ReLU Functions using Finite State Machines for Stochastic Computing

Kanta Yoshioka*, Ichiro Kawashima†‡ and Hakaru Tamukoh†‡
*School of Engineering, Kyushu Institute of Technology, Japan
†Gaduate Shool of Life Science and System Engineering, Kyushu Institute of Technology, Japan
‡Research Center for Neuromorphic AI Hardware, Kyushu Institute of Technology, Japan

***Abstract*—This study presents novel ReLU functions-based on stochastic computing (SC-based ReLU). The conventional SC-based ReLU for an addition method using approximate parallel counter (APC), which outputs the sum of multiple inputs every clock cycle works as a ReLU using accumulator. However, this conventional method loses the high fault tolerance, which is one of the advantages of stochastic computing (SC). Therefore, we propose an SC-based ReLU using a finite state machine (FSM) instead of an accumulator for an addition method using a multiplexer (MUX) which determines the outputs stochastically. For example, if an error occurs in the value of the accumulator in the conventional method, all subsequent operations will be wrong. However, in the case of the proposed SC-based ReLU using FSM, even if an error occurs in the current state of the FSM, it quickly returns the correct state of the FSM. Thus, the proposed SC-based ReLU using FSM for MUX is more fault tolerant than the conventional SC-based ReLU using the accumulator. However, the accuracy of the addition methods using MUX is poor because the output is determined stochastically, and a large length of bitstream (BSL) is required for accurate calculation. As the BSL becomes larger, the latency for the calculation increases, decreasing the performance. Therefore, we propose a SC-based ReLU using FSM for APC. The addition method using APC doesn't require BSL as large as the addition method using MUX to obtain correct calculation results. Compared with the conventional SC-based ReLU using an accumulator for APC, the proposed SC-based ReLU using FSM for APC shows sufficient accuracy even with a small BSL. The proposed SC-based ReLU using FSM has also a high fault tolerance, unlike the conventional SC-based ReLU using an accumulator.**

I. Introduction

Machine learning technology has brought benefits to many aspects of modern society, such as web search, image recognition and automatic translation [1]. As the societal demands increase, higher accuracy and faster computational speed are required, and the layers of the neural network (NN) are getting deeper. However, with the end of Moore's Law, the computing performances of central processing units (CPUs) and graphics process units (GPUs) has reached a ceiling, and the power consumption of GPUs has also become a serious problem in large-scale data centers. Field programable gate arrays (FPGAs) have received attention recently. By optimizing circuits to our problem, FPGAs are expected to increase the calculation speed and consume less power than GPUs [2], [3]. However, they also have the disadvantage that they are not suitable for complex operations. Stochastic computing (SC) [4], [5], which can realize arithmetic operations by bit operation, is also receiving attention. SC is a method of performing arithmetic operations based on probabilistic information, and is known to be more area-efficient in hardware implementations and easier to implement in large-scale arithmetic circuits than other general arithmetic methods. The main focus of SC research is the efficient implementation of NNs in FPGA without losing accuracy. This includes the efficient implementation of activation functions and arithmetic units to operate with a shorter bitstream length (BSL) and more accuracy.

One of the most common activation functions today is ReLU. One of the reasons why ReLU is widely used is the ease of achiving high accuracy. ReLU is less prone to gradient vanishing than conventional activation functions such as the sigmoid and hyperbolic tangent, which makes it easier to build up layers and improves performance. Therefore, it is necessary to develop and implement ReLU functions based on stochastic computing (SC-based ReLU) in FPGAs. However, the conventional SC-based ReLU [6] requires an accumulator for each neuron, which causes a heavy circuit resource consumption [7]. Because the number of neurons in the latest NNs can be several thousand or more, the implementation using conventional methods may run out of hardware resources. Additionally, conventional SC-based ReLU changes the bitstream which is a series of '1' or '0' to represent the numerical value in SC to a binary number to perform operations, which undermines the advantage of the SC's superior fault tolerance [7]. Therefore, conventional SC-based ReLU has some disadvantages.

In this study, we propose novel SC-based ReLUs, which do not have the disadvantages described above. The first proposed SC-based ReLU with FSM is for the addition method using a multiplexer (MUX), which determines the output stochastically. If an error occurs in the value in the accumulator, all subsequent output will be wrong, but if an FSM is used, the correct value will be output quickly. Therefore, using the FSM does not lose the fault tolerance, which is one of the advantages of the SC. However, this addition method using the MUX requires large BSL to calculate correctly, so it is not practical. Therefore, we propose SC-based ReLU using FSM for the addition method using an approximate parallel counter (APC) which does not require large BSL to calculate correctly. This is the second proposed SC-based ReLU with FSM. From comparison with the conventional SC-based ReLU [6], we achieved sufficient accuracy in calculation. Additionally, unlike the conventional SC-based ReLU for APC, the proposed

SC-based ReLU uses FSM instead of an accumulator has high fault tolerance.

II. Stochastic Computing

SC is a method that performs arithmetic operations based on probability information, and is known to be easier to implement area-efficient hardware than general binary arithmetic methods. For example, by using SC, we can substitute a single gate for a multiplier, which consumes a lot of hardware resources in general binary arithmetic methods.

In the domain of SC, binary numbers or values are treated as Bernoulli sequences, which are probabilities represented by random bitstreams of '0' and '1' [4] [8] . A numerical value is represented by the probability $Px(0 \leq Px \leq 1)$ of the existence of '1' in BSL. There are two ways to represent numerical values in the SC domain: unipolar and bipolar, each of which has a different definition to represent numerical values. In the unipolar domain, the number $X = Px(0 \leq X \leq 1)$, which means that we can handle only positive numbers. Conversely, in the bipolar domain, the number $X = 2Px - 1$ and $-1 \leq X \leq 1$, which means that we can handle positive and negative numbers. The order of '0' and '1' in the bitstream has no meaning. For example, if BSL is 5, (00011), (01010) and (01001) represent the same numerical value.

In SC, basic operations such as addition, subtraction, multiplication, and division can be implemented with a single gate array, which has a great advantage in reducing hardware resources. First, the two-input multiplier in unipolar is implemented with a single two-input AND element. The probability Pc of the output C is expressed in (1) using the probabilities Pa and Pb of the inputs A and B to the AND element.

$$Pc = Pa \cdot Pb \tag{1}$$

In unipolar, the probability corresponds directly to the numerical value, so $C = A \cdot B$ can be implemented with a single AND element. In this case, because the calculation result may differ depending on the position of the '1', there is almost always an error with the theoretical value, but the error can be reduced by increasing the BSL. However, there is a trade-off between the performance of the hardware such as latency and the accuracy of the calculation. A large BSL is needed to achieve high accuracy and the hardware performance will decrease.

The two-input multiplier in bipolar is implemented by a single two-input EXNOR element. Similarly, using the probabilities Pa and Pb of inputs A and B to the EXNOR element and the probability Pc of output C, (2) can be defined.

$$Pc = (1 - Pa) \cdot (1 - Pb) + Pa \cdot Pb \tag{2}$$

In this case, the output C is expanded as follows (3), thus $C = A \cdot B$ is obtained.

$$\begin{aligned} C &= 2 \cdot Pc - 1 \\ &= 2 \cdot (\frac{A \cdot B + 1}{2}) - 1 \\ &= A \cdot B \end{aligned} \tag{3}$$

Therefore, in both unipolar and bipolar, the multiplier can be realized with a single logic gate, resulting in a significant area reduction effect.

There are two main types of adders that can be used in both unipolar and bipolar. First, the addition method is based on the MUX [8], in which the probability Pc of the output C of is defined as (4).

$$Pc = Ps \cdot Pa + (1 - Ps) \cdot Pb \tag{4}$$

Pa and Pb are the probabilities of inputs A and B of the MUX and Ps is the probability of the selection signal of the MUX. The probability of the selection signal Ps is set to 0.5, i.e., the signal outputs '1' with a probability of 50% and '0' with a probability of 50%. As a result, (4) becomes (5).

$$Pc = 0.5 \cdot (Pa + Pb) \tag{5}$$

It is possible to weigh and add by changing Ps, but the output will always be scaled. This has a serious disadvantage that as the number of inputs to the adder increases, the accuracy of the calculation decreases significantly.

The second addition method is based on the approximate parallel counter (APC) [9]. This addition method based on APC does not use a stochastic bitstream but a binary number to perform the addition. The APC outputs the sum of multiple inputs every clock cycle, resulting in a $\log_2 n$ bit width binary bitstream. While a single bit output can only represent numbers in the range of [-1,1], the APC can represent a wide range of numbers and reduce the amount of information that is removed by each addition. This addition method based on APC solves the scaling problem of the MUX described above, making it possible to obtain highly accurate calculation results even with a small number of BSL. However, it also has the disadvantage of a larger circuit size compared to a MUX [10].

In addition to the basic operations described above, SC can also realize nonlinear operations using FSM. Fig. 1 shows the stochastic hyperbolic tangent function [8], which is an SC approximation of the hyperbolic tangent function. The input X is represented by the SC bitstream $x(t)$ and the state S_i of the FSM transition according to $x(t)$ to determine output $y(t)$. When $x(t) = 1$, the state moves one to the right, so $Si = Si + 1$. When $x(t) = 0$, the state moves one to left, so $S_i = Si - 1$. The output bit $y(t)$ is determined by the number of states Si. If S_i is smaller than half of the number of states, $y(t) = 0$, otherwise $y(t) = 1$. The output bitstream y after repeating the state transitions many times is approximated by the hyperbolic tangent function. Other exponential [8] and absolute value functions [11], as shown Fig. 2 and Fig. 3 can also be realized with FSM. The state S_i in both the FSM for the exponential and absolute functions moves like FSM for hyperbolic FSM in Fig. 1. N is the number of states in FSM. In Fig. 2, G is a parameter that changes the gradient of the exponential function. If S_i is smaller than $N-G-1$, $y(t) = 0$, otherwise $y(t) = 1$. The output bitstream is approximated by the exponential function [8].

In Fig. 3, if S_i is smaller than half of the number of states and i is an odd number, $y(t) = 0$, otherwise $y(t) = 1$.

If S_i is bigger than half of the number of states and i is an odd number, $y(t) = 1$, otherwise $y(t) = 0$. The output bitstream is approximated by the absolute function [11]. These functions implemented by FSMs are highly fault-tolerant. For example, even if an error occurs in the current state number S_i of the FSM, it will quickly return to the correct S_i because the number of FSM states is small in comparison with BSL. Therefore, it will output the correct bit with few incorrect outputs.

Such functions are difficult to realize in ordinary binary numbers in hardware, and the scale of the circuit becomes large. However, in SC, we can easily implement this function in hardware with FSM.

ReLU has also been proposed in SC and is shown in Fig. 4 [6]. In Fig. 4, l represents the bit width of the state number of the up down counter, which is implemented with the FSM shown in Fig. 1. The conventional SC-based ReLU is compatible with the APC-based addition method and can achieve high calculation accuracy even when the BSL is relatively small. However, it also has some disadvantages. Firstly, it requires modules that convert bitstream to a binary number, which eliminates the advantages of SC in terms of circuit simplicity and faults tolerance [7]. For example, the accumulator in Fig. 4 is used to accumulate values input from the APC every clock cycle and make positive and negative decisions. However, if an error occurs in the accumulator, all subsequent operations will be wrong. Thus, the conventional SC-based ReLU eliminates the advantage of SC.

Secondly, it consumes a lot of hardware resources. The conventional SC-based ReLU shown in Fig. 4 is installed for each neuron, which requires an accumulator and an FSM for the output decision, respectively. Recently, deep neural networks are becoming increasingly large, and the number of neurons can be thousands or tens of thousands. Therefore, it is very difficult to implement such conventional SC-based ReLU in hardware. Conventional SC-based ReLU has these disadvantages. However, using an accumulator has the advantage of excellent reconfigurability, as it can approximate any function by freely reconfiguring the subsequent circuits [10].

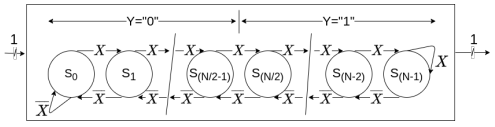


Fig. 1: Conventional hyperbolic tangent function

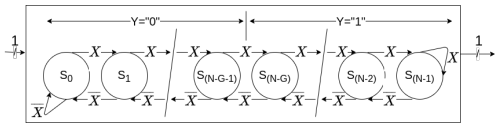


Fig. 2: Conventional exponential function

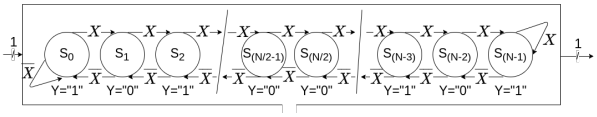


Fig. 3: Conventional absolute function

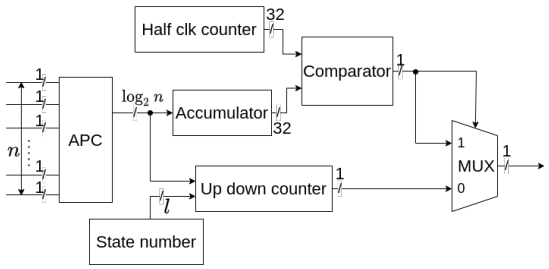


Fig. 4: Conventional SC-based ReLU function

III. Proposed Methods

There are two proposed methods of addition in the domain of SC: one using MUX and the other using APC. In this section, we introduce novel SC-based ReLUs with FSM for both the MUX-based and the APC-based addition methods.

A. ReLU Function for MUX

We introduce a novel SC-based ReLU with FSM for the MUX-based addition method. The proposed method is an FSM as shown in Fig. 5. This FSM is a combination of two FSMs. The first FSM clips a negative number to 0. In bipolar, the number of '0' and the number of '1' are equal when the value becomes 0. Therefore, the first FSM outputs '0' and '1' alternately. This FSM builds the lower half of the proposed SC-based ReLU for the MUX in Fig. 5. The second FSM outputs the input bit as it is, to represent the positive number. This FSM builds the upper half of the proposed SC-based ReLU for MUX in Fig. 5.

We explain the SC-based ReLU for MUX in detail in (6) and (7). x is the input binary number and Px is x's probability of '1'. y is the output binary number and Py is y's probability of '1'. We assume an FSM which has N states, so $S_{max} = N-1$, $S_{half} = (N/2) + 1$. We define S as the current state.

$When : 0 \leq S \leq S_{half}$

$$oData = \begin{cases} 0 \text{ (even)} \\ 1 \text{ (odd)} \end{cases} \tag{6}$$

$When : S_{half} + 1 \leq S \leq S_{max}$

$$oData = iData \tag{7}$$

When $Px \leq 0.5$, i.e., a negative number, enters the proposed SC-based ReLU, the number of '0' in the input bitstream becomes larger than '1', and thus $0 \leq S \leq S_{half}$ is approximated in (6). In this case, as shown in Fig. 5, when the input is '0' or '1' at $S = 0$, the transition is to $S = 1$. This means that the probability of outputting '1' and that of

outputting '0' are equal when $0 \leq S \leq S_{half}$, and $Py = 0.5$. This means that $y = 2 \cdot Py - 1 = 0$ and negative numbers are clipped to 0.

When $0.5 \leq Px$, i.e., a positive number, enters the proposed SC-based ReLU, the number of '1' in the input bitstream is larger than the number of '0', and thus $S_{half} + 1 \leq S \leq S_{max}$ is approximated. In this case, as shown in (7), the input data is output as it is, so $Px = Py$. From this, $y = x$. From the above, the proposed FSM works as a ReLU function.

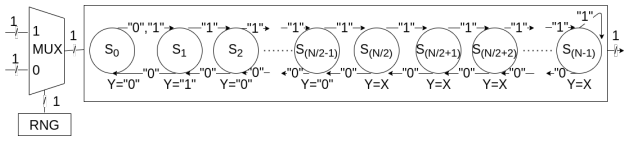


Fig. 5: Proposed SC-based ReLU Function for MUX

B. ReLU Function for APC

We introduce a novel SC-based ReLU with FSM for the APC-based addition method. The proposed SC-based ReLU for APC consists of two-stage FSMs as shown in Fig. 6. The second-stage FSM is proposed SC-based ReLU for MUX introduced in the former section.

Next, we discuss the first-stage FSM. If the number of signal lines input to the APC is n, the output bit width of the APC is $\log_2 n$. By making the output multi-bits, the APC can achieve high accuracy even with a small BSL. However, the output from APC is a binary number not a series of '0' or '1' in SC, which makes it difficult to calculate. The input of the proposed SC-based ReLU for MUX is a single bit, thus we have to convert the binary number, which is the output of APC, into a single bit in SC. The first-stage FSM works like an adapter to convert the binary number into a single bit without removing any information at this time. Ideally, we are considering of an FSM that works as a pure line function [12]. We are currently using the hyperbolic tangent for APC introduced in [13]. In this FSM which works as a hyperbolic tangent for APC, the number of state transitions V is calculated as (8) with $Count(t)$ which is the input binary number from APC. Then, the current state S moves V to next state S' like (9).

$$V = Count(t) \cdot 2 - n \tag{8}$$

$$S' = S + V \tag{9}$$

IV. Experimental Results

We verified the proposed SC-based ReLU using FSM for MUX and APC in software. First, we experimented with the numerical simulation of the proposed SC-based ReLU using defferent BSL. We generated 128 numbers for each experiment to test the SC-based ReLU accuracy. These 128 numbers were created by dividing the probability [0,1] of a single bit in the bitstream, becoming '1' by 128. In other words, if the probability of a single bit being '1' is 1, it represents 1, and if the probability of a single bit being '1' is $1/2$, it

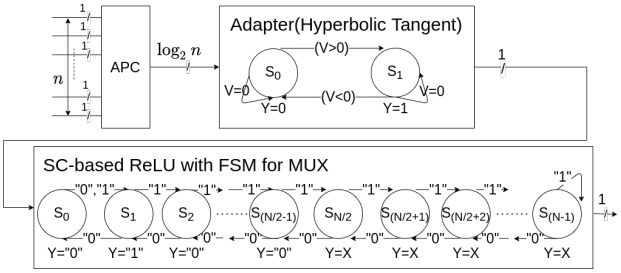


Fig. 6: Proposed SC-based ReLU Function for APC

represents 0 in bipolar. We calculated root mean square error (RMSE) between ReLU and SC-based ReLU for MUX to see how well the proposed SC-based ReLU and ReLU are fitted. Secondly, we experimented with the variation of the accuracy of the proposed SC-based ReLU with BSL to see how the RMSE changes with the BSL change. Thirdly, we randomly generated 1000 numbers for each experiment to test the proposed SC-based ReLU accuracy. We calculated the average inaccuracy (the difference between the ReLU and the proposed SC-based ReLU) of using 128 and 1024 BSL to compare with conventional SC-based ReLU [6]. Finally, we experimented with the inference for the Iris dataset using the proposed SC-based ReLU. The Iris dataset has 150 data that distinguish between three types of iris-based on data such as the length of the four petals of the iris. We separated the 150 iris data into 80% (i.e. 120) for training and 20% (i.e. 30) for testing. The network configuration was 4-4-2-3, thus, this NN has two hidden layers. The test accuracy on a single precision floating point was 93.33%. In addition, we experimented about fault torelance. The error was simulated by randomly updating the state of the FSM working as ReLU every 100 BSL. This means that when BSL is 5000, a sum of 50 errors are occured. Then, we experimented the inference for the iris dataset when the error was included.

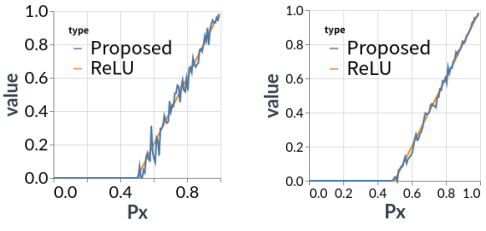
The results of the first experiment, comparing ReLU and proposed SC-based ReLU, are shown in Fig. 7 and Fig. 8. Fig. 7 shows the result of SC-based ReLU for MUX and $Px(0 \leq Px \leq 1)$ indicates the probability that the input to SC-based ReLU, which is shown in Fig. 5, is '1'. When the BSL was 256 and 1024, RMSE was 0.03336 and 0.01491 respectively. Fig. 8 shows the result of SC-based ReLU for APC. In this case, the number of bitstreams input to the APC is set to four. This number matches the number of neurons in NN for the inference Iris later. $Px(0 \leq Px \leq 1)$ in Fig. 8 shows that the probability that input to APC is "1" and we set same Px to four inputs of APC. For example, $Px = 1$ means that the APC approximately always outputs 4; $Px = 1/2$ means that the APC approximately always outputs 2. When the BSL was 256 and 1024, the RMSE was 0.19141 and 0.18711 respectively.

The results of the second experiment, observing the change of RMSE, are shown in Fig. 9. As the BSL is increased, the RMSE of SC-based ReLU for MUX is decreased but the RMSE of SC-based ReLU for APC is not decreased.

The results of the third experiment, comparing inaccuracy of conventional and proposed SC-based ReLU, are listed in Table I. We used the same test method in [6] to test the proposed SC-based ReLU for APC; however, we did not use the same test method to test the proposed SC-based ReLU for MUX. The proposed SC-based ReLU is for the MUX, whereas the conventional one is for the APC. Therefore, in this test, we did not calculate addition using the MUX, and input to a series of '0' or '1', which is made from the generated random numbers by the binarization method in SC, to the FSM directly. For this reason, the verification method is different from the method used in [6]. We obtained a small average inaccuracy in the SC-based ReLU for both MUX and APC.

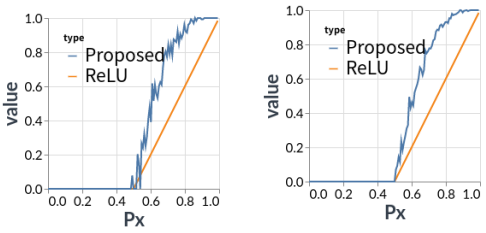
The results of the fourth experiment, the inference of the Iris dataset, are shown in Fig. 10. Note that the horizontal axis, BSL, is a logarithmic axis in Fig. 10. When the BSL is small, the accuracy is low, but as it increases, the accuracy increases for the proposed SC-based ReLU for MUX. It cannot be calculated correctly when BSL is small, because the addition using the MUX is always scaled, From these characteristics, it can be inferred that the low accuracy when the BSL is small is caused by the MUX, and the proposed SC-based ReLU for MUX is working correctly. While the inference was calculated by APC and the proposed SC-based ReLU for APC, we achieved a higher accuracy than using the MUX even though the BSL was small. However, the SC-based ReLU for APC does not fit well into the ReLU, and the RMSE is large in Fig. 8 and Fig. 9. Therefore, the maximum accuracy of the SC-based ReLU for APC is 86.67%, which is not as high as the 93.33% calculated on a single precision floating point.

The results of the fourth experiment, the inference of the Iris dataset including errors, are also shown in Fig. 10. In the case of the MUX, the maximum accuracy including simulated errors was 93.33%, the same as the case without error. On the other hand, in the case of the APC, there was a slight decrease in accuracy, with the maximum accuracy decreasing by about 3.33 points from 86.67% to 83.33%. However, in this case, the number of test data was only 30. In other words, just one wrong data would cause the decrease in accuracy by 3.3 points. From this, it can be said that there is almost no change in accuracy. From these results, it is found that both of the proposed SC-based ReLUs have high fault tolerance.



(a) BSL is 256 (b) BSL is 1024

Fig. 7: Propsoed SC-based ReLU for MUX



(a) BSL is 256 (b) BSL is 1024

Fig. 8: Propsoed SC-based ReLU for APC

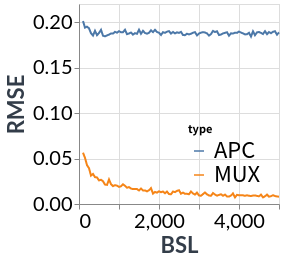


Fig. 9: Change in RMSE when BSL is increased

V. Discussion

First, we discuss the accuracy of the proposed SC-based ReLU. We discuss about the proposed SC-based ReLU for MUX. The proposed SC-based ReLU with FSM has a high fault tolerance, which is one of the advantages of SC that conventional SC-based ReLU with accumulator loss. Fig. 7 shows that as the BSL increases, the RMSE decreases, and accuracy increases. In general, [256,1024] BSL are used for

TABLE I: Average Inaccuracy (lower is better) Comparison between Conventional and Proposed

	BSL=128	BSL=1024
Conventional	0.057	0.031
Proposed for MUX	0.007	0.003
Proposed for APC	0.026	0.045

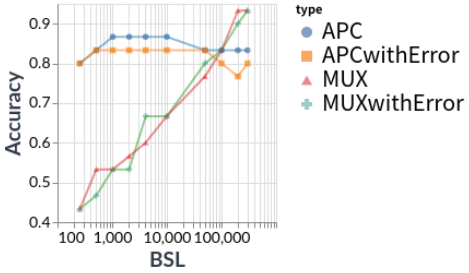


Fig. 10: Change in Iris test accuracy when BSL is increased

SC operations. When the BSL was 256, RMSE was 0.03336. When the BSL was 1024, RMSE was 0.01491. Therefore, it can be inferred that enough accuracy is obtained for SC operations. Table I also shows that the accuracy of SC-based ReLU for MUX is sufficient when compared to the conventional method. However, the decrease in accuracy owing to the MUX is not taken into account, because a series of "0" or "1" is directly input to the proposed SC-based ReLU using FSM. Futermore, Fig. 10 shows that the accuracy increases as the BSL increase, compared to the low accuracy when the BSL is small. As mentioned in Section II, information is significantly reduced in the case of addition using MUX, BSL should be increased to obtain correct calculation results. From the characteristics of this MUX-based addition method, it can be inferred that the proposed SC-based ReLU for MUX works correctly. However, it is not realistic to perform 300,000 operations for the Iris 3-class classification.

Therefore, we consider the proposed SC-based ReLU for APC which is an accurate addition method even when the BSL is small. We substitute the hyperbolic tangent for APC [13] as the first-stage FSM, which functions as an adaptor in Fig. 6. Therefore, the approximate shape of the graph is shown in Fig. 9. The negative part is properly clipped to zero, but the positive part shows the positive form of the hyperbolic tangent. Additionally, Fig. 9 shows that the RMSE did not decrease when the BSL was increased and it ranged between approximately 0.185 and 0.190. Table I also shows a slightly lower accuracy than the conventional method. However, Fig. 10 shows the result of the Iris inference for the addition method using APC, which has improved the disadvantages of the addition method using MUX. From the results in Fig. 10, it can be observed that higher accuracy is obtained with a smaller BSL than when MUX is used for addition. However, there is a difference with 93.33% accuracy when inferring with a single-precision floating point. We think that large RMSE of the proposed SC-based ReLU for APC is one of the most important reasons for this difference in accuracy between inferring with single-precision floating point and without it. We substituted the hyperbolic tangent for APC [13] as the first-stage FSM in the proposed SC-based ReLU for APC. From Fig. 8 and Fig. 9, the graph of the proposed SC-based ReLU for APC is very different from that of the correct ReLU, thus the inferring accuracy is decreased. The accuracy may deteriorate further depending on the weight data for inference.

Next, we discuss the hardware resources. We compare the hardware resources in the proposed SC-based ReLU using FSM for APC with the conventional SC-based ReLU using an accumulator for APC. In the conventional SC-based ReLU shown in Fig. 4, there are two large resource elements, the accumulator and the up down counter. The accumulator needs to be large depending on the number of inputs to the APC n and BSL, and the up down counter is realized with the FSM as in [13]. When comparing this to the proposed SC-based ReLU, the proposed SC-based ReLU for APC is expected to reduce hardware resources by the difference between the accumulator and the proposed FSM. The reduction of hardware resources in a neuron is small, but as the number of neurons increases, the overall reduction becomes significant.

VI. Conclusion

In this study, we proposed SC-based ReLUs using the FSM for MUX and APC. In the proposed method, we implemented the SC-based ReLU using the FSM unlike the conventional SC-based ReLU using the accumulator [6]. The proposed SC-based ReLU using FSM is more fault-tolerant than the conventional SC-based ReLU using the accumulator. Additionally, the hardware resources are expect to be reduced. In the case of addition using MUX, the loss of accuracy is so severe that it is impractical, but in the case of addition using APC, high accuracy was obtained even with a small BSL. However it did not reach the accuracy on a single precision floating point. The proposed SC-based ReLU for APC has a large RMSE and does not have a good fit to ReLU. In the future, the proposed SC-based ReLU for APC can be further improved in terms of accuracy.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". *nature*, 521(7553):436–444, 2015.

[2] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. "Performance comparison of FPGA, GPU and CPU in image processing". In *2009 international conference on field programmable logic and applications*, pages 126–131. IEEE, 2009.

[3] Yuexuan Tu, Saad Sadiq, Yudong Tao, Mei-Ling Shyu, and Shu-Ching Chen. "A power efficient neural network implementation on heterogeneous FPGA and GPU devices". In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 193–199. IEEE, 2019.

[4] Brian R Gaines. "Stochastic computing systems". In *Advances in information systems science*, pages 37–172. Springer, 1969.

[5] Paishun Ting and John P Hayes. "On the role of sequential circuits in stochastic computing". In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 475–478, 2017.

[6] Zhe Li, Ji Li, Ao Ren, Ruizhe Cai, Caiwen Ding, Xuehai Qian, Jeffrey Draper, Bo Yuan, Jian Tang, Qinru Qiu, et al. "HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(8):1543–1556, 2018.

[7] Yawen Zhang, Runsheng Wang, Yixuan Hu, Weikang Qian, Yanzhi Wang, Yuan Wang, and Ru Huang. "Accurate and Energy-Efficient Implementation of Non-Linear Adder in Parallel Stochastic Computing using Sorting Network". In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[8] Bradley D Brown and Howard C Card. "Stochastic neural computation. I. Computational elements". *IEEE Transactions on computers*, 50(9):891–905, 2001.

[9] Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. "Approximate derandomizer for stochastic circuits". In *2015 International SoC Design Conference (ISOCC)*, pages 123–124. IEEE, 2015.

[10] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. "A Survey of Stochastic Computing Neural Networks for Machine Learning Applications". *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[11] Peng Li, David J Lilja, Weikang Qian, Marc D Riedel, and Kia Bazargan. Logical computation on stochastic bit streams with linear finite-state machines. *IEEE Transactions on Computers*, 63(6):1474–1486, 2012.

[12] Yidong Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. An energy-efficient stochastic computational deep belief network. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1175–1178. IEEE, 2018.

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