Vol.2018-ARC-230 No.36
Vol.2018-SLDM-183 No.36
Vol.2018-EMB-47 No.36
2018/3/8

効率的なストカスティック数複製器と合成関数回路を用いたその評価

石川 遼太[1]　多和田 雅師[1]　柳澤 政生[1]　戸川 望[1]

概要：人工知能や，画像処理などの分野のハードウェア実装のため，回路面積を小さくできる計算方法の需要が高まっている．ストカスティックコンピューティングでは算術演算を単純な論理回路で実装できるため，注目を集めている．ストカスティックコンピューティングでは，ランダムに生成されたビット列 (ストカスティック数) が論理回路に 1 ビットずつ入力される．同じ値を複数回回路中で使用する場合，ストカスティック数を複製する必要があるが，複製されたものが独立でない場合，それらを用いた演算の出力は正しいものにならない．本稿では，RRR 複製器と呼ばれる複製器を紹介し，再収斂を持つ合成関数回路にそれを実装，評価する．RRR 複製器ではビット並び替えによって複製され，複製されたストカスティック数は入力ストカスティック数に対して等しい値と独立なビット列を持つ．この RRR 複製器を合成関数回路に埋め込み，その精度をシミュレーションにより算出し，論理合成によりその面積と遅延を算出する．合成関数回路に実装した際に，既存の複製器と比べて MSE を 64%–94%削減することを確認した．

1. はじめに

人工知能や，画像処理などの分野の情報量の増大により，回路面積が抑えられる計算手法が注目されている．回路面積が小さい計算手法として，算術演算を単純な論理回路で実装できるストカスティックコンピューティング (SC)[1] があげられる．SC では，ストカスティック数 (SN) を用いて計算を行なう．SN はランダムに生成されたビット列から構成され，SN の値はそのビット列の内の 1 の割合で定義される．SC について，[2], [3] ではニューラルネットワークの，[4], [5] では画像処理の実装例が実験されている．

SN を用いた算術演算回路を構成するとき，多くの場合は SN の複製が必要になる．例えば，回路内の信号パスにマルチファンアウトがある場合は，信号パスに沿って値を複製する必要がある．しかし，複製された SN が非独立であれば，2 章で示すように，その算術演算結果には誤差が生まれる．

入力 SN と同じ値を持つが，異なるビット列を持つ SN を出力する生成器を考える．このような生成器は複製器と呼ばれる．複製器の最も簡単な実装は，2 章の図 3 にある，FF による 1 ビットシフトレジスタである [6]．ビットシフトにより，入力 SN と出力 SN は異なるビット列になりつつも，十分に長いビット列においては 1 の出現回数を同じと見なしてよい．しかし，入力 SN に対して出力 SN が一意に決定されるため，再収斂を持つ回路でビットシフトによって複製されたビット列が複数回使われると，非独立な SN で演算を行うことになり正しい出力は得られない．

本稿では，*RRR* (Register based Re-arrangement circuit using a Random bit stream) という複製器を紹介し，再収斂パスを含む算術演算回路に埋め込み，評価する．RRR 複製器では，乱数によるビット並び替えにより，値が等しい，ビット列が異なる SN を出力する．実験結果では，RRR 複製器は MSE (Mean Square Error) を既存手法 [6] と比べて 64–94%削減し，再収斂パスを含む回路で，RRR 複製器は正しい結果を得られることを示している．

本稿の貢献点は以下の通りである．

(1) ビット並び替えに基づく SN 複製器である，RRR 複製器を導入する．RRR 複製器は，入力 SN と等しい値を持つ SN を出力する．また，同じビット列で表される入力 SN が複数回入力されても出力 SN は複製の度に異なる．
(2) RRR 複製器が非独立な再収斂パス[*1] を持つ回路 (本稿では $\exp(-x^2)$ の回路) では既存の複製器に比べ，MSE を 64–94%削減する．ビット長を長くすると，より MSE を抑えることができる．
(3) RRR 複製器が独立な再収斂パス[*1]を持つ回路 (本稿では x^2 の回路) でも，既存の複製器と比べて MSE はほとんど変わらない．

[1] 早稲田大学

[*1] 独立・非独立な再収斂パスについて，4 章で詳しく議論する．

Vol.2018-ARC-230 No.36
Vol.2018-SLDM-183 No.36
Vol.2018-EMB-47 No.36
2018/3/8

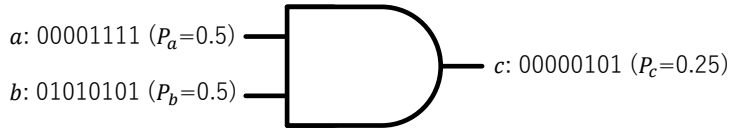

図 1 AND ゲートを用いた SN の乗算の例.

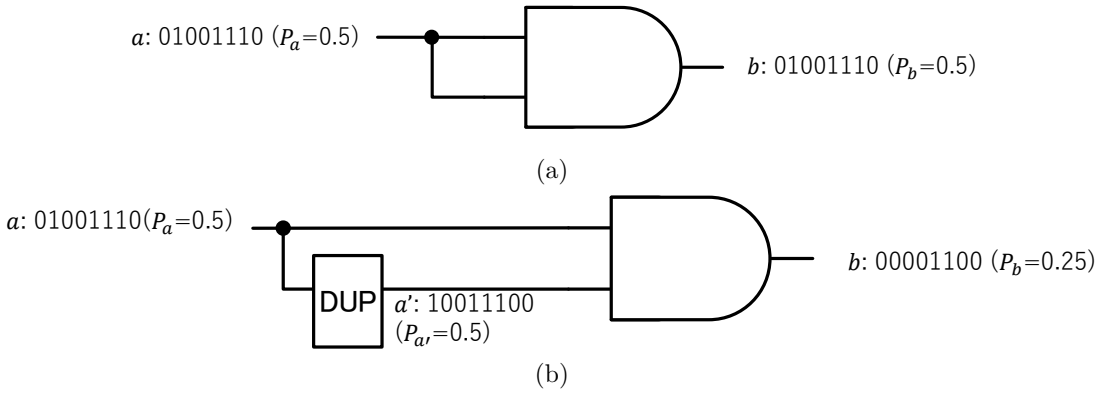

図 2 AND ゲートを用いた SN の二乗器の例. (a) 入力 SN と同じ出力が得られ，正しく演算が行われていない. (b) 正しく 2 乗されている.

本稿の構成は以下の通りである．2 章ではストカスティック数 (SN) を紹介し，何故 SN の複製器が必要なのかを議論する．3 章では乱数によるビット並び替えに基づくストカスティック数複製器である RRR 複製器を紹介する．4 章では実験の評価を通して RRR 複製器の有用性を示す．5 章では本稿をまとめる．

2. ストカスティック数の背景とその複製器

2.1 ストカスティック数 (SN)

ストカスティックコンピューティング (SC) では，ストカスティック数 (SN) を用いて計算する．SN は，各ビットが 0 と 1 で構成される任意の長さのビット列である．SN x について，ビット長を $|x|$，i 番目のビットを x_i と表す．SN x のビット列の内の 1 の出現回数を S_x とすると，x の値 V_x は 以下の式 (1) で定義される．

$$V_x = P_x = S_x/|x| \tag{1}$$

ここで，P_x は x のビット列中の 1 の出現頻度であり，$0 \leq P_x \leq 1$ が成り立つ．例えば，$x = 01000100$ のとき，その値は $V_x = 0.25$ となる．このように $V_x = P_x$ を満たす表現方法を単極表現と呼ぶ．単極表現では，$0 \leq V_x \leq 1$ が成り立つ．単極表現の他にも両極表現 ($V_x = 2 \times P_x - 1$ で定義され，$-1 \leq V_x \leq 1$ が成り立つ) など，SN の表現方法は多数あるが，本稿では，単極表現のみを取り扱う．

SC では，論理回路に SN のビット列から 1 ビットずつ順に入力することで算術演算を行う．AND ゲートで乗算を，MUX 回路で加算を，NOT ゲートで減算をそれぞれ実装できる [1].

例えば，乗算には AND (論理積) ゲートを用いる．入力 SN を a, b，出力 SN を c としたときの V_c は式 (2) のように表される．

$$V_c = P_c = P_a \times P_b = V_a \times V_b \tag{2}$$

図 1 では，$V_a = V_b = 0.5$ である a (= 00001111) と b (= 01010101) を掛けて $V_c = 0.25$ である c (= 00000101) を得ている．

2.2 ストカスティック数複製器

SC の演算は 2.1 章に示したように，簡単な論理回路により実装できる．ここで 2 乗器を設計することを考える．乗算は AND ゲートで実現できるが，図 2(a) のように，AND ゲートにビット列の等しい 2 つの SN を入力すると，入力 SN がそのまま出力される．つまり，正しく 2 乗できていないことがわかる．この問題を解決するためには，図 2(b) のように，DUP の位置で，入力された SN a と値が変わらずビット列が異なる新たな SN a' を生成し，a と a' を AND ゲートに入力する必要がある．

ある SN d について，$V_o = V_d$ となり，ビット列が入力 SN d とは異なる SN o を出力することを複製と呼ぶ．SN を複製する回路を複製器と呼ぶ．本稿では，精度の高い出力が得られる効率的な複製器を紹介し，再収斂パスを含む回路でもより理論値に近い値を得られることを示す．そのためには，複製器は以下の条件を満たす必要がある．

条件 (1) 入力 SN d と出力 SN o について，これらの値が等しく，すなわち $V_d = V_o$ となり，d と o のビット列が異なる．

条件 (2) 同じ SN d を複製器に入力しても，その出力 SN o は，複製のたびにビット列が異なる．この条件は図 6 のような非独立な再収斂を持つ回路で正しい演算結果を得るために必要である．

Vol.2018-ARC-230 No.36
Vol.2018-SLDM-183 No.36
Vol.2018-EMB-47 No.36
2018/3/8

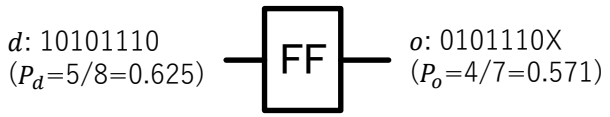

図 3　FF を用いた SN 複製器 [6].

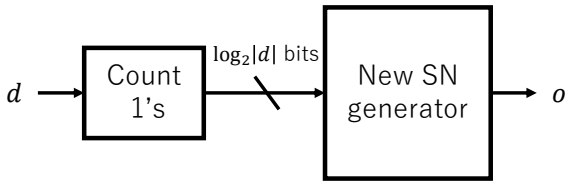

図 4　SN の再生成による複製器 [7].

2.3　既存のストカスティック数複製器 [6], [7]

今までに何種類かの複製器が提案されている [6], [7].

条件 (1) を満足するために，最も簡単なアプローチとして，[6] では，複製器として 1 つの 1 ビットフリップフロップ (FF) によるビットシフトを利用したものが提案されている (図 3)．ところが，FF を利用した SN 複製器 [6] は，ビット列が異なり値が等しい SN を生成することができるが，入力 SN に対して出力 SN が一意に決定されるため，[6] による複製器は条件 (2) を満たさない．

一方，[7] による複製器では，再度 SN を生成する．入力 SN d に含まれる 1 の数を数えることで P_d の値を算出し，確率 P_d で 1 が現れる新たなビット列を持つ SN o を再生成することで複製を実現する (図 4)．[7] による複製器では，まず入力 SN d に含まれる 1 の数を数え P_d を算出する．その後 $P_o = P_d$ となる SN o を出力する．SN o の生成方法は以下のとおりである．

(1) 乱数 $rand$ を生成する．
(2) $rand \leq P_d$ であれば 1 を，$rand > P_d$ であれば 0 を出力する．
(3) $|o| = |d|$ となるまで上記 1，2 を繰り返す．

[7] による複製器では，LFSR (線形帰還シフトレジスタ) によって乱数 $rand$ を生成している．新規の SN を再生成するため，条件 (1) と条件 (2) を満足できる．しかし，この複製器は $log_2|d|$ 個の FF を必要とし回路が大きく，複製器自体の面積や遅延が大きくなる．さらに，複製された SN を出力する前に入力 SN の 1 の数を数える必要があるため，[6] による複製器のように 1 クロックごとに SN を複製することができず，そのレイテンシはビット長以上になってしまう．そのため，[7] による複製器は実用的ではない．

以上のように，いかに面積や遅延，レイテンシ等を抑えつつ，条件 (1) と条件 (2) を満足する SN 複製器を設計するかが大きな課題となる．

3. 乱数によるビット並び替えに基づくストカスティック数複製器

[6] による複製器では入力される SN に対して出力 SN が常に一意に決定されるため，条件 (2) を満足せず，算術演算回路に組み込んだときに演算結果が正しく得られないという問題がある．この問題を解決するため，複製器の入力 SN に対して出力 SN が一意に決定されないようにする必要がある．そのため，複製器において入力に対して出力が一意に決まらないように，入力 SN とは独立なランダムビット列を導入し出力 SN を並び替えることを考える．複製器ごとにランダムビット列を導入し並び替えることで，算術演算で用いられる各々の複製器が出力する SN を独立にできる．この考えから，ごく少数のバッファを使用し出力するバッファを変え，入力 SN のビット列を並び替えたビット列を出力 SN として出力する複製器，RRR 複製器を紹介する．

図 5 に RRR 複製器 (Register based Rearrangement circuit using Random bit streams) を示す．もし $r_i = 0$ であれば FF_0 に格納されているビットを o_i に出力し，d_i を FF_0 に格納する．FF_1 には変更を加えない．同様に，もし $r_i = 1$ であれば FF_1 に格納されているビットを o_i に出力し，d_i が新しく FF_1 に格納される．FF_0 には変更を加えない．r を複製器ごとに変えることで，同じ入力 SN に対して異なる SN を出力でき，条件 (2) を満たす．RRR 複製器では，ランダムビット列による選択で失われるビットがなく，ほぼ全てのビットが出力され，入出力の SN の値にほとんど差がないことがわかる．ここで，d_i の入力のタイミングで FF_0 に d_j が格納されていれば 1 を返す 0/1 変数 $F^0_{j,i}$ と，FF_1 に d_j が格納されていれば 1 を返す 0/1 変数 $F^1_{j,i}$ はそれぞれ以下の式 (3)，(4) となる．

$$F^0_{j,i} = (1 - r_j) \times \prod_{k=j+1}^{i-1} r_k \tag{3}$$

$$F^1_{j,i} = r_j \times \prod_{k=j+1}^{i-1} (1 - r_k) \tag{4}$$

よって，FF_0，FF_1 の初期値をそれぞれ R_0，R_1 とすると，RRR 複製器の出力 SN の各ビット o_i は以下の式 (5) のようになる．

$$\begin{aligned} o_i = {} & R_0 \times (1 - r_i) \times \prod_{j=0}^{i-1} r_j \\ & + R_1 \times r_i \times \prod_{j=0}^{i-1} (1 - r_j) \\ & + (1 - r_i) \times \sum_{j=0}^{i-1} (d_j \times F^0_{j,i}) \\ & + r_i \times \sum_{j=0}^{i-1} (d_j \times F^1_{j,i}) \end{aligned} \tag{5}$$

ここで，$P_r = 1/2$ とし，i は十分大きいと考えると，o_i の

Vol.2018-ARC-230 No.36
Vol.2018-SLDM-183 No.36
Vol.2018-EMB-47 No.36
2018/3/8

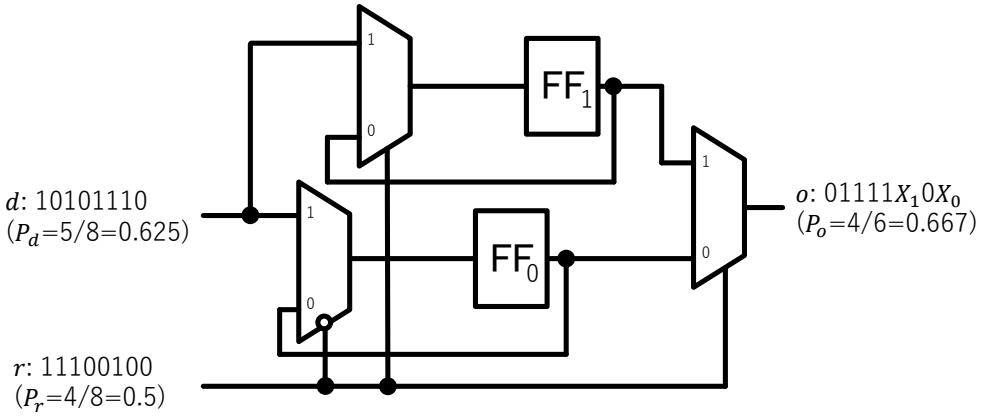

図 5　RRR 複製器.

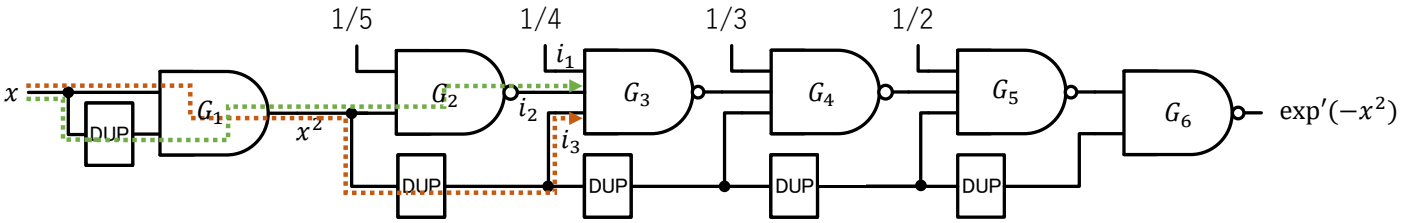

図 6　非独立な再収斂パスを含む，$\exp'(-x^2)$ を表現する回路.

期待値 E_i は以下の式 (6) のようになる．

$$E_i = \sum_{j=1}^{i} \frac{d_{i-j}}{2^j} = P_d = V_d \quad (6)$$

SN d=10101110, r=11100100 を RRR 回路に入力すると，SN o=01111$X_1$0X_0[*2]が出力される．

RRR 複製器では，ランダムビット列 r の i ビット目 r_i が 0 のとき入力ビットが FF_0 に格納され，FF_0 に格納されたビットは次に r のビットが 0 になった時に出力される．r_i が 1 のときは FF_0 の代わりに FF_1 が使用される．そのため，各 FF に格納されたビットは FF に残った最後のビットを除き，必ず出力される (r のビット列が最後に 0，あるいは最後に 1 になった際に入力されたビットは出力されず，各 FF に格納されたままになる)．FF に残った最後のビットの代わりに出力されるのが，もともと各 FF に格納されていたビットであるため，複製の際の最大誤差は 2 ビットになる．よって，入力 SN d に対する最大複製誤差は $2/|d|$ となる．これは，入力 SN のビット長が長いほど，誤差が小さくなることを示している．

4. 評価実験

4.1 複製器の精度比較

まず，再収斂を含むベンチマーク回路に各複製器を埋め込んだ際の出力をシミュレートすることで，精度を評価する．

[*2] X_0, X_1 はそれぞれ FF_0, FF_1 の初期値である．

4.1.1 評価指標

本実験では，平均二乗差 (Mean Square Error, MSE) を用いて精度を評価する．評価関数を f とした場合の MSE は以下の式 (7) のようになる．

$$MSE(f,x,n) = \frac{1}{n}\sum_{i=1}^{n}(f_{theory}(x) - f_{actual}(x,i))^2 \quad (7)$$

ここで，n は試行の回数を示す．$f_{theory}(x)$ は入力が x のときの関数 f の理論値であり，$f_{actual}(x,i)$ は x の値を持つ SN を入力した，i 番目の試行の際の回路の出力 SN の値である．

4.1.2 実験条件

以下の条件で本実験を行う．

- 実行環境: Python 3.6.3
- 試行回数: $n = 1000$
- 入力 SN のビット長: $|d| = 255,\ 4095$
- 入力 SN の値: $x = V_d = 0.0,\ 0.1,\ \ldots,\ 1.0$
- 複製器: [6] による複製器，[7] による複製器，FSR 複製器，RRR 複製器
- 評価関数: $f(x) = x^2$ (図 2(b))，$\exp(-x^2)$ (図 6)

4.1.3 評価関数

本実験では，2 乗器を他の回路と組み合わせて実装する．以下の算術演算回路を実装する．

(1) **2 乗器 (図 2(b)):**
図 2(a) に示すように，同じ SN x が 2 入力 AND ゲートに入力されると，入力 SN x が出力される．そのため，AND ゲートに入力する前に x を複製する必要がある．

Vol.2018-ARC-230 No.36
Vol.2018-SLDM-183 No.36
Vol.2018-EMB-47 No.36
2018/3/8

(2) **a) と $\exp(-x)$ の 5 次近似関数を組み合わせた $\exp'(-x^2)$ 関数 (図 6):**
ホーナー法 [8] に基づき，$\exp(-x)$ を構築することを考える．$\exp(-x)$ は以下の式 (8) で表される．

$$\begin{aligned}\exp(-x) &\approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} \\ &= 1 - x\left(1 - \frac{x}{2}\left(1 - \frac{x}{3}(1 - \frac{x}{4}(1 - \frac{x}{5}))\right)\right)\end{aligned} \tag{8}$$

a) と上の回路を組み合わせることで，図 6 の $\exp'(-x^2)$ 回路を構築できる．NAND ゲート G_3 に注目すると，i_1, i_2, i_3 の 3 つの入力がある．i_1 は 1/4 の値を持つ定数である．x から i_2 へのパスには，DUP を 1 つだけ通る経路がある (図 6 の緑色の点線)．また，x から i_3 へのパスにも，DUP を 1 つだけ通る経路がある (図 6 の橙色の点線)．この回路には，DUP を同数含む再収斂パスが存在する．NAND ゲート G_4–G_6 についても同様のことが言える．このような再収斂パスを非独立な再収斂パスと呼ぶ．この回路は非独立な再収斂パスを大く持つため，[6] による複製器を用いると，全ての NAND ゲートへの入力が非独立になる．

独立な再収斂パスを含む回路 a) では [6] による複製器を用いても算術誤差は発生しないが，非独立な再収斂パスを含む回路 b) では算術誤差が発生する．これは，[6] による複製器の出力が入力 SN にのみ依存し，複数のパスに含まれる DUP の数が等しければそれらの信号線が非独立になるためである．

4.1.4 実験結果

各複製器によるそれぞれの関数の MSE について，入力 SN のビット長が 255 の場合を表 1 に，4095 ビットの場合を表 2 に示す．これらの表の各値は，1000 回の試行の MSE (式 (7) で表される $MSE(f, x, 1000)$) を表す．「平均」の行は，各関数，複製器について，全ての x における MSE の平均を取ったものであり，その下の行は [6] による複製器の MSE の平均に対する割合を示す．最下段「総合平均」は全ての関数の MSE の平均を表す．

ビット長が 255 ビットのときの MSE を表 1 に示す．非独立な再収斂パスを含む $\exp'(-x^2)$ 回路では，RRR 複製器は MSE を ([7] による複製器とほぼ等しい) 約 64%削減する．独立な再収斂パスを含む x^2 回路では，[6] による複製器の MSE が最小になった．[7] による複製器は 1.25×10^{-4} となり，RRR 複製器よりも大きくなった．

ビット長が 4095 ビットのときの MSE を表 2 に示す．ビット長が長くなっても，RRR 複製器は [6] による複製器よりも MSE を抑えられた．$\exp'(-x^2)$ 回路では MSE を 94% 削減し， [7] による複製器とほぼ同じ値を得られた．独立な再収斂パスを含む x^2 回路では，RRR 複製器の MSE が最小になった．

表 1 各関数の出力の MSE ($|d|$=255).

関数	x	[6]	RRR
x^2	0.0	0	0
	0.1	3.18×10^{-5}	3.34×10^{-5}
	0.2	9.91×10^{-5}	9.36×10^{-5}
	0.3	1.80×10^{-4}	1.86×10^{-4}
	0.4	2.17×10^{-4}	2.18×10^{-4}
	0.5	2.48×10^{-4}	2.39×10^{-4}
	0.6	2.16×10^{-4}	2.45×10^{-4}
	0.7	1.74×10^{-4}	1.75×10^{-4}
	0.8	9.54×10^{-5}	1.04×10^{-4}
	0.9	4.18×10^{-5}	4.40×10^{-5}
	1.0	0	0
	平均	1.18×10^{-4} (100%)	1.22×10^{-4} (102%)
$\exp'(-x^2)$	0.0	0	0
	0.1	2.94×10^{-5}	2.95×10^{-5}
	0.2	9.37×10^{-5}	9.35×10^{-5}
	0.3	2.07×10^{-4}	1.85×10^{-4}
	0.4	4.69×10^{-4}	2.83×10^{-4}
	0.5	8.76×10^{-4}	2.93×10^{-4}
	0.6	1.23×10^{-3}	2.93×10^{-4}
	0.7	1.61×10^{-3}	3.43×10^{-4}
	0.8	1.41×10^{-3}	4.24×10^{-4}
	0.9	7.44×10^{-4}	3.34×10^{-4}
	1.0	2.34×10^{-4}	2.46×10^{-4}
	平均	6.27×10^{-4} (100%)	2.29×10^{-4} (36%)
	総合平均	3.73×10^{-4} (100%)	1.76×10^{-4} (47%)

ここで，$\exp'(-x^2)$ 回路では，[7] による複製器の MSE が最も小さくなったが，これは再生成によるためである．しかし，[7] による複製器では，255×5 または 4095×5 クロックサイクルのレイテンシがあり，実用的ではない．

4.2 複製器の論理合成

[6] による複製器，RRR 複製器の回路について，Design Compiler version D-2010.03-SP5 を用いて論理合成を行い，それぞれのクリティカルパス遅延と回路面積を調べた．3 手法の論理合成の結果を表 3，表 4 に記す． 各表の回路面積は NAND ゲート換算の値であり，クリティカルパス遅延の単位は [ns]，レイテンシの単位はクロックサイクルである．

ビット長が 255 ビットのときの結果を表 3 に示す．この表から，RRR 複製器は [6] による複製器のおよそ 9 倍の面積であるが，[7] による複製器と比べると 50%以上面積を削減できたことがわかる．クリティカルパス遅延はどの複

表 2　各関数の出力の MSE ($|d|$=4095).

関数	x	[6]	RRR
x^2	0.0	0	0
	0.1	2.05×10^{-6}	2.20×10^{-6}
	0.2	7.24×10^{-6}	7.01×10^{-6}
	0.3	1.29×10^{-5}	1.31×10^{-5}
	0.4	1.60×10^{-5}	1.59×10^{-5}
	0.5	1.96×10^{-5}	2.02×10^{-5}
	0.6	2.26×10^{-5}	2.01×10^{-5}
	0.7	1.92×10^{-5}	1.93×10^{-5}
	0.8	1.74×10^{-5}	1.67×10^{-5}
	0.9	1.74×10^{-5}	1.75×10^{-5}
	1.0	0	0
平均		1.22×10^{-5} (100%)	1.20×10^{-5} (98%)
$\exp'(-x^2)$	0.0	0	0
	0.1	1.93×10^{-6}	2.02×10^{-6}
	0.2	1.33×10^{-5}	6.44×10^{-6}
	0.3	7.52×10^{-5}	1.28×10^{-5}
	0.4	2.65×10^{-4}	2.59×10^{-5}
	0.5	6.08×10^{-4}	3.83×10^{-5}
	0.6	1.02×10^{-3}	4.83×10^{-5}
	0.7	1.24×10^{-3}	5.92×10^{-5}
	0.8	1.06×10^{-3}	5.55×10^{-5}
	0.9	4.53×10^{-4}	2.89×10^{-5}
	1.0	1.88×10^{-5}	1.88×10^{-5}
平均		4.32×10^{-4} (100%)	2.69×10^{-5} (6%)
総合平均		2.22×10^{-4} (100%)	1.95×10^{-5} (9%)

表 3　3 手法の論理合成結果 ($|d|$=255).

	面積	遅延 [ns]	レイテンシ [cycles]
[6]	5.75	0.39	0
[7]	114.5	0.50	255
RRR	51.25	0.49	0

表 4　3 手法の論理合成結果 ($|d|$=4095).

	面積	遅延 [ns]	レイテンシ [cycles]
[6]	5.75	0.39	0
[7]	199.75	0.50	4095
RRR	92.5	0.49	0

製器もほぼ変わらなかった．[6] による複製器と RRR 複製器は入力 SN の各ビットが入力される度にビットが出力されたが，[7] による複製器は入力 SN に含まれる 1 の数を数えてから出力するため，255 クロックサイクルかかった．

ビット長が 4095 ビットのときの結果を表 4 に示す．この表から，RRR 複製器は [6] による複製器のおよそ 16 倍の面積であるが，[7] による複製器と比べると 50%以上面積を削減できたことがわかる．クリティカルパス遅延はどの複製器もほぼ変わらなかった．[6] による複製器と RRR 複製器は入力 SN の各ビットが入力される度にビットが出力されたが，[7] による複製器は入力 SN に含まれる 1 の数を数えてから出力するため，4095 クロックサイクルかかった．

5. おわりに

本稿では，RRR という SN 複製器を紹介し，再収斂を含む合成関数に実装することで評価した．RRR 複製器は乱数の入力により，非独立な再収斂パスを含む関数で大幅な精度向上が見られた．具体的には，$\exp(-x^2)$ の回路で既存の複製器と比べて MSE を 64%–94%削減した．また，独立な再収斂パスを含む回路に実装しても，既存の複製器と MSE はほとんど変わらなかった．

今後は，本稿で提案した RRR 複製器をさらに別の回路にも実装・検証するとともに，これを更に拡張してより精度の高い複製器を設計したい．

謝辞

本研究開発は一部，総務省 SCOPE(受付番号 171503005)の委託を受けた．

参考文献

[1] B.R. Gains, “Stochastic computing,” in *Proc. Spring Joint Computer Conference*, pp.149–156, 1967.
[2] B.D. Brown and H.C. Card, “Stochastic neural computation I: Computational elements,” *IEEE Transactions on Computing*, vol.50, no.9, pp.891–905, 2001.
[3] V. Canals, A. Morro, A. Oliver, M.L. Alomar, and J.L. Rossell, “A new stochastic computing methodology for efficient neural network implementation,” *IEEE Transactions on Neural Networks and Learning Systems*, vol.27, no.3, pp.551–564, March 2016.
[4] P. Li and D.J. Lilja, “Using stochastic computing to implement digital image processing algorithms,” in *Proc. 29th International Conference on Computer Design (ICCD)*, pp.154–161, 2011.
[5] A. Alaghi, C. Li, and J.P. Hayes, “Stochastic circuits for real-time image-processing applications,” in *Proc. 50th Annual Design Automation Conference*, pp.136:1–136:6, DAC ’13, ACM, New York, NY, USA, 2013.
[6] K. Parhi and Y. Liu, “Computing arithmetic functions using stochastic logic by series expansion,” *IEEE Transactions on Emerging Topics in Computing*, pp.1–1, 2016.
[7] 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, pp.4883–4896, 2010.
[8] W.G. Horner, “A new method of solving numerical equations of all orders, by continuous approximation,” *Philosophical Transactions of the Royal Society of London*, vol.109, pp.309–355, 1819.