Simultaneous Area and Latency Optimization for Stochastic Circuits by D Flip-flop Insertion

Zhijing Li, Zhao Chen, Yili Zhang, Zixin Huang, and Weikang Qian, *Member, IEEE*

***Abstract*—Stochastic computing (SC) is an unconventional computing technique using digital circuits. It performs arithmetic computation on stochastic bit streams, which encode real values through the ratios of ones in the streams. Despite its advantages such as simple arithmetic units and strong error tolerance, SC faces two big challenges: long computation latency and large hardware overhead to generate independent stochastic bit streams. A recent work proposes to insert D flip-flops (DFFs) into the stochastic circuit to reduce the overhead to generate stochastic bit streams. In this work, observing that DFFs can also be exploited to reduce circuit delay, we propose a novel method to insert DFFs into a stochastic circuit to simultaneously reduce the computation latency of the circuit and the overhead of generating stochastic bit streams, thus addressing both challenges at the same time. Experimental results showed that compared to the state-of-the-art method in optimizing stochastic circuits with DFF insertion, our method can reduce the computation latency by 14.3% and the number of DFFs by 48.1%.**

***Index Terms*—stochastic computing; stochastic number generator; decorrelation; D flip-flop insertion;**

I. Introduction

Stochastic computing (SC), first introduced by Gaines back to 1960s [1], attracts more attention recently due to its appealing properties including small arithmetic units [2] and strong tolerance to bit flip errors [3]. It has been applied to various application domains including digital filter design [4], [5], image processing [6], [7], artificial neural networks [8], [9], and decoding of modern error-correcting codes [10], [11].

SC operates on stochastic bit streams (SBSs). One encoding format of SBSs is the *unipolar encoding*, where an SBS encodes a real value through the ratio of 1s in the stream [12]. For example, the stream x_1 in Fig. 1(a) represents $1/2$. The unipolar encoding can only represent non-negative values in the interval $[0, 1]$. Another encoding format is the bipolar encoding, where an SBS with ratio of 1s as r encodes a real value $(2r - 1)$. With the bipolar encoding, any value in the interval $[-1, 1]$ can be encoded. The techniques proposed in this paper work for both encodings. For simplicity, in the rest of this paper, we discuss based on the unipolar encoding.

The stochastic encoding allows SC to use very simple digital circuitry to realize arithmetic computation. For example, Fig. 1(a) shows that the multiplication of two real numbers in the range $[0, 1]$ can be realized by an AND gate. In fact, when a bit stream is long enough, the value encoded by the stream approaches the probability of a bit in that stream being 1. Thus, the values encoded by the input SBS x_1, input SBS x_2, and output SBS y are equal to $P(x_1 = 1)$, $P(x_2 = 1)$, and $P(y = 1)$, respectively, where $P(x = 1)$ denotes the probability of a bit in bit stream x being 1. For the AND gate, if its two input SBSs are independent, we clearly have $P(y = 1) = P(x_1 = 1)P(x_2 = 1)$, which means it realizes a multiplication. We refer to a circuit that takes SBSs as inputs and outputs as an **SC core**. SC core typically has small area, delay, and power consumption.

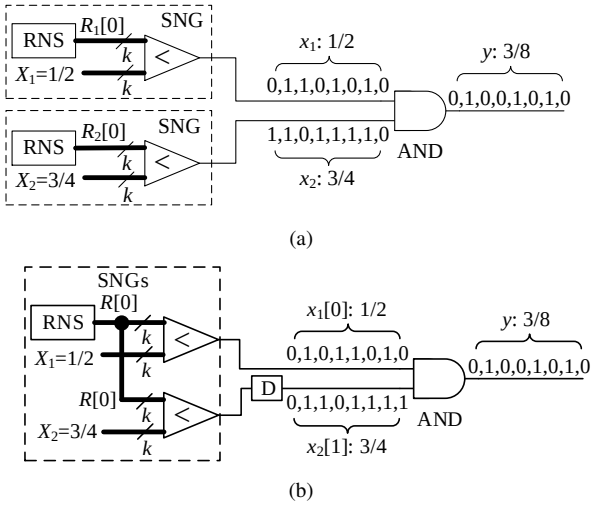


Fig. 1: Two versions of stochastic multiplier based on an AND gate: (a) using two separate SNGs to provide two independent input bit streams; (b) inserting a DFF to simplify the SNGs.

However, there are several challenges for SC. Among them, two most significant ones are large hardware cost to generate SBSs and the long computation latency.

To convert a binary number into an SBS, a stochastic number generator (SNG) is needed. However, it usually occupies a large portion of the area and power consumption of a SC circuit. One typical SNG is composed of a random number source (RNS) and a comparator (CMP) [13], as shown in Fig. 1(a). The RNS generates a uniformly distributed random binary number R at each clock cycle. A widely-used choice is a linear feedback shift register (LFSR) [14]. The CMP compares R with a constant binary number X and outputs a 1 if $R < X$, and a 0 otherwise. Therefore, the CMP produces an SBS of probability equal to the binary number X [1]. SC generally requires all the input SBSs to be independent. Therefore, the number of SNGs needed equals the number of inputs of the circuit. As a result, all the SNGs in a stochastic circuit take a much larger area and power consumption than the SC core, significantly reducing the area and power efficiency of the SC core [3].

Another challenge of SC is its long computation latency. By its unary encoding nature, to represent a value of precision c, the length of an SBS should be at least 2^c. For a SC circuit, since it produces one bit in the bit stream per clock cycle, the computation latency equals the circuit delay times the bit stream length, which is long for a high precision.

In this work, we propose a novel method based on D flip-flop (DFF) insertion to address these two challenges simul-

Zhijing Li, Zhao Chen, Yili Zhang, and Weikang Qian are with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China. Zixin Huang is with the College of Liberal Arts and Sciences at the University of Illinois at Urbana-Champaign, U.S.A.

Corresponding author: Weikang Qian; email: qianwk@sjtu.edu.cn

[1]Throughout this paper, we treat a binary number as a binary fraction.

taneously. Next, we first introduce the principle of inserting DFFs to reduce the cost of SNGs.

One way to reduce the cost of the SNGs is to share one RNS among all SNGs. However, simply doing this will produce SBSs that are maximally correlated, causing a wrong function. To decorrelate these input streams, DFFs can be inserted into the circuit to produce SBSs delayed by different numbers of clock cycles. Since an SBS generated by an SNG is typically a Bernoulli sequence, which has the property that different bits at different positions are mutually independent [1], therefore, at each clock cycle, the bits on different delayed streams are independent. Thus, the desired function can be realized. For example, Fig. 1(b) shows that although the two SNGs share one RNS, by inserting a DFF into the output of one CMP, the two SBSs are decorrelated and the AND gate realizes multiplication correctly.

Although the principle of utilizing DFFs to decorrelate input SBSs is well-known long time ago [1], it is until recently that a systematic study on how to optimally insert DFFs into a stochastic circuit was conducted [15]. In that work, observing that each DFF inserted still incurs some area overhead, the authors considered how to insert the fewest DFFs into the circuit to guarantee the correctness of the function.

Our work also considers inserting DFFs to reduce the cost of the SNGs. However, in our work, besides reducing the SNG cost, we also exploit DFF insertion to reduce the circuit delay and consequently, the computation latency. The basic idea is that by properly inserting DFFs into a circuit, it can potentially break a critical path and hence, reduce the circuit delay. Fig. 2(a) shows a stochastic circuit with the minimum number of DFFs inserted that realizes the function $X_1 + X_1^2 - X_1^3$ stochastically. In the figure, the square with a "D" inside denotes a DFF inserted on a wire. The input $x_1[0]$ is an SBS with value X_1, generated from an SNG, which we omit from the figure. $x_1[1]$ and $x_1[2]$ are the SBS $x_1[0]$ delayed by 1 and 2 clock cycles, respectively. By relocating the DFFs, Fig. 2(b) shows a stochastic circuit realizing the same function as that in Fig. 2(a) using the same number of DFFs. However, the delay of the stochastic circuit in Fig. 2(b) is reduced.

The circuit delay reduction will reduce the computation latency. Suppose the delays of the circuit before and after DFF relocation are d_1 and d_2, respectively, where $d_1 > d_2$. Suppose the bit stream length is N. The DFF relocation may increase the number of pipeline stages of a circuit. Assume the number of pipeline stages of the circuit after DFF relocation is P. For example, for the circuit shown in Fig. 2(b), we have $P = 2$. Typically, we have $P \ll N$. For the circuit after DFF relocation, the generation of the first bit in the bit stream takes P clock cycles, while the generation of each of the remaining $(N-1)$ bits takes 1 additional clock cycle. Thus, the total computation latency of the new circuit is $T_2 = (N-1+P)d_2$. Given that $P \ll N$, we have $T_2 \approx Nd_2$. For the original circuit, since it needs N clock cycles in total, the computation latency is $T_1 = Nd_1$. Thus, given that $d_2 < d_1$, we have $T_2 < T_1$. Furthermore, the reduction ratio of the computation latency roughly equals that of the circuit delay. Thus, the more we can reduce the circuit delay, the more we can reduce the total computation latency.

To summarize, in our work, for the first time, we propose a method to take advantage of DFF insertion to reduce both the area overhead of SNGs and the computation latency. We consider the problem of **relocating** the DFFs in an input stochastic circuit to obtain an equivalent circuit with the fewest number of inserted DFFs and the shortest circuit delay. The proposed approach is based on a novel technique to analyze the stochastic function of a circuit with DFFs inserted. With this technique, we formulate an integer linear programming (ILP) problem to determine the optimal number and positions of the DFFs in order to minimize the DFF number and the circuit delay simultaneously, while maintaining the function of the stochastic circuit. It should be noted that this proposed technique works for any SNG structure that is composed of an RNS and a CMP. There is no limitation to the RNS as long as it generates a uniformly distributed random binary number. Examples of RNS include the widely-used LFSR and some novel RNSs that generate the low-discrepancy sequences such as Halton [16] and Sobol sequences [17]. However, this proposed technique may not work for some other SNGs that do not use a CMP, such as the weighted binary generator-based SNG [18] and the SNG proposed in [19], since the approach relies on the assumption that each SBS is produced through a CMP. Also, it does not work for a deterministic bit stream generator [20], since such a generator does not produce Bernoulli bit sequences. Thus, in the rest of the paper, we assume that the SNG is composed of an RNS and a CMP.

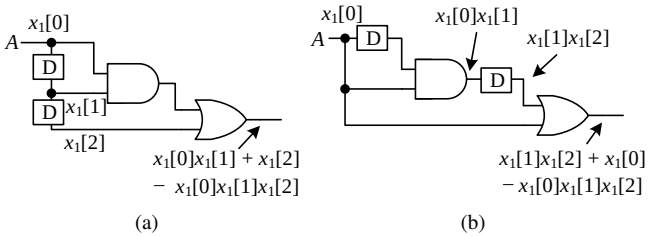


Fig. 2: Circuit delay reduction with the minimum DFF number: (a) original input stochastic circuit; (b) a stochastic circuit of the same function, but with a smaller delay.

Furthermore, another contribution of our work is that for the case where the RNS is an LFSR, we propose a way to further reduce the number of DFFs inserted. In the previous work [15], the authors only consider inserting DFFs **outside** the SNGs, i.e., after the CMPs. In this work, we first demonstrate a way to insert DFFs **inside** the SNGs so that it is equivalent to inserting DFFs after the CMPs. With this extension, our design allows DFFs to be inserted both inside and outside the SNGs. As a result, our method explores a large design space and hence, can further reduce the DFF number. Fig. 3 shows an example using a 3-input AND gate to realize the multiplication on three numbers. In Fig. 3(a), all the DFFs are inserted outside the SNGs. With this restriction, the minimum number of DFFs needed is 3. However, using our proposed method to insert DFFs inside the SNGs, we can reduce the number of inserted DFFs to 2, as shown in Fig. 3(b) (More details will be shown in Section III). Again, we formulate an ILP problem to determine the optimal number and positions of the DFFs, assuming that the DFFs can also be inserted inside the SNGs.

The rest of this paper is organized as follows. In Section II, we show the impact of DFF insertion and discuss a related work. In Section III, for the case where the RNS is an LFSR, we present a way to insert DFFs inside the SNGs so that it is equivalent to inserting DFFs outside. In Section IV, we present a procedure to obtain the stochastic function of a given stochastic circuit with DFFs inserted. Based on this procedure, in Section V, we show a procedure to derive a set of linear integer equality and inequality constraints that guarantees the functional equivalence between a new stochastic circuit with DFFs inserted and a given one. Based on the functional equivalence constraints obtained using our proposed procedure, in Section VI, we present an ILP formulation to minimize the number of DFFs in a stochastic circuit. Based on the method

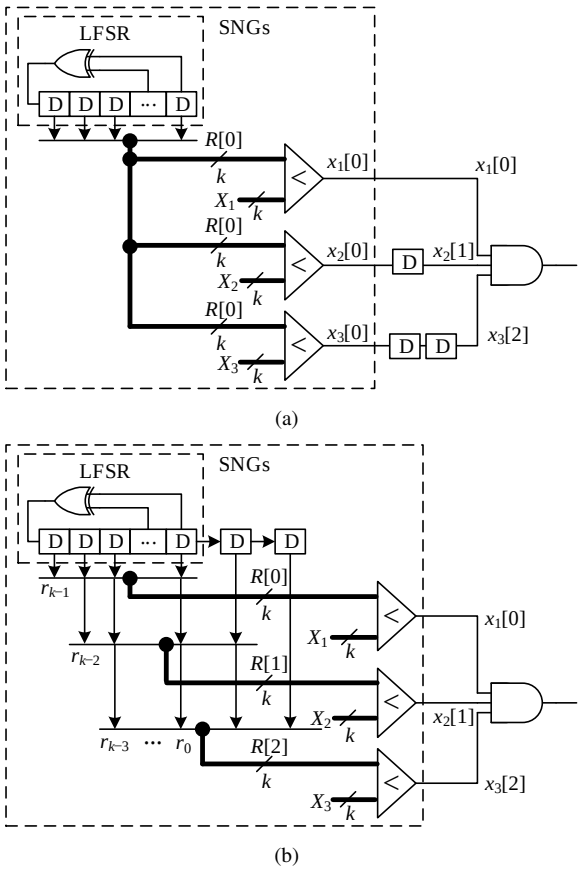


Fig. 3: DFF number reduction for a stochastic circuit: (a) a circuit with DFFs only inserted outside the SNGs; (b) a circuit with DFFs inserted inside the SNGs.

to minimize the DFF number, in Section VII, we present an algorithm to minimize the delay of a stochastic circuit. In Section VIII, we show the experimental results. Finally, in Section IX, we conclude this paper.

II. Preliminaries and Related Works

In this section, we first show the impact of DFF insertion on the power, area, and accuracy of a stochastic circuit. We then discuss a related work on DFF number minimization for stochastic circuits.

A. Impact of DFF Insertion

In this section, we present the basic experimental study on the impact of DFF insertion to stochastic circuits.

We first studied the impact of DFF insertion on the power and area of the entire stochastic circuit. We synthesized two versions of stochastic multiplier shown in Figs. 1(a) and 1(b), respectively, using Synopsys Design Compiler [21] with a NanGate 45nm cell library [22]. The RNS was chosen as a 8-bit LFSR. Since the delays of the two circuits are the same, we did not compare the delays. The power and area breakdown of different components in these two circuits is listed in Table I. From the table, we can see that SNGs consume a large portion of the power and area in a stochastic circuit. By simplifying the SNGs through DFF insertion, a stochastic multiplier can save 29.3% power and 28.5% area. Furthermore, the single DFF in the circuit shown in Fig. 1(b) consumes 4.3% of the total power and 3.9% of the total area of the circuit. Although the area and power consumption of a single DFF are small, they are non-negligible. For a circuit with many DFFs, it will be beneficial to further reduce the number of DFFs.

TABLE I: The power and area breakdown of the two versions of stochastic multiplier shown in Figs. 1(a) and 1(b).

Component		Separate SNGs		DFF insertion	
		Power (μW)	Area (μm^2)	Power (μW)	Area (μm^2)
SNGs	RNS(s)	37.4	107.5	20.7	53.7
	CMPs	12.8	56.4	13.0	56.4
SC core	DFF	0	0	1.9	4.8
	AND	0.6	1.1	0.5	1.1
Other (i.e., buffers)		11.9	6.3	8.2	6.4
Total		62.7	171.3	44.3	122.4

DFF insertion inevitably affects the computation accuracy of a stochastic circuit. Due to the existence of randomness in SC, the value encoded by the output SBS of a stochastic circuit is a random variable. The *accuracy* of the output value of a stochastic circuit can be characterized by the *root mean square error (RMSE)* of the output random variable. RMSE is also related to the *numerical precision*: the smaller the RMSE is, the higher the numerical precision.

The RMSE of the output random variable of a circuit with DFF insertion is generally different from that of a circuit using separate SNGs. The theoretical analysis of the RMSE of the output of a stochastic circuit with DFF insertion is challenging. Currently, it can only be done for some simple circuits such as a squarer [23]. The theoretical analysis for general circuits is an important problem deserving further study, but is out of the scope of this paper. Here, we empirically studied the RMSEs of circuits with DFF insertion and compared the results with the circuits using separate SNGs. For this purpose, we randomly generated 100 circuits with gate numbers in between 7 and 21 and numbers of inputs in between 4 and 6. For each circuit, we also randomly generated 10 sets of input probabilities. We considered two ways to provide input SBSs. The first one uses different SNGs for different inputs and the second one inserts DFFs to reduce the SNG cost. For each circuit and each set of input probabilities, we obtained the RMSEs for 8 stream lengths: $2^5, 2^6, \ldots, 2^{12}$. The average RMSE over all the circuits and all the sets of input probabilities for each stream length is shown in Fig. 4.

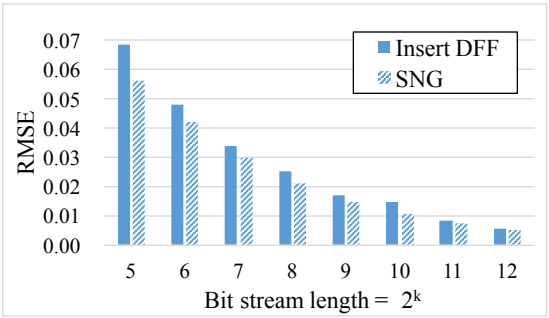


Fig. 4: Comparison of the RMSEs between circuits with DFF insertion and circuits using separate SNGs.

For the circuits with DFF insertion, their RMSEs decrease as the bit stream length increases. Although the RMSE of a circuit with DFF insertion is generally larger than that of the counterpart using separate SNGs, the difference between their RMSEs is quite small and the relative difference reduces with

the bit stream length. Therefore, considering its significant reduction in power and area of SNGs, DFF insertion should be preferable when a slight accuracy loss is tolerable.

B. DFF Number Minimization for Stochastic Computing

In the recent work [15], an algorithm called VAIL, which is based on integer linear programming (ILP), was proposed to minimize the number of DFFs in a stochastic circuit. The algorithm takes as inputs a target circuit without any DFFs inserted and a list of pairs of inputs that should be provided with independent SBSs. It outputs a stochastic circuit with all the specified input pairs decorrelated using the fewest DFFs.

As we pointed out in Section I, although the algorithm VAIL aims at minimizing the DFF number, it only adds DFFs outside the SNGs of a given circuit; it does not allow DFFs to be added inside the SNGs. Furthermore, it does not consider reducing the circuit delay through the DFF insertion.

III. DFF Insertion inside SNGs for LFSR-based RNS

In this section, for the case where the RNS is an LFSR, we present a way to insert DFFs inside the SNGs, while the circuit function is still kept. This technique allows further reduction of the inserted DFFs, which will be exploited in our proposed optimal DFF insertion algorithm.

The proposed way is shown in Fig. 3(b). Assume that the LFSR generates a k-bit random binary number $R[0] = (0.r_{k-1}[0]\dots r_0[0])_2$, where for all $i = 0, \dots, k-1$, $r_i[0] \in \{0, 1\}$, and the 0's in the brackets denote that the random bits are at the current clock cycle. Assume that we want to generate m additional random binary numbers. We insert m DFFs at the output of the least significant bit r_0 to record the values of r_0 $1, \dots, m$ clock cycles before. For any $1 \le i \le m$, we use $r_0[i]$ to denote the value of r_0 i clock cycles before.

There are two cases in terms of the values of m and k: $m < k$ and $m \ge k$. For the case where $m < k$, for each $1 \le i \le m$, we construct the i-th random binary number $R[i]$ by using $r_0[1], \dots, r_0[i]$ as the i least significant bits of $R[i]$ from higher to lower bit positions and $r_{k-i-1}[0], \dots, r_0[0]$ as the remaining most significant bits of $R[i]$. Mathematically, $R[i]$ is constructed as follows

$$R[i] = (0.r_{k-i-1}[0]r_{k-i-2}[0]\dots r_0[0]r_0[1]\dots r_0[i])_2 .$$

Fig. 3(b) shows an example for $m < k$ and $m = 2$.

For the case where $m \ge k$, for each $1 \le i < k$, we construct the i-th random binary number $R[i]$ by using $r_0[1], \dots, r_0[i]$ as the i least significant bits of $R[i]$ from higher to lower bit positions and $r_{k-i-1}[0], \dots, r_0[0]$ as the remaining most significant bits of $R[i]$. For each $k \le i \le m$, we construct the $R[i]$ by using $r_0[i-k+1], \dots, r_0[i]$ as all the bits of $R[i]$ from higher to lower bit positions. Mathematically, $R[i]$ is constructed as follows

$$R[i] = \begin{cases} (0.r_{k-i-1}[0]r_{k-i-2}[0]\dots r_0[0]r_0[1]\dots r_0[i])_2 , & 1 \le i < k, \\ (0.r_0[i-k+1]r_0[i-k+2]\dots r_0[i])_2 , & k \le i \le m. \end{cases}$$

By our proposed construction, the random binary number $R[i]$ is equivalent to the random binary number produced by the LFSR i clock cycles before. Therefore, the output SBS of a CMP that takes $R[i]$ and a constant X as inputs is identical to the i-cycle delayed output SBS of a CMP that takes $R[0]$ and X as inputs. For example, the bit stream $x_3[2]$ in Fig. 3(b) is identical to the bit stream $x_3[2]$ in Fig. 3(a). Therefore, inserting i DFFs inside the SNGs is equivalent to inserting i DFFs after the CMP. We use the notation $x_j[i]$ to denote an output SBS of a CMP that takes $R[i]$ and a binary constant number X_j as inputs. By our previous discussion, it is easily seen that $x_j[i]$ is equivalent to $x_j[0]$ delayed by i clock cycles. Note that the value encoded by the bit stream $x_j[i]$ is X_j.

The benefit of this design is that we can reduce the number of required DFFs. For example, if we want to generate $m+1$ uncorrelated input bit streams $x_1[0], x_2[1], \dots, x_{m+1}[m]$, where their encoded values $X_1, X_2, \dots, X_{m+1}$ are different, we only need to insert m DFFs inside the SNGs. On the other hand, if we are only allowed to insert DFFs after the CMPs, then we need to use $1 + 2 + \dots + m = m(m+2)/2$ DFFs. Fig. 3 shows an example for $m = 2$.

It should be noted that this proposed method cannot be applied when the RNS is not an LFSR. In this case, the random binary number $R[i]$ may not be equal to the random binary number produced by the RNS i clock cycles before. Therefore, the stream produced by comparing $R[i]$ with a constant X is not identical to the stream that is produced by first comparing $R[0]$ with a constant X and then delaying i clock cycles. In other words, the stream produced by inserting the DFFs inside the SNG is not identical to the one produced by inserting the DFFs outside the SNG.

IV. Obtaining Stochastic Function

In this section, we show a procedure to obtain the stochastic function of a given stochastic circuit with DFFs inserted. This technique will be used in Section V to establish the constraint for the functional equivalence of stochastic circuits.

The basic idea to obtain the stochastic function is to propagate the stochastic function of each gate in the circuit in a topological order from the inputs to the output. Our proposed method can be viewed as an extension to an algorithm for calculating the output probability of a general combinational circuit proposed by Parker and McCluskey [24].

In what follows, we will also interpret $x_j[i]$ as the probability that the random binary number $R[i]$ is smaller than the constant binary number X_j. The product $x_j[i]x_l[k]$ then is interpreted as the probability that the random binary numbers $R[i]$ and $R[k]$ are smaller than the constant binary numbers X_j and X_l, respectively.

Suppose that the circuit has n inputs and that the i-th ($1 \le i \le n$) input is an SBS $x_i[u_i]$, which is produced by a CMP taking $R[u_i]$ and X_i as inputs. Note that when the RNS is an LFSR, we could insert DFFs into the SNGs and hence, u_i could be any non-negative integer, which corresponds to the number of delayed clock cycles. However, when the RNS is not an LFSR, we cannot insert DFFs into the SNGs and hence, u_i must be 0.

For each signal S in the circuit, the probability that the signal is 1 is a function of the form $F_S(x_{S_1}[t_1], \dots, x_{S_{m_S}}[t_{m_S}])$, where m_S specifies the number of variables that F_S depends on, S_i is in the set $\{1, 2, \dots, n\}$, t_i is a non-negative integer, and $x_{S_i}[t_i]$ is the probability that the random binary number $R[t_i]$ is smaller than the constant binary number X_{S_i}. We call this function the **stochastic function** at the signal S.

Suppose that on a signal wire, we insert w DFFs and the stochastic function at the direct input of this sequence of w DFFs is $F_S(x_{S_1}[t_1], \dots, x_{S_{m_S}}[t_{m_S}])$. Then, the stochastic function at the direct output of this sequence of w DFFs is

$$F_S(x_{S_1}[t_1 + w], \dots, x_{S_{m_S}}[t_{m_S} + w]).$$

The stochastic function at the direct output of a gate is obtained by applying the **stochastic gate function** on the stochastic functions at the direct inputs of the gate. The

stochastic gate function of a gate represents the output probability of the gate in terms of its input probabilities. For example, the stochastic gate functions for an inverter, an AND gate, and an OR gate under the unipolar encoding[2] are

$$z = 1 - x, z = xy, z = x + y - xy,$$

respectively, where x and y are the input probabilities of the gate and z is the output probability. In our propagation procedure, for example, if the two stochastic functions at the direct inputs of an OR gate are F_S and F_T, then the stochastic function at the direct output of the OR gate is $F_R = F_S + F_T - F_S F_T$. Since the stochastic gate function of any gate only involves arithmetic addition and multiplication, the stochastic function at the output of any gate in the circuit is a polynomial on the variables $x_j[i]$'s.

After we obtain the polynomial for the final output of the stochastic circuit, we simplify it. Note that some product terms of the final polynomial may contain a variable $x_j[i]$ such that its degree is larger than 1.

Example 1

Suppose the last gate is an AND gate and its two input stochastic functions are $F_S = x_1[2]x_2[3]$ and $F_T = x_2[3]x_3[2]$. Then, the final polynomial is a product term $F_O = F_S F_T = x_1[2]x_2^2[3]x_3[2]$, where the degree of the variable $x_2[3]$ is 2. □

Mathematically, the term $x_j^k[i]$ where $k > 1$ is the probability that k identical random binary numbers $R[i]$ are all smaller than X_j, which is equal to the probability that a single random binary number $R[i]$ is smaller than X_j. Therefore, we have $x_j^k[i] = x_j[i]$. We first apply this simplification to all the product terms. Then, the degree of each variable $x_j[i]$ in each product term is just 1. For the case in Example 1, we have

$$F_O = x_1[2]x_2[3]x_3[2]. \tag{1}$$

Then, for each product term, we partition all the variables $x_j[i]$'s in that term into a number of sets according to the value of i. If a set contains a single variable $x_j[i]$, then this variable is replaced by a term X_j in the final product term. If a set contains more than one variable, we assume the set of variables is $\{x_{j_1}[i], x_{j_2}[i], \ldots, x_{j_k}[i]\}$. Then, this set of variables is replaced by a term $\min\{X_{j_1}, \ldots, X_{j_k}\}$ in the final product term. The reason is that by definition, the product $\prod_{l=1}^{k} x_{j_l}[i]$ is the probability that the random binary number $R[i]$ is smaller than the binary constant X_{j_l}, for all $l = 1, \ldots, k$. This probability is equal to the probability that $R[i]$ is smaller than the smallest one among $X_{j_1}, \ldots, X_{j_k}$. Therefore, we have

$$\prod_{l=1}^{k} x_{j_l}[i] = \min\{X_{j_1}, \ldots, X_{j_k}\}. \tag{2}$$

For the case in Example 1, by replacing $x_2[3]$ by X_2 and the set of variables $\{x_1[2], x_3[2]\}$ by $\min\{X_1, X_3\}$ in Eq. (1), we eventually obtain $F_O = \min\{X_1, X_3\} \cdot X_2$. It is not hard to see that one key factor enabling the simplification shown in Eq. (2) is our assumption that each bit stream is produced by a CMP. This explains why we state in Section I that our method is only applicable to SNGs composed of an RNS and a CMP.

[2]The stochastic gate function of a gate under the bipolar encoding is different from that under the unipolar encoding. However, expect the stochastic gate functions, other components of our proposed algorithms are the same for these two encodings.

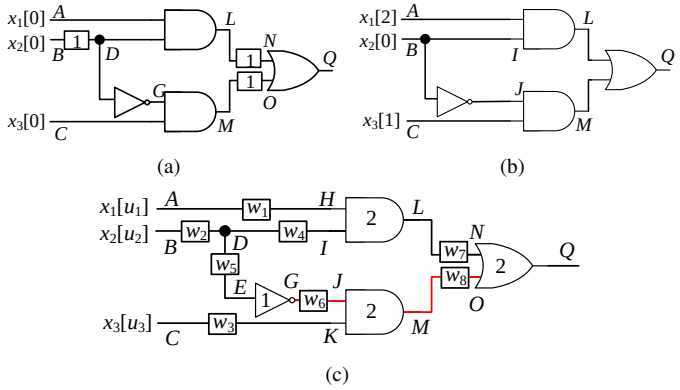


Fig. 5: Stochastic circuits with DFFs inserted: (a) a given circuit; (b) another circuit with the same stochastic function as that in (a); (c) a general circuit for the circuit in (a).

Example 2

We use the circuit shown in Fig. 5(a) to illustrate our procedure for obtaining the stochastic function of a stochastic circuit with DFFs inserted. In the figure, a square containing a number k on a wire indicates that there is a sequence of k DFFs inserted on the wire. We denote the stochastic function at a signal S as F_S. Through the stochastic function propagation, we can obtain

$$\begin{aligned}
&F_D = x_2[1], \quad F_G = 1 - F_D = 1 - x_2[1],\\
&F_L = x_1[0]F_D = x_1[0]x_2[1],\\
&F_M = x_3[0]F_G = x_3[0](1 - x_2[1]) = x_3[0] - x_2[1]x_3[0],\\
&F_N = F_L(x_1[0+1], x_2[1+1]) = x_1[1]x_2[2],\\
&F_O = F_M(x_2[1+1], x_3[0+1]) = x_3[1] - x_2[2]x_3[1].
\end{aligned}$$

Finally, we obtain

$$\begin{aligned}
F_Q = F_N + F_O - F_N F_O = x_1[1]x_2[2] + x_3[1] - x_2[2]x_3[1]\\
- x_1[1]x_2[2]x_3[1] + x_1[1]x_2^2[2]x_3[1]
\end{aligned}$$

By our simplification rules, we can eventually obtain that

$$\begin{aligned}
F_Q = X_1X_2 + X_3 - X_2X_3 - \min\{X_1, X_3\} \cdot X_2\\
+ \min\{X_1, X_3\} \cdot X_2 = X_1X_2 + X_3 - X_2X_3.
\end{aligned}$$

Notice that in the above simplification, we have applied the rule that $x_1[1]x_3[1] = \min\{X_1, X_3\}$. □

V. Establishing Constraints for Functional Equivalence

In our work, we transform a given stochastic circuit with DFFs inserted into an equivalent one through relocating DFFs so that the new one has the fewest DFFs inserted and the shortest circuit delay. One key constraint is the *functional equivalence*. It means the stochastic function computed by the circuit does not change after the modification. The stochastic function can be obtained by the method described in Section IV. In this section, we show a procedure to establish a set of linear equalities and inequalities that guarantees the functional equivalence. It involves the following 4 major steps:

1) Construct a general circuit for the given stochastic circuit.
2) Obtain the output stochastic function of the general circuit.
3) Put the actual values extracted from the given stochastic circuit into the output stochastic function of the general circuit and perform simplification.
4) Based on the simplification result, establish a set of linear equalities and inequalities that guarantees the functional equivalence.

We discuss these steps in details in the following subsections.

A. Constructing the General Circuit

To construct the **general circuit** for a given stochastic circuit, we first remove all existing DFFs in the given circuit and then insert w_i DFFs on each signal wire in the given circuit, where w_i's are unknown non-negative integers we need to solve. In the case where the RNS is an LFSR, we assume that the i-th input of the circuit to be $x_i[u_i]$, where u_i is also an unknown non-negative integers we need to solve. In the case where the RNS is not an LFSR, the i-th input is $x_i[0]$. For example, for the given circuit shown in Fig. 5(a), its general circuit is shown in Fig. 5(c), assuming that the RNS is an LFSR. In what follows, we only consider the situation where the RNS is an LFSR, since the situation where the RNS is not an LFSR can be easily treated as a special case of the situation where the RNS is an LFSR.

B. Obtaining the Stochastic Function for the General Circuit

Now, we apply the method described in Section IV to obtain the final output stochastic function of the general circuit. Note that since the number of DFFs on each signal wire is an unknown variable, the final output function is just a polynomial that cannot be simplified.

Example 3

For the general circuit shown in Fig. 5(c), we obtain the final output stochastic function as

$$F_Q = F_N + F_O - F_N F_O = P_1 + P_2 + P_3 + P_4 + P_5, \quad (3)$$

where

$$P_1 = x_1[u_1 + w_1 + w_7]x_2[u_2 + w_2 + w_4 + w_7], \quad (4)$$

$$P_2 = x_3[u_3 + w_3 + w_8], \quad (5)$$

$$P_3 = -x_2[u_2 + w_2 + w_5 + w_6 + w_8]x_3[u_3 + w_3 + w_8], \quad (6)$$

$$P_4 = -x_1[u_1 + w_1 + w_7]x_2[u_2 + w_2 + w_4 + w_7] \cdot x_3[u_3 + w_3 + w_8], \quad (7)$$

$$P_5 = x_1[u_1 + w_1 + w_7]x_2[u_2 + w_2 + w_4 + w_7] \cdot x_2[u_2 + w_2 + w_5 + w_6 + w_8]x_3[u_3 + w_3 + w_8]. \quad (8)$$

□

It can be seen that the final output stochastic function of the general circuit is a sum of N product terms $P_1, \ldots, P_N$. For any $1 \le k \le N$, we have

$$P_k = c_k \prod_{i \in S_k} \prod_{j=1}^{n_{k,i}} x_i[t_{k,i,j}],$$

where c_k is a constant coefficient, $S_k \subseteq \{1, 2, \ldots, n\}$ is the set of indices i's for all x_i's appearing in the product term P_k, $n_{k,i}$ is the number of x_i terms in the product P_k, and $t_{k,i,j}$ is the value inside the brackets of the j-th x_i term in the product P_k, which is a sum of one u_l and multiple w_r's.

Example 4

Consider the product term P_5 shown in Eq. (8), we have $k = 5$, and $c_k = c_5 = 1$. The set of indices i's for all x_i's appearing in the product P_5 is $S_5 = \{1, 2, 3\}$. For $i = 2$, the number of x_i terms in the product P_5 is $n_{5,2} = 2$. They are $x_2[u_2 + w_2 + w_4 + w_7]$ and $x_2[u_2 + w_2 + w_5 + w_6 + w_8]$. For the first x_2 term, the value inside the brackets is

$$t_{5,2,1} = u_2 + w_2 + w_4 + w_7;$$

for the second x_2 term, the value is

$$t_{5,2,2} = u_2 + w_2 + w_5 + w_6 + w_7. \quad \square$$

It should be noted that each $t_{k,i,j}$ that appears in the final output stochastic function corresponds to the sum of the DFF numbers along a path from *the output of the RNS* to the output of the general stochastic circuit. For example, the value of $t_{5,2,1}$ in Example 4 equals $u_2 + w_2 + w_4 + w_7$. It corresponds to the sum of the DFF numbers along the path from the output of the RNS to the final output Q via the points B, D, I, L, N.

C. Simplification Based on the Given Circuit

Now, we consider the given stochastic circuit. We put the actual values of u_i's and w_i's extracted from the given circuit into the output stochastic function of the general circuit and obtain an updated function. To indicate the difference, we add an apostrophe to any term that is updated with the actual value. Specifically, we use P'_k to denote the k-th product term in the updated function and $t'_{k,i,j}$ the updated value in the brackets.

Next, we perform simplification, which involves the following two steps.

1) Apply the simplification rules described in Section IV for each product term P'_k. Assume that P'_k is simplified to the product term Q'_k. Each Q'_k can be represented as $c_k \cdot H'_k$, where c_k is the coefficient of Q'_k and H'_k is a product of the components of the form $\min_{i \in S} X_i$. Notice that c_k is same as the coefficient of the original product term P_k. We call H'_k the **kernel** of Q'_k, or, alternatively, the **simplified kernel** of P'_k.
2) Merge all the product terms Q_k's in the sum by combining those Q'_k's with the same kernel. The result is a sum of M ($M \le N$) product terms $Z'_1, Z'_2, \ldots, Z'_M$. Each Z'_k can be represented as $d_k \cdot W'_k$, where d_k is the coefficient of Z'_k and W'_k is a product of the components of the form $\min_{i \in S} X_i$. We call W'_k the **kernel** of Z'_k.

Example 5

Suppose the given stochastic circuit is shown in Fig. 5(a). For this circuit, we have

$$u_1 = u_2 = u_3 = 0, w_2 = w_7 = w_8 = 1,$$
$$w_1 = w_3 = w_4 = w_5 = w_6 = 0.$$

Putting these values into the product terms $P_1, \ldots, P_5$ in Eqs. (4)–(8), respectively, we obtain

$$P'_1 = x_1[1]x_2[2], \quad P'_2 = x_3[1], \quad P'_3 = -x_2[2]x_3[1],$$
$$P'_4 = -x_1[1]x_2[2]x_3[1], \quad P'_5 = x_1[1]x_2^2[2]x_3[1].$$

Applying the simplification rules to P'_k's, we obtain their corresponding simplified expression Q'_k's as

$$Q'_1 = X_1X_2, \quad Q'_2 = X_3, \quad Q'_3 = -X_2X_3,$$
$$Q'_4 = -\min\{X_1, X_3\} \cdot X_2, \quad Q'_5 = \min\{X_1, X_3\} \cdot X_2.$$

The kernels of Q'_k's are

$$H'_1 = X_1X_2, \quad H'_2 = X_3, \quad H'_3 = X_2X_3,$$
$$H'_4 = \min\{X_1, X_3\} \cdot X_2, \quad H'_5 = \min\{X_1, X_3\} \cdot X_2.$$

Further combining Q'_k's, we obtain the final expression of the output stochastic function as $Z'_1 + Z'_2 + Z'_3$, where

$$Z'_1 = X_1X_2, \quad Z'_2 = X_3, \quad Z'_3 = -X_2X_3.$$

The kernels of Z'_i's are

$$W'_1 = X_1X_2, \quad W'_2 = X_3, \quad W'_3 = X_2X_3. \quad \square$$

By our simplification steps, it can be easily seen that for each $1 \leq i \leq M$, the term W'_i must equal one of the terms H'_k. However, for each term H'_k, there may or may not exist a term W'_i ($1 \leq i \leq M$) such that $H'_k = W'_i$. The situation that there does not exist a term W'_i ($1 \leq i \leq M$) such that $H'_k = W'_i$ happens when the sum of all the Q'_l's with the kernel equal to H'_k is 0. For example, in Example 5, we have $H'_k = W'_k$ for $k = 1, 2, 3$. However, for H'_4, there does not exist a W'_i ($1 \leq i \leq 3$) such that $H'_4 = W'_i$. The reason is because the Q'_l's with the kernel as H'_4 are Q'_4 and Q'_5 and they sum to 0.

For each H'_k ($1 \leq k \leq N$), whether or not there exists a term W'_i such that $H'_k = W'_i$ leads to two different sets of constraints for functional equivalence, which we will describe in detail in the next two subsections.

D. Establishing Constraints When H'_k Is Equal to One W'_i

By the steps discussed in the previous section, we obtain the terms $H'_1, \ldots, H'_N$. Now, for each H'_k, we establish a set of constraints that guarantees the functional equivalence between the new circuit and the given one. In this section, we describe how this is done when the term H'_k equals one W'_i.

When the term H'_k equals one W'_i, we want the simplified expression of P_k to be equal to that of P'_k. Here, P_k is the k-th term in the output stochastic function of the general circuit, obtained by the steps described in Section V-B. We establish equalities and inequalities on the $t_{k,i,j}$'s in the product term P_k to achieve the requirement. A **sufficient condition** is that for any $i_1, i_2 \in S_k$, $1 \leq j_1 \leq n_{k,i_1}$, and $1 \leq j_2 \leq n_{k,i_2}$, we have $t_{k,i_1,j_1} \neq t_{k,i_2,j_2}$, if $t'_{k,i_1,j_1} \neq t'_{k,i_2,j_2}$, and $t_{k,i_1,j_1} = t_{k,i_2,j_2}$, otherwise. If this is satisfied, then by the simplification procedure, P_k will have the same simplified expression as P'_k.

Example 6

Following Example 5, we have that the term H'_1 equals one W'_i, i.e., W'_1. In this case, we want the simplified expression of P_1 to equal that of P'_1. From P'_1, we have $t'_{1,1,1} = 1 \neq t'_{1,2,1} = 2$. Therefore, for P_1 in Eq. (4), if we want its simplified expression to equal that of P'_1, one sufficient condition is that $t_{1,1,1} \neq t_{1,2,1}$. Note that if this is satisfied, the simplified expression of P_1 will be X_1X_2, same as that of P'_1. The above sufficient condition gives the following inequality

$$u_1 + w_1 + w_7 \neq u_2 + w_2 + w_4 + w_7. \tag{9}$$

Similarly, for H'_3, we can obtain another inequality

$$u_2 + w_2 + w_5 + w_6 + w_8 \neq u_3 + w_3 + w_8. \quad \square \tag{10}$$

E. Establishing Constraints When H'_k Is not Equal to any W'_i

In this section, we describe how to establish a set of constraints when H'_k is not equal to any W'_i. Given such a H'_k, let V_k denote the set of l's ($1 \leq l \leq N$) such that $H'_l = H'_k$. Since H'_k is not equal to any W'_i, we have $\sum_{l \in V_k} Q'_l = 0$. Given that $Q'_l = c_l H'_l = c_l H'_k$, we further have $\sum_{l \in V_k} c_l = 0$. Assume that for any product term P_k, its simplified expression is Q_k and its simplified kernel is H_k. We have $Q_k = c_k H_k$, where c_k is the same coefficient of the expression Q'_k. A **sufficient condition** for the functional equivalence is that for all the $l \in V_k$, the kernels H_l's are equivalent. This condition guarantees the functional equivalence, because in this case, we have

$$\sum_{l \in V_k} Q_l = \sum_{l \in V_k} c_l H_l = H_k \sum_{l \in V_k} c_l = 0,$$

which means that those product terms that are cancelled together in the output stochastic function of the given circuit remain to be cancelled together in the output stochastic function of the new circuit. We establish the equalities and inequalities on $t_{l,i,j}$'s for all $l \in V_k$, $i \in S_l$, and $1 \leq j \leq n_{l,i}$ to satisfy that sufficient condition.

Example 7

Following Example 5, we have that the term H'_4 is not equal to any W'_i ($1 \leq i \leq 3$). The set of l's such that $H'_l = H'_4$ is $\{4, 5\}$. Therefore, for P_4 and P_5 as shown in Eqs. (7) and (8), respectively, we require their simplified kernels to be equivalent. A sufficient condition for this is $t_{5,2,1} = t_{5,2,2}$, which further gives the following equality

$$u_2 + w_2 + w_4 + w_7 = u_2 + w_2 + w_5 + w_6 + w_8. \tag{11}$$

Note that the above equality is sufficient for P_4 and P_5 to have the same simplified kernel, because under this condition, we have

$$\begin{aligned} P_5 =& x_1[u_1 + w_1 + w_7]x_2[u_2 + w_2 + w_4 + w_7] \\ & \cdot x_3[u_3 + w_3 + w_8] = -P_4. \quad \square \end{aligned}$$

It should be noted that in establishing the equality and inequality constraints for the case where the term H'_k is **not** equal to any W'_i, we do not need to enforce that P_k and P'_k have the same simplified kernels. The reason is because we have enforced that $\sum_{l \in V_k} Q_k = 0$ and hence, the actual simplified kernel of P_k does not matter. For example, although from P'_5 shown in Example 5, we have the relations such as $t'_{5,1,1} = t'_{5,3,1}$ and $t'_{5,1,1} \neq t'_{5,2,1}$, we do not need to enforce them for the corresponding $t_{k,i,j}$'s in P_5.

F. Establishing the Final ILP Constraints

Finally, we put all the equality and inequality constraints for all the H'_k's together. By our way to construct the constraints, it can be seen that if these constraints are satisfied, then the simplified expression for $\sum_{k=1}^{N} P_k$ should be the same as that for $\sum_{k=1}^{N} P'_k$, which means that the output stochastic function of the new circuit is same as that of the given circuit. Thus, the set of constraints guarantees the functional equivalence. It should be noted that in obtaining the constraints for the functional equivalence, we ignore some complicated situations that can still guarantee the functional equivalence. Therefore, this set of constraints is just a sufficient condition for the functional equivalence. Nevertheless, it works well as we will show in the experimental results.

Example 8

Given the input circuit as shown in Fig. 5(a), the set of H'_k is shown in Example 5. We put all the equality and inequality constraints for all the H'_k's together. The constraints for H'_1 and H'_3 are shown in Eqs. (9) and (10), respectively. The constraints for H'_4 and H'_5 are the same, which are shown in Eq. (11). Since P_2 is a product term with a single component, there is no constraint for H'_2. In summary, the set of constraints that is sufficient for the new stochastic circuit to be equivalent to the circuit in Fig. 5(a) is as follows:

$$\begin{aligned} & u_1 + w_1 + w_7 \neq u_2 + w_2 + w_4 + w_7, \\ & u_2 + w_2 + w_5 + w_6 + w_8 \neq u_3 + w_3 + w_8, \\ & u_2 + w_2 + w_4 + w_7 = u_2 + w_2 + w_5 + w_6 + w_8. \end{aligned} \tag{12}$$

One set of variables u_i's and w_i's that satisfies the above set of constraints is

$$u_1 = 2, u_2 = 0, u_3 = 1, u_2 = 0, w_4 = w_6 = 0,$$
$$w_1 = w_2 = w_3 = w_5 = w_7 = w_8 = 0.$$

The set of variables corresponds to the stochastic circuit shown in Fig. 5(b). It can be verified that this circuit realizes the same stochastic function as the given circuit shown in Fig. 5(a). □

Another thing to note is that the linear inequalities we obtain are of the form $t_{k_1,i_1,j_1} \neq t_{k_2,i_2,j_2}$. This kind of linear inequalities cannot be handled by an ILP solver. We need to further transform them into a set of linear equalities and inequalities of the form $A \leq B$. We apply the method from [15] to do this. As we mentioned before, in our case, each $t_{k,i,j}$ corresponds to the sum of the DFF numbers along a path. Since we eventually want to reduce the total number of DFFs inserted, the value of $t_{k,i,j}$ should be no more than the total number of DFFs in the given circuit, L. Then, an inequality $t_{k_1,i_1,j_1} \neq t_{k_2,i_2,j_2}$ can be transformed into a set of linear equalities and inequalities of the form $A \leq B$ as follows

$$t_{k_1,i_1,j_1} = \sum_{l=0}^{L} l d_{1,l}, \quad t_{k_2,i_2,j_2} = \sum_{l=0}^{L} l d_{2,l},$$
$$\sum_{l=0}^{L} d_{1,l} = 1, \ \sum_{l=0}^{L} d_{2,l} = 1,$$
$$d_{1,l} + d_{2,l} \leq 1, \text{ for } l = 0, 1, \ldots, L,$$

where for all $m = 1, 2$ and $l = 0, 1, \ldots, L$, $d_{m,l}$ is a 0-1 variable that is 1 if $t_{k_m,i_m,j_m} = l$, and 0 otherwise.

It should also be noted that the set of equality and inequality constraints for the functional equivalence established by our method has a similar form as that obtained by the method proposed in [15]; for that method, the numbers of DFFs along different input-to-output paths in the circuit are considered and the relations on these numbers are established. However, there are a few notable differences between our method and the method in [15]:

1) Our method takes a different input than the method in [15]. Specifically, our method takes a circuit with DFFs inserted as input. However, the method in [15] takes a circuit without DFFs inserted as input.
2) The method in [15] requires the additional input information on whether different pairs of circuit inputs can be correlated or not. Our method does not require that input information. It naturally captures that information through the proposed procedure.
3) Our method can take into consideration the insertion of DFFs inside the SNGs, while the method in [15] cannot.
4) The method in [15] requires that the all paths from a primary input x_i to the primary output should contain the same number of DFFs. In contrast, our method relaxes this requirement.

G. Handling Input Circuit without DFFs Inserted

It should be noted that our proposed procedure for establishing the functional equivalence constraints requires that the input circuit should have DFFs inserted. If we are only given a circuit without DFFs inserted, then we need to first transform it into one with DFFs inserted. In this case, there should be some additional information available, since otherwise, the synthesis target is not well-defined. The additional information could be given in two forms. In the first form, the additional information is like that given in [15], i.e., whether different pairs of circuit inputs can be correlated or not. Then, we can apply the VAIL algorithm [15] to generate an initial stochastic circuit with DFFs inserted. In the second form, the final function is given. Then, we can synthesize an initial stochastic circuit with DFFs inserted as follows. We first apply the two steps described in Sections V-A and V-B to obtain the output stochastic function of the general stochastic circuit. Then, we can compare the output stochastic function with the given target function to identify a feasible set of DFF numbers at all the wires that realizes the target function. This will give an initial stochastic circuit with DFFs inserted.

H. Time Complexity Analysis

In this section, we analyze the worst-case time complexity of establishing the constraints for functional equivalence. One step in this procedure is to obtain the output stochastic function of the general stochastic circuit. This step needs to apply the stochastic gate function of each gate to its input stochastic functions. For simplicity, assume that the circuit is composed of inverters, AND gates, and OR gates. For an inverter, the number of product terms in its output stochastic function roughly equals that in its input stochastic function. For a two-input gate, suppose its two input stochastic functions have k_1 and k_2 product terms, respectively. Then, the output stochastic function of an AND gate has $k_1 k_2$ product terms and that of an OR gate has $k_1 + k_2 + k_1 k_2$ product terms in the worst case. Suppose the circuit has V gates in total. For the worst case, we consider a chain of V OR gates. For all $i = 2, 3, \ldots, V$, the two inputs of the i-th OR gate in the chain are two different delayed versions of the output of the $(i-1)$-th OR gate. Assume that the number of product terms in the output stochastic function of the i-th ($1 \leq i \leq V$) OR gate is n_i. Then, we have $n_i = 2n_{i-1} + n_{i-1}^2$ and $n_1 = 3$. By solving this recursion, we can derive that $n_V = 2^{2^V} - 1$, which indicates that the final output stochastic function has $2^{2^V} - 1$ product terms. Since obtaining each product term takes constant time, the total runtime of deriving the final output stochastic function is $\Theta(2^{2^V})$, where V is the number of gates in the circuit. Since obtaining the output stochastic function of the general stochastic circuit is one step in the procedure of establishing the functional equivalence constraints, the worst-case time complexity for the entire procedure is at least $\Theta(2^{2^V})$.

VI. Minimizing DFF Number

In this section, we show an integer linear programming (ILP) based formulation for optimizing the DFF number.

We consider the case where the RNS is an LFSR. As we show in Section III, when the RNS is an LFSR, we can also insert DFFs inside the SNGs, while still keeping the stochastic function. With this extension, we can potentially further reduce the number of DFFs needed compared to the method in [15].

We assume that m DFFs are inserted inside the SNGs. The variables of the ILP formulation are m, $u_1, \ldots, u_n$, $w_1, \ldots, w_{|E|}$, where $|E|$ is the total number of edges in the circuit where DFFs can be inserted. The ILP formulation minimizes the total number of DFFs inserted. In other words, the objective function is

$$m + \sum_{i=1}^{|E|} w_i.$$

The set of constraint includes the set of equalities and inequalities that guarantees the functional equivalence, obtained

by the procedure shown in Section V. By our proposed way of inserting DFFs inside the SNGs, we can generate input SBS $x_i[u_i]$ for any $0 \leq u_i \leq m$. Therefore, we have the following additional constraints

$$u_i \leq m, \text{ for } i = 1, \ldots, n.$$

Furthermore, all the variables $m, u_1, \ldots, u_n, w_1, \ldots, w_{|E|}$ are required to be **non-negative integers**.

Example 9
Suppose we are given a stochastic circuit shown in Fig. 5(a) and its RNS is an LFSR. To obtain an equivalent circuit with the fewest DFFs, we solve the following ILP problem on the variables $m, u_1, u_2, u_3, w_1, \ldots, w_8$:

Minimize $m + w_1 + w_2 \cdots + w_8$,

subject to the set of constraints shown in Eq. (12) and the constraints $u_i \leq m$, for $i = 1, 2, 3$, where the variables $m, u_1, u_2, u_3, w_1, \ldots, w_8$ are all non-negative integers. Note that those linear inequalities of the form $A \neq B$ shown in Eq. (12) are actually transformed into a set of linear equalities and linear inequalities of the form $A \leq B$ by the method described at the end of Section V-F. □

For the situation where the RNS is not an LFSR, we cannot insert DFFs into the SNGs. However, it can be treated as a special case for the situation where the RNS is an LFSR; the special part is that we fix the variable m as 0.

Once the ILP problem is formulated, we can apply an existing ILP solver to solve it. The total runtime of the procedure for minimizing the DFF number equals the time for establishing the functional equivalence constraints plus the time for solving the ILP formulation. By the time complexity analysis in Section V-H, we can see its worst-case time complexity is at least $\Theta(2^{2^V})$, where V is the number of gates in the circuit. Although the worst-case time complexity seems prohibitively large, our experimental results in Section VIII-E show that in practice, the runtime is affordable for a normal stochastic circuit, since the value V for a typical stochastic circuit tend to be small and the time complexity for a normal stochastic circuit could be far less than that of the worst case.

VII. Minimizing Circuit Delay

In this section, we present an algorithm that minimizes the delay for a given stochastic circuit with DFFs inserted, under a constraint on the number of DFFs inserted. If we set the DFF number constraint as the minimum DFF number returned by solving the ILP formulation shown in Section VI, then our algorithm can minimize both the circuit delay and DFF number simultaneously. The algorithm is based on a key subroutine that minimizes the DFF number under a delay constraint. We first present this subroutine and then show the algorithm that minimizes the delay under a DFF number constraint.

A. Minimizing DFF Number under Delay Constraint

The subroutine to minimize the DFF number under a delay constraint is based on solving an ILP formulation. Next, we present this ILP formulation.

Given a delay constraint D, all consecutive sequences of gates are analyzed. Since we require the delay of the circuit to be no larger than D, then, for each sequence of gates such that its delay sum is larger than D, we need to insert at least one DFF within this sequence of gates to split the path into multiple shorter sub-paths. Mathematically, suppose a sequence of gates $G_1, \ldots, G_k$ has the delay sum larger than D, then we will add an inequality constraint

$$\sum w_i \geq 1,$$

where the sum is over all the w_i's from the output of G_1 to the input of G_k along the path $G_1, \ldots, G_k$.

Example 10
Consider the stochastic circuit shown in Fig. 5(c), where the number inside each gate denotes the delay of that gate. Now suppose the delay constraint is $D = 4$ time units. Then, by visiting all the consecutive sequences of gates, we find that the sequence of the inverter, the bottom AND gate, and the final OR gate, as indicated by the red line, has its delay sum larger than D. As a result, we need to insert at least one DFF within that path. This introduces an additional constraint

$$w_6 + w_8 \geq 1.$$

The above new constraint ensures that there will be at least one DFF inserted into this longer path to break it into shorter ones with delays below the constraint D. □

The above set of new path-based constraints is added to the ILP formulation for minimizing the DFF number, which is described in Section VI. This creates the ILP formulation for minimizing the DFF number under a delay constraint. By solving this ILP problem, we can obtain the fewest DFFs needed under a delay constraint.

B. Minimizing Circuit Delay under DFF Number Constraint

Now, we present our algorithm for minimizing the circuit delay under a DFF number constraint for a given stochastic circuit. The algorithm is shown in Algorithm 1.

Algorithm 1 An algorithm for minimizing the circuit delay under a DFF number constraint for a given stochastic circuit.

```
1: Inputs: A stochastic circuit C_in with DFFs inserted, a threshold
   M on the allowed number of DFFs, and a delay granularity ε.
2: Outputs: A functional equivalent circuit C_op with the minimum
   delay under the given DFF number constraint.
3: C_op ⇐ C_in;   delay_op ⇐ C_in.delay;   delay_un ⇐ 0;
4: while |delay_op − delay_un| ≥ ε do
5:    C ⇐ getMinDFFforDelay(C_in, (delay_op + delay_un)/2);
6:    if C.numDFF > M then
7:       delay_un ⇐ (delay_op + delay_un)/2;
8:    else
9:       C_op ⇐ C;   delay_op ⇐ C.delay;
10:   end if
11: end while
12: return C_op;
```

The basic idea of the algorithm is to perform a binary search to obtain the optimal delay. It takes as inputs a given stochastic circuit C_{in}, a threshold M on the allowed number of DFFs, and a delay granularity ϵ, which controls when to stop the binary search (see Line 1).

We maintain and update three important variables throughout the algorithm, the optimal circuit so far, C_{op}, the optimal delay so far, $delay_{op}$, and the largest unachievable delay so far, $delay_{un}$. Initially, we set C_{op} as C_{in}, $delay_{op}$ as the delay of the circuit C_{in}, and $delay_{un}$ as 0 (see Line 3).

Then, we perform a binary search loop to update the optimal circuit C_{op} until the gap between $delay_{op}$ and $delay_{un}$ is less than the delay granularity ϵ (see Lines 4–11). In each iteration, we first obtain the circuit C with the fewest

DFFs under the delay constraint set as the middle value of $delay_{op}$ and $delay_{un}$; this is achieved by calling the function *getMinDFFforDelay*, which realizes the subroutine described in Section VII-A (see Line 5). We compare the number of DFFs of the circuit C with the threshold M on the DFF number. If the number is larger than M, then we fail to obtain a circuit with at most M DFFs under the delay constraint that is the middle value of $delay_{op}$ and $delay_{un}$. Then, we update the largest unachievable delay, $delay_{un}$, to that middle value (see Line 7). Otherwise, we update the optimal circuit so far, C_{op}, to C, and the optimal delay so far, $delay_{op}$, to the delay of C (see Line 9). Once the loop terminates, we return C_{op} as the resulting stochastic circuit (see Line 12).

The runtime of Algorithm 1 equals the runtime of each iteration in the binary search times the number of iterations. The work in each iteration roughly equals the work for minimizing the DFF number plus the work for checking all consecutive sequences of gates in the circuits to add some additional path-based constraints. Thus, the worst-case time complexity of each iteration is at least $\Theta(2^{2^V})$, where V is the number of gates in the circuit. The number of binary search iterations depends on the granularity and the delay of the input circuit. Suppose the number of iterations is K. Then, the worst-case time complexity of Algorithm 1 is at least $\Theta(2^{2^V}K)$. Although the worst-case time complexity of Algorithm 1 is very large, our experimental results in Section VIII-E show that in practice, its runtime is affordable.

VIII. Experimental Results

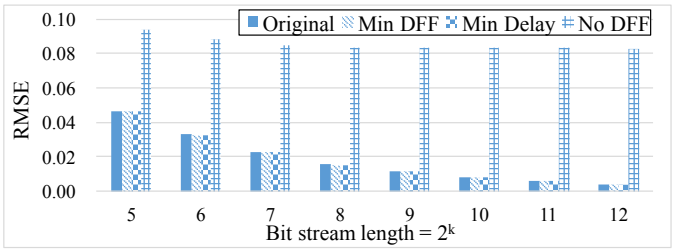
In this section, we experimentally studied our proposed algorithms for minimizing the DFF number and the circuit delay. For simplicity, we refer to these two algorithms as *min-DFF algorithm* and *min-delay algorithm*, respectively. We first verified the correctness of our proposed algorithms by comparing the error between the input circuit with DFFs inserted and the circuits produced by our algorithms. Then, we studied the performance and runtime of our algorithms and compared them with the algorithm VAIL proposed in [15]. Finally, we did a case study of applying our algorithms to optimize several stochastic circuits for arithmetic functions. All the experiments were conducted on a personal computer with a 2.8-GHz Intel® Core™ i5 CPU, an 8-GB RAM, and an Ubuntu 16.04 operating system. The mixed integer linear programming solver Gurobi Optimizer 7.5 was used to solve the ILP formulations [25].

A. Functional Equivalence of the Proposed Algorithms

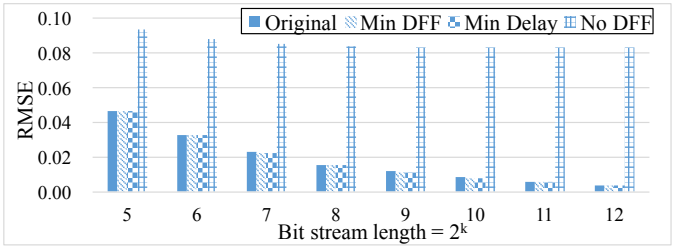
In this set of experiments, we checked whether the functional equivalence is maintained after applying our methods to the original circuit.

We randomly generated two groups of *original circuits* with DFFs inserted. The first group consists of 4-input circuits with gate counts ranging from 7 to 15. The second group consists of 6-input circuits with gate counts ranging from 11 to 21. Such a size range is typical for stochastic circuits [15]. For each input size and gate count, the number of original circuits we randomly generated is 200. For each original circuit, we applied our proposed min-DFF and min-delay algorithms to generate the corresponding *min-DFF circuit* and the *min-delay circuit*. Note that in the min-delay algorithm, we set the DFF number threshold as the DFF number obtained by the min-DFF algorithm. For each original circuit, we also obtained a so called *no-DFF circuit* by simply removing the DFFs in the original circuit. We assumed that LFSRs are used as the RNSs for the stochastic circuits.

To validate the functional equivalence, for each original circuit, we randomly generated 100 sets of input probabilities. For each set of input probabilities, we realized it with 8 bit stream lengths: $2^5, 2^6, \ldots, 2^{12}$. We applied each set of input SBSs to the original circuit and the corresponding min-DFF circuit, the min-delay circuit, and the no-DFF circuit. For each circuit, we obtained the error between the actual output of the circuit and the expected output. Then, for each circuit type, input size, and bit stream length, we obtained the root mean square error (RMSE) over all the test cases of that circuit type, input size, and bit stream length.



(a)



(b)

Fig. 6: Root mean square error comparison among original circuit and the corresponding min-DFF, min-delay, and no-DFF circuits for various bit stream lengths: (a) 4-input circuits; (b) 6-input circuits.

The RSMEs for all the circuit types and bit stream lengths are shown in Figs. 6(a) and (b) for the groups of 4-input and 6-input circuits, respectively. From the plot, we can see that both the min-DFF and the min-delay circuits have almost the same RMSE as the original circuit. This demonstrates that the circuits produced by the proposed algorithms have the same function as the input circuits with DFFs inserted, which is expected. The RMSEs of the min-DFF and the min-delay circuits decrease with the bit stream length. However, when the DFFs are removed from the original circuit, the computation result deviates far from the correct one. This big difference is caused by the functional change due to the DFF removal.

B. Performance of the Min-DFF Algorithm

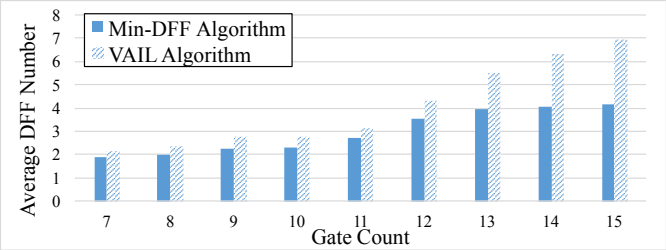
In this set of experiments, we compared our proposed min-DFF algorithm with VAIL [15] on the DFF numbers of the final stochastic circuits.

We assumed that LFSRs are used as the RNSs for the stochastic circuits. Same as the experiments in Section VIII-A, we randomly generated 4-input and 6-input circuits with gate counts ranging from 7 to 15 and from 11 to 21, respectively. For each input size and gate count, we randomly generated 300 input stochastic circuits. For each circuit and each input pair of that circuit, we randomly decided whether the input pair can be correlated or not. Each generated circuit was first fed into VAIL to produce a stochastic circuit with DFFs inserted. Then, our min-DFF algorithm was further applied to the output stochastic circuit of VAIL.

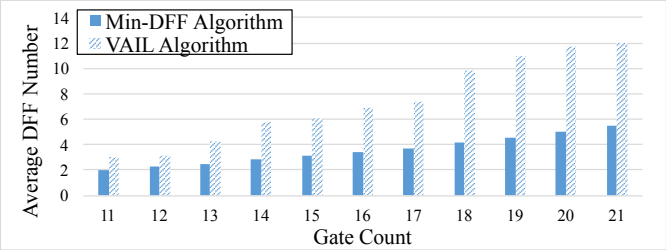
For each input size and gate count, the average DFF number over the 300 circuits was obtained. Fig. 7 compares the average DFF numbers of our min-DFF algorithm to those of VAIL.

On average, our algorithm reduces the DFF number by 22.0% for 4-input circuits and by 48.1% for 6-input ones compared to VAIL. Thus, our method can further reduce the area and power consumption of a stochastic circuit. From the figure, we can also see that the larger the gate count is, the more percentage of DFFs our algorithm saves over VAIL.

Note that if we do not allow DFFs to be inserted inside the SNGs, then our min-DFF algorithm produces the same number of DFFs as VAIL. This is expected, since VAIL is optimal when the DFFs can only be inserted outside the SNGs. This means that our proposed technique of inserting DFFs inside the SNGs plays an important role in reducing the DFF number. However, only this technique alone cannot reduce the DFFs. The reduction is achieved by further combining this technique with the proposed min-DFF algorithm.



(a)



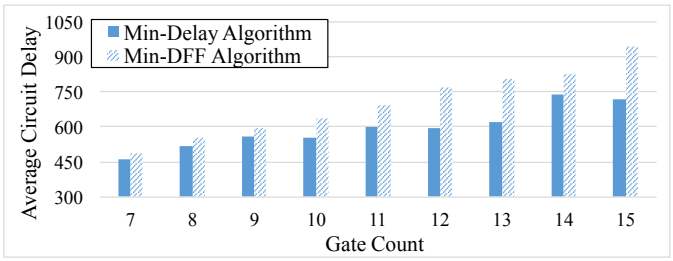
(b)

Fig. 7: DFF number comparison between our min-DFF algorithm and the algorithm VAIL [15] for circuits with various input sizes and gate counts: (a) 4-input circuits; (b) 6-input circuits.

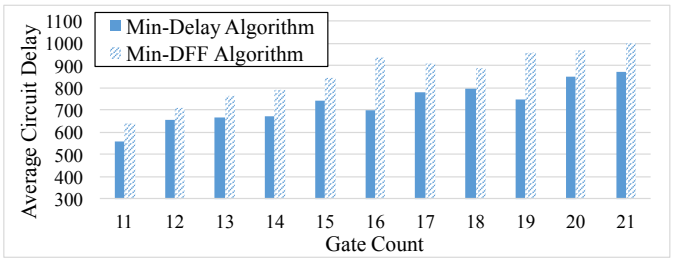
C. Performance of the Min-Delay Algorithm

In this set of experiments, we compared the delay of a given circuit before and after applying our min-delay algorithm. The same set of circuits used in the experiments in Section VIII-B was used here. The delay of each gate type was assigned with one of the three values: $100ps$, $150ps$, and $200ps$. The delay granularity ϵ in Algorithm 1 was set as $50ps$. Notice that the min-delay algorithm takes a threshold on the DFF number as an input. We chose a stringent threshold, i.e., the DFF number returned by our min-DFF algorithm. The comparison result is shown in Fig. 8.

On average, our min-delay algorithm reduces the circuit delay by 13.8% for 4-input circuits and by 14.3% for 6-input ones, compared to the min-DFF algorithm. As we analyzed in Section I, the circuit delay reduction causes the computation latency reduction and their reduction ratios are almost the same. Thus, the proposed min-delay algorithm could effectively reduce the computation latency of stochastic circuits. Note that this is achieved with the smallest possible threshold



(a)



(b)

Fig. 8: Circuit delay comparison between our min-delay algorithm and our min-DFF algorithm for circuits with various input sizes and gate counts: (a) 4-input circuits; (b) 6-input circuits. The DFF number threshold for the min-delay algorithm is set as the DFF number produced by the min-DFF algorithm.

on the DFF number. Thus, it demonstrated the effect of the proposed min-delay algorithm in reducing the computation latency even with the most stringent DFF number constraint.

D. Trade-off between DFF Number and Circuit Delay

Since our min-delay algorithm takes a DFF number threshold as an input, it can be exploited to explore the trade-off between the DFF number and the circuit delay. We studied this effect in this set of experiments.

We tested on 200 randomly generated input circuits with 6 inputs and 16 gates. For each circuit and each input pair of the circuit, we randomly decided whether the input pair can be correlated or not. We first applied VAIL to each circuit to insert DFFs. We recorded the DFF number and the circuit delay. Then, we applied our min-DFF algorithm to that circuit to find the minimum DFF number. Assume that the DFF numbers obtained by VAIL and by our min-DFF algorithm are n and m, respectively. We chose DFF number threshold as $m, m+1, \ldots, n$ and ran our min-delay algorithm for each threshold. We recorded the resultant DFF number and circuit delay.

Fig. 9 presents the experimental results. The horizontal axis D is the difference between the threshold given to the min-delay algorithm and the DFF number produced by VAIL for a given circuit. The more negative the value D is, the more DFFs are reduced over VAIL. The vertical axis is the average delay over a set of circuits. The red point represents the average circuit delay of VAIL, while the black points are the average circuit delays of our min-delay algorithm for different DFF differences D. Notice that the red point and the black point for $D = 0$ are the average of 200 circuits. However, as the DFF difference D becomes more negative, the circuits with the DFF difference D become fewer and the black point for the difference D is the average delay of fewer circuits. From the figure, we can see that as more DFFs are reduced, the circuit delay becomes larger. This is the expected trade-off between the DFF number and the circuit delay. We can also see that by properly choosing the DFF number threshold, our min-delay

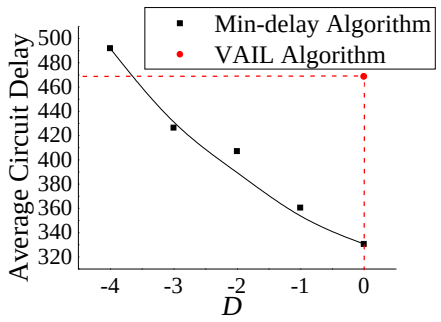


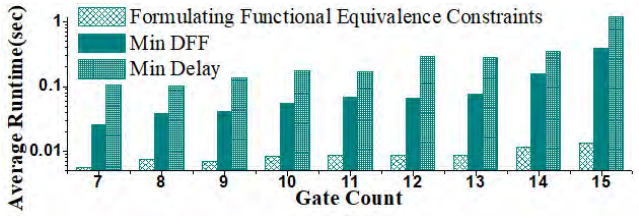
Fig. 9: The trade-off between the DFF number and the circuit delay. The horizontal axis D is the difference between the threshold given to the min-delay algorithm and the DFF number of VAIL. A negative value means reduction over the DFF number of VAIL.

algorithm can achieve simultaneous DFF number and circuit delay reduction compared to VAIL. This again demonstrates the advantage of our method over VAIL.

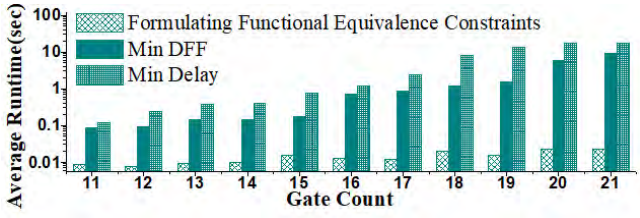
E. Runtime of the Proposed Algorithms

In this set of experiments, we studied the runtime of our proposed min-DFF and min-delay algorithms.

The same set of circuits used in Section VIII-B was used here. We assumed that LFSRs are used as RNSs for the stochastic circuits. The DFF number threshold for the min-delay algorithm was set as the DFF number returned by VAIL [15]. For each input size and gate count, we obtained the average runtime for formulating the functional equivalence constraints, that for the min-DFF algorithm, and that for the min-delay algorithm over all the test cases of that input size and gate count. Fig. 10(a) and (b) show the average runtimes for input size of 4 and 6, respectively.



(a)



(b)

Fig. 10: Average runtimes for formulating the functional equivalence constraints, for the min-DFF algorithm, and for the min-delay algorithm: (a) 4-input circuits; (b) 6-input circuits.

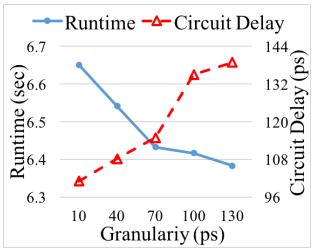
On average, the runtimes of both the min-DFF and the min-delay algorithms grow exponentially with the circuit gate count. Despite this, the average runtimes for both algorithms are within $22s$ even for the largest test case set (i.e., circuits with 6 inputs and 21 gates). The maximum runtimes for the min-DFF and the min-delay algorithms over all the tested circuits are $210s$ and $458s$, respectively. Thus, for stochastic circuits of a typical size, the maximal runtime is still acceptable. Moreover, the proposed algorithms guarantee to give the exact optimal solution. Thus, for many situations where quality is more important than runtime, our proposed approaches are an ideal choice.

By our experimental results, the formulation of functional equivalence constraints on average occupies 8.50% of the total runtime of the min-DFF algorithm. Thus, the most time-consuming part is the ILP solving. The runtime of the min-delay algorithm is on average 3.39 times that of the min-DFF algorithm.

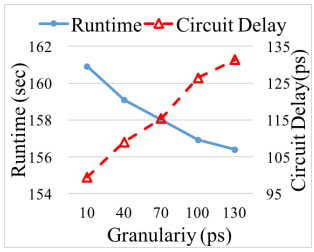
F. Trade-off between Runtime and Returned Minimal Circuit Delay of the Min-Delay Algorithm

The min-delay algorithm involves a tuning parameter ϵ. By choosing different ϵ's, we can trade the returned minimal circuit delay with the runtime of the algorithm: the large the ϵ is, the smaller the runtime and the larger the returned minimal circuit delay. In this set of experiments, we studied the trade-off between the runtime and the returned minimal circuit delay of the min-delay algorithm.

We tested on 4-input circuits with $13, 14, 15$ gates and 6-input circuits with $19, 20, 21$ gates. For each input size and gate count, 100 circuits were randomly generated as the test cases. Each circuit is composed of inverters, AND gates, and OR gates with gate delays of $10ps$, $50ps$, and $60ps$, respectively. Thus, the circuit delay can only be a multiple of $10ps$. We considered 5 delay granularity ϵ's as $10ps$, $40ps$, $70ps$, $100ps$, and $130ps$. The DFF number threshold for the min-delay algorithm was set as the value returned by VAIL [15]. For each delay granularity and each input size, we obtained the average runtime and the average minimal circuit delay over all the test cases of that input size.



(a)



(b)

Fig. 11: The trade-off between the runtime and the minimal circuit delay returned by the min-delay algorithm by setting different delay granularity ϵ's: (a) 4-input circuits; (b) 6-input circuits.

Fig. 11(a) shows the average-runtime-versus-granularity and the average-minimal-delay-versus-granularity curves for the circuits of 4 inputs, while Fig. 11(b) shows the two curves for the circuits with 6 inputs. The results show that by increasing the delay granularity, the runtime decreases and the final circuit delay increases, which is expected. However, the decrease of runtime is small. This is because by increasing the delay granularity, only few extra iterations in the binary search are saved. The experimental results indicate that a small delay granularity is preferred.

G. Case Study on SC Circuits for Arithmetic Functions

In this section, we applied our proposed min-DFF and min-delay algorithms to several SC circuits for arithmetic functions.

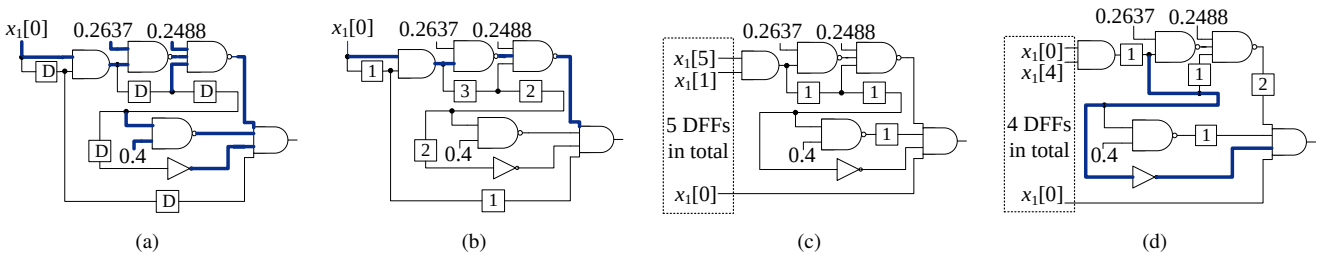


Fig. 12: SC circuits calculating $\sin(\pi x)$: (a) a circuit given by [26] with DFF locations specified; (b) the circuit with the minimal total number of DFFs inserted at the specified locations, which is used as the input to our proposed algorithms; (c) the circuit optimized by our min-DFF algorithm; (d) the circuit optimized by our min-delay algorithm.

A recent work [26] proposed an efficient implementation of a scaled version of $\sin(\pi x)$ by factoring its 9-th order Maclaurin expansion. The stochastic implementation is shown in Fig. 12(a). The numbers in the figure are constant input probabilities, which correspond to the coefficients in the factored form of the Maclaurin expansion. The generation of these constant probabilities was ignored in [26]. To make a fair comparison, we also ignored it. Each D indicates the location where DFF(s) should be inserted. However, the number of DFFs needed at each location is not shown in the paper. To recover the correct numbers of DFFs needed, we applied the VAIL algorithm to the circuit, while setting the variables w_i's at those locations with the symbol D as unknowns and the variables w_i's at the locations indicated by the blue wires in the figure as 0. The resultant circuit returned by VAIL is shown in Fig. 12(b). This circuit has the minimal total DFF numbers inserted at the specified locations.

We then applied our proposed min-DFF and min-delay algorithms to the circuit in Fig. 12(b). The DFF number threshold for the min-delay algorithm was set as the total DFF number of the input circuit. We assumed that the RNS is an LFSR. NanGate Open Cell $45nm$ Library was used, where the AND2, AND4, NAND2, NAND3, and INV gates have delays of $50ps, 90ps, 30ps, 40ps$, and $10ps$, respectively [22].

The circuits optimized by the min-DFF and the min-delay algorithms are shown in Figs. 12(c) and 12(d), respectively. Since we assumed that the RNS is an LFSR, we can share the DFFs inserted into the SNGs. The total number of DFFs inserted in the SNGs is shown inside the dotted box in each figure. From Fig. 12(c), we can see that our min-DFF algorithm decreases DFFs of the input circuit from 9 to 8. The reduction of DFF number is not large since the input circuit is already an optimal one with DFFs inserted at the specified locations. However, the result still demonstrates the ability of our min-DFF algorithm in reducing the DFF number. The critical paths of the input circuit and the circuit optimized by our min-delay algorithm are highlighted in blue in Figs. 12(b) and 12(d), respectively. Our min-delay algorithm reduces the delay of the input circuit from $210ps$ to $100ps$.

Besides $\sin(\pi x)$, we also tested our algorithms on two other stochastic circuits proposed in [26] for the functions $\cos(\pi x)$ and x^{10}. For each circuit, we further included the SNG part and applied Synopsys Design Compiler [21] to the entire circuit to obtain the area, delay, power, and energy. The results are shown in Table II. From the table, we can see that the circuit generated by the min-DFF algorithm has smaller area and power than the original circuit and the same delay as the original circuit. The circuit generated by the min-delay algorithm has almost the same area and power as the original circuit, but a much smaller delay than the original circuit. The proposed min-delay algorithm can also reduce the energy of the input circuit by up to $2.4\times$.

TABLE II: Comparison among the original circuits, the circuits produced by our min-DFF algorithm, and the circuits produced by our min-delay algorithm for three arithmetic functions.

Stochastic circuits		Area (μm^2)	Power (μW)	Delay (ps)	Energy (fJ)
$\sin(\pi x)$	Original	396.8	144.8	210	30.4
	Min-DFF	390.8	141.2	210	29.7
	Min-delay	397.7	145.1	100	14.5
$\cos(\pi x)$	Original	588.9	208.7	330	68.9
	Min-DFF	578.3	199.8	330	65.9
	Min-delay	590.0	209.0	170	35.5
x^{10}	Original	134.3	54.8	170	9.3
	Min-DFF	129.5	53.3	170	9.1
	Min-delay	134.3	55.0	70	3.9

IX. Conclusion

In this paper, we considered the problem of inserting DFFs into a stochastic circuit to reduce the hardware cost of SNGs and hence, the entire cost of a stochastic circuit. For the case where the RNS is an LFSR, we presented a way to insert DFFs inside the SNGs, while still keeping the original function. With this extension, we proposed a method for minimizing the number of DFFs inserted into a given stochastic circuit. Furthermore, we proposed a method to reduce the circuit delay of a stochastic circuit. Our proposed approach addresses two major challenges of stochastic computing, i.e., the long computation latency and the high cost to generate stochastic bit streams, simultaneously. Our experimental results demonstrated the superiority of our proposed method to the state-of-the-art method in optimizing stochastic circuits with DFFs inserted. Also, our proposed optimization methods rely on our proposed systematic way to analyze the stochastic function of a stochastic circuit with DFFs inserted. This analysis method gives key insights into the behavior of such stochastic circuits. For example, it allows us to establish the functional equivalence constraints, which are applied in our proposed optimization methods.

In our future work, we plan to further apply the proposed techniques to optimize some large stochastic computing systems, for example, the stochastic implementation of Garbor filter [27]. Also, we plan to analyze how DFF insertion affects the accuracy of stochastic circuits and develop a framework that jointly optimizes the area, delay, and accuracy of stochastic circuits with DFF insertion.

Acknowledgment

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61204042 and 61472243.

REFERENCES

[1] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, pp. 37–172.

[2] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

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

[4] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, "Compact and accurate digital filters based on stochastic computing," *accepted by IEEE Transactions on Emerging Topics in Computing*, 2016.

[5] Y. Liu and K. K. Parhi, "Architectures for recursive digital filters using stochastic computing," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3705–3718, 2016.

[6] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Design Automation Conference*, 2013, pp. 136:1–136:6.

[7] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 3, pp. 449–462, 2014.

[8] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier," in *International Symposium on FPGA*, 2016, pp. 36–41.

[9] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Design Automation Conferererence*, 2016, pp. 124:1–124:6.

[10] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Transactions on Signal Processing*, vol. 56, no. 11, pp. 5692–5703, 2008.

[11] S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Transactions on Signal Processing*, vol. 58, no. 9, pp. 4883–4896, 2010.

[12] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *accepted by IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[13] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, p. 92, 2013.

[14] S. W. Golomb, *Shift Register Sequences, Revised Ed.* Aegean Park Press, 1981.

[15] P.-S. Ting and J. P. Hayes, "Isolation-based decorrelation of stochastic circuits," in *International Conference on Computer Design*, 2016, pp. 88–95.

[16] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Design, Automation & Test in Europe*, 2014, pp. 76:1–76:4.

[17] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Design, Automation & Test in Europe*, 2017, pp. 650–653.

[18] P. K. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[19] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Asia and South Pacific Design Automation Conference*, 2016, pp. 256–261.

[20] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *International Conference on Computer-Aided Design*, 2016, pp. 102:1–102:8.

[21] Synopsys, Inc., http://www.synopsys.com/.

[22] NanGate, Inc., http://www.nangate.com/.

[23] T.-H. Chen and J. P. Hayes, "Analyzing and controlling accuracy in stochastic circuits," in *International Conference on Computer Design*, 2014, pp. 367–373.

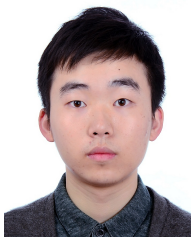
[24] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. 24, no. 6, pp. 668–670, 1975.

[25] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

[26] K. Parhi and Y. Liu, "Computing arithmetic functions using stochastic logic by series expansion," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[27] N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu, "Gabor filter based on stochastic computation," *IEEE Signal Processing Letters*, vol. 22, no. 9, pp. 1224–1228, 2015.

Zhijing Li is an undergraduate student in the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. She is interested in electronic design automation and high level synthesis.



Zhao Chen is an undergraduate student in the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. His research interests include novel computing paradigms and deep learning.

Yili Zhang is an undergraduate student in the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. He is interested in computer system and cybersecurity.

Zixin Huang is an undergraduate student in the College of Liberal Arts and Sciences at the University of Illinois at Urbana-Champaign. Her research interests focus on probabilistic programming and program analysis.

Weikang Qian is an associate professor in the University of Michigan-Shanghai Jiao Tong University Joint Institute at Shanghai Jiao Tong University. He received his Ph.D. degree in Electrical Engineering at the University of Minnesota in 2011 and his B.Eng. degree in Automation at Tsinghua University in 2006. His main research interests include electronic design automation and digital design for emerging technologies. His research works were nominated for the Best Paper Awards at the 2009 International Conference on Computer-Aided Design (ICCAD) and the 2016 International Workshop on Logic and Synthesis (IWLS).