International Journal of Multidisciplinary Studies and Innovative Technologies

Volume : 5
Number: 1
Year: 2021
Pages: 94 - 97

On the Limit of Multiplexers in Stochastic Computing

Sercan Aygun[1,2*] and Ece Olcay Gunes[1]

[1]Department of Electronics and Communication Engineering, Istanbul Technical University, Istanbul, Turkey
[2]Department of Computer Engineering, Yildiz Technical University, Istanbul, Turkey
[*]Corresponding author: ayguns@itu.edu.tr

Abstract – Stochastic computing (SC) is an approach used in today's re-emerging hardware environments. Known deterministic circuit elements are fed by binary sequences with probability, and the output sequence probability expresses a mathematical operation in terms of the probability of input sequences. Pulse trains expressed with probability values feed deterministic logic systems by expressing unipolar or bipolar encoding techniques, and an output pulse train with a probability value is obtained. This approach, which provides benefits in terms of complexity, low power, and durability especially for arithmetic operations, appears in applications with flexible fault tolerance such as computer vision. In this context, the multiplexer (MUX) logic system is used as a scaled adder; in other words, the sum of binary probabilistic sequences coming to the inputs of a MUX is seen at the output at the rate of a coefficient. In this study, the limits of the MUX structure within the scope of SC are underlined. With the MUX structures created with different hardware configurations, the architectures are investigated for performance.

Keywords – bitstream processing, multiplexers, optimization, scaled adder, stochastic computing

I. Introduction

Stochastic computing (SC) systems are frequently encountered in today's hardware environments due to their low power and non-complex structures. In fact, these systems, dating back to the 1960s, showed slow progress until the early 2000s, but they became preferred, especially due to the advantages they provide with the advancement of image processing and neural network architectures. SC systems basically meet the following advantageous criteria: (i) reduced element complexity, (ii) low power, (iii) high parallelism, and (iv) fault robustness.

Standard adder architectures include element-crowded digital systems such as full adder & half adder. Although such sensitive circuit topologies are used for deterministic and precise addition operations, approximate approaches offer low-budget solutions for applications where the sensitivity can be within a certain tolerance specific to the application. Especially due to the visual flexibility provided by image processing and neural network applications, these are the areas where the most applications are developed within the scope of SC. Since negligible visual errors that the human eye cannot perceive can be tolerated, this disadvantage of hardware approximation can be ignored.

Basic arithmetic operations required for applications such as computer vision and neural networks; addition, subtraction, comparison, sorting, etc., are performed within the scope of SC when bitstreams feed deterministic circuits. For example, multiplication is simply done with a 2-input AND gate, and the approximate *product* of two numbers is obtained. *Addition*, which is the most used arithmetic operation after multiplication in both application areas, can be performed using the counter, OR gate, or multiplexer (MUX). While the OR gate is used to add small-value numbers, counters cause problems when adding signed numbers. Working with both signed and unsigned numbers, MUX is advantageous in terms of hardware complexity but disadvantageous in terms of accuracy. In this study, the trade-off of the MUX hardware structure when it is used as an accumulator within the scope of SC has been revealed, and the critical remarks that are crucial to pay attention are determined. Thus, one can drive the steps to be decided before using MUX by including the pre-design hardware plan.

In the following sections, a review of the literature will be given, and then the basic SC concepts will be presented. Then, by mentioning the MUX usage limitations, possible points to be considered will be summarized for the hardware design issues.

II. Previous Studies

When we look at previous studies within the scope of SC, it is seen that architectures are generally suitable for hardware design of image processing and neural network systems [1]–[7]. In addition, it is also seen that communication systems are performed within the scope of SC that is even be used in the recent robotic technologies and power systems [8]–[10].

Although SC efforts date back to the 1960s [11], there is a continuous dark period until the 2000s [12]. SC, which gained importance especially with the new developments in neural networks, quickly emerged as a popular computing area, especially in 2015 and after. SC-based architectures, which provide noise and error robustness with a lightweight neural network structure, result in a very acceptable range in terms of accuracy.

Until 2015, it is seen that there are various examples in the literature, especially in the field of image processing [13], [14]. Besides, there are various neural network applications in 2015 and beyond [1]–[7]. We encounter noise-sensitive SC-based applications in image processing, in which the MUX element is often used as an *adder*. In particular, applications that include kernel-based arithmetic, such as edge detection, use the MUX hardware structure [14].

In the literature, there have been major efforts in the implementation of efficient arithmetic hardware architecture unique to the application [1]–[3]. Basic arithmetic operations

in SC are done using a single gate or simple logic system. Multiplication is based on the encoding type. In bipolar encoding (BPE), XNOR is the multiplier, whereas in unipolar encoding (UPE), AND gate is the multiplier [15]. Li et al. propose a new stochastic multiplier for quantized NNs [4]. On the other hand, there are many solutions in the accumulation [3], [5], [6], including single OR gate [16], modulo counter [7], approximate parallel counter (APC) [17], and MUX [18]. The accumulator hardware differs depending on the overall hardware structure, application requirements, and the bitstream encoding scheme. For instance, MUX or APC suits the design in SC-based full-precision neural network applications [19], [20], whereas a simple pop-up counter suffices in binarized networks [21]. Nonetheless, the use of APC results in an immense number of data conversions (*Stochastic-to-Binary* & *Binary-to-Stochastic*), since the APC circuit accepts binary stream and outputs a deterministic resultant. The use of MUXs helps us to keep SC space in the application, and parallel processing is widely supported.

III. Background

SC basics will be covered in this section. First, looking at the general design framework for SC, a design pyramid is presented as shown below in Fig. 1. Although the *algorithm design* at the bottom of these 4-basic steps is common to all systems, the critical steps are *hardware-software joint simulation* and *fault & noise analysis*. Since SC circuits offer robust solutions, the analysis of the 3rd step is fundamental. At this point, it should be noted that the *accuracy* versus *hardware efficiency* balance must be considered.

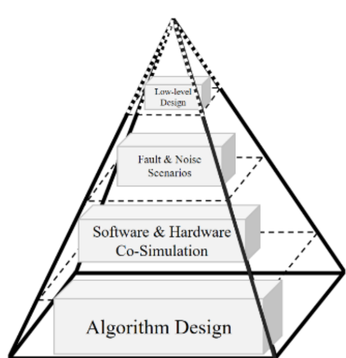

Fig. 1 Hardware design pyramid of SC systems [22]

The essential step in this computing methodology is representing the operands, i.e., scalar values. Instead of the conventional deterministic expression known in digital systems, numbers are expressed in terms of probabilities.

A. Number Representation

This subsection summarizes the two most common number representation encoding techniques. While the traditional approach expresses the numbers as 2^n-bit priority format (where n is the bit position); it gives a different significance to each bit of the binary code, which is defined using the most significant bit and the least significant bit. Nevertheless, in the random pulse processing systems, the numbers are defined by imitating biological signals such as high and low values occur on a random bit flow. Thus, the non-complex digital design and the noise- and fault-robust architectures are achieved for next-generation computing environments. In the frame of the pulse processing, a number can be represented in the UPE or BPE format as making the stream stochastic, which has built-in fault robustness on behalf of SC. If negative numbers are to be used, then the unipolar encoding cannot be employed. For any stream with N elements, $N \in \mathbb{Z}^+$, $\boldsymbol{X}$ represents a stream that starts from *1st* bit and ends at the Nth bit s.t. $\boldsymbol{X} = \{X_1, \ldots, X_N\}$. $N \geq 2$ as bitstream size is preferably set to construct $\boldsymbol{X}$. Any m^{th} element in $\boldsymbol{X}$ as X_m is either binary *1* or *0*. In a stream, encoding any number X lies in $0 \leq \frac{X}{N} \leq 1$ is obtained via UPE, while any number lies in $-1 \leq \frac{X}{N} \leq 1$ is obtained via BPE. The majority of *1*s, i.e., *high values*, in the stream is the indication of the positive sign, whereas the majority of *0*s is the indication of the negative sign in BPE. The logic-*1*s in the stream are randomly permuted. The occurrence probability of *1*s in UPE is $P = \frac{X}{N}$. In BPE, $P = \frac{\lfloor (X+N)/2 \rceil}{N}$ is the related probability, where $\lfloor \quad \rceil$ is for rounding to the nearest integer. Based on these probabilities, the bitstreams are randomly generated thanks to the total count of logic-*1*s. For instance, if $X = 2$ and $N = 8$, then UPE-based stream ($P = \frac{2}{8}$) can be $\boldsymbol{X} = 11000000$ or $\boldsymbol{X} = 10000010$ or $\boldsymbol{X} = 00010010$, etc., based on the random procedure applied through the stream generation. The total count of *1*s is 2, and their occurrence is random, which is generally obtained by the Bernoulli distribution. The same example for BPE is obtained via $P = \frac{\lfloor (2+8)/2 \rceil}{8} = \frac{5}{8}$, and any of the random cases are gathered, such as $\boldsymbol{X} = 11111000$ or $\boldsymbol{X} = 10110011$ or $\boldsymbol{X} = 01010111$, etc.

This way of representation is robust in terms of faults and soft errors. The related example of the bit-flip vulnerability is given in Table 1 by highlighting the outperformance of SC.

Table 1. Vulnerabilities in traditional binary representation

Value	**Binary Representation** (8-bit)	**SC Unipolar Representation (UPE)** (N=8-bit)
$X1 = (3)_{10}$	$(X1)_2 = 00000011$	$\boldsymbol{X1} = 10001100$
$X2 = (7)_{10}$	$(X2)_2 = 00000111$	$\boldsymbol{X2} = 11111101$
Soft Errors on both Approaches	Applying soft error to $(X2)_2$	Applying soft-error to $\boldsymbol{X2}$
Bit-flip Position	00000111	11111101
Bit-flip Soft Error Type	1→ 0 Bit-flip	1→ 0 Bit-flip
Bit-flip Effect in Base-10	$00000011 = (3)_{10}$	$11111001 = (6)_{10}$
The Absolute Error	$(7)_{10} \rightarrow (3)_{10}$ $\lvert 7-3 \rvert = 4$	$(7)_{10} \rightarrow (6)_{10}$ $\lvert 7-6 \rvert = 1$
SC Superiority	$4 > 1$ and SC is more robust to the soft errors	

B. Basic Operations

The basic operators used within the scope of SC can be discussed in terms of logic systems. The *multiply-and-accumulate* operation, which is widely used in computer vision & neural network system design, is the most obvious example of SC-based systems. In many studies, multiplication is exemplified as the basic arithmetic operation [1]–[3], [5]. Accordingly, AND gate for UPE-based and the XNOR gate for BPE-based streams are used for the multiplication of binary stream probabilities. More precisely, the bitstreams are processed as bit-by-bit for the multiplication resulting in a bitstream with probability, P_Y, as the product of inputs, $P_{X1} \times P_{X2}$. The other crucial operation is the accumulation, a.k.a. addition, whose details, together with the limitations, remarks, and recommendations, are presented in the following section.

IV. Scaled Adder: Limitations, Remarks, and Recommendations

In this section, we indicate the limits that scaled adders can cause within the scope of SC. We will share our observations obtained during the designs. First, we will define the scaled adder and then mention possible configurations. In the meantime, some points will be shared based on our experimental observations, which the designers should take care of.

A. What is a scaled adder?

Any deterministic $2^m - to - 1$ MUX is the adder of two stochastic operands for $m = 1$. Having encoded $X1$ and $X2$ operands in bitstreams, $\boldsymbol{X1}$ and $\boldsymbol{X2}$ streams with N elements are obtained. By setting MUX selection stream, S, to $1/2$, the output stream $\boldsymbol{Y}$ is obtained. When $\boldsymbol{Y}$ is decoded, $Y = \frac{X1+X2}{2}$ is yielded as a scaled addition. In Fig. 2, an example is depicted.

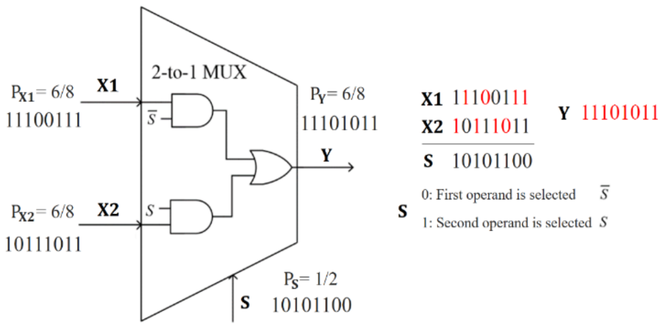

Fig. 2 2-input MUX example for a scaled addition

In the software simulation of the MUX, the logic expression can be utilized directly, while each input ($X1$, $X2$) and selection, S, is sourced with bitstreams. The bitstream format could be UPE or BPE, which does not change the hardware structure. This is different from the multiplication operation as AND & XNOR, accordingly used for related encoding format. In Table 2, the MATLAB-related SC adders show how simply creating a single line expression is possible for simulations.

Table 2. Simulation-related single-line implementations of SC adders

Operation	MATLAB Syntax
§***Scaled Adder*** `(in1 + in2)/2`	`(~S)&X1 | (S)&X2`
****Unipolar Addition*** `(Counter)`	`sum(bitstream)`
****Small Number Adder***	`or(X1, X2)`

`X1`: first input bitstream and `X2`: second input bitstream,
`S`: selection bitstream with $\frac{1}{2}$ probability for 2-to-1 MUX
*: using readily available functions, §: using bitwise operators

B. Limitations

When simulations of MUX-based SC adders are performed, several limitations are observed. The most important disadvantage that comes from using MUX is the scaling factor. The total result comes in a smaller proportion depending on the selection port (S). For example, if *eight* numbers are to be accumulated, the total value obtained at the output with 3-port selections using $8 - to - 1$ MUX will be obtained with a scale of $1/2^3$.

The other limit is related to the decision of the structure to be built when a large number of terms are to be accumulated. Fig. 3 illustrates that a large number of terms are generated as random arrays from stochastic number generators (SNG) and then sent to a MUX tree to be accumulated after multiplied by XNORs, exemplifying operation from the neural networks. At this point, attention has also been drawn to how things can be easily paralleled. Because there are no costly multipliers and adders, only individual XNORs multiply concurrently, and additions are determined thanks to the MUX tree performing simultaneously. However, here the decision of the depth (β) and element size (α) of the stages connected in cascade requires an optimization. If β=1 is selected (which means single-stage), a single MUX structure with inputs as many as incoming elements to be accumulated is employed. On the other hand, if the *bottom-to-top* hardware design approach is used, the MUX tree structure is obtained by cascading small-input MUXs.

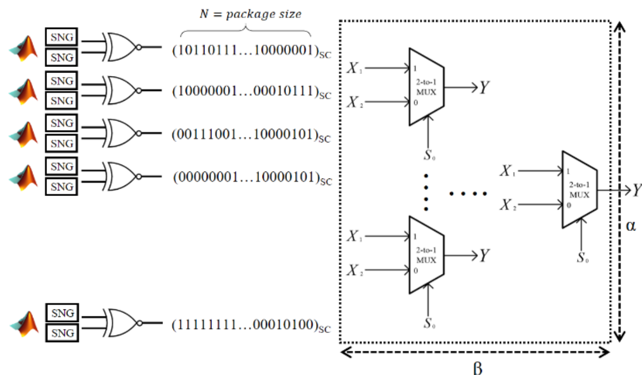

Fig. 3 Decision of the cascaded MUX architecture

To investigate MUX tree performance, we tested the situations that could be set up during the sum of 16 numbers and showed the error performance results in Table 3. The 16 numbers to be added up are processed either with $16 - to - 1$ MUX (blue background), or $8 - to - 1$ MUXs (yellow background), or $4 - to - 1$ MUXs (green background), or $2 - to - 1$ MUXs (red background). The MUX tree created by each may be different in terms of α and β. Beyond the physically obtained architecture, the error performance of the addition operation is given using mean absolute error (MAE) as an average of 1000 random accumulation trials, while the N bit length precision is changed from 8 to 1024.

Table 3. MAE performance of different cascading architectures

N							
8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
0.1347	0.0942	0.0672	0.0463	0.0314	0.0232	0.0157	0.0112
0.1305	0.0925	0.0660	0.0452	0.0328	0.0234	0.0156	0.0116
0.1275	0.0953	0.0673	0.0472	0.0332	0.0245	0.0164	0.0117
0.1349	0.0954	0.0684	0.0454	0.0323	0.0237	0.0169	0.0119
2-to-1		4-to-1		8-to-1		16-to-1	

As can be seen from Table 3, the bitstream size affects the efficiency. The cascaded architectures from $2 - to - 1$ (bottom-to-top) to $16 - to - 1$ (top-to-bottom) are investigated. Designers need to perform an analysis focusing on element complexity and latency, considering that there is not a large error difference between each architecture.

C. Recommendations & Remarks

According to the remarks we have obtained, when we support our tests with hardware simulations, it can be made some suggestions in terms of the limits noted. Our first recommendation is for the scaling factor; *the value obtained in*

base-10 format after decoding can be up-scaled. This can be performed in the context of application; since there can be an easier solution. For example, in a neural network, since the total accumulation coming to a neuron from the preceding layer is directly used in the activation function as a pre-activation value, the range of the activation function can be normalized to the corresponding range. Thus, no up-scaling, i.e., an extra multiplication, is required.

Secondly, the indicated design combination of a MUX tree is investigated, and the issues related to the hardware resource utilization & latency are highlighted. Since error for small group addition does not differ significantly, the bottom-to-top and top-to-bottom design considerations are crucial for their hardware complexity and latency. To the best of our experiences, bottom-to-top design requires many selection port sources and causes a delay. However, it supplies preferable randomness, while the top-to-bottom design is more efficient in terms of element complexity and delay. At this point, designers need to balance the two by paying attention to the contradiction.

Table 4. Observation on the MUX usage of different network architectures

	Multiplication	Accumulation
SC-FPNN	XNOR	MUX-tree, APC
SC-QNN		MUX-tree, APC
SC-BNN		Pop-count, modulo-T counter, parallel modulo-T counter

Finally, we deepen our recommendations and remarks on the use of the MUX SC primitive in neural networks and present observation of different network architectures in Table 4. Considering three neural network architectures, we present which ones will be used for which network in the presence of other accumulators as well as MUX usage. For this, the versions of the *full-precision neural network without quantization* (FPNN), a *quantized neural network with 2-or-more-bit quantization* (QNN), and *binarized neural network* (BNN) structures are decorated with the bitstreams; namely, SC-FPNN, SC-QNN, and SC-BNN. While MUX is an alternative adder for SC-FPNN and SC-QNN, it is emphasized that MUX is not included in SC-BNN. A simple counting operation handles the accumulation operation in SC-BNNs.

V. CONCLUSION

In this study, the limits and suggestions of a frequently used hardware element for SC, namely MUX, are mentioned. Limits and solutions are suggested for both the scale factor and the bulk accumulation design, i.e., the MUX tree. Combining our observations with hardware simulation from a deeper perspective, we highlighted that designers should consider the required randomness, element complexity, and latency. On the other hand, we conclude that MUX usage should be decided based on an application like neural network structures. Several topologies like BNN do not require more than a single counter in SC-based design; therefore, MUX has an alternative.

ACKNOWLEDGMENT

This work is supported by the Istanbul Technical University, BAP, with the project ID MDK-2018-41532.

REFERENCES

[1] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," *Proc. 54th Annu. Des. Autom. Conf. 2017 - DAC '17*, pp. 1–6, 2017, doi: 10.1145/3061639.3062290.

[2] H. Sim and J. Lee, "Cost-effective stochastic MAC circuits for deep neural networks," *Neural Networks*, vol. 117, pp. 152–162, 2019, doi: 10.1016/j.neunet.2019.04.017.

[3] B. Li, M. H. Najafi, B. Yuan, and D. J. Lilja, "Quantized neural networks with new stochastic multipliers," *Proc. - Int. Symp. Qual. Electron. Des. ISQED*, pp. 376–382, 2018, doi: 10.1109/ISQED.2018.8357316.

[4] D. J. Lilja, "Low-cost stochastic hybrid multiplier for quantized," *J. Emerg. Technol. Comput. Syst.* vol. 15(2), 2019. doi: 10.1145/3309882.

[5] A. Ardakani, *et al.*, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688-2699, Oct. 2017, doi: 10.1109/TVLSI.2017.2654298.

[6] V. T. Lee, *et al.*, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," *2017 Des. Autom. Test Eur. DATE*, pp. 13–18, 2017, doi: 10.23919/DATE.2017.7926951.

[7] S. Aygun, E. O. Gunes, and C. De Vleeschouwer, "Efficient and robust bitstream processing in binarised neural networks," *Electron. Lett.*, vol. 57, no. 5, pp. 219–222, 2021, doi: 10.1049/ell2.12045.

[8] I. Perez-Andrade, et al., "Stochastic computing improves the timing-error tolerance and latency of turbo decoders : Design guidelines and tradeoffs," *IEEE Access*, vol. 4, pp. 1008-1038, 2016, doi: 10.1109/ACCESS.2016.2523063.

[9] R. P. Duarte, H. Neto, and M. Véstias, "XtokaxtikoX: A stochastic computing-based autonomous cyber-physical system," *IEEE Int. Conf. on Rebooting Comp., ICRC 2016,* doi: 10.1109/ICRC.2016.7738716.

[10] D. Zhang and H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *IEEE Trans. Ind. Electron.*, vol. 55, no. 2, pp. 551–561, 2008, doi: 10.1109/TIE.2007.911946.

[11] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science: Volume 2*, J. T. Tou, Ed. Boston, MA: Springer US, 1969, pp. 37–172.

[12] W. J. Gross and V. C. Gaudet, *Stochastic Computing: Techniques and Applications*. Springer, Cham, 2019, doi: 10.1007/978-3-030-03730-7.

[13] A. Alaghi, Cheng Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," *50th Des. Autom. Conf. - DAC '13*, pp. 1–6, 2013, doi: 10.1145/2463209.2488901.

[14] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," *IEEE Int. Conf. Comput. Des.*, pp. 154–161, 2011, doi: 10.1109/ICCD.2011.6081391.

[15] A. Alaghi, "*The Logic of Random Pulses : Stochastic Computing,*" 2015, Ph.D. Dissertation. University of Michigan, Ann Arbor, USA.

[16] J. A. Dickson, R. D. Mcleod, and H. C. Card, " Stochastic arithmetic implementations of neural networks with in situ learning," *IEEE Int. Conf. on Neural Networks*, doi: 10.1109/ICNN.1993.298642.

[17] K. Kim, *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," *2016 53nd ACM/EDAC/IEEE Des. Autom. Conf.*, pp. 1–6, 2016, doi: 10.1145/2897937.2898011.

[18] A. Alaghi and J. P. Hayes, "On the functions realized by stochastic computing circuits," *Great Lakes Symposium on VLSI*, pp. 331–336, 2015, doi: 10.1145/2742060.2743758.

[19] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier," *2016 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays - FPGA '16*, pp. 36–41, 2016, doi: 10.1145/2847263.2847340.

[20] Z. Li, *et al.*, "Towards budget-driven hardware optimization for deep convolutional neural networks using stochastic computing," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018.

[21] T. Hirtzlin, *et al.*, "Stochastic computing for hardware ımplementation of binarized neural networks," *IEEE Access*, vol. 7, pp. 76394–76403, 2019, doi: 10.1109/ACCESS.2019.2921104.

[22] S. Aygun and E. O. Gunes, "On the simulation of software-driven stochastic computing for emerging applications," *SCONA Workshop, Des. Autom. Test Eur. DATE*, 2020.