SC の定数生成におけるエラー率を考慮した面積コスト削減手法

坂本 雄大 † 山下 茂 ‡

1. はじめに

Stochastic Computing (以降，SC) は，ビット列中の 1 の存在確率を表す Stochastic Number (以降，SN) を用いた近似演算手法である [1]．SC は従来の二進数による演算手法と比べ，演算に必要なゲート数が非常に少ない．例えば，乗算は AND ゲート一つで実現することが可能である．また，SC は従来の演算と異なり，ビット列中の 1 の存在確率を用いているため，SN の桁の重みは均一である．よって，ビット反転が生じても，生じる誤差は小さい．以上のことから，SC は低面積，低消費電力な回路設計が可能であり，近年研究が進められている [2]．SC は主に画像処理やニューラルネットワークへの応用が期待されている ([3], [4], [5], [6], [7], [8], [9])．

SN を生成する最も一般的な手法としては，Linear Feedback Shift Register（以降，LFSR）と比較器を使用する手法がある．この手法では，多くの SN を生成する必要がある場合，回路の面積が大きくなる．例えば，SC で算術関数 $\sin x$ を演算するとき，x やいくつかの定数が必要となる場合がある．x については，x のべき乗を演算するために，x を値とする複数の異なる SN が必要であることに注意する．これは，SC において同一の SN x を正確に乗算すると，x^2 ではなく x が得られるため，異なる SN x が必要となる．より正確には，2.1 節で説明するように，x を値とする互いに相関しない複数の異なる SN が必要である．

そこで，SN を複製して同じ値の SN を小さな相関関係で生成する RRR（Register based Re-arrangement circuit using a Random bit stream）という効率的な手法が提案されている [10]．

しかし，多くの定数を値とする SN を生成する上で回路の面積をどのように削減するかはわかっていない．文献 [11] では，非常に少ない数の SN から多くの SN を生成する手法が提案されている．

よって，多くの定数を値とする SN を生成するための回路の面積を削減するために，文献 [11] の手法の使用を検討することができる．しかし，文献 [11] の手法は相互に高い相関を保持する SN を生成する．そのため，演算式によっては誤差が大きくなる可能性がある．つまり，3 章で説明するように，文献 [11] の手法がうまく機能しない可能性がある．(これは，文献 [11] の手法が相関の低い SN を必要としない別の目的のために使用されているからである.)

本論文では，上記の問題点を考慮した上で文献 [11] の手法を利用する手法を提案する．本論文では，3 つの手法を検討した．そして，いくつかの実験を行い，その手法が効果的であるかを検討した．

本論文は，以下のように構成されている．2 章で，SN 間の相関がどのように演算に影響を与えるかを含む，SC の基礎知識と RRR について説明する．3 章で，誤差を大きく増加させることなく，定数を値とする SN を生成する回路の面積を削減する手法を説明する．その章で，3 つの手法についても説明する．4 章で，提案した手法が効果的であるかを検証する実験を示す．最後に 5 章で，今後の課題についてまとめる．

2. 予備知識

2.1 Stochastic Computing

SC では，ビット列中の 1 の存在確率を数値として表す．例えば，ビット列”11101010”は 8 ビット中 5 ビットが 1 なので $\frac{5}{8}$ を表す．このようにビット列中の 1 の存在確率を数値としたものを Stochastic Number と呼ぶ．よって，SN は 1 の存在確率が等しいとき，同じ数値を表す．例えば，”10101010”と”11110000”はともに $\frac{1}{2}$ を表す SN である．SN はビット列の長さが長いほど数値の表現の精度が向上する．以降では，数値 x, y を値とする SN をそれぞれ対応する大文字 X, Y で表す．そして，X, Y における 1 の存在確率をそれぞれ $Prob(X = 1) = x, Prob(Y = 1) = y$ で表す．

SN は，図 1 に示すように 2 進数の定数と LFSR で生成された乱数を比較することで生成することができる．図 1 のような SN 生成器を Stochastic Number Generator（以降，SNG）と呼ぶ．

SN は，$[0, 1]$ の範囲の実数値のみしか表すことができない．よって，範囲外の数値に対する演算を行う場合，$[0, 1]$ の範囲に収まるように入力をスケーリングし，全体の確率演算の後で最終結果を適切に再スケーリングする必要がある．

SC は，非常に単純な論理ゲートで演算を行うことが可能である．乗算は，2 つの SN，A と B を AND ゲートに入力するだけで行うことができる．2 つの SN，A と B が互いに独立している場合，$Prob(C = 1) = Prob(A = 1) \times Prob(B = 1)$ となる．図 2 の場合，A, B ($Prob(A = 1) = \frac{4}{8}$，$Prob(B = 1) = \frac{6}{8}$) が AND ゲートに入力され

† 立命館大学大学院 情報理工学研究科
‡ 立命館大学 情報理工学部

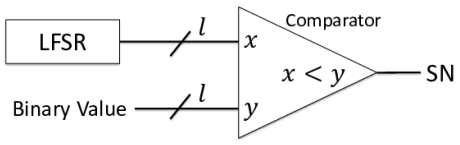

図 1: Stochastic Number Generator

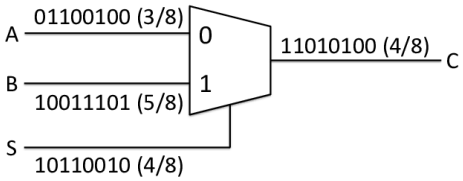

図 3: MUX による SC の加算

たとき，SN C（$Prob(C=1)=\frac{3}{8}$）が出力されている．

加算は図 3 のマルチプレクサ（以降，MUX）を用いて行うことができる．MUX を用いた加算は，2 つの SN，A と B が互いに独立している場合，$Prob(C=1) = Prob(S=1)\times Prob(A=1)+Prob(S=0)\times Prob(B=1)$ となり，A と B の加算結果が S によってスケーリングされる．なお，基本的に S の値は $\frac{1}{2}$ である（$Prob(S=1)=\frac{1}{2}$）．図 3 の場合，A, B, S（$Prob(A=1)=\frac{3}{8}$，$Prob(B=1)=\frac{5}{8}$，$Prob(S=1)=\frac{4}{8}$）に対し，C（$Prob(C=1)=\frac{4}{8}$）が出力されている．

また，加算は図 4 の OR ゲートを用いて行うこともできる．ただし，OR ゲートを用いて加算を行うことができるのは，$Prob(A=1)+Prob(B=1)$ の値が 1 を超えず，A と B がともに 1 となるビットが存在しないときのみである．図 4 の場合，A, B（$Prob(A=1)=\frac{4}{8}$，$Prob(B=1)=\frac{2}{8}$）が OR ゲートに入力されたとき，SN C（$Prob(C=1)=\frac{6}{8}$）が出力されている．

SC では 2 つの SN に相関がある場合，演算結果に誤差が生じる．また，相関の度合いがより強いほど，演算結果の誤差は大きくなる．2 つの SN の相関の強さを表す指標として，Stochastic Computing Correlation (以降，SCC) が存在する [12]．SCC の範囲は $[-1, 1]$ であり，SCC の値が 0 に近いほど，2 つの SN の相関が弱いことを表す．

AND ゲートに入力する 2 つの SN が全く同じ，すなわち相関が高い極端な場合を考える．この場合，2 つの SN の SCC は 1 である．そして，AND ゲートの出力は入力と同じ SN になる．つまり，SN，A と A を AND ゲートに入力した場合，A が出力され，A^2 を得ることはできない．特に，AND ゲートに入力する SN の値が $\frac{1}{2}$ に近い場合，2 つの入力の相関があると，誤差の絶対値は大きくなる．したがって，通常，各 SN はそれぞれ独立した SNG を使用して生成する．

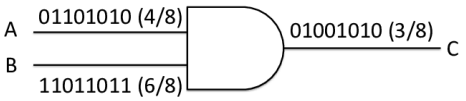

図 2: AND ゲートによる SC の乗算

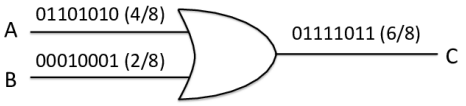

図 4: OR ゲートによる SC の加算

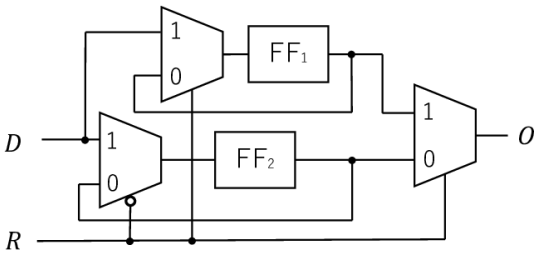

図 5: Register based Re-arrangement circuit using a Random bit stream

2.2 SN duplicators (DUP) [10]

DUP は，入力した SN と同じ値で異なるビット列を有する SN を生成するもののことを指す．したがって，DUP は次の条件を満たす必要がある．

- DUP に入力する SN を D とし，出力される SN を O とする．このとき，D と O は異なるビット列（$D \neq O$）であり，かつ，同一の値を保持する（$d = o$）．

文献 [13] では，1 ビットフリップフロップを用いた DUP が提案されている．しかし，1 ビットフリップフロップを用いた DUP の場合，入力 D を与えて得られる出力 O は，毎回同一のビット列となる．理想的な DUP を実現するには，以下の条件を満たす必要がある．

- 理想的な DUP は同じ入力 D が与えられても，出力 O のビット列は生成されるたびに異なるものとなる．

理想的な DUP として，RRR が [10] で提案されている．図 5 が RRR である．RRR は 3 つのマルチプレクサ (MUX) と 2 つの 1 ビットフリップフロップ (FF_1，FF_2)

から構成されている．図 5 の場合，R が 1 のときに FF_1 の値が出力され，入力 D が FF_1 に格納される．そのとき，FF_2 の値に変化はない．また，R が 0 のときに FF_2 の値が出力され，入力 D が FF_2 に格納される．そのとき，FF_1 の値に変化はない．RRR の場合，同一の入力 D が与えられても，入力 R によって出力 O のビット列は異なるものとなる．

また，RRR の場合，入力 D のビットのうち，最後に FF_1，FF_2 に格納されているビット以外のビットはすべて出力 O として出力される．そして，代わりに FF_1，FF_2 に最初に格納されていたビットが出力 O として出力される．つまり，ビットエラーが生じる可能性があるのは FF_1 と FF_2 の初期ビットの 2 ビットのみである．よって，RRR 使用時の誤差は入力 D のビット長を $|D|$ とすると，最大でも $\frac{2}{|D|}$ である．文献 [10] では，演算する関数 $f(x)$ の入力 x を複製するのに RRR を用いる．

3. SN の生成における回路の面積の削減

$\sin x$ はマクローリン展開によって，演算することができる．そして，$\sin x$ のマクローリン展開は a_i が定数である（$1-a_ix^2$）の積として近似することができる [13]．以下の式 (1) が $\sin x$ の演算式である．

$$\begin{aligned}\sin(x) &\simeq x-\tfrac{x^3}{3!}+\tfrac{x^5}{5!}-\tfrac{x^7}{7!}+\tfrac{x^9}{9!}-\tfrac{x^{11}}{11!}+\tfrac{x^{13}}{13!}-\tfrac{x^{15}}{15!}+\tfrac{x^{17}}{17!}\\ &= x(1-\tfrac{1}{6}x^2(1-\tfrac{1}{20}x^2(1-\tfrac{1}{42}x^2(1-\tfrac{1}{72}x^2(1-\tfrac{1}{110}x^2\\ &\quad (1-\tfrac{1}{156}x^2(1-\tfrac{1}{210}x^2(1-\tfrac{1}{272}x^2)))))))). \qquad (1)\end{aligned}$$

1 個の AND ゲートで 1 回の乗算ができ，1 個の NOT ゲートで F から $(1-F)$ を得ることができる．そのため，式 (1) は SC で非常に簡単に演算が可能である．しかし，式 (1) を演算するためには多くの SN が必要となる．上記の $\sin x$ の場合，8 つの定数値に対して，それぞれその値を保持する SN を生成する必要がある．また，x^2 を得るために x を複製し，次に x^2 を 7 回複製して x^2 を値とする 8 つの異なる SN を生成する必要がある．したがって，必要なすべての SN を生成するためには多くの SNG が必要となり，SC の利点を損なう可能性がある．そこで，複数の x や x^2 を値とする SN を生成するために 2.2 節で説明した RRR [10] がうまく機能することが示されている．

しかし，RRR は入力と同一の値を保持する SN を生成するため，異なる定数値の生成に RRR を使用することはできない．したがって，それぞれの定数値を保持する SN をそれぞれ生成する必要がある．以下では，演算の誤差をあまり増加させることなく，定数値を保持する SN を生成する SNG の面積を減らすための効率的な手法を提案する．

3.1 少ない数の SN から多数の SN を生成する手法

提案手法では，文献 [11] で提案された手法を利用し，少数の SN から多数の異なる定数値の SN を生成する．文献 [11] で提案されている手法は次のようになる．$R_1, R_2, \cdots, R_m$ を値とした m 個の SN$(r_1, r_2, \cdots, r_m)$ があるとする．このとき，AND ゲートに r_i，または r_i の否定 $(\overline{r_i})$ を入力することで R_i または $(1-R_i)$ を乗算した値を保持する SN を生成することができる．

例えば，$r_1 \cdot \overline{r_2} \cdot r_3$ で $R_1 \cdot (1-R_2) \cdot R_3$ を値とする SN を生成することができる．また，このようにして生成された SN を OR 演算することで多くの SN を生成することができる．したがって，直感的には論理回路を追加することで少ない数の SN から多くの SN を生成することができる．実際，$\frac{M}{1024}(1 \le M \le 1023)$ を値とする複数の SN を同時に生成したい場合，最大 4 個の SN で生成できることを証明している [11]．以下では，上述の少ない数の SN で多数の SN を生成する手法を GMCS（Generating Many Constant SNs from Few SNs）と呼ぶことにする．GMCS の場合，$\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ を値とする 4 個の SN で $\frac{M}{1024}(1 \le M \le 1023)$ を値とする全ての SN を同時に生成することができる．

GMCS によって，一般的な SC の定数値を保持する SN を生成する回路の面積を減らすことができる．しかし，GMCS の場合，以下の問題が生じる．式 (1) の $\sin x$ のような $(1-a_ix^2)$ の積として近似される関数を演算することを考える．GMCS によって，各 a_i を値とする SN を生成した場合，各 a_i 間に相関があるため，正しい関数の演算値を得ることは難しい．例えば，a_1 が $r_1 \cdot \overline{r_2} \cdot r_3$，$a_2$ が $r_1 \cdot r_2 \cdot \overline{r_3}$ とした場合，同じ r_1, r_2, r_3 を使用して生成するため，a_1 と a_2 の間に強い相関が存在する．したがって，実際に演算をするときに GMCS を単純に利用して，SN の数を減らすことはできない．

文献 [14] では，GMCS は Binary Combination Polynomial（以降，BCP）における定数を値とする SN の生成に用いることが提案されている．BCP では，定数を値とする SN はマルチプレクサの入力としてのみ使用される．そのため，定数を値とする SN の間に相関が生じていても問題がない [15]．一方で，本論文では定数を値とする SN 同士を乗算する回路について考える．したがって，単純に GMCS を使用することはできない．

3.2 RRR による SN の相関の削減

本論文では，GMCS によって生成された SN 同士の相関を削減するために RRR を使用することを考える．本節では，その手法について説明する．

SC によって，いくつかの関数を演算するために 8 個の定数を値とする SN，$a_1, \cdots, a_8$ を生成するとする．そのとき，GMCS を使用して，次のようにする．まず，

$r_1, \cdots, r_4$ を値とする 4 個の SN を用意する．(具体的には，文献 [11] のように $r_1, \cdots, r_4$ の値をそれぞれ $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ とする.) すると，$r_1, \cdots, r_4$ から $a_i = \frac{M}{1024} (1 \leq M \leq 1023)$ を値とする必要な SN，$a_1, \cdots, a_m$ を生成することができる．例えば，a_1 を $r_1 \cdot \overline{r_2} \cdot r_3$，$a_2$ を $r_1 \cdot r_2 \cdot r_3$ とする．この場合，上述のように a_1 と a_2 は同じ r_1，r_2，r_3 で生成されるので相関が高い．したがって，r_i と同じ値の異なる SN を生成するために RRR によって r_i を複製する．なお，r_i を m 回複製したときの各 SN を $r_i^0 (= r_i)$, r_i^1, r_i^2, $\cdots$, r_i^m とする．次に，a_1 を $r_1^0 \cdot \overline{r_2^1} \cdot r_3^2$，$a_2$ を $r_1^2 \cdot r_2^0 \cdot r_3^1$ とする．こうすることで，a_1 と a_2 の間の相関を減少することができると考えられる．

上記の考えが妥当であるかを確認するために，以下の 2 つの手法のので生成される SN の SCC を比較した．1 つ目の手法では，RRR を使用せずに，GMCS によって r_1, r_2, r_3, r_4 から 8 個のランダムな値を保持する各 a_i を生成する．2 つ目の手法では，RRR を使用し，r_1, r_2, r_3, r_4 を複製することで 8 個のランダムな値を保持する各 a_i を生成する．そして，生成された 8 つの SN から a_i と a_j の任意のペアの間で SCC を演算する．この試行を 100 回実行し平均の SCC を調べると，RRR を使用しない場合が 0.74，RRR を使用する場合が 0.28 であった．

このことから，GMCS を使用して定数を値とする SN を生成するための入力数を減らす場合，RRR を使用することで演算の精度を高めることができると考えられる．

3.3 LFSR の共有手法

本論文では，提案した手法に加え，LFSR の数を削減することを検討する．本研究では，複数の SNG や RRR の間で LFSR を共有する手法を使用する．これは，文献 [15] で提案されている．2 つ（またはそれ以上）の SN を生成するとき，複数の SNG や RRR の間で LSFR を共有する．しかし，LSFR を単純に共有すると生成された SN の相関は大きくなる．そこで，相関を小さくするために各比較器に入力される LFSR の値をそれぞれ循環ビットシフトする．具体的には，図 6 のようにして 1 個の LSFR から 2 個（またはそれ以上）の SN を生成する．図 6 では，2 個の SN，C_1 と C_2 を生成している．

実際に，各ビットシフト量ごとの SCC の平均の値を調べる．表 1 が各ビットシフト量ごとの $(1,1), (1,2), (1,3), \cdots, (255,253), (255,254), (255,255)$ の全ての組み合わせの $SCC(C_1, C_2)$ の平均値を示す．(このとき，組み合わせは全部で $255^2 = 65,025$ である.)

表 1 から，C_1 と C_2 との間の相関は循環ビットシフトによって削減され，LSFR のビット長が l の場合，ビットシフト量 k が $\frac{l}{2}$ のときに相関が最も低くなることがわかる．よって，以下の提案では LSFR を共有する異なる比較器の入力間で，できるだけ均等にシフト量を設定する．

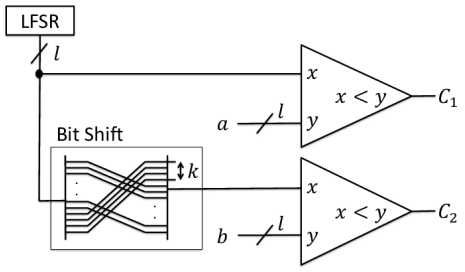

図 6: SNG における LFSR の共有

表 1: シフト量ごとの平均の SCC（$l = 8$）

シフト量	$SCC(C_1, C_2)$
$k = 0$	1.000000
$k = 1$	0.684493
$k = 2$	0.451243
$k = 3$	0.317352
$k = 4$	0.274315
$k = 5$	0.317352
$k = 6$	0.451243
$k = 7$	0.684493

3.4 SN の生成における回路の面積の削減

本節では，ここまでで話した内容を基に定数を値とする SN の生成の面積を削減する手法を説明する．本研究では，これまでの内容を基に図 7 に示す手法を提案する．図 7 は，SC において，$\sin x$ を演算するために使用する x^2 と定数を値とする SN$(a_1 \cdots a_m)$ をどのように生成するかを示している．

図 7 は，精度が $\frac{1}{1024}$ の 8 個の定数を値とする SN, $a_1, \cdots, a_8$ を使用するため，GMCS の場合は入力として 4 個の SN，$r_1, \cdots, r_4$ が必要となる．図 7 では，LFSR 1 を共有して，x と $r_1, \cdots, r_4$ を生成する．次に，3.2 節で示したように RRR を使用して r_i を複製し，$r_i^0 (= r_i)$, r_i^1, r_i^2, $\cdots$, r_i^7 を生成する．そして，$r_1^{i_1}$, $r_2^{i_2}$, $r_3^{i_3}$, $r_4^{i_4}$ を用いて，a_i を生成する．(a_i を生成する各論理回路は，$r_1, \cdots, r_4$ と同じ値を持つ 4 個の入力の SN であるが，上記のように RRR を使用することで $a_1, \cdots, a_4$ の相関は低くなるはずである.) 提案手法では，各 RRR に対して 1 個の SN（図 5 の R）が必要となる．LFSR 2 は，r_i を複製する全ての RRR に使用する全ての SN を生成するために共有される．また，LFSR 3 は x を複製する全ての RRR に使用する全ての SN を生成するために共有される．

したがって，図 7 の提案を実現するためには，3 個の

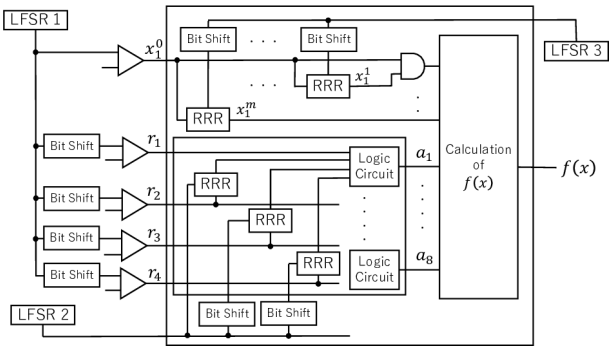

図 7: SC の演算における提案手法

LSFR と 5 個の比較器と，RRR とビットシフトが必要となる．LSFR と比較器の面積がその中で大きく回路の面積を占めるので，以下では，面積に関しては主に LSFR と比較器を考慮する．

ここで，さらに回路の面積を減らす可能性を考える．図 7 の 3 個の LFSR を共有することを考える．実際に，図 7 において LFSR 2 と LFSR 3 を共有した場合と 3 個の LFSR を全て共有した場合に（a_i と a_j の間の）SCC の値がどのように変化するかを調べた．ランダムに生成された $a_1, \cdots, a_8$ の平均の SCC の値を演算する試行を 100 回行った結果，LFSR 2 と LFSR 3 を共有した場合の平均の SCC が 0.301，3 個の LFSR を全て共有した場合の平均の SCC が 0.295 であった．

以上の結果より，本提案では GMCS に RRR を用いた手法で LFSR を共有しても，その差はわずかであると考えられる．そこで，以下の 3 つの手法を提案する．3 つの手法の違いは，どのように LSFR を共有するかであり，4 章で実験による比較を行う．

Method 1: この手法では，LFSR 1，LFSR 2，LFSR 3 で共有を行わない．つまり，この手法では 3 個の LFSR が必要である．

Method 2: この手法では，すべての RRR に対して 1 個の LFSR を共有する．つまり，LFSR 2 と LFSR 3 を共有する．よって，この手法では 2 個の LFSR が必要である．

Method 3: この手法では，LFSR 1，LFSR 2，LFSR 3 の 3 個の LFSR を全て共有する．つまり，この手法では 1 個の LFSR が必要である．

4. 実験と考察

本章では，特定の式を GMCS と RRR を用いた Method 1 から Method 3 の手法とそれ以外の手法で演算したときの誤差とその回路の面積を調べ，結果を考察する．本実験では，提案手法の効率性を確認するために 3 つの提案手法に加え，以下の GMCS と RRR のどちらかまたはその両方を使用しない手法 Method 4 から Method 8 でも実験を行った．

本実験では，$\sin x$, $\cos x$, $\log(x+1)$ を演算したときの誤差を比較した．$\sin x$ と $\cos x$ は，いずれも 3 章の式 (1) のような $(1-a_i x^2)$ の積として近似される [13]．したがって，$\sin x$ と $\cos x$ を演算するには 8 つの定数の SN，1 つの x，7 つの x^2 が必要となる．また，$\log(x+1)$ は $(1-a_i x)$ の積として近似される．したがって，$\log(x+1)$ を演算するには 8 つの定数の SN，8 つの x が必要となる．

Method 4: この手法は，GMCS を使用して図 7 に示すように，必要な 8 個の定数の SN（$a_1, \cdots, a_8$）を 4 つの SN（$r_1, \cdots, r_4$）で生成する．また，この手法では（$a_1, \cdots, a_8$）の生成に RRR を使用しない．よって，図 7 の LFSR 2 は使用されない．この手法では，2 個の LFSR を使用する．LFSR 1 が x と 8 個の SN（$a_1, \cdots, a_8$）を生成するために使用される．そして，LFSR 3 は x および x^2 を複製する RRR に使用される．

Method 5: この手法は，Method 4 とほぼ同様の手法である．しかし，Method 4 で使用される 2 個の LFSR を共有して 1 個とする．つまり，この手法で使用する LFSR は 1 個である．

Method 6: この手法では GMCS を使用しない．つまり，図 1 の SNG から直接 8 個の SN（$a_1, \cdots, a_8$）を生成する必要がある．よって，8 個の比較器が必要となる．また，この場合は 8 個の SN（$a_1, \cdots, a_8$）を生成するために RRR を使用しない．よって，図 7 の LFSR 2 は使用されない．この手法では，2 個の LFSR を使用する．LFSR 1 が x と 8 個の SN（$a_1, \cdots, a_8$）を生成するために使用される．そして，LFSR 3 は x および x^2 を複製する RRR に使用される．

Method 7: この手法は，Method 6 とほぼ同様の手法である．しかし，Method 6 で使用される 2 個の LFSR を共有して 1 個とする．つまり，この手法で使用する LFSR は 1 個である．

Method 8: 上記の Method 4 から Method 7 では，LFSR1 を用いて x と 8 個の SN（$a_1, \cdots, a_8$）を生成する．一方で，この手法では各 a_i を生成するためにそれぞれ別の LFSR を使用する．つまり，8 個の SN（$a_1, \cdots, a_8$）を生成するために 8 個の LFSR を使用する．また，この手法ではさらに 2 個の LFSR を使用する．1 個は x を生成するために使用し，1 個は x を複製する RRR に使用される．したがって，この手法では合計 10 個の LFSR を使用する．

8 つの手法すべてにおいて，x や $a_1, \cdots a_8$，または，$a_1, \cdots a_8$ を生成するための $r_1, \cdots, r_4$ の生成に使用する LFSR を共有する．なお，この 8 つの手法の場合，Method 8 が最も演算の誤差が少ないと考えられる．

表 2: 各手法における回路の面積

Method	♯ LFSRs	♯ comparators	♯ RRRs
1	3	5	36
2	2	5	36
3	1	5	36
4	2	5	8
5	1	5	8
6	2	9	8
7	1	9	8
8	10	9	8

本実験では，演算の精度を $\frac{1}{1024}$ とする．よって，各 LFSR のビット長は 10 ビットである．そのため，1 つの LFSR に対して $2^{10} = 1024$ の異なる初期値が存在する．本実験では，使用する LFSR が 1 個の手法の場合，1024 通りの全ての初期値を試した．しかし，2 個以上の LFSR を使用する手法の場合，すべてのケースを試すことは困難である．よって，本実験では 2 個以上の LFSR を使用する手法の場合，すべての LFSR の初期値をランダムに指定した試行を 1024 回行う．また，x は $\frac{1}{1024}, \frac{2}{1024}, \cdots, \frac{1024}{1024}$ の 1024 パターン存在する．しかし，x について 1024 パターンをすべて試すのは非常に時間がかかる．そこで，本実験では x の値を $\frac{2}{1024}$ から順に $\frac{50}{1024}$ ずつ増やして演算を行った．(つまり，$x = \frac{2}{1024}, \frac{52}{1024}, \frac{102}{1024}, \cdots, \frac{1002}{1024}$ となる．)

さらに，本実験では $\sin x$ のマクローリン展開における定数の値をランダムに指定した場合についても試した．(つまり，$(1 - a_i x^2)$ の a_i の値をランダムにした式についても演算を行った．) そして，a_i の値をランダムにした場合として以下の 2 つの実験を行った．(A)：式 1 の各定数 a_i は $\frac{1}{1024}$ から $\frac{1023}{1024}$ の値からランダムに指定される．(B)：式 1 における各定数 a_i は $\frac{412}{1024}$ から $\frac{612}{1024}$ の値から指定される．本実験で（B）を行った理由は，2.1 節で述べたように SN の値が $\frac{1}{2}$ に近い場合，2 個の SN の乗算に対する SN の相関の影響が非常に大きくなるためである．

4.1 回路の面積の比較

最初に，8 つの手法の回路の面積を比較する．表 2 は，各手法の LFSR，比較器，および RRR の数を表す．

ここで，RRR に比べて LSFR（10 ビットシフトレジスタ）と 10 ビット比較器のほうが面積が大きいことは明らかである．よって，上記の 8 つの手法は以下のように分類することができると考えられる．

- 非常に面積の大きい手法: Method 8
- 面積の大きい手法: Method 6 と Method 7
- 中間の手法: Method 1
- 面積の小さい手法: Method 2 と Method 3
- 非常に面積の小さい手法: Method 4 と Method 5

4.2 演算の誤差の比較

表 3 は，各手法の演算結果の平均誤差を表す．表 3 の各値は，Method 1 に対する比率を表している．(よって，値が低いほど演算結果が良い．) 表 3 の 2 番目と 3 番目の列は，式 (1) における各定数 a_i をランダムに指定した式の結果を表している．2 番目の列が，各定数 a_i を $\frac{1}{1024}$ から $\frac{1023}{1024}$ の値からランダムに指定した時の結果である．また，3 番目の列が，各定数 a_i を $\frac{412}{1024}$ から $\frac{612}{1024}$ の値からランダムに指定した時の結果である．そして，4〜6 番目の列が $\sin x$, $\cos x$, $\log(x+1)$ のそれぞれの演算結果である．

表 3 から，以下のことがわかる．

- Method 8 は演算の誤差の点において，最良である．($\sin x$ は例外で，Method 1 と比べてわずかに演算結果が悪い．)
- Method 1 と Method 2 は全てのテストケースにおいて，悪い結果が生じていない．
- Method 3 から Method 7 はいくつかのまたは，ほぼすべてのテストケースにおいて演算結果が悪くなる場合があり，うまくいかない．

2.1 節で述べたように，特に入力の値が 1/2 に近い場合，乗算に対する入力の相関が結果に大きく影響を与える．実際に，Method 3 から Method 7 は Random (B) において，演算結果が悪い．しかし，Method 1 と Method 2 は Random (B) における相関の問題を解決していることがわかる．つまり，Method 1 と Method 2 は Random (B) において，良い結果をもたらしている．

4.3 考察

実験結果より，Method 1 and Method 2 は演算結果において，ほぼ同等であり，Method 3 は提案した手法の中で演算結果は悪いものとなった．そして，回路の面積を考慮すると，Method 2 が提案した手法の中で最も良い手法であると考えられる．

提案手法と Method 4 から Method 8 を以下で比較する．Method 8 は精度の点では最良であると考えられる．しかし，回路の面積という点で非常に大きくなる．Method 6 と Method 7 の場合は提案手法と比べ，演算の誤差と回路の面積の両方において，劣っていることが

表 3: 平均の誤差の比率

Method	Random (A)	Random (B)	$\sin x$	$\cos x$	$\log(x+1)$
1	1.000	1.000	1.000	1.000	1.000
2	0.996	1.031	1.013	0.924	1.194
3	1.286	2.666	1.630	3.328	1.143
4	1.265	7.049	0.674	0.681	1.330
5	1.527	7.253	1.228	1.947	1.375
6	1.137	4.214	3.077	1.966	1.342
7	1.590	6.233	4.086	4.447	1.354
8	0.467	0.783	1.050	0.981	0.332

わかる．Method 4 と Method 5 は提案手法と比べ，回路の面積という点で優れている．また，Method 4 に関しては $\sin x$ と $\cos x$ の演算結果も良い．しかし，Method 4 と Method 5 は一般的な場合の演算で提案手法と比べて，演算結果が悪化すると考えられる．特に，定数の値が $\frac{1}{2}$ に近いほど結果が悪くなることが Random (B) から判断できる．

結論として，Method 2 は上記の 8 つの手法の中で演算の精度の面においてほとんどの場合で比較的良い結果をもたらし，回路の面積においても比較的小さい．よって，Method 2 は 8 つの手法の中で最良の手法であると考えられる．ただし，Method 2 が常に最良な結果をもたらす手法であるとは限らない．いくつかのケースにおいては，Method 2 よりも他の手法のほうが優れている．

5. まとめ

本論文では，回路の面積と演算の誤差を考慮して，定数を値とする SN の様々な生成手法を検討した．提案手法では，GMCS を用いて生成する SN の数を減らし，RRR を用いて GMCS の問題点である SC の間の相関を削減する手法を提案した．実験結果より，Method 2 が回路の面積と演算結果の良いトレードオフを実現することがわかる．つまり，提案手法は回路の面積と演算精度を十分に考慮して，定数を値とする SN を生成するということがわかる．今後の研究としては，他の関数で実験を行い，より多くの結果を得る必要がある．また，本論文では LFSR の初期値は考慮していない．しかし，LFSR の初期値も演算の誤差に影響を与える可能性があるため，今後，LFSR の初期値の影響も検討する必要がある．

参考文献

[1] BR Gaines, Stochastic computing systems, Advances in information systems science, pp. 37-172, 1969.

[2] Alaghi, Armin and Hayes, John P. Survey of Stochastic Computing. ACM Trans. Embed. Comput. Syst., Vol. 12, No. 2s, pp. 92:1–92:19, May 2013.

[3] Armin Alaghi, Cheng Li, and John P. Hayes. Stochastic circuits for real-time image-processing applications. In Proceedings of the 50th Annual Design Automation Conference, pp. 136:1 - 136:6, 2013.

[4] Bradley D. Brown and Howard C. Card. Stochastic neural computation i: Computational elements. IEEE Trans. Comput., 50(9):891-905, September 2001.

[5] Shoichi Iizuka, Masafumi Mizuno, Dan Kuroda, Masanori Hashimoto, and Takao Onoye. Stochastic error rate estimation for adaptive speed control with field delay testing. In Proceedings of the International Conference on Computer-Aided Design, ICCAD '13, pages 107-114, Piscataway, NJ, USA, 2013. IEEE Press.

[6] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In Proceedings of the 53rd Annual Design Automation Conference, pp. 124:1 - 124:6, 2016.

[7] Peng Li and David J. Lilja. Using stochastic computing to implement digital image processing algorithms. In Proceedings of the 2011 IEEE 29th International Conference on Computer Design, ICCD '11, pages 154-161, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Yin Liu and Keshab K. Parhi. Computing hyperbolic tangent and sigmoid functions using stochastic logic. In Conference Record of the 50th Asilomar Conference on Signals, Systems and Computers, ACSSC 2016, pages 1580-1585. IEEE Computer Society, 3 2017.

[9] Zhiheng Wang, Naman Saraf, Kia Bazargan, and Arnd Scheel. Randomness meets feedback: Stochastic implementation of logistic map dynamical system. In Proceedings of the 52Nd Annual Design Automation Conference, DAC '15, pages 132:1-132:7, New York, NY, USA, 2015. ACM.

[10] Ryota Ishikawa. Masashi Tawada. Masao Yanagisawa. Nozomu Togawa. Stochastic number duplicators based on bit re-arrangement using randomized bit streams. IEICE Transactions on Fundamentals of Electronics,Communications and Computer Sciences, E101-A(7), 2018, To Appear.

[11] Ritsuko Muguruma and Shigeru Yamashita, Stochastic Number Generation with the Minimum Inputs, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, pp. 1661-1671, 2017.

[12] A. Alaghi and J. P. Hayes. Exploiting correlation in stochastic circuit design. In 2013 IEEE 31st International Conference on Computer Design (ICCD), volume 00, pages 39-46, Oct. 2013.

[13] K. Parhi and Y. Liu. Computing arithmetic functions using stochastic logic by series expansion. IEEE Transactions on Emerging Topics in Computing, page 1, 2016.

[14] Zheng Zhao and Weikang Qian, A General Design of Stochastic Circuit and Its Synthesis, Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 1467-1472, 2015.

[15] Hideyuki Ichihara. Shota Ishii. Daiki Sunamori. Tsuyoshi Iwagaki. Tomoo Inoue, Compact and accurate stochastic circuits with shared random number sources, IEEE 32nd International Conference on Computer Design (ICCD), pp. 361-366, 2014