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

石川 遼太[1]　多和田 雅師[1]　柳澤 政生[1]　戸川 望[1]

概要：近年，ビット誤りに耐性を持ち，簡易な回路で算術演算を実現するストカスティック数による演算手法が注目されている．値の等しいストカスティック数が複数現われる演算回路では，ストカスティック数を複製する必要がある．ところが，複製により非独立なストカスティック数を生成すると，目的の演算結果が得られないため，いかに独立性の高いストカスティック数を複製するかが，最大の問題である．本稿では，非独立なストカスティック数の複製を防ぐため，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を提案する．提案するストカスティック数複製器では，乱数を導入し，乱数に応じてバッファされたストカスティック数のビット列を並び換えることで入力ストカスティック数と値の等しい，独立なストカスティック数を複製する．複数個のストカスティック数の複製器を持つ演算回路を実装・評価した結果，再収斂のある回路では，提案手法は既存手法と比べ出力の平均二乗誤差 (MSE) を 54%削減した．

1. はじめに

近年，回路の微細化により，ソフトエラーが深刻な脅威となっている．ソフトエラーに耐性を持つストカスティック数による演算手法，ストカスティックコンピューティング (Stochastic Computing, SC) が注目されている [1]．ストカスティック数 (Stochastic Number, SN) は，1967 年から研究されている，2 進数とは異なる実数表記である [2]．2003 年に LDPC デコーダの有用性が示された [3]．SN x には，0 と 1 からなるビット列が与えられ，その値は，x のビット列中の 1 の出現頻度で定義される．SN の算術演算は単純な論理回路により実装できる．AND ゲートで乗算を，MUX 回路で加算を，NOT ゲートで減算を実装できる．

SN の演算回路では，SN の複製が必要になる場合がある．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，FF による 1 ビットシフトレジスタである [4]．シフトレジスタを用いることで，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， 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，二つの複製器 FSR，RRR を提案する．FSR と RRR 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，MSE を既存手法 [4] と比べて 54%削減し，再収斂を持つ回路でも，複製器 RRR 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二点である．

- 乱数によるビット並び替えに基づく SN 複製器の提案．
- 複製器 RRR では，再収斂を持つ回路でも，既存手法に比べて MSE を 50%以上削減できる点．

2. SN と SN の複製器

2.1 SN

2.1.1 SN の定義

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 と表わす．SN x 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，$0 \leq V_x \leq 1$ が成り立つ．例えば，$x = 01010101$ のとき，その値は $V_x = 0.5$ となる．

この他にも SN の表現方法は複数あるが，本稿では，上記の表現のみを取り扱う．

2.1.2 SN の演算

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 SN の算術演算を行う．AND ゲートで乗算を，

[1] 早稲田大学

表 1　AND ゲートの真理値表

a_i	b_i	c_i	出現頻度
0	0	0	$(1-P_a)\times(1-P_b)$
0	1	0	$(1-P_a)\times P_b$
1	0	0	$P_a\times(1-P_b)$
1	1	1	$P_a\times P_b$

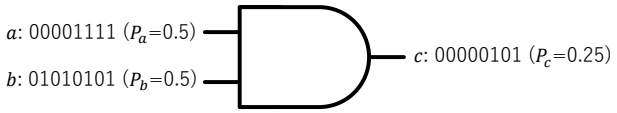

図 1　AND ゲートを用いた SN の乗算の例.

表 2　MUX 回路の真理値表

a_i	b_i	s_i	c_i	出現頻度
0	0	0	0	$(1-P_a)\times(1-P_b)\times(1-P_s)$
0	0	1	0	$(1-P_a)\times(1-P_b)\times P_s$
0	1	0	0	$(1-P_a)\times P_b\times(1-P_s)$
0	1	1	1	$(1-P_a)\times P_b\times P_s$
1	0	0	1	$P_a\times(1-P_b)\times(1-P_s)$
1	0	1	0	$P_a\times(1-P_b)\times P_s$
1	1	0	1	$P_a\times P_b\times(1-P_s)$
1	1	1	1	$P_a\times P_b\times P_s$

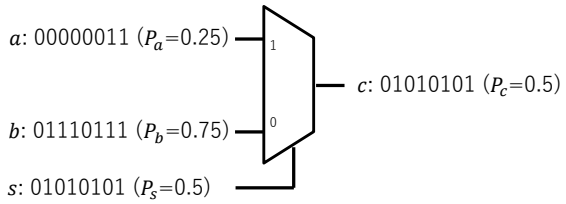

図 2　MUX 回路を用いた SN の加算の例.

MUX 回路で加算を，NOT ゲートで減算をそれぞれ実装できる．

乗算には AND (論理積) ゲートを用いる．SN x の任意のビットが 1 である確率は P_x，0 である確率は $(1-P_x)$ である．表 1 に入力 SN を a, b，出力 SN を c とした AND ゲートの真理値表と各状態の出現頻度を示す．表 1 から，V_c は式 (2) のように表わされる．

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

図 1 に AND ゲートを用いた SN の乗算の例を示す．図 1 では, $P_a = P_b = 0.5$ である $a\,(=00001111)$ と $b\,(=01010101)$ を掛けて $P_c = 0.25$ $(c = 00000101)$ を得ている．

加算には MUX (マルチプレクサ) 回路を用いる．表 2 に入力 SN を a, b，選択制御入力を s，出力 SN を c とした MUX 回路の真理値表と各状態の出現頻度を示す．表 2 から，V_c は式 (3) のように表わされる．

$$\begin{aligned} V_c = P_c =& (1-P_a)\times P_b\times P_s + P_a\times(1-P_b)\times(1-P_s) \\ &+ P_a\times P_b\times(1-P_s) + P_a\times P_b\times P_s \\ =& P_a\times(1-P_s) + P_b\times P_s \\ =& V_a\times(1-V_s) + V_b\times V_s \end{aligned} \tag{3}$$

図 2 に MUX 回路を用いた SN の加算の例を示す．図 2 では，$P_a = 0.25$ である $a\,(=00000011)$ と $P_b = 0.75$ である $b\,(=01110111)$ を入力とし，選択制御入力を $P_s = 0.5$ である $s\,(=01010101)$ とすることで，$P_c = 0.5$ $(c = 01010101)$ を得ている．

表 3　NOT ゲートの真理値表

a_i	b_i	出現頻度
0	1	$(1-P_a)$
1	0	P_a

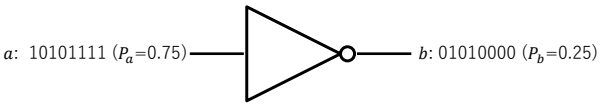

図 3　NOT ゲートを用いた SN の減算の例.

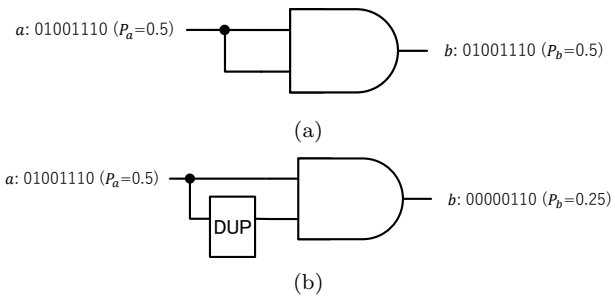

図 4　AND ゲートを用いた SN の二乗器の例.

減算には NOT (否定) ゲートを用いる．表 3 に入力 SN を a，出力 SN を b とした AND ゲートの真理値表と各状態の出現頻度を示す．表 3 から，V_b は式 (4) のように表わされる．

$$V_b = P_b = 1 - P_a = 1 - V_a \tag{4}$$

図 3 に AND ゲートを用いた SN の乗算の例を示す．図 3 では，$P_a = P_b = 0.5$ である $a = 10101111 (V_a = 0.75)$ を入力し，$b = 01010000 (V_b = 0.25)$ を得ている．

2.2　SN の複製器

SN の演算は 2.1.2 項に示したように，論理回路により簡単に実装できる．ここ で二乗器を設計することを考える．前述の通り，乗算は AND ゲートで実現できるが，図 4(a) のように，AND ゲートにビット列の等しい 2 つの SN を入力すると誤った演算結果が出力される．これを解決するためには，図 4(b) のように，DUP の 位置で，入力された SN x と値が変わらずビット列が異なる新たな SN y を複製し，x と y を AND ゲートを入力すれば良い．

今，ある SN d について，$V_o = V_d$ となり，ビット列が d とは異なる SN o を作ることを複製と呼ぶ．SN を複製する回路を複製器と呼ぶ．複製器は，まず以下の条件を満足する必要がある．

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，即ち $V_d = V_o$ となり，d と o のビット列が異なる．

条件 (1) を満足するために，最も簡単なアプローチとして [4] では，複製器として 1 つのフリップフロップ (FF) を利用したものが提案されている (図 6)．ところが，FF を利用した SN 複製器は，ビット列が異なり値が等しい SN

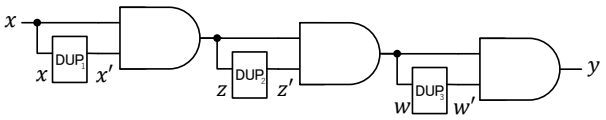

図 5 SN の 8 乗器.

を生成することができるが，回路中に再収斂パスが存在すると正しい演算結果が得られない．

例えば，図 5 のような SN を 8 乗する回路を考える．図 5 は 2 乗器を 3 段とする構成をとることで 8 乗器を構成している．この際，各 2 乗器の入力は複製器 (DUP_1〜DUP_3) を利用することで，SN が複製されている．ここで，複製器として FF を利用した SN 複製器を用いることを仮定しよう．8 乗器の出力 SN y の i ビット目に着 目すると，

$$z_i = x_i \wedge x_i' = x_i \wedge x_{i-1} \tag{5}$$

$$\begin{aligned} w_i &= z_i \wedge z_i' = [x_i \wedge x_{i-1}] \wedge [x_{i-1} \wedge x_{i-2}] \\ &= x_i \wedge x_{i-1} \wedge x_{i-2} \end{aligned} \tag{6}$$

$$\begin{aligned} y_i &= w_i \wedge w_i' = [x_i \wedge x_{i-1} \wedge x_{i-2}] \wedge [x_{i-1} \wedge x_{i-2} \wedge x_{i-3}] \\ &= x_i \wedge x_{i-1} \wedge x_{i-2} \wedge x_{i-3} \end{aligned} \tag{7}$$

となる．つまりビット列全体として SN y の値 V_y は，

$$V_y = P_y = {P_x}^4 = {V_x}^4 \tag{8}$$

となり，入力 SN が表す値 V_x の 8 乗を計算することはできず，V_x が 4 乗された値が出力されることになる．FF を利用した SN 複製器は，入力として与えられた SN d を複製する際，必ず同じビット列の SN o が生成されるが，回路中に再収斂パスがあると，結果的に同じ演算が複数回実行されるためである．

つまり，SN の複製器には条件 (1) に加えて，以下の条件 (2) が必要となる．

条件 (2) 同じ SN d を複製器に入力しても，その出力 SN o は，複製のたびに出力のビット列が異なる．

2.3 既存の SN 複製器

SN の複製器として，[4] と [5] のアプローチが知られている [4] のアプローチは，前述の通り，上記条件 (1) を満足するが，条件 (2) を満足しない．一方，[5] のアプローチは，入力 SN d に含まれる 1 の数を数えることで V_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ものである (図 7)．新規の SN を再生成するために条件 (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，[4] のように 1 クロックごとに SN を複製することができない．

以上のように，いかに小コストで，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．

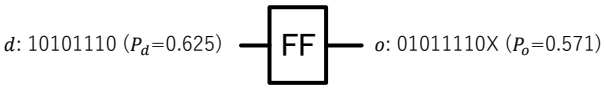

図 6 FF を用いた SN の複製器 [4].

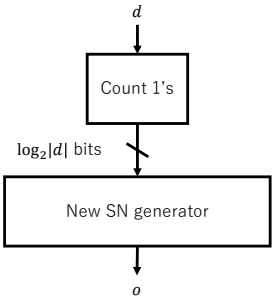

図 7 [5] で提案された複製器.

3. 乱数によるビット並び替えに基づく SN 複製器

[4] では入力される SN に対して出力 SN が常に一意となるため，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 が一意に決まらないようにする必要がある．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 とは独立な乱数列を導入し出力 SN を並び替える．複製器ごとに乱数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．

本稿では，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を提案する．提案複製器では算術演算回路に複数実装しても算術演算結果に影響を与えにくい．

o の i 番目のビット o_i について，乱数 SN の i 番目のビット r_i に応じて d のどのビットを出力する決める複製器を二種類提案する．3.1 節では FSR 回路を，3.2 節では RRR 回路を提案する．

3.1 複製器 FSR

図 8 に一つ目の提案手法である複製器 FSR(Flip-flop Selecting circuit using Random bit streams) を示す．複製の度に異なるビットパターンを出力するために，1 ビット遅れと 2 ビット遅れのビット列からランダムにビットを選択し，出力することを考える．この回路では，FF_0 と FF_1 の二つの FF と MUX を用いて，1 ビット遅れと 2 ビット遅れのビット列を作り出し，ビット毎にどちらのビット列から採用するかを乱数で決める．図 8 の FF_1 の出力が 1 ビット遅れのビット列を表わし，FF_0 が 2 ビット遅れのビット列を表わす．乱数 r を用いて o_i に d_{i-1} と d_{i-2} のうちどちらを出力するかを決める．これによって，[4] よりも独立性が増す．複製器 FSR の入出力 SN の関係は以下の式 (9) で表わされる．

$$o_i = d_{i-1} \times r_i + d_{i-2} \times (1 - r_i) \tag{9}$$

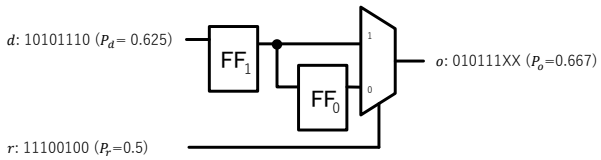

図 8 複製器 FSR.

ここで，$V_r = 1/2$ である乱数 SN r を入力することを考えると，o_i の期待値 E_i は以下のようになる．

$$E_i = \frac{1}{2}(d_{i-1} + d_{i-2}) = P_d = V_d \tag{10}$$

d=10101110, r=11100100 を FSR 回路に入力すると，o=010111XX となる．

[4] は常に一つ前のビットのみを出力していたのに対し，複製器 FSR では二つ前のビットも出力するため，独立性が上がる．

図 5 の 8 乗器を考える．複製器 DUP_1 において，入力 SN x の i ビット目 (x_i) に注目する．DUP_1 を複製器 FSR によって実現すると，D_1 サイクル ($D_1 = 1$ あるいは $D_1 = 2$) 遅れて出力 x' が生成される．すなわち，$x'_i = x_{i-D_1}$ となる．同様に，DUP_2, DUP_3 を複製器 FSR によって実現すると，それぞれ D_2 サイクルあるいは D_3 サイクル遅れて出力が生成され，$z'_i = z_{i-D_2}, w'_i = w_{i-D_3}$ となる．このとき，図 5 の 8 乗器全体の出力 y の i ビット目 y_i に対して，以下の式が成立する．

$$\begin{aligned} y_i = x_i \wedge x_{i-D_1} \wedge x_{i-D_2} \wedge x_{i-D_3} \wedge x_{i-D_1-D_2} \\ \wedge x_{i-D_2-D_3} \wedge x_{i-D_3-D_1} \wedge x_{i-D_1-D_2-D_3} \end{aligned} \tag{11}$$

式 (11) から，D_1, D_2, D_3 を入れ替えても y_i は変わらないため，本稿では $1 \le D_1 \le D_2 \le D_3$ であるとする．式 (11) において，複製器 FSR の乱数 SN r 全てが $V_r = 1/2$ であるとすると，D_1, D_2, D_3 それぞれ 1/2 の確率で 1，1/2 の確率で 2 となる．表 4 にとりうる D_1, D_2, D_3 と，その出現確率，出力を示す．よって，V_y は以下のようになる．

$$\begin{aligned} V_y = P_y &= \frac{3}{8}P_x^6 + \frac{3}{8}P_x^5 + \frac{1}{4}P_x^4 \\ &= \frac{3}{8}V_x^6 + \frac{3}{8}V_x^5 + \frac{1}{4}V_x^4 \end{aligned} \tag{12}$$

式 (8)，式 (14) から，[4] で構成した 8 乗器よりも FSR で構成した 8 乗器の方が理想的な 8 乗器により近づくことがわかる．

3.2 複製器 RRR

FSR よりさらに過去のビットを遡って出力するために，複製器 FSR に MUX をさらに二つ加える．図 9 に二つ目の提案手法である複製器 RRR(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．複製器 RRR では，乱数による選択で失われるビットがなく，全てのビットが出力され，入出力の SN の値に差がないことがわかる．

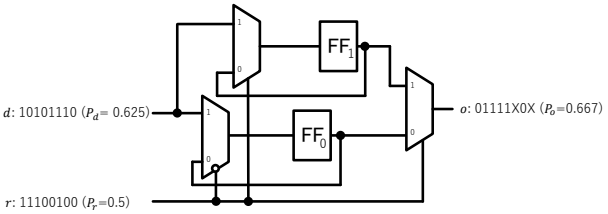

図 9 複製器 RRR.

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 を表す 0–1 変数 $F^0_{j,i}$ と，FF_1 に d_n が格納されていれば 1 を返す 0–1 変数 $F^1_{j,i}$ は以下の式 (13)，(14) となる．

$$F^0_{j,i} = (1 - r_j) \times \prod_{k=j+1}^{i} r_k \tag{13}$$

$$F^1_{j,i} = r_j \times \prod_{k=j+1}^{i} (1 - r_k) \tag{14}$$

よって，FF_0，FF_1 の初期値をそれぞれ R_0，R_1 とすると，複製器 RRR の出力 SN の各ビット o_i は以下の式 (15) のようになる．

$$\begin{aligned} o_i = &R_0 \times (1 - r_i) \times \prod_{j=0}^{i-1} r_j \\ &+ R_1 \times r_i \times \prod_{j=0}^{i-1} (1 - r_j) \\ &+ (1 - r_i) \times \sum_{j=0}^{i} (d_j \times F^0_{j,i}) \\ &+ r_i \times \sum_{j=0}^{i} (d_j \times F^1_{j,i}) \end{aligned} \tag{15}$$

ここで，$P_r = 1/2$ とし，i は十分大きいと考えると，o_i の期待値 E_i は以下の式 (16) のようになる．

$$E_i = \sum_{j=1}^{i} \frac{d_{i-j}}{2^j} = P_d = V_d \tag{16}$$

d=10101110, r=11100100 を RRR 回路に入力すると，o=01111X0X となる．

この手法では，独立性の高い SN を複製できる．図 5 の 8 乗器を考える．式 (15) から，n ビット前のビットが出力される確率は $1/2^n$ $(n \ge 1)$ である．これを利用して，D_1, D_2, D_3 について以下の五つの状態になる確率を算出する．

状態 (1) $D_1 = D_2 = D_3$
状態 (2) $D_1 = D_2 = D_3/2$
状態 (3) $D_1 = D_2 \neq D_3/2$ または $D_2 = D_3$
状態 (4) $D_1 \neq D_2$ かつ $D_1 + D_2 = D_3$
状態 (5) 上記以外

表 5 に各状態になる確率と，その状態が表わす関数を示す．

表 4　複製器 FSR の D_1〜D_3 の出現率とそれぞれの場合の 8 乗器の出力.

D_1	D_2	D_3	出現率	y_i
1	1	1	1/8	$x_i \wedge x_{i-1} \wedge x_{i-2} \wedge x_{i-3}$
1	1	2	3/8	$x_i \wedge x_{i-1} \wedge x_{i-2} \wedge x_{i-3} \wedge x_{i-4}$
1	2	2	3/8	$x_i \wedge x_{i-1} \wedge x_{i-2} \wedge x_{i-3} \wedge x_{i-4} \wedge x_{i-5}$
2	2	2	1/8	$x_i \wedge x_{i-2} \wedge x_{i-4} \wedge x_{i-6}$

表 5　8 乗器の各状態の出現確率とその表現関数.

状態	各状態になる確率	y_i
(1)	1/7	$x_n \wedge x_{n-D_1} \wedge x_{n-2D_1} \wedge x_{n-3D_1}$
(2)	1/5	$x_n \wedge x_{n-D_1} \wedge x_{n-2D_1} \wedge x_{n-3D_1} \wedge x_{n-4D_1}$
(3)	13/35	$x_n \wedge x_{n-D_1} \wedge x_{n-2D_1} \wedge x_{n-D_3} \wedge x_{n-D_1-D_3} \wedge x_{n-2D_1-D_3}$ $x_n \wedge x_{n-D_1} \wedge x_{n-D_2} \wedge x_{n-D_1-D_2} \wedge x_{n-2D_2} \wedge x_{n-2D_1-D_2}$
(4)	2/15	$x_n \wedge x_{n-D_1} \wedge x_{n-D_2} \wedge x_{n-D_3} \wedge x_{n-D_1-D_3} \wedge x_{n-D_2-D_3} \wedge x_{n-D_1-D_2-D_3}$
(5)	16/105	$x_n \wedge x_{n-D_1} \wedge x_{n-D_2} \wedge x_{n-D_3} \wedge x_{n-D_1-D_2} \wedge x_{n-D_1-D_3} \wedge x_{n-D_2-D_3} \wedge x_{n-D_1-D_2-D_3}$

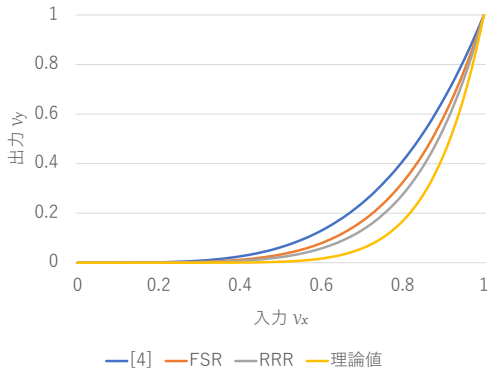

図 10　図 5 の 8 乗器 $V_y = V_x^8$ を各手法で実装した際の出力.

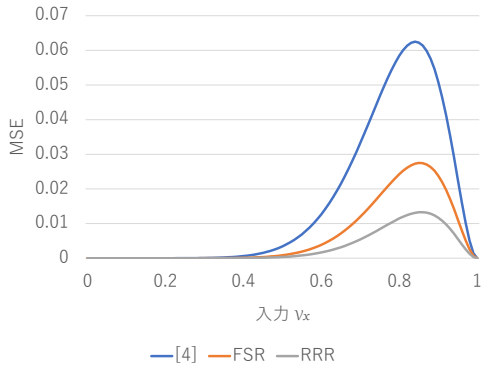

図 11　図 5 の 8 乗器 $V_y = V_x^8$ を各手法で実装した際の MSE.

この表から，8 乗器は実際には以下の関数を計算していることがわかる．

$$V_y = \frac{16}{105}V_x^8 + \frac{2}{15}V_x^7 + \frac{13}{35}V_x^6 + \frac{1}{5}V_x^5 + \frac{1}{7}V_x^4 \qquad (17)$$

各手法で図 5 を実装した際の出力を図 10 に，理論値に対する MSE を図 11 に示す．このグラフから，複製器 RRR では大幅に MSE を削減できていることがわかる．また，ほとんど全てのビットが出力され，出力 SN と入力 SN との値の差も小さく，演算の誤差にはつながらない．

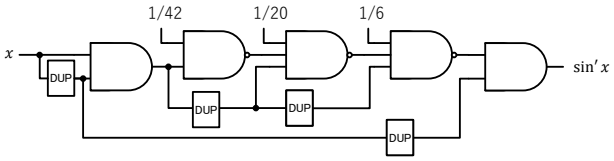

図 12　実験回路 (1).

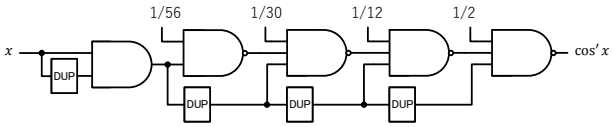

図 13　実験回路 (2).

4. 実験

4.1 精度の比較

4.1.1 実験回路

本実験では，多項式近似関数を少ない乗算回数で表現するホーナー法に基づく sin 関数の 7 次近似関数 sin′，cos 関数の 8 次近似関数 cos′ を実装する [4], [6]．図 12，13 に実験回路を，式 (18)，(19) に実験関数を示す．

$$\begin{aligned}\sin x &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \\ &= x(1 - \frac{x^2}{6}(1 - \frac{x^2}{20}(1 - \frac{x^2}{42}))) \\ &= \sin' x \end{aligned} \qquad (18)$$

$$\begin{aligned}\cos x &\approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \\ &= 1 - \frac{x^2}{2}(1 - \frac{x^2}{12}(1 - \frac{x^2}{30}(1 - \frac{x^2}{56}))) \\ &= \cos' x \end{aligned} \qquad (19)$$

これらの回路の DUP は複製器を表わす．

4.1.2 実験

[4]，[5]，複製器 FSR，複製器 RRR の四つの複製器で実験回路 (1)，(2) を実装し，出力 SN をシミュレーションで測定した．ただし，[5] で提案された複製器では，LFSR を乱数生成器として用いられているが，本シミュレーション

表 6　各複製器による評価関数の出力と理論値との MSE.

	x	[4]	[5]	FSR (Proposed)	RRR (Proposed)
sin'	0.0	0	0	0	3.47×10^{-6}
	0.1	8.62×10^{-5}	2.68×10^{-4}	1.63×10^{-4}	8.82×10^{-5}
	0.2	1.68×10^{-4}	4.58×10^{-4}	2.85×10^{-4}	1.48×10^{-4}
	0.3	2.83×10^{-4}	5.82×10^{-4}	3.91×10^{-4}	1.93×10^{-4}
	0.4	4.39×10^{-4}	6.55×10^{-4}	4.68×10^{-4}	2.22×10^{-4}
	0.5	6.00×10^{-4}	6.45×10^{-4}	5.19×10^{-4}	2.39×10^{-4}
	0.6	7.12×10^{-4}	6.10×10^{-4}	5.44×10^{-4}	2.39×10^{-4}
	0.7	7.02×10^{-4}	4.92×10^{-4}	4.90×10^{-4}	2.24×10^{-4}
	0.8	5.33×10^{-4}	3.75×10^{-4}	3.85×10^{-4}	1.86×10^{-4}
	0.9	2.87×10^{-4}	2.54×10^{-4}	2.45×10^{-4}	1.48×10^{-4}
	1.0	1.35×10^{-4}	1.33×10^{-4}	1.33×10^{-4}	1.28×10^{-4}
Average		3.59×10^{-4} (100%)	4.06×10^{-4} (113%)	3.29×10^{-4} (91%)	1.65×10^{-4} (46%)
cos'	0.0	9.56×10^{-7}	9.56×10^{-7}	9.56×10^{-7}	9.3×10^{-7}
	0.1	6.23×10^{-6}	1.36×10^{-5}	9.12×10^{-6}	6.17×10^{-6}
	0.2	2.32×10^{-5}	5.29×10^{-5}	3.39×10^{-5}	2.26×10^{-5}
	0.3	5.25×10^{-5}	11.5×10^{-5}	7.28×10^{-5}	5.04×10^{-5}
	0.4	9.32×10^{-5}	1.91×10^{-4}	1.25×10^{-4}	8.73×10^{-5}
	0.5	1.40×10^{-4}	2.71×10^{-4}	1.81×10^{-4}	1.31×10^{-4}
	0.6	1.95×10^{-4}	3.40×10^{-4}	2.45×10^{-4}	1.71×10^{-4}
	0.7	2.36×10^{-4}	4.03×10^{-4}	2.95×10^{-4}	2.17×10^{-4}
	0.8	2.63×10^{-5}	4.07×10^{-5}	3.04×10^{-5}	2.41×10^{-5}
	0.9	2.61×10^{-4}	3.48×10^{-4}	2.99×10^{-4}	2.59×10^{-4}
	1.0	2.45×10^{-4}	2.50×10^{-4}	2.42×10^{-4}	2.44×10^{-4}
Average		1.38×10^{-4} (100%)	2.18×10^{-4} (157%)	1.64×10^{-4} (119%)	1.30×10^{-4} (94%)

表 7　3 手法の論理合成結果.

	回路面積 (NANDs)	遅延 [ns]
[4]	5.75	0.39
FSR	17	0.39
RRR	20.5	0.39

では，ソフトウェアにおける乱数生成器を用いる．

本実験では，平均二乗差 (Mean Square Error, MSE) を用いて複製器を組み込んで実験回路の近似精度を評価する．評価関数を f とした場合の MSE は以下のようになる．

$$MSE(f,x,n) = \frac{1}{n}\sum_{i=1}^{n}(f_{theory}(x) - f_{actual}(x,i))^2 \quad (20)$$

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 i 番目の試行の際の回路の出力 SN の値である．

本実験では，$n = 10000$ とし，用いる入力 SN のビット長は 16,384 ビットであり，0.0 から 1.0 までの値を入力する．評価関数は，式 (18)，(19) で表される $\sin' V_x$，$\cos' V_x$ を用いる．

4.1.3　実験結果

各複製器によるそれぞれの関数の MSE を表 6 に示す．この表から，複製器 RRR は，$\sin'$ 回路では，MSE を平均 54%削減した．複製器 FSR では，平均 6%削減できた．$\cos'$ 回路についても，複製器 RRR は MSE を 6%削減した．

4.2　各回路の論理合成

[4] で提案された複製器，複製器 FSR，複製器 RRR 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，それぞれの遅延と回路面積を調べる．3 手法の論理合成の結果を表 7 に記す．

表 8　手法間の性能の比較.

回路	MSE	V_d と V_o の誤差		回路面積
		Min	Max	
[4]	中	1 ビット	1 ビット	小
FSR	中	1 ビット	$\|d\|/2$ ビット	中
RRR	小	1 ビット	2 ビット	大

5.　手法間の比較

表 8 に 3 手法間の性能を比較を示す．MSE が大きいほど，再収斂への耐性がないことを示している．生成誤差は，入力の SN のビット列のうち，出力されないビット列の数である．FF 回路は最後のビットが出力されず，RRR 回路では最後にバッファに残っていた 1-2 ビット (最後の 1 ビットを含む) が出力されない．FSR 回路は r が常に 1 または 0 のときに生成誤差が最小となり，r が 101010... など特定のパターンの中に 10 が交互に入力されると，1 が入力される 1 ビット前の d のビットが出力されず，最大で $|d|/2$ のビットが出力されない．

RRR 回路が最も MSE が小さい回路である．RRR 回路ではバッファに残ったビット以外の全てのビットを採用し，独立性の高い SN が複製できている．一方で，FSR 回路では最大でビット列の半分が出力されず，既に出力されたビットが再度出力されるため，MSE が大きくなっている．

6.　おわりに

本稿では，SN の複製器として，二つの複製器 FSR と RRR を提案した．提案した複製器は既存の複製器と比べて MSE を最大 54%削減することに成功した．今後，本稿で提案した複製器を実際の SC 回路に実装していきたい．

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] S. Iizuka, M. Mizuno, D. Kuroda, M. Hashimoto, and T. Onoye, “Stochastic error rate estimation for adaptive speed control with field delay testing,” in *Proc. ICCAD 2013*, pp.107–114, 2013.

[2] B.R. Gains, “Stochastic computing,” in *Proc. Spring Joint Computer Conference*, pp.149–156, 1967.

[3] V.C. Gaudet and A.C. Rapley, “Iterative decoding using stochastic computation,” *Electronics Letters*, vol.39, no.3, pp.299–301, 2003.

[4] 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.

[5] 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.

[6] 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.