Exploring Target Function Approximation for Stochastic Circuit Minimization

Chen Wang[1], Weihua Xiao[1], John P. Hayes[2], Weikang Qian[1,3,4]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, USA
[3]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
[4]State Key Laboratory of ASIC & System, Fudan University, Shanghai, China
Email: wangchen_2011@sjtu.edu.cn, 019370910014@sjtu.edu.cn, jhayes@eecs.umich.edu, qianwk@sjtu.edu.cn

Abstract

Stochastic computing (SC) is an emerging paradigm for designing circuits to perform complicated computation with simple circuitry. Although SC circuits have small area and critical-path delay, due to the need of many clock cycles to perform computation, they have a large overall latency and energy consumption. One solution to this problem is to further minimize the circuits. In this work, we explore target function approximation to derive an SC circuit with significantly reduced area and delay. We propose two static methods that first construct a set of functions close to the given target function and then select the best synthesized SC circuit realizing one of these functions. We also propose an efficient dynamic method that simultaneously searches for the best approximated target function and the corresponding minimized SC circuit. The experimental results show that on average, our dynamic method dramatically reduces the area, critical-path delay, and area-delay product of the SC circuits by 80%, 59%, and 91%, respectively, over the state-of-the-art Maclaurin polynomial-based method for a given error bound of 2%. The code of our methods is made open-source.

Keywords

stochastic computing, stochastic circuit synthesis, circuit simplification

1 Introduction

Stochastic computing (SC) is an unconventional computing paradigm [2]. Unlike binary computing, SC operates on stochastic bit streams and uses the probability of 1s in a bit stream to represent a value. It can realize complicated computations with simple circuitry. For example, it can realize multiplication with a single AND gate. Thus, SC has great potential to realize applications with much smaller circuit area. Applications where SC has been found successful include image processing [1, 8] and neural networks [4, 14, 16].

However, in order to achieve a high precision, stochastic bit streams should be long enough. Since SC uses one clock cycle to process each bit, the number of clock cycles needed is large. For an SC circuit, the latency and energy consumption for each computation are evaluated as the clock period times the number of clock cycles and the average power consumption times the latency, respectively. Consequently, SC has long computation latency and large energy consumption. One solution to this problem is to further minimize the SC circuit. Since the clock period is determined by the *critical-path delay* (hereafter referred to as *delay*) of the SC circuit, minimizing the SC circuit will reduce its delay and hence, the clock period. It also leads to the reduction of the total area of the SC circuit. Since the power consumption is roughly proportional to the area, reduction of the area further leads to the reduction of the average power consumption. In this case, both the computation latency and energy consumption can be reduced.

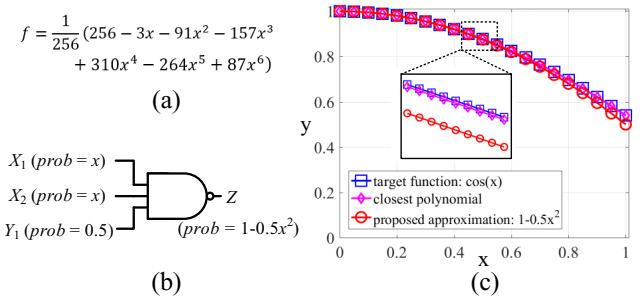

Figure 1: Implementing target function $\cos(x)$: (a) the closest degree-6 polynomial approximation by [13]; (b) an SC circuit realizing another approximated target function $1-0.5x^2$; (c) the curves of the target function $\cos(x)$ and its approximated functions.

In this work, we focus on SC circuits built with combinational logic. Such a circuit can only implement polynomials. Thus, given an arbitrary arithmetic target function, the existing synthesis methods first approximate the target by a polynomial and then synthesize an SC circuit for the polynomial [11, 13]. The polynomial approximation introduces error. Conventionally, this error is always minimized. As an example, in order to implement the target function $\cos(x)$, the previous method [13] will instead implement a polynomial shown in Fig. 1(a), which is the closest degree-6 polynomial approximation. Fig. 1(c) plots the curves for both the target $\cos(x)$ and the approximated target, which shows that they are very close, but still have difference as shown in the inset. If the approximation error is relaxed a little bit, we may find many different target polynomials. This creates a much larger design space

Chen Wang, Weihua Xiao, and Weikang Qian were supported in part by the National Key R&D Program of China under Grant 2020YFB2205501 and by the State Key Laboratory of ASIC & System Open Research Grant 2019KF004. John P. Hayes was supported by the U.S. National Science Foundation under Grant CCF-2006704. Corresponding author: Weikang Qian.

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415701

where an SC circuit with reduced cost may exist. For the example of $\cos(x)$, the optimized SC circuit realizing the closest approximated target has 37 gates and a critical path of 7 gates. However, if we allow more approximation error, then it can be approximated by $1-0.5x^2$, which corresponds to the red curve in Fig. 1(c). This new target polynomial, although has a larger approximation error, can be realized by a significantly simplified SC circuit with only a single *NAND3* gate, as shown in Fig. 1(b). This new design achieves an area-delay product (ADP) reduction of 259×! Previously, this kind of target function approximation was never explored. In our study, from this new perspective, we systematically explore the target function approximation in order to significantly reduce the area and delay of an SC circuit. The problem we consider is: *given a target function and an error bound, synthesize a minimal SC circuit realizing an approximated target function with error no more than the bound.* Such SC circuits are applicable to error-tolerant applications like image processing and neural networks.

However, a major challenge is how to determine a good approximation to the original target function. We propose three methods to find good approximations and synthesize the corresponding SC circuit. Note that in this work, we focus on univariate target functions to illustrate our methods. However, it is also possible to extend our methods to handle multivariate functions. The experimental results show that our methods effectively reduce the area, delay, and ADP of the SC circuit with a small relaxation to the approximation error. Since the power consumption is roughly proportional to the area, and the computation latency is exactly proportional to the delay, ADP is an indicator of the energy consumption. Therefore, the ADP reduction indicates the reduction of the energy consumption.

The main contributions of this work are listed as follows.

(1) For the first time, we propose to explore the approximation of a target function to derive an SC circuit with dramatically reduced area and delay.
(2) We propose two static approximation methods to minimize an SC circuit. They first construct a promising set of approximated target functions and then apply an existing SC circuit synthesis method to them to identify the best solution.
(3) We propose an efficient dynamic approximation method that integrates the search of the approximated target function into the SC circuit synthesis process. It is the most powerful method and the single-gate implementation of $\cos(x)$ shown in Fig. 1(b) is synthesized by this method.

The code of our proposed methods is made open-source at https://github.com/SJTU-ECTL/TFASC.

The rest of the paper is organized as follows. Section 2 discusses the related works on SC circuit synthesis. Section 3 provides the preliminaries. Section 4 overviews our methods. Sections 5 and 6 present the proposed static and dynamic approximation methods, respectively. Section 7 shows the experimental results. Finally, Section 8 concludes the paper.

2 Related Works

In recent years, a number of works studied the synthesis of SC circuits [3, 5, 7, 11–13, 17]. In [13] and [7], methods to synthesize reconfigurable combinational and sequential SC circuits are proposed, respectively. The works [3, 17] further consider synthesizing fixed SC circuits, which have smaller area than reconfigurable ones. As revealed by [5, 17], SC circuits have a much larger solution space than traditional logic circuits since many circuits with different Boolean functions can realize the same target function stochastically. Unfortunately, the methods in [3, 17] only exploit a limited subset of the whole solution space. In [11], a state-of-the-art method is proposed to design SC circuits based on Maclaurin series expansion of the target function. However, since no thorough exploration of the solution space is employed, the solution optimality is not guaranteed. In [12], another state-of-the-art method is proposed that systematically searches for the optimal solution within a larger solution space through a search tree. Our work is based on this method and further advances it by exploring the feasibility of target function approximation. By this, the solution space is enlarged and better circuits can be found.

3 Preliminaries

3.1 A General SC Circuit

We consider a general SC circuit design realizing univariate polynomials proposed in [17]. It is shown in Fig. 2. The circuit has n+m inputs. The first n inputs $X_1, \ldots, X_n$ are supplied with n independent input bit streams with the same variable probability x. The last m inputs $Y_1, \ldots, Y_m$ are given independent bit streams with a constant probability of 0.5.

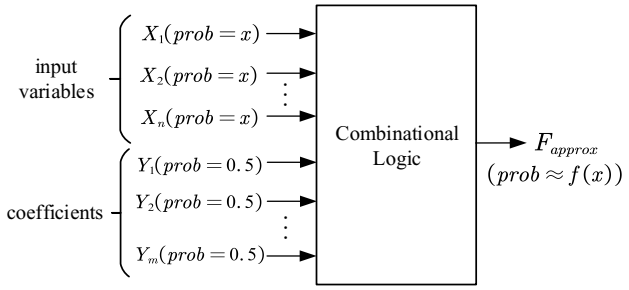

Figure 2: A general form of the SC circuit to realize the target function [17].

Assume that the Boolean function of the combinational circuit in Fig. 2 is $F(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$. As shown in [12], the output probability of the circuit is a function in the form of

$$g(x) = \sum_{i=0}^{n} \frac{G(i)}{2^m} x^i (1-x)^{n-i}, \tag{1}$$

where $G(i)$ is the number of minterms (a_1,…,a_n,b_1,…,b_m) satisfying that $F(a_1,\ldots,a_n,b_1,\ldots,b_m)=1$ and $\sum_{j=1}^{n} a_j = i$. An example for $G(i)$ is illustrated using the 2-dimensional truth table shown in Fig. 3 with n=3 and m=2. In this truth table, the columns and the rows list the combinations of X's and Y's, respectively. For example, $G(1)$ is the total number of 1s in the columns with $X_1X_2X_3$ as 001, 010, and 100 (i.e., the 3 columns in the lightest grey color in the figure), since these columns satisfy that $X_1+X_2+X_3=1$. In this case, $G(1)=2$. Some additional $G(i)$ values and the columns giving these $G(i)$'s are also shown in the figure. From Eq. (1), m determines the precision of the coefficients. Thus, it is called the *precision parameter* [12]. For the Boolean function shown in Fig. 3, its output probability realizes the following function

$$g(x) = \frac{2}{2^2} x(1-x)^2 + \frac{4}{2^2} x^2(1-x) + \frac{2}{2^2} x^3.$$

	$G(0)$=0	$G(1)$=2	$G(2)$=4	$G(3)$=2				
$Y_1Y_2 \setminus X_1X_2X_3$	000	001	010	011	100	101	110	111
00	0	0	0	0	0	0	0	0
01	0	0	0	0	0	0	0	0
10	0	0	0	0	1	1	1	1
11	0	0	0	0	1	1	1	1

Figure 3: The 2-dimensional truth table of a Boolean function. $G(i)$ **($0 \le i \le 3$) above the table corresponds to the total number of minterms in the columns with the same shading color.**

3.2 The Previous Synthesis Method

In this section, we describe the previous method proposed in [12] to synthesize the SC circuit in Fig. 2. Suppose the target arithmetic function is $f(x)$.

The form of Eq. (1) resembles a special type of polynomial called *Bernstein polynomial* [9]. Its general form is

$$B(x) = \sum_{i=0}^{n} b_i \binom{n}{i} x^i (1-x)^{n-i}, \tag{2}$$

where b_i is a *Bernstein coefficient* and $\binom{n}{i}x^i(1-x)^{n-i}$ is a *Bernstein basis polynomial*. The method in [12] exploits the above connection. Fig. 4(a) shows a flow chart of the method. It can be decomposed into two major phases: the target transformation phase and the Boolean function synthesis phase.

3.2.1 The Target Transformation Phase In this phase, the method first applies a quadratic programming approach proposed in [13] to obtain a Bernstein polynomial $B^*(x)$ (with the Bernstein coefficient b_i^*) closest to the target $f(x)$:

$$B^*(x) = \sum_{i=0}^{n} b_i^* \binom{n}{i} x^i (1-x)^{n-i}. \tag{3}$$

Comparing Eqs. (1) and (3), we can see that if we choose a Boolean function such that its $G(i) = 2^m b_i^* \binom{n}{i}$, for all $0 \le i \le n$, the corresponding combinational circuit can realize $B^*(x)$. However, since $G(i)$ is an integer, $G(i)$ is set to $G^*(i) = \text{round}(2^m b_i^* \binom{n}{i})$, where round() is the rounding function. This eventually gives the following *original target Bernstein polynomial (OTBP)* to be implemented by the combinational logic:

$$B_T(x) = \sum_{i=0}^{n} \frac{G^*(i)}{2^m} x^i (1-x)^{n-i}. \tag{4}$$

Note that the OTBP is characterized by a vector $\overrightarrow{G^*} = (G^*(0), \ldots, G^*(n))$. We call it the *feature vector (FV)* of the OTBP, or *original FV* for short, which is the same as the *problem vector* defined in [12].

3.2.2 The Boolean Function Synthesis Phase By the definition of $G(i)$, there exist many different Boolean functions that can realize the FV $\overrightarrow{G^*}$. In this phase, the method synthesizes a Boolean function with good quality among these candidates.

The final solution is constructed by adding a sequence of cubes (i.e., product terms) into the on-set of the Boolean function. Each entry in the FV specifies how many minterms in a specific group of columns in the 2D truth table should be assigned with the value 1. For simplicity, the method requires that the later selected cubes be *disjoint* to any already selected cubes. Each time a cube is chosen, some entries in the FV are reduced to indicate that some minterms in some specific groups of columns have been assigned with the value 1. Due to the disjoint property, the reduction is given by the *cube vector (CV)* of the cube, denoted in brackets as $[C(0), C(1), \ldots, C(n)]$, where $C(k)$ is the number of minterms of the cube that are covered by the columns in the 2D truth table with $\sum_{i=1}^{n} X_i = k$. For example, with $n = 3$ and $m = 2$, the 2D truth table of the cube X_1Y_1 is shown in Fig. 3. From the figure, we can see that the CV of the cube is $[C(0), C(1), C(2), C(3)] = [0, 2, 4, 2]$. The reduced FV is called the *remaining feature vector (RFV)*. For example, if the original FV is $(1, 3, 6, 2)$, as the cube with CV $[0, 2, 4, 2]$ is chosen, the FV is reduced to $(1, 1, 2, 0)$ as the RFV. The procedure continues until the RFV reduces to zero. Each time a cube is chosen, it must satisfy the *capacity constraint*, which requires that each entry of the CV should be no more than the corresponding entry of the RFV. For example, the cube with CV $[0, 2, 4, 2]$ violates the capacity constraint of the RFV $(0, 1, 3, 2)$, while another cube with CV $[0, 1, 2, 1]$ does not.

The cubes chosen first will influence the later cube selection. Thus, in order to determine a good solution, a search tree is built. Each node in the search tree stores both the set of chosen cubes and the RFV. For each node, since many new cubes can be extracted from the RFV of that node, it will be expanded to multiple nodes in the next level. The search tree reaches a leaf when the RFV becomes zero. In this case, a set of cubes satisfying the original FV is obtained. The final solution is given by the best leaf node. To speed up the search, at each node, only the largest cubes that satisfy both the disjoint property and the capacity constraint are chosen.

4 Overview of the Proposed Methods

From Fig. 4(a), we can see that the OTBP is an important link between the given target function and the synthesized combinational circuit. If we change the OTBP slightly, its error over the target function may increase. However, the final synthesized circuit corresponding to the changed OTBP may improve. We call the changed OTBP the *new target Bernstein polynomial (NTBP)*. In this work, we exploit this idea and try to find an NTBP satisfying the given error bound and giving the best circuit quality. We measure the error of a target Bernstein polynomial B_T (either NTBP or OTBP) as the L2-norm distance between it and the target function f, namely

$$||B_T(x) - f(x)||_2 = \sqrt{\int_0^1 |B_T(x) - f(x)|^2 dx}. \tag{5}$$

We adopt the L2-norm to measure the error, which follows the convention of approximation error measurement in the SC literature [13]. It is also possible to use alternative error measures such as mean absolute error (MAE), which is the L1-norm. We evaluate the L2-norm by numerical integration.

In the following, we begin with an initial attack to this problem in which we propose a straightforward solution, that is, we first construct a set of NTBPs and then select the best synthesized circuit realizing one of them. We call it the *static approximation method*. In view of its low efficiency, we further develop a more powerful solution, called the *dynamic approximation method*, which synthesizes a good NTBP and the corresponding circuit simultaneously.

5 Initial Attack: Static Approximation Method

In this section, we present the static approximation (SA) method. Its basic idea is to search within a set of NTBPs for one giving the best SC circuit. It first constructs a set of NTBPs. Then, it only keeps

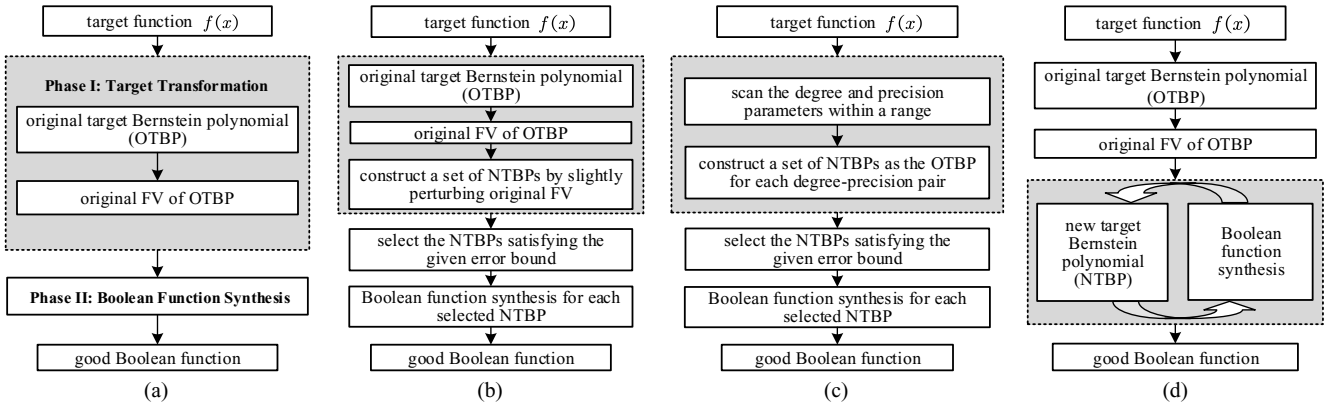

Figure 4: The flow charts for (a) baseline (BS) in [12], (b) perturbation (PER) method, (c) degree-precision scanning (DPS) method, and (d) dynamic approximation (DA) method.

those satisfying the given error bound and applies the synthesis method in [12] to each kept one to obtain the corresponding SC circuit. Finally, it picks the SC circuit with the best quality.

Following the basic idea, we propose two detailed SA methods. They only differ by how the set of NTBPs is constructed, which is highlighted in the grey blocks in Figs. 4(b) and (c), respectively. The first one, called the *perturbation (PER) method*, slightly perturbs the OTBP (see Fig. 4(b)). This method keeps both the degree n and the precision m, and it constructs the set of NTBPs so that their corresponding FVs $(v_0, \ldots, v_n)$ satisfying that for all $i = 0, \ldots, n$, $|v_i - G^*(i)| \leq 1$, where $G^*(i)$'s are the FV entries of the OTBP (see Eq. (4)). The second method, shown in Fig. 4(c), is called the *degree-precision scanning (DPS) method*. It forms the set of NTBPs as the OTBPs with degree n' and precision m', where n' runs from 1 to n. For each n', m' runs from 1 to $n + m - n'$. Afterwards, both the PER and DPS methods select the NTBPs with errors satisfying the given error bound, and synthesize the circuits for these NTBPs one after another using the method in [12]. Finally, the best synthesized circuit is returned.

6 More Powerful Solution: Dynamic Approximation Method

The SA methods have a rigid partition between the NTBP construction and the circuit synthesis. They first construct a set of initial NTBPs and then apply the synthesis method to each of them, which causes a long runtime. In this section, we propose a more powerful solution, the *dynamic approximation (DA) method*. It takes a different approach by searching the NTBPs and synthesizing the circuit simultaneously.

6.1 Basic Idea

The dynamic method shown in Fig. 4(d) modifies the Boolean function synthesis phase of the previous method [12]. One key modification is that we allow some changes to the original FV during the search. Our major contribution is that we simultaneously search for the FVs corresponding to the NTBPs and synthesize the Boolean function to realize these NTBPs, as shown in the grey block in Fig. 4(d). The main procedure of the DA method is shown in Algorithm 1. The inputs are the original FV $FV_{org} = (G(0), \ldots, G(n))$, the optimization objective obj, the given error bound e_b, and the target function $f(x)$. The optimization objective is typically a hardware cost measure, such as the literal count of a sum-of-product (SOP) expression or the ADP of a multi-level circuit. Assume that the error of the OTBP is e_{org}. A requirement on the input parameters is that $e_b > e_{org}$.

Algorithm 1: The proposed dynamic approximation (DA) method.

Input : the original feature vector $FV_{org} = (G(0), \ldots, G(n))$, an optimization objective obj, an error bound e_b, and the target function $f(x)$.
Output : the final Boolean function and the corresponding circuit.

```
1 S_sol ← ∅; N_root.RFV ← FV_org; N_root.SCC ← ∅;
2 S_node ← {N_root};
3 while S_sol = ∅ do
4 |   R ← candNodesGen(S_node, e_b);
5 |   {S_new, S_sol} ← classifyNodes(R, e_b);
6 |   S_node ← pruneNodes(S_new);
7 return getBest(S_sol, obj);
```

As in [12], this method builds a search tree. Each node in the tree consists of two important data members, namely the set of chosen cubes (SCC) and RFV (see Section 3.2.2 for details). Line 1 initializes an empty set S_{sol} to hold the solution nodes. It further defines the root node N_{root} with its RFV as the original FV and its SCC empty. The procedure expands the nodes level by level. The set of nodes to be expanded at a level is stored in S_{node}. Line 2 initializes S_{node} to be the set of nodes at Level 0, i.e., $\{N_{root}\}$. Lines 3–6 are the main expansion loop. In each iteration, one level of the nodes in the tree are expanded to the next level. Line 4 first calls the function *candNodesGen* to generate a set R of candidate nodes that are promising to be developed into the final solution nodes. Each node in R expands one node N in S_{node} by adding a new cube into the SCC of N and updating the RFV. Next, Line 5 calls the function *classifyNodes* to classify the nodes in R into a set S_{new} of new nodes and a set S_{sol} of the solution nodes. Then, Line 6 calls the function *pruneNodes* to prune the redundant nodes and the unpromising nodes in S_{new}. The non-redundant promising nodes are kept in S_{node} and passed to the next iteration. The loop terminates when the solution node set S_{sol} is nonempty at a certain level. Then, the function *getBest* is called to obtain the best solution from the nodes in S_{sol} according to the objective obj (Line 7). In this study, *getBest*

chooses the circuit with the minimum ADP. The following example illustrates the proposed method.

Example 1. *Suppose that the target function is* $-0.26x^2 + 1.06x$ *and the parameters are* $n = 2$ *and* $m = 4$. *The L2-norm of the target function is 0.5. The closest Bernstein polynomial* $B^*(x)$ *(see Eq. (3)) has coefficients* $(b_0^*, b_1^*, b_2^*) = (0, 0.53, 0.8)$. *Then, we can obtain the FV of the OTBP as* $(0, 17, 13)$ *using the method shown in Section 3.2.1. The error* e_{org} *of the OTBP is 0.0059. We choose the error bound* e_b *as* 0.02, *which is* 4% *of the L2-norm of the target function. The search tree constructed by our method is shown in Fig. 5. The root node* n_0 *is expanded into 4 levels until the solution node is found. Each vector in brackets represents a CV, corresponding to a chosen cube, while each vector in parentheses represents an RFV. The set of CVs below each node corresponds to the SCC of the node. For example, at node* n_{21}, *its SCC includes two cubes with the CVs of* $[0, 16, 0]$ *and* $[0, 0, 8]$, *respectively. Its RFV is* $(0, 1, 5)$. *The nodes* n_{22} *and* n_{33} *in the red dashed rectangles are pruned during the search due to the duplication of their assigned truth tables (see Section 6.4 for details). Meanwhile, the nodes* n_{41}, n_{42}, *and* n_{43} *in the green solid rectangles are the solution nodes with ADPs of* 46.2, 51.6, *and* 43.2 *units, respectively. Finally, the solution node* n_{43} *is returned as the result due to the smallest ADP. By adding the CVs at the final solution node* n_{43}, *we obtain the FV of the final solution as* $(0, 18, 12)$ *with error 0.0178 (relative error* 3.56%*). The final FV is different from the original one. Therefore, the corresponding function (i.e.,* $-0.375x^2 + 1.125x$*) is also different from the target function. This shows an important feature of the DA method—synthesizing the Boolean function and adapting the original FV occur simultaneously. We will return to some additional details of this example search tree later. The ADP of the circuit corresponding to the new target function is* 43.2 *units, which is smaller than that corresponding to the OTBP, which is* 51.2 *units.*

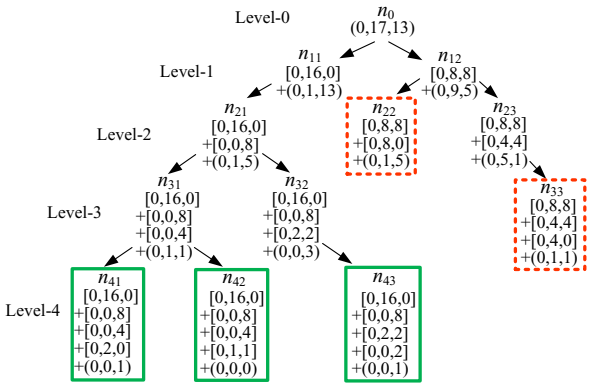

Figure 5: A search tree constructed by the dynamic approximation method.

Next, we will describe the details of the functions *candNodesGen*, *classifyNodes*, and *pruneNodes*.

6.2 Generating the Set of Candidate Nodes

The function *candNodesGen* expands the nodes in the current level of the search tree (i.e., in the set S_{node}) to a set of new candidate nodes. Each candidate node is derived from a node N in S_{node} by adding a *candidate cube* to the SCC of the node N. The procedure of *candNodesGen* is shown in Algorithm 2. The procedure takes S_{node} and a given error bound e_b as inputs and outputs a set R of the candidate nodes.

Algorithm 2: The procedure *candNodesGen*.

```
   Input  : a set S_node containing all nodes in the current level and
            the error bound e_b.
   Output : a set R of the candidate nodes.
1  R ← ∅;
2  foreach node N in S_node do
3      cubeSize ← 2^⌈log2 sum(N.RFV)⌉; L ← ∅;
4      while no cube in L satisfies the capacity constraint of N.RFV
         and cubeSize > 0 do
5          obtain the set S of cube vectors of cubeSize minterms;
6          foreach cube vector V_c in S do
7              RFV' ← modify N.RFV by V_c;
8              NFV ← V_c + add(N.SCC) + RFV';
9              compute error e_est of NTBP corresponding to NFV;
10             if e_est ≤ α e_b then
11                 S_cube ← cubesFromVector(V_c, N, h);
12                 foreach cube C in S_cube do
13                     N_n.RFV ← RFV', N_n.SCC ← N.SCC ⋃ C;
14                     L ← L ⋃ C, R ← R ⋃ N_n;
15         cubeSize ← cubeSize/2;
16 return R;
```

In Algorithm 2, Line 1 initializes R as an empty set. Next, for each node N in S_{node}, new candidate nodes are generated by adding a candidate cube to the SCC of node N (Lines 2–15). The key problem is to select proper candidate cubes.

We call the number of minterms in a cube its size. The size of a cube is 2^q, where q is a non-negative integer. We try to find the candidate cubes with the largest sizes, since they lead to an SOP with a smaller literal count, a measure of the hardware cost we use during the search. In our problem, since we do not require cubes to exactly satisfy the capacity constraint as [12] does, we start to check the cubes with size $cubeSize = 2^{\lceil \log_2 \text{sum}(N.RFV) \rceil}$ (Line 3), where the function $\text{sum}(N.RFV)$ gives the sum of all the entries in the RFV of N. In this case, the cube size can be a little larger than the total minterm number required by the RFV of N. This exploits the opportunity brought by the approximation. As an example, consider the node n_{21} in Fig. 5. Its RFV is $(0, 1, 5)$. Since the sum of all the RFV entries is 6, we start to test the cubes of size 8. Line 3 also initializes a set L as empty. It is used to keep the candidate cubes.

Then, we enter the main loop to obtain the candidate nodes expanded from N (Lines 4–15). The loop continues when no cube in L satisfies the capacity constraint of the RFV of N and the size of the cubes to be checked is larger than zero (Line 4). Since we do not require the candidate cubes to satisfy the capacity constraint, if none of them in L satisfies the constraint, it is possible that we cannot derive a final solution satisfying the given error bound. To ensure that at least one solution is found, we require at least one cube in L to satisfy the capacity constraint of the RFV of N.

Within the loop, we first obtain the set S of CVs of *cubeSize* minterms by enumerating them according to their special forms [12] (Line 5). Next, for each CV V_c in S, we first evaluate the estimated error e_{est} of the potential new FV if any cube corresponding to the CV V_c is chosen as a candidate cube (Lines 7–9). A key idea of the DA method is the simultaneous NTBP search and circuit synthesis. A *new FV (NFV)* is defined as the FV of an NTBP. If e_{est} is smaller than a relaxed error bound, we create candidate nodes by expanding node N with cubes corresponding to V_c (Lines 10–14).

In order to obtain e_{est}, the new RFV RFV' is first obtained (Line 7). This is done by first subtracting V_c from the RFV of N. Since the cube is not required to satisfy the capacity constraint, the subtraction result can have some negative entries. Then, those negative entries are set to zero to get the final RFV'. This essentially modifies the original FV. As an example, the RFV of the node n_{21} in Fig. 5 is $(0, 1, 5)$. As a cube with the CV $[0, 2, 2]$ violating the capacity constraint is added to this node, the RFV first becomes $(0, -1, 3)$ by subtracting the CV, and then is modified to $(0, 0, 3)$. This leads to the node n_{32} in Fig. 5. Line 8 obtains the potential NFV by adding V_c, CVs of all the cubes in SCC of N, and RFV', where the function $add(N.SCC)$ gives the sum of CVs of all the cubes in SCC of N. For the example in Fig. 5, when the CV $[0, 2, 2]$ is considered for n_{21}, the RFV $(0, 1, 5)$ of n_{21} is first modified to RFV' as $(0, 0, 3)$. Then, by adding the CV $[0, 2, 2]$, the CVs $[0, 16, 0]$ and $[0, 0, 8]$ of the cubes in n_{21}'s SCC, and RFV', the potential NFV is obtained as $(0, 18, 13)$, which is different from the original FV $(0, 17, 13)$. Line 9 further calculates an *estimated error* e_{est} of the NTBP corresponding to the potential NFV.

Then, Lines 10–14 add the cubes corresponding to V_c to SCC of node N to expand it to new nodes if the estimated error e_{est} is less than a relaxed error bound αe_b, where $\alpha > 1$. The relaxed error bound is used because e_{est} is just an estimated error. By the later function *classifyNodes*, when a node is expanded into a solution node, its RFV may be ignored, as is the node n_{43} in Fig. 5. This may lead to the error drop. In this case, the previous estimated error is an overestimate. Given this reason, for a CV, even if $e_{est} > e_b$, the node may still be developed into a final solution. In order to keep such promising CVs, the relaxed error bound is applied during the search. We set α to 1.02 in this study. If $e_{est} \leq \alpha e_b$, Line 11 calls the function *cubesFromVector* from [12] to obtain a set of cubes S_{cube} with CV V_c. The function takes V_c, N, and an additional parameter h as inputs, where h limits the size of S_{cube} to trade solution quality with runtime. Each cube C in S_{cube} is a candidate cube. For each C, Line 13 creates a candidate node N_n from N and C by setting N_n's RFV and SCC properly. Line 14 adds C and N_n into the sets L and R, respectively.

When all CVs in S have been evaluated, Line 15 reduces *cubeSize* to evaluate the cubes with their sizes halved in the next iteration. After all the nodes in S_{node} are processed, Line 16 returns R.

6.3 Classifying the Candidate Nodes

The function *classifyNodes* partitions the set R of candidate nodes into two sets, namely the set S_{new} of the nodes to be further evaluated and the set S_{sol} of the solution nodes. For each node N in R, the function evaluates the error of its assigned FV, where the assigned FV is the sum of the CVs of all the cubes in its SCC. If the error is no larger than e_b, the node N is considered as a solution node and added into the set S_{sol}, even though the RFV of N is possibly non-zero. Otherwise, it is considered as a node to be possibly passed to the next level and added into the set S_{new}.

6.4 Pruning Redundant Nodes and Unpromising Nodes

After we obtain the set S_{new} of the nodes to be possibly passed to the next level, the function *pruneNodes* further prunes the redundant ones and the unpromising ones.

If multiple nodes in S_{new} have the same truth table given by their SCCs, only one of them is further kept in S_{new} to eliminate redundant evaluation. For example, in Fig. 5, the nodes n_{22} and n_{33} are pruned since they have the same assigned truth tables as the nodes n_{21} and n_{31}, respectively.

Besides pruning the redundant nodes, the function *pruneNodes* also prunes the unpromising nodes. For our problem, its objective is to find a circuit with good quality, while satisfying the error constraint. However, there is a potential contradiction between the objective and the constraint. If we only keep the nodes with better circuit quality, their errors may be large and thus they will be pruned later. However, if only the nodes with small errors are kept, the nodes with better quality may be dropped, leading to a sub-optimal result. To solve this problem, the function *pruneNodes* further uses two passes of sorting and pruning. It has three pruning parameters w, k_L, and k_E.

For the first pass, *pruneNodes* first sorts the nodes in S_{new} by the estimated error e_{est} in ascending order. Then, it sorts the nodes with the same e_{est} by the literal count in ascending order. The first k_E sorted nodes are added into the set S_{kept}. The second pass works on the remaining nodes. It first sorts them by the literal count. A heuristic in [12] is adopted to prune the nodes with w literals more than the minimum among all the remaining nodes. It further sorts the nodes with the same literal count by e_{est} in ascending order. Finally, the first k_L sorted nodes are added into the set S_{kept}, and S_{kept} is returned as the result. For example, in Fig. 5, the set S_{new} at Level 3 contains two non-redundant nodes n_{31} and n_{32}. With k_E and k_L both set as 1, the nodes n_{31} and n_{32} are added to S_{kept} by the first and second passes, respectively. From these two passes, we can keep both the nodes with small estimated error and literal count to ensure finding a valid solution and achieving good quality.

7 Experimental Results

In this section, we show the experimental results. All the experiments were done on a computer with an Intel CPU of 2.93GHz and an 8GB memory. In the function *getBest* called by the proposed DA method (see Line 7 of Algorithm 1), the logic synthesis tool ABC [10] was applied with the commands "*collapse; sop; fx; strash; dch; balance; map*" to synthesize a multi-level SC circuit and obtain its ADP. In all the following experiments, ABC was also used to optimize the SC circuits generated by the previous methods and the proposed PER and DPS methods for a fair comparison. All the circuits were mapped with the MCNC standard cell library [15].

7.1 Performance on Arithmetic Functions

In this section, we study the performance of the proposed methods on 12 target functions from [13] and [11], which are listed in Table 1. For each target function f, we set the error bound e_b to $\beta||f||_2$, where $0 < \beta < 1$ is an adjustable relative error bound and $||f||_2$ is the L2-norm of f in the unit interval. The relative error bounds were chosen as 2% and 5%. We considered 4 degree-precision pairs (n, m) as $(4, 4)$, $(4, 8)$, $(6, 4)$, and $(6, 8)$. This gives 48 benchmarks in total. However, two of them have original errors e_{org} larger than the error bound e_b, and were excluded due to the requirement that $e_b > e_{org}$. Thus, 46 benchmarks were finally chosen.

The prior state-of-the-art method [12] was chosen as the *baseline (BS) method* for comparison. Our methods include the PER, DPS, and

Table 1: The target functions used in our experiments.

ID	function	ID	function	ID	function	ID	function
1	$\sin(x)$	4	$\log(x+1)$	7	$\tanh(4x)$	10	$1/(1+\exp(-x))$
2	$\cos(x)$	5	$\sin(\pi x)/\pi$	8	$x^{0.45}$	11	$x^{2.2}$
3	$\exp(-x)$	6	$\tanh(x)$	9	$\exp(-2x)$	12	$0.5\cos(\pi x)+0.5$

DA methods. The proposed PER and DPS methods take a long time to evaluate all the NTBPs in the candidate set. For a fair comparison with the baseline, for these two methods, we set a runtime bound for each benchmark as 2.5 times the runtime of the baseline. Once the runtime bound was reached, we stopped the search even if some candidate NTBPs had not been evaluated, and chose the current best result. For the proposed DA method, 4 parameters h, w, k_L, and k_E were used to control the solution quality and the runtime, where h is described in Section 6.2 and w, k_L, and k_E are pruning parameters described in Section 6.4. From extensive parameter study, we set k_E=1 to ensure the finding of a solution, and set h=1, w=2, and k_L=4 to ensure a good trade-off between the runtime and solution quality. For the BS method, three parameters h, w, and k_L were used [12], and we set them as h=2, w=2, and k_L=5. This allows it to run for a longer time than the DA method, which ensures sufficient exploration of the solution space. For the PER and DPS methods, which are based on the BS method, the same parameters were used.

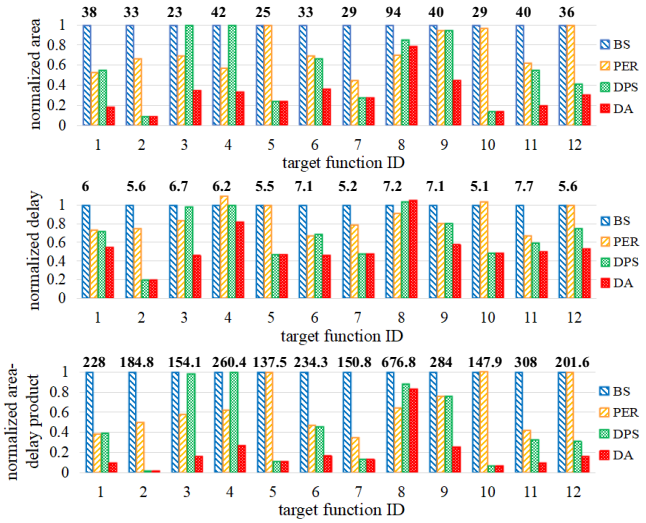

Figure 6: The area, delay, and area-delay product normalized to the BS method for the 12 arithmetic benchmarks with (n, m) set to $(6, 4)$. The results by the BS method are listed above each column.

Fig. 6 plots the area, delay, and ADPs of the circuits produced by our methods normalized to those of the baseline for the 12 target functions with (n, m) set to $(6, 4)$ and a 2% relative error bound. For most of these target functions, the proposed PER, DPS, and DA methods all synthesize SC circuits with smaller area, delay, and ADP than the BS method. For some target functions, the circuits synthesized by the PER method are better than those synthesized by the DPS method, while for others, the DPS method is better. For all the target functions except the 8th one, the DA method synthesizes the best circuits with the smallest area, delay, and ADP. For the DA method, the ADP of the synthesized circuits for 9 out of the 12 target functions is reduced by more than 80%. This is because as up to 2% relative error is allowed, the DA method finds an NTBP with FV different from the original FV, which leads to an SOP Boolean function with much fewer cubes.

Table 2 lists the average hardware cost, the average mean absolute error (MAE), and the average synthesis runtime over the 46 benchmarks together with the relative improvement of our methods over the BS method (in parentheses). To evaluate the MAE, 100 simulations were done for each of the 9 x values $0.1, 0.2, \ldots, 0.9$, and the MAE was obtained by averaging the absolute errors with respect to the target function over all these 900 tests. The bit stream length for each simulation was 1024. For the average area, delay, and ADP, all the proposed methods are better than the baseline. The DA method is better than the SA methods (i.e., PER and DPS). For the relative error bound of 2% (resp. 5%), it reduces the area, delay, and ADP over the baseline by 71% (resp. 79%), 44% (resp. 50%), and 81% (resp. 89%), respectively, with a 12% (resp. 62%) MAE degradation. Since the average MAE of the baseline is as small as 0.0105, such a relative MAE degradation is not large in terms of the magnitude. This shows the effectiveness of the proposed methods.

Remarkably, for some target functions, the proposed DA method can aggressively minimize the SC circuits. One notable example is $\cos(x)$. As mentioned in Section 1, it can be realized by a *single NAND3* gate. In fact, it is synthesized by the DA method under 2% relative error bound with (n, m) as $(6, 8)$. It has an ADP reduction of 99.5% over the SC circuit synthesized by the BS method. In terms of the gate count, the DA method finds the *optimal* solution for this case, showing its optimality. Interestingly, the approximated target function $1-0.5x^2$ is exactly the degree-2 Maclaurin polynomial of $\cos(x)$, indicating that the DA method finds a quite reasonable approximation. Moreover, the SC circuit synthesized by the DA method requires only 3 input bit streams instead of 14 required by the one synthesized by the BS method. Therefore, it also *greatly reduces the cost of the stochastic number generator*. Besides, for the target functions $\tanh(x)$ and $1/(1+\exp(-x))$, the DA method can achieve 93.3% and 97.5% ADP reduction, respectively, both with only 0.001 MAE degradation compared to the BS method.

In terms of the average runtime, the proposed SA methods are slower than the baseline. In contrast, the proposed DA method is faster. It runs even faster with a larger error bound. This shows another advantage of the DA method: it can terminate the solution search early without having to reach a zero RFV, while the baseline method must continue the search until a zero RFV is reached. Overall, the proposed DA method shows significant advantages over the baseline: *it reduces the hardware cost dramatically, while being faster than the baseline*. The DA method also outperforms the other two proposed methods, PER and DPS, in terms of efficiency and solution quality. Therefore, the DA method is the best among the three proposed methods.

7.2 Comparison to the Maclaurin Polynomial Approximation Method

In this section, we further compared the DA method to another state-of-the-art SC circuit synthesis method [11]. It approximates the target function with a degree-n Maclaurin polynomial and synthesizes the circuit based on the factorization of the Maclaurin polynomial. We call it the *Maclaurin polynomial (MP) method*. We

Table 2: Average hardware cost and synthesis runtime for the arithmetic benchmarks.

relative	average area				average delay				average ADP				average MAE				average runtime (s)			
error bound	BS	PER	DPS	DA	BS	PER	DPS	DA	BS	PER	DPS	DA	BS	PER	DPS	DA	BS	PER	DPS	DA
2%	47.2	38.3 (19%)	18.1 (62%)	13.5 (71%)	6.2	5.5 (11%)	4.0 (36%)	3.5 (44%)	334.4	244.8 (27%)	94.8 (72%)	64.3 (81%)	0.0105	0.0106 (-2%)	0.0112 (-7%)	0.0117 (-12%)	98.1	195.4 (-99%)	182.3 (-86%)	93.5 (5%)
5%	47.2	37.9 (20%	15 (68%)	10.1 (79%)	6.2	5.4 (12%)	3.7 (40%)	3.1 (50%)	334.4	242 (28%)	69.8 (79%)	37.9 (89%)	0.0105	0.0108 (-3%)	0.0132 (-26%)	0.0169 (-62%)	98.1	195.7 (-99%)	193.3 (-97%)	62.3 (37%)

selected 8 out of the 12 target functions from Table 1 for evaluation. Their IDs are listed in the first column of Table 3, together with the degrees of the Maclaurin polynomials for the MP method shown in the parentheses (MP degree). The other target functions were not evaluated since for up to the degree of 20, their Maclaurin polynomials cannot satisfy the coefficient constraints required by the MP method. For the DA method, $(n, m) = (6, 8)$ was used to obtain the OTBPs from the target functions, and the relative error bound was set as 2%.

For each selected target function, we used the method of [11] to synthesize the SC core circuit, while the circuits to generate the bit streams with constant probabilities were synthesized by the method of [3]. These required constant probabilities are transformed from independent bit streams of probability 0.5. Moreover, since the SC circuits synthesized by the DA method are combinational, for a fair comparison, the SC circuits synthesized by the MP method were also transformed into equivalent combinational circuits. Thus, the x^2 terms in it were realized by an *AND2* gate with two independent input bit streams of probability x instead of a delay element [11].

Table 3: Experimental Results for the MP and DA methods. The relative error bound of the DA method is 2%.

target function	area		delay		ADP		MAE	
ID (MP degree)	MP	DA	MP	DA	MP	DA	MP	DA
1 (7)	56	7	7.5	3.3	420	23.1	0.0107	0.0126
2 (6)	41	3	5.5	1.1	225.5	3.3	0.00804	0.0106
3 (6)	60	8	9	3.1	540	24.8	0.0117	0.0129
4 (7)	81	29	10.9	5.7	882.9	165.3	0.0136	0.0126
5 (9)	41	6	6	2.6	246	15.6	0.00976	0.00985
6 (5)	36	12	6.8	3.3	244.8	39.6	0.0136	0.0121
9 (6)	85	18	10.9	4.1	926.5	73.8	0.0110	0.0125
10 (5)	42	4	6.3	2.5	264.6	10	0.0124	0.0135
average	55.3	10.9 (80%)	7.9	3.2 (59%)	468.8	44.4 (91%)	0.0113	0.0121 (-6.5%)

The experimental results for area, delay, ADP, and MAE are shown in Table 3, where the MAE was evaluated by the same approach as in Section 7.1. The DA method with 2% relative error bound improves the circuit area, delay, and ADP by 80%, 59%, and 91%, respectively, over the MP method on average, with only 6.5% degradation of MAE. Note that the computation latency has the same reduction as the delay. Therefore, the proposed DA method synthesizes a much smaller and faster SC circuit than the MP method with only a little MAE degradation.

7.3 Case Study: Gamma Correction

In this section, we evaluated the DA method for an image processing application, gamma correction, which has the target function $x^{0.45}$. We compared it with the BS method. We set (n, m) to $(4, 4)$. The area, delay, and ADPs of the circuits produced by the two methods are listed in Table 4, showing that the proposed DA method synthesizes a much better circuit than the BS method with a given error bound. Specifically, the circuit synthesized by the DA method has a significant reduction in ADP by 35%, over the BS method for a 2% relative error bound. As the relative error bound increases from 2% to 5%, the ADP of the circuit is further reduced by 16%.

Table 4: The hardware cost of the circuits and the average PSNR and average worst-case absolute error (WAE) of processed images for gamma correction.

relative error	area		delay		ADP		average PSNR (dB)		average WAE	
bound	BS	DA	BS	DA	BS	DA	BS	DA	BS	DA
2%	34	25 (26%)	5.2	4.6 (12%)	176.8	115 (35%)	34.15	33.54 (-1.8%)	0.117	0.122 (-4.1%)
5%	34	19 (44%)	5.2	4.6 (12%)	176.8	87.4 (51%)	34.15	32.08 (-6.1%)	0.117	0.127 (-8.1%)

(a) (b) (c) (d)

Figure 7: Images for the gamma correction experiment. (a): input image; (b): reference output image; (c): output image by the SC circuit synthesized by the BS method; (d): output image by the SC circuit synthesized by the DA method for a 5% relative error bound.

The processed images of a sample are shown in Fig. 7. Compared with the images in Figs. 7(b) and (c), the image produced by the DA method shows no significant quality degradation.

In order to quantitatively evaluate the quality of the images processed by each circuit, we used 10 input images from [6] for testing. Since an SC circuit has random errors, 100 random simulations were done for each image with the stochastic bit stream length set to 512. The average peak signal-to-noise ratio (PSNR) and the average worst-case absolute error (WAE) over the 100 simulations for each image was obtained, and the averages of these mean values over the 10 images are listed in Table 4. For the DA method, both the average PSNR and WAE show small relative degradation over the BS method for 2% relative error bound. As the relative error bound increases to 5%, the average PSNR and WAE further degrade. However, such degradation is not serious as shown in Fig. 7, but a further 16% ADP improvement is achieved by the DA method.

8 Conclusions

In this work, we explore the target function approximation to minimize an SC circuit. We proposed two static and one dynamic approximation methods to identify a good target function approximation and the corresponding SC circuit. All the proposed methods can produce an SC circuit with much smaller area, delay, and area-delay product than the prior state-of-the-art methods. Furthermore, as the most effective and efficient proposed method, the dynamic method also shows runtime advantage over the prior method [12].

References

[1] A. Alaghi et al. 2013. Stochastic Circuits for Real-Time Image-Processing Applications. In *DAC*. 136:1–136:6.

[2] A. Alaghi et al. 2018. The Promise and Challenge of Stochastic Computing. *IEEE TCAD* 37(8) (2018), 1515–1531.

[3] A. Alaghi and J. P. Hayes. 2015. STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis. *IEEE TCAD* 34(11) (2015), 1770–1783.

[4] A. Ardakani et al. 2017. VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing. *IEEE TVLSI* 25(10) (2017), 2688–2699.

[5] T. H. Chen and J. P. Hayes. 2015. Equivalence among Stochastic Logic Circuits and its Application. In *DAC*. 131:1–131:6.

[6] Imageprocessingplace. 2019. Image Database. http://imageprocessingplace.com/root_files_V3/image_databases.htm

[7] P. Li et al. 2012. The Synthesis of Complex Arithmetic Computation on Stochastic Bit Streams Using Sequential Logic. In *ICCAD*. 480–487.

[8] P. Li et al. 2014. Computation on Stochastic Bit Streams: Digital Image Processing Case Studies. *IEEE TVLSI* 22(3) (2014), 449–462.

[9] G. Lorentz. 1953. *Bernstein Polynomials*. University of Toronto Press.

[10] A. Mishchenko. 2012. ABC Logic Synthesis Package. https://people.eecs.berkeley.edu/~alanmi/abc/abc.htm

[11] K. K. Parhi and Y. Liu. 2019. Computing Arithmetic Functions Using Stochastic Logic by Series Expansion. *IEEE TETC* 7(1) (2019), 44–59.

[12] X. Peng and W. Qian. 2018. Stochastic Circuit Synthesis by Cube Assignment. *IEEE TCAD* 37(12) (2018), 3109–3122.

[13] W. Qian et al. 2011. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *IEEE TC* 60(1) (2011), 93–105.

[14] H. Sim and J. Lee. 2017. A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks. In *DAC*. 29:1–29:6.

[15] S. Yang. 1991. *Logic synthesis and optimization benchmarks*. Technical Report. Microelectronics Center of North Carolina.

[16] Y. Zhang et al. 2020. When Sorting Network Meets Parallel Bitstreams: A Fault-Tolerant Parallel Ternary Neural Network Accelerator based on Stochastic Computing. In *DATE*. 1287–1290.

[17] Z. Zhao and W. Qian. 2015. A General Design of Stochastic Circuit and its Synthesis. In *DATE*. 1467–1472.