Hardware Optimization of Stochastic Computing

ストカスティックコンピューティングのハードウェア最適化

February, 2022

Ryota ISHIKAWA
石川　遼太

Hardware Optimization of Stochastic Computing

ストカスティックコンピューティングのハードウェア最適化

February, 2022

Waseda University Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering, Research on Information System Design

Ryota ISHIKAWA
石川　遼太

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Backgrounds

The number of hardware devices used in our everyday lives has been growing rapidly. For example, most people in the whole world own smartphones, very large amount of money are deposit in bank systems, and even crucial systems such as automobiles and planes are controlled by computer systems. Not only the amount, developing hardware devices on ones own also has been realistic. For example, Xilinx Inc.[3] has developed low-cost and highly integrated System on Chips (SoC) products which integrates software programmability of an ARM-based processor with hardware programmability of an FPGA (Field-Programmable Gate Array) core[1]. Therefore, developing high-functioning hardware devices became more easily than ever.

When implementing applications on hardware, not so much precision is required in many areas. For example, in areas like image processing and machine learning, small errors will cause insensible errors in images and no errors in its output in most cases, respectively. Instead of high precision with larger circuit size, smaller circuits are desired due to the increasing amount of data and more complex operations. For its small-sized implementation, compared with conventional computation, approximate computing[5] is attracting interest.

Stochastic computing (SC) [6] is one of the approximate computing methods, aiming to miniaturize circuits by allowing errors in calculation results. SC repeats bit-level logical operations to achieve stochastic arithmetic operations using stochastic numbers (SNs). SNs are bit streams whose values are defined by the appearance rates of 1's in their bit

[1]The author and his colleagues have developed PYNQs and Raspberry Pi[4] based autonomous vehicle for the "Design Competition" as in ⟨5⟩ and ZYNQ-based puzzle solver systems for the "Algorithm Design Contest" as in ⟨4⟩, ⟨13⟩, ⟨18⟩, and ⟨22⟩, and won the prizes as in ⟨24⟩, ⟨26⟩, and ⟨27⟩.

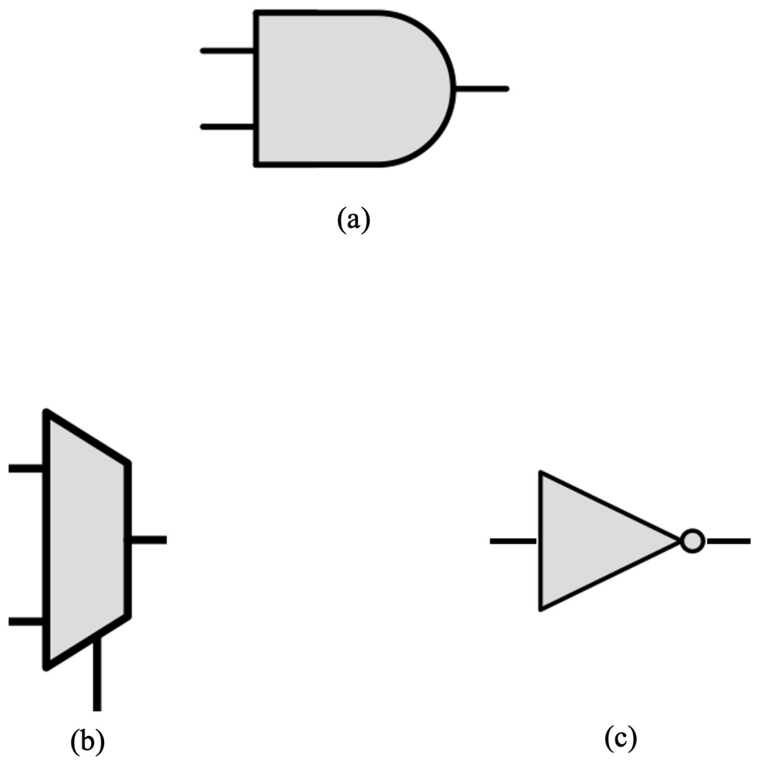


Figure 1.1: Components of SC. (a) Multiplication. (b) Scaled addition. (c) Inversion.

streams. In SC, SNs' bit streams are input into simple logic circuits sequentially for arithmetic calculations. For example, an AND gate, a MUX circuit, and a NOT gate implement multiplication, weighted addition, and inversion, respectively, as in Figure 1.1 [6]. Hence, SC enables smaller-sized implementation compared with binary computing and was applied to many hardware applications such as neural networks [7, 8, 9, 10, 11] and image processing [12, 13, 14, 15, 16].

Other than its small-sized hardware implementation, SC has many characteristics. For example, SC has noise resiliency. This is because all the bits of the bit streams of SNs have the same weight. This means that when one bit of an SN is mistakenly flopped, the value of an SN has small errors. On the other hand, when one bit of a binary number is flopped, the value can have error of half the range in maximum. With this characteristic, SNs have been applied to reliable systems [17, 18, 19, 20, 21, 22, 23, 24][2]. SC also

[2]The author and his colleagues have developed error correction systems that can correct values in very high error rates utilizing SNs as in ⟨6⟩, ⟨7⟩, ⟨11⟩, ⟨15⟩, and ⟨16⟩.

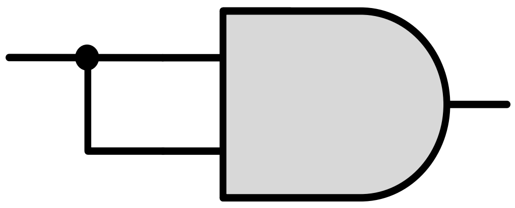


Figure 1.2: Necessity of SN generation. (An incorrect squarer)

has the flexibility over accuracy and SNs are recursive[3], i.e., the SC-based circuits can change their accuracies by only changing the length of the inputs, without changing their circuits.

Even with these merits, SC is yet to be used in practical situations. One reason for this is: SN-to-binary converter is required for calculation but its hardware cost is large.

Then, why is SN-to-binary conversion required? There are two main reasons:

1. To guarantee the independency of the bit streams of SNs.

2. To implement steep or discontinuous functions such as step function.

The root cause of reason 1 is the dependency of bit streams of SNs. To obtain the correct calculation result, the bit streams of the input SNs must be independent. For example, by using a multiplier as in Figure 1.1(a), a squarer should be expressed in Figure 1.2 if the two input SNs are independent. However, in this example, this squarer outputs the input SN as output SN, meaning that the correct square value cannot be obtained. Thus independency of the bit streams of SNs are required, i.e., generating new SNs are required. Reason 2 is especially required to implement activation functions, such as the Rectified Linear Unit (ReLU) function, the step function, and their composite functions. However, without re-generation of SNs, no steep or discontinuous functions were proposed alone. For example, in [25], all functions, which is representable by Bernstein polynomials[26] with coefficient in-between the range, are shown to be implemented in SC.

The problem of SN-to-binary conversion is its large hardware cost. In terms of circuit area, SN-to-binary conversion, when 255-bit SNs are generated, costs more than 50 times larger compared with an SN multiplier. Not only the case of circuit area, SN-to-binary conversion also requires large latency. If n SN-to-binary converters are used in series as in Figure 1.3, $255 \times n$ clock cycles are required after the first input bit to output the first output bit. In addition, flexibility over accuracy is lost due to this conversion.

[3]The author and his colleagues have developed an image format apply SN's recursivity to reduce circuit area for resolution conversion as in ⟨3⟩ and ⟨14⟩.

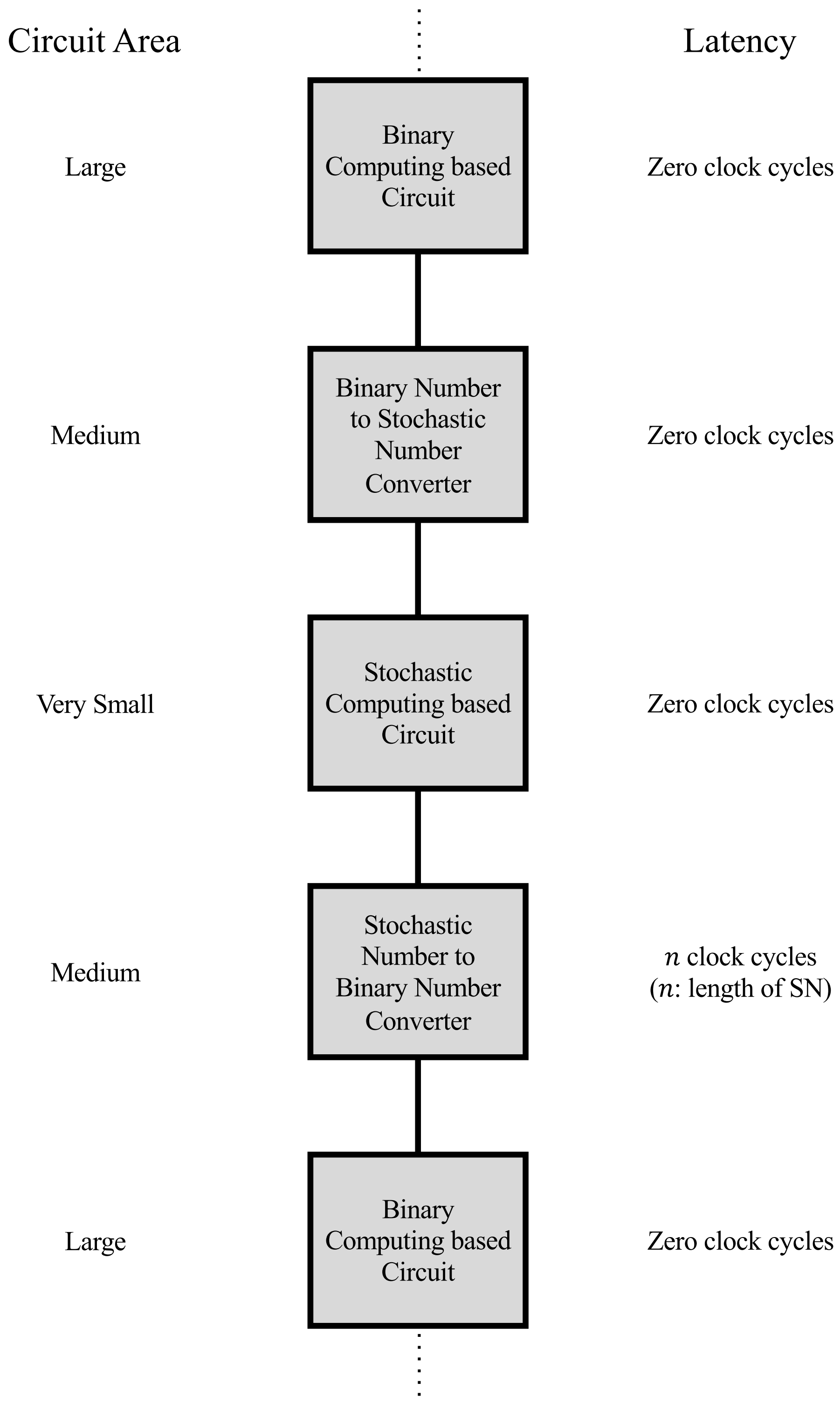

Circuit Area
Latency
Large
Binary Computing based Circuit
Zero clock cycles
Medium
Binary Number to Stochastic Number Converter
Zero clock cycles
Very Small
Stochastic Computing based Circuit
Zero clock cycles
Medium
Stochastic Number to Binary Number Converter
n clock cycles (n: length of SN)
Large
Binary Computing based Circuit
Zero clock cycles

Figure 1.3: Hardware costs of stochastic circuits with SN generators.

There are researches trying to optimize the use of SN generators[27, 28, 29, 30, 31]. However, this dissertation these problems in a different way. **This dissertation aims to omit the use of SN generators as much as possible**, proposing two types of circuits: SN duplicators; and a step function circuit with its applications.

In SC, the bit streams of the input SNs must be independent to obtain the correct calculation result. In many functions, two or more identical values are input. In these cases, duplicating SNs, or re-generating SNs with the equivalent value and independent bit stream, is required. In a conventional method, a flip-flop is used to delaying one bit of the input SN, creating a different, 1-bit delayed bit stream[1]. Since the output SN has the (nearly) same value and different bit stream compared withe the input SN, this circuit can be said to have duplicated an SN. However, the output SN's bit stream is always the same if the same input SN is given. In [1], circuits using FFs as SN duplicators have been proposed. However, these circuits had errors since FFs output the same SN when the same SNs are given. As mentioned above, if bit streams of the duplicated SNs are dependent on each other, their arithmetic operation results become inaccurate. Here, we consider a randomized bit stream being introduced to re-arrange the bits stored in flip-flops. This dissertation proposes FSR (Flip-flop Selecting circuit using a Random bit stream) and RRR (Register based Re-arrangement circuit using a Random bit stream) duplicators which have the equivalent values but have independent bit streams, by effectively utilizing bit re-arrangement using randomized bit streams. Also, we extend the proposed RRR duplicator, enabling to change the accuracy of the circuit itself. The proposing 2^nRRR duplicator outputs different SNs every time and are all independent of each other. The 2^nRRR duplicator can be theoretically proved to flexibly change the accuracies of the arithmetic circuits. Also from the experimental evaluation results, we clarify that the 2^nRRR duplicator enables accuracy-flexible circuits. Experimental evaluation results demonstrate that the proposed SN duplicator obtains more accurate results compared with a conventional SN duplicator.

With the rise of neural network and image processing, implementation of activation functions are becoming popular. However, steep functions and discontinuous functions are examples of arithmetic functions that are difficult to implement in SC due to the nature of SNs. Of the activation functions, the Rectified Linear Unit (ReLU) function, the step function, and their composite functions appear as steep functions and discontinuous functions. Implementation of steep functions and discontinuous functions is indispensable for the practical application of SC. However, when implementing arithmetic operations in SC, there is a constraint that the original function must be differentiable. There has been some steep functions that has been proposed, but they can only be used in specific situations, and cannot be used alone. To solve the problems of the conventional methods, we firstly propose hardware implementation of step function using SNs. The proposing circuit of utilizes flip-flops and an adder to perform as step function uniquely calculating the stored bits in the flip-flops. The proposed step function circuit can be theoretically proved to perform as a step function. We confirm that the proposing circuit behaves as a step function through experimental evaluations. Also as an application, steep functions or discontinuous functions can be realized by applying the discontinuity of the step function. As a steep function, this dissertation also proposes hardware implementation of absolute function and discontinuous function, by synthesizing an arbitrary function as a discontinuous function. As an example, a composite function of trigonometric function of sin and cos function is implemented. Through experimental evaluations, we confirm that the circuits of step function, absolute function, and discontinuous function perform as target function.

1.2 Dissertation Overview

This dissertation proposes hardware optimization of SC, especially to omit use of SN generators. This dissertation is organized as follows:

Chapter 2 [Stochastic Number Duplicators Based on Bit Re-arrangement Using Randomized Bit Streams] proposes *FSR* and *RRR* duplicators, which generates and outputs a new SN which has the equivalent value with the input SN but has an independent bit stream. In SC, the bit streams of the input SNs must be independent to obtain the correct calculation result. In many functions, two or more identical values are input. In these cases, duplicating SNs, or re-generating SNs with the equivalent value and independent bit stream, is required. In a conventional method, a flip-flop is used to delaying one bit of the input SN, creating a different, 1-bit delayed bit stream. Since the output SN has the (nearly) same value and different bit stream compared withe the input SN, this circuit can be said to have duplicated an SN. However, the output SN's bit stream is always the same if the same input SN is given. In [1], circuits using FFs as SN duplicators have been proposed. However, these circuits had errors since FFs output the same SN when the same SNs are given. As mentioned above, if bit streams of the duplicated SNs are dependent on each other, their arithmetic operation results become inaccurate. In this chapter, a randomized bit stream is introduced to re-arrange the bits stored in flip-flops. The SNs duplicated by the proposing FSR and RRR duplicators have the equivalent values but have independent bit streams, by effectively utilizing bit re-arrangement using randomized bit streams. Experimental evaluation results demonstrate that the RRR duplicator, in particular, obtains more accurate results, reducing the mean square errors by 20%–89% compared with a conventional SN duplicator. Also, this chapter discusses the behavior of the proposed SN duplicators when the bit length of the input SN becomes longer.

Chapter 3 [Scalable Stochastic Number Duplicators for Accuracy-flexible Arithmetic Circuit Design] proposes $2^{n}RRR$ duplicator, which uniquely extends the RRR duplicator and has a scalable structure by changing the numbers of flip-flops for bit re-arrangement. As a nature of SC, changing the length of the input SNs will change the whole circuit's accuracy. However, in some implementations with re-convergence paths, the circuit itself will cause errors due to the dependency of the SNs, i.e., the length of the input SNs does not change that circuit's accuracy. In this chapter, we extend the RRR duplicator proposed in Chapter 2, enabling to change the accuracy of the circuit itself. The proposing 2^{n}RRR duplicator outputs different SNs every time and are all independent of each other. The 2^{n}RRR duplicator can be theoretically proved to flexibly change the accuracies of the arithmetic circuits. Also from the experimental evaluation results, this chapter clarifies that the 2^{n}RRR duplicator enables accuracy-flexible circuits. In a particular case, one instance of the proposed 2^{n}RRR duplicator reduces the mean square errors by more than 50% compared with the RRR duplicator proposed in Chapter 2.

Chapter 4 [Hardware Implementation of Step Function in Stochastic Computing and Its Applications] proposes step function in SC and its applications. With the rise of neural network and image processing, implementation of activation functions are becoming popular. However, steep functions and discontinuous functions are examples of arithmetic functions that are difficult to implement in SC due to the nature of SNs. Of the activation functions, the Rectified Linear Unit (ReLU) function, the step function, and their composite functions appear as steep functions and discontinuous functions. Implementation of steep functions and discontinuous functions is indispensable for the practical application of SC. However, when implementing arithmetic operations in SC, there is a constraint that the original function must be differentiable. There has been some steep functions that has been proposed, but they can only be used in specific situations, and cannot be used alone. To solve the problems of the conventional methods, this chapter firstly proposes hardware implementation of step function using SNs. The proposing circuit of utilizes flip-flops and an adder to perform as step function uniquely calculating the stored bits in the flip-flops. The proposed step function can be theoretically proved to perform as a step function. This chapter confirms that the proposing circuit behaves as a step function through experimental evaluations. Also as an application, steep functions or discontinuous functions can be realized by applying the discontinuity of the step function. As a steep function, this chapter also proposes hardware implementation of absolute function and discontinuous function, by synthesizing an arbitrary function as a discontinuous function. As an example, a composite function of trigonometric function of sin and cos function is implemented in this chapter. Through experimental evaluations, this chapter confirms that the circuits of step function, absolute function, and discontinuous function actually perform as target function.

Chapter 5 [Conclusion] summarizes this dissertation and gives future directions on hardware optimization of SC.

Chapter 2

Stochastic Number Duplicators Based on Bit Re-arrangement Using Randomized Bit Streams[1]

2.1 Introduction

2.1.1 Backgrounds

When constructing complex circuits based on SC, duplicating the SNs along the signal paths is quite necessary, especially when the signal path in a circuit has multi-fanouts. However, if bit streams of the duplicated SNs are somewhat related, or dependent on each other, the arithmetic operation results using them become inaccurate (See Section 2.2.3 for detail).

Here, let us consider an SN generator that outputs an SN whose value is equivalent as the input SN, and whose bit stream is different from the input SN. This SN generator is called an *SN duplicator*. The simplest implementation of an SN duplicator can is a 1-bit shift register using a single 1-bit flip-flop (FF) [1] (see also Figure 2.5). This SN duplicator makes the output bit stream different from the input bit stream but the appearance rate of 1's be almost equivalent if the bit stream is long enough, i.e, the input SN is duplicated using a different bit stream. Even with this SN duplicator, when the duplicated bit streams are re-used multiple times in a re-convergence path in a logic circuit, an inaccurate arithmetic result may still be generated since the bit streams are not independent of each other.

[1]Technical contents in this chapter have been presented in the publications ⟨2⟩, ⟨9⟩, ⟨19⟩, ⟨20⟩, and ⟨21⟩.

2.1.2 Proposal

This chapter proposes *two* SN duplicators, *FSR* and *RRR*. By buffering the input bits using randomized bit streams, the SNs duplicated by the FSR and RRR duplicators have the equivalent values and independent bit streams. From experimental evaluations, the RRR duplicator, in particular, obtains more accurate results, reducing the mean square errors (MSE) by 20%–89% compared with a conventional 1-bit FF-based SN duplicator even if a circuit has re-convergence paths.

2.1.3 Contributions

The contributions of this chapter are as follows:

1. This chapter proposes two efficient SN duplicators, FSR and RRR duplicators, based on bit re-arrangement using randomized bit streams;
2. In particular, the RRR duplicator reduces the MSE by up to 89% compared with a conventional 1-bit FF-based SN duplicator, even if a target circuit has re-convergence paths.

2.1.4 Organization

The rest of this chapter is organized as follows: Section 2.2 introduces stochastic numbers (SNs) and their duplication and discusses the necessity of their duplicators; Section 2.3. proposes SN duplicators based on bit re-arrangement using a randomized bit stream; Section 2.4 demonstrates the effectiveness of our proposed SN duplicators compared with conventional ones through experimental evaluations and discusses the behavior of the proposed SN duplicators when the bit length of the input SN becomes longer; Section 2.5 gives several concluding remarks.

2.2 Background of SN and Its Duplicators

2.2.1 Stochastic Numbers (SNs)

Stochastic numbers (SNs) are arbitrary-lengthed bit streams of where of 0's and 1's. For an given SN x, let $|x|$ be the length of the bit stream of the SN x, x_i be the i-th bit of the SN x, S_x be the number of 1's in the SN x, and V_x be x's value. V_x is given by:

$$V_x = P_x = S_x/|x|, \tag{2.1}$$

Table 2.1: Truth table of an AND gate.

a_i	b_i	c_i	Appearance rate
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	$P_a \times P_b$

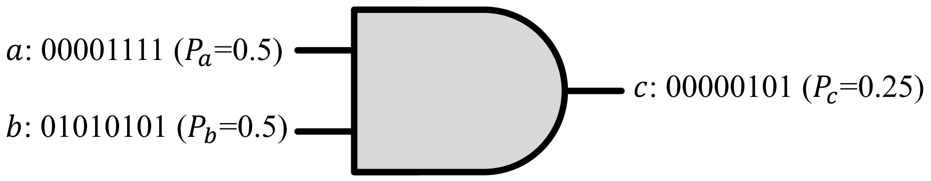

a: 00001111 (P_a=0.5)
b: 01010101 (P_b=0.5)
c: 00000101 (P_c=0.25)

Figure 2.1: Example of SN multiplication using an AND gate.

where P_x is the appearance rate of 1's in the bit stream of x ($0 \leq P_x \leq 1$). For example, for an 8-bit SN $x = 01100101$, $|x| = 8$ and $V_x = P_x = 4/8 = 0.5$ hold. Note that, several types of SN expression other than the one above [6] exists, but this chapter focuses on the SN expression above satisfying $V_x = P_x$. This SN expression is called the uni-polar expression.

In SC, SNs can be calculated by inputting the bit streams of them to a logic circuit sequentially. In uni-polar expressed SC, multiplication, weighted addition, and inversion are implemented by just using an AND gate, a MUX circuit and a NOT gate, respectively [6].

(a) Multiplication: An AND (logical product) gate is used for multiplication. The truth table of an AND gate with two input SNs a and b and an output SN c is shown in Table 2.1. From Table 2.1, V_c is represented by:

$$V_c = P_c = P_a \times P_b = V_a \times V_b. \tag{2.2}$$

Equation (2.2) shows that multiplication is realized by an AND gate. For example, in Figure 2.1, by multiplying $a = 00001111$ ($V_a = 0.5$) and $b = 01010101$ ($V_b = 0.5$), $c = 00000101$ ($V_c = 0.25$) is obtained by performing AND operation between them.

(b) Scaled addition: A MUX (multiplexer) circuit is used for scaled addition. The truth table of a MUX circuit, with two input SNs a and b, a data selector SN s, and an

Table 2.2: Truth table of a MUX circuit.

a_i	b_i	s_i	c_i	Appearance rate
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	$P_a \times P_b \times P_s$

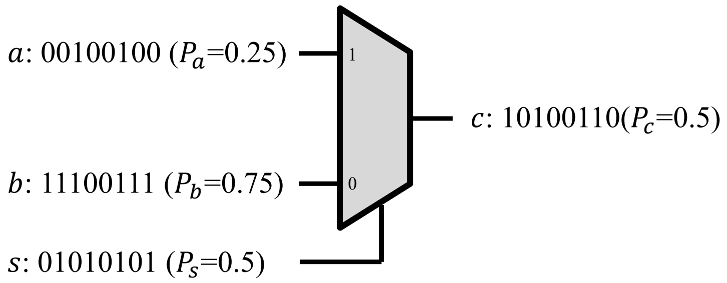


Figure 2.2: Example of SN scaled addition using a MUX circuit.

output SN c, is shown in Table 2.2. From Table 2.2, V_c is represented by:

$$\begin{aligned} V_c = P_c &= (1 - P_a) \times P_b \times P_s \\ &\quad + P_a \times (1 - P_b) \times (1 - P_s) \\ &\quad + P_a \times P_b \times (1 - P_s) \\ &\quad + P_a \times P_b \times P_s \\ &= P_a \times (1 - P_s) + P_b \times P_s \\ &= V_a \times (1 - V_s) + V_b \times V_s. \end{aligned} \tag{2.3}$$

By appropriately setting the data selector SN s, addition is realized by a MUX circuit. For example, in Figure 2.2, a data selector s = 01010101 (V_s = 0.5) is given. When SNs a = 00100100 (V_a = 0.25) and b = 11100111 (V_b = 0.75) are given, its output becomes c = 10100110 (V_c = 0.5) by performing MUX operation.

Table 2.3: Truth table of a NOT gate

a_i	b_i	Appearance rate
0	1	$1 - P_a$
1	0	P_a

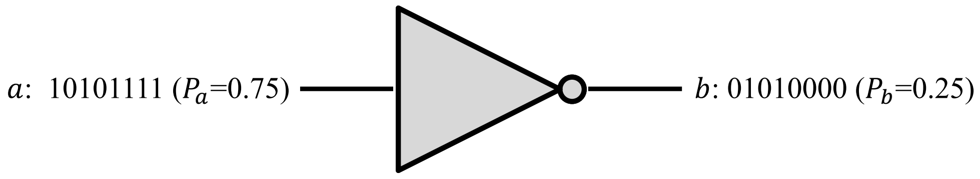


Figure 2.3: Example of SN inversion using a NOT gate.

(c) Inversion: A NOT (logical inverter) gate is used for inversion. The truth table of a NOT gate with an input SN a and an output SN $b$is shown in Table 2.3. From Table 2.3, V_b is represented by:

$$V_b = P_b = 1 - P_a = 1 - V_a. \tag{2.4}$$

As in Equation (2.4), the value of output SN b can be inverted value of the input SN a. For example, in Figure 2.3, an input a = 10101111 (V_a = 0.75) is given and an output b = 01010000 (V_b = 0.25) is obtained by performing NOT operation.

2.2.2 SN Duplicators

Arithmetic circuits of SC can be implemented by logic circuits as shown in Section 2.2.1. Here, let us consider a *squarer* based on an AND gate expressed by $V_b = V_a \times V_a$. From the discussion in Section 2.2.1, a squarer seems to be implementable using an AND gate as in Figure 2.4(a). However, the squarer in Figure 2.4(a) outputs the same bit steam as the input SN. A correct square value cannot be obtained by this circuit.

To obtain the correct square value, *duplicating* the input SN is necessary as in Figure 2.4(b). In Figure 2.4(b), the SN duplicator DUP duplicates the input SN. In this case, the input a is 01001110 ($V_a = 4/8 = 0.5$) and output a' is 10011100 ($V_{a'} = 4/8 = 0.5$). DUP generates an SN a' with the equivalent value as a but with the different bit stream. After that, by multiplying a and a', output SN b = 00001100 ($V_b = 2/8 = 0.25$) is obtained, with the correct square value.

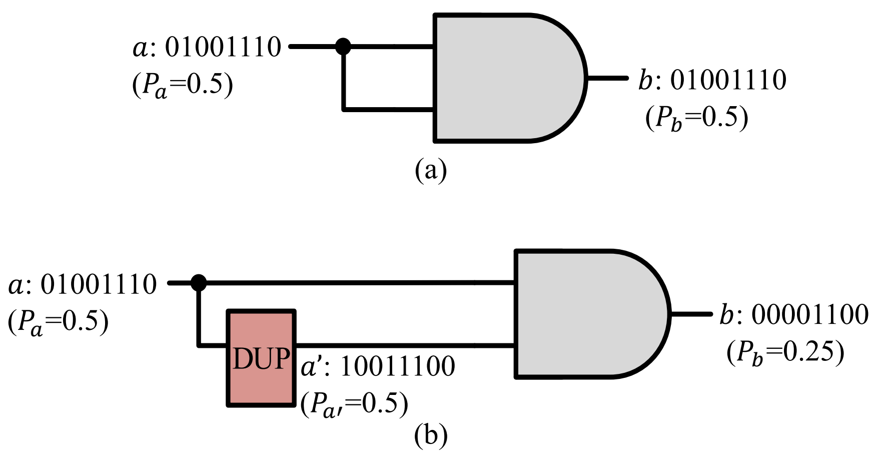


Figure 2.4: SN squarer using an AND gate. (a) Correct. (b) Incorrect.

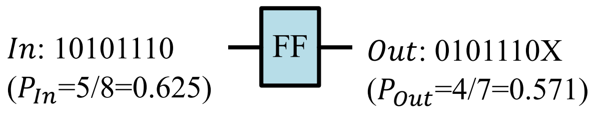


Figure 2.5: SN duplicator using FF [1].

Here, we define SN duplication as follows: When an input SN In is given, generating an SN Out which has a different bit stream from In but with the equivalent value, i.e., $V_{Out} = V_{In}$. This chapter proposes effective SN duplicators which obtain accurate enough output values, even if a circuit has re-convergence paths. To do so, the SN duplicators must satisfy the following condition:

Condition 1. The values of input SN In and output SN Out must be equal ($V_{In} = V_{Out}$) and the bit streams of them differ ($In \neq Out$).[2]

An SN duplicator using a single 1-bit FF has been proposed in [1] as the simplest way to satisfy Condition 1. As in Figure 2.5, a 1-bit delayed SN is obtained by using a 1-bit FF-based SN duplicator. Thus, the value of the output SN is nearly equal to the value of the input SN, if the bit length of input SN is long enough. Note that, X in this figure is the unknown, initial bit in the FF. However, if a target circuit has re-convergence paths, a correct output cannot be obtained.

For example, consider a circuit that outputs the eighth power unit calculating the

[2]As discussed in Sections 2.3 and 2.5, the expected values of V_{Out} in the proposed FSR and RRR duplicators become equal to V_{In}, if the bit length of the input SN is long enough. Especially, in the RRR duplicator, if the bit length of the input SN is longer, the duplication error always becomes smaller.

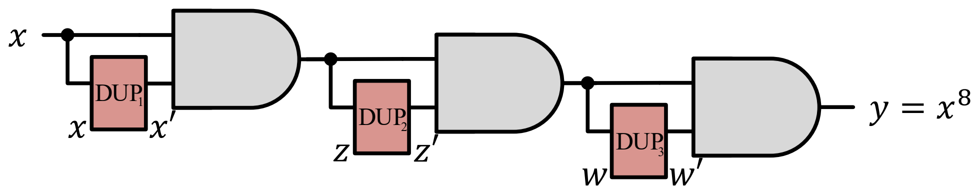


Figure 2.6: Eighth power unit using SN duplicators.

following equation:

$$V_y = V_x^8. \tag{2.5}$$

In Figure 2.6, three squarers in Figure 2.4(b) are connected in series. DUP$_1$, DUP$_2$, and DUP$_3$ represent SN duplicators in this figure. The SN x is the input and the SN y is the output of this whole circuit. x' is the output of DUP$_1$. z and z' are the input and output SNs of DUP$_2$, respectively. w and w' are the input and output SNs of DUP$_3$, respectively.

Assuming the SN duplicator proposed in [1] is used as DUP$_1$, DUP$_2$, and DUP$_3$, the i-th bit of each SN in Figure 2.6 become:

$$\begin{aligned}
z_i &= x_i \wedge x'_i = x_i \wedge x_{i-1}, && (2.6)\\
w_i &= z_i \wedge z'_i = [x_i \wedge x_{i-1}] \wedge [x_{i-1} \wedge x_{i-2}] \\
&= x_i \wedge x_{i-1} \wedge x_{i-2}, && (2.7)\\
y_i &= w_i \wedge w'_i \\
&= [x_i \wedge x_{i-1} \wedge x_{i-2}] \wedge [x_{i-1} \wedge x_{i-2} \wedge x_{i-3}] \\
&= x_i \wedge x_{i-1} \wedge x_{i-2} \wedge x_{i-3}. && (2.8)
\end{aligned}$$

Therefore, the value V_y of the output SN y becomes:

$$V_y = P_y = {P_x}^4 = {V_x}^4, \tag{2.9}$$

meaning that this circuit cannot express eighth power as in Equation (2.5). Instead, it expresses fourth power, regardless of the bit length of the input SN.

This is because of the following reason: When an SN *In* is input to the 1-bit FF-based SN duplicator in [1], an output SN *Out* is obtained. If the same SN *In* is input, the bit stream of its output SN *Out* is exactly the same every time. Therefore, the re-convergence path in the target circuit leads to a correlated result. Thus, an accurate result cannot be obtained if the target circuit has re-convergence paths.

Overall, in addition to Condition 1, Condition 2 below is also required for an *ideal* SN duplicator:

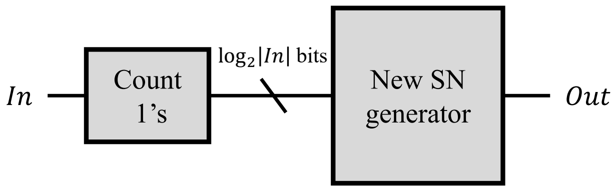


Figure 2.7: SN Duplicator proposed in [2].

Condition 2. When SN In is input to two SN duplicators with the same circuit, the bit stream of the output SN Out of one SN duplicator differs from that of the output SN Out' of the other SN duplicator.

2.2.3 Existing SN Duplicators [1, 2]

Several SN duplicators has been proposed so far [1, 2]. In [1], a 1-bit FF-based SN duplicator as in Figure 2.5 was proposed. However, as discussed above, this SN duplicator satisfies only Condition 1 but does not satisfy Condition 2.

In [2], an SN duplicator which newly generates an SN is proposed (Figure 2.7). This SN duplicator generates a completely new SN by the following steps:

1. Count up the number of 1's in the input SN In
2. Calculates V_{In}
3. Newly re-generate an output SN Out using LFSR (Linear Feedback Shift Register)
    (a) A random number r ($0 \leq r < 1$) is generated
    (b) If $r > V_{In}$, the current output bit is set to be one and otherwise zero
    (c) Repeat step 3.(b) until enough bit length is obtained

By following these steps, a new SN Out with the equivalent value as In is generated.

This SN duplicator can be considered as an *ideal* SN duplicator, satisfying both Condition 1 and Condition 2. However, this SN duplicator requires $\log_2 |In|$ FFs to count up 1's in the input SN In, making the circuit area much larger. Also, since counting all the 1's in the input SN is necessary, this SN duplicator cannot output its output SN until this counting is completely finished. The latency of this SN duplicator becomes larger than or equal to the length of its input SN. Therefore, the SN duplicator proposed in [2] is ideal but cannot be used in a practical situation. Note that, in [32], improvement of LFSRs' accuracy is discussed.

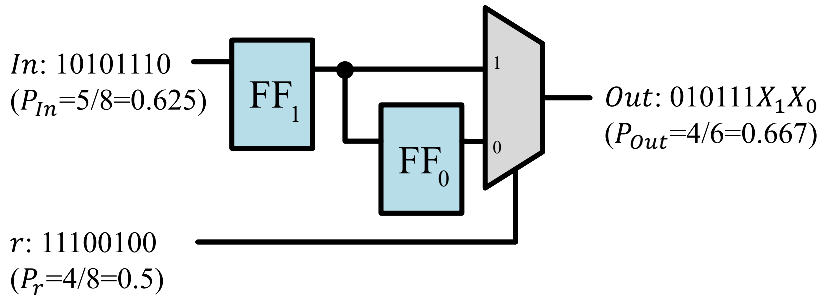


Figure 2.8: FSR duplicator.

From discussions above, the main issue here is designing an ideal SN duplicator that satisfies both Condition 1 and Condition 2, but has low area overheads and low latency.

2.3 SN Duplicators Based on Bit Re-arrangement

2.3.1 Proposal

To satisfy Condition 2, this chapter proposes an SN duplicator introducing random bit streams which is independent from the input SN, to re-arrange the input SN. The SN duplicators itselves generate different output bit streams even if the same input SN is given, i.e., these SN duplicators satisfy Condition 2. Further, by differentiating this random bit stream in every SN duplicator, all the SN duplicators can output different SNs.

Based on this idea, this chapter proposes two SN duplicators, FSR and RRR, utilizing small amount of buffers and output the re-arranged inputs in a bit-by-bit manner. As demonstrated in Section 2.4, the proposed SN duplicators reduces their errors compared with the conventional 1-bit FF-based SN duplicator in a circuit with re-convergence paths.

2.3.2 FSR Duplicator

Firstly, this chapter proposes the FSR (Flip-flop Selecting circuit using a Random bit stream) duplicator shown in Figure 2.8.

This SN duplicator selects a 1-bit delayed or 2-bit delayed SN *randomly* by introducing a random bit stream to select each output bit. The two FFs, FF_0 and FF_1, generate 1- or 2-bit delayed SN, respectively. The output of FF_1 becomes the 1-bit delayed SN and the output of FF_0 becomes the 2-bit delayed SN. Then the MUX circuit and a random bit stream select the bit to output. Since a 1-bit delayed SN and a 2-bit delayed SN is selected randomly, Condition 2 will be satisfied. As in Figure 2.8, the output can be obtained in a bit-by-bit manner, i.e., the latency of an FSR duplicator becomes zero clock cycles, same as the SN duplicator in [1].

The i-th bit Out_i of the output SN Out becomes:

$$Out_i = In_{i-1} \times r_i + In_{i-2} \times (1 - r_i), \tag{2.10}$$

where In is the input SN and r ($r_i = 0$ or 1) is a random bit stream. If $V_r = 1/2$ and i is sufficiently large, the expected value E_i of Out_i becomes:

$$E_i = \frac{1}{2}(In_{i-1} + In_{i-2}) \approx P_{In} = V_{In}, \tag{2.11}$$

meaning that the expected value of output V_{Out} is the same as the input value V_{In} and thus the FSR duplicator satisfies Condition 1. Note that, Section 2.5 discusses the errors between V_{In} and V_{Out} when the bit length of the input SN becomes longer.

As an example of implementation of an FSR duplicator, if $In = 10101110$ and $r = 11100100$ are input, $Out = 010111X_1X_0$ is output, where X_0 and X_1 is the unknown, initial bits of FF_0 and FF_1, respectively. In this case, the values of input and output SN becomes $V_{In} = 5/8 = 0.675$ and $V_{Out} = 4/6 = 0.667$, respectively.[3]

Table 2.4 shows examples of input/output SNs and their values of an FSR duplicator where the initial values of FF_0 and FF_1 are set to 0 and 1, respectively. Here, the duplication errors become smaller if SNs with longer bit lengths are input.

Let us apply this FSR duplicator to the eighth power unit as in Figure 2.6 when $V_r = 1/2$. For simplicity, we also assume that $|In|$ is sufficiently large and the delays of each SN duplicator are fixed over time. Focusing on x_i, the i-th bit of input SN x of DUP$_1$. If DUP$_1$ is implemented by FSR, SN x' is output with D_1 clock cycle ($D_1 = 1$ or $D_1 = 2$) delays, i.e., $x'_i = x_{i-D_1}$. In the same way, DUP$_2$ and DUP$_3$ output D_2 and D_3 clock cycle (D_2 and D_3 are 1 or 2, respectively) delayed input, respectively, i.e., $z'_i = z_{i-D_2}$ and $w'_i = w_{i-D_3}$. Therefore, the i-th bit y_i of the output SN y of the whole eighth power unit becomes:

$$\begin{aligned} y_i = x_i &\wedge x_{i-D_1} \wedge x_{i-D_2} \wedge x_{i-D_3} \wedge x_{i-D_1-D_2} \\ &\wedge x_{i-D_2-D_3} \wedge x_{i-D_3-D_1} \wedge x_{i-D_1-D_2-D_3}. \end{aligned} \tag{2.12}$$

In Equation (2.12), D_1, D_2, and D_3 become 1 or 2 with the possibility of 50%. Table 2.5 shows all the patterns of the delays and their corresponding output y_i. From Table 2.5,

[3]In the calculation example of V_{Out} here, we exclude 'X's in Out for simplicity. Thus we have $V_{Out} = 4/6 = 0.667$.

Table 2.4: Examples of input/output SNs of an FSR duplicator.

Bit length	Input SN In (V_{In})	Random SN r (V_r)	Output SN Out (V_{Out})
8 bits	10110110 (0.625)	00111010 (0.5)	11101000 (0.5)
16 bits	11011110 01011100 (0.625)	00001001 01111101 (0.5)	01111000 00111011 (0.5625)
24 bits	11110101 10011100 01110101 (0.625)	00110001 01001010 11011110 (0.5)	11100111 00111001 11001010 (0.5833···)
32 bits	10101011 01001100 10110111 11101101 (0.625)	00011111 01001101 10010111 00110000 (0.5)	10110110 00111011 01001111 10010110 (0.59375)

the value of the output SN y becomes:

$$\begin{aligned} V_y = P_y &= \frac{3}{8}P_x^6 + \frac{3}{8}P_x^5 + \frac{1}{4}P_x^4 \\ &= \frac{3}{8}V_x^6 + \frac{3}{8}V_x^5 + \frac{1}{4}V_x^4. \end{aligned} \tag{2.13}$$

Still, V_y does not completely represent the eighth power of an input SN, but approaches to the correct value compared with the SN duplicator proposed in [1].

2.3.3 RRR Duplicator

To re-arrange the bit stream of an input SN further, two more MUXs are added to the FSR duplicator. Here, this chapter secondly proposes the RRR (Register based Re-arrangement circuit using a Random bit stream) duplicator shown in Figure 2.9.

The RRR duplicator also utilizes a random bit stream r as well as the FSR duplicator. If the i-th bit r_i in r is zero, i.e., if $r_i = 0$, SNs are duplicated in the following steps:

1. The bit stored in FF_0 is output as its i-th bit Out_i of output SN Out.

Table 2.5: Appearance rates of D_1–D_3 and its function.

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

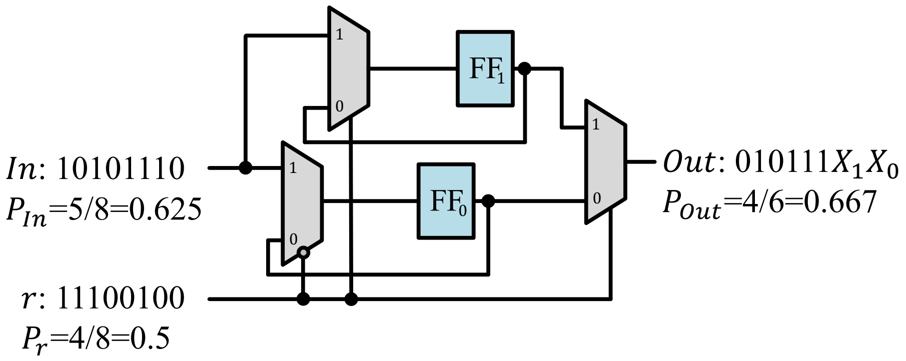


Figure 2.9: RRR duplicator.

2. The i-th bit In_i of input SN In is newly stored into FF_0.

3. The bit stored in FF_1 is not changed.

In the same way, if $r_i = 1$, FF_1 is used instead of FF_0, and the bit stored in FF_0 is not changed. Note that, this SN duplicator outputs almost all the bits in the input SN and thus no inaccuracy is expected in arithmetic operations, i.e., the RRR duplicator also satisfies Condition 2. As in Figure 2.9, the output can be obtained in a bit-by-bit manner, i.e., its latency becomes zero clock cycles as well as the SN duplicator in [1] and the FSR duplicator.

Here, let $F^0_{j,i}$ $(j < i)$ be a 0/1 variable, which is 1 if the j-th input bit In_j is stored in

FF_0 when the current input bit is In_i and otherwise 0. $F^0_{j,i}$ can be calculated by:

$$F^0_{j,i} = (1 - r_j) \times \prod_{k=j+1}^{i-1} r_k. \tag{2.14}$$

In the same way, a 0/1 variable $F^1_{j,i}$ for FF_1 becomes:

$$F^1_{j,i} = r_j \times \prod_{k=j+1}^{i-1} (1 - r_k). \tag{2.15}$$

Let X_0 and X_1 be the initial bits stored in FF_0 and FF_1, respectively. The i-th bit Out_i of the output SN Out becomes:

$$\begin{aligned} Out_i &= X_0 \times (1 - r_i) \times \prod_{k=0}^{i-1} r_k \\ &+ X_1 \times r_i \times \prod_{k=0}^{i-1}(1 - r_k) \\ &+ (1 - r_i) \times \sum_{j=0}^{i-1}(In_j \times F^0_{j,i}) \\ &+ r_i \times \sum_{j=0}^{i-1}(In_j \times F^1_{j,i}). \end{aligned} \tag{2.16}$$

When $V_r = 1/2$ and i is sufficiently large, the expected value E_i of Out_i becomes:

$$E_i = \sum_{j=0}^{i-1} \frac{In_j}{2^{i-j}} \approx P_{In} = V_{In}, \tag{2.17}$$

meaning that the expected value of output V_{Out} is the same as the input value V_{In} and thus the output of the RRR duplicator also satisfies Condition 1.

For example, if $In = 10101110$ ($V_{In} = 5/8 = 0.675$) and $r = 11100100$ are input, $Out = 01111X_10X_0$ ($V_{Out} = 4/6 = 0.667$) is output. Thus V_{Out} is almost equal to V_{In} if the input SN is long enough.

The RRR duplicator duplicates SNs whose bits streams are independent of each other. In addition, since most input bits in the bit steam of the input SN are used as output, inaccuracy in arithmetic operations can be ignored. Note that, Section 2.5 discusses the transition of errors between V_{In} and V_{Out} when the bit length of the input SN changes.

Let us apply this RRR duplicator to the eighth power unit as in Figure 2.6 when $V_r = 1/2$. For simplicity, we also assume that $|In|$ is sufficiently large and the delays of each SN duplicator are fixed over time. Alike the eighth power unit using the FSR duplicator, the i-th bit y_i of output SN y of the eighth power unit using the RRR duplicator is given by Equation (2.12). From Equation (2.17), the possibility of the RRR duplicator outputting the n-bit delayed input becomes $(1/2)^n$ $(n \geq 1)$. Alike the FSR duplicator, by considering all the patterns of D_1, D_2, and D_3 in Equation (2.12), the value of the output SN of the eighth power unit using the RRR duplicator becomes:

$$V_y = \frac{16}{105}V_x^8 + \frac{2}{15}V_x^7 + \frac{13}{35}V_x^6 + \frac{1}{5}V_x^5 + \frac{1}{7}V_x^4. \qquad (2.18)$$

Equation (2.18) includes the term of V_x^8 and thus it approaches to the correct value compared with the value of the output SN of the same circuit using SN duplicator proposed in [1] or that using FSR duplicator proposed in the previous section.

2.4 Experimental Evaluations

2.4.1 Accuracy Comparison

Firstly, the calculation accuracy of the proposed SN duplicators, using several arithmetic circuits including re-convergence paths, is evaluated.

Benchmark Circuits

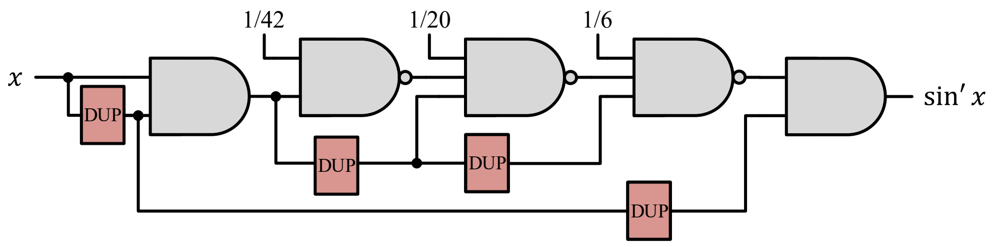
As benchmark circuits, the following circuits are used:

1. The eighth power unit as in Figure 2.6;

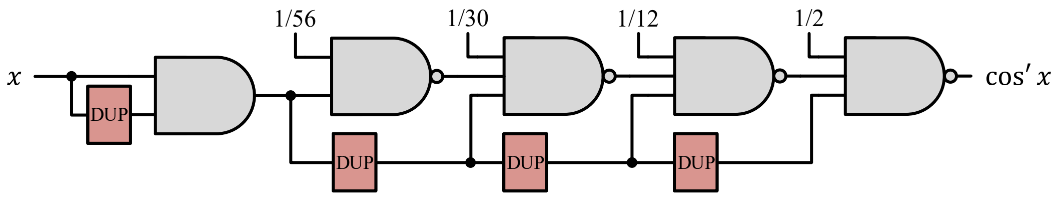
2. The 7th order approximation sine function denoted by sin′ as in Figure 2.10(a);

3. The 8th order approximation cosine function denoted by cos′ as in Figure 2.10(b).

In general, stochastic circuits under the following conditions can be implemented by using duplicators: the range and domain are between 0 and 1; the function is differentiable. Note that, circuits 2 and 3 are based on the Horner's method [1, 33]. Only functions with coefficients, which are alternately positive and negative and their magnitudes are monotonically decreasing, can be implemented by this method [1].



(a)



(b)

Figure 2.10: Circuits of functions using SN duplicators.
(a) $\sin' x$. (b) $\cos' x$.

The $\sin' x$ function is expressed by:

$$\begin{aligned} \sin x &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \\ &= x\left(1 - \frac{x^2}{6}\left(1 - \frac{x^2}{20}(1 - \frac{x^2}{42})\right)\right) \\ &= \sin' x. \end{aligned} \tag{2.19}$$

The circuit of $\sin' x$ is shown in Figure 2.10(a). The DUPs in Figure 2.10(a) show SN duplicators.

The $\cos' x$ function is expressed by:

$$\begin{aligned} \cos x &\approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \\ &= 1 - \frac{x^2}{2}\left(1 - \frac{x^2}{12}\left(1 - \frac{x^2}{30}(1 - \frac{x^2}{56})\right)\right) \\ &= \cos' x. \end{aligned} \tag{2.20}$$

The circuit of $\cos' x$ is shown in Figure 2.10(b). The DUPs in Figure 2.10(b) also show SN duplicators.

Setup

The following four SN duplicators are compared: the 1-bit FF-based SN duplicator proposed in [1], the ideal SN duplicator proposed in [2], our FSR duplicator, and our RRR duplicator. In this experiment, mean square errors (MSE) and maximum errors are used to evaluate the SN duplicators.

(a) MSE: MSE is a statistical value that measures how close the obtained values are to the theoretical value, which is calculated by:

$$MSE(f, x, n) = \frac{1}{n}\sum_{i=1}^{n}(f_{theory}(x) - f_{actual}(x, i))^2, \tag{2.21}$$

where $f_{theory}(x)$ is the theoretical value when a real number x is input to Equation (2.5), Equation (2.19), or Equation (2.20), n is the number of trials, and $f_{actual}(x, i)$ is the obtained value of the output SN of each circuit through numerical simulation at the i-th trial when an SN with value x is input.

(b) Maximum error: Maximum error is calculated by:

$$ERR_{max}(f, x, n) = \max_{i \in \{1, \cdots, n\}} |f_{theory}(x) - f_{actual}(x, i)|. \tag{2.22}$$

Python 3.6.3[34] was used for this simulation. Each input SN x are input 1,000 times. For each SN duplicator in all trials, a random bit stream r was generated and then the output SNs were evaluated by MSE values and maximum errors. The input x ranges from 0.0 to 1.0. The two cases below were performed for both MSE values and maximum errors:

Case 1: The bit length of every SN is set to 255 bits based on [12]. Here, an 8-bit LFSR is used to generate a random bit stream r in our FSR and RRR duplicators. An 8-bit LFSR is also used in [2] when newly generating an SN.

Case 2: To evaluate the longer bit length of SNs, the bit length of every SN set to 4,095 bits. In this case, a 12-bit LFSR is used to generate a random bit stream r in our FSR and RRR duplicators. A 12-bit LFSR is also used in [2] when newly generating an SN.

Results

Table 2.6 summarizes the MSE values of Case 1. Each value in Table 2.6 shows the MSE value ($MSE(f, x, n)$ given by Equation (2.21)) of 1,000 trials ($n = 1{,}000$) for each

Table 2.6: Experimental evaluations in MSE values (bit length: 255 bits).

Function	x	[1]	[2]	FSR (Proposed)	RRR (Proposed)
x^8	0.0	0	0	0	0
	0.1	4.65×10^{-7}	1.00×10^{-16}	4.65×10^{-8}	1.54×10^{-8}
	0.2	9.35×10^{-6}	3.10×10^{-8}	3.52×10^{-6}	9.22×10^{-7}
	0.3	1.04×10^{-4}	3.79×10^{-7}	3.45×10^{-5}	8.80×10^{-6}
	0.4	7.07×10^{-4}	2.90×10^{-6}	2.40×10^{-4}	5.05×10^{-5}
	0.5	3.65×10^{-3}	1.55×10^{-5}	1.30×10^{-3}	3.10×10^{-4}
	0.6	1.26×10^{-2}	7.17×10^{-5}	5.05×10^{-3}	1.38×10^{-3}
	0.7	3.12×10^{-2}	2.34×10^{-4}	1.41×10^{-2}	4.71×10^{-3}
	0.8	5.65×10^{-2}	4.88×10^{-4}	2.79×10^{-2}	1.02×10^{-2}
	0.9	4.69×10^{-2}	7.77×10^{-4}	2.52×10^{-2}	1.14×10^{-2}
	1.0	0	0	0	0
Average		1.38×10^{-2} (100%)	1.46×10^{-4} (1%)	6.73×10^{-3} (49%)	2.56×10^{-3} (19%)
$\sin' x$	0.0	0	0	0	0
	0.1	1.07×10^{-5}	4.58×10^{-6}	3.13×10^{-4}	1.56×10^{-5}
	0.2	6.50×10^{-5}	1.04×10^{-5}	5.56×10^{-4}	3.01×10^{-5}
	0.3	1.38×10^{-4}	2.08×10^{-5}	7.05×10^{-4}	6.45×10^{-5}
	0.4	3.46×10^{-4}	3.95×10^{-5}	9.36×10^{-4}	6.98×10^{-5}
	0.5	5.01×10^{-4}	6.63×10^{-5}	9.85×10^{-4}	1.24×10^{-4}
	0.6	6.07×10^{-4}	1.00×10^{-4}	1.02×10^{-3}	1.90×10^{-4}
	0.7	7.40×10^{-4}	1.37×10^{-4}	9.41×10^{-4}	1.82×10^{-4}
	0.8	4.96×10^{-4}	1.47×10^{-4}	6.46×10^{-4}	1.58×10^{-4}
	0.9	2.49×10^{-4}	1.21×10^{-4}	3.88×10^{-4}	1.27×10^{-4}
	1.0	3.74×10^{-5}	3.88×10^{-5}	3.79×10^{-5}	2.76×10^{-5}
Average		2.90×10^{-4} (100%)	6.25×10^{-5} (22%)	5.95×10^{-4} (205%)	9.01×10^{-5} (31%)
$\cos' x$	0.0	0	0	0	0
	0.1	3.45×10^{-5}	3.30×10^{-5}	4.68×10^{-5}	1.83×10^{-5}
	0.2	8.28×10^{-5}	7.61×10^{-5}	1.25×10^{-4}	6.21×10^{-5}
	0.3	1.38×10^{-4}	1.32×10^{-4}	2.12×10^{-4}	1.19×10^{-4}
	0.4	2.12×10^{-4}	2.08×10^{-4}	3.39×10^{-4}	2.06×10^{-4}
	0.5	3.10×10^{-4}	2.69×10^{-4}	4.38×10^{-4}	2.49×10^{-4}
	0.6	3.44×10^{-4}	2.95×10^{-4}	5.59×10^{-4}	2.76×10^{-4}
	0.7	4.12×10^{-4}	3.56×10^{-4}	6.59×10^{-4}	2.97×10^{-4}
	0.8	3.53×10^{-4}	3.21×10^{-4}	5.84×10^{-4}	3.07×10^{-4}
	0.9	3.01×10^{-4}	2.80×10^{-4}	4.47×10^{-4}	2.14×10^{-4}
	1.0	9.70×10^{-5}	9.47×10^{-5}	1.02×10^{-4}	8.22×10^{-5}
Average		2.08×10^{-4} (100%)	1.89×10^{-4} (90%)	3.20×10^{-4} (153%)	1.67×10^{-4} (80%)

functions (x^8, $\sin' x$, and $\cos' x$), input value x ($x = 0.0, 0.1, 0.2, \ldots, 1.0$) and the SN duplicators ([1], [2], FSR, and RRR). The row “Average” shows the overall average of the MSE values of all x’s. The row under “Average” shows the ratio of each SN duplicator’s “Average” to the “Average” of the SN duplicator in [1].

From this table, our FSR duplicator reduces MSE by 51% and our RRR duplicator reduces MSE by 81% compared with the SN duplicator in [1] for the eighth power unit. Note that, the SN duplicator in [2] reduces MSE to 1% compared with the SN duplicator in [1], since the SN duplicator in [2] is an ideal SN duplicator and generates a completely new SN. However, this SN duplicator requires large latency and large area compared with the other three SN duplicators as discussed in the next subsection. Especially, its latency becomes 255 clock cycles, meaning that the output cannot be obtained in a bit-by-bit manner unlike the other three SN duplicators.

In the cases of the $\sin'$ circuit and $\cos'$ circuit, the RRR duplicator reduces MSE by 69% and 20% compared with the SN duplicator in [1], respectively. Especially, in the case of the $\cos'$ circuit, the RRR duplicator realizes the smallest MSE among the all four SN duplicators.

Table 2.7 summarizes the MSE values of Case 2. When the bit length of SNs becomes longer, the proposed RRR duplicator realizes the best MSE values, reducing up to 89% in the case of the $\sin'$ circuit.

Tables 2.8 and 2.9 summarize the maximum errors of Cases 1 and 2, respectively. Each value in Tables 2.8 and 2.9 show the maximum error ($ERR_{max}(f, x, n)$ given by Equation (2.22)) of 1,000 trials ($n = 1{,}000$) for each functions (x^8, $\sin' x$, and $\cos' x$), input value x ($x = 0.0, 0.1, 0.2, \ldots, 1.0$) and the SN duplicators ([1], [2], FSR, and RRR). The row “Average” shows the overall average of the maximum errors of all x’s. The row under “Average” shows the ratio of each SN duplicator’s “Average” to the “Average” of [1].

From Tables 2.8 and 2.9, the RRR duplicator’s maximum error becomes the smallest in most cases using SN duplicators. Especially when the bit length is long (4,095 bits), the RRR duplicator’s maximum error becomes the smallest in all the cases using SN duplicators. The FSR duplicator also reduces maximum error with longer bit length, but sometimes is inferior to the SN duplicator in [1]. Thus, from the aspect of maximum errors, the RRR duplicator is superior to the SN duplicator in [1] and the FSR duplicator.

Table 2.7: Experimental evaluations in MSE values (bit length: 4,095 bits).

Function	x	[1]	[2]	FSR (Proposed)	RRR (Proposed)
x^8	0.0	0	0	0	0
	0.1	3.82×10^{-8}	6.25×10^{-11}	1.19×10^{-8}	2.12×10^{-9}
	0.2	3.14×10^{-6}	7.41×10^{-10}	6.41×10^{-7}	8.54×10^{-8}
	0.3	6.65×10^{-5}	1.88×10^{-8}	1.50×10^{-5}	1.96×10^{-6}
	0.4	6.33×10^{-3}	1.77×10^{-7}	1.72×10^{-3}	2.91×10^{-5}
	0.5	3.45×10^{-3}	1.06×10^{-6}	1.15×10^{-3}	2.44×10^{-3}
	0.6	1.27×10^{-2}	5.02×10^{-6}	4.94×10^{-3}	1.27×10^{-3}
	0.7	3.35×10^{-2}	1.58×10^{-5}	1.49×10^{-2}	4.55×10^{-3}
	0.8	5.85×10^{-2}	4.74×10^{-5}	2.94×10^{-2}	1.01×10^{-2}
	0.9	5.09×10^{-2}	1.27×10^{-4}	2.82×10^{-2}	1.09×10^{-2}
	1.0	0	0	0	0
Average		1.50×10^{-2} (100%)	1.78×10^{-5} (0.1%)	7.29×10^{-3} (48%)	2.66×10^{-3} (17%)
$\sin' x$	0.0	0	0	0	0
	0.1	7.36×10^{-6}	1.37×10^{-5}	2.56×10^{-5}	5.17×10^{-6}
	0.2	3.30×10^{-5}	1.51×10^{-5}	4.74×10^{-5}	7.42×10^{-6}
	0.3	1.10×10^{-4}	1.56×10^{-5}	8.21×10^{-5}	1.52×10^{-5}
	0.4	2.53×10^{-4}	1.48×10^{-5}	1.37×10^{-4}	2.88×10^{-5}
	0.5	4.09×10^{-4}	1.74×10^{-5}	1.93×10^{-4}	4.21×10^{-5}
	0.6	5.33×10^{-4}	1.71×10^{-5}	2.32×10^{-4}	5.37×10^{-5}
	0.7	5.39×10^{-4}	1.79×10^{-5}	2.39×10^{-4}	5.82×10^{-5}
	0.8	4.07×10^{-4}	2.09×10^{-5}	1.66×10^{-4}	4.43×10^{-5}
	0.9	1.66×10^{-4}	1.79×10^{-5}	7.90×10^{-5}	2.52×10^{-5}
	1.0	6.31×10^{-6}	6.07×10^{-6}	6.12×10^{-6}	6.24×10^{-6}
Average		2.23×10^{-4} (100%)	1.42×10^{-5} (6%)	1.09×10^{-4} (48%)	2.60×10^{-5} (11%)
$\cos' x$	0.0	0	0	0	0
	0.1	1.04×10^{-6}	2.78×10^{-6}	1.88×10^{-6}	1.19×10^{-6}
	0.2	4.33×10^{-6}	7.46×10^{-6}	7.16×10^{-6}	4.41×10^{-6}
	0.3	8.74×10^{-6}	1.16×10^{-5}	1.49×10^{-5}	8.68×10^{-6}
	0.4	1.51×10^{-5}	1.68×10^{-5}	2.40×10^{-5}	1.25×10^{-5}
	0.5	2.24×10^{-5}	2.09×10^{-5}	3.64×10^{-5}	1.67×10^{-5}
	0.6	3.02×10^{-5}	2.20×10^{-5}	4.36×10^{-5}	1.88×10^{-5}
	0.7	3.96×10^{-5}	2.48×10^{-5}	5.19×10^{-5}	2.27×10^{-5}
	0.8	3.72×10^{-5}	2.24×10^{-5}	4.38×10^{-5}	2.17×10^{-5}
	0.9	2.89×10^{-5}	2.29×10^{-5}	3.15×10^{-5}	1.86×10^{-5}
	1.0	9.92×10^{-6}	9.54×10^{-6}	9.83×10^{-6}	9.96×10^{-6}
Average		1.79×10^{-5} (100%)	1.46×10^{-5} (81%)	2.40×10^{-5} (134%)	1.22×10^{-5} (68%)

Table 2.8: Experimental evaluations in maximum errors (bit length: 255 bits).

Function	x	[1]	[2]	FSR (Proposed)	RRR (Proposed)
x^8	0.0	0	0	0	0
	0.1	3.92×10^{-3}	1.00×10^{-8}	3.92×10^{-3}	3.92×10^{-3}
	0.2	1.96×10^{-2}	3.92×10^{-3}	1.18×10^{-2}	7.84×10^{-3}
	0.3	3.52×10^{-2}	3.86×10^{-3}	3.52×10^{-2}	1.17×10^{-2}
	0.4	8.56×10^{-2}	1.50×10^{-2}	6.21×10^{-2}	3.86×10^{-2}
	0.5	1.33×10^{-1}	2.35×10^{-2}	9.41×10^{-2}	5.49×10^{-2}
	0.6	1.99×10^{-1}	3.03×10^{-2}	1.48×10^{-1}	1.01×10^{-1}
	0.7	2.64×10^{-1}	5.22×10^{-2}	2.36×10^{-1}	1.54×10^{-1}
	0.8	3.19×10^{-1}	7.37×10^{-2}	2.79×10^{-1}	2.36×10^{-1}
	0.9	2.75×10^{-1}	7.93×10^{-2}	2.60×10^{-1}	2.01×10^{-1}
	1.0	8.88×10^{-16}	8.88×10^{-16}	8.88×10^{-16}	8.88×10^{-16}
Average		1.21×10^{-1} (100%)	2.56×10^{-2} (21%)	1.03×10^{-1} (85%)	7.35×10^{-2} (61%)
$\sin' x$	0.0	0	0	0	0
	0.1	1.36×10^{-2}	5.72×10^{-3}	5.31×10^{-2}	1.75×10^{-2}
	0.2	2.61×10^{-2}	1.04×10^{-2}	7.58×10^{-2}	1.83×10^{-2}
	0.3	3.67×10^{-2}	1.71×10^{-2}	9.55×10^{-2}	2.49×10^{-2}
	0.4	5.22×10^{-2}	2.08×10^{-2}	9.53×10^{-2}	3.65×10^{-2}
	0.5	5.20×10^{-2}	2.84×10^{-2}	1.11×10^{-1}	4.02×10^{-2}
	0.6	5.88×10^{-2}	3.52×10^{-2}	1.06×10^{-1}	5.09×10^{-2}
	0.7	7.17×10^{-2}	4.03×10^{-2}	8.74×10^{-2}	4.81×10^{-2}
	0.8	5.46×10^{-2}	3.95×10^{-2}	9.38×10^{-2}	5.46×10^{-2}
	0.9	4.21×10^{-2}	3.43×10^{-2}	5.78×10^{-2}	3.82×10^{-2}
	1.0	2.52×10^{-2}	1.74×10^{-2}	2.13×10^{-2}	2.13×10^{-2}
Average		3.94×10^{-2} (100%)	2.27×10^{-2} (58%)	7.24×10^{-2} (184%)	3.19×10^{-2} (81%)
$\cos' x$	0.0	0	0	0	0
	0.1	1.85×10^{-2}	1.85×10^{-2}	3.42×10^{-2}	1.46×10^{-2}
	0.2	2.71×10^{-2}	2.71×10^{-2}	3.50×10^{-2}	2.71×10^{-2}
	0.3	2.98×10^{-2}	4.16×10^{-2}	5.34×10^{-2}	3.77×10^{-2}
	0.4	4.76×10^{-2}	5.05×10^{-2}	5.15×10^{-2}	5.83×10^{-2}
	0.5	5.01×10^{-2}	4.62×10^{-2}	6.97×10^{-2}	5.01×10^{-2}
	0.6	5.31×10^{-2}	5.70×10^{-2}	6.88×10^{-2}	5.31×10^{-2}
	0.7	5.87×10^{-2}	6.26×10^{-2}	7.44×10^{-2}	7.05×10^{-2}
	0.8	5.62×10^{-2}	4.96×10^{-2}	7.19×10^{-2}	6.80×10^{-2}
	0.9	6.47×10^{-2}	4.91×10^{-2}	6.07×10^{-2}	5.69×10^{-2}
	1.0	3.05×10^{-2}	2.83×10^{-2}	2.83×10^{-2}	3.22×10^{-2}
Average		3.97×10^{-2} (100%)	3.91×10^{-2} (98%)	4.98×10^{-2} (125%)	4.26×10^{-2} (107%)

Table 2.9: Experimental evaluations in maximum errors (bit length: 4,095 bits).

Function	x	[1]	[2]	FSR (Proposed)	RRR (Proposed)
x^8	0.0	0	0	0	0
	0.1	1.22×10^{-3}	1.00×10^{-8}	7.33×10^{-4}	4.88×10^{-4}
	0.2	4.39×10^{-3}	2.42×10^{-4}	2.68×10^{-3}	1.46×10^{-3}
	0.3	1.56×10^{-2}	6.67×10^{-4}	8.48×10^{-3}	4.33×10^{-3}
	0.4	3.52×10^{-2}	2.03×10^{-3}	2.11×10^{-2}	1.01×10^{-2}
	0.5	7.28×10^{-2}	3.66×10^{-3}	4.69×10^{-2}	2.52×10^{-2}
	0.6	1.35×10^{-1}	8.11×10^{-3}	9.09×10^{-2}	5.57×10^{-2}
	0.7	2.06×10^{-1}	1.42×10^{-2}	1.47×10^{-1}	9.03×10^{-2}
	0.8	2.71×10^{-1}	2.10×10^{-2}	2.07×10^{-1}	1.31×10^{-1}
	0.9	2.46×10^{-1}	3.61×10^{-2}	1.97×10^{-1}	1.36×10^{-1}
	1.0	8.88×10^{-16}	8.88×10^{-16}	8.88×10^{-16}	8.88×10^{-16}
Average		8.97×10^{-2} (100%)	7.82×10^{-3} (9%)	6.57×10^{-2} (73%)	4.13×10^{-2} (46%)
$\sin' x$	0.0	0	0	0	0
	0.1	6.55×10^{-3}	1.01×10^{-2}	1.66×10^{-2}	5.08×10^{-3}
	0.2	1.14×10^{-2}	1.11×10^{-2}	2.24×10^{-2}	7.70×10^{-3}
	0.3	1.69×10^{-2}	1.08×10^{-2}	2.86×10^{-2}	9.07×10^{-3}
	0.4	2.43×10^{-2}	1.04×10^{-2}	3.26×10^{-2}	1.31×10^{-2}
	0.5	2.89×10^{-2}	1.35×10^{-2}	3.55×10^{-2}	1.45×10^{-2}
	0.6	3.25×10^{-2}	1.40×10^{-2}	3.94×10^{-2}	1.69×10^{-2}
	0.7	3.54×10^{-2}	1.78×10^{-2}	3.37×10^{-2}	2.00×10^{-2}
	0.8	3.07×10^{-2}	1.33×10^{-2}	2.99×10^{-2}	1.97×10^{-2}
	0.9	2.22×10^{-2}	1.47×10^{-2}	2.51×10^{-2}	1.41×10^{-2}
	1.0	6.64×10^{-3}	7.37×10^{-3}	7.52×10^{-3}	6.88×10^{-3}
Average		1.96×10^{-2} (100%)	1.12×10^{-2} (57%)	2.47×10^{-2} (126%)	1.15×10^{-2} (59%)
$\cos' x$	0.0	0	0	0	0
	0.1	4.77×10^{-3}	5.26×10^{-3}	4.28×10^{-3}	3.55×10^{-3}
	0.2	6.93×10^{-3}	9.86×10^{-3}	8.70×10^{-3}	6.68×10^{-3}
	0.3	9.25×10^{-3}	1.51×10^{-2}	1.10×10^{-2}	9.50×10^{-3}
	0.4	1.37×10^{-2}	1.47×10^{-2}	1.48×10^{-2}	1.18×10^{-2}
	0.5	1.57×10^{-2}	1.35×10^{-2}	2.16×10^{-2}	1.34×10^{-2}
	0.6	1.54×10^{-2}	1.81×10^{-2}	2.33×10^{-2}	1.70×10^{-2}
	0.7	2.03×10^{-2}	1.56×10^{-2}	2.10×10^{-2}	1.68×10^{-2}
	0.8	1.81×10^{-2}	1.71×10^{-2}	2.25×10^{-2}	1.64×10^{-2}
	0.9	1.72×10^{-2}	1.65×10^{-2}	2.01×10^{-2}	1.38×10^{-2}
	1.0	8.92×10^{-3}	1.11×10^{-2}	1.09×10^{-2}	8.66×10^{-3}
Average		1.18×10^{-2} (100%)	1.24×10^{-2} (105%)	1.44×10^{-2} (122%)	1.07×10^{-2} (91%)

Table 2.10: Area and delay comparison (bit length: 255 bits).

SN Duplicator	Gate count [gates]	Delay [ns]	Latency [cycles]
[1]	5.75	0.39	0
[2]	114.5	0.50	255
FSR (Proposed)	46.25	0.49	0
RRR (Proposed)	51.25	0.49	0

Table 2.11: Area and delay comparison (bit length: 4,095 bits).

SN Duplicator	Gate count [gates]	Delay [ns]	Latency [cycles]
[1]	5.75	0.39	0
[2]	199.75	0.50	4,095
FSR (Proposed)	87.5	0.49	0
RRR (Proposed)	92.5	0.49	0

2.4.2 Area and Delay Comparison of SN Duplicators

Logic synthesis has been performed using Verilog[35] by Design Compiler version D-2010.03-SP5[36] and STARC 90 nm library[37], and the areas and delays of the four SN duplicators were evaluated: the 1-bit FF-based SN duplicator proposed in [1], the ideal SN duplicator proposed in [2], our FSR duplicator, and our RRR duplicator.

Table 2.10 summarizes the results of Case 1. In this table, “Gate count” shows the required number of equivalent 2-input NAND gates. “Delay” shows the critical path delay in nanoseconds. “Latency” shows the required number of clock cycles to output the first bit of the duplicated SN. From this table, our proposed FSR and RRR duplicators require 9 times more area compared with the SN duplicator in [1] but reduce the area by more than 50% compared with the SN duplicator in [2]. The critical path delays of all the SN duplicators are almost the same. In this case, the SN duplicator in [1], FSR and RRR duplicators generate an output SN in a bit-by-bit manner. Therefore, the latency of these circuits become 0. On the other hand, the SN duplicator in [2] firstly counts the 1’s in the input SN taking 255 clock cycles and then start to generate a new SN. Thus the SN duplicator in [2] requires 255 clock cycles to generate the first bit to output.

Table 2.11 summarizes the results of Case 2. From this table, our proposed FSR and RRR duplicators require 16 times more area compared with the SN duplicator in [1] but reduce the area by more than 50% compared with the SN duplicator in [2]. The critical path delays of all the SN duplicators are almost the same. In this case, the SN duplicator in [1], FSR and RRR duplicators can also generate an output SN in a bit-by-bit manner.

Therefore, the latency of these circuits become 0. On the other hand, the SN duplicator in [2] firstly counts the 1's in the input SN taking 4,095 clock cycles and then start to generate a new SN. Thus the SN duplicator in [2] requires 4,095 clock cycles to generate the first bit to output.

Overall, our FSR and RRR duplicators just introduce around 50 gates or 90 gates but realize more accurate arithmetic operations compared with a conventional approach, without latency alike the SN duplicator in [2]. Note that, to implement our FSR duplicator and RRR duplicator in an actual circuit, a "new" LFSR is not necessarily required to generate a random bit stream if LFSRs already exist in the circuit. Instead, the one of the LFSRs can be reused and consider them to generate a random bit stream [38]. In [38], a single LFSR can generate multiple non-correlated bit streams by circular-shifting its output bits. This shifting requires no additional gates other than the original LFSR, only by changing wires. Assume that a circuit block includes LFSR L_1 to generate a random bit stream. If another circuit block in the same circuit needs another LFSR L_2 to generate a random bit stream, LFSR L_1 can be reused instead of LFSR L_2, without additional gates. Therefore, when multiple SN duplicators are required in a single circuit, some of the LFSRs can be reduced. If the proposed FSR and RRR duplicators do not require LFSRs by reusing an LFSR somewhere in the circuit, their circuit sizes are further reduced to 13 gates and 18 gates, respectively, regardless of the bit length of the input SN, which is comparable to [1]. In the same way, if the SN duplicator in [2] does not require LFSRs, its circuit size is reduced to 81.25 gates and 125.25 gates when the bit length of the input SN is 255 and 4,095, respectively. Still, the circuit sizes of the proposed FSR and RRR duplicators are much smaller than that of the SN duplicator in [2].

2.4.3 Total Latency of the Benchmark Circuits

Table 2.12 summarizes the latencies of x^8, $\sin' x$, and $\cos' x$ circuits when 255-bit SN and 4,095-bit SN is input. In column with 255-bit SN, the SN duplicator in [2], FSR, and RRR include an 8-bit LFSR, and in column with 4,095-bit SN, the SN duplicator in [2], FSR, and RRR include a 12-bit LFSR. "Function" shows the type of function implemented by the benchmark circuit. "SN Duplicator" shows the SN duplicator which is used. "Latency" shows the required number of clock cycles to output the first bit of each function.

From this table, each function using the SN duplicator in [2] requires clock cycles in response to the length of the input SN and the number of SN duplicators in series. On the other hand, each function using the SN duplicator in [1], FSR, or RRR duplicators

Table 2.12: Delay and latency of the benchmark circuits.

Function	SN Duplicator	Latency [cycles]	
		255-bit SN	4,095-bit SN
x^8	[1]	0	0
	[2]	255×3	4,095×3
	FSR	0	0
	RRR	0	0
$\sin' x$	[1]	0	0
	[2]	255×3	4,095×3
	FSR	0	0
	RRR	0	0
$\cos' x$	[1]	0	0
	[2]	255×4	4,095×4
	FSR	0	0
	RRR	0	0

outputs the first bit as soon as the first bit is input, though the initial value in FFs in the SN duplicators are used for calculation of the first several bits.

2.4.4 Discussions

This section discusses the duplication error of the SN duplicators in [1] and [2], the proposed FSR duplicator, and RRR duplicator when the bit lengths of input SNs become longer. Here, we define the duplication error as the difference between the values of the input and output SNs.

SN Duplicator in [1]: The input bits are stored sequentially into the FF as in Figure 2.5. After a bit is stored in the FF, that bit is always output at the next clock cycle. As a whole, the last bit of the input SN will remain in the FF, i.e., the last bit will not be output. Instead of that bit, the bit stored in the FF initially is output at first. Therefore, the erroneous bit at duplication using [1] is no more than one bit. Thus, the maximum duplication error of the SN duplicator in [1] becomes $1/|In|$ which is in inverse proportion to the bit length of the input SN.

SN Duplicator in [2]: Let us assume that the bit length of LFSR used in the SN duplicator in [2] be l, and its cycle be $2^l - 1$. If the length of input SN In is $2^l - 1$, i.e., $|In| = 2^l - 1$, there will be no erroneous bits, regardless of $|In|$. The reason for this is: an LFSR is used as a pseudo random number generator generating integers from 1 to $|In|$, which is compared with S_{In}, the number of 1's in In. This makes the SN duplicator in [2] generate an output SN Out satisfying $|Out| = |In|$ and $S_{Out} = S_{In}$, i.e., $V_{Out} = V_{In}$. Thus, the SN duplicator in [2] has no duplication errors in this case.

FSR duplicator: From Table 2.4, the duplication error of the FSR duplicator becomes smaller with longer bit length of input SN. The reason for this is: the first two bits of output SN may be the initial bits stored in FF_0 and FF_1 and the last two bits of the input may not be output. However, the effect of these bits becomes relatively small compared with tho whole SN if the bit length of input SN is long enough. However, depending on the random bit stream r, the duplication error may become large since some bits from the input SN are not output.

RRR duplicator: Let In_i and r_i be the i-th bit of the input SN In and the random bit stream r, respectively. In an RRR duplicator, In_i is stored in FF_0 if $r_i = 0$, and is output the next time the bit of r becomes 0. If $r_i = 1$, FF_1 is used instead of FF_0. Therefore, excluding the remaining bits in FF_0 and FF_1, all the bits that were once stored in the FFs are output. Instead, the initial bits stored in the two FFs are output at first. This means that: the erroneous bit at duplication using RRR is no more than two bits. Thus, maximum duplication error of RRR is $2/|In|$ which is in inverse proportion to the bit length of the input SN.

Overall, the duplication errors in the SN duplicators in [1], and the proposed RRR duplicator become smaller if the bit length of the input SN becomes longer. The duplication error of the SN duplicator in [2] is always zero.

2.5 Conclusions

This chapter proposed two new SN duplicators, FSR and RRR. The proposed SN duplicators effectively reduce MSE values compared with a conventional 1-bit FF-based SN duplicator. Thus realize a more accurate arithmetic operation.

Chapter 3

Scalable Stochastic Number Duplicators for Accuracy-flexible Arithmetic Circuit Design[1]

3.1 Introduction

3.1.1 Backgrounds

Implementation of arithmetic circuits with SN duplicators have been proposed in Chapter 2 or in [39]. These implementations do have smaller arithmetic errors compared with the conventional method [1] but cannot reduce enough errors with long input SNs. This is due to the re-convergence paths of the circuits causing arithmetic errors, regardless of the length of SNs. Therefore, these circuits do not have flexible accuracy.

3.1.2 Proposal

This chapter aims to regain highly flexible accuracy, which is the nature of SN, in circuits using SN duplicators. Focusing on changing the dependency of the output, this chapter proposes a *scalable* SN duplicator, 2^nRRR. The 2^nRRR duplicator is a unique extension of the RRR duplicator proposed in Chapter 2 or in [39]. By using multiple random bit streams and more FFs (flip-flops), flexible bit re-arrangement becomes available, making every output independent of each other. Through experimental evaluations and discussions, this chapter clarifies that the proposed 2^nRRR duplicator enables accuracy-flexible circuits, confirming that the circuits implemented by the 2^nRRR duplicator can

[1]Technical contents in this chapter have been presented in the publications ⟨1⟩, ⟨8⟩, and ⟨17⟩.

change its accuracies depending on the value of n. From discussions, this chapter theoretically proves that, if the length of the input SN is sufficiently large, increasing n will improve the circuit's accuracy. The only limitation of this proposal is that this SN duplicator has larger circuit area than those of the conventional SN duplicators, still very small compared with those of binary computing.

3.1.3 Contributions

The contributions of this chapter are as follows:

1. This chapter proposes a 2^nRRR duplicator, which uniquely extends the RRR duplicator proposed in Chapter 2 or in [39].
2. This chapter theoretically proves that the 2^nRRR duplicator can flexibly change the accuracies of the arithmetic circuits.
3. The circuits implemented with the 2^nRRR duplicator have less errors compared with those of the circuits implemented by other SN duplicators by selecting the correct instance of 2^nRRR duplicators.

3.1.4 Organization

The rest of this chapter is organized as follows: Section 3.2 introduces duplication of SN and why the conventional SN duplicators fail to change the accuracy. Section 3.3 proposes a 2^nRRR duplicator and compares its hardware cost with that of the conventional SN duplicators. Section 3.4 demonstrates the effectiveness of the 2^nRRR duplicator through experimental evaluations. Section 3.5 discusses the overall errors and areas of circuits using 2^nRRR duplicators and theoretically proves how the 2^nRRR duplicator flexibly changes the accuracy. Section 3.6 gives several concluding remarks.

3.2 Background of SN Duplicators and Re-convergence Paths

3.2.1 Conditions Required for SN duplicators

From discussions in Chapter 2, Conditions 1 and 2 below are required for SN duplicators to output accurate enough value when used in circuits:

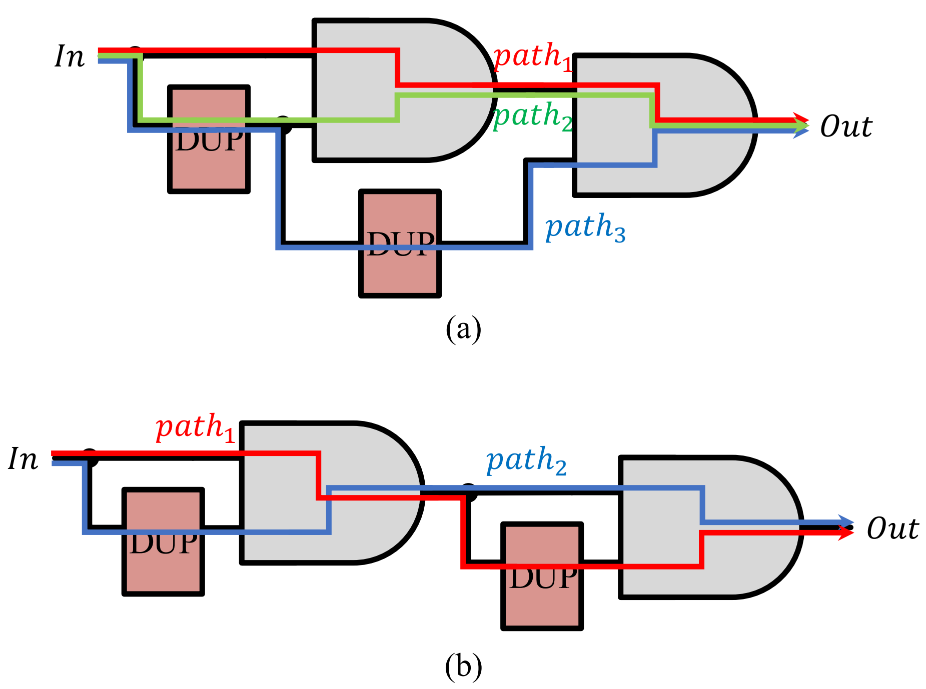


Figure 3.1: Examples of re-convergence paths. (a) Independent. (b) Dependent.

Condition 1. The values of input SN In and output SN Out must be equal ($V_{In} = V_{Out}$) and the bit streams of them differ ($In \neq Out$).

Condition 2. When SN In is input to two SN duplicators with the same circuit, the bit stream of the output SN Out of one SN duplicator differs from that of the output SN Out' of the other SN duplicator.

3.2.2 Dependent and Independent Re-convergence Paths

In this section, we define dependent and independent re-convergence paths, considering arithmetic circuits by SC with re-convergence paths shown in Figure 3.1.

The circuit in Figure 3.1(a) has three signal paths $path_1$, $path_2$, and $path_3$, from the input In to the output Out. $path_1$ includes no DUP's. $path_2$ includes one DUP. $path_3$ includes two DUP's. Since these three signal paths branch off from the same input and join as the same output, they make a re-convergence path. However, the number of DUP's included in the three paths are different. Therefore, when FFs are used as SN duplicators, the bits reffered by each path become different, i.e., the bit streams are *independent*. In this case, we define this re-convergence path as *independent* re-convergence path.

The circuit in Figure 3.1(b) has several signal paths from the input In to the output

Out, one of which includes one DUP ($path_1$) and another one also includes one DUP ($path_2$). Since these three signal paths branch off from the same input and join as the same output, they make a re-convergence path. Also, the number of DUP's passed in these two paths are the same. Therefore, when FFs are used as SN duplicators, the bits reffered by each path become the same, i.e., the bit streams are *dependent*. In this case, we define this re-convergence path as *dependent* re-convergence path.

Considering *dependent* re-convergence paths, Condition 2 is quite necessary for an SN duplicator, since every bit in SNs in these paths are co-related. Thus, the target function cannot be calculated.

3.2.3 The RRR Duplicator

As conventional SN duplicators, [1, 2] have been proposed. However, SN duplicator in [1] does not satisfy Condition 2, and SN duplicator in [2] cannot be used in a practical situation due to its large sized circuit and long latency. To satisfy Condition 2 with a small sized circuit and low latency, we introduced a random bit stream into an SN duplicator. Based on this idea, the RRR (Register based Re-arrangement circuit using a Random bit stream) duplicator is proposed in Chapter 2 (Figure 2.5).

However, the RRR duplicator does not always satisfy Condition 2. This is because: Assuming that the two DUP's in Figure 3.1(b) are implemented by the RRR duplicator, the bit delays of the two DUP's are small and they can be the same with high possibility since the RRR duplicator includes only two FFs. The two RRR duplicators output the same bit with high possibility when the same input SN is given.

3.3 Improved SN Duplicator Based on Bit Re-arrangement

3.3.1 Proposal

This chapter proposes the 2^nRRR duplicator, which is an extended version of the SN duplicator in [1] and the RRR duplicator in Chapter 2 or in [39]. The SN duplicator in [1] is composed of a single FF. Adding one more FF and re-arranging input bits makes the RRR duplicator in Chapter 2 or in [39]. By extending this scheme, the 2^n RRR duplicator shown in Figure 3.2(a) is obtained as follows: While the RRR duplicator is composed of one random bit stream and two FFs, the 2^nRRR duplicator is composed of n random bit streams and 2^n FFs. By adding FFs, the 2^nRRR duplicator can further re-arrange the input SNs. The RU$_m$'s in Figure 3.2(a) are the m-th ($0 \leq m \leq 2^n - 1$)

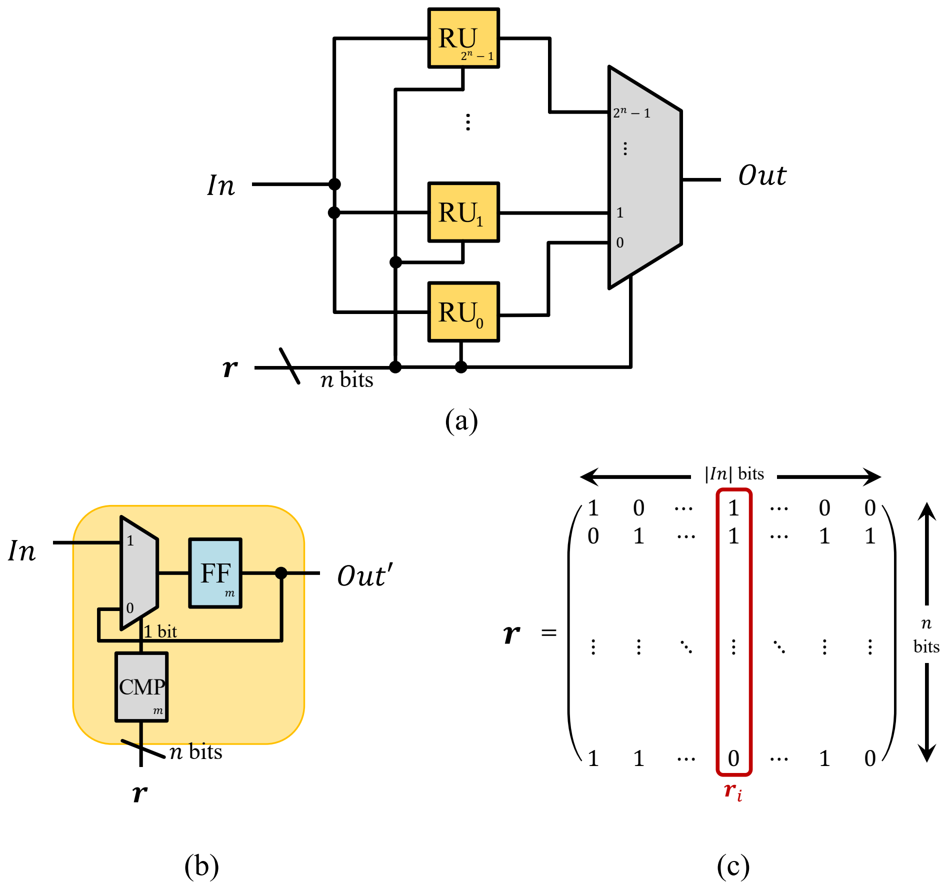


Figure 3.2: 2^nRRR duplicator. (a) Overall model. (b) m-th register unit RU$_m$. (c) n-bit wide $|In|$-bit long random bit stream $\boldsymbol{r}$.

register units (Figure 3.2(b)). Let $\boldsymbol{r}$ be the n-bit width $|In|$-bit long random bit stream and let $\boldsymbol{r}_i$ $(0 \leq \boldsymbol{r}_i \leq 2^n - 1)$ be an i-th n-bit width random bit stream as in Figure 3.2(c). CMP$_m$ in Figure 3.2(b) is a comparator which outputs 1 when $\boldsymbol{r}_i$ is equal to m.

3.3.2 Characteristic

Let In_i be the i-th bit of the input SN In. Let $c_{m,i}$ be the output of the comparator CMP$_m$ of the m-th register unit RU$_m$ when In_i is input. When the i-th bit Out_i of the output SN Out is calculated, the 0/1 function $F^m_{j,i}$ indicates whether In_j is kept in FF_m until In_i comes or not. $F^m_{j,i}$ becomes 1 if In_j is actually stored in FF_m and becomes 0 if another bit of In is stored in FF_m. $F^m_{j,i}$ becomes:

$$F^m_{j,i} = c_{m,j} \times \prod_{k=j+1}^{i-1} (1 - c_{m,k}). \tag{3.1}$$

The output Out_i of the 2^nRRR duplicator becomes:

$$\begin{aligned} Out_i &= \sum_{m=0}^{2^n-1} (c_{m,i} \times R_m \prod_{k=0}^{i-1} (1 - c_{m,k})) \\ &+ \sum_{m=0}^{2^n-1} (c_{m,i} \times \sum_{j=1}^{i-1} (In_j \times F^m_{j,i})). \end{aligned} \tag{3.2}$$

Here, $R_0, R_1, \ldots, R_{2^n-1}$ are the initial bits of $FF_0, FF_1, \ldots, FF_{2^n-1}$, respectively. If all the bits composing the n-bit random bit stream $\boldsymbol{r}_i$ become 0 and 1 in the same possibility of $1/2$, the expected value $E(c_{m,i})$ of $c_{m,i}$ becomes:

$$E(c_{m,i}) = 1/2^n, \tag{3.3}$$

regardless of m and i. Therefore, when i is large enough, the effect of initial bits can be ignored and Equation (3.2) becomes:

$$Out_i = \sum_{m=0}^{2^n-1} (c_{m,i} \times \sum_{j=1}^{i-1} (In_j \times F^m_{j,i})). \tag{3.4}$$

The expected value $E(Out_i)$ of the output Out_i becomes:

$$E(Out_i) = 1/2^n \sum_{j=1}^{i-1} ((1 - 1/2^n)^{j-1} E(In_{i-j})) \approx P_{In} = V_{In}. \tag{3.5}$$

Here, $E(In_{i-j})$ is the expected value of In_{i-j} and i is sufficiently large. Thus, the 2^nRRR duplicator also satisfies Condition 1.

The m-th register unit RU$_m$ of the 2^nRRR duplicator outputs the bit stored in FF_m and stores the input bit into FF_m. Therefore, all the bits stored in the FFs except the ones still remaining in the FFs are output. Hence, the maximum duplicating error becomes $2^n/|In|$, meaning that the longer the input SN is, the smaller the duplicating error becomes.

The 2^nRRR duplicator has a higher possibility satisfying Condition 2, compared with the RRR duplicator. This is because: Assuming that the two DUP's of Figure 3.1(b) are implemented by the 2^nRRR duplicator, the bit delays of the two DUP's is evenly distributed and the possibility of becoming the same is low since the 2^nRRR duplicator includes 2^n FFs.

Table 3.1: Hardware costs of the 2^nRRR duplicators.

SN Duplicator	1RRR	2RRR	4RRR	8RRR
Area [NANDs]	5.75	19	40.5	86.5
Delay [ns]	0.49	0.49	0.49	0.49

3.3.3 Instances of the 2^nRRR and Their Hardware Costs

By setting n to be 0 or 1, the 1RRR and 2RRR duplicators are equivalent to the SN duplicator in [1] and the RRR duplicator in Chapter 2 or in [39], respectively. This chapter focuses on the 4RRR and 8RRR duplicators among with these conventional ones.

Logic synthesis using Verilog[35] has been performed to evaluate the SN duplicators' hardware costs in NAND gates. Table 3.1 summarizes the results, when Design Compiler version D-2010.03-SP5[36] and STARC 90 nm library[37] are used. From these results and the measurements of each element, the area becomes proportional to 2^n and the critical path delay becomes constant.

Note that, to implement 2^nRRR duplicators, a new "LFSR" is not necessarily required for generation of random bit streams. Instead, an LFSR can be reused and considered to generate another random bit stream [38]. [38] shows that one SN generator can share LFSR with other SN generators. In this case, each bit in an LFSR represent an SN with value of 1/2. Based on this idea, an 8-bit LFSR can be shared 7 more times (after its first SN generation) and a 12-bit LFSR can be shared 11 more times. (Here, we focus on 8 and 12-bit LFSR based on the conditions of the experimental evaluations in the next section.) A new "LFSR" is only required when more than 7 (or 11) r's are required in a circuit.

For comparison, we performed logic synthesis of an SN generator and an SN to binary converter. Both of them are essential for all SC-based circuits. An SN generator is composed of an LFSR and a comparator, and an SN to binary converter is composed of a counter. Table 3.2 shows the areas of the two essentials. "Total" shows the total area of an SN generator (an LFSR and a comparator) with optimization. From these results, SN duplicators have relatively small circuit areas compared with these two essentials. Note that, a new "LFSR" is required for an SN duplicator when more than 7 (or 11) r's are required in a circuit, as described in the former paragraph. For further discussions, the areas of the whole circuit of the benchmark circuits shown in Section 3.4.2 are evaluated in Section 3.5.4.

Table 3.2: Areas of SN generators and SN to binary converters.

Circuit	SN generator			SN to binary converter
	LFSR	Comparator	Total	Counter
255 bit SN (8 bit LFSR)	133	166	175	258
4,095 bit SN (12 bit LFSR)	298	211	473	501

* The unit of this table is [NANDs].

Table 3.3: Conditions of accuracy comparison.

Language for simulation	Python 3.6.3[34]
Lengths of input SNs	$\|In\| = 255,\ 4{,}095$
Values of input SNs	$V_{In} = 0.0,\ 0.1,\ \ldots\ ,\ 1.0$
SN Duplicators	DUP = 1RRR [1], 2RRR, 4RRR, 8RRR
Functions	$f(x) = x^2,\ x^8,\ \sin' x,\ \cos' x,\ \tanh' x,\ \exp'(-x^2)$
Number of trials	1,000
Performance indicators	MAE, MSE and maximum error

3.4 Experimental Evaluations

3.4.1 Setup

In this section, the SN duplicators are implemented into arithmetic circuits and their output SNs' values are compared with theoretical values. The conditions for this experiment are summarized in Table 3.3. All the SNs are 255 (4,095)-bit long and are generated by an 8 (12)-bit LFSR. The functions with prime (′) are the approximated functions. The simulation for each value and SN duplicator is done 1,000 times to calculate each performance indicator.

3.4.2 Benchmark Circuits

As benchmark circuits, the following six circuits are used: (1) the squarer as shown in Figure 2.4(b); (2) the eighth power unit; (3) the 7th order approximated sine function denoted by $\sin' x$; (4) the 8th order approximated cosine function denoted by $\cos' x$; (5) the 9th order approximated hyperbolic tangent function denoted by $\tanh' x$; and (6) the 10th order approximated exponential function denoted by $\exp'(-x^2)$. The approximated circuits are constructed based on the Horner's method [33, 1]. Other than the squarer in Figure 2.4, the arithmetic circuits (2)–(6) are constructed as follows:

(2) The x^8 function is expressed by:

$$x^8 = ((x^2)^2)^2. \tag{3.6}$$

Its circuit is shown in Figure 2.6 and includes dependent re-convergence paths.

(3) The $\sin' x$ function is expressed by:

$$\begin{aligned} \sin x &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \\ &= x\left(1 - \frac{x^2}{6}\left(1 - \frac{x^2}{20}\left(1 - \frac{x^2}{42}\right)\right)\right) \\ &= \sin' x. \end{aligned} \tag{3.7}$$

Its circuit is shown in Figure 2.10(a) and includes dependent re-convergence paths.

(4) The $\cos' x$ function is expressed by:

$$\begin{aligned} \cos x &\approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \\ &= 1 - \frac{x^2}{2}\left(1 - \frac{x^2}{12}\left(1 - \frac{x^2}{30}\left(1 - \frac{x^2}{56}\right)\right)\right) \\ &= \cos' x. \end{aligned} \tag{3.8}$$

Its circuit is shown in Figure 2.10(b) and includes dependent re-convergence paths.

(5) The $\tanh' x$ function is expressed by:

$$\begin{aligned} \tanh x &\approx x - \frac{x^3}{3} + \frac{2x^5}{15} - \frac{17x^7}{315} + \frac{62x^9}{2935} \\ &= x\left(1 - \frac{x^2}{3}\left(1 - \frac{2x^2}{5}\left(1 - \frac{17x^2}{42}\left(1 - \frac{62x^2}{153}\right)\right)\right)\right) \\ &= \tanh' x. \end{aligned} \tag{3.9}$$

Its circuit is shown in Figure 3.3(a) and includes dependent re-convergence paths.

(6) The $\exp'(-x^2)$ function is implemented by combining the squarer and the $\exp'(-x)$ function expressed by:

$$\begin{aligned} \exp(-x) &\approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} \\ &= 1 - x\left(1 - \frac{x}{2}\left(1 - \frac{x}{3}\left(1 - \frac{x}{4}\left(1 - \frac{x}{5}\right)\right)\right)\right) \\ &= \exp'(-x). \end{aligned} \tag{3.10}$$

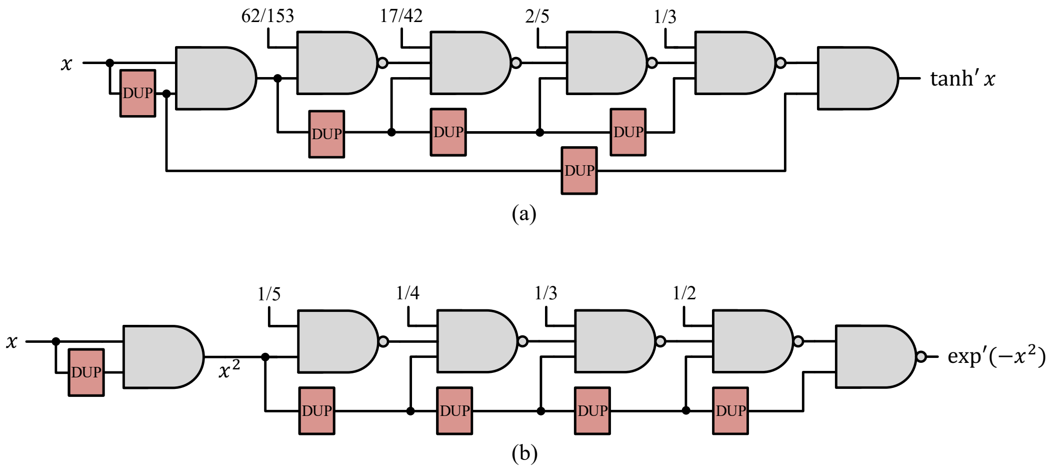


Figure 3.3: Circuits of functions using SN duplicators.
(a) tanh$'\,x$ (b) exp$'(-x^2)$

Its circuit is shown in Figure 3.3(b) and includes dependent re-convergence paths.

3.4.3 Performance Indicators

The mean absolute errors (MAE), the mean square errors (MSE) and the maximum errors are used as performance indicators in this evaluation.

(a) MAE:

The MAE of function f is denoted by:

$$MAE(f,x) = \frac{1}{1{,}000}\sum_{i=1}^{1{,}000} |f_{theory}(x) - f_{actual}(x,i)| \tag{3.11}$$

Here, 1,000 is the number of trials, $f_{theory}(x)$ is the theoretical value when the input value is x, and $f_{actual}(x,i)$ is the actual output value of the i-th trial when each SN duplicator in Figures 2.5, 2.9, 3.2 is used.

(b) MSE:

The MSE of function f is denoted by:

$$MSE(f,x) = \frac{1}{1{,}000}\sum_{i=1}^{1{,}000} (f_{theory}(x) - f_{actual}(x,i))^2. \tag{3.12}$$

(c) Maximum error:

The maximum error of function f is denoted by:

$$MAX(f,x) = \max_{i \in \{1,\ldots,1{,}000\}} |f_{theory}(x) - f_{actual}(x,i)|. \tag{3.13}$$

3.4.4 Results

The results of this experiment are summarized in Table 3.4. The upper value in each cell of the table is the actual value of each performance indicator and the lower value is the value compared to that of 1RRR's. When the length of the input SN is 255, the 2RRR has the smallest or the second smallest values in all the cases. When the length of the input SN is 4,095, the 4RRR generally has smaller errors compared to the 2RRR, except for the maximum errors. The 8RRR generally has larger errors compared with the 2RRR or the 4RRR. The errors of these results are caused by the errors due to initial bits and the errors due to re-convergence paths as described in the next section.

3.5 Discussions

3.5.1 Errors due to Initial Bits

Here, "initial bits" are defined as bits stored in 2^n FFs in a 2^nRRR duplicator before duplication and a "duplication error" is defined as the absolute difference between the value of input SN In and that of output SN Out of an SN duplicator, i.e., $|V_{In} - V_{Out}|$. Note that, erroneous bits, i.e., $|V_{In} - V_{Out}| \times |In|$ is equal to the difference between the numbers of 1's (or 0's) stored in FFs before and after duplication. The "maximum duplication error," which is the maximum error of all possible input SNs and random bit streams, of a 2^nRRR duplicator is $2^n/|In|$. Therefore, in a circuit with N_D 2^nRRR duplicators, the duplication error of the whole circuit becomes up to $2^n/|In| \times N_D$. However, the duplication error will be maximized only when the 2^nRRR duplicator meets the following two conditions:

1. All the initial bits stored in FFs before duplication are the same (0's or 1's).
2. The bits remaining in the FFs after duplication are all 1's or 0's (different from the bits in (1)).

Further, since the maximum error is the inverse proportion to $|In|$, the error will converge to zero when $|In|$ is sufficiently large. The above discussions confirm that the proposed 2^nRRR duplicators satisfy Condition 1 when $|In|$ is sufficiently large compared with 2^n.

Table 3.4: Errors of circuits using SN duplicators.

Length of input		255 bit				4095 bit			
Duplicator		1RRR	2RRR	4RRR	8RRR	1RRR	2RRR	4RRR	8RRR
MAE	x^2	7.61×10^{-3} (100%)	7.69×10^{-3} (101%)	9.97×10^{-3} (131%)	1.39×10^{-2} (183%)	2.20×10^{-3} (100%)	2.20×10^{-3} (100%)	2.61×10^{-3} (119%)	3.25×10^{-3} (148%)
	x^8	8.23×10^{-2} (100%)	3.13×10^{-2} (38%)	4.03×10^{-2} (49%)	8.89×10^{-2} (108%)	7.70×10^{-2} (100%)	1.62×10^{-2} (21%)	1.23×10^{-2} (16%)	1.46×10^{-2} (19%)
	$\sin' x$	1.19×10^{-2} (100%)	7.28×10^{-3} (61%)	7.63×10^{-3} (64%)	1.17×10^{-2} (98%)	9.39×10^{-3} (100%)	2.63×10^{-3} (28%)	2.54×10^{-3} (27%)	3.66×10^{-3} (39%)
	$\cos' x$	1.01×10^{-2} (100%)	9.29×10^{-3} (92%)	1.09×10^{-2} (108%)	1.39×10^{-2} (138%)	2.66×10^{-3} (100%)	2.16×10^{-3} (81%)	2.74×10^{-3} (103%)	3.01×10^{-3} (113%)
	$\tanh' x$	1.06×10^{-2} (100%)	8.30×10^{-3} (78%)	7.80×10^{-3} (74%)	8.59×10^{-3} (81%)	8.51×10^{-3} (100%)	5.81×10^{-3} (68%)	2.33×10^{-3} (28%)	5.02×10^{-3} (59%)
	$\exp'(-x^2)$	1.75×10^{-2} (100%)	8.59×10^{-3} (49%)	6.67×10^{-3} (38%)	1.74×10^{-2} (99%)	1.31×10^{-2} (100%)	3.01×10^{-3} (23%)	2.48×10^{-3} (19%)	6.28×10^{-3} (48%)
Average		2.33×10^{-2} (100%)	1.21×10^{-2} (52%)	1.39×10^{-2} (59%)	2.57×10^{-2} (110%)	1.88×10^{-2} (100%)	5.33×10^{-3} (28%)	4.17×10^{-3} (22%)	5.98×10^{-3} (32%)
MSE	x^2	1.18×10^{-4} (100%)	1.22×10^{-4} (102%)	2.19×10^{-4} (186%)	3.41×10^{-4} (289%)	1.22×10^{-5} (100%)	1.20×10^{-5} (98%)	1.50×10^{-5} (123%)	2.01×10^{-5} (165%)
	x^8	1.50×10^{-2} (100%)	2.66×10^{-3} (17%)	6.62×10^{-3} (44%)	1.57×10^{-2} (105%)	1.38×10^{-2} (100%)	2.56×10^{-3} (19%)	1.80×10^{-3} (13%)	2.40×10^{-3} (17%)
	$\sin' x$	2.90×10^{-4} (100%)	9.01×10^{-5} (31%)	9.57×10^{-5} (33%)	1.80×10^{-4} (62%)	2.23×10^{-4} (100%)	2.60×10^{-5} (11%)	2.45×10^{-5} (11%)	5.13×10^{-5} (23%)
	$\cos' x$	2.08×10^{-4} (100%)	1.67×10^{-4} (80%)	2.35×10^{-4} (113%)	3.37×10^{-4} (162%)	1.79×10^{-5} (100%)	1.22×10^{-5} (68%)	1.75×10^{-5} (98%)	2.33×10^{-5} (130%)
	$\tanh' x$	2.29×10^{-4} (100%)	1.41×10^{-4} (62%)	1.41×10^{-4} (62%)	2.11×10^{-4} (92%)	1.83×10^{-4} (100%)	9.99×10^{-5} (55%)	4.23×10^{-5} (23%)	1.13×10^{-4} (62%)
	$\exp'(-x^2)$	6.27×10^{-4} (100%)	2.29×10^{-4} (36%)	1.94×10^{-4} (31%)	3.32×10^{-4} (53%)	4.32×10^{-4} (100%)	2.69×10^{-5} (6%)	1.30×10^{-5} (3%)	6.48×10^{-5} (15%)
Average		2.55×10^{-3} (100%)	5.52×10^{-4} (22%)	1.25×10^{-3} (49%)	2.86×10^{-3} (112%)	2.64×10^{-3} (100%)	4.73×10^{-4} (18%)	3.19×10^{-4} (12%)	4.45×10^{-4} (17%)
MAX	x^2	1.03×10^{-3} (100%)	1.38×10^{-3} (133%)	2.19×10^{-3} (212%)	3.66×10^{-3} (354%)	7.30×10^{-5} (100%)	7.95×10^{-5} (109%)	1.04×10^{-4} (142%)	1.25×10^{-4} (171%)
	x^8	1.21×10^{-1} (100%)	7.35×10^{-2} (61%)	1.10×10^{-1} (91%)	1.59×10^{-1} (131%)	8.97×10^{-2} (100%)	4.13×10^{-2} (46%)	5.29×10^{-2} (59%)	8.88×10^{-2} (99%)
	$\sin' x$	3.94×10^{-2} (100%)	3.19×10^{-2} (81%)	5.08×10^{-2} (129%)	6.74×10^{-2} (171%)	1.96×10^{-2} (100%)	1.15×10^{-2} (59%)	1.27×10^{-2} (65%)	2.18×10^{-2} (111%)
	$\cos' x$	3.97×10^{-2} (100%)	4.26×10^{-2} (107%)	6.27×10^{-2} (158%)	7.98×10^{-2} (201%)	1.18×10^{-2} (100%)	1.07×10^{-2} (91%)	1.51×10^{-2} (128%)	1.79×10^{-2} (152%)
	$\tanh' x$	3.61×10^{-2} (100%)	3.60×10^{-2} (100%)	3.75×10^{-2} (104%)	5.52×10^{-2} (153%)	2.89×10^{-2} (100%)	2.52×10^{-2} (87%)	1.13×10^{-2} (39%)	2.63×10^{-2} (91%)
	$\exp'(-x^2)$	9.88×10^{-2} (100%)	5.73×10^{-2} (58%)	5.93×10^{-2} (60%)	9.09×10^{-2} (92%)	6.82×10^{-2} (100%)	2.52×10^{-2} (37%)	2.66×10^{-2} (39%)	5.53×10^{-2} (81%)
Average		5.60×10^{-2} (100%)	4.05×10^{-2} (72%)	5.38×10^{-2} (96%)	7.59×10^{-2} (136%)	3.64×10^{-2} (100%)	1.90×10^{-2} (52%)	1.98×10^{-2} (54%)	3.50×10^{-2} (96%)

* Values in parenthesis show relative values compared with 1RRR duplicator.

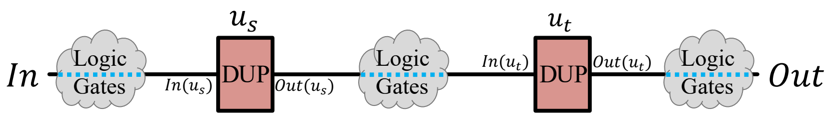


Figure 3.4: Signal path p consisting of two SN duplicators u_s and u_t.

3.5.2 Errors due to Re-convergence Paths

This section discusses the errors due to re-convergence paths. Here, we assume that long enough time has passed and none of the initial bits are used in calculation of the output. Firstly, we define a path delay as follows: As in Figure 3.4, let p be a signal path including several SN duplicators in it. For each path, circuits except the SN duplicators are ignored. The overall input and output among the signal path p are denoted by In and Out, respectively. Let $U(p)$ be a set of SN duplicators in p. In the case of Figure 3.4, $U(p) = \{u_s,\ u_t\}$. The input and output of each SN duplicator $u \in U(p)$ are denoted by $In(u)$ and $Out(u)$, respectively.

Here, we consider that the i-th bit Out_i is output from the path p. Let $\boldsymbol{R}$ be the set of all $\boldsymbol{r}$'s. Assume that the $y(p,u,i,\boldsymbol{R})$-th bit output $Out_{y(p,u,i,\boldsymbol{R})}(u)$ of an SN duplicator $u \in U(p)$, and the $x(p,u,i,\boldsymbol{R})$-th bit input $In_{x(p,u,i,\boldsymbol{R})}(u)$ ($x(p,u,i,\boldsymbol{R}) < y(p,u,i,\boldsymbol{R})$) are used when calculating Out_i. The value of $(y(p,u,i,\boldsymbol{R}) - x(p,u,i,\boldsymbol{R}))$ is called the delay of the SN duplicator u and denoted by $d(p,u,i,\boldsymbol{R})$ (> 0), where

$$Out_{y(p,u,i,\boldsymbol{R})}(u) = In_{x(p,u,i,\boldsymbol{R})}(u) = In_{y(p,u,i,\boldsymbol{R})-d(p,u,i,\boldsymbol{R})}(u). \tag{3.14}$$

Then the path delay $delay(p,i,\boldsymbol{R})$ of the path p when the i-th bit Out_i is output from p is defined by:

$$delay(p,i,\boldsymbol{R}) = \sum_{u \in U(p)} d(p,u,i,\boldsymbol{R}). \tag{3.15}$$

If the target circuit C has several signal paths and their path delay is different from each other at bit output timing i, every signal path is no longer co-related to any other signal paths. Therefore, errors due to dependent and independent re-convergence paths can be minimized. Note that, if an SN duplicator satisfies Condition 2, all the signal paths in a target circuit are not co-related to any other signal paths. Here, this chapter discusses on the delays of the signal paths.

The proposed 2^nRRR duplicators satisfy the following powerful theorem:

Theorem 1. *Let C be a single-input and single-output circuit with a 2^nRRR duplicator for each fan-out and m be the total number of 2^nRRR duplicators in C. In this case, the*

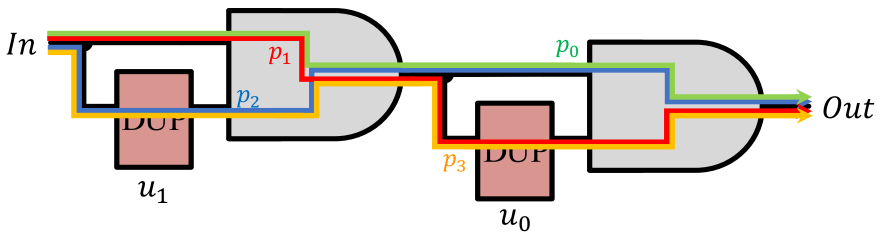


Figure 3.5: Example of all re-convergence paths in a circuit with two SN duplicators.

probability that any two paths of the circuit C having the same path delays converges to zero, if n is sufficiently large.

Proof. Assume that the circuit C has a 2^nRRR duplicator for each fan-out and let m be the total number of 2^nRRR duplicators. The m 2^nRRR duplicators are named u_0, u_1, ..., u_{m-1}, so that the indices are aligned in descending order in every path. Each path can be said to include or exclude an SN duplicator u which can be any of the m 2^nRRR duplicators. Therefore, circuit C has a maximum of 2^m signal paths, since each fan-outs have an SN duplicator and no paths contain the exact same set of SN duplicators. The 2^m paths are named p_0, p_1, ..., p_{2^m-1} so that all of them satisfy the following: For path p_j $(0 \leq j \leq 2^m - 1)$, if j is represented by binary digits and the k-th bit (counting from least significant bit, $0 \leq k \leq m - 1$) is 1, the path p_j includes the SN duplicator u_k. Let $U(p_j)$ be the set of SN duplicators included in the path p_j.

For example, Figure 3.5 shows a circuit of $m = 2$. In this case, there are four paths, p_0, p_1, p_2, p_3. The path p_0 includes no SN duplicators, while the paths p_1 and p_2 include the SN duplicators u_0 and u_1, respectively, and the path p_3 includes both the SN duplicators u_0 and u_1. Also as an example, Table 3.5 shows the case of $m = 3$. In this case, the circuit C has a maximum of $2^3 = 8$ signal paths. Here, $U(p_0) = \emptyset$ and $U(p_5) = \{u_2, u_0\}$.

Let P be the set of all possible signal paths that a circuit with m 2^nRRR duplicators can have, i.e., $P = \{p_0, p_1, \ldots, p_{2^m-1}\}$. Here, note that all circuits with m 2^nRRR duplicators do not necessarily have all the paths included in P. However, all the paths in any circuit with m 2^nRRR duplicators are included in P. For example, the circuit in Figure 3.1(a) only has 3 paths, but based on the definition above, paths $path_1$, $path_2$ and $path_3$ in Figure 3.1(a) are named p_0, p_2, $p_3 \in P$, respectively. Therefore, if this theorem can proved for all paths in P, same can be applied to all circuits with m 2^nRRR duplicators, meaning that Theorem 1 holds in general.

Now, when the i-th bit Out_i is output from the paths, for any $p_s, p_t \in P$, we prove that $delay(p_s, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R})$ holds asymptotically if $p_s \neq p_t$ and n is sufficiently

Table 3.5: Correspondence table of path p_j and SN duplicator u_k ($m = 3$).

Signal paths	SN Duplicators		
	u_2	u_1	u_0
p_0	–	–	–
p_1	–	–	✓
p_2	–	✓	–
p_3	–	✓	✓
p_4	✓	–	–
p_5	✓	–	✓
p_6	✓	✓	–
p_7	✓	✓	✓

large, by mathematical induction.

Initial Step. When $m = 1$, P is $\{p_0, p_1\}$. Since $delay(p_0, i, \boldsymbol{R}) = 0$ and $delay(p_1, i, \boldsymbol{R}) = d(p_1, u_0, i, \boldsymbol{R}) > 0$, these two paths never have the same path delay. Therefore, when $m = 1$, $delay(p_0, i, \boldsymbol{R}) \neq delay(p_1, i, \boldsymbol{R})$.

Inductive Step. The inductive assumption is: When $m = k$, $delay(p_s, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R})$ holds asymptotically if $p_s \neq p_t$ ($0 \leq s,\ t < 2^k$) and n is sufficiently large. In the inductive step, when $m = k + 1$, we prove that:

$$delay(p_s, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R}) \tag{3.16}$$

holds asymptotically, if $p_s \neq p_t$ ($0 \leq s,\ t < 2^{k+1}$) and n is sufficiently large.

For example, when $k = 2$, we assume that $delay(p, i, \boldsymbol{R})$ differs asymptotically for each $p \in \{p_0,\ p_1,\ p_2,\ p_3\}$ when n is sufficiently large. In the case of Table 3.5, we assume that the signal paths in its upper half have the different delays. Then we prove that $delay(p, i, \boldsymbol{R})$ differs asymptotically for each $p \in \{p_0, \ldots, p_7\}$ when n is sufficiently large.

<u>(a) When $s < 2^k$ and $t < 2^k$:</u>

From the inductive assumption, $delay(p_s, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R})$ holds asymptotically when n is sufficiently large.

<u>(b) When $s \geq 2^k$ and $t < 2^k$, or, $s < 2^k$ and $t \geq 2^k$:</u>

Without loss of generality, we assume $s \geq 2^k$ and $t < 2^k$. From Equation (3.15), the following equation holds when $2^k \leq j < 2^{k+1}$:

$$delay(p_j, i, \boldsymbol{R}) \ = \ delay(p_{j-2^k}, i, \boldsymbol{R}) + d(p_j, u_k, i, \boldsymbol{R}). \tag{3.17}$$

For example, when $k = 2$, $delay(p_5, i, \boldsymbol{R}) = delay(p_1, i, \boldsymbol{R}) + d(p_5, u_3, i, \boldsymbol{R})$. By substituting Equation (3.16) for Equation (3.17), we prove that the following holds asymptotically when n is sufficiently large:

$$\begin{aligned} delay(p_t, i, \boldsymbol{R}) &\neq delay(p_s, i, \boldsymbol{R}) \\ &= delay(p_{s-2^k}, i, \boldsymbol{R}) + d(p_s, u_k, i, \boldsymbol{R}), \end{aligned} \tag{3.18}$$

which can be re-written by:

$$d(p_s, u_k, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R}) - delay(p_{s-2^k}, i, \boldsymbol{R}). \tag{3.19}$$

Here, we prove that Equation (3.19) holds asymptotically when n is sufficiently large. As similarly as calculating coefficients of Equation (3.5), the probability P_d of $d(p_s, u_k, i, \boldsymbol{R})$ being a particular value e ($e \geq 1$) becomes:

$$P_d = \frac{1}{N}\left(1 - \frac{1}{N}\right)^{e-1}, \tag{3.20}$$

regardless of p_s and u_k, where $N = 2^n$ and the initial bits can be ignored.

If the right side of Equation (3.19) is zero or negative, Equation (3.19) holds since $d(p_s, u_k, i, \boldsymbol{R}) > 0$. If the right side of Equation (3.19) is positive, the possibility P_{nb} that $d(p_s, u_k, i, \boldsymbol{R}) = delay(p_t, i, \boldsymbol{R}) - delay(p_{s-2^k}, i, \boldsymbol{R})$, i.e., Equation (3.19) does not hold, becomes:

$$P_{nb} = \frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay(p_t,i,\boldsymbol{R}) - delay(p_{s-2^k},i,\boldsymbol{R}) - 1}. \tag{3.21}$$

When $N = 2^n$ is sufficiently large, P_{nb} approaches:

$$\lim_{N\to\infty} P_{nb} = \lim_{N\to\infty} \frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay(p_t,i,\boldsymbol{R}) - delay(p_{s-2^k},i,\boldsymbol{R}) - 1} = 0. \tag{3.22}$$

Based on the discussions above, Equation (3.19) holds asymptotically when n is sufficiently large.

<u>(c) When $s \geq 2^k$ and $t \geq 2^k$:</u>

By substituting Equation (3.16) for Equation (3.17), we prove that the following holds asymptotically when n is sufficiently large:

$$\begin{aligned} delay(p_s, i, \boldsymbol{R}) &\neq delay(p_t, i, \boldsymbol{R}) \\ delay(p_{s-2^k}, i, \boldsymbol{R}) + d(p_s, u_k, i, \boldsymbol{R}) &\neq delay(p_{t-2^k}, i, \boldsymbol{R}) + d(p_t, u_k, i, \boldsymbol{R}), \end{aligned} \tag{3.23}$$

which can be re-written by:

$$d(p_s, u_k, i, \boldsymbol{R}) - d(p_t, u_k, i, \boldsymbol{R}) \neq delay(p_{t-2^k}, i, \boldsymbol{R}) - delay(p_{s-2^k}, i, \boldsymbol{R}). \tag{3.24}$$

Here, we prove that Equation (3.24) holds asymptotically when n is sufficiently large. Without loss of generality, we assume $delay(p_{t-2^k}, i, \boldsymbol{R}) - delay(p_{s-2^k}, i, \boldsymbol{R}) = \alpha > 0$. The possibility P_{nc} that Equation (3.24) does not hold can be calculated as follows: Let $delay$ be the value of $d(p_t, u_k, i, \boldsymbol{R})$. Equation (3.24) will not hold when $d(p_s, u_k, i, \boldsymbol{R}) = delay + \alpha$. The possibility P_s of $d(p_s, u_k, i, \boldsymbol{R})$ being $(delay + \alpha)$ becomes:

$$P_s = \frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay+\alpha-1}. \tag{3.25}$$

The possibility P_t of $d(p_t, u_k, i, \boldsymbol{R})$ being $delay$ becomes:

$$P_t = \frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay-1}. \tag{3.26}$$

Since $delay$ can range from 1 to ∞, the possibility P_{nc} that Equation (3.24) does not hold becomes:

$$\begin{aligned} P_{nc} &= \sum_{delay=1}^{\infty} (P_s \times P_t) \\ &= \sum_{delay=1}^{\infty} \left(\frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay+\alpha-1} \times \frac{1}{N}\left(1 - \frac{1}{N}\right)^{delay-1} \right) \\ &= \frac{1}{2N-1}\left(1 - \frac{1}{N}\right)^{\alpha}. \end{aligned} \tag{3.27}$$

When $N = 2^n$ is sufficiently large, P_{nc} approaches:

$$\lim_{N\to\infty} P_{nc} = \lim_{N\to\infty} \frac{1}{2N-1}\left(1 - \frac{1}{N}\right)^{\alpha} = 0. \tag{3.28}$$

Based on the discussion above, Equation (3.24) holds asymptotically when n is sufficiently large.

From (a)–(c), if n is sufficiently large, Equation (3.16) holds asymptotically, when $m = k + 1$.

Conclusion. By the principle of mathematical induction, for any $p_s, p_t \in P$, $delay(p_s, i, \boldsymbol{R}) \neq delay(p_t, i, \boldsymbol{R})$ holds asymptotically if $p_s \neq p_t$, when n is sufficiently large, i.e., the paths of the circuit C asymptotically have the different path delays, if n is sufficiently large. □

From Theorem 1, 2^nRRR duplicators make all the signal paths in a target circuit asymptotically have different delays when n is large enough. Hence, errors due to re-convergence paths can be minimized.

3.5.3 Overall Errors

From Sections 3.5.1 and 3.5.2, errors in stochastic arithmetic circuits implemented by SN duplicators can divided into errors due to initial bits and errors due to re-convergence paths. When n of 2^nRRR duplicators becomes larger, errors due to re-convergence paths become smaller (see Section 3.5.2) while errors due to initial bits become larger (see Section 3.5.1). Therefore, when the bit length of the input SNs become longer, errors of the circuits using SN duplicators tends to become smaller by using larger 2^nRRR duplicators.

Since both errors cannot be evaluated quantitatively, it is impossible to decide which 2^nRRR duplicator makes the whole circuit's error smaller by calculation. However, from the experimental evaluations in Table 3.4, the 2RRR duplicator tends to have the smallest errors when the bit length is set to be 255 bits, and the 4RRR duplicator tends to have the smallest errors when the bit length is set to be 4,095 bits. The 8RRR duplicator had the smallest errors only for x^8. If 2^n is sufficiently large, only the errors due to initial bits remains, i.e., the errors will be inverse proportional to the length of the input SN. To estimate the errors of other functions and evaluate the best SN duplicator, experiment for each circuit is required.

3.5.4 Circuit Areas of Benchmark Circuits

The circuit areas of benchmark circuits are shown in Table 3.6. The circuits implemented by SN duplicators include an SN generator and an SN to binary converter. The circuits implemented by binary computing are synthesised from the equations of benchmark circuits as in Section 3.4.2, with the precision of 8 or 12 bit binaries (256 or 4096 steps). The values in parenthesis shows the proportion of area of the SN duplicators (including additional LFSRs if required) among the whole circuit size. The results show that if the appropriate SN duplicator as in Section 3.5.3 (2RRR for 255 bit SNs and 4RRR for 4,095 bit SNs) is selected, the area overhead will become about one-third or less compared with the whole circuit.

3.6 Conclusions

This chapter proposed an SN duplicator, 2^nRRR. The proposed SN duplicator was proved to flexibly reduce the errors depending on the required accuracy.

Table 3.6: Areas of circuits using SN duplicators.

Length of input	255 bit					4,095 bit				
SN Duplicator	1RRR	2RRR	4RRR	8RRR	Binary	1RRR	2RRR	4RRR	8RRR	Binary
x^2	439.75	453	474.5	653.5	2239	980.75	994	1015.5	1061.5	10537
	(1.3%)	(4.2%)	(8.5%)	(34%)	(–)	(0.59%)	(1.9%)	(4.0%)	(8.1%)	(–)
x^8	453.25	493	690.5	1094.5	6717	994.25	1034	1396.5	1832.5	31611
	(3.8%)	(12%)	(37%)	(60%)	(–)	(1.7%)	(5.5%)	(30%)	(47%)	(–)
$\sin' x$	461	647	866	1316	16285	1002	1055	1439	1921	74689
	(5.0%)	(32%)	(49%)	(67%)	(–)	(2.3%)	(7.2%)	(32%)	(49%)	(–)
$\cos' x$	461	647	866	1316	18728	1002	1055	1439	1921	85536
	(5.0%)	(32%)	(49%)	(67%)	(–)	(2.3%)	(7.2%)	(32%)	(49%)	(–)
$\tanh' x$	467.75	667	907.5	1536.5	20967	1008.75	1075	1480.5	2306.5	96073
	(6.1%)	(34%)	(52%)	(71%)	(–)	(2.9%)	(8.8%)	(34%)	(58%)	(–)
$\exp'(-x^2)$	467.75	667	907.5	1536.5	21171	1008.75	1075	1480.5	2306.5	96383
	(6.1%)	(34%)	(52%)	(71%)	(–)	(2.9%)	(8.8%)	(34%)	(58%)	(–)

* The unit of this table is [NANDs]. Values in parenthesis show proportional areas of SN duplicators.

Chapter 4

Hardware Implementation of Step Function in Stochastic Computing and Its Applications[1]

4.1 Introduction

4.1.1 Backgrounds

With the rise of neural network and image processing, implementation of activation functions are becoming popular. However, steep functions and discontinuous functions are examples of arithmetic functions that are difficult to implement in SC due to the nature of SNs. Of the activation functions, the Rectified Linear Unit (ReLU) function, the step function, and their composite functions appear as steep functions and discontinuous functions. Implementation of steep functions and discontinuous functions is indispensable for the practical application of SC. To solve the problems of the conventional methods, this chapter firstly proposes hardware implementation of step function using SNs. The proposing circuit of utilizes flip-flops and an adder to perform as step function uniquely calculating the stored bits in the flip-flops. This chapter confirms that the proposed circuit behaves as a step function through experimental evaluations. Also as an application, steep functions or discontinuous functions can be realized by applying the discontinuity of the step function. As a steep function, this chapter also proposes hardware implementation of absolute function and discontinuous function, by synthesizing an arbitrary function as a discontinuous function.

[1]Technical contents in this chapter have been presented in the publications ⟨10⟩ and ⟨12⟩.

4.1.2 Proposal

To solve the problems of the conventional methods, this chapter firstly proposes hardware implementation of step function using SNs. The proposing circuit of utilizes flip-flops and an adder to perform as step function uniquely calculating the stored bits in the flip-flops. This chapter confirms that the proposed circuit behaves as a step function through experimental evaluations. Also as an application, steep functions or discontinuous functions can be realized by applying the discontinuity of the step function. As a steep function, this chapter also proposes hardware implementation of absolute function and discontinuous function, by synthesizing an arbitrary function as a discontinuous function. The calculation accuracy is evaluated for the composite function of the implemented absolute value function and trigonometric function.

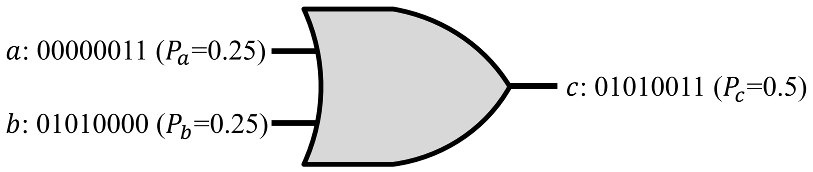
4.1.3 Contributions

The contributions of this chapter are as follows:

1. This chapter proposes hardware implementation of step function in SC.
2. This chapter theoretically proves that the above circuit converges to step function by adding additional gates.
3. This chapter proposes hardware implementations of absolute function and discontinuous function using the circuit of step function.
4. This chapter confirms that the circuits of step function, absolute function, and discontinuous function actually perform as target function through experimental evaluations.

4.1.4 Organization

The rest of this chapter is organized as follows: Section 4.2 introduces related works, especially ones related to step function. Section 4.3 proposes a hardware implementation of step function and discusses its characteristics. Section 4.4 proposes a hardware implementation of absolute function and discusses its characteristics. Section 4.5 proposes a hardware implementation of discontinuous function and discusses its characteristics. Section 4.6 demonstrates the effectiveness of the three proposed circuits through experimental evaluations. Section 4.7 gives several concluding remarks.



a: 00000011 (P_a=0.25)
b: 01010000 (P_b=0.25)
c: 01010011 (P_c=0.5)

Figure 4.1: Example of SN addition using an OR gate.

4.2 Related Works

4.2.1 Additional Arithmetic Operation Circuits Using SNs

As discussed in Chapter 2, in uni-polar expressed SC, multiplication, weighted addition, and inversion are implemented by an AND gate, a MUX circuit and a NOT gate, respectively. Here, addition in uni-polar expressed SC and multiplication in bi-polar SC are introduced.

An OR (logical sum) gate is used for addition. Let a and b be the two input SNs and c be the output SN of an OR gate. V_c is represented by:

$$V_c = P_c = P_a + P_b - P_a \times P_b = V_a + V_b - V_a V_b. \tag{4.1}$$

In Figure 4.1, by adding a = 00000011 and b = 01010000 ($V_a = V_b = 0.25$), c = 01010011 ($V_c = 0.5$) is obtained by performing OR operation between them. Note that, an OR operation will correctly perform as an adder, only when $V_a \times V_b$ is small enough.

On the other hand, in bi-polar expressed SC, multiplication, weighted addition, and inversion are implemented by an XNOR gate, a MUX circuit and a NOT gate, respectively [6]. In particular, an XNOR (exclusive non-disjunction) gate is used for addition. Let a and b be the two input SNs and c be the output SN of an XNOR gate. V_c is represented by:

$$\begin{aligned} V_c &= 2 \times P_c - 1 \\ &= 2 \times \{1 - (P_a \times (1 - P_b) \\ &\quad + P_b \times (1 - P_a))\} - 1 \\ &= (2 \times P_a - 1) \times (2 \times P_b - 1) \\ &= V_a \times V_b. \end{aligned} \tag{4.2}$$

For example, in Figure 4.2, by multiplying a = 00111111 ($V_a = 0.5$) and b = 01010101 ($V_b = 0$), c = 10010101 ($V_c = 0$) is obtained by performing XNOR operation between them.

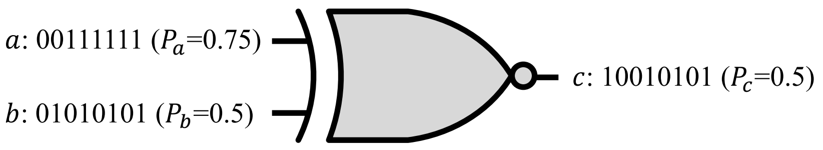

a: 00111111 (P_a=0.75)
b: 01010101 (P_b=0.5)
c: 10010101 (P_c=0.5)

Figure 4.2: Example of bi-polar SN multiplication using an XNOR gate.

4.2.2 Conventional Implementation of Activation Functions

As functions related to step function, this section introduces conventional implementation of activation functions.

In [11], analog random noise is generated and used to express step function, Sigmoid function, and ReLU function, etc. This method generates SNs from the value of the weighted sums in neurons, but has a problem that SNs cannot be input. Further, additional circuits such as a random noise generation circuit and a digital-to-analog conversion circuit are required.

In [9], neurons that introduce SC-based tanh functions and ReLU functions are designed and optimized. Similar to [11], this method also generates SN from the value of the weighted sum in the neuron, and cannot take SNs as inputs. Step function is not been discussed in this reference.

In [10], a Sigmoid function is implemented in a small circuit area in exchange for the accuracy. Also, Chapter 3, the tanh function is implemented with high accuracy using the SN duplicator. However, the problem with these methods is that only functions that are differentiable can be implemented. Since the step function is discontinuous at $x = 0$, that is, it cannot be differentiated, it cannot be implemented by these methods.

Thus, no SC circuit that expresses the step function alone has been proposed so far.

4.2.3 Conversion of SNs to Binary Numbers

To convert an SN x to a binary number, the number of 1's S_x in x, in Equation (2.1), is required. If x is generated by an m-bit LFSR, its bit length $|x|$ becomes $2^m - 1$. Figure 4.3 is the circuit counting the number of 1's in x with bit length $|x| = 2^m - 1$. In this circuit, "m-bit counter" counts the number of 1's S_x in x out of $2^m - 1$ bits. By using this S_x and $2^m - 1$, the value V_x of x can be obtained.

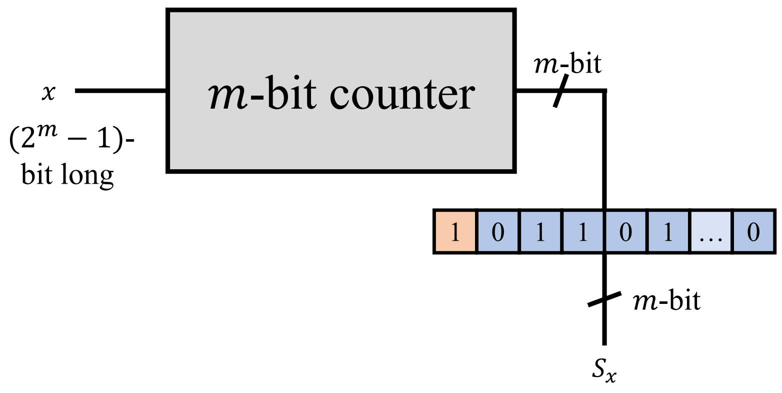


Figure 4.3: Circuit converting SNs to binary numbers.

4.3 Hardware Implementation of Step Function in SC

To solve the problem of hardware implementation of the conventional activation function shown in Section 4.2.2, this chapter proposes a step function circuit based on the circuit that converts SN to binary number shown in Section 4.2.3. We also discuss the input/output relationship, circuit hardware cost, and technical advantages.

4.3.1 Proposal

A step function outputs 0 or 1 depending on its input. Here, we consider implementing a step function that uses SN as input and output. For example, when 1/2 is set as the threshold value, the step function is $f(x)$ becomes as follows:

$$f(x) = \begin{cases} 1, & (x \geq 1/2) \\ 0, & (x < 1/2). \end{cases} \tag{4.3}$$

To obtain an accurate output, comparing the output with the threshold value after the binary conversion introduced in Section 4.2.3 is required For example, if 1/2 is set as the threshold value, the MSB (Most Significant Bit, colored in red) value of the counter in the Figure 4.3 should be output $|x|$ times after counting all the bits in SN x. However, bit-by-bit operations, which is an advantage of SC becomes impossible with such an implementation. Therefore, this section proposes a step function circuit using SN.

The proposing circuit of step function is shown in Figure 4.4. This circuit substituted FFs for the counters in Figure 4.3. The circuit in Figure 4.4 keeps the last $2^n - 1$ from the bit stream of the input SN x, counts the number of 1's in them, and output the MSB as output SN y.

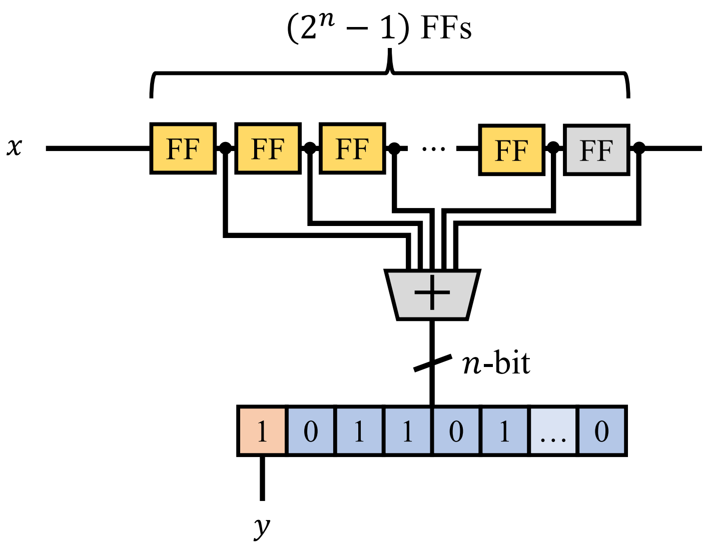


Figure 4.4: Hardware implementation of step function in SC.

Here, we assume that all the bits stored in FFs are the bits of the input SN x. A bit of output SN y becomes 1 when the majority of the bits stored in N FFs is 1. Therefore, assuming that the appearance rate of 1's in the bit stream of input SN x is p, the expected value $E(P_y)$ of appearance rate P_y of 1's in output SN y becomes:

$$E(P_y, N, p) = \sum_{i=\frac{N+1}{2}}^{N} ({}_N\mathrm{C}_i \times p^i (1-p)^{N-i}), \tag{4.4}$$

where $N = 2^n - 1$. In Figure 4.5, the appearance rate of 1's p of the bit stream of input SN x is on the horizontal axis, and the expected value $E(P_y)$ of the appearance rate of 1's in output SN y is on the vertical axis. In this graph, $n = 1, 2, 3, \ldots, 10$, i.e., $N = 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023$. From this graph, n or N become larger, the output becomes the closer to the step function. The expected formula of the step function is expressed as follows:

$$P_y = \begin{cases} 1, & (\frac{1}{2} < P_x \leq 1) \\ 0, & (0 \leq P_x < \frac{1}{2}) \\ \frac{1}{2}, & (P_x = \frac{1}{2}). \end{cases} \tag{4.5}$$

This equation can be proved as follows:

Theorem 2. *The expected value of the output SN y of the step function circuit in Figure 4.4 converges to Equation (4.5), when N becomes large.*

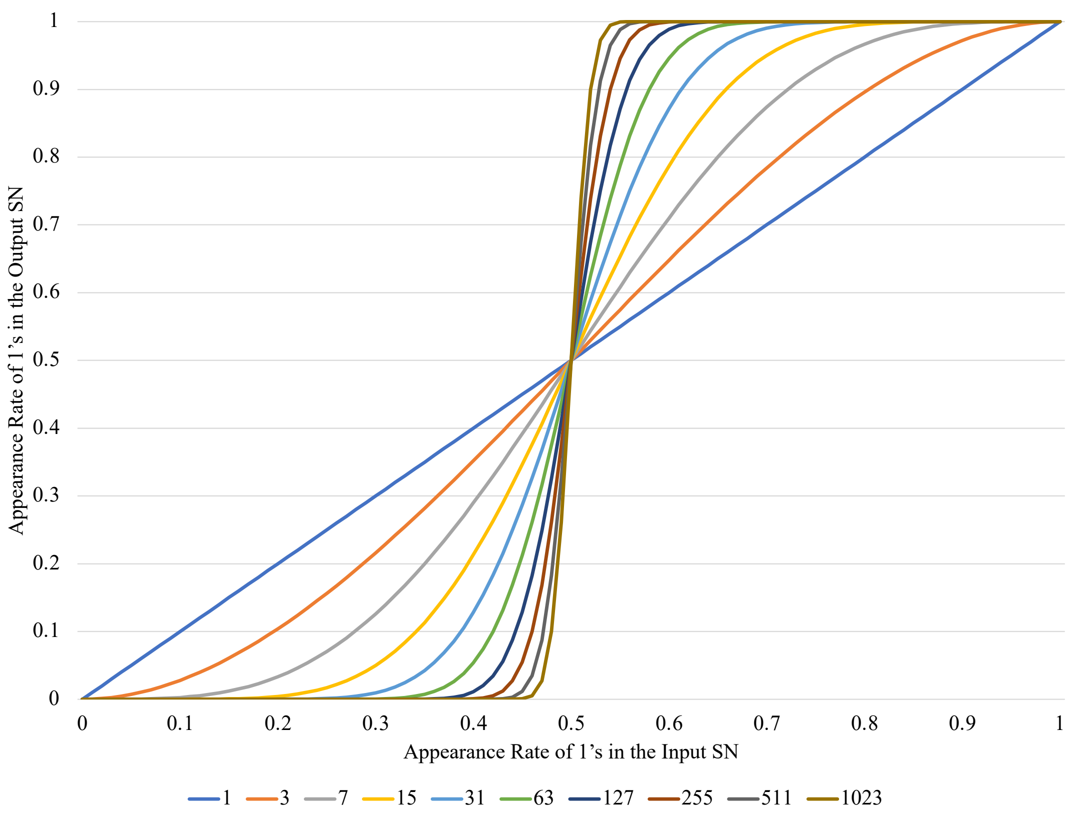


Figure 4.5: Theoretical values of input/output SN of a step function circuit.

Proof. Here, we prove that:

$$\lim_{N\to\infty} E(P_y, N, p) = \begin{cases} 1, & (\frac{1}{2} < p \leq 1) \\ 0, & (0 \leq p < \frac{1}{2}) \\ \frac{1}{2}, & (p = \frac{1}{2}), \end{cases} \tag{4.6}$$

where p is the appearance rate of 1's in the input SN.
(a) When $\frac{1}{2} < p \leq 1$:
From law of large numbers, the number of 1's in the FFs in Figure 4.4 converges Np when $N \to \infty$. Since p is larger than $\frac{1}{2}$, $Np > \frac{N}{2}$ holds, and the MSB in Figure 4.4 converges to 1. Therefore,

$$\lim_{N\to\infty} E(P_y, N, p) = 1. \tag{4.7}$$

(b) When $0 \leq p < \frac{1}{2}$:

In the same way as (a), the following equation holds:

$$\lim_{N\to\infty} E(P_y, N, p) = 0. \tag{4.8}$$

<u>(c) When $p = \frac{1}{2}$:</u>

By substituting $1 - p$ for p in Equation (4.4),

$$\begin{aligned} E(P_y, N, 1-p) &= \sum_{i=\frac{N+1}{2}}^{N} ({}_N\mathrm{C}_i \times (1-p)^i p^{N-i}) \\ &= \sum_{i=0}^{\frac{N-1}{2}} ({}_N\mathrm{C}_i \times p^i (1-p)^{N-i}) \\ &= \sum_{i=0}^{N} ({}_N\mathrm{C}_i \times p^i (1-p)^{N-i}) - \sum_{i=\frac{N+1}{2}}^{N} ({}_N\mathrm{C}_i \times p^i (1-p)^{N-i}) \\ &= 1 - E(P_y, N, p) \end{aligned} \tag{4.9}$$

holds. Since Equation (4.4) is differentiable, it is point symmetry at $p = \frac{1}{2}$ and its expected value becomes $E(P_y, N, \frac{1}{2}) = \frac{1}{2}$.

<u>From (a)–(c),</u>

$$\lim_{N\to\infty} E(P_y, N, p) = \begin{cases} 1, & (\frac{1}{2} < p \le 1) \\ 0, & (0 \le p < \frac{1}{2}) \\ \frac{1}{2}, & (p = \frac{1}{2}) \end{cases} \tag{4.10}$$

holds. □

The concept of this proposal is to output the first bit as soon as the first bit of SN is input. Therefore, the initial bits stored in FF are also used to obtain the output of the first N bits. Here, we consider minimizing the error due to the initial bits. 1's and 0's are stored with a probability of 1/2 for each of the $(N - 1)$ FFs shown in yellow in the Figure 4.4. The output of the i-th $(1 \le i < N)$ is calculated based on the first i bits of the input SN and the $(N - i)$ bits of the initial bits. Therefore, if the expected value of the initial bits in FFs is 1/2, the majority bit of the first i bits of the input SN is output. Note that, the FF shown in gray in Figure 4.4 is overwritten when the first bit of the input SN is input.

Table 4.1: The hardware cost of the step function circuit using SN.

N [FFs]	Delay [ns]	Area [NANDs]
1	0.39	5.75
3	0.39	67
7	0.45	158
15	0.50	297

4.3.2 Hardware Costs

The hardware cost of the step function circuit using SN was obtained by logic synthesis, and its results are shown in Table 4.1. Logic synthesis was performed using Verilog[35] by Design Compiler version D-2010.03-SP5[36] and the STARC 90nm library[37]. Except for $N = 1$, which requires no adders, the circuit area became around $20N$ gates in NAND gates conversion. The delay increased as N increased since the addition became more complicated.

In this chapter, we have not evaluated the step function circuit in binary computing. This is because comparisons between binary computing and SC is impossible under the same conditions. In the implementation using binary numbers, the circuit of step function is uniquely determined by the maximum bit length of the input binary number, and the error always becomes 0. On the other hand, in the implementation using SN, the circuit of step function is determined by the permissible error, regardless of the bit length of the input SN.

4.3.3 Technical Merits

Implementation of step function using SN has the following technical merits:

1. By changing the circuit, step functions with arbitrary threshold can be implemented.

2. New functions can be expressed by combining with other circuits.

For the first merit, threshold can be simply changed by modifying the output bit. For example, by outputting XOR of the two MSB's (see Figure 4.6), its output's expected value becomes:

$$P_y = \begin{cases} 1, & (\frac{1}{4} < P_x < \frac{3}{4}) \\ 0, & (0 \leq P_x < \frac{1}{4},\ \frac{3}{4} < P_x \leq 1), \end{cases} \tag{4.11}$$

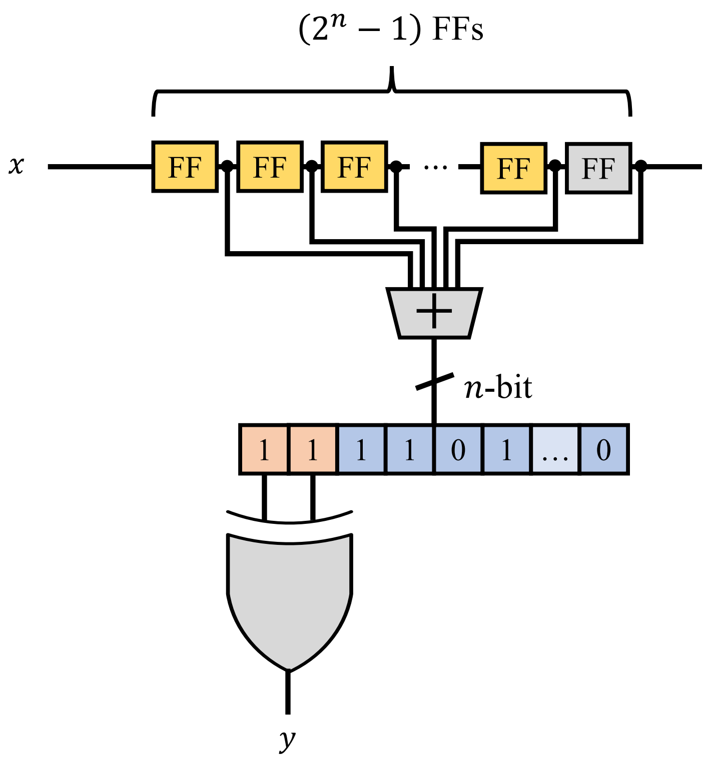


Figure 4.6: Hardware implementation of step function in SC with changed threshold.

as in Figure 4.7. In this graph, cases which are $n = 2, 3, \ldots, 10$, i.e., $N = 3, 7, 15, 31, 63, 127, 255, 511, 1023$ are shown. The second merit is discussed in the next two sections.

4.4 Implementation of Absolute Function Using Step Function

4.4.1 Proposal

SN has a bi-polar expression defined by $V_x = 2 \times P_x - 1$ as shown in Section 4.2. In bi-polar expression, arithmetic operations are performed by inputting the bit streams of SNs sequentially to logic circuits, as well as uni-polar expression. This section proposes a circuit calculating the absolute value using the SN expressed by bi-polar representation.

Absolute function using SNs can be implemented by combining the XNOR gate in Figure 4.2 and the step function circuit in Figure 4.4. As in Figure 4.8, let a, c be the input/output SN of the whole the circuit of the step function, respectively, and b be the output SN of the step function circuit. a and b are input into the XNOR circuit and c is

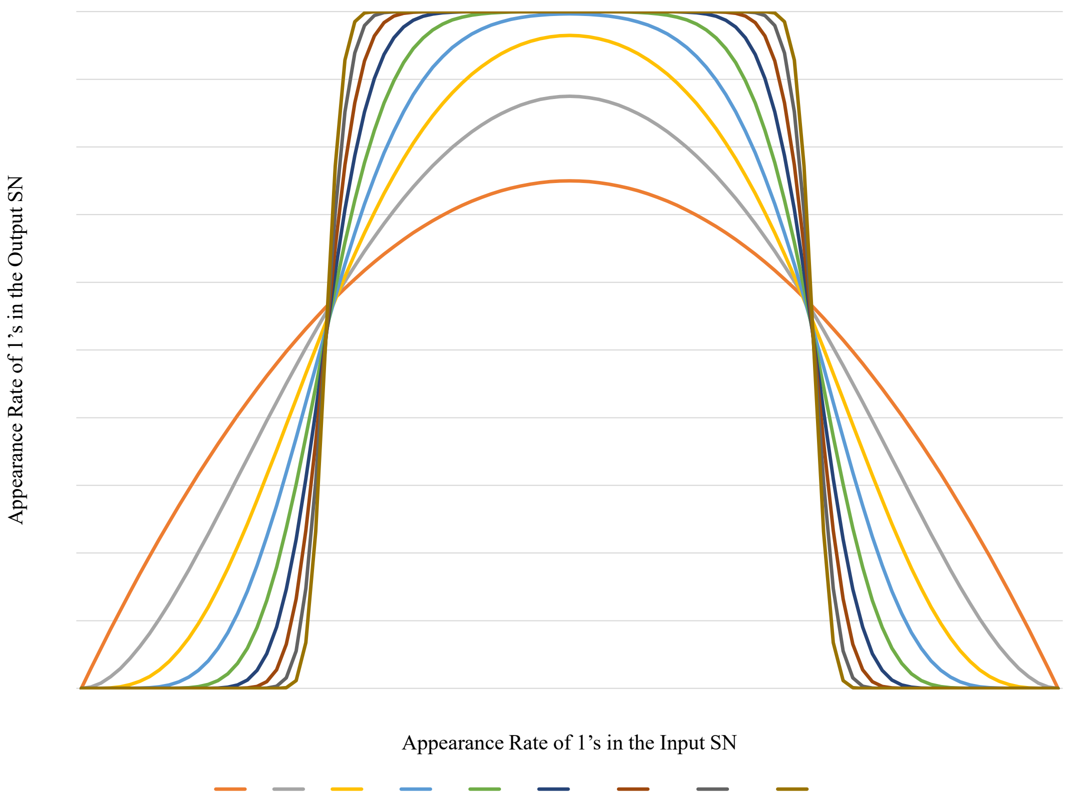


Figure 4.7: Theoretical values of input/output SN of a step function circuit with changed threshold.

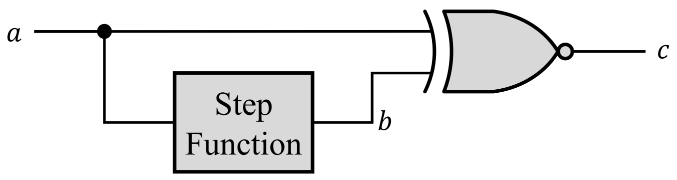


Figure 4.8: Hardware implementation of absolute function using bi-polar SN.

obtained. Their expected values of V_b and V_c becomes:

$$\begin{aligned} E(V_b) &= 2P_b - 1 \\ &= \begin{cases} 2 \times 1 - 1, & (P_a > \frac{1}{2}) \\ 2 \times 0 - 1, & (P_a < \frac{1}{2}) \\ 2 \times 1 - 1, & (P_a = \frac{1}{2}) \end{cases} \\ &= \begin{cases} 1, & (V_a > 0) \\ -1, & (V_a < 0) \\ 0, & (V_a = 0), \end{cases} \end{aligned} \tag{4.12}$$

Table 4.2: The circuit of the absolute function circuit using SN.

N [FFs]	Area [NANDs]
1	13.75
3	75
7	166
15	305

$$
\begin{aligned}
E(V_c) &= V_a \times V_b \\
&= \begin{cases} V_a \times 1, & (V_a > 0) \\ V_a \times -1, & (V_a < 0) \\ V_a \times 0, & (V_a = 0) \end{cases} \\
&= \begin{cases} V_a, & (V_a > 0) \\ -V_a, & (V_a < 0) \\ 0, & (V_a = 0) \end{cases} \\
&= |V_a|. \qquad (4.13)
\end{aligned}
$$

Therefore, by combining XNOR gate and step function, absolute value of SN in bi-polar expression can be calculated.

4.4.2 Circuit Areas

The hardware cost of the step function circuit using SN, obtained by logic synthesis, is shown in Table 4.2. Logic synthesis was performed using Verilog[35] by Design Compiler version D-2010.03-SP5 and the STARC 90nm library. Except for $N = 1$, which requires no adders, the circuit area became around $20N$ gates in NAND gates conversion.

4.5 Implementation of Discontinuous Function Using Step Function

4.5.1 Proposal

This section proposes a circuit calculating the discontinuous function of SNs. The circuit is shown in Figure 4.9. Stochastic circuits A, B, and C are placed in the former stage

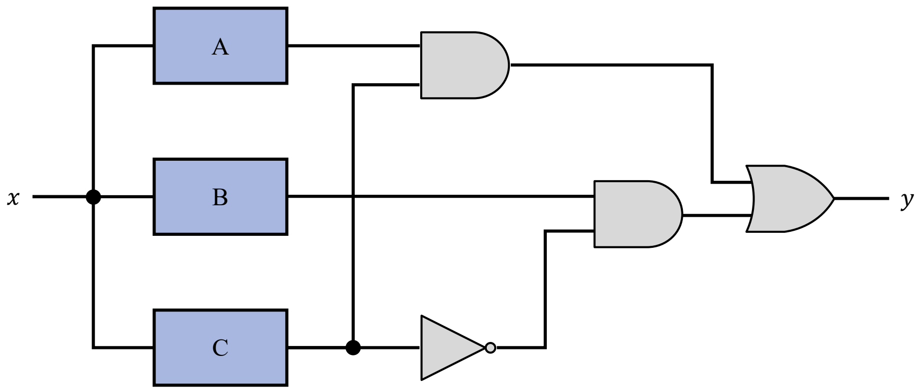


Figure 4.9: Hardware implementation of discontinuous function in SC.

of the proposed circuit, and circuit equivalent to the multiplexer circuit is placed in the latter stage. A and B are arbitrary stochastic circuits. C is a step function with arbitrary threshold. The proposed circuit has input SN x and output SN y.

4.5.2 Prerequisite Circuits

As well as the step function in Figure 4.6, fictions using SN duplicators proposed in Chapters 2 and 3, or in [39] and [40], are used.

The $\sin' x$ function is expressed by Equation (4.14) and its circuit is shown in Figure 2.10(a). The DUPs in Figure 2.10(a) are the 2^nRRR duplicators proposed in Chapter 3 or in [40].

$$\begin{aligned}\sin x &\approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \\ &= x(1 - \frac{x^2}{6}(1 - \frac{x^2}{20}(1 - \frac{x^2}{42}))) \\ &= \sin' x. \end{aligned} \tag{4.14}$$

The $\cos' x$ function is expressed by Equation (4.15) and its circuit is shown in Figure 2.10(b). The DUPs in Figure 2.10(b) also show the 2^nRRR duplicators proposed in Chapter 3 or in [40].

$$\begin{aligned}\cos x &\approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \\ &= 1 - \frac{x^2}{2}(1 - \frac{x^2}{12}(1 - \frac{x^2}{30}(1 - \frac{x^2}{56}))) \\ &= \cos' x. \end{aligned} \tag{4.15}$$

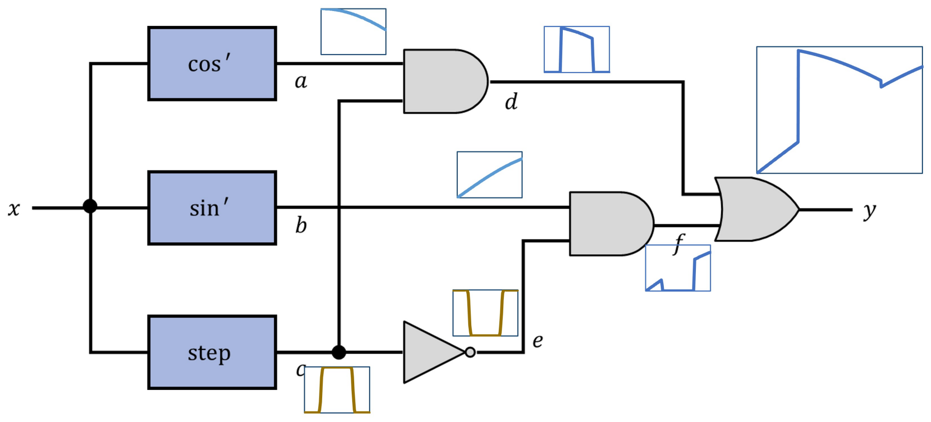


Figure 4.10: Hardware implementation of discontinuous function in SC (example).

4.5.3 Example of Discontinuous Function

Here, as an example, cos′ function are implemented as A, sin′ function as B, step function with threshold of 1/4 and 3/4 as C, as in Figure 4.10. In particular, approximation function of sin and cos, step function in Figure 4.6 are used.

A discontinuous function can be implemented by the circuit in the previous section. Consider inputting SN x to the step function circuit as in Figure 4.10. Here, a–f is the output SN of each circuit or gate. The expected value of these SNs is expressed by:

$$E(V_a) = \cos'(V_x), \tag{4.16}$$

$$E(V_b) = \sin'(V_x), \tag{4.17}$$

$$E(V_c) = \begin{cases} 1, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ 0, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \tag{4.18}$$

$$\begin{aligned} E(V_d) &= V_a \times V_c \\ &= \begin{cases} \cos'(V_x), & (\frac{1}{4} < V_x < \frac{3}{4}) \\ 0, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.19}$$

$$\begin{aligned} E(V_e) &= 1 - V_c \\ &= \begin{cases} 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ 1, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.20}$$

$$\begin{aligned} E(V_f) &= V_b \times V_e \\ &= \begin{cases} 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.21}$$

Table 4.3: The circuit of the discontinuous function circuit using SN.

N [FFs]	1RRR	2RRR	4RRR	8RRR
3	2082	2188	2956	3920
7	2173	2279	3047	4011
15	2312	2418	3186	4150

* The unit of this table is [NANDs].

$$\begin{aligned} E(V_y) &= V_d + V_f - V_d \times V_f \\ &= \begin{cases} \cos'(V_x) + 0 - \cos'(V_x) \times 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ 0 + \sin'(V_x) - 0 \times \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x) \end{cases} \\ &= \begin{cases} \cos'(V_x), & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x). \end{cases} \end{aligned} \tag{4.22}$$

As above, by combining step function circuit with conventional circuits, discontinuous functions can be calculated.

4.5.4 Circuit Areas

The hardware cost of the discontinuous function circuit using SN, obtained by logic synthesis, is shown in Table 4.3. Logic synthesis was performed using Design Compiler version D-2010.03-SP5[36] and the STARC 90nm library[37]. The circuit area became around $20N$ gates larger, in NAND gates conversion, per FFs in the step function. Also, the circuit area increased significantly when a large sized SN duplicator was used.

4.6 Experimental Evaluations

4.6.1 Setup

In this section, circuits of absolute function and discontinuous function are simulated and evaluated. The simulation is performed under the following conditions:for $N = 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023$. (Due to its design, the step function in this discontinuous function cannot have $N = 1$.)

- Language for simulation: Python 3.6.3[34]
- Functions: Step function, absolute function, discontinuous function

- Appearance rate P_x of input SN x: 0, 0.01, 0.02, 0.03, . . ., 1
- Length of input SN x: 10,000 bits
- Initial bits stored in FFs: 1010. . . 01
- Performance indicator: MSE

MSE (Mean Square Error) is obtained by:

$$MSE = \frac{1}{101}\sum_{i=0}^{100}\left(f_{theor}(\frac{i}{100}) - f_{actual}(\frac{i}{100})\right)^2, \tag{4.23}$$

where $f_{theor}(\frac{i}{100})$ and $f_{actual}(\frac{i}{100})$ are the theoretical value and the obtained value when $i/100$ is input, respectively.

4.6.2 Results

Figure 4.11 shows the output when SN x with 101 kinds of P_x is input to the circuit of the step function of the Figure 4.4 for each ten N's. From this figure, as N increases, the output approaches the step function. In addition, the MSE of the step function is shown in the Table 4.4. Also from this table, as N increases, the output approaches the step function.

Figure 4.12 shows the output when SN x with 101 kinds of P_x is input to the circuit of the absolute function of the Figure 4.8 for each ten N's. From this figure, as N increases, the output approaches the absolute function. In addition, the MSE of the absolute function is shown in the Table 4.5. Also from this table, as N increases, the output approaches the absolute function.

Table 4.6 shows the MSE values when SN x with 101 kinds of P_x is input to the circuit of the discontinuous function of the Figure 4.10 for each nine N's and four SN duplicators. From this table, as N increases, the output approaches the discontinuous function. Under the conditions of this experiment, the circuits implemented by 2RRR or 4RRR duplicator had the smallest errors. However, the error due to the SN duplicators was smaller than that of FFs in step function. Figure 4.13 shows the output of the case with the smallest MSE, where $N = 1023$ and 2RRR duplicator was used. The errors became large around the inputs 0.25 and 0.75.

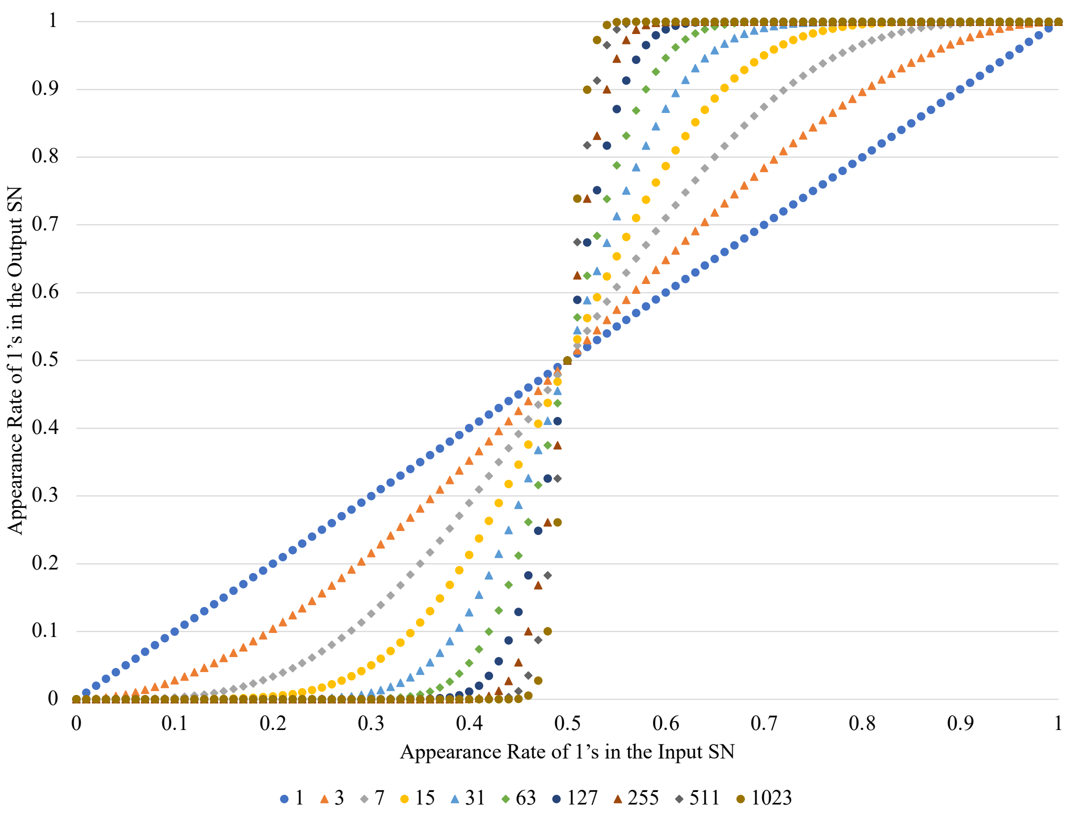


Figure 4.11: Actual values of input/output SNs of a step function circuit

4.6.3 Discussions

The errors became large around the inputs 0.25 and 0.75 since the step function used in the experiment is not an ideal step function. The solid line in Figure 4.13 assumes an ideal function (whose value clearly switches from 0 to 1 at the threshold). However, the circuit of the step function used in the experiment takes subtle values near the threshold as shown in Figure 4.7. Therefore, errors occur in the addition by the OR gate in Figure 4.1.

The output values of function described in Figure 4.7 when the input value is 0.25 or 0.75 cannot be proved to converge to certain values. From Figure 4.7, the theoretical values of output SN y depending on $N$when the values of input SN x are 0.25 or 0.75 are shown in Table 4.7, and seems to converge to 0.5 in this example. It is obvious that the output value when input value is 0.25 or 0.75 takes a value between 0 and 1 for any

Table 4.4: MSE value of step function using SC.

N [FFs]	MSE
1	8.00×10^{-2}
3	5.59×10^{-2}
7	3.87×10^{-2}
15	2.66×10^{-2}
31	1.81×10^{-2}
63	1.21×10^{-2}
127	7.90×10^{-3}
255	4.97×10^{-3}
511	2.93×10^{-3}
1023	1.56×10^{-3}

N, so let $0 < \alpha_N < 1$ be that value. In this case, Equation 4.18–4.22 become:

$$E(V_c) = \begin{cases} 1, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \alpha_N, & (V_x = \frac{1}{4}, \frac{3}{4}) \\ 0, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \tag{4.24}$$

$$\begin{aligned} E(V_d) &= V_a \times V_c \\ &= \begin{cases} \cos'(V_x), & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \alpha_N \cos'(V_x), & (V_x = \frac{1}{4}, \frac{3}{4}) \\ 0, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.25}$$

$$\begin{aligned} E(V_e) &= 1 - V_c \\ &= \begin{cases} 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ 1 - \alpha_N, & (V_x = \frac{1}{4}, \frac{3}{4}) \\ 1, & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.26}$$

$$\begin{aligned} E(V_f) &= V_b \times V_e \\ &= \begin{cases} 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ (1 - \alpha_N) \sin'(V_x), & (V_x = \frac{1}{4}, \frac{3}{4}) \\ \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x), \end{cases} \end{aligned} \tag{4.27}$$

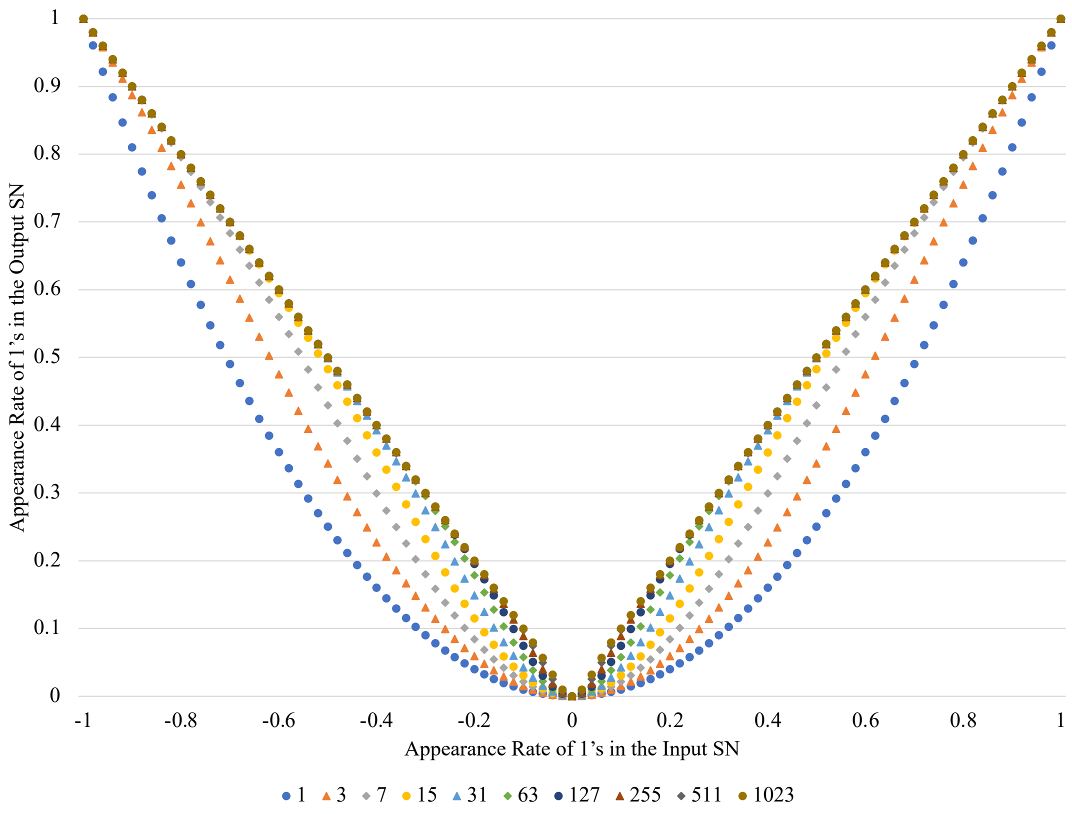


Figure 4.12: Actual values of input/output SNs of a absolute function circuit

$$
\begin{aligned}
E(V_y) &= V_d + V_f - V_d \times V_f \\
&= \begin{cases} \cos'(V_x) + 0 - \cos'(V_x) \times 0, & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \alpha_N \cos'(V_x) + (1-\alpha_N)\sin'(V_x) & \\ \quad -\alpha_N \cos'(V_x) \times (1-\alpha_N)\sin'(V_x), & (V_x = \frac{1}{4}, \frac{3}{4}) \\ 0 + \sin'(V_x) - 0 \times \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x) \end{cases} \\
&= \begin{cases} \cos'(V_x), & (\frac{1}{4} < V_x < \frac{3}{4}) \\ \alpha_N \cos'(V_x) + (1-\alpha_N)\sin'(V_x) & \\ \quad -\alpha_N(1-\alpha_N)\cos'(V_x)\sin'(V_x), & (V_x = \frac{1}{4}, \frac{3}{4}) \\ \sin'(V_x), & (V_x < \frac{1}{4}, \frac{3}{4} < V_x). \end{cases}
\end{aligned} \tag{4.28}
$$

Therefore, when $V_x = \frac{1}{4}, \frac{3}{4}$,

$$
E(V_y) = \alpha_N \cos'(V_x) + (1-\alpha_N)\sin'(V_x) - \alpha_N(1-\alpha_N)\cos'(V_x)\sin'(V_x) \tag{4.29}
$$

holds. What we try to implement is $E(V_y) = \alpha_N \cos'(V_x) + (1-\alpha_N)\sin'(V_x)$, whose difference is $-\alpha_N(1-\alpha_N)\cos'(V_x)\sin'(V_x)$. This value is a constant regardless of N

Table 4.5: MSE value of absolute function using SC.

N [FFs]	MSE
1	3.30×10^{-2}
3	1.34×10^{-2}
7	5.05×10^{-3}
15	1.85×10^{-3}
31	6.64×10^{-4}
63	2.37×10^{-4}
127	8.41×10^{-5}
255	2.98×10^{-5}
511	1.05×10^{-5}
1023	3.66×10^{-6}

Table 4.6: MSE value of discontinuous function using SC.

N [FFs]	1RRR	2RRR	4RRR	8RRR
3	4.75×10^{-2}	4.32×10^{-2}	4.11×10^{-2}	4.44×10^{-2}
7	2.85×10^{-2}	2.68×10^{-2}	2.58×10^{-2}	2.71×10^{-2}
15	1.77×10^{-2}	1.70×10^{-2}	1.65×10^{-2}	1.71×10^{-2}
31	1.20×10^{-2}	1.12×10^{-2}	1.10×10^{-2}	1.18×10^{-2}
63	7.96×10^{-3}	7.33×10^{-3}	7.26×10^{-3}	7.83×10^{-3}
127	5.15×10^{-3}	4.77×10^{-3}	4.73×10^{-3}	5.19×10^{-3}
255	3.26×10^{-3}	2.96×10^{-3}	2.99×10^{-3}	3.19×10^{-3}
511	1.99×10^{-3}	1.77×10^{-3}	1.81×10^{-3}	1.95×10^{-3}
1023	1.09×10^{-3}	9.90×10^{-4}	1.02×10^{-3}	1.11×10^{-3}

(except for the slight difference of α_N). Therefore, assuming $\alpha_N = 1/2$, the errors when $V_x = \frac{1}{4}, \frac{3}{4}$ become $1/4 \sin'(0.25) \cos'(0.25) \approx 0.0599$, $1/4 \sin'(0.25) \cos'(0.25) \approx 0.1247$, respectively. The maximum errors of $\sin' \cos'$ depending on SN duplicators become around 0.02 (see Table 3.4 for detail), and are relatively small compared to the errors discussed in this section.

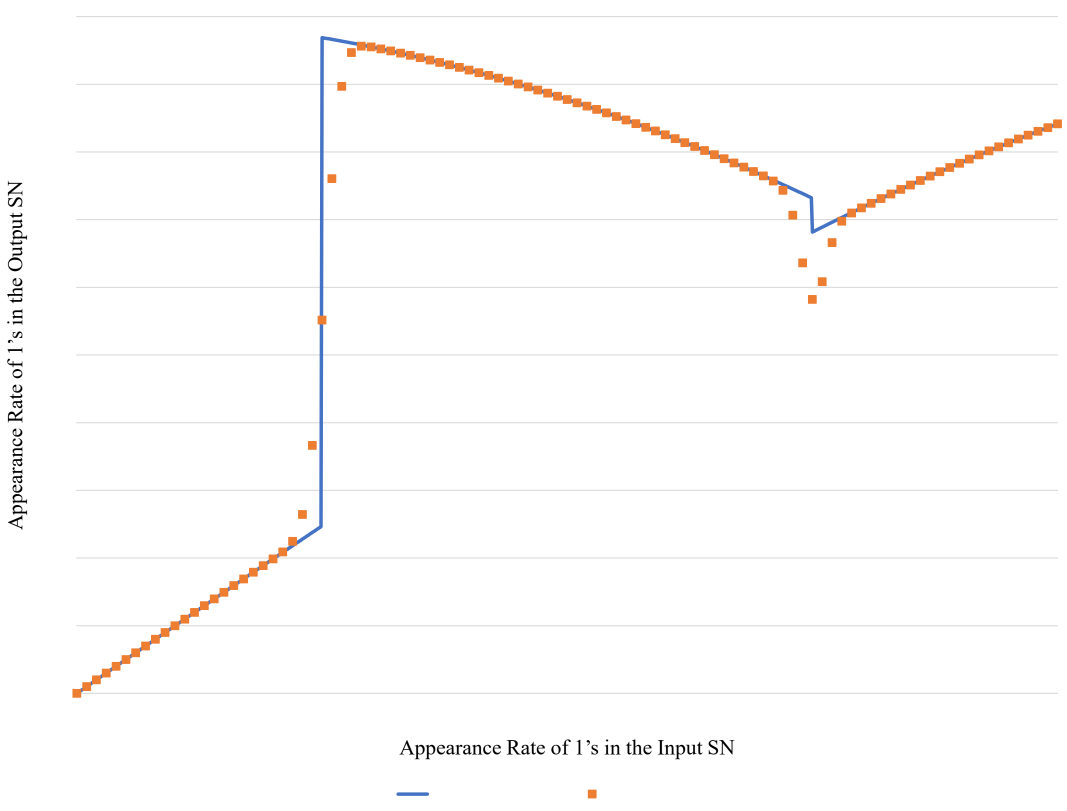


Figure 4.13: Actual values of input/output SNs of a discontinuous function circuit

Table 4.7: Theoretical values of output SN y depending on $N$when the values of input SN x are 0.25 or 0.75 in Figure 4.7.

N	3	7	15	31	63	127	255	511	1023
V_y	0.563	0.554	0.539	0.527	0.519	0.514	0.510	0.507	0.505

4.7 Conclusions

In this chapter, we proposed and evaluated step function in SC and their related functions.

Chapter 5

Conclusions

This dissertation proposed hardware optimization of SC. The goal of this dissertation is **to omit the use of SN generator as much as possible** which enables to reduce the hardware cost of the circuits, especially the latency. Overall, this dissertation demonstrated that the proposing circuits operate as expected in the experiments, without the new use of SN generators. In this point, this dissertation has successfully achieved the goal.

Chapter 2 [Stochastic Number Duplicators Based on Bit Re-arrangement Using Randomized Bit Streams] proposed *FSR* and *RRR* duplicators, which generates and outputs a new SN which has the equivalent value with the input SN but has a independent bit stream. In this chapter, a randomized bit stream was introduced to re-arrange the bits stored in flip-flops. The SNs duplicated by the proposing FSR and RRR duplicators had the equivalent values but have independent bit streams, by effectively utilizing bit re-arrangement using randomized bit streams. Experimental evaluation results demonstrated that the RRR duplicator, in particular, obtains more accurate results, reducing the mean square errors by 20%–89% compared with a conventional SN duplicator, without adding any SN generators. Also, this chapter discussed the behavior of the proposed SN duplicators when the bit length of the input SN became longer.

Chapter 3 [Scalable Stochastic Number Duplicators for Accuracy-flexible Arithmetic Circuit Design] proposed 2^n*RRR* duplicator, which uniquely extends the RRR duplicator and has a scalable structure by changing the numbers of flip-flops for bit re-arrangement. In this chapter, we extended the RRR duplicator proposed in Chapter 2, enabling to change the accuracy of the circuit itself. The proposed 2^nRRR duplicator output different SNs every time and are all independent of each other. The 2^nRRR duplicator could be theoretically proved to flexibly change the accuracies of the arithmetic circuits. Also from the experimental evaluation results, this chapter clarified that the 2^nRRR duplicator enables accuracy-flexible circuits. In a particular case, one instance of the proposed 2^nRRR duplicator reduced the mean square errors by more than 50% compared with the RRR duplicator proposed in Chapter 2. From proof and experimental evaluations, more accurate calculations are enabled without the use of SN generators by selecting the correct instance for each situation.

Chapter 4 [Hardware Implementation of Step Function in Stochastic Computing and Its Applications] proposed step function in SC and its applications. The proposed circuit of utilizes flip-flops and an adder to perform as step function uniquely calculating the stored bits in the flip-flops. The proposed step function could be theoretically proved to perform as a step function. This chapter confirmed that the proposed circuit behaved as a step function through experimental evaluations. Also as an application, steep functions or discontinuous functions could be realized by applying the discontinuity of the step function. As a steep function, this chapter also proposed hardware implementation of absolute function and discontinuous function, by synthesizing an arbitrary function as a discontinuous function. As an example, a composite function of trigonometric function of sin and cos function was implemented in this chapter. Through experimental evaluations, this chapter confirmed that the circuits of step function, absolute function, and discontinuous function perform as target function.

In conclusion, hardware optimization of SC has future prospect. However, there remain several tasks to be done in the near future. Our future works are summarized as follows:

- Optimize the proposed SN duplicators and utilize them so that they can generate new SNs;
- Combine the proposed step function with a different function to implement another unrealizable function;
- Apply the proposed circuits into practical applications such as machine learning or image processing;
- Merge the above practical applications as a system.

First, we aim to optimize the proposed SN duplicators depending on the situation and utilize them to generate new SNs, to reduce the circuit area of conventional LFSR-based SN generators. Second, we search for other functions to combine with the proposed step function and implement another function which used to be unrealizable. Third, we will apply the proposed circuits into practical applications such as machine learning or image processing, and validate them through experimental evaluations. Finally, we will combine the above practical applications as a system, comparing circuit area and operation results with the equivalent system implemented with binary computing.

Acknowledgments

First and foremost, I would like to express my deepest thanks of gratitude to Prof. Nozomu Togawa (戸川望教授) at the Department of Computer Science and Communications Engineering of Waseda University for his continuous supports on my research work for five and a half years.

Secondly, I appreciate the strong supports and valuable advices from Prof. Masao Yanagisawa (柳澤政生教授) at the Department of Electronic and Physical Systems of Waseda University, and Prof. Keiji Kimura (木村啓二教授) at the Department of Computer Science and Communications Engineering of Waseda University.

I am deeply thankful of Dr. Masashi Tawada (多和田雅師氏) and Dr. Kazushi Kawamura (川村一志氏) at the Department of Communications Engineering of Waseda University for their helpful technical advices.

I would like to offer my special thanks to Ms. Shuko Watanabe (渡部周子氏) for supporting my research life. I had a meaningful and enjoyable time at Togawa Laboratory with my colleagues Dr. Siya Bao (鮑思雅氏), Dr. Kento Hasegawa (長谷川健人氏), and Dr. Daisuke Oku (於久太祐氏). I also thank all the students in the Togawa Laboratory and the Information System Laboratory.

Outside the laboratory, my family and my friends always encouraged me. I take this opportunity to thank everyone involved.

References

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

[2] 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.

[3] “Xilinx.” https://www.xilinx.com/

[4] “Raspberry pi.” https://www.raspberrypi.org/

[5] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

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

[7] 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.

[8] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, “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, 2016.

[9] J. Li, Z. Yuan, Z. Li, C. Ding, A. Ren, Q. Qiu, J. Draper, and Y. Wang, “Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks,” in *Proc. International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1230–1236.

[10] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function," in *Proc. International Conference on Computer Design (ICCD)*, 2017, pp. 97–104.

[11] I. Yeo, S. Gi, B. Lee, and M. Chu, "Stochastic implementation of the activation function for artificial neural networks," in *Proc. Biomedical Circuits and Systems Conference (BioCAS)*, 2016, pp. 440–443.

[12] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Proc. International Conference on Computer Design (ICCD)*, 2011, pp. 154–161.

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

[14] S. Aygun, M. Altun, and E. O. Gunes, "Sobel filter operation in image processing via stochastic arithmetic-logic unit design," in *Proc. Signal Processing and Communications Applications Conference (SIU)*, 2017, pp. 1–4.

[15] R. Seva, P. Metku, K. K. Kim, Y. Kim, and M. Choi, "Approximate stochastic computing (ASC) for image processing applications," in *Proc. International SoC Design Conference (ISOCC)*, 2016, pp. 31–32.

[16] M. H. Najafi and M. E. Salehi, "A fast fault-tolerant architecture for sauvola local image thresholding algorithm using stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 808–812, 2016.

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

[18] W. Qian, C. Wang, P. Li, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.

[19] 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. International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 107–114.

[20] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computing*, vol. 63, no. 6, pp. 1474–1486, 2014.

[21] Z. Wang, N. Saraf, K. Bazargan, and A. Scheel, “Randomness meets feedback: Stochastic implementation of logistic map dynamical system,” in *Proc. Design Automation Conference (DAC)*, 2015, pp. 1–7.

[22] Y. Liu and K. K. Parhi, “Computing hyperbolic tangent and sigmoid functions using stochastic logic,” in *Proc. Asilomar Conference on Signals, Systems and Computers (ACSSC)*, 2016, pp. 1580–1585.

[23] P. S. Ting and J. P. Hayes, “Isolation-based decorrelation of stochastic circuits,” in *Proc. International Conference on Computer Design (ICCD)*, 2016, pp. 88–95.

[24] C. Ma, S. Zhong, and H. Dang, “High fault tolerant image processing system based on stochastic computing,” in *Proc. International Conference on Computer Science and Service System (CSSS)*, 2012, pp. 1587–1590.

[25] W. Qian and M. D. Riedel, “The synthesis of robust polynomial arithmetic with stochastic logic,” in *2008 45th ACM/IEEE Design Automation Conference*, 2008, pp. 648–653.

[26] G. G. Lorenz, *Bernstein Polynomials*. AMS Chelsea Publishing, 1986.

[27] Y. Sakamoto and S. Yamashita, “Efficient methods to generate constant sns with considering trade-off between error and overhead and its evaluation,” *IEICE Transactions on Information and Systems*, vol. E103.D, pp. 321–328, 02 2020.

[28] F. Neugebauer, I. Polian, and J. P. Hayes, “Building a better random number generator for stochastic computing,” in *Proc. Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 1–8.

[29] R. Muguruma and S. Yamashita, “Stochastic number generation with the minimum inputs,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100.A, pp. 1661–1671, 08 2017.

[30] S. A. Salehi, “Low-correlation low-cost stochastic number generators for stochastic computing,” in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2019, pp. 1–5.

[31] D. H. K. Hoe and C. Pajardo, “Implementing stochastic bayesian inference : Design of the stochastic number generators,” in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2019, pp. 1105–1109.

[32] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *Proc. Design, Automation, and Test in Europe (DATE)*, 2016, pp. 1550–1555.

[33] 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.

[34] "Python." https://www.python.org/

[35] "Ieee standard for systemverilog–unified hardware design, specification, and verification language," *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, pp. 1–1315, 2013.

[36] "Design compiler graphical." https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html

[37] "Ieee standard vhdl language reference manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp. 1–640, 2009.

[38] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *Proc. International Conference on Computer Design (ICCD)*, 2014, pp. 361–366.

[39] R. Ishikawa, M. Tawada, M. Yanagisawa, and N. Togawa, "Stochastic number duplicators based on bit re-arrangement using randomized bit streams," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101-A, no. 7, pp. 1002–1013, 2018.

[40] R. Ishikawa, M. Tawada, M. Yanagisawa, and N. Togawa, "Scalable stochastic number duplicators for accuracy-flexible arithmetic circuit design," *IPSJ Transactions on System LSI Design Methodology (TSLDM)*, vol. 13, pp. 10–20, 2020.

List of Publications

Peer Review Journal Articles

⟨1⟩ ○ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "Scalable Stochastic Number Duplicators for Accuracy-flexible Arithmetic Circuit Design," *IPSJ Transactions on System LSI Design Methodology (T-SLDM)*, vol. 13, pp. 10–20, February 2020.

⟨2⟩ ○ 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*, vol. E101-A, no. 7, pp. 1002–1013, July 2018.

Peer Review Conference Papers

⟨3⟩ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "Multi-Resolutional Image Format Using Stochastic Numbers and Its Hardware Implementation," in *Proc. 11th IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, pp. 1–4, February 2020.

⟨4⟩ Kento Hasegawa, Ryota Ishikawa, Makoto Nishizawa, Kazushi Kawamura, Masashi Tawada, Nozomu Togawa, "FPGA-based Heterogeneous Solver for Three-Dimensional Routing," in *Proc. 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 11–12, January 2020.

⟨5⟩ Kento Hasegawa, Kazunari Takasaki, Makoto Nishizawa, Ryota Ishikawa, Kazushi Kawamura, Nozomu Togawa, "Implementation of a ROS-Based Autonomous Vehicle on an FPGA Board," in *Proc. International Conference on Field-Programmable Technology (FPT)*, pp. 457–460, December 2019.

⟨6⟩ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "Error Correction System Using Stochastic Numbers in Symmetric Channels and Z Channels," in *Proc. 26th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, pp. 578–581, November 2019.

⟨7⟩ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "Error Correction Coding of Stochastic Numbers Using BER Measurement," in *Proc. 25th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 243–246, July 2019.

⟨8⟩ ○ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "2^nRRR: Improved Stochastic Number Duplicator Based on Bit Re-arrangement," in *Proc. New Generation of Circuits and Systems Conference (NGCAS)*, pp. 182–185, November 2018.

⟨9⟩ ○ Ryota Ishikawa, Masashi Tawada, Masao Yanagisawa, Nozomu Togawa, "An Effective Stochastic Number Duplicator and its Evaluations using Composite Arithmetic Circuits," in *Proc. 24th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 53–56, July 2018.

Domestic Conference Papers

⟨10⟩ 石川 遼太, 多和田 雅師, 戸川 望, "ストカスティック数を用いた絶対値関数及び不連続関数の実装と評価 ," 情報処理学会 DA シンポジウム論文集, pp. 65–70, 2021 年 9 月.

⟨11⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, "ストカスティック数を用いた非対称通信路の誤り訂正," 電子情報通信学会総合大会講演論文集, p. 26, 2020 年 3 月.

⟨12⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, "ストカステイック数を用いたステップ関数の実装と評価," 電子情報通信学会技術研究報告, vol. 119, no. 282, pp. 69–74, 2019 年 11 月.

⟨13⟩ 西澤 誠人, 石川 遼太, 長谷川 健人, 川村 一志, 多和田 雅師, 戸川 望, "配置配線のためのアンサンブルソルバシステム," DA シンポジウム 2019, 2019 年 8 月.

⟨14⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, "ストカスティック数を用いた再帰的分割による解像度解釈可変な画像形式," 電子情報通信学会技術研究報告, vol. 119, no. 154, pp. 71–76, 2019 年 7 月.

⟨15⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “ストカスティック数を用いた Z 通信路の誤り訂正,” 電子情報通信学会技術研究報告, vol. 119, no. 77, pp. 109–114, 2019 年 6 月.

⟨16⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “BER 測定を用いたストカスティック数の誤り訂正,” 研究報告システムと LSI の設計技術 (SLDM), vol. 2019-SLDM-187, no. 50, pp. 1–6, 2019 年 3 月.

⟨17⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “2^nRRR: 高度な並び替えにより誤り耐性を強化したストカスティック数複製器,” 電子情報通信学会技術研究報告, vol. 335, no. 118, pp. 95–100, 2018 年 12 月.

⟨18⟩ 石川 遼太, 長谷川 健人, 西澤 誠人, 川村 一志, 多和田 雅師, 戸川 望, “ナンバーリンクソルバのための FPGA 協調システム,” DA シンポジウム 2018, 2018 年 8 月.

⟨19⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “再収斂による計算誤りに耐性を持つストカスティック数複製器を用いた活性化関数の実装と評価,” 電子情報通信学会技術研究報告, vol. 118, no. 83, pp. 167–172, 2018 年 6 月.

⟨20⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “効率的なストカスティック数複製器と合成関数を用いたその評価,” 研究報告システムと LSI の設計技術 (SLDM), vol. 2018-SLDM-183, no. 36, pp. 1–6, 2018 年 3 月.

⟨21⟩ 石川 遼太, 多和田 雅師, 柳澤 政生, 戸川 望, “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,” 情報処理学会 DA シンポジウム論文集, pp. 169–174, 2017 年 9 月.

⟨22⟩ 長谷川 健人, 石川 遼太, 寺田 晃太朗, 川村 一志, 多和田 雅師, 戸川 望, “組込みデバイスと FPGA を用いたナンバーリンクソルバの設計と実装,” DA シンポジウム 2017, 2017 年 8 月.

Awards

⟨23⟩ TSLDM Best Paper Award, *IPSJ Transactions on System LSI Design Methodology*, Volume 13, September 2020. (Awarded for 1)

⟨24⟩ 情報処理学会 DA シンポジウム アルゴリズムデザインコンテスト 特別賞, DA シンポジウム 2019, 2019 年 8 月. (業績 13 に対し受賞)

⟨25⟩ 情報処理学会 SLDM 研究会 デザインガイア 2018 優秀発表学生賞, DA シンポジウム 2019, 2019 年 8 月. (業績 17 に対し受賞)

⟨26⟩ 情報処理学会 DA シンポジウム アルゴリズムデザインコンテスト 特別賞, DA シンポジウム 2018, 2018 年 8 月. (業績 18 に対し受賞)

⟨27⟩ 情報処理学会 DA シンポジウム アルゴリズムデザインコンテスト 最優秀賞, DA シンポジウム 2017, 2017 年 8 月. (業績 22 に対し受賞)