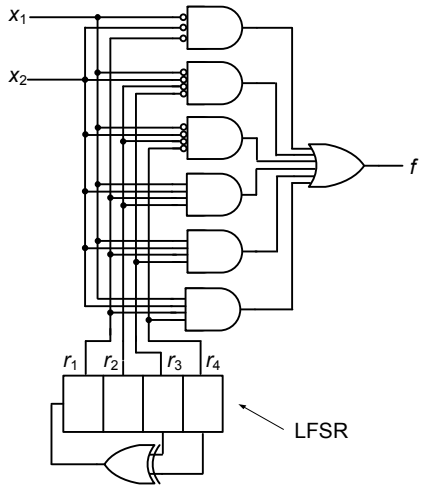

Figure 6. Implementation C_1 of $\hat{F}(X) = 0.4375 - 0.25X - 0.5625X^2$ obtained by the proposed synthesis procedure.

Example 4. Next, we implement the representative "gain" function $\hat{F}(X) = 1.5X$. An XOR-gate IBP multiplier cannot be used in this case, because the constant multiplicand exceeds 1. (This function could be viewed as division by 2/3; however, SC dividers only exist in approximate forms [2] and will not be considered here.) $\hat{F}$ maps [−1,1] to [−1.5,1.5] which is outside the IBP interval, so we start by modifying the target function to implement a "saturated gain" defined as follows:

$$\hat{F}(X) = \begin{cases} 1 & X > \frac{2}{3} \\ 1.5X & -\frac{2}{3} \le X \le \frac{2}{3} \\ -1 & X < -\frac{2}{3} \end{cases}$$

The modification is due to the fact that $\hat{F}(X) = 1.5X$ goes out of bounds for inputs outside the range [−2/3, 2/3]. This non-polynomial function is approximated by a polynomial $\hat{P}(X)$ in Step 1 of the synthesis method. The cost of the final circuit increases with the degree of $\hat{P}(X)$, i.e., with the accuracy of the approximation. We chose the degree-5 polynomial $\hat{P}(X) = 0.36X^5 - 1.45X^3 + 1.75X$, which seems to be a reasonable compromise between accuracy and cost. After applying the rest of synthesis procedure, we arrive at a circuit (not shown) that has 51 gates and 6 flip-flops. Like Example 3, the flip-flops are a part of the LFSR used as the random number generator.

There is another cost-accuracy trade-off in the synthesis process which is controlled by the value of *max_iterations* in Step 4 (Figure 4). Increasing the number of iterations in Step 4 increases the accuracy of the synthesized circuit, but leads to larger TTs with more auxiliary inputs, and hence a larger circuit. We set *max_iterations* = 6 for Example 4, which again seems a reasonable compromise between cost and accuracy.

After synthesis, we simulated the SC circuit of this example for a full range of X values on the interval [−1,1]. Figure 7 shows simulation results comparing $\hat{F}^*(X)$ with the target function $\hat{F}(X)$ and the approximating polynomial $\hat{P}(X)$. The circuit output $\hat{F}^*$ follows $\hat{P}$ very closely. They both differ from $\hat{F}$ for $X > 0.5$ and $X < -0.5$, however, because $\hat{P}$ is not a good approximation in those ranges. This example shows that the proposed synthesis method can generate circuits that accurately implement a given polynomial or a well-approximated non-polynomial function.

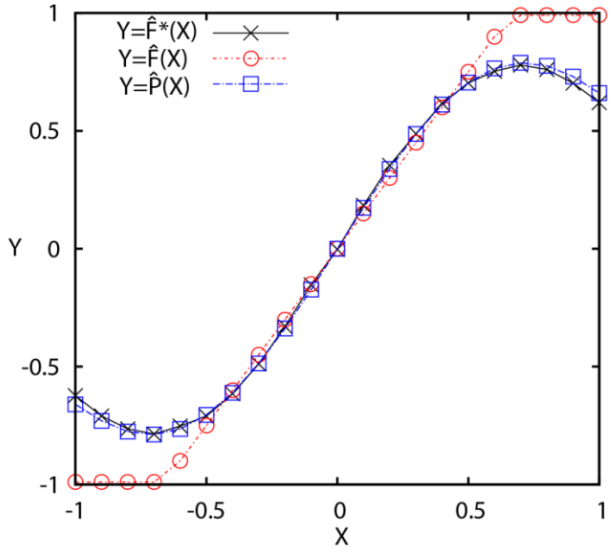

Figure 7. Simulation results for Example 4 comparing the synthesized circuit output $\hat{F}^*(X)$, the target (saturated gain) function $\hat{F}(X)$, and the polynomial approximation $\hat{P}(X)$.

The spectral transform approach discussed so far for combinational circuits can be extended to sequential circuits. To illustrate this, we consider reverse engineering the circuit in Figure 1. It has two inputs p_1 and p_2, and one output p_3 which is also the state variable of this 2-state machine. The combinational logic C computes the circuit's next state p_3* according to the values of p_1, p_2 and p_3. We can use the Fourier transform to extract the IBP SC behavior of C thus:

$$P_3^* = 0.5(P_1 + P_2 + P_3 - P_1P_2P_3)$$

Since the next-state and the current-state variables are essentially the same bit-streams with one clock cycle delay, we can assign $P_3^* = P_3$ and solve the above equation for P_3.

$$P_3 = \frac{P_1+P_2}{1+P_1P_2}$$

This describes the circuit's SC behavior in the IBP domain. Applying the appropriate conversion rule from Table 1 yields the UP behavior defined by Equation (1).

V. Comparison to Prior Work

Very little previous research has addressed the systematic design of SC circuits. Qian et al. [15] recently developed a non-spectral method of implementing a single-variable SC function $\hat{F}(X)$ defined on the UP domain. They convert $\hat{F}(X)$ to a Bernstein polynomial of the form

$$\hat{F}(X) = \sum_{i=0}^{n} C_i \binom{n}{i} X^i(1-X)^{n-i} \quad (8)$$

in which the C_i's are constant coefficients. This polynomial is then mapped to a specific style of logic circuit termed a "reconfigurable architecture". This architecture 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. Figure 8(a) shows the reconfigurable architecture implementing Equation (8). 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 (8). 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 (8). Note that the x_i inputs are independent SNs representing the number X, similar to the x_1 and x_2 inputs of Figure 6. This structure requires n SNGs to generate the x_i's and $n + 1$ SNGs to generate the c_i's, which implies a significant area cost. Although not explicitly discussed in [15], this design approach can be extended to multi-variate functions, and to the bipolar domain.

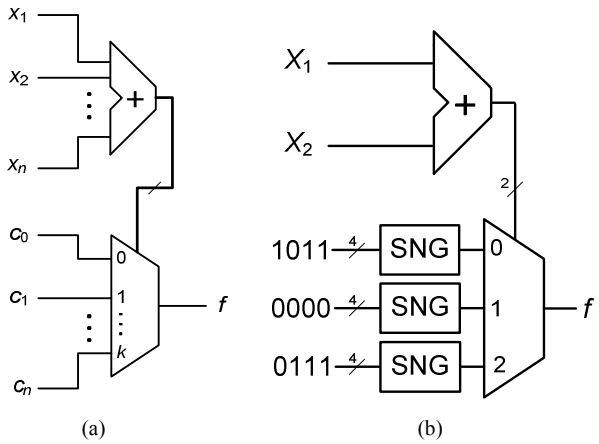

Figure 8. (a) Reconfigurable architecture proposed by Qian et al. [15]; (b) Implementation C_2 of $\hat{F}(X) = 0.4375 - 0.25X - 0.5625X^2$ using this architecture.

We now show that the method of [15] can be re-interpreted in terms of a spectral transform, a "Bernstein transform" $\mathcal{B}$ with a basis different to that of the Fourier transform $\mathcal{F}$. We also compare the two synthesis approaches.

Consider the 2-variable Boolean function $f(x_1,x_2)$. We define a new spectral basis for it 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 $B_3 = 0.25(1 - X_1)(1 - X_2)$. The resulting transform $\mathcal{B}$ of f can then be written as:

$$F(X_1, X_2) = \sum_{i=0}^{3} C_i B_i \quad (9)$$

Somewhat surprisingly, this corresponds to the same multi-linear expression generated by the Fourier transform. To see this, we expand (9) thus:

$$F(X_1, X_2) = 0.25\big(\; C_0(1+X_1)(1+X_2) + C_1(1+X_1)(1-X_2) + C_2(1-X_1)(1+X_2) + C_3(1-X_1)(1-X_2)\big)$$

$$= 0.25(C_0 + C_1 + C_2 + C_3 + (C_0 - C_1 + C_2 - C_3)X_2 + (C_0 + C_1 - C_2 - C_3)X_1 + (C_0 - C_1 - C_2 + C_3)X_1X_2)$$

which is the multi-linear polynomial produced by the Fourier transform, as the following equation demonstrates:

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

Despite similarities between the Fourier and Bernstein transforms, the design approach we are proposing here is more comprehensive than that of [15]. Our method can synthesize multi-variate polynomials directly in any of the UP, BP and IBP representations. Furthermore, it starts from the more convenient power-form polynomial rather than the Bernstein form, and it produces a BF that can be implemented using any existing logic optimization techniques. The method of [15], on the other hand, maps the target function to a specific reconfigurable architecture consisting of an adder, a multiplexer, and SNGs, whose total cost may be sub-optimal.

To evaluate the two approaches, we compared our design C_1 for Example 3 (see Figure 6) with a design C_2 for the same example produced by the method of [15] adjusted for the IBP domain; see Figure 8(b). Besides its adder and multiplexer, C_2, contains three SNGs, each consisting of a 4-bit comparator and a 4-bit LFSR, as in Figure 3. Note, however, that C_2 can also 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 [16] to optimize both designs and map them to a generic library of gates. Table 3 shows the comparison between the two implementations, along with two other representative circuits. The area cost is reported in terms of unit cells. These results show the fact that the spectral approach can significantly reduce the high cost associated with the introduction of SNGs.

Table 3. Comparison between the proposed synthesis method and that of [15]; area is reported in terms of unit cells from a generic library.

Target design		Example 3	Example 4	Gamma correction [15]
Reconfigurable architecture [15]	Area	110	378	957
	Gates	20	79	177
	Flip-flops	8	24	70
Spectral transform [this paper]	Area	67	182	324
	Gates	15	51	95
	Flip-flops	4	6	10

VI. Conclusions

Stochastic computing is re-emerging as an important technology for certain applications requiring massive parallelism and/or massive error tolerance. Although SC has been recognized for many years, its underlying theory is not well developed. We have shown that transforms linking the Boolean and the spectral domains provide some fundamental insights into SC behavior. We have successfully applied spectral analysis to SC circuits in a way that easily accommodates the most useful stochastic number formats. Furthermore, we have presented a novel synthesis technique based on spectral transforms. Comparing this work to a recently proposed SC design methodology based on Bernstein polynomials, we found that our more general approach can lead to significant cost savings.

Acknowledgement

This work was supported by Grant CCF-1017142 from the U.S. National Science Foundation.

References

[1] Alaghi, A., Hayes, J.P., "Survey of stochastic computing," *ACM Trans. Embedded Computing Systems*, 2012, to appear.

[2] Gaines, B.R., "Stochastic computing," *Proc. AFIPS Spring Joint Computer Conf.*, pp.149-156, 1967.

[3] Gaines, B.R., "Stochastic computing systems," *Advances in Information Systems Science*, vol. 2, pp.37-172, 1969.

[4] Gross, W.J., Gaudet, V.C., Milner, A., "Stochastic implementation of LDPC decoders," *Proc. Asilomar Conf. Signals, Systems and Computers*, pp.713-717, 2005.

[5] Hurst, S.M., Miller, D.M., Muzio, J.C., *Spectral Techniques for Digital Logic,* Academic Press, 1985.

[6] IEEE. 2009. IEEE Standard 802.11n for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks. http://standards.ieee.org.

[7] Jeavons, P., Cohen, D.A., Shawe-Taylor, J., "Generating binary sequences for stochastic computing," *IEEE Trans. Information Theory*, vol.40, pp.716-720, 1994.

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

[9] Li, P., Lilja, D.J., "Using stochastic computing to implement digital image processing algorithms," *Proc. Conf. Computer Design*, pp.154-161, 2011.

[10] Li, P., Qian, W., Riedel, M.D., Bazargan, K., Lilja, D.J., "The synthesis of linear finite state machine-based stochastic computational elements," *Proc. ASP-DAC.*, pp.757-762, 2012.

[11] Naderi, A., Mannor, S., Sawan, M., Gross, W.J., "Delayed stochastic decoding of LDPC codes," *IEEE Tran. Signal Processing*, vol.59, pp.5617-5626, 2011.

[12] O'Donnell, R., "Some topics in the analysis of Boolean functions," *Proc. ACM STOC Conf.*, pp.569-578, 2008.

[13] Poppelbaum, W.J., "Statistical processors," *Advances in Computers*, pp.187-230, 1976.

[14] Poppelbaum, W.J., Afuso, C., Esch, J.W., "Stochastic computing elements and systems," *Proc. AFIPS Fall Joint Computer Conf.*, pp.635-644, 1967.

[15] Qian, W., LI, X., Riedel, M.D., Bazargan, K., Lilja, D.J., "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Computers*, vol.60, pp.93-105, 2011.

[16] Sentovich, E.M., *et al.*, "SIS: A System for sequential circuit synthesis," Univ. of California, Berkeley, Tech. Report UCB/ERL M92/41, Electronics Research Lab, 1992.