MinSC: An Exact Synthesis-Based Method for Minimal-Area Stochastic Circuits under Relaxed Error Bound

Xuan Wang[1], Zhufei Chu[2], and Weikang Qian[1,3]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China
[3]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
Emails: xuan.wang@sjtu.edu.cn, chuzhufei@nbu.edu.cn, qianwk@sjtu.edu.cn

***Abstract*—Stochastic computing (SC) operates on stochastic bit streams, which can realize complex arithmetic functions with simple circuits. A previous work shows that by introducing a little approximation error for the target function, the cost of SC circuits can be dramatically reduced. However, the previous heuristic method only explores a limited subset of the solution space, so the optimality of the results cannot be guaranteed. In this paper, we propose MinSC, an exact synthesis-based method for minimal-area stochastic circuits under relaxed error bound. First, a novel search method is proposed to find the best approximation polynomial for a target function. Then, considering gates with different fanin numbers and areas, an exact SC synthesis method using satisfiability modulo theories is designed to obtain an area-optimal SC circuit realizing the best approximation polynomial. The experimental results show that compared with the state-of-the-art method, given an error ratio 0.05, MinSC on average reduces the gate number, area, delay, and area-delay-product of the SC circuits by 60.24%, 47.24%, 7.10%, 57.07%, respectively.**

***Index Terms*—stochastic computing, stochastic circuit synthesis, exact synthesis, satisfiability modulo theories (SMT).**

I. Introduction

With the development of CMOS technology, reliability has become a great concern. Stochastic computing (SC), a re-emerging computing paradigm [1], has received much attention due to its strong fault tolerance. Instead of using binary radix encoding, SC encodes a real value $x \in [0, 1]$ by a stochastic bit stream, where x is the probability of being a 1 for each bit. SC can use simple digital circuits to realize many complex arithmetic functions. For example, the multiplication of two numbers can be realized by a simple AND gate. Besides, it is tolerant to bit flipping errors. Due to these advantages, SC has been applied successfully to several applications, including error-correcting coding [2], [3], image processing [4], [5] and neural networks [6]–[8].

In recent years, many methods are proposed for the synthesis of SC circuits. The work [9] proposes a method to synthesize sequential SC circuits. In [10], a method is designed to synthesize reconfigurable combinational SC circuits. Besides, there are several works for the synthesis of fixed combinational SC circuits [11]–[14], which is also the focus of our work. The work [11] proposes a spectral transform-based method to synthesize SC circuits. The work [12] proposes to search for the SC circuits within an equivalence class. In [13], another method is proposed that finds a good SC circuit through a heuristic breadth-first search algorithm. The work [14] synthesizes SC circuits based on truncated Maclaurin series polynomials.

However, these existing works still have the following issues.

- Issue 1: To realize a target function, existing methods first approximate the function by the closest polynomial with the minimal error and then synthesize an SC circuit for that polynomial. However, realizing the closest polynomial usually leads to a large hardware cost.
- Issue 2: As shown in [12], [15], there are many different Boolean functions realizing the same approximation polynomial, which results in an extremely large solution space. In [11]–[14], they only search a limited part of the space, so the results are sub-optimal.

For some error-tolerant applications (e.g., image processing and neural networks), the approximation error can be relaxed to some extent. With the relaxed approximation error, we may find a better approximation polynomial for the target function, which can realize an SC circuit with reduced cost. This idea can address the above Issue 1. In [16], given an error bound and a target function, a dynamic approximation (DA) method is proposed to heuristically search a good approximation polynomial and synthesize the corresponding SC circuit simultaneously. However, the method still has the following drawbacks: 1) It does not search for the whole space to obtain the best approximation polynomial, and the quality of the approximation polynomial depends on the starting point of search; 2) The above Issue 2 still remains: given a target approximation polynomial, there are many different Boolean functions realizing it. However, the heuristic method in [16] only explores a limited subset of the solution space. Therefore, the obtained solution is still not optimal.

To reach the optimal solution, we apply exact synthesis to generate SC circuits in this work. Exact synthesis is an approach of finding an optimum Boolean network (e.g., in size or depth) for a given Boolean function [17]. Several prior works apply exact synthesis for traditional logic circuits. The work [18] compares different conjunction normal form (CNF) encoding schemes and conducts satisfiability (SAT)-based exact synthesis under different constraints. In [19], SAT-based exact synthesis is applied to find a minimum-size network under delay constraints. The work [17] compares different CNF encoding schemes and uses topology information to leverage parallelism. In [20], majority-inverter graphs (MIGs) are used as logic representation, and satisfiability modulo theories (SMT)-based exact synthesis is applied to minimize the MIG's size and depth.

However, due to the exponential complexity, exact synthesis can only find optimum solutions for small-scale circuits. Fortunately, as many SC circuits only require a small number of gates, it is suitable for SC circuit synthesis. As a preliminary attempt, the work [21] applies SAT-based exact synthesis to realize an SC circuit with the fewest gates, where SC circuits are represented as MIGs. However, due to the complex constraints of SC circuits, it can only synthesize very simple SC circuits. In addition, all the existing exact synthesis methods have one common shortcoming, i.e., they do not take gate area into account, as they only work on a technology-independent representation of circuits. Although they can obtain circuits with the fewest gates, the circuit area may not be optimal after technology mapping, due to a gap between technology-independent synthesis and technology mapping.

In this paper, by addressing the aforementioned challenges, we propose MinSC, an exact synthesis-based method for minimal-area

This work is supported in part by the National Key R&D Program of China under Grant 2020YFB2205501, National Natural Science Foundation of China under Grant 61871242, and the State Key Laboratory of ASIC & System under Grant 2021KF008. Corresponding author: Weikang Qian.

stochastic circuits under relaxed error bound. First, given a target function and an error bound, a novel search method is proposed to find the best approximation polynomial for the given target. Then, an SMT-based exact SC synthesis method is proposed to obtain an *area-optimal* SC circuit realizing the best approximation polynomial. The main contributions of our work are listed as follows.

- We propose a novel search method to find the best approximation polynomial for the target function. This method searches directly based on the input error bound, which guarantees the optimality of the best approximation polynomial.
- To directly reach an area-optimal circuit, we propose a method to take gate area into account during exact synthesis, which has never been explored before. With this method, an SMT-based exact synthesis method is designed to synthesize an area-optimal SC circuit directly with no need of technology mapping.
- To synthesize complex SC circuits, we propose two novel techniques to reduce the search space with minimal influence to the optimality: partial one assignment (POA) and multi-granularity search (MGS).

The experimental results show that compared with the state-of-the-art DA method [16], given an error ratio 0.05, MinSC reduces the gate number, area, delay, and area-delay-product (ADP) of SC circuits by 60.24%, 47.24%, 7.10%, and 57.07%, respectively. The code of our proposed method is made open-source at https://github.com/SJTU-ECTL/MinSC.

The rest of the paper is organized as follows. Section II provides the preliminaries. Section III presents the proposed MinSC method. Section IV discusses several speed-up techniques. Section V shows the experimental results. Section VI concludes the paper.

II. Preliminaries

In this section, we introduce a general model of SC circuits, a basic synthesis flow for traditional SC circuits, and the background of exact synthesis.

A. A General Model of SC Circuits

A general model of an SC combinational circuit is shown in Fig. 1, which is proposed in [15]. The first n input independent bit streams are for the input variable x, while the next m input independent bit streams with constant 0.5 are the coefficients. Suppose that the Boolean function of the combinational circuit is F. According to [13], the output function of this SC circuit is

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

where $G(i)$ are integers denoting the number of input combinations $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ satisfying that $F(X_1, \ldots, X_n, Y_1, \ldots, Y_m) = 1$ and $\sum_{j=1}^{n} X_j = i$. The range of $G(i)$ is $0 \le G(i) \le \binom{n}{i} \times 2^m$. The vector $G = (G(0), \ldots, G(n))$ is called *feature vector (FV)* [16]. The function $g_{n,m}(x)$ is called *SC polynomial (SCP)*. The parameters n and m are the *degree* and the *precision* of $g_{n,m}(x)$, respectively. We call the sum of the degree and precision of an SCP its *degree-precision sum (DPS)*.

By the above discussion, the FV G is determined by the Boolean function F. If we represent F as a 2-dimensional truth table with $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ defining the columns and rows, respectively, then by the definition of $G(i)$, it corresponds to the number of 1s in the columns with $\sum_{j=1}^{n} X_j = i$. Table I shows an example of such a 2-dimensional table for a Boolean function F on X_1, X_2, Y_1. Then, $G(1)$ is the number of 1s in the columns with X_1X_2 being 01 and 10, as $X_1 + X_2 = 1$. In this case, $G(1) = 2$.

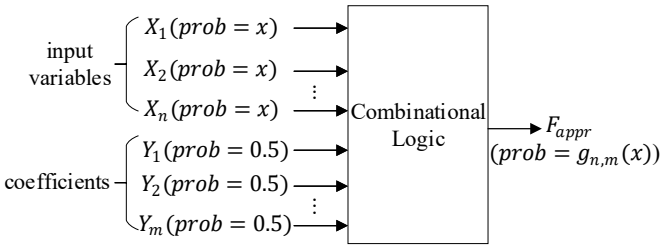

Fig. 1. A general model of an SC circuit realizing the target function.

Table I. The 2-dimensional truth table of a Boolean function with $n = 2$ and $m = 1$.

$Y_1 \backslash X_1X_2$	00	01	10	11
0	1	0	1	0
1	0	1	0	1

B. Traditional SC Circuit Synthesis and a Basic Synthesis Flow

The traditional SC circuit synthesis produces an SC circuit with a small cost realizing a given target arithmetic function $f(x)$. A basic SC synthesis flow involves the following two steps [13].

1) Target function approximation: Given a target function $f(x)$ and a degree n, a quadratic programming method proposed in [10] is applied to obtain a degree-n Bernstein polynomial (BP) $B_n(x)$ closest to $f(x)$ in the form of

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

where b_i is a *Bernstein coefficient* and $\binom{n}{i} x^i (1-x)^{n-i}$ is a *Bernstein basis polynomial*. Then, given a precision m, by setting

$$G(i) = \text{round}\left(2^m b_i \binom{n}{i}\right), \tag{3}$$

we round the real coefficients in Eq. (2) to integers to obtain the SCP $g_{n,m}(x)$ shown in Eq. (1). Thus, $g_{n,m}(x) \approx B_n(x) \approx f(x)$.

2) SC circuit synthesis: Given the SCP from the previous step, we further obtain an SC circuit with a small cost for the SCP. This is essentially to determine a Boolean function so that 1) its corresponding FV equals the FV of the SCP and 2) it leads to a circuit with a small cost. The state-of-the-art work [13] uses a heuristic breadth-first search algorithm to find a good but sub-optimal SC circuit.

C. Exact Synthesis

Exact synthesis is the problem of finding a logic network that exactly meets a specification (e.g., the number of gates in the network) [17]. It can be applied to find an optimum logic network for a given Boolean function. A logic network is called a *Boolean chain* in exact synthesis, which is a directed acyclic graph (DAG). Suppose that K-input operators are used, where K is an arbitrary but fixed value. Given n primary inputs (PIs) $x_1, \ldots, x_n$, a Boolean chain of r K-input operators can be represented as a sequence of gates $x_{n+1}, \ldots, x_{n+r}$ with

$$x_i = \phi_i(x_{j(i,1)}, \ldots, x_{j(i,K)}),\ n+1 \le i \le n+r, \tag{4}$$

where ϕ_i is the operator of gate x_i and $x_{j(i,\cdot)}$ is the fanin of gate x_i with $1 \le j(i,\cdot) < i$ [17]. For a single-output function, the output of a Boolean chain is the last gate x_{n+r}. For a multi-output function, the outputs can be any gate x_i. As an SC circuit is single-output, we only consider single-output Boolean functions in this paper.

Example 1 *A Boolean chain is shown in Fig. 2. It has three PIs* x_1, x_2, x_3 *and two gates* x_4, x_5*, where* $x_4 = x_2 \vee x_3$ *and* $x_5 = x_1 \oplus x_4$*. The Boolean chain has only one output* $g_1 = x_5$*.*

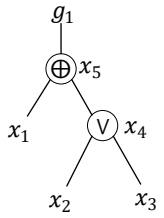

Fig. 2. An example of a Boolean chain.

To solve an exact synthesis problem, a formulation with proper encoding variables and constraints on them is built to ensure the meeting of the specification. If the formulation contains integer or real coefficients, integer or real variables, and linear arithmetic constraints, it can be transformed into an SMT problem and solved by SMT solvers.

III. Proposed Method

In this section, we present our proposed MinSC method to synthesize an area-optimal SC circuit for a given target function and an error bound.

A. Overview of the Proposed Method

The basic idea of MinSC is that given a target function and an error bound, first we find a set of good approximate SCPs (ASCPs). Usually, the smaller the DPS an ASCP has, the smaller the cost of the corresponding SC circuit. Thus, we first aim at finding the ASCPs with the smallest DPS satisfying the error bound (Section III-B).[1] This is different from the target function approximation in the basic synthesis flow shown in Section II-B1, which finds the closest SCP with a given n and m for the target function. Once the ASCPs with the smallest DPS have been found, instead of using the heuristic SC synthesis method as before, an SMT-based exact synthesis method is proposed to obtain the area-optimal SC circuits (Section III-C).

The main procedure of MinSC is shown in Algorithm 1. The inputs are the target function $f(x)$, a given error bound e_b, and the maximum difference vector (MDV) V_l, which is used to generate the candidate ASCPs for the target function. The output of the algorithm is an area-optimal SC circuit. In this work, the error metric can be any error measures, such as mean absolute error (MAE), mean square error (MSE), and worst-case absolute error (WCAE).

In Algorithm 1, set S holds the candidate SC circuits. Line 1 initializes it as an empty set. Line 2 calls the function *ASCPSearch* to generate the ASCP set S_{SCP} with the smallest DPS for the target function. For each ASCP g in S_{SCP}, we apply an SMT-based exact synthesis method to obtain the area-optimal SC circuits for the ASCP (Lines 4 and 5). Specifically, Line 4 calls the function *gateOptimalES* to obtain a gate-number-optimal SC circuit C_{gate}. Line 5 calls the function *areaOptimalES* to obtain an area-optimal SC circuit C_{area} for the ASCP g, using the circuit C_{gate} as a starting design. Then, Line 6 adds C_{area} into S. Line 7 chooses the SC circuit C^*_{area} with the smallest area from the set S and returns it.

Next, we will describe the details of the functions *ASCPSearch*, *gateOptimalES*, and *areaOptimalES*.

[1]Strictly speaking, the ASCP with the smallest DPS may not always lead to the optimal circuit cost. However, we find that in most cases, it is true. Thus, for efficiency consideration, we focus on ASCPs with the smallest DPS.

Algorithm 1: The proposed MinSC method.

Input : the target function $f(x)$, an error bound e_b, and the MDV V_l.
Output: an area-optimal SC circuit C_{area}.
1 $S \leftarrow \emptyset$;
2 $S_{SCP} \leftarrow$ *ASCPSearch*$(f(x), e_b, V_l)$;
3 **foreach** $g \in S_{SCP}$ **do**
4 $\quad C_{gate} \leftarrow$ *gateOptimalES*(g);
5 $\quad C_{area} \leftarrow$ *areaOptimalES*(g, C_{gate});
6 $\quad S \leftarrow S \cup \{C_{area}\}$;
7 **return** *the SC circuit* C^*_{area} *with the smallest area in* S;

B. Finding ASCP Set with the Smallest Degree-Precision Sum

In this section, we present the details for finding the ASCPs with the smallest DPS for the target function satisfying the error bound. The whole procedure is shown in Algorithm 2. The basic idea is to first find the lower bound on the degree of the ASCPs satisfying the error bound, i.e., n_{min}, which also gives a lower bound on the smallest DPS (Lines 2–3). From n_{min}, an upper bound on the smallest DPS, i.e., dps^{new}_{up}, is also found (Lines 4–9). Then, we search from dps^{new}_{up} downwards to derive the ASCPs with the smallest DPS (Lines 10–13).

Algorithm 2: The procedure *ASCPSearch*.

Input : the target function $f(x)$, an error bound e_b, and the MDV V_l.
Output: the ASCP set S_{SCP} with the smallest DPS.
1 $S_{SCP} \leftarrow \emptyset$; $dps^{new}_{up} \leftarrow -1$;
2 **for** *n from* 1 *to* $+\infty$ **do**
3 $\quad$ **if** *error*$(B_n(x), f(x)) \le e_b$ **then** $n_{min} \leftarrow n$; **break**;
4 **for** *dps from* n_{min} *to* $+\infty$ **do**
5 $\quad$ Generate a set P of degree-precision pairs (n, m) with $n + m = dps$ and $n \ge n_{min}$;
6 $\quad$ **foreach** $(n, m) \in P$ **do**
7 $\quad\quad$ **if** *error*$(\hat{g}_{n,m}(x), f(x)) \le e_b$ **then**
8 $\quad\quad\quad$ $dps^{new}_{up} \leftarrow dps$; **break**;
9 $\quad$ **if** $dps^{new}_{up} > 0$ **then break**;
10 **do**
11 $\quad$ $dps^{old}_{up} \leftarrow dps^{new}_{up}$;
12 $\quad$ $\{dps^{new}_{up}, S_{SCP}\} \leftarrow$ *perturbSearch*$(f(x), e_b, V_l, n_{min}, dps^{old}_{up})$;
13 **while** $dps^{new}_{up} < dps^{old}_{up}$;
14 **return** S_{SCP};

The inputs of this procedure are the target function $f(x)$, an error bound e_b, and the MDV V_l. The output is the set S_{SCP} of ASCPs with the smallest DPS. In Algorithm 2, Line 1 initializes S_{SCP} as an empty set and dps^{new}_{up} as -1. Lines 2–3 find the smallest degree n (i.e., n_{min}) so that the error between $B_n(x)$ and $f(x)$ is no more than the error bound e_b, where $B_n(x)$ is the degree-n BP closest to $f(x)$ obtained by the method in [10]. As the precision $m \ge 0$, the smallest DPS is at least n_{min}, which means that n_{min} also gives a lower bound on the smallest DPS.

Lines 4–9 search the upper bound on the smallest DPS, i.e., dps^{new}_{up}. To reduce the unnecessary computation, the search starts from the lower bound on the smallest DPS we just obtain, i.e., n_{min} (see Line 4). For a given DPS dps, Line 5 generates a set P of degree-precision pairs (n, m) satisfying that $n + m = dps$ and $n \ge n_{min}$. For each degree-precision pair (n, m), Line 7 obtains the initial SCP $\hat{g}_{n,m}(x)$ with degree n and precision m for the target function $f(x)$ by the method described in Section II-B1, and calculates the error

between $\hat{g}_{n,m}(x)$ and $f(x)$. If the error is no more than the error bound e_b, we obtain the upper bound dps_{up}^{new} as the current dps (Line 8) and break the search loop (Line 9).

Example 2 *Consider a synthesis problem with the target function as $x^{0.45}$ and the error bound as $e_b = 0.05||x^{0.45}||_2$. We can obtain its $n_{min} = 2$. To search the upper bound dps_{up}^{new} on the smallest DPS, we start from the lower bound $n_{min} = 2$. Only when the DPS reaches 5, we can find a degree-precision pair $(n, m) = (2, 3)$ such that the error between the initial SCP $\hat{g}_{n,m}(x)$ obtained by the method described in Section II-B1 and $f(x)$ is smaller than the error bound e_b. Thus, we obtain $dps_{up}^{new} = 5$.*

In searching the upper bound dps_{up}^{new} on the smallest DPS, for each degree-precision pair, we only consider one ASCP, i.e., the one obtained by the method described in Section II-B1. Since we do not search extensively, the actual smallest DPS may be smaller than the upper bound dps_{up}^{new}. Fortunately, the actual smallest DPS is close to dps_{up}^{new}. Thus, we further perform a more exhaustive search for the smallest DPS starting from dps_{up}^{new} downwards. Lines 10–13 shows the loop for that search. When the loop terminates, the smallest DPS and a set of ASCPs with that DPS are obtained.

In the loop, Line 11 keeps the current upper bound dps_{up}^{new} in the variable dps_{up}^{old}. Note that the current upper bound is also the *known* smallest DPS. Thus, we will refer to the current upper bound as the known smallest DPS later. Line 12 calls the function *peturbSearch* to update the known smallest DPS dps_{up}^{new} and also obtain the ASCP set S_{SCP} with the DPS dps_{up}^{old} satisfying the error bound. If $dps_{up}^{new} < dps_{up}^{old}$, the known smallest DPS reduces and the loop will continue (Line 13). Otherwise, it means that the known smallest DPS cannot be improved. Consequently, the known smallest DPS dps_{up}^{old} is indeed the smallest DPS. Thus, the loop terminates and Line 14 returns the ASCP set S_{SCP} with the smallest DPS dps_{up}^{old}.

Algorithm 3: The procedure *perturbSearch*.

Input : the target function $f(x)$, an error bound e_b, the MDV V_l, the smallest degree n_{min}, and the known smallest DPS dps_{up}^{old}

Output: the updated known smallest DPS dps_{up}^{new} and the ASCP set S_{SCP} with DPS dps_{up}^{old}.

1 $S_{SCP} \leftarrow \emptyset$;
2 Generate a set P of degree-precision pairs (n, m) with $n + m = dps_{up}^{old}$ and $n \geq n_{min}$;
3 **foreach** $(n, m) \in P$ **do**
4 Obtain the initial SCP $\hat{g}_{n,m}(x)$ with degree n and precision m for the target function $f(x)$ by the method in Section II-B1;
5 Given $\hat{g}_{n,m}(x)$ and V_l, get the candidate ASCP set S_{cd};
6 **foreach** *ASCP* $g_{n,m}(x) \in S_{cd}$ **do**
7 **if** $error(g_{n,m}(x), f(x)) \leq e_b$ **then**
8 $g_{n,m'}(x) \leftarrow reducePrecision(g_{n,m}(x))$;
9 $g_{n',m'}(x) \leftarrow reduceDegree(g_{n,m'}(x))$;
10 $dps_{up}^{new} \leftarrow n' + m'$;
11 **if** $dps_{up}^{new} < dps_{up}^{old}$ **then**
12 **return** $\{dps_{up}^{new}, S_{SCP}\}$;
13 **else** $S_{SCP} \leftarrow S_{SCP} \cup g_{n,m}(x)$;
14 **return** $\{dps_{up}^{old}, S_{SCP}\}$;

The procedure of the function *peturbSearch* is shown in Algorithm 3. Its inputs are the target function $f(x)$, an error bound e_b, the MDV V_l, the smallest degree n_{min}, and the known smallest DPS dps_{up}^{old}. The outputs are the updated known smallest DPS dps_{up}^{new} and the ASCP set S_{SCP} with the DPS dps_{up}^{old}. Line 1 initializes the set S_{SCP} as an empty set. Line 2 generates a set P of degree-precision pairs (n, m) satisfying that $n+m = dps_{up}^{old}$ and $n \geq n_{min}$. For each degree-precision pair (n, m), Line 4 applies the method described in Section II-B1 to obtain the initial SCP $\hat{g}_{n,m}(x)$ with degree n and precision m for the target function $f(x)$. Next, given $\hat{g}_{n,m}(x)$ with FV $\hat{G} = (\hat{G}(0), \ldots, \hat{G}(n))$ and the MDV $V_l = (V_l(0), \ldots)$, Line 5 constructs the set of candidate ASCPs S_{cd} so that their corresponding FVs $(G(0), \ldots, G(n))$ satisfy the following conditions for all $0 \leq i \leq n$: 1) $|G(i) - \hat{G}(i)| \leq V_l(dps_{up}^{old})$; 2) $0 \leq G(i) \leq \binom{n}{i} \times 2^m$. Then, for each ASCP $g_{n,m}(x)$ in the set S_{cd}, if the error between $g_{n,m}(x)$ and the target function $f(x)$ is no more than the error bound e_b, we will check whether there exists an ASCP with smaller DPS and equivalent to $g_{n,m}(x)$ (Lines 8–9).

- Line 8 calls the function *reducePrecision* to obtain the new ASCP $g_{n,m'}(x)$ with the reduced precision m'. For the ASCP $g_{n,m}(x)$ with the FV $G = (G(0), \ldots, G(n))$, it finds the largest integer $0 \leq t \leq m$ so that each entry in G can be divided by 2^t. Then, it obtains the new equivalent ASCP $g_{n,m'}(x)$ with the FV $G' = (\frac{G(0)}{2^t}, \ldots, \frac{G(n)}{2^t})$ and the reduced precision $m' = m - t$.
- Line 9 calls the function *reduceDegree* to obtain the equivalent ASCP $g_{n',m'}(x)$ with the reduced degree n'. According to [22], given an SCP $g_{n+1,m}(x)$ of degree $n + 1$ with the FV $G = (G(0), \ldots, G(n + 1))$ and another SCP $g_{n,m}(x)$ of degree n with the FV $G' = (G'(0), \ldots, G'(n))$, they are equivalent if and only if Eq. (5) is satisfied:

$$G(i) = \begin{cases} G'(0), & \text{for } i = 0 \\ G'(i-1) + G'(i), & \text{for } 1 \leq i \leq n \\ G'(n), & \text{for } i = n+1. \end{cases} \quad (5)$$

  In the function *reduceDegree*, given the ASCP $g_{n,m'}(x)$ of the original degree n, we first check whether there exists an ASCP $g_{n-1,m'}(x)$ of degree $n-1$ so that the FVs of $g_{n,m'}(x)$ and $g_{n-1,m'}(x)$ satisfy Eq. (5). If it does, we continue this process until Eq. (5) cannot be satisfied for the FVs of an ASCP $g_{n-i,m'}(x)$ of degree $(n - i)$ and an ASCP $g_{n-i-1,m'}(x)$ of degree $(n - i - 1)$. Then, $g_{n-i,m'}(x)$ is assigned to $g_{n',m'}(x)$, and the reduced degree $n' = n - i$.

After obtaining the equivalent ASCP $g_{n',m'}(x)$, Line 10 updates the known smallest DPS dps_{up}^{new} as $n' + m'$. If $dps_{up}^{new} < dps_{up}^{old}$, it means the known smallest DPS can be reduced and Line 12 returns the updated known smallest DPS dps_{up}^{new} and the ASCP set S_{SCP} with DPS dps_{up}^{old} that has been found so far. Otherwise, Line 13 adds the SCP $g_{n,m}(x)$ with DPS dps_{up}^{old} to the set S_{SCP}. If for all ASCPs satisfying the error bound, we cannot reduce their DPSs, then Line 14 returns the dps_{up}^{old} as the updated known smallest DPS and the ASCP set S_{SCP} with DPS dps_{up}^{old}.

Example 3 *For the previous case in Example 2, the MDV is set as $V_l = [0, 0, 2, 2, 2, 3, 3]$. We search its ASCPs with the smallest DPS from the known smallest DPS $dps_{up}^{old} = dps_{up}^{new} = 5$. During the search, for the degree-precision pair $(2, 3)$, one candidate ASCP is $g_{2,3}(x)$ with the FV $G = (2, 12, 8)$. Then, we further check whether there exists an equivalent ASCP with smaller DPS for $g_{2,3}(x)$. To reduce precision, we find the largest integer $0 \leq t \leq 3$ so that each entry in the FV G can be divided by 2^t. Here, $t = 1$ satisfies the condition. Thus, the reduced precision $m' = m - 1 = 2$, and the equivalent ASCP is $g_{2,2}(x)$ with its FV $G' = (1, 6, 4)$. To reduce degree, we check whether there exists an ASCP $g_{1,2}(x)$ so that the FVs of $g_{1,2}(x)$ and $g_{2,2}(x)$ satisfy Eq. (5). However, no such ASCP exists. Thus, the final equivalent ASCP for $g_{2,3}(x)$ is $g_{2,2}(x)$, and $dps_{up}^{new} = n' + m' = 4$. Since $dps_{up}^{new} < dps_{up}^{old}$, we set*

$dps_{up}^{old} = dps_{up}^{new} = 4$, and perform the above process again. In the next iteration, we find $dps_{up}^{new} = dps_{up}^{old} = 4$, and there are 3 ASCPs with DPS as dps_{up}^{old} satisfying the error bound, i.e., $S_{SCP} = \{g_{2,2}(x)$ with the FV $(1, 6, 4)$, $g_{4,0}(x)$ with the FV $(0, 3, 3, 4, 1)$, $g_{4,0}(x)$ with the FV $(0, 4, 2, 4, 1)\}$. Thus, the loop terminates, and S_{SCP} is the ASCP set with smallest DPS as $dps_{up}^{old} = 4$.

C. SMT-based Exact SC Synthesis

For each ASCP in the ASCP set S_{SCP}, an SMT-based exact SC synthesis method is proposed to obtain an area-optimal SC circuit. It corresponds to Lines 4 and 5 in Algorithm 1. To eliminate the gap between SC logic synthesis and technology mapping, gate area is considered during exact synthesis. Here, we assume that the gate areas are integers. Due to the area constraint, there exist linear arithmetic constraints. Therefore, SMT solver is used to solve the problem.

First, given an ASCP g, the function *gateOptimalES* is called to obtain a gate-number-optimal SC circuit. The whole procedure is shown in Algorithm 4. In Line 1, the gate number r of the circuit is set as zero, and an SMT solver is initialized. In the main loop, we formulate a problem whether there exists an SC circuit with r gates realizing the given ASCP g. It is realized by resetting the solver and adding encoding variables, connection constraints, fanin constraints, ASCP constraints, and gate index constraints to the solver (Lines 3–5). If the solver returns UNSAT, it means that such a circuit does not exist, so Line 6 increases the gate number r by 1. Otherwise, a gate-number-optimal SC circuit C_{gate} is found (Line 8). Then, Line 9 returns the circuit.

Algorithm 4: The procedure *gateOptimalES*.

```
   Input  : an ASCP g.
   Output: a gate-number-optimal SC circuit C_gate.
1  r ← 0; s ← getSMTSolver();
2  while true do
3      resetSMTSolver(); addVars(r);
4      addConnectionCons(r); addFaninCons(r);
5      addASCPCons(g, r); addGateIndexCons(r);
6      if solve(s)=UNSAT then r ← r + 1;
7      else
8          C_gate ← s.model();
9          return the gate-number-optimal SC circuit C_gate;
```

After obtaining a gate-number-optimal SC circuit, the function *areaOptimalES* is called to obtain an area-optimal SC circuit. The main procedure is shown in Algorithm 5. Line 1 initializes the area-optimal SC circuit C_{area} as the gate-number-optimal SC circuit C_{gate} obtained from Algorithm 4. It also initializes an SMT solver. Note that the area of C_{gate} is a known smallest area. Line 2 sets the next area to be checked (i.e., A). Under the assumption that the gate areas are integers, this value is the area of C_{gate} minus 1. Besides, the gate number r_{min} is set as the gate number r of C_{gate}. During the whole process, the gate number of the circuit is fixed as r_{min}. In the main loop, we formulate a problem whether there exists an SC circuit with r_{min} gates and area no more than A realizing the given ASCP g. It is realized by adding encoding variables, the above four types of constraints, and an additional area constraint to the solver (Lines 4–7). If the solver returns SAT, such a circuit exists. Then, Line 9 obtains the SC circuit C_{area} with the area no more than A, and the area of C_{area} minus 1 will be assigned to A, which is the next area to be checked. Then, the loop continues. Otherwise, it means that the area of the SC circuit C_{area} obtained in the last iteration is indeed the smallest. Then, Line 10 returns the circuit.

We note that Algorithm 5 works under the assumption that the gate areas are integers. If they are not, we can convert them to integers through rounding. Alternatively, as SMT solvers can also handle problems with real coefficients, we can change the search framework to binary search to find the area-optimal SC circuit.

Algorithm 5: The procedure *areaOptimalES*.

```
   Input  : an ASCP g, a gate-number-optimal SC circuit C_gate.
   Output: an area-optimal SC circuit C_area.
1  C_area ← C_gate; s ← getSMTSolver();
2  A ← C_gate.A − 1; r_min ← C_gate.r;
3  while true do
4      resetSMTSolver(); addVars(r_min);
5      addConnectionCons(r_min); addFaninCons(r_min);
6      addASCPCons(g, r_min); addGateIndexCons(r_min);
7      addAreaCons(A);
8      if solve(s)=SAT then
9          C_area ← s.model(); A ← C_area.A − 1;
10     else return the area-optimal circuit C_area ;
```

Next, we will describe the details of encoding variables and constraints. Our method uses multiple selection variable (MSV) encoding [17]. The operator size is set as the maximum fanin number (i.e., K) of the gates in a library. For a gate with the fanin number less than K, we propose a method to extend them to K-input gates. Thus, our method can be applied to a realistic gate library whose gates have different fanin numbers. In the following illustration, we assume that $K = 4$.

1) Encoding Variables: Assume the number of gates in the SC circuit is r. The encoding variables are defined as follows:

- x_{it}: the t-th bit in the global truth table of node x_i, where $1 \leq i \leq n+m+r$ and $0 \leq t < 2^{n+m}$. As our circuit has $n+m$ PIs, we let $x_1, \ldots, x_{n+m}$ be the PIs and $x_{n+m+1}, \ldots, x_{n+m+r}$ be the gates. As we only consider a single-output function, the final output is mapped to the last gate x_{n+m+r}.
- s_{ij}: whether gate x_i has input from gate x_j, where $n+m < i \leq n+m+r$ and $1 \leq j < i$.
- f_{ipquv}: the output of gate x_i with its local input combination (p, q, u, v), where $n+m < i \leq n+m+r$ and $p, q, u, v \in \{0, 1\}$.
- v_{il}: whether gate x_i is the l-th gate in the given gate library, where $n+m < i \leq n+m+r$ and $1 \leq l \leq L$. Here, L is the total number of gates in the given library.

2) Connection Constraint: According to [17], we formulate constraints to ensure that the circuit encoded by the variables is legal and gives the correct output. The constraint is shown below:

$$\begin{aligned}&\bar{s}_{ij} \vee \bar{s}_{ik} \vee \bar{s}_{il} \vee \bar{s}_{ip} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c)\\ &\vee (x_{lt} \oplus d) \vee (x_{pt} \oplus e) \vee (f_{ibcde} \oplus \bar{a}) = 1.\end{aligned} \tag{6}$$

It must hold for all $n+m < i \leq n+m+r$, $1 \leq j < k < l < p < i$, and $a, b, c, d, e \in \{0, 1\}$. It means that if the inputs of gate x_i are x_j, x_k, x_l, and x_p, and the t-th bits of the global truth tables of x_i, x_j, x_k, x_l, and x_p are a, b, c, d, e, respectively, then the output of gate x_i with local input combination (b, c, d, e) must be a.

3) Fanin Constraint: As each gate has exactly four inputs, Eq. (7), a cardinality constraint, must be satisfied for all $n+m < i \leq n+m+r$:

$$\sum_{j=1}^{i-1} s_{ij} = 4. \tag{7}$$

4) ASCP Constraint: Given an ASCP with degree n, precision m, and FV $G = (G(0), \ldots, G(n))$, the output function of an SC

circuit, which is the output of gate x_{n+m+r}, must satisfy the FV. Thus, Eq. (8) must hold for all $0 \leq i \leq n$:

$$G(i) = \sum_{t \in S_i} x_{(n+m+r)t}, \tag{8}$$

where S_i represents the set of indexes of input combinations $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ satisfying that $\sum_{j=1}^{n} X_j = i$. Note that the above constraints are cardinality constraints. They are converted to two sets of CNFs according to the method in [17].

Example 4 *Given an ASCP with $n = 2$, $m = 1$, and the FV as $(2, 4, 1)$, assume there are $r = 2$ gates in the SC circuit. Table II shows the global truth table of gate x_5, which gives the output function of the circuit. Clearly, the sets S_0, S_1, and S_2 are $\{0, 1\}$, $\{2, 3, 4, 5\}$, and $\{6, 7\}$, respectively. Thus, to make sure that the output function satisfies the FV, the following constraints must be satisfied: $G(0) = x_{50} + x_{51} = 2, G(1) = \sum_{t=2}^{5} x_{5t} = 4, G(2) = x_{56} + x_{57} = 1$.*

Table II. The global truth table of the output gate x_5 in Example 4.

$Y_1 \backslash X_1X_2$	00	01	10	11
0	x_{50}	x_{52}	x_{54}	x_{56}
1	x_{51}	x_{53}	x_{55}	x_{57}

5) Gate Index Constraint: As we want to reduce the gap between logic synthesis and technology mapping, we need to take a realistic gate library into account, which has gates of different fanin numbers. However, the above encoding requires all the gates to have 4 inputs (see the variable f_{ipquv}). A novel method is proposed to address this inconsistency. Note that the method can be applied to any gate library. The MCNC standard cell library [23] is used as an example to illustrate the method.

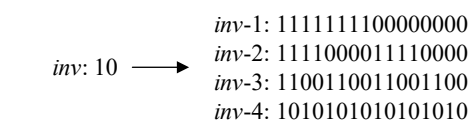

Fig. 3. Extend 1-fanin inverter *inv* to 4-fanin inverters.

The key idea is that for a gate with the fanin number less than 4, it is extended to a 4-input gate with some fake fanins. For example, we extend the inverter *inv* to a 4-input gate by introducing 3 additional fake fanins. Since we need to select one input from the 4 inputs as the actual fanin, there are $\binom{4}{1} = 4$ kinds of *inv* gates with 4 fanins in total. Fig. 3 shows the truth tables represented as row vectors of *inv* gates with 1 fanin and 4 fanins, respectively, where *inv-i* $(1 \leq i \leq 4)$ represents the 4-input inverter with the i-th fanin as the real one. Similarly, the other gates with fanin numbers less than 4 are extended to 4-input gates. Finally, we obtain $L = 86$ gates in the library.

As each gate x_i is mapped to exactly one gate in the given library, Eq. (9) must be satisfied for $n + m < i \leq n + m + r$:

$$\sum_{l=1}^{L} v_{il} = 1. \tag{9}$$

Furthermore, for each gate x_i, we need to set a constraint on the local truth table variables f_{ipquv} and the gate index variables v_{il} $(1 \leq l \leq L)$. For example, for the *inv*-1 gate shown in Fig. 3, assume its index is $l = 1$ in the gate library. Eq. (10) shows the constraint on the variables f_{ipquv} and v_{i1}:

$$v_{i1} = 1 \Longleftrightarrow f_{ipquv} = \begin{cases} 1, & \text{for } pquv \in \{0000, \ldots, 0111\} \\ 0, & \text{for } pquv \in \{1000, \ldots, 1111\} \end{cases}. \tag{10}$$

It means that if gate x_i is mapped to *inv*-1, then the local truth table of x_i must equal the truth table of *inv*-1. Similar constraints as above should be set on variables f_{ipquv} for gate x_i and the other v_{il}'s.

6) Area Constraint: Let the area of the l-th $(1 \leq l \leq L)$ gate in the library be a_l. In each iteration of the function *areaOptimalES*, as we look for an SC circuit with area no more than A, the following constraint should be satisfied, which is a linear arithmetic constraint:

$$\sum_{i=n+m+1}^{n+m+r} \sum_{l=1}^{L} a_l v_{il} \leq A. \tag{11}$$

IV. Speed-Up Techniques

In this section, we present some speed-up techniques for the solving process, including filtering unpromising ASCPs and dealing with the complex ASCP constraints.

A. Filtering Unpromising ASCPs

For each ASCP in the ASCP set, we conduct SMT-based exact synthesis to find an area-optimal circuit. If many ASCPs exist, it will take much time to handle all of them. In fact, some unpromising ASCPs can be filtered out to reduce the runtime. We propose two ways to filter out unpromising ASCPs.

1) Applying the gate-number-optimal exact synthesis: For each ASCP in the ASCP set, we conduct gate-number-optimal exact synthesis to obtain the minimal gate number for the ASCP. Then, we only keep the ASCPs with the smallest gate number and filter out the other ASCPs.

2) Applying the heuristic method: For each ASCP in the ASCP set, the state-of-the-art heuristic SC synthesis method [13] is applied to obtain an SC circuit for the ASCP, whose area can be regarded as the upper bound on the circuit area corresponding to this ASCP. Among these ASCPs with different areas, the smallest area is denoted as A_{min}. Then we only keep the ASCPs whose circuit area is less than or equal to $A_{min} + w$, where w is a non-negative integer.

B. Dealing with Complex ASCP Constraints

Consider an ASCP with degree n, precision m, and FV $G = (G(0), \ldots, G(n))$. To synthesize an SC circuit realizing the ASCP, we need to assign $G(i)$ 1s to the cells satisfying that $\sum_{j=1}^{n} X_j = i$ in the truth table of the form shown in Table I. Since the number of such cells is $G_m(i) = \binom{n}{i} \times 2^m$ in total, there are $\binom{G_m(i)}{G(i)}$ different assignments in total. However, as n and m increase, $\binom{G_m(i)}{G(i)}$ will increase exponentially. We propose two novel techniques to reduce the solution space.

1) Partial One Assignment (POA): The basic idea of this method is to partially assign some 1s in the truth table to reduce each $G(i)$ in the FV. Thus, the search space becomes smaller. Note that the assignment of the 1s should favor the synthesis of a small SC circuit. For this purpose, we borrow the idea from work [13]. It constructs a solution tree and iteratively assigns 1s in the form of the largest cubes among all valid cubes until the FV becomes a zero vector. In our method, we assign 1s based on the first several cubes decided by the method in [13]. To reduce the solution space with minimal influence to the optimality, usually, we only choose the first one or two cubes.

2) Multi-granularity Search (MGS): The basic idea of this method is that sometimes when we assign 1s, we assign multiple adjacent 1s in the truth table together. For example, if we want to assign two 1s to the first column of the truth table shown in Fig. 4 (i.e., the set of minterms satisfying that $X_1 + X_2 + X_3 = 0$), we choose either the cells 00000 and 00001 together or the cells 00010 and 00011

together. On the one hand, this method reduces the total number of choices. On the other hand, the final circuit is still close to optimal, as a small circuit usually has many adjacent 1s in its truth table. The number of 1s we assign together is called the *granularity*. In our implementation, we first decompose the FV into several sub-FVs, which is guided by the method in [13]. Then, for different sub-FVs, we assign 1s in the truth table with different granularities. We choose the granularity for a sub-FV as a power of 2 that divides all the entries of the sub-FV. The 1s belonging to different sub-FVs should occupy different cells in the truth table.

$Y_1Y_2\backslash X_1X_2X_3$	000	001	010	011	100	101	110	111	
00	x_0	x_4	x_8	x_{12}	x_{16}	x_{20}	x_{24}	x_{28}	y_{14}
01	x_1	x_5	x_9	x_{13}	x_{17}	x_{21}	x_{25}	x_{29}	
10	x_2	x_6	x_{10}	x_{14}	x_{18}	x_{22}	x_{26}	x_{30}	y_{15}
11	x_3	x_7	x_{11}	x_{15}	x_{19}	x_{23}	x_{27}	x_{31}	

y_0: x_0, x_1; y_1: x_2, x_3

Fig. 4. The truth table with different granularities for the output gate.

Example 5 *Consider an ASCP with* $n = 3, m = 2$*, and FV* $G = (3, 8, 9, 2)$*. Guided by the method in [13], the FV* G *is decomposed into two sub-FVs,* $G_1 = [2, 6, 6, 2]$ *and* $G_2 = [1, 2, 3, 0]$*. The granularity of* G_1 *and* G_2 *are 2 and 1, respectively. To assign* G_1 *with granularity 2 in the truth table, as shown in Fig. 4, two consecutive* x_t*'s* $(0 \leq t < 32)$ *are combined to generate a new variable* y_v*, where* $0 \leq v < 16$*. Then, we assign 1s of the sub-FV* G_1 *with granularity 2 using variable* y_v*. The sub-FV* G_2 *is assigned with granularity 1 using variable* x_t*. To realize the MGS method, from the variable relationship shown in Fig. 4, Eq.* (12) *should be satisfied for all* $0 \leq v < 16$*:*

$$y_v = 1 \iff x_{2v} = 1, x_{2v+1} = 1. \tag{12}$$

Besides, we need to introduce additional constraints to satisfy the FV G *based on the sub-FVs with different granularities. We illustrate this using* $G(0)$ *as an example. For* $G(0) = G_1(0)+G_2(0) = 2+1 = 3$*, it should satisfy the following two constraints:*

$$y_0 + y_1 = 1,$$
$$(1 - y_0)(x_0 + x_1) + (1 - y_1)(x_2 + x_3) = 1.$$

The first constraint ensures that only one of y_0 *and* y_1 *is* 1*. Correspondingly, a group of two adjacent cells in the first column of the truth table in Fig. 4 is assigned with* 1*s. Thus,* $G_1(0) = 2$ *is satisfied. The second constraint ensures that in the remaining two cells in the first column, only one is assigned with a* 1*. Thus,* $G_2(0) = 1$ *is satisfied.*

V. Experimental Results

In this section, we show the experimental results of the proposed MinSC method. All the experiments are conducted on a desktop with 3.59GHz Intel CPU and 31.9GB memory. The 14 benchmark functions are listed in Table III, which are obtained from work [16], [24], [25]. The gate library is chosen as the MCNC library. In addition, we use the state-of-the-art SMT solver z3 [26] to solve the SMT-based exact synthesis problem.

The approximation error is measured as the L2-norm distance between an ASCP $g_{n,m}(x)$ and the target function $f(x)$, i.e., $||g_{n,m}(x)-f(x)||_2$. Moreover, the error bound e_b is set as $\gamma||f(x)||_2$, where error ratio γ $(0 < \gamma < 1)$ is used to adjust the error bound. In our experiments, we choose the error ratio γ as 2% and 5%. The MDV is set as $V_l = [0, 0, 2, 2, 2, 3, 3, 8]$, which is used to generate the candidate ASCPs. As MinSC calls SMT solving multiple times, to reduce the runtime, we set a timeout of 1 minute for SMT solving. For the 14th function in Table III with the error ratio 2%, MinSC cannot synthesize it in reasonable time due to the large scale of its SC circuit. Thus, we exclude this function for the error ratio 2%.

To speed up the solving process, for the error ratio 2%, we apply POA and MGS techniques to the 5th, 8th, 11th, 12th, and 13th functions in Table III. For the error ratio 5%, these two techniques are applied to the 13th and 14th functions.

Table III. The 14 arithmetic functions used in our experiments.

ID	function	ID	function	ID	function	ID	function	ID	function
1	$\sin(x)$	4	$\exp(-x)$	7	$\sin(\pi x)/\pi$	10	$\tanh(\pi x)$	13	$\tan(x/2)$
2	$\cos(x)$	5	$\log(x+1)$	8	$\exp(-2x)$	11	$x^{0.45}$	14	$x\ln(x/2)+1$
3	$\tanh(x)$	6	$x^{2.2}$	9	$1/(x+1)$	12	$\sqrt{x}$		

A. Comparison to the DA method

In this section, we compare MinSC with the state-of-the-art method DA [16]. Starting from an initial SCP $\hat{g}_{n_0,m_0}(x)$ with degree n_0 and precision m_0, DA simultaneously searches for a good ASCP and conducts SC synthesis. The initial error between $\hat{g}_{n_0,m_0}(x)$ and the target function $f(x)$ should be smaller than the error bound e_b. Thus, for the 13th function with the error ratio 2% in Table III, the initial degree-precision pair (n_0, m_0) for DA is set as $(6, 4)$. For the other functions, we set (n_0, m_0) as $(4, 4)$.

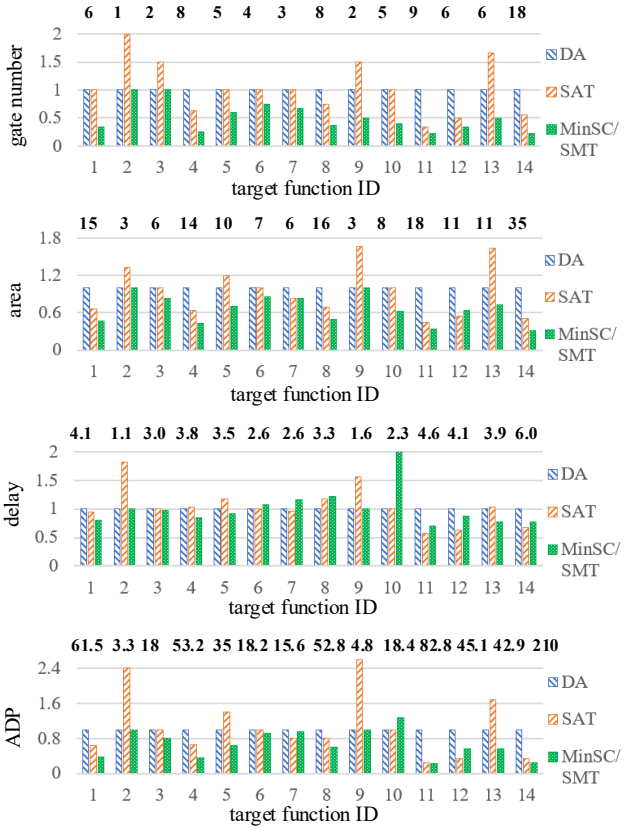

Fig. 5. The normalized gate number, area, delay, and ADP of the 14 arithmetic functions, which is normalized to the DA method. The results of the DA method are listed above each column.

Fig. 5 plots the gate number, area, critical path delay, and ADP of SC circuits obtained by MinSC with error ratio 5%, which is

Table IV. The comparison between MinSC and DA on the average results over the benchmarks.

error ratio	average DPS		average gate number		average area		average delay		average ADP		average MAE		average runtime(s)	
	DA	MinSC	DA	MinSC	DA	MinSC	DA	MinSC	DA	MinSC	DA	MinSC	DA	MinSC
2%	5.61	5.30 (5.48%)	5.54	2.62 (52.78%)	11.38	7.76 (31.76%)	3.61	3.46 (4.05%)	45.70	29.74 (34.91%)	0.0085	0.0087 (−2.35%)	0.125	270.883
5%	5.14	4.35 (15.28%)	5.93	2.36 (60.24%)	11.64	6.14 (47.24%)	3.32	3.08 (7.10%)	47.25	20.28 (57.07%)	0.0164	0.0126 (23.17%)	0.122	101.582

normalized to those of DA. For all the target functions, the gate number and the area of the SC circuits produced by MinSC are all better than or equal to those produced by DA. The gate number for 9 out of 14 functions is reduced by more than 50%, and the area for 7 out of 14 functions is reduced by more than 35%. This is because MinSC can find the ASCP with the smallest DPS and synthesize an area-optimal SC circuit for the ASCP, while DA cannot guarantee the optimality of ASCPs, and it uses a heuristic method to conduct SC synthesis. Usually, there exists a trade-off between area and delay. For most functions, the delay of the SC circuits produced by MinSC is no larger than that by DA, while for the others, the delay increases. Usually, the large reduction in area can compensate the increase in delay. For all the functions except the 10th one, the ADP of the SC circuits obtained by MinSC is smaller than that by DA.

Table IV shows the average hardware cost, the average MAE, and the average runtime for error ratios 2% and 5% together with the reduction ratio of MinSC over DA in parentheses. To calculate the average MAE, we select 9 points $x = 0.1, \ldots, 0.9$ for each function $f(x)$ in simulation, and the length of SC bit streams is set as a representative value of 1024. Note that the length affects the error due to stochastic variation [10]. However, it is not the focus of our work. Thus, we do not consider different bit stream lengths. From Table IV, the average DPS, gate number, area, delay, and ADP of MinSC are all better than those of DA. For error ratio 2% (resp. 5%), MinSC reduces the DPS, gate number, area, delay, and ADP over DA by 5.48% (resp. 15.28%), 52.78% (resp. 60.24%), 31.76% (resp. 47.24%), 4.05% (resp. 7.10%), and 34.91% (resp. 57.07%), respectively, without much MAE degradation. However, the runtime of MinSC is much longer than that of DA. The reason is that MinSC calls SMT solving multiple times to obtain the optimal result. Besides, the ASCPs of some functions have large DPS, which costs much time synthesizing the corresponding SC circuits. However, considering that the SC circuit synthesis is a one-time design effort and our proposed method produces a high-quality design, the longer runtime is acceptable. Furthermore, it is possible to apply some advanced techniques in [17] to speed up the process.

B. Comparison to the SAT-based Exact Synthesis Method

In this section, we compare the proposed SMT-based exact synthesis method with the state-of-the-art work [21]. It conducts SAT-based exact synthesis to obtain a gate-number-optimal SC circuit, where the SC circuits are represented as MIGs. We call it *SAT-based method*. The focus here is to compare the performance of the proposed SMT-based method and the SAT-based method to synthesize a given ASCP. Thus, we choose the ASCP with the smallest area obtained by MinSC as the input to both methods. For the SAT-based method, after it obtains the gate-number-optimal MIG networks, we use ABC [27] to conduct logic optimization and technology mapping. For the 11th and the 13th functions in Table III with error ratio 2%, the SAT-based method cannot synthesize them in reasonable time. Therefore, we exclude these two functions for error ratio 2%.

Fig. 5 plots the gate number, area, critical path delay, and ADP of SC circuits obtained by the SMT-based and the SAT-based methods with error ratio 5%, which is normalized to those of DA. Note that the results for the SMT-based method are just those for MinSC, as we choose the ASCP with the best area as the input to the SMT-based method. For almost all target functions, the gate number and the area of the SC circuits synthesized by the SMT-based method are better than or equal to those produced by the SAT-based method. For all the functions except the 7th, 10th, and 12th one, the ADP of the SC circuits obtained by the SMT-based method also outperforms that of the SAT-based method. This shows there exists a gap between logic synthesis and technology mapping, and our proposed SMT-based exact synthesis method can eliminate the gap.

Table V. The average hardware cost and runtime comparison between the SAT-based and the proposed SMT-based exact synthesis method.

error ratio	gate number		area		delay		ADP		runtime(s)	
	SAT	SMT	SAT	SMT	SAT	SMT	SAT	SMT	SAT	SMT
2%	5.6	2.5 (55.4%)	11.1	7.1 (36.1%)	3.6	3.1 (13.9%)	44.6	23.6 (47.1%)	60.8	38.14 (37.3%)
5%	4.9	2.4 (51.0%)	9.1	6.1 (33.0%)	3.14	3.08 (2.0%)	31.0	20.3 (34.5%)	349.5	17.85 (94.9%)

Table V lists the average hardware cost and the average runtime for error ratios 2% and 5% together with the reduction ratio of the SMT-based method over the SAT-based method in parentheses. Note that the runtime is only the synthesis time for the best ASCP with the smallest area. Clearly, the SMT-based method synthesizes a much smaller SC circuit than the SAT-based method in a shorter time.

Table VI. The runtime (s) for several functions with error ratio 5%.

founction ID	6	7	8	9	10	11	12	13	14
SAT	0.08	0.06	2.5	0.01	0.26	0.03	0.08	**4605.78**	**272.96**
SMT	3.65	0.50	52.91	0.11	0.26	0.31	0.30	**61.93**	**66.94**

Table VI shows the synthesis time for several target functions with error ratio 5%. Due to the page limit, we only list the last 9 functions. The SMT-based method not only conducts gate-number-optimal exact synthesis, but also conducts area-optimal exact synthesis. Thus, its runtime sometimes is longer than the SAT-based method. The major runtime improvement comes from the 13th and 14th functions, which have complex ASCP constraints with large DPS. For these two functions, we apply POA and MGS techniques to reduce the solution space, which results in a large runtime reduction.

VI. Conclusion

In this work, given a target function and an error bound, we propose a novel search method to find the ASCP with the smallest DPS and an SMT-based exact synthesis method to obtain an area-optimal SC circuit realizing the ASCP. Compared with the start-of-the-art methods, the proposed method can synthesize SC circuits with much smaller area and ADP. We note that a limit of the proposed method is the scalability. For large-scale SC circuits, the search space will increase exponentially, which makes the solution infeasible. Our future work will explore how to apply decomposition to synthesize large-scale SC circuits.

REFERENCES

[1] A. Alaghi *et al.*, "The Promise and Challenge of Stochastic Computing," *IEEE TCAD*, vol. 37, no. 8, p. 1515—1531, 2018.

[2] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully Parallel Stochastic LDPC Decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, 2008.

[3] S. S. Tehrani *et al.*, "Majority-Based Tracking Forecast Memories for Stochastic LDPC Decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, 2010.

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

[5] P. Li *et al.*, "Computation on Stochastic Bit Streams: Digital Image Processing Case Studies," *IEEE TVLSI*, vol. 22, no. 3, pp. 449–462, 2014.

[6] A. Ardakani *et al.*, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *IEEE TVLSI*, vol. 25, no. 10, pp. 2688–2699, 2017.

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

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

[9] P. Li *et al.*, "The Synthesis of Complex Arithmetic Computation on Stochastic Bit Streams Using Sequential Logic," in *ICCAD*, 2012, pp. 480–487.

[10] W. Qian *et al.*, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE TC*, vol. 60, no. 1, p. 93—105, 2011.

[11] A.Alaghi and J. P. Hayes, "STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis," *IEEE TCAD*, vol. 34, no. 11, pp. 1770–1783, 2015.

[12] T. H. Chen and J. P. Hayes, "Equivalence among Stochastic Logic Circuits and its Application," in *DAC*, 2015, pp. 131:1–131:6.

[13] X. Peng and W. Qian, "Stochastic Circuit Synthesis by Cube Assignment," *IEEE TCAD*, vol. 37, no. 12, pp. 3109–3122, 2018.

[14] K. K. Parhi and Y. Liu, "Computing Arithmetic Functions Using Stochastic Logic by Series Expansion," *TETC*, vol. 7, no. 1, pp. 44–59, 2019.

[15] Z. Zhao and W. Qian, "A General Design of Stochastic Circuit and its Synthesis," in *DATE*, 2015, pp. 1467–1472.

[16] C. Wang, W. Xiao, J. P. Hayes, and W. Qian, "Exploring Target Function Approximation for Stochastic Circuit Minimization," in *ICCAD*, 2020, pp. 1–9.

[17] W. Haaswijk *et al.*, "SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism," *IEEE TCAD*, vol. 39, no. 4, pp. 871–884, 2020.

[18] M. Soeken *et al.*, "Practical Exact Synthesis," in *DATE*, 2018, pp. 309–314.

[19] ——, "Busy Man's Synthesis: Combinational Delay Optimization with SAT," in *DATE*, 2017, pp. 830–835.

[20] ——, "Exact Synthesis of Majority-Inverter Graphs and Its Applications," *IEEE TCAD*, vol. 36, no. 11, pp. 1842–1855, 2017.

[21] X. He and Z. Chu, "Stochastic Circuit Design Based on Exact Synthesis," in *CSTIC*, 2021, pp. 1–3.

[22] W. Qian, M. D. Rosenberg, and I. Rosenberg, "Uniform approximation and Bernstein polynomials with coefficients in the unit interval," *European Journal of Combinatorics*, vol. 32, no. 3, pp. 448–463, 2011.

[23] S. Yang, "Logic synthesis and optimization benchmarks," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[24] Z. Qin *et al.*, "A Universal Approximation Method and Optimized Hardware Architectures for Arithmetic Functions Based on Stochastic Computing," *IEEE Access*, vol. 8, pp. 46 229–46 241, 2020.

[25] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE TOC*, vol. 56, no. 6, pp. 826–838, 2007.

[26] L. D. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *ITACAS*, 2008, pp. 337–340.

[27] A. Mishchenko, "ABC logic synthesis package," 2012. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/abc.htm