Speeding up MUX-FSM based Stochastic Computing for On-device Neural Networks

Jongsung Kang, Taewhan Kim
Dept. of Electrical and Computer Engineering
Seoul National University, Korea
{jskang, tkim}@snucad.snu.ac.kr

***Abstract*—We propose an acceleration technique for processing multiplication operations using stochastic computing (SC) in on-device neural networks. Recently, MUX-FSM based SCs, which employ a MUX controlled by an FSM to generate a bit stream for a multiplication operation, considerably reduces the processing time of MAC operations over the traditional stochastic number generator based SC. Nevertheless, the existing MUX-FSM based SCs still do not meet the multiplication processing time required for a 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 to the problem of speeding 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 shortening the length of the required bit sequence significantly, together with analytically formulating the amount of computation cycles. Through experiments, it is shown that our enhanced SC technique is able to reduce the processing time by 44.1% on average over the conventional MUX-FSM based SCs.**

***Index Terms*—Convolutional neural networks, Stochastic computing**

I. Introduction

On-device neural networks accelerator is gained popularity from no network latency in processing data and privacy. Its processing cost should be minimized further for limited power budget, logic area, and performance demands. Among diverse methods for speeding up the processing, this work belongs to introducing alternative computing methods for speeding up the processing time, especially the stochastic computing.

Stochastic computing (SC) [2] is a collection of techniques that represent continuous values by streams of random bits. Complex computations can then be computed by simple bit-wise operations on the streams. Fig. 1 illustrates an architecture for SC based multiplication, in which the two stochastic number generators (SNG) generates bit streams, which are close approximations of two inputs I and W. Then, bit-wise AND operations for the bit streams produce a bit stream corresponding to the value of $I \times W$, from which its binary representation is obtained by counting 1-bits.

Though the hardware logic for bit-wise operations in SC is simple, it entails a couple of limitations, which are (1) *a considerable logic area for SNGs* and (2) *a long bit stream* to maintain no accuracy loss in operation as well as representation. A bit stream with length of up to 2^n should be prepared to be probabilistically exact to represent a value that uses n bits in binary representation.

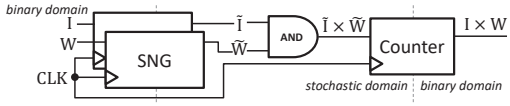


Fig. 1. Illustration of the stochastic computing (SC) for a multiplication.

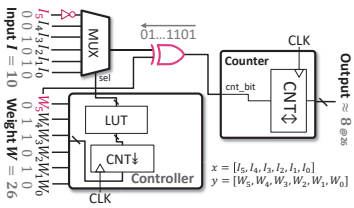


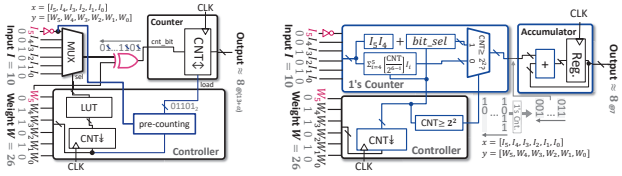
Fig. 2. Structure of MUX-FSM based SC [6]. Red colored gates (INV, XOR) account for the support of signed operations.

To overcome the limitations, recently a number of deterministic SC structures have been proposed. One noticeable structure, called SC-DNN [6], is shown in Fig. 2. SC-DNN replaces the two expensive SNGs in Fig. 1 with a MUX and a simple counter-based FSM, in which the MUX inputs are the bit values of activation input I and the bit stream is formed by an iterative selection of MUX inputs that is controlled by the FSM and the length is weight W. However, since this SC structure still requires the counter in FSM to operate 2^n clock cycles at the worst case (i.e., the W value close to the largest magnitude), saving the multiplication processing time is limited.

There are architectures that enhance the multiplication processing of the structure in Fig. 2. Fig. 3(a) [9] called *MUX-FSM based SC with pre-count* exploits that the most significant bit (MSB) of input I is selected every two clock cycles in a row in Fig. 2. It sets the initial value of its counter to the total count of the MSB in the sequence and then starts to count up the rest, thereby reducing the processing time roughly by half.

On the other hand, the structure in Fig. 3(b) [6] 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 the cost of additional 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 base SC structures in [6], [9] is that total counting time in a multiplication ($I \times W$) is tightly bounded by the absolute value of W. This work proposes an alternative method of lowering down this bound. Precisely, (1) we introduce a novel concept called *split-shift* and apply it to W on all



(a) Structure of MUX-FSM based SC with pre-count [9]. (b) Structure of MUX-FSM based SC with bit-parallel processing [6].

Fig. 3. Enhanced structures of MUX-FSM based SC.

MUX-FSM based SC structures. (2) We analytically formulate the amount 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 that by the conventional MUX-FSM based SC in Fig. 2, and its two enhanced SCs in Figs. 3(a) and 3(b), respectively.

II. Preliminary

This section explains how the existing MUX-FSM based SCs works, which is essential to understand our work in Sec. III. For brevity of presentation, we assume the multiplication inputs are all unsigned numbers.

MUX-FSM based SC: The upper part in Fig. 4 shows how MUX-FSM based SC in [6] calculate $I \times W$ where $I = [I_5 I_4 I_3 I_2 I_1 I_0] = 001010_2 (= 10)$ and $W = 011010_2 (= 26)$. To calculate $I \times W$ by SC, it assigns each binary bit of I 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], shown in the middle of Fig. 4. In addition, the length of S exactly equals the value of W. Thus, the bit stream corresponding to the S is [1 0 1 1 ... 1 0]. It then counts the number of 1-bit values in the bit stream, one cycle at a time, taking total of 26 (= W) clock cycles.

The rationale of using the fixed index sequence S, shown after 'I_i' in Fig. 4, is the following: For n-bit W and I, $I \times W$ can be approximated by evaluating *Eq.1*, or by counting the value of each I_{n-j} $\lceil W * 2^{-j} \rceil$ times.

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

It means that the number of I_j, $_{j=n-1,\cdots,1}$ in S is twice more than that of I_{j-1}. Thus, the W value can be distributed into $|W|$ bits (i.e. index sequence) of $I_{n-1}, \cdots, I_0$ such that total sum of the bit values is the value in *Eq.1*. (The details on the construction of index sequence can be found in [6].) Thus, the number of clock cycles in the worst case is bounded by 2^n.

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

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

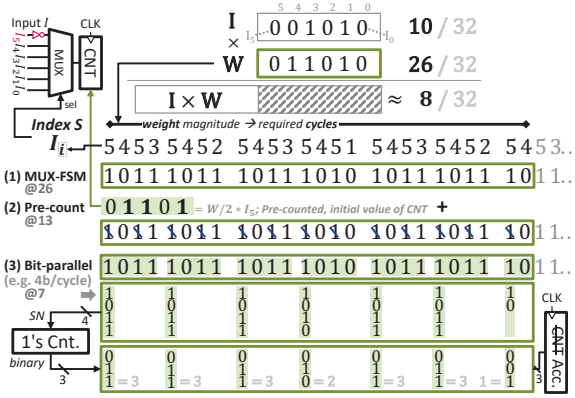


Fig. 4. Illustration of the process of multiplication by the existing (1) MUX-FSM based SC, (2) MUX-FSM based SC with pre-count, and (3) MUX-FSM based SC with bit-parallel processing.

(in a single cycle), thereby reducing the clock cycles by $\lceil W/r \rceil$, equivalently up to $\lceil 2^n/r \rceil$ (e.g., for $r = 4$ in Fig. 4(3), $\lceil 26/4 \rceil = 7$ cycles). The additional hardware is a counter with accumulation, 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

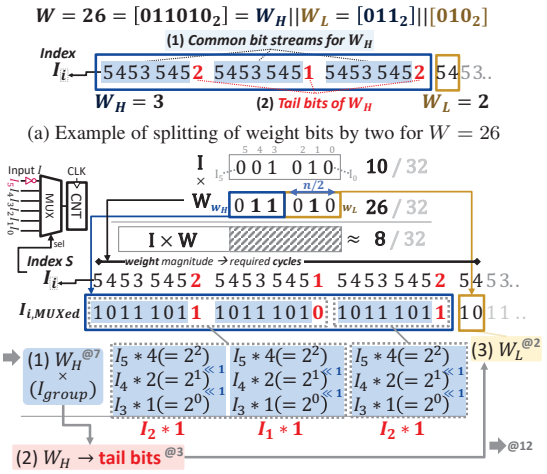
A. New Algorithm for Stochastic Computing

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 by exploiting the abundance of common bit sub-streams in S.

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 (CIS) [5453 545]. We call this 7-length bit sub-streams in S *common bit streams (CBS) for* W_H and the each last bit right after the CBSs 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. CBSs for W_H, tail bits for W_H, and bit stream for W_L are shown in blue, red, and yellow boxes.

Our algorithm for MUX-FSM based SC, shown in Fig. 6(a), has three steps: counting (1) *CBSs*, (2) *tail bits* and (3) *the rest*.

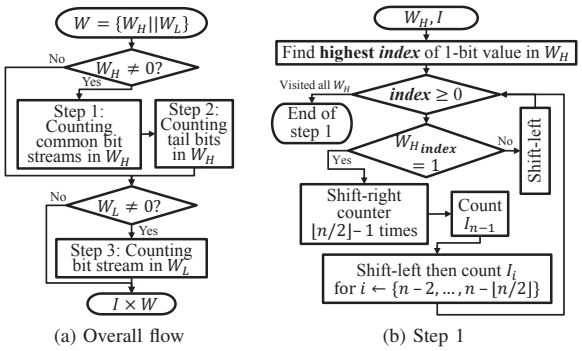
Step 1 (**Counting CBSs for** W_H): Since the CBSs have already been known when the multiplication input bit-width n is given, we will count the number of 1-bit values very efficiently. For example, if $n = 6$, [5453 545] is the CIS, for which we have to count the value of I_5 4 times, I_4 2 times, and I_3 once. Consequently, it suffices to count up the value of I_5, followed by simultaneously shifting the counter and counting up I_4, and finally simultaneously shifting the counter and counting up I_3. Thus, the counting process for one CBS takes 3 clock cycles. Then, the count value obtained can be doubled by one shift operation. Since there are three CBSs in the example for $W_H = 011_2$, the total cycles is 7 = (3 + 1 (shift) + 3 (processing one more CBS)). In comparison with the conventional MUX-FSM



(a) Example of splitting of weight bits by two for $W = 26$

(b) Grouping input bit stream regarding weight bits, and computations.

Fig. 5. Splitting weight, grouping index sequence and computation steps.



(a) Overall flow (b) Step 1

Fig. 6. The flow of our proposed algorithm.

SC in [6], which needs 21 (= 7 × 3) clock cycles, we are able to reduce the number of clock cycles. The blue shaded part in Fig. 5(b) illustrates shifting and counting process that how the three common bit streams for W_H are counted.

Step 2 (Counting tail bits for W_H) We can store the bit index values of the tail bits, which are already known, in a lookup table (LUT), and fetch the index according to the position of CBSs. For the example in Fig. 5(b), the red numbers 2, 1, and 2 (I_2, I_1, and I_2) are the tail bit indices of the 8-bit groups in S, respectively. Thus, it needs 3 (= W_H) clock cycles.

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 W_L, for the Fig. 5(b) example, it requires 2 (= W_L) clock cycles to count up all bits.

Our algorithm can be extended to support the MUX-FSM based SC with pre-count [9] or with bit-parallel processing [6] by simple treatments. Table I summarizes the analytic analysis of time complexity (i.e. the number of clock cycles) used by the conventional three MUX-FSM based SCs and our algorithm. The curves in Fig. 7 shows the visualization of the changes of clock cycles required as the bit-width of input W varies.

B. The Supporting Hardware Architecture

Fig. 8 shows the hardware architecture that supports our algorithm, which upgrades the conventional architecture in Fig. 2. Our architecture has three features: (1) *a logic circuit for one bit shift operation*, (2) *a master FSM* that controls our three-step algorithm, and (3) *three slave FSMs* carry out the counting process for the three steps sequentially, one by one.

To save the clock cycles in bit stream counting, it is necessary to install a logic block that can perform one bit shift operation, which helps relieve the (redundant) counting burden on the 1-bit counter block in the blue box in Fig. 8.

The master FSM, shown as the green box in Fig. 8, controls the three slave FSMs, to sequentially triggering the three steps. It can skip steps 1 and 2 (or step 3) if W_H (or W_L) is 0. The slave FSM for step 1 (blue) performs counting W_H CBSs with utilizing shift operation, illustrated in Fig. 6(b). The role of the slave FSMs for steps 2 and 3 (red, yellow) are basically identical to that of the conventional FSM-MUX based SC, but counts W_H tail bits (step 2) or W_L bits (step 3).

TABLE I
THE NUMBER OF COMPUTATION CYCLES OF OUR PROPOSED AND REFERENCE SCHEMES FOR N-BIT $I \times W$.

Scheme	Cycles*
SC + serial [6]	W
SC + pre-count [9]†	$\lceil W/2 \rceil$
SC + parallel [6]	$\lceil W/r \rceil$
Ours + serial	$pcnt(W_H) \cdot (2 \cdot \lceil n/2 \rceil - 1) + W_H + W_L$
Ours + pre-count†	$pcnt(W_H) \cdot (2 \cdot (\lceil n/2 \rceil - 1) - 1) + W_H + \lceil W_L/2 \rceil$
Ours + parallel	$\lceil log_2(W_H + 1) \rceil + \lceil W_H/r \rceil + \lceil W_L/r \rceil$

*$W = W_H * 2^{n/2} + W_L$, $pcnt$ = Hamming weight †No pre-count cycles

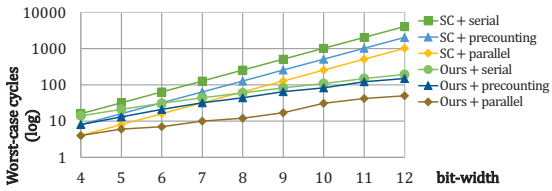


Fig. 7. The changes of the worst case clock cycles as the operand bit-width changes. (Bit-parallel processing level r is set to 4.)

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.1* by using the conventional model of MUX-FSM based SC (SC + serial), one with pre-count (SC + pre-count), and

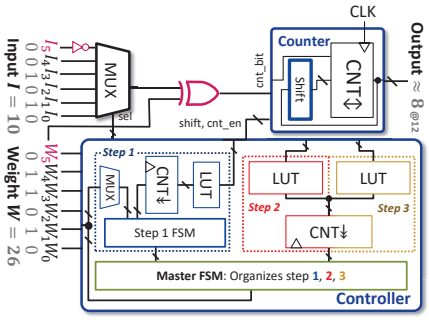


Fig. 8. The architecture supporting our MUX-FSM SC.

TABLE II
THE 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 8.

Model	n	#cycles	reduction	n	#cycles	reduction
SC + serial [6]	6	16.00	-	8	64.00	-
SC + pre-count [9]		9.25	-		33.25	-
SC + parallel [6]		4.38	-		8.44	-
Ours + serial		8.64	46.0%		18.93	70.4%
Ours + pre-count		7.09	23.3%		15.67	52.9%
Ours + parallel		3.31	24.3%		4.39	47.9%

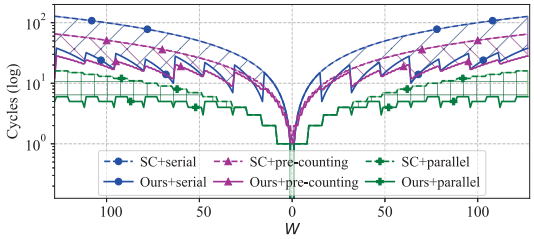


Fig. 9. The changes of the number of clock cycles used by our and existing SC models as weight W changes in multiplication of bit-width n = 8.

one with bit-parallel (SC + parallel) as well as our proposed model of MUX-FSM based SC. All the models exhibit the same approximation accuracy since every model exactly computes the formulation in *Eq.1*. Then, we implement our model and the conventional three models and synthesize them into RTL design by using Synopsys Design Compiler with industrial 28nm cell library, setting the clock frequency to 500MHz. Cell area and (vectorless) estimated power consumption are extracted from the synthesized designs by using Design Compiler.

B. Performance Comparison

Table II shows the comparison of the number of average clock cycles used by the conventional MUX-FSM based SC models, 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 multiplication of n = 8. It clearly reveals that the bigger the weight magnitude is, the more the number of clock cycles to be required is, but the gap by our models is much shorter than that of the conventional models.

C. Hardware Area and Energy Comparison

Table III compares the hardware area, power and energy consumed by the conventional MUX-FSM based SCs, non-SC, and ours for computing the 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 with the non-SC model. The energy saving by our model is about 1/2 for 6-bit multiplication and about 1/3 for 8-bit over that by MUX-FSM SC + serial [6]. Likewise, our SC models with pre-count and bit-parallel processing show the tendency similar to the SC model with serial counting. One

TABLE III
COMPARISON OF AREA AND ENERGY CONSUMED BY OUR AND CONVENTIONAL MUX-FSM BASED SC MODELS FOR PROCESSING THE SUMMATION OF 16 MULTIPLICATIONS: $I1 \times W + I2 \times W + \cdots + I16 \times W$.

n	Model	Area	Power†	Energy*	red.	Min. clk_pd‡
6	Non-SC	1,551.54	600.91	1.202	-	1.90
	SC + serial [6]	603.25	272.75	8.728	-	1.88
	SC + pre-count [9]	616.36	261.50	4.838	-	1.71
	SC + parallel [6]	775.71	274.27	2.400	-	1.84
	Ours + serial	812.10	277.13	4.789	45.1%	1.90
	Ours + pre-count	724.81	259.51	3.682	23.9%	1.88
	Ours + parallel	1,117.47	273.93	1.815	24.4%	1.89
8	Non-SC	2,661.05	935.69	1.871	-	1.89
	SC + serial [6]	829.65	365.11	46.734	-	1.89
	SC + pre-count [9]	868.96	369.93	24.601	-	1.86
	SC + parallel [6]	1,577.51	391.78	6.611	-	1.90
	Ours + serial	1,318.94	387.04	14.653	68.6%	1.90
	Ours + pre-count	1,177.60	368.75	11.555	53.0%	1.91
	Ours + parallel	2,113.49	416.44	3.660	44.6%	1.91

†uW, *pJ, ‡ns

exception is our SC with bit-parallel in 6-bit multiplication, which demands relatively a considerable area overhead.

V. CONCLUSION

In this paper, we analyzed the bit counting pattern produced by MUX-FSM based SCs and replaced the counting redundancy by inexpensive shift operations, resulting in reducing the length of the required bit sequence significantly. In addition, we analytically formulated and compared the amount of computation cycles in processing multiplication for 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 $\sim$ 1/3.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.2020R1A4A4079177).

REFERENCES

[1] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM TECS*, Vol. 12, No. 2, pp. 1–19, 2013.
[2] A. Alaghi, et al., "The Promise and Challenge of Stochastic Computing," *IEEE TCAD*, Vol. 37, No. 8, pp. 1515–1531, 2018.
[3] Z. Li, et al., "Structural design optimization for deep convolutional neural networks using stochastic computing," *DATE*, pp. 250–253, 2017.
[4] W. Romaszkan, et al., "ACOUSTIC: Accelerating Convolutional Neural Networks through Or-Unipolar Skipped Stochastic Computing," *DATE*, 2020.
[5] M. H. Najafi, et al., "Performing Stochastic Computation Deterministically," *IEEE TVLSI*, 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 et al., "SkippyNN: An Embedded Stochastic-Computing Accelerator for Convolutional Neural Networks," *DAC*, pp. 1–6, 2019.
[8] H. Sim and J. Lee, "Log-quantized stochastic computing for memory and computation efficient DNNs," *ASP-DAC*, pp. 280–285, 2019.
[9] E. Azari and S. Vrudhula, "ELSA: A Throughput-Optimized Design of an LSTM Accelerator for Energy-Constrained Devices," *ACM TECS*, Vol. 19, No. 1, pp. 1–21, 2020.