Proceedings of the on Great Lakes Symposium on VLSI 2017, pages 475-478.

On the Role of Sequential Circuits in Stochastic Computing

Paishun Ting and John P. Hayes
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109, USA
{paishun, jhayes}@umich.edu

ABSTRACT

Interest in stochastic computing (SC) has been growing due to the need for low-cost circuitry in areas like image processing and machine learning. While combinational stochastic circuits are relatively easy to design, their limitations such as having few implementable functions and inaccuracies due to correlation, call for sequential components and design methods. The theory underlying sequential stochastic circuits is not fully understood, however. Almost all existing sequential SC circuits fall into the up/down-counter-based (UCB) class. In this paper, we identify and investigate a new SC circuit class: shift-register-based (SRB). We focus on the properties of SRB circuits and demonstrate their central role in stochastic computing. This leads to an algorithm MOUSE for sequential design optimization. By exploiting sequential stochastic equivalence, MOUSE can reduce SRB circuit cost without compromising performance or accuracy.

Keywords

Stochastic computing; sequential circuits; correlation and decorrelation; approximate computing

1. INTRODUCTION

Stochastic computing (SC) has recently shown considerable potential in applications such as image processing [1], artificial neural networks [4], and ECC decoding [8]. It processes data in the form of clocked (pseudo) random sequences of 0s and 1s, referred to as stochastic numbers (SNs). SC's key advantages are very small arithmetic circuits and high tolerance of soft errors compared with conventional "binary" computing. Its drawbacks are low accuracy and complex design needs.

In SC, a set of SNs $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ with probability values $\{X_1, X_2, \ldots, X_n\}$ is typically applied to a combinational logic circuit C realizing some Boolean function $f(x_1, x_2, \ldots, x_n)$. Viewed as a stochastic circuit, C computes an arithmetic function $F(X_1, X_2, \ldots, X_n)$ determined by f and the joint probability distribution of $\mathcal{X}$ [3]. For example, the AND gate of Fig. 1*a* has $f(x_1, x_2) = x_1 \wedge x_2$ and $F(X_1, X_2) = X_1 \cdot X_2$, assuming bit-streams $\mathbf{X}_1$ and $\mathbf{X}_2$ are independent, and so serves as a unipolar stochastic multiplier. Despite its use of bit *sequences,* it is tempting to treat C as part of a combinational rather than a sequential digital system. Indeed, the SC literature focuses on "combinational" design, except where sequential elements such as D flip-flops (DFFs) and feedback are explicitly added to C. It is clear, however, that the random number sources (RNSs) and the stochastic number generators (SNGs) driving the input bit sequences of C are inherently sequential. Although they are usually the costliest part of the system [11], RNSs and SNGs are often overlooked. Here, we examine stochastic circuit behavior focusing on its overall sequential nature. This viewpoint reveals unexpected costs, as well as design limitations and opportunities.

GLSVLSI'17, May 10–12, 2017, Banff, AB, Canada.

DOI: http://dx.doi.org/10.1145/3060403.3060453

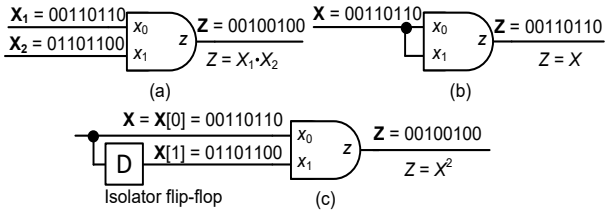

Figure 1. Stochastic circuits that serve as: (a) a multiplier; (b) a bad combinational squarer; (c) a good sequential squarer.

While "pure" combinational stochastic circuits are relatively easy to design, they have the big drawback of implementing very few functions. To see why, consider the class of single-input, single-output logic functions. It has only two members $f(x) = x$ and $g(x) = x'$, corresponding to $F(X) = X$ and $G(X) = 1 - X$ in the stochastic domain. Thus, no single-input/output functions except F and G can be implemented exactly by a combinational stochastic circuit!

Another problem is that undesired correlation among bit-streams cannot be handled by combinational circuits alone. Suppose we want to use an AND gate to compute $X \cdot X = X^2$. This requires two independent, i.e., uncorrelated, bit-streams to represent X. If the naïve scheme in Fig. 1*b* with two identical copies of $\mathbf{X}$ is used, the circuit computes X instead of X^2, and so is a bad approximation to a squarer. This problem is resolved either by using two RNSs to generate two independent copies of $\mathbf{X}$, or else by decorrelation, a process that eliminates the effect of correlation [14]. Fig. 1*c* shows how the AND squarer is decorrelated by a DFF which acts as a 1-bit delay element called an isolator [6]. Assuming that each bit of the SN $\mathbf{X}$ is independently generated, i.e., $\mathbf{X}$ is a Bernoulli sequence, then consecutive bits of $\mathbf{X}$ are also independent. Thus, the delayed version of $\mathbf{X}$, namely $\mathbf{X}[1]$, applied to x_1 and the original version $\mathbf{X} = \mathbf{X}[0]$ applied to x_0 are independent in every clock cycle. Hence, $Z = X[0] \cdot X[1] = X^2$, as desired. This highlights the fact that decorrelation makes a combinational circuit sequential.

Combinational synthesis methods for SC [2][11] expand the class of implementable functions by generating multiple independent copies of the input variable SNs, and by introducing constant SNs as ancillary inputs. The synthesis process can then focus on combinational arithmetic design. However, generation of the variable and constant inputs requires sequential elements, which make the final circuit sequential. For example, the ReSC synthesis method [11] implements functions expressed as Bernstein polynomials. A Bernstein polynomial of degree n takes the form

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

Given n independent versions of $\mathbf{X}$ to represent X, the probability that there are i 1s in the SNs is $\binom{n}{i} X^i (1 - X)^{n-i}$. If this event occurs, an SN of a constant value c_i is selected to be sent to the

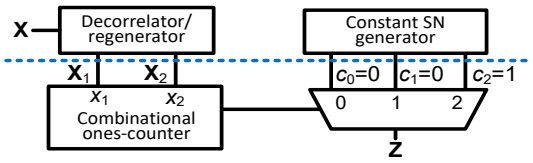

Figure 2. ReSC implementation of a squarer [11]. The sequential components above the dotted line provide independent copies of SNs needed by Eq. (1).

output. By adjusting the values of the c_i's, ReSC uses Bernstein polynomials to approximate a wide range of functions.

Example 1. Using ReSC to synthesize $F(X) = X^2$ with $n = 2$ results in the stochastic circuit of Fig. 2 where $c_0 = c_1 = 0$ and $c_2 = 1$. When $\mathbf{X}_1^{(t)} = \mathbf{X}_2^{(t)} = 1$, the combinational ones-counter outputs "2", which selects $c_2 = 1$ to be sent to the output z. In this case, the ReSC structure is logically equivalent to an AND-gate SC multiplier. This is because $z = 1$ if and only if $x_1 = x_2 = 1$. In Fig. 2, while the arithmetic components below the dotted line are combinational, the pre-processing units above the dotted line are not. To generate independent copies of SNs, sequential elements are needed.

Despite the prominent role of sequential components in SC, the literature on explicitly sequential stochastic circuits is small. Much of it, going back to early ADDIE designs [6], concerns *up/down-counter-based* (UCB) circuits. We will point out some limitations of the UCB approach, and explore a new class of sequential stochastic circuits that we call *shift-register-based* (SRB), which turn out to be widespread in SC and have some distinct advantages.

2. BACKGROUND

Sequential elements overcome the limitations of combinational SC designs by providing multiple states that implement different stochastic functions [4][6]. These depend on two factors: (1) the frequency or probability $\pi_i = p(S = s_i)$ for the circuit to visit a state s_i, and (2) the corresponding output function F_i, which is the stochastic function implemented in state s_i.

A sequential circuit is a finite-state machine (FSM) whose behavior is defined by a state-transition graph (STG). The state behavior is probabilistic due to the randomness of the input SNs, and may be viewed as a Markov chain [7]. The long-term probability π_i for the FSM to visit state s_i is obtained from the chain's steady-state probability distribution. The overall stochastic function realized is $F = \boldsymbol{\pi}^{\mathrm{T}} \cdot \mathbf{F}$, where $\boldsymbol{\pi} = [\pi_0 \pi_1 \ldots \pi_{n-1}]^{\mathrm{T}}$ and $\mathbf{F} = [F_0 F_1 \ldots F_{n-1}]^{\mathrm{T}}$.

Only a few FSM types have been studied in the SC context, notably Moore-type UCB circuits [4][6]. As depicted in Fig. 3, the STG of a UCB circuit has a "linear" structure [6], where each state can only transition to two adjacent states. Prior work has led to many variants of the basic UCB structure. Gaines studied an element called an ADDIE, which works with an up/down counter and a global feedback loop to approximate non-linear functions [6]. Brown and Card use UCB circuits with ancillary inputs to implement functions for artificial neural networks [4]. They restrict the members of the function set **F** to binary constants. More recent work extends the range and efficiency of UCB designs [10] [13].

It is noteworthy that none of the above UCB models apply to the circuit for $F(X) = X^4$ in Fig. 4*a*, where three DFFs are used to generate three independent copies of **X** [5]. Observe that its STG (Fig. 5) does not have a UCB STG's linear structure. The designs in Fig. 4 represent a large class of stochastic circuits formed by

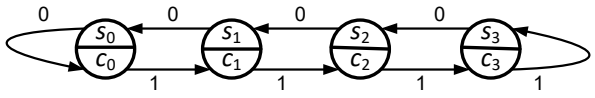

Figure 3. State transition graph (STG) for a 4-state Moore-type UCB circuit.

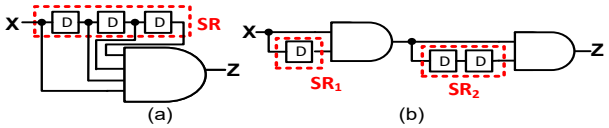

Figure 4. Implementations of $F(X) = X^4$: (a) canonical SRB design, and (b) non-canonical SRB design.

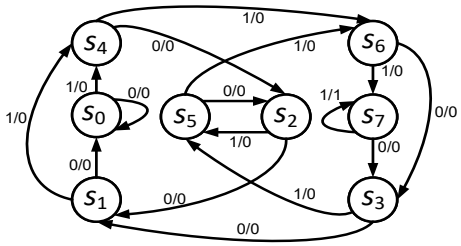

Figure 5. STG for the SRB circuit of Fig. 4*a*.

inserting DFFs into combinational logic for decorrelation purposes [5][14]. These DFFs create feed-forward shift registers, so we refer to the resulting sequential stochastic circuits as *shift-register based* (SRB). Besides enabling decorrelation, SRB circuits enrich the class of implementable stochastic functions by producing multiple independent copies of an SN. In the squarer of Fig. 1*c*, the shift register shifts the input SN **X** by 1 bit, thereby creating the independent copy of **X** the multiplier needs to function as a squarer.

Example 2. The circuit in Fig. 6 designed by the STRAUSS synthesizer [2] computes the (inverted) bipolar stochastic function $F(X) = 0.4375 - 0.25X - 0.5625X^2$. Although $F(X)$ is a single-input function, its combinational part requires two independent copies of **X** and four independent constant SNs $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$, each of value 0.5. The constant SNs can be generated by a 4-bit LFSR, as is done in [2], or they can be produced by a chain of three DFFs if a random source **c** is available, as in Fig. 6. Generation of all six independent inputs can be done by two shift registers as indicated in the figure. Taken together, they make the final circuit highly sequential.

While shift registers are a powerful tool for decorrelation, they have only been used in ad hoc fashion in SC, as in Example 2 and in some examples in [5][14]. In this work, we identify these circuits as members of a well-defined class of sequential stochastic circuits and investigate their role in SC. The rest of the paper systematically examines the properties of SRB circuits and the stochastic functions they implement. It also describes a Monte-Carlo algorithm that explores opportunities for optimizing SRB designs.

3. SRB CIRCUITS

It turns out that SRB circuits implement a classical form of FSM called a *definite* or *finite-input-memory machine* [9, Sec. 14.2]. The name reflects the fact that the current state is determined by the most recent m inputs for some fixed m. This implies that for a definite machine, every length-m input sequence is a synchronizing sequence. For example, the SRB circuit with three DFFs in Fig. 4*a* has an input memory of $m = 3$. As Fig. 5 shows,

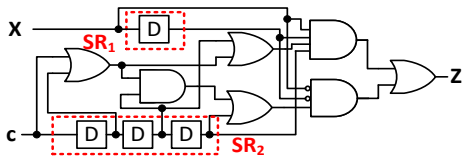

Figure 6. Circuit [2] requiring two copies of variable SN **X** and four independent constant SNs generated by shift registers SR_1 and SR_2.

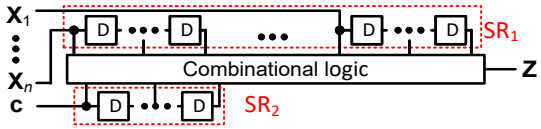

Figure 7. A general canonical single-output SRB circuit taking $\mathbf{X}_1$, $\mathbf{X}_2$, …, $\mathbf{X}_n$ as variable inputs and **c** as a constant input.

applying a 3-bit sequence like 101 leads it to s_5, regardless of the initial state.

While a definite FSM can have different implementations, it has a canonical form which is an SRB circuit with each shift register taking its input directly from a primary input [9]. Fig. 7 shows a canonical SRB circuit for SC purposes, where the shift registers generate multiple independent copies of the variable and constant inputs. The circuits in Figs. 4*a* and 6 are in canonical form. The SRB circuit in Fig. 4*b* is a non-canonical definite circuit since the internal shift register SR_2 has an internal signal as its input. We will focus on the canonical SRB forms of definite machines.

Definiteness has a useful role in SC design. As noted in [4], sequential components can introduce autocorrelation into SNs, which may reduce overall accuracy. Further, when its input SNs change value, a sequential circuit cannot respond completely until steady state is reached. While such warmup delays can be of indefinite length in general circuits, the warmup delay of an SRB circuit is bounded by its finite input memory—a definite advantage.

Fig. 8 shows two SC designs realizing a complex single-input function $tanh(2X)$. One is a UCB design from [4]; the other is a new SRB design of similar accuracy. Fig. 8*c* compares the errors of the two circuits as the input changes. Both are fed with a bipolar SN of value $X = 0.9$ for a warmup period, then X is changed to 0.3. The figure plots the average errors against the number of clock cycles after the value transition takes place. The SRB design reaches steady state 4 cycles after the input transition, while the UCB design takes about 12 cycles. The short response time of the SRB design is due to its definiteness property, which is important in delay-sensitive applications like ECC decoding. The above observations lead directly to the following result.

Theorem 1. The number of clock cycles required by an SRB circuit to reach a steady state is at most m, the input memory length.

4. SRB FUNCTIONS

Next, we examine the stochastic functions implemented by SRB circuits. We consider circuits containing a single shift register; the multiple shift-register case is similar. A shift register SR composed of m DFFs has 2^m states. Let s_i denote the state when SR stores b_i, the radix-2 form of i. For example, when SR is composed of three DFFs and is in state s_6, the DFFs are storing $b_6 = 110$. SR's STG structure is a binary, directed de Bruijn graph, where two nodes s_i and s_j are connected if b_j is a shifted value of b_i, with the empty position filled with 0 or 1. For example, the 8-node de Bruijn graph in Fig. 5 has an outgoing edge from s_7 to s_3, because $b_3 = 011$ is obtained by shifting a 0 into $b_7 = 111$.

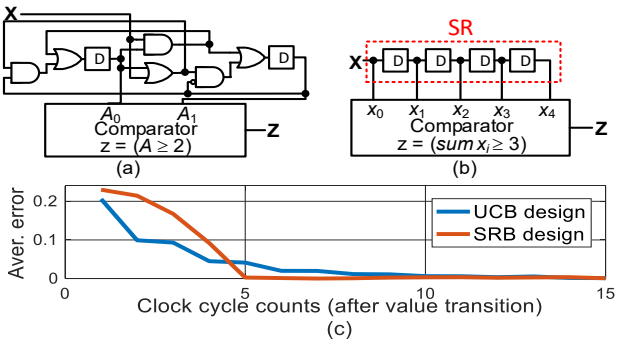

Figure 8. SC implementations of $Z = tanh(2X)$ using (a) a UCB design [4], and (b) an SRB design. (c) Error comparison between the two designs when input X changes from 0.9 to 0.3.

While standard methods [7] can be used to obtain the steady-state distribution of an SRB circuit, an intuitive approach is the following. With a shift register composed of m DFFs, associate a set of state groups $G_0, G_1, \ldots, G_m$, where state s_i belongs to group G_j, if the number of 1s in b_i is j. For example, s_3, s_5 and s_6 in the STG of Fig. 5 belong to G_2 because $b_3 = 011$, $b_5 = 101$ and $b_6 = 110$ all contain two 1s. Since the FSM has an input memory of $m = 3$, the current state is determined by the most recent three inputs, which are stored in the DFFs. This implies that π_i, the probability of visiting s_i, depends only on the most recent three inputs. Since the probability of seeing a 1 in the input SN **X** is X, the probability of receiving a 3-bit pattern with j 1s is $X^j(1-X)^{3-j}$. For example, the probability of receiving the pattern 101 is $X^2(1-X)$, which is π_5, the probability of visiting s_5. It follows that in a shift register with m DFFs, the steady-state probability distribution of $s_i \in G_j$ is $\pi_i = X^j(1-X)^{m-j}$. This leads immediately to the next result.

Theorem 2. A single-input SRB circuit having m DFFs implements the stochastic function

$$F(X) = \boldsymbol{\pi}^{\mathrm{T}}\cdot\mathbf{F} = F_0\cdot(1-X)^m + F_1\cdot X(1-X)^{m-1} + F_2\cdot X(1-X)^{m-1} + F_3\cdot X^2(1-X)^{m-2} + \ldots + F_{n-1}\cdot X^m \qquad (2)$$

which is a polynomial in X.

For example, the circuit in Fig. 4*a* has $F_0 = F_1 = \ldots = F_6 = 0$, $F_7 = X$ and $m = 3$. From Eq. (2), we know immediately that it computes $F(X) = F_7\cdot X^3 = X^4$, as expected. Eq. (2) is essentially equivalent to the Bernstein polynomial implemented by ReSC [11]; see Eq. (1). This is because by setting $F_i = c_j$ for all $s_i \in G_j$, $F(X)$ becomes $c_0\cdot(1-X)^m + c_1\cdot X(1-X)^{m-1} + c_1\cdot X(1-X)^{m-1} + \ldots + c_m\cdot X^m = \sum_{i=0}^{n} c_i \binom{n}{i} X^i(1-X)^{n-i}$. Hence, all ReSC-implementable functions can be realized by an SRB circuit. Eq. (2) can be easily extended to handle multi-variate Bernstein polynomials.

It is helpful to view a sequential stochastic circuit in terms of its state transition probabilities. The circuit's overall state behavior is given by $\boldsymbol{\pi}^{\mathrm{T}}\cdot\mathbf{F} = \sum_{i=0}^{n}\sum_{\mathbf{b}}\pi_i p(\mathbf{x}=\mathbf{b}) f_i(\mathbf{b})$. The term $\pi_i\cdot p(\mathbf{x}=\mathbf{b})$ is the probability that the machine is in state s_i and the current input is $\mathbf{x} = \mathbf{b}$. This is the probability of a state transition from s_i given input **b**, which we denote by $p(T_i^{(\mathbf{b})})$. For example, in the STG of Fig. 5, $\pi_2\cdot p(x=0) = T_2^{(0)}$ is the probability of being in s_2 with input 0; this is also the probability of a transition from s_2 to s_1. We can now describe the stochastic function of a (Mealy) sequential circuit using $p(T_i^{(\mathbf{b})})$ and the corresponding output value $f_i(\mathbf{b})$ as follows:

$$\boldsymbol{\pi}^{\mathrm{T}}\cdot\mathbf{F} = \sum_{i=0}^{n}\sum_{\mathbf{b}} p(T_i^{(\mathbf{b})}) f_i(\mathbf{b}) \qquad (3)$$

An interesting and useful fact is that many state transitions in an SRB circuit occur with the same probability, a phenomenon

```
Input:    Target SRB circuit: Ckt
Output:   Result SRB circuit with reduced cost: Ckt_out
  While iteration limit is not exceeded
    For each stochastically equivalent transition class S in Ckt
      w = sum over all fi(ai) associated with S
      Randomly pick w fi(ai)'s and assign them value 1
      For the unpicked fi(ai)'s, assign them value 0
    End For
    Ckt_cur = STAMINA(Ckt), Cost = state count of Ckt_cur
    If Cost < Cost_Low
      Cost_Low = Cost, Ckt_out = Ckt_cur
    End If
  End While
```

Figure 9. Pseudo-code for optimization algorithm MOUSE.

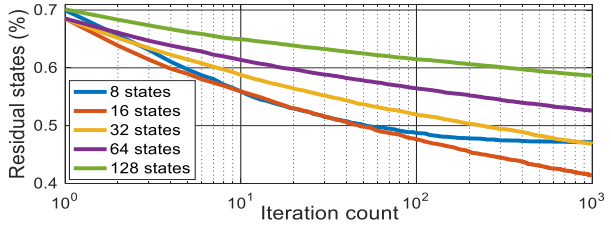

Figure 10. Percentage of residual states vs. iteration count for single-input SRB circuits with various initial state counts.

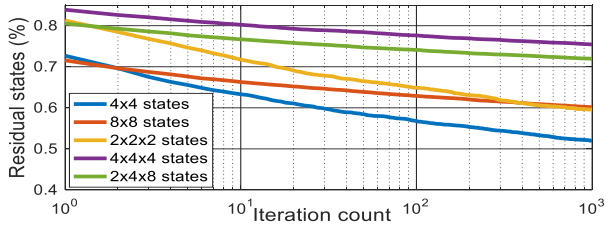

Figure 11. Percentage of residual states vs. iteration count for multi-input SRB circuits with various initial state counts.

seldom seen in UCB circuits. For example, in Fig. 5, $p(T_7^{(0)}) = p(T_5^{(1)})$, i.e., the transition from s_7 to s_3, has the same probability as that from s_5 to s_6. We define state transitions as *stochastically equivalent* if they have the same probability. Eq. (3) implies that the output values $f_i(\mathbf{a})$ and $f_j(\mathbf{b})$ of two stochastically equivalent transitions $T_i^{(\mathbf{a})}$ and $T_j^{(\mathbf{b})}$ can be switched without changing the overall stochastic function. This leads to the following easily-proven result.

Theorem 3. Suppose a sequential stochastic circuit has state transitions with output values $f_i(\mathbf{a}) \neq f_j(\mathbf{b})$. These values can be switched without changing the underlying stochastic function if and only if $p(T_i^{(\mathbf{a})}) = p(T_j^{(\mathbf{b})})$.

As Theorem 3 suggests, in sequential stochastic circuit design, especially in the SRB case, a set $\mathcal{F}$ of distinct FSMs can implement the same stochastic function, but with potentially different costs. This raises an interesting new design question: Find an FSM in $\mathcal{F}$ that, when standard optimization techniques are applied, results in the greatest hardware cost reduction. In the following section, we devise a Monte-Carlo algorithm to achieve this goal.

5. DESIGN OPTIMIZATION

We now outline an algorithm *MOUSE* (**M**onte-Carlo **O**ptimization **U**sing **S**tochastic **E**quivalence) for optimizing SRB designs. Given an SRB circuit, MOUSE randomly searches for a better design by switching state function values based on Theorem 3. The cost metric used is state count. In each iteration, MOUSE explores the space of stochastically equivalent transitions of a target circuit, and generates a new circuit implementing the same stochastic function by randomly switching equivalent state function values. A fast state reduction procedure STAMINA [12] assesses the new circuit's cost. MOUSE keeps the best design it finds until a stop criterion is met. The algorithm is summarized in Fig. 9. The final circuit, which may implement a very different FSM, preserves all the stochastic properties of the original design, including its output function and the property of finite input memory.

Since SRB circuits are a new concept in SC and hence no suitable benchmarks are available, we assess MOUSE's performance using random circuits. Fig. 10 plots the percentage of residual states against iteration count for single-input SRB circuits produced by averaging over 500 random circuits. The state count is frequently reduced by more than 50% within 1,000 iterations. For small FSMs such as SRB circuits with three DFFs, the state reduction process slows down and reaches its optimal value within 1,000 iterations. This is because with a small FSM, few possibilities exist for function value switches, so MOUSE can usually find a good solution after just a few iterations. Similar phenomena are observed for multi-input SRB circuits, as shown in Fig. 11.

MOUSE can also be applied to existing SRB designs for hardware cost reduction. For example, the multi-input SRB in Fig. 6 has 16 states. Applying STAMINA directly to this circuit results in 13 states—a 19% decrease in state count. MOUSE, on the other hand, reduces the number of states from 16 to seven after about 90 iterations. Hence, in this particular case, MOUSE enables one of the original circuit's four DFFs to be eliminated.

6. CONCLUSIONS

We examined the role of sequential components in SC, and identified two key classes of sequential stochastic circuits, UCB and SRB. Some basic properties of SRB circuits were determined, and an optimization method MOUSE for them was given. We showed that MOUSE can effectively reduce the state count by exploiting the property of sequential stochastic equivalence.

7. ACKNOWLEDGEMENT

This work was supported by Grant CCF-1318091 from the U.S. National Science Foundation.

8. REFERENCES

[1] Alaghi, A. et al. "Stochastic circuits for real-time image-processing applications." *Proc. DAC*, pp.136, 2013.

[2] Alaghi, A. and Hayes, J.P. "STRAUSS: spectral transform use in stochastic circuit synthesis." *IEEE Trans. CAD*, 34, pp. 1770-1783, 2015.

[3] Alaghi, A. and Hayes, J.P. "On the functions realized by stochastic computing circuits." *Proc. GLSVLSI*, pp.331-336, 2015.

[4] Brown, B. D. and Card, H. "Stochastic neural computation I." *IEEE Trans. Computers*, 50, pp.891-905, 2001.

[5] Chen, T.-H. and Hayes, J.P. "Analyzing and controlling accuracy in stochastic circuits." *Proc. ICCD*, pp.367-373, 2014.

[6] Gaines, B.R. "Stochastic computing systems." *Adv. Inform. Syst. Sci.* 2, pp.37-172, 1969.

[7] Gallager, R.G. *Discrete Stochastic Processes*. Springer, 1996.

[8] Gaudet, V.C. and Rapley, A.C. Iterative decoding using stochastic computation. *Electon. Letters*, 39, pp.299-301, 2003.

[9] Kohavi, Z. and Jha, N.K. *Switching and Finite Automata Theory,* 3rd ed. Cambridge Univ. Press, 2010.

[10] Li, P. et al. "The synthesis of linear finite state machine-based stochastic computational elements." *Proc. ASP-DAC*, pp. 757-762, 2012.

[11] Qian, W. et al. "An architecture for fault-tolerant computation with stochastic logic." *IEEE Trans. Comp.*, 60, pp.93-105, 2011.

[12] Rho, J.-K. et al. "Exact and heuristic algorithms for the minimization of incompletely specified state machines." *IEEE Trans. CAD*, 13, pp. 167-177, 1994.

[13] Saraf, N. and Bazargan, K. "Polynomial arithmetic using sequential stochastic logic." *Proc. GLSVLSI*, pp.245-250, 2016.

[14] Ting, P.-S. and Hayes, J.P. "Isolation-based decorrelation of stochastic circuits." *Proc. ICCD*, pp.88-95, 2016.