JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 16, No. 2, June 2022, pp. 79-87

Improving Speed of MUX-FSM-based Stochastic Computing for On-device Neural Networks

Jongsung Kang and Taewhan Kim*
Department of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea
jskang@ssl.snu.ac.kr, tkim@ssl.snu.ac.kr

Abstract
We propose an acceleration technique for processing multiplication operations using stochastic computing (SC) in on-device neural networks. Recently, multiplexor driven finite state machine (MUX-FSM)-based SCs, which employ a MUX controlled by an FSM to generate a (repeated but short) bit sequence of a binary number to count up for a multiplication operation, considerably reduce the processing time of MAC operations over the traditional stochastic number generator (SNG) based SC. Nevertheless, the existing MUX-FSM-based SCs still do not meet the multiplication processing time required for the wide adoption of on-device neural networks in practice even though it offers a very economical hardware implementation. In this respect, this work proposes a solution that speeds up the conventional MUX-FSM-based SCs. Precisely, we analyze the bit counting pattern produced by MUX-FSM and replace the counting redundancy by shift operation, resulting in a shortening of the length of the required bit sequence significantly, together with analytically formulating the number of computation cycles. Through experiments, we have shown that the enhanced SC technique can reduce the processing time by 44.1% on average over the conventional MUX-FSM-based SCs.

Category: Embedded Systems

Keywords: Neural processing unit; Hardware; Stochastic computing; Embedded systems

I. Introduction

On-device (or edge device) neural networks accelerator is gaining popularity mainly because it eliminates network latency in processing data and it also enhances privacy safety as it eliminates the need to send sensitive data to the cloud. For example, an autonomous driving solution requires very low processing latency and no failures, which might not be attained with the current or near-future wireless networking standards. Smartphones and other handheld devices utilize machine learning for many applications, e.g., photo processing and text translation, which should take into account privacy issues.

On-device processing costs should be minimized by limiting power consumption, logic area, and demands for higher performance for enabling advanced applications. The methods used for speeding up the neural network processing can be classified as (1) quantizing bit-width or lowering computation precision, which directly reduces logic area and power consumption but without affecting the accuracy; (2) pruning model components, weights, and convolution filters which are very unlikely to affect the final results; and (3) alternating computing methods such as log-scale and stochastic computing (SC). This work follows stochastic computing to speed up the processing time.

Open Access **http://dx.doi.org/10.5626/JCSE.2022.16.2.79** **http://jcse.kiise.org**

Received 17 March 2022; **Accepted** 25 May 2022
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

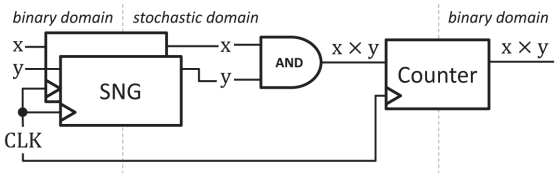

Fig. 1. Illustration of the original stochastic computing (SC) structure for a multiplication operation. Two inputs x and y on a multiplication are converted to stochastic numbers, which are then applied to bitwise AND operations, followed by counting up the 1-bits of the AND outputs to produce a binary number of the multiplication result.

SC [1-5] is a collection of techniques that represent continuous values by streams of random bits. Complex computations can then be computed by simple bitwise operations on the streams. Fig. 1 illustrates an architecture for SC-based multiplication, in which the two SNG (stochastic number generators) generates bit streams such that the ratios of 1-bits of the streams should be very close to the values of multiplication of the inputs x for activation and y for weight in the neural network. Then, bitwise AND operations are applied to the bit streams coming out from SNGs in Fig. 1, producing a bit stream corresponding to the value of $x \times y$, from which its binary representation is obtained by counting the number of 1-bits in the bit stream. Though the supporting hardware logic for bitwise operations in SC is simple, it has a couple of limitations: (1) *a considerably large logic area for SNGs* and (2) *a very long bit stream* to maintain no-accuracy-loss operation as well as representation. A bit stream with a length of up to 2^n should be prepared to be probabilistically exact to represent a value that uses n bits in a binary representation. Consequently, the long bit streams, particularly for the values close to 0, affect unfavorably the SC's ultimate objective of fast processing time through the simple bitwise operations.

To overcome the limitations, recently a number of deterministic SC structures have been proposed. One such structure, called stochastic computing neural network (SC-DNN) [6-8], is shown in Fig. 2. SC-DNN replaces the two expensive SNGs in Fig. 1 with a multiplexer (MUX) and a simple counter-based finite state machine (FSM), in which the MUX inputs are the bit values of activation input x and the bit stream is formed by an iterative selection of MUX inputs that are controlled by the FSM. However, since this SC structure still requires the counter in FSM to operate 2^n clock cycles in the worst case (i.e., the y value close to the largest magnitude), saving the multiplication processing time is limited. However, it is able to reduce the averaged processing time significantly in comparison to the processing time used by SNG-based SC (The details on the semantics of counter-based FSM in Fig. 2 will be described in Section II. In summary, we assume the multiplication inputs are all unsigned numbers. Handling signed numbers done in

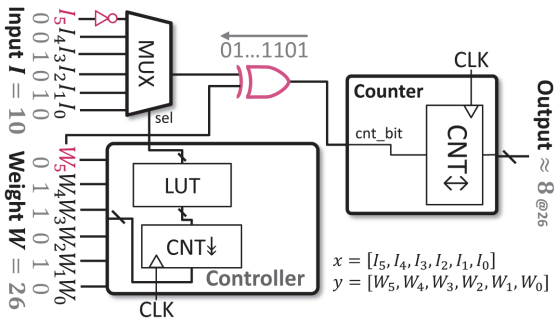

Fig. 2. Structure of MUX-FSM-based SC [6].

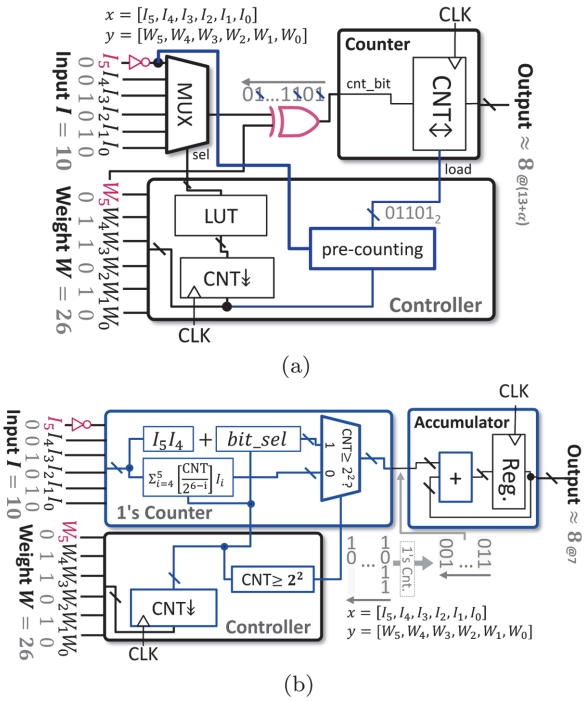

Fig. 3. Enhanced structures of MUX-FSM-based SC. (a) Structure of MUX-FSM-based SC with pre-counting [9]. (b) Structure of MUX-FSM-based SC with bit-parallel processing [6].

all existing architectures can be naturally migrated into our proposed architecture with no special modification. The inverter gate, marked in red color, placed at one of the MUX inputs on every MUX-FSM-based SC architecture in this paper accounts for the support of signed operations).

Fig. 3(a) [9] and Fig. 3(b) [6] show the architectures that enhance the multiplication processing of the MUX-FSM-based SC in Fig. 2.

The structure in Fig. 3 called *MUX-FSM-based SC with pre-counting* exploits the fact that the most significant bit (MSB) (i.e., the highest-order bit) of x in the MUX inputs should be selected every two clock cycles in a row to form a bit sequence if the structure in Fig. 2 is used and it is possible to know the total count of the MSB's 1-bits in the sequence by merely examining the MSB value in

advance. Thus, the structure of MUX-FSM-based SC with pre-counting instantaneously sets the initial value of its counter in Fig. 3(a) to a constant (preprocessed) value and then starts to count up the rest as does the structure in Fig. 2, thereby reducing the processing time roughly by half at the expense of more hardware resource i.e., *pre-counting* block in Fig. 3(a).

On the other hand, the structure in Fig. 3(b) called *MUX-FSM-based SC with bit-parallel processing* performs the counting process with repetition for the 1-bits in the MUX inputs corresponding to multiple high-order bits in x concurrently at additional cost (hardware), which amounts to the three new sub-blocks placed immediately before the MUX in Fig. 3(b).

One common rule that governs the 1-bit counting process in the MUX-FSM-based SC structures in [6, 9] is that the total counting time in a multiplication ($x \times y$) is tightly bounded by the absolute value of y. This work proposes a method of lowering down this bound with no accuracy loss to speed up the counting process. Precisely, (1) we introduce a novel concept called *split-shift* and apply it to y on all MUX-FSM-based SC structures. (2) We analytically formulate the number of computation cycles in terms of the bit width and value of multiplication input W. (3) In addition, we show that applying our counting technique integrated with the existing MUX-FSM-based SCs to a set of neural network benchmark models is able to reduce the multiplication processing time by 58.2%, 38.1%, and 20.1% on average over the conventional MUX-FSM-based SC in Fig. 2, and its two enhanced SCs in Fig. 3(a) and 3(b), respectively.

II. PRELIMINARY

This section explains how the existing MUX-FSM-based SCs works, which is essential to understanding our work in Section III.

MUX-FSM-based SC: The upper part in Fig. 4 shows how MUX-FSM based SC in [6] calculate $x \times y$ where $x = I = [I_5 I_4 I_3 I_2 I_1 I_0] = 001010_2 (= 10)$ and $y = W = 011010_2 (= 26)$. To calculate $I \times W$ by SC, MUX-FSM-based SC assigns the bit values of I_5, I_4, I_3, I_2, I_1, and I_0 to the MUX inputs and produces a sequence of MUX outputs by applying the sequence, S, of MUX inputs' index values [5 4 5 3 5 4 5 2 5 4 5 3 5 4 5 1 5 4 5 3 5 4 5 2 5 4], shown in the middle of Fig. 4. It should be noted that the index sequence S, coming out from an FSM, is known in advance and is independent of the values of W and I. In addition, the length of S is exactly equal the value of W. Thus, the bit stream corresponding to the S is [1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0]. For example, since value 5 in the first element in S indicates $\bar{I}_5$ in the MUX inputs, the first element in the bit stream is $\bar{I}_5 = \bar{0} = 1$ while the second and third elements are $I_4 = 0$ and $\bar{I}_5 = \bar{0} = 1$, and so on. MUX-FSM-based SC then counts the number of 1-bit values in the bit stream, one cycle at a time, taking a total of 26 (=W) clock cycles.

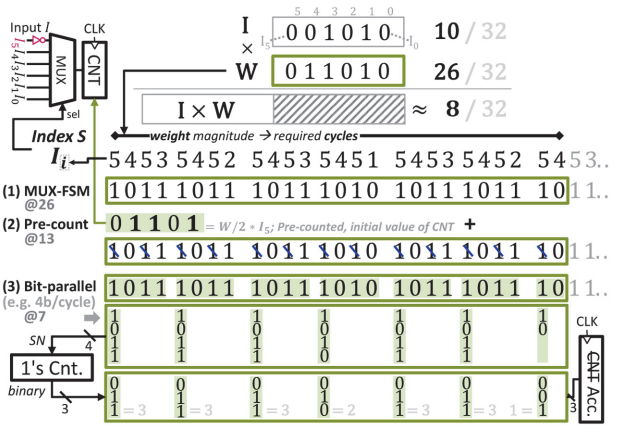

Fig. 4. Illustration of the process of multiplication by the existing (1) MUX-FSM-based SC, (2) MUX-FSM-based SC with pre-counting, and (3) MUX-FSM-based SC with bit-parallel processing.

The rationale for using the fixed index sequence S, shown in I_i in Fig. 4, is the following: For n-bit W and n-bit I, the outcome of $I \times W$ can be approximated by evaluating

$$I \times W = W \cdot \sum_{j=1}^{n} 2^{-j} \cdot I_{n-j} \approx \sum_{j=1}^{n} \lceil W * 2^{-j} \rceil \cdot I_{n-j} . \quad (1)$$

Thus, Eq. (1) tells us that $I \cdot W$ can be approximated by counting the value of each I_{n-j} $\lceil W * 2^{-j} \rceil$ times. For example, for $n = 6$, the right term in Eq. (1) becomes

$$\lceil W * 2^{-1} \rceil \cdot I_5 \cdot \lceil W * 2^{-2} \rceil \cdot I_4 + \cdots + \lceil W * 2^{-6} \rceil \cdot I_0 \quad (2)$$

which means that the number of counting I_j, $j = 5, \ldots 1$ is twice more than that of I_{j-1}. Thus, the W value can be distributed into W bits (i.e., index sequence) of I_5, …, I_0 such that the total sum of the bit values is the value in Eq. (2) (We omit the details on the construction of the index sequence, which can be found in [6]). Thus, the number of clock cycles required in the worst case is bounded by 2^n.

MUX-FSM-based SC with pre-counting: MUX-FSM-based SC with pre-counting [9], shown in Fig. 3(a) and Fig. 4(2), makes use of the fact that the MSB of I appears in the index sequence for every other position (e.g., I_5 in Fig. 4(2)), initially setting the counter to half of the weight magnitude (e.g., initial value to 26/2 = 13 = 1101_2 in Fig. 4(2)). Consequently, the required number of clock cycles is $\lceil W/2 \rceil$, which is bounded by 2^{n-1}.

MUX-FSM-based SC with bit-parallel processing: MUX-FSM-based SC bit-parallel processing [6], shown in Fig. 3(b) and Fig. 4(3), selects and counts r bits

altogether (in a single cycle) at the price of expensive hardware rather than one bit at a time, thereby reducing the processing clock cycles by $\lceil W/r \rceil$, equivalently up to $\lceil 2^n/r \rceil$ (e.g., for $r = 4$ in Fig. 4(3), $\lceil W/r \rceil = \lceil 26/4 \rceil = 7$ clock cycles). The additional hardware is a counter with accumulation capability, whose responsibility is to take appropriate r bits from I and count the number of 1-bits in a single cycle.

III. THE PROPOSED MUX-FSM-BASED SC

A. New Algorithm for Stochastic Computing

A common concept used in the MUX-FSM-based SCs in computing $I \times W$ is to count the number of 1-bits in a bit stream, S, of size W where S is composed of bits on I in two's complement binary representation. Our key idea of speeding up the counting process is to split W in n-bit binary representation into two parts, W_H and W_L, of equal bit length (i.e., $|W_H| = |W_L| = n/2$) such that $W_H||W_L = W$ and count the bits in S corresponding to W_H very efficiently by exploiting the abundance of common bit sub-streams in S [10].

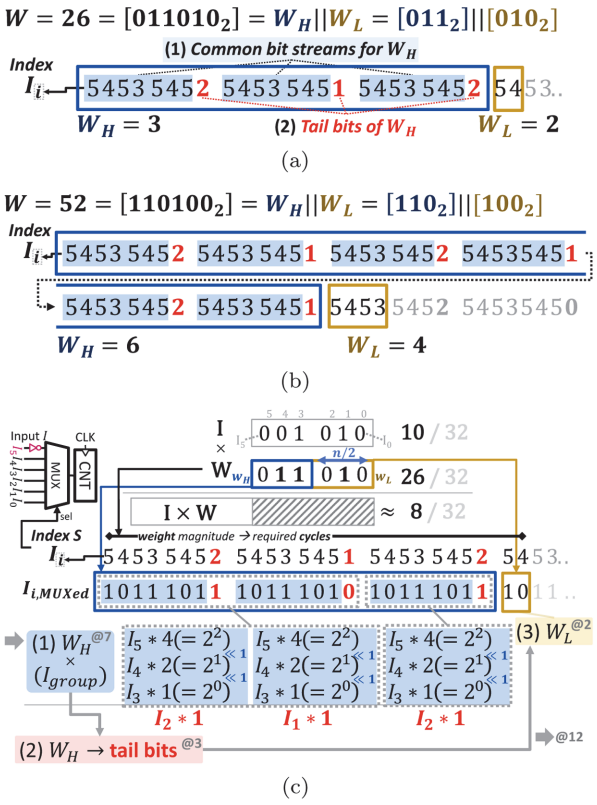

Fig. 5. Examples of splitting weight, grouping index sequence, and proposed steps for computation. (a) Split example for W = 26. (b) Split example for W = 52. (c) Splitting of weight bits by two, grouping input bit stream regarding upper weight bits.

For example, Fig. 5(a) shows the index sequence of I corresponding to S for $W = W_H||W_L = [011_2]|||[010_2] = 26$. We partition the linear sequence from the leftmost to the right so that each group has 8 ($=2^{|W_L|}$) elements. We can observe that every group except the last one has a common index subsequence [5 4 5 3 5 4 5]. We call the 7-length bit sub-streams of the common index subsequence in S common bit streams for W_H and the last (single) bits right after the common bit streams in S tail bits for W_H. In addition, we call the bit stream of the last group whose bit length is W_L bit stream for W_L. For $W = 26 = [011_2]|||[010_2]$ and $I = [I_5 I_4 I_3 I_2 I_1 I_0]$, its common bit streams for W_H, tail bits for W_H, and bit stream for W_L are shown in blue, red, and yellow boxes in Fig. 5(a), respectively, while Fig. 5(b) shows the common bit streams and tail bits for W_H, and the bit stream for W_L when $W = [110_2]|||[100_2]$.

Our 1-bit counting algorithm for MUX-FSM-based SC is performed in three steps: (Step 1) *counting common bit streams*, (Step 2) *counting tail bits*, and (Step 3) *counting the rest*. The overall flow of our algorithm is shown in Fig. 6(a).

Step 1 (Counting common bit streams for W_H): Since the common bit streams are already known when the multiplication input bit-width n is given, we will count the number of 1-bit values very easily. For example, if

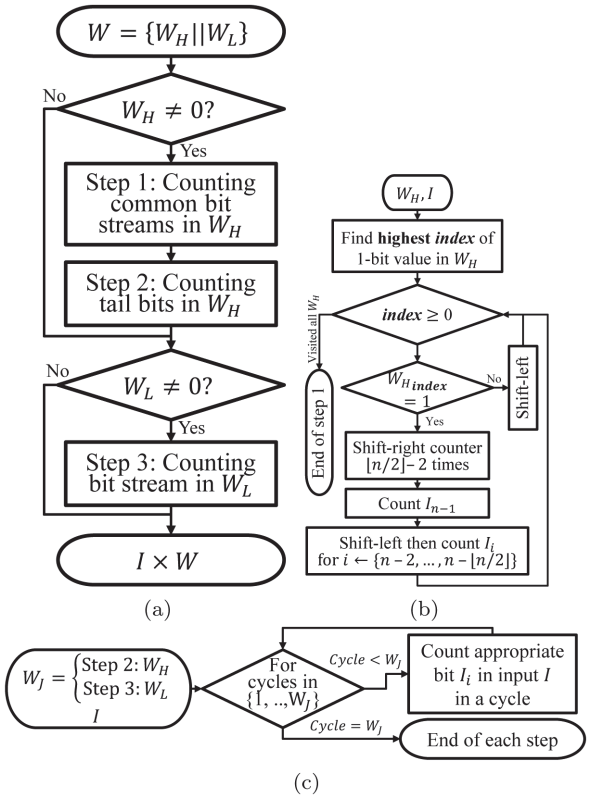

Fig. 6. Flow of our proposed algorithm. (a) Overall flow. (b) Step 1. (c) Steps 2 and 3.

Table 1. Number of computation cycles of our proposed and reference schemes for $I \times W$

Scheme	Cycles[a]
SC + serial [6]	W
SC + precount[b] [9]	$\lceil W/2 \rceil$
SC + parallel [6]	$\lceil W/r \rceil$
Ours + serial	$popcount(W_H) \cdot (2 \cdot \lceil n/2 \rceil - 1) + W_H + W_L$
Ours + precount[b]	$popcount(W_H) \cdot (2 \cdot (\lceil n/2 \rceil - 1) - 1) + W_H + \lceil W_L/2 \rceil$
Ours + parallel	$\lceil log_2(W_H+1) \rceil + \lceil W_H/r \rceil + \lceil W_L/r \rceil$

[a]Assume n-bit operands weight $W = W_H * 2^{n/2} + W_L$, *popcount* = Hamming weight.
[b]Excluding pre-counting cycles.

$n = 6$, [5 4 5 3 5 4 5] is the common input bit index stream, for which we have to count the value of I_5 4 times, I_4 2 times, and I_3 once. Consequently, it suffices to count up the value of I_5, followed by simultaneously shifting the counter and counting up the value of I_4, and finally simultaneously shifting the counter and counting up the value of I_3. Thus, the counting process for one common bit stream takes 3 (= 1+1+1) clock cycles. Then, the count value obtained can be doubled by a one-shift operation. Since there are three common bit streams in the example for $W_H = 011_2$, the total time is 7 = (3 + 1 (for shift operation) + 3 (for processing one more common bit stream)). In comparison with the conventional MUX-FSM SC in [6], which needs 21 (= 7×3) clock cycles, we are able to reduce the number of clock cycles from 21 to 7. The shifting and counting process is shown in Fig. 6(b) and the blue shaded part in Fig. 5(c) illustrates how the three common bit streams for WH are counted when $W = 011010_2$.

Step 2 (Counting tail bits for W_H): Since the bit index of I of the tail bits is already known, we can store the index value in a lookup table (LUT), and fetch the index value according to the position of common bit streams. For example, the red numbers 2, 1, and 2 in Fig. 5(c) are the tail bit indices of the 1st, 2nd, and 3rd 8-bit groups in S, respectively. Thus, for the example in Fig. 5(c), it needs 3 (= W_H) clock cycles to count up the values of the input bits I_2, I_1, and I_2.

Step 3 (Counting the bit stream for W_L): It is to count up the values of the input bits corresponding to the last group in S. Since the bit length of the group is exactly W_L, for the example in Fig. 5(c), it requires 2 (= W_L) clock cycles to count up all bit values. The counting process of Steps 2 and 3 is shown in Fig. 6(c).

It should be noted that our split-shift-based algorithm can be extended to support the MUX- FSM-based SC with pre-counting [9] and MUX-FSM-based SC with bit-parallel processing [6] by simple treatments in the following: applying split-shift to MUX-FSM with pre-counting may cause a conflict with its shift rule. However, it can be simply resolved by non-shifting the pre-counted initial value, while for MUX-FSM with bit-parallel processing, the parallel counting process can be updated in a way that the bits in the common bit streams and tail bits in a bit sequence S should be counted in separate counting steps rather than counting every r bits in S at once. Table 1 summarizes the analytic analysis of time complexity (i.e., the number of clock cycles) used by the existing three MUX-FSM-based SCs and our MUX-FSM using the split-shift concept. The three terms in the formulas of our models account for the amounts of clock cycles spent on the three steps of our algorithm in Fig. 6. The curves in Fig. 7 show the visualization of the changes in clock cycles required as the value of input W of multiplication varies. In summary, when we apply our

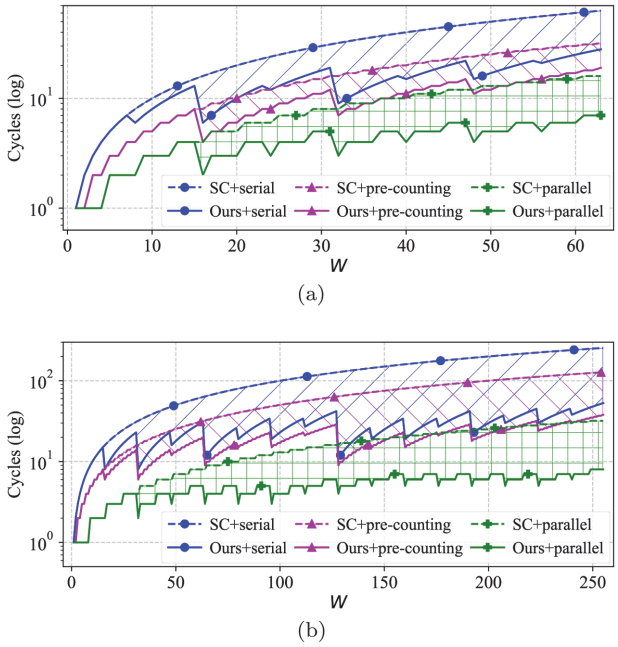

Fig. 7. Visualization of the analytic analysis on the number of computation cycles in Table 1 by varying the value of multiplication operand W. (a) n = 6-bit (r = 4). (b) n = 8-bit (r = 4).

counting algorithm to "serial," "pre-counting," and "parallel" schemes, the average theoretical cycle reduction ratios are 45.7%, 32.2%, and 39.1% for 6-bit, and 69.8%, 60.3%, and 57.2% for 8-bit.

B. The Supporting Hardware Architecture

Fig. 8 shows the hardware architecture that supports our MUX-FSM-based SC with split-shift processing, which upgrades the conventional architecture in Fig. 2. Our architecture has three features: (1) installing *a logic circuit for one-bit shift operation*, (2) *a master FSM* that controls our three-step algorithm of counting 1-bits in the bit stream, and (3) *three slave FSMs* guided by the master FSM and carries out the rules of 1-bit counting process for the three steps sequentially, one by one.

To save the clock cycles in our counting processing, it is necessary to install a logic block that can perform a 1-bit shift operation, which helps relieve the (redundant) counting burden on the 1-bit counter block in the blue box in Fig. 8 (Note that it suffices to have a simple shifter to perform just one bit shift operation rather than an expensive shifter like barrel shifter that can perform variable-length bit shift operation). To temporarily keep the bit values coming out from the least significant bit (LSB) in the course of shift operations, the 1-bit counter block has a register internally.

The role of the master FSM, shown as the green box in Fig. 8, controls the three slave FSMs, shown as the blue, red, and yellow boxes in Fig. 8. That is, it sequentially triggers the three steps of 1-bit counting process. It skips steps 1 and 2 if $W_H = 0$ while it skips step 3 if $W_L = 0$.

The slave FSM for step 1 performs the following. First, it receives one common bit stream of W_H from the small MUX in Fig. 8 and counts 1-bit values. Since the slave FSM has to count 1-bit values on the W_H number of common bit streams, it asks the shifter for counting up. The LUT in the slave FSM is responsible for storing the correct signals to control the shift operations according to the value of W_H. We use an additional hardware logic for identifying the leading 1-bit value in the binary representation of W_H to save cycles in scanning bits for small values of W_H. The role of the slave FSMs for steps 2 and 3 are basically identical to that of the conventional FSM-MUX-based SC. The slave FSM for step 2 scans and counts up 1-bit values on the W_H tail bits whose corresponding MUX input indexes are stored in LUT. Likewise, the slave FSM for step 3 counts 1-bit values on the W_L last bits on the bit streams of length W, sharing the counter logic block with the slave FSM for step 2.

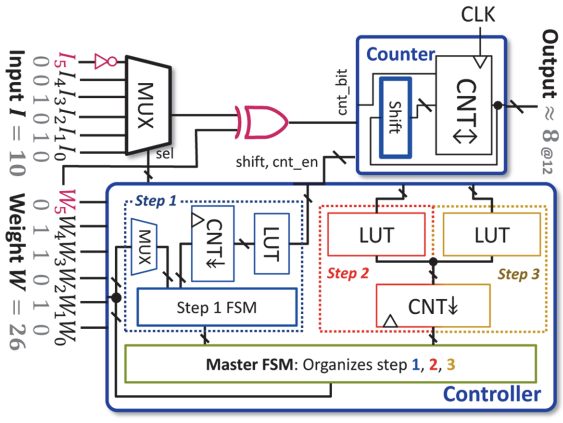

Fig. 8. The architecture supporting our MUX-FSM SC.

IV. EXPERIMENTAL RESULTS

A. Experiments Setup

First, we analytically calculate the number of clock cycles required to complete the multiplication following the step in Eq. (2) by using the conventional models of MUX-FSM-based SC, MUX-FSM-based SC with pre-counting, and MUX-FSM-based SC with bit-parallel processing as well as our proposed model of MUX-FSM-based SC with split and shift. It should be noted that all the models exhibit the same approximation accuracy since every model computes the formulation in Eq. (2). Moreover, besides referring to the theoretical analysis in Sections II and III, depending on the models applied, we include the clock cycles spent on their auxiliary circuits (via circuit simulation) in the total count of clock cycles. Then, we implement this cycle counting process for n-bit by n-bit multiplication in Eq. (2) using Python script.

Then, we implement our MUX-FSM SC model and the conventional three MUX-FSM SC models with Verilog HDL and synthesize them into RTL design by using Synopsys Design Compiler (version L-2016.03-SP5-5) with an industrial 28nm cell library, setting the clock frequency to 500MHz. For the rest, we applied default tool options. Cell area and (vectorless) estimated power consumption are extracted from the synthesized designs by using the Design Compiler. We generated the selection sequence of MUX input bits using the Python script.

B. Performance Comparison

Table 2 shows the comparison of the number of average clock cycles used by the conventional MUX-FSM-based SC (SC + serial), MUX-FSM SC with pre-counting (SC + pre-counting), MUX-FSM-based SC with bit-parallel (SC + parallel), and our models. The cycle reductions by our models are consistent and more effective as the input bit width of multiplication increases.

Fig. 9 shows the distribution of the number of clock cycles as the (weight) input value changes in the

Table 2. Comparison of the number of average clock cycles used by the conventional MUX-FSM SC models and ours for multiplication of input bit-width $n = 6$ and $n = 8$

Scheme	$n = 6$		$n = 8$	
	#Cycles	Reduction (%)	#Cycles	Reduction (%)
SC + serial [6]	16.00	-	64.00	-
SC + precount [9]	9.25	-	33.25	-
SC + parallel [6]	4.38	-	8.44	-
Ours + serial	8.64	46.0	18.93	70.4
Ours + precount	7.09	23.3	15.67	52.9
Ours + parallel	3.31	24.3	4.39	47.9

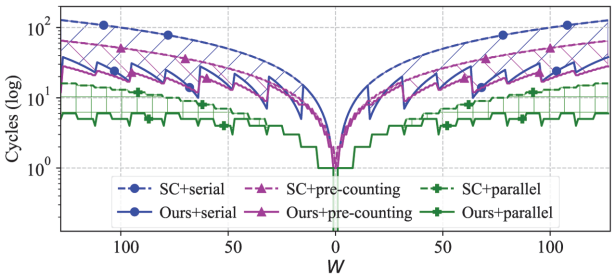

Fig. 9. Changes in the number of clock cycles used by our and existing SC models as weight W changes in the multiplication of bit-width $n = 8$.

multiplication of $n = 8$. It reveals that the bigger the weight magnitude is, the more the number of clock cycles required, but the gap in our models is much shorter than that of the conventional models.

C. Hardware Area and Energy Comparison

Table 3 compares the hardware area, the amount of power and energy consumed by the conventional MUX-FSM-based-SCs, non-SC, i.e., the conventional fast multiplier with inputs in fixed-point binary representation, and ours for computing a single multiplication $I \times W$. Our architecture includes a new shift module as well as one more FSM in comparison with MUX-FSM-based SC + serial [6]. Thus, the area increases by 67% for a 6-bit multiplication, causing more power consumption from 38.36 μW to 48.97 μW. However, due to the drastic reduction of clock cycles, the energy consumption drops by 31%, from 1.228 pJ to 0.846 pJ. Moreover, for 8-bit multiplication, the energy-saving by our SC is considerable, from 6.131 pJ to 2.275 pJ, reducing the energy consumption by 63%. In comparison with MUX-FSM with pre-counting, which eliminates the time for processing MSB in input I, our pre-counting model slightly saves the area over our serial version, even installing small pre-counting logic. On the other hand, in comparison to bit-parallel processing, our SC uses more circuit area over our serial one, adversely affecting energy saving in 6-bit multiplication. However, for 8-bit multiplication, our bit-parallel model is very effective, drastically saving the energy consumption due to much faster processing time.

Table 4 compares the hardware area, the amount of power, and energy consumed by the conventional MUX-FSM-based SCs, non-SC, and ours for computing the

Table 3. Comparison of hardware area and energy consumed by our MUX-FSM-based SC model and the conventional models for processing a single multiplication: $I \times W$

	Scheme	Area	Power (μW)	Energy (pJ)	Reduction (%)	Clock (ns)
$n = 6$	Non-SC	97.93	38.45	0.077	-	1.89
	SC + serial [6]	90.67	38.36	1.228	-	1.57
	SC + precount [9]	95.12	37.14	0.687	-	1.62
	SC + parallel [6]	115.95	46.53	0.407	-	1.52
	Ours + serial	153.86	48.97	0.846	31.1	1.89
	Ours + precount	148.47	45.34	0.643	6.4	1.89
	Ours + parallel	167.66	43.34	0.287	29.5	1.89
$n = 8$	Non-SC	152.10	58.21	0.116	-	1.76
	SC + serial [6]	110.56	47.90	6.131	-	1.87
	SC + precount [9]	121.21	48.28	3.211	-	1.69
	SC + parallel [6]	211.77	64.77	1.093	-	1.89
	Ours + serial	222.65	60.09	2.275	62.9	1.86
	Ours + precount	206.97	56.11	1.758	45.2	1.80
	Ours + parallel	243.71	54.43	0.478	56.2	1.90

Table 4. Comparison of hardware area and energy consumed by our MUX-FSM-based SC model and the conventional models for processing the summation of 16 multiplications: $I_1 \times W + I_2 \times W + \ldots + I_{16} \times W$

	Scheme	Area	Power (μW)	Energy (pJ)	Reduction (%)	Clock (ns)
$n = 6$	Non-SC	97.93	38.45	0.077	-	1.89
	SC + serial [6]	90.67	38.36	1.228	-	1.57
	SC + precount [9]	95.12	37.14	0.687	-	1.62
	SC + parallel [6]	115.95	46.53	0.407	-	1.52
	Ours + serial	153.86	48.97	0.846	31.1	1.89
	Ours + precount	148.47	45.34	0.643	6.4	1.89
	Ours + parallel	167.66	43.34	0.287	29.5	1.89
$n = 8$	Non-SC	152.10	58.21	0.116	-	1.76
	SC + serial [6]	110.56	47.90	6.131	-	1.87
	SC + precount [9]	121.21	48.28	3.211	-	1.69
	SC + parallel [6]	211.77	64.77	1.093	-	1.89
	Ours + serial	222.65	60.09	2.275	62.9	1.86
	Ours + precount	206.97	56.11	1.758	45.2	1.80
	Ours + parallel	243.71	54.43	0.478	56.2	1.90

summation of 16 multiplications in which one input to every multiplication is *W*. Thus, not only our SC but also the existing MUX-FSM SCs are able to deploy just one FSM based controller to select the inputs of 16 MUXes in parallel. This leads to a significant area saving in comparison to the non-SC model. Since our model reduces the number of clock cycles 8 times more than that for single multiplication, the energy-saving by our model for the architecture processing 16 multiplications in parallel is more significant, reducing by about 1/2 for 6-bit multiplication and about 1/3 saving for 8-bit over that by MUX-FSM SC + serial [6]. Likewise, our SC models with pre-counting and bit-parallel processing show a tendency similar to the SC model with serial counting. One exception is our SC with bit-parallel in 6-bit multiplication, which demands a considerable area overhead.

V. CONCLUSION

This paper proposed a new stochastic computing algorithm and architecture to overcome the critical limitation of the prior MUX-FSM-based SCs, which was the infeasibility of catching up with the inference speed required for on-device neural networks. Precisely, we analyzed the bit counting pattern produced by MUX-FSM-based SCs and replaced the counting redundancy with inexpensive shift operations, resulting in reducing the length of the required bit sequence significantly. In addition, we analytically formulated and compared the number of computation cycles in processing multiplication for all MUX-FSM-based SCs, including our proposed ones. Through experiments, it was shown that our enhanced SC technique was able to shorten the average processing time by 44.1% over the conventional MUX-FSM-based SCs. We also showed that by deploying our architecture for processing 16 multiplications in parallel, the energy consumption was reduced by 1/2 to 1/3. Finally, we validated that the concept of split-shift in our proposed MUX-FSM-based SC model could be applied to any existing variant of MUX-FSM-based SC to make a synergy effect.

ACKNOWLEDGMENTS

This work was supported in part by Samsung Electronics Company Ltd. (No. IO201216-08205-01 and FOUNDRY-2020-10DD003F), in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MEST) (No. 2020-R1A4A4079177 and 2021-R1A2C2008864), in part by the Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by Korea government (MSIT) (No. 2021-0-00754, Software Systems for AI Semiconductor Design), and in part by the BK21 Four Program of the Education and Research Program for Future ICT Pioneers, Seoul National University. The EDA tool was supported by the IC Design Education.

REFERENCES

1. A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded Computing Systems*, vol.

12, no. 2S, pp. 1-19, 2013.
2. A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515-1531, 2018.
3. Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, and Y. Wang, "Structural design optimization for deep convolutional neural networks using stochastic computing," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lausanne, Switzerland, 2017, pp. 250-253.
4. W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "ACOUSTIC: accelerating convolutional neural networks through or-unipolar skipped stochastic computing," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2020, pp. 768-773.
5. M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2925-2938, 2019.
6. H. Sim and J. Lee, "Cost-effective stochastic MAC circuits for deep neural networks," *Neural Networks*, vol. 117, pp. 152-162, 2019.
7. R. Hojabr, K. Givaki, S. R. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "SkippyNN: an embedded stochastic-computing accelerator for convolutional neural networks," in *Proceedings of 2019 56th ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, NV, 2019, pp. 1-6.
8. H. Sim and J. Lee, "Log-quantized stochastic computing for memory and computation efficient DNNs," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, Tokyo, Japan, 2019, pp. 280-285.
9. E. Azari and S. Vrudhula, "ELSA: a throughput-optimized design of an LSTM accelerator for energy-constrained devices," *ACM Transactions on Embedded Computing Systems*, vol. 19, no. 1, pp. 1-21, 2020.
10. J. Kang and T. Kim, "Speeding up MUX-FSM based stochastic computing for on-device neural networks," in Proceedings of 2019 *Design, Automation and Test in Europe Conference (DATE)*, Grenoble, France, 2021.

Jongsung Kang

Jongsung Kang received the B.S. and Ph.D degrees in electrical and computer engineering, from Seoul National University, Seoul, South Korea, in 2014 and 2020, respectively. In 2020, he joined Samsung Electronics, System LSI Business, working as a System-on-Chip Design Engineer. His research interests include low power computation for mobile neural network processing units.

Taewhan Kim

Taewhan Kim received the B.S. degree in computer science and statistics and the M.S. degree in computer science from Seoul National University, Seoul, South Korea, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, in 1993. He is currently a Professor with the School of Electrical Engineering and Computer Science, Seoul National University. He has published over 300 technical papers in international journals and conferences. His current research interests include computer-aided design of integrated circuits ranging from the architectural synthesis through physical designs, specifically focusing on logic and physical synthesis and automatic cell layout generation. He is currently serving on Associate Editors of Integration-VLSI Journal.