Liu, S., Tang, X., Niknia, F., Reviriego, P., Liu, W., Louri, A. & Lombardi, F. (2021). Stochastic Dividers for Low Latency Neural Networks. *IEEE Transactions on Circuits and Systems I: Regular Papers, 68*(10), 4102–4115.

DOI: 10.1109/tcsi.2021.3103926

Stochastic Dividers for Low Latency Neural Networks

Shanshan Liu, *Member, IEEE*, Xiaochen Tang, *Member, IEEE*, Farzad Niknia, *Student Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, Weiqiang Liu, *Senior Member, IEEE*, Ahmed Louri, *Fellow, IEEE*, and Fabrizio Lombardi, *Life Fellow, IEEE*

***Abstract*— Due to the low complexity in arithmetic unit design, stochastic computing (SC) has attracted considerable interest to implement Artificial Neural Networks (ANNs) for resources-limited applications, because ANNs must usually perform a large number of arithmetic operations. To attain a high computation accuracy in an SC-based ANN, extended stochastic logic is utilized together with standard SC units and thus, a stochastic divider is required to perform the conversion between these logic representations. However, the conventional divider incurs in a large computation latency, so limits an SC implementation for ANNs used in applications needing high performance. Therefore, there is a need to design fast stochastic dividers for SC-based ANNs. Recent works (e.g., a binary searching and triple modular redundancy (BS-TMR) based stochastic divider) are targeting a reduction in computation latency, while keeping the same accuracy compared with the traditional design. However, this divider still requires N iterations to deal with 2^N-bit stochastic sequences, and thus the latency increases in proportion to the sequence length. In this paper, a decimal searching and TMR (DS-TMR) based stochastic divider is initially proposed to further reduce the computation latency; it only requires two iterations to calculate the quotient, so regardless of the sequence length. Moreover, a trade-off design between accuracy and hardware is also presented. An SC-based Multi-Layer Perceptron (MLP) is then considered to show the effectiveness of the proposed dividers over current designs. Results show that when utilizing the proposed dividers, the MLP achieves the lowest computation latency while keeping the same classification accuracy; although incurring in an area increase, the overhead due to the proposed dividers is low over the entire MLP. When using as combined metric for both hardware design and computation complexity the product of the implementation area, latency, power and number of clock cycles, the proposed designs are also shown to be superior to the SC-based MLPs (at the same level of accuracy) employing other dividers found in the technical literature as well as the commonly used 32-bit floating point implementation.**

***Index Terms*— Divider, stochastic computing, decimal searching, artificial neural network.**

Manuscript received June 17, 2021; revised August 5, 2021; accepted August 7, 2021. The work of Shanshan Liu, Farzad Niknia, and Fabrizio Lombardi was supported by the NSF Grant CCF-1953961 and Grant 1812467. The work of Pedro Reviriego was supported in part by the Spanish Ministry of Science and Innovation under project ACHILLES (Grant PID2019-104207RB-I00) and the Go2Edge Network (Grant RED2018-102585-T), and in part by the Madrid Community Research Agency under Grant TAPIR-CM P2018/TCS-4496. The work of Weiqiang Liu was supported by the NSFC under Grant 62022041 and Grant 61871216. The work of Ahmed Louri was supported by the NSF Grant CCF-1812495 and Grant 1953980. This article was recommended by Associate Editor C. H. Chang. *(Shanshan Liu and Xiaochen Tang contributed equally to this work.) (Corresponding author: Shanshan Liu.)*

Shanshan Liu, Farzad Niknia, and Fabrizio Lombardi are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: ssliu@coe.neu.edu).

Xiaochen Tang is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA.

Pedro Reviriego is with the Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Leganés, 28911 Madrid, Spain.

Weiqiang Liu is with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China.

Ahmed Louri is with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3103926.

Digital Object Identifier 10.1109/TCSI.2021.3103926

I. Introduction

ARTIFICIAL Neural Networks (ANNs) are some of the most widely used machine learning hardware systems to perform for example classification tasks over a wide range of applications [1], [2]. Among ANNs used in todays' machine learning, the Multi-Layer Perceptron (MLP) is the simplest but still powerful type [3]. By increasing the dimension of the features, the MLP can perform classification for nonlinear separable data with excellent accuracy. For example, when an ANN is applied to Internet of Things (IoT) applications, its training process can be run in a processor, while the inference process is implemented in dedicated hardware [5]–[8]. Since ANNs usually deal with large volume of data, the required hardware may not be acceptable in resource-limited platforms (even though the implementation of some arithmetic operations can be improved [9], [10]); thus in this case, a stochastic computing (SC) design is attractive for implementing the ANN [11]–[15], because SC arithmetic units have a significantly lower complexity (e.g., to perform multiplication, only one *AND* gate is needed for the unipolar SC design) [16]–[18]. This feature is also desirable when the ANN is employed in some safety-critical applications, because an SC design also provides a high error-tolerance capability, so generating a reliable outcome in the presence of errors during the computation process. Since the SC units target the hardware implementation, in addition to the conventional ANNs, they can also be utilized/integrated to implement some low-cost or quantized ANNs that reduce the network complexity (e.g., reducing the number of weights, or the number of bits for representing data) by introducing a small accuracy degradation [19], to further reduce the hardware requirements as well as providing error-tolerance.

However, an SC implementation has disadvantages in terms of computation accuracy and latency. To improve accuracy,

the input stochastic sequences of the SC units are usually either uncorrelated, or loosely correlated to each other [20]. Moreover, since SC utilizes the probability of each bit being "1" in the stochastic sequence to represent a real value, the result is bounded either in [0, 1] for unipolar computation and [−1, 1] for bipolar computation; this leads to an accuracy loss because the real value is computed over the entire real value field. To overcome this issue, the so-called extended stochastic logic (ESL) consisting of two stochastic sequences has been presented in [8]; by taking the quotient of two sequences, the unipolar (bipolar) range can be extended to $[0, 2^N)$ $((-2^{N-1}, 2^{N-1}))$, where 2^N is the length of the stochastic sequence. However, the length of a stochastic sequence may need to be increased to provide an accurate mapping from the real value; therefore, a trade-off can be made between computation latency and accuracy, as most SC units require 2^N clock cycles.

A stochastic divider is utilized to perform the conversion between the standard (traditional) SC units and the ESL; the divider is the most complex SC arithmetic unit and requires a rather long computation process (the required number of clock cycles must be significantly larger than 2^N) to reach a stable state and provide a correct result, because a negative feedback exists in the traditional design (the detailed structure will be reviewed in the next section). Therefore, arithmetic SC design has focused on improving the computation latency of the divider. For example, a binary searching method (also referred to as stepped velocity) has been utilized to calculate the quotient by progressive precision [17]; it reduces the required number of clock cycles, but introduces an accuracy loss. It has been improved by utilizing a triple modular redundancy (TMR) structure [11] to further reduce the latency and keep the accuracy the same as the conventional divider. However, existing dividers based on binary searching still require N iterations to complete the computation process, so incurring in a latency proportional to the length of the stochastic sequences (i.e., 2^N). An alternative solution is to design several parallel sequences as input of the divider [21], but this is only suitable for specific sequences (e.g., low-discrepancy sequences like Sobol [21]), rather than the most widely used linear feedback shift register (LFSR)-based sequences. Since Sobol-based sequences cannot provide an accurate computation for some SC units (e.g., the Finite State Machine (FSM)-based units) required to implement an ANN, such divider is not suitable for SC-based ANNs (in which the FSM-based activation functions must be utilized). The correlation property of input sequences can also be utilized to reduce area and power of the divider with a highly accurate outcome [22], [23]; however, such divider requires correlated inputs which is not the case for other units of an SC-based ANN implementation, and the area overhead is also not of primary importance because the divider accounts for a small fraction in the entire system (even though it significantly affects the length of the critical computation path). Therefore, there is a need for a low latency stochastic divider for implementing SC-based ANNs.

This paper focuses on the reduction of the latency of SC dividers using a novel decimal searching algorithm, and thus it benefits SC-based ANNs in terms of hardware metrics like computation latency. This novel divider only requires only two iterations regardless of the length of the stochastic sequences, so it significantly reduces the computation latency compared with existing dividers while at architectural level, it permits a constant output flow for pipelining; moreover, it provides the same accuracy of SC-based ANNs as existing dividers. A trade-off divider design between different figures of merit is also presented. It is shown that this divider accounts for a small hardware complexity in an SC-based ANN (at the same accuracy of SC schemes); therefore, a small increase in the hardware area of the proposed dividers results in significant reductions in latency and power. When the combined metric of the product of area, latency, power and number of clock cycles (so by considering both hardware design and computational complexity) is used in the evaluation, the SC-based ANNs utilizing the proposed dividers have the best performance among those with all dividers found in the technical literature.

The rest of this paper is organized as follows. In Section II, the basic SC units are reviewed; the existing stochastic dividers are also presented. Section III initially presents the proposed decimal searching algorithm; it is then employed to design the first proposed divider. The trade-off design between accuracy and hardware for the divider is also discussed in this section. These proposed schemes are evaluated and compared with existing stochastic dividers in Section IV. Section V considers the Multi-Layer Perceptron as an application to analyze the effectiveness of the proposed divider for implementing an SC-based ANN. Finally, the paper ends in Section VI with the conclusion.

II. Preliminaries

A. Stochastic Computing

Stochastic sequences are represented by bit-streams of logic "1" and "0"; arithmetic operations in stochastic computing (SC) are commonly performed in a unipolar range of [0, 1] or a bipolar range of [−1, 1]. Define p as the probability of each bit being "1" in the stochastic sequence x; the numerical value for x is equal to p ($2 \cdot p$-1) in unipolar (bipolar) representation. Since in this paper, SC is studied to implement Artificial Neural Networks (ANNs) in which both negative and positive values are computed, the bipolar representation will be considered next.

1) Stochastic Number Generator: The probability p (corresponding to a numerical value) is encoded to a stochastic sequence by utilizing a stochastic number generator (SNG); the SNG is shown in Figure 1. In an SNG, pseudorandom numbers from 0 to 2^N-1 are generated by the random number generator (RNG) first and then compared with the N-bit binary representation of $p \cdot 2^N$; a "1" is generated if the pseudorandom number is larger, and "0" otherwise. After a period of 2^N, the stochastic sequence for p is obtained as per the comparison results; the positions of "1" and "0" are usually considered to be uniformly distributed. The RNG is usually implemented by a linear feedback shift register (LFSR) [17]. Recent works have shown that low-discrepancy sequences (like Sobol sequences) provide a better computation accuracy for some combinational and integrator-based SC units [21]; however, they have poor performance for some SC units based on a Finite State Machine (e.g., some SC activation functions used in an ANN).

2) Probability Estimator: A probability estimator (PE, also referred to as the probability to digital converter) is utilized

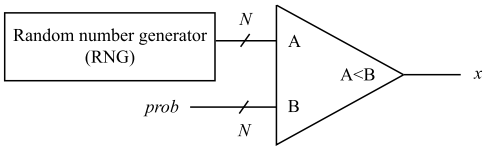

Fig. 1. Stochastic number generator: *prob* is the N-bit binary representation of $p(x)\cdot 2^N$; x is the stochastic bit.

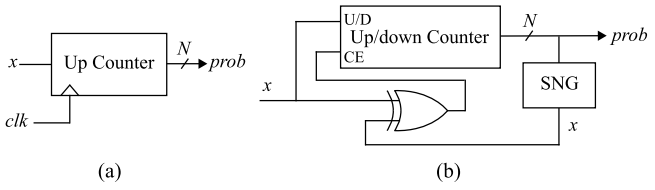

Fig. 2. Probability estimator: (a) simple PE; (b) PE with negative feedback loop (CE (U/D) is the enable signal (up/down control signal) of the Up/Down Counter, i.e., *prob* increases by 1 when CE = 1 and C/D = 1 and decreases by 1 when CE = 1 and U/D = 0).

to convert the stochastic sequences back to the probabilities, and thus the digital values. A PE is simply configured as shown in Figure 2 (a) [18]; it includes a counter to count the number of "1" in the stochastic sequence (i.e., for the N-bit binary representation of $p\cdot 2^N$). There are also some more complex structures for designing the PE; for example an SNG is introduced in addition to the counter and a negative feedback loop is present [17] (Figure 2 (b)); this PE generates an output for each input bit, but it requires a large number of clock cycles to reach a stable state and generate the final result.

3) Extended Stochastic Logic: Since real numbers are encoded to probabilities to implement SC and arithmetic operations are performed within a limited range, a computation accuracy degradation is likely introduced; this is an inherent drawback of SC. However, this problem can be resolved by employing the extended stochastic logic (ESL) [8]; two stochastic sequences are utilized in ESL and the quotient of their probabilities represents the real number. Therefore, the computation range can be extended from $[-1, 1]$ to $(-2^{N-1}, 2^{N-1})$. For example, an ESL sequence x with a value of 2 can be represented by $p(x) = \frac{P(x_1)}{P(x_2)} = \frac{0.6}{0.3} = 2$. When ESL units are utilized, the size of the circuits is nearly doubled compared to the standard SC units; for example, the SC multiplier and adder in both versions are illustrated in Figures 3 and 4 respectively. In standard SC, an *XNOR* gate performs the multiplication operation in the bipolar representation (Figure 3 (a)); when ESL is utilized, the product result $p(z)$ of two stochastic sequences x and y with the corresponding probabilities $p(x)$ and $p(y)$ must be calculated by Eq. (1), i.e., two *XNOR* gates are required (Figure 3 (b)).

$$p(x)\cdot p(y) = \frac{P(x_1)}{P(x_2)}\cdot\frac{P(y_1)}{P(y_2)} = \frac{P(z_1)}{P(z_2)} = p(z) \qquad (1)$$

Different from the standard stochastic multiplication between two stochastic sequences that can be performed in the interval of $[-1, 1]$, addition results may exceed the range and thus must be scaled. A standard stochastic adder can be built as illustrated in Figure 4 (a). In the ESL version, the sum of x and y is obtained by Eq. (2), and the structure of the adder

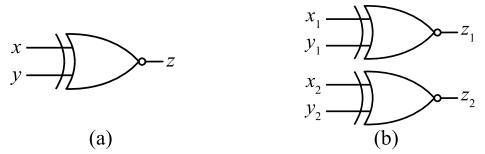

Fig. 3. A bipolar stochastic multiplier: (a) standard version, (b) ESL version.

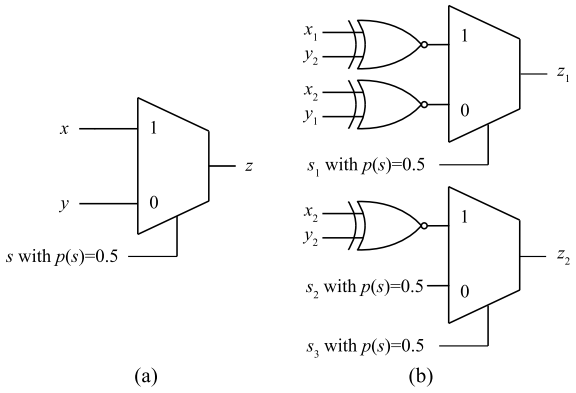

Fig. 4. A bipolar stochastic adder: (a) standard version, (b) ESL version.

is illustrated in Figure 4 (b).

$$\begin{aligned} p(x)+p(y) &= \frac{P(x_1)}{P(x_2)} + \frac{P(y_1)}{P(y_2)} \\ &= \frac{P(x_1)\cdot P(y_2) + P(x_2)\cdot P(y_1)}{P(x_2)\cdot P(y_2) + 0} \\ &= \frac{P(z_1)}{P(z_2)} = p(z) \end{aligned} \qquad (2)$$

B. Stochastic Dividers

The division is the most challenging operation among basic SC arithmetic operations; it is needed to perform the conversion between the standard SC and the ESL results. The conventional stochastic divider and existing binary searching-based stochastic dividers are reviewed next.

1) Conventional Stochastic Divider: Figure 5 illustrates a conventional bipolar divider. The feedback is constructed as per the comparison between the values of $p(x)\cdot p(y)$ and $p(x)^2\cdot\frac{p(y)}{p(x)}$; so, there are three multipliers (i.e., *XNOR* gates) in the divider circuit. The probability of the counter is initially set to 0 (in the bipolar representation), and the quotient is calculated once the feedback is stable. However, using existing divider designs the convergence process always requires a significant number of stochastic bits. Especially when the inputs have a small probability, the comparison result in the feedback is the same for many bits, so the counter always keeps its current value (i.e., not increasing/decreasing), making the convergence slow. Therefore, a stochastic divider based on binary searching has been designed in [17] to improve the convergence process; this design has been improved in terms of accuracy by introducing a triple modular redundancy (TMR) structure [11]. These dividers based on binary searching will be discussed next.

2) Binary Searching-Based Stochastic Dividers: In the binary searching-based divider, the initial probability in the counter is also set to 0 in the bipolar presentation (i.e., "1" on the most significant bit and "0" on the other bits); its increase/decrease step starts with 2^{N-2} (N is the width of the

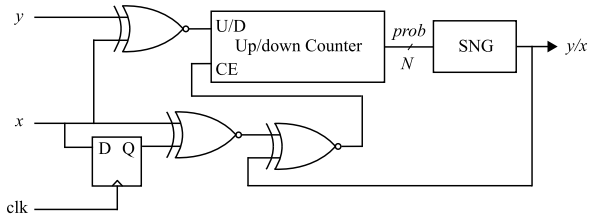

Fig. 5. A conventional bipolar stochastic divider.

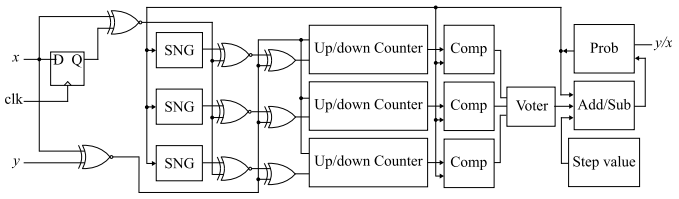

Fig. 6. The binary search and TMR-based bipolar stochastic divider of [11].

counter) and then scales by a factor of two at each iteration, until it reaches 1. For example, if the counter has 8 bits, the probability starts with "10000000" and increases/decreases with a step of "01000000" at the second search iteration, "00100000" at the third iteration, and so on. This algorithm significantly increases the convergence speed, because the possible range of the quotient is determined on a half-by-half fashion, rather than one-by-one.

The binary searching-based divider has been improved in [11] to further decrease the computation latency. The divider of [11] is shown in Figure 6; its operation includes two phases: the computation phase and the stabilization phase. In the computation phase, the quotient is first estimated by utilizing the binary search algorithm, but with a shortened number of stochastic bits in each searching step. Since the small number of stochastic bits introduce fluctuation errors, a triple modular redundancy (TMR) structure is utilized for generating three different and independent stochastic sequences with the same probability for each input; then the majority voting outcome is considered as the correct result to mask fluctuation errors. The quotient estimated in the computation phase is then refined in the stabilization phase; the divider enables two redundant blocks of the TMR structure and works in a conventional fashion to further calculate the quotient. As per the results of [11], the binary searching-based divider with TMR can significantly improve the convergence speed without accuracy degradation; this is made possible by selecting an appropriate number of stochastic bits for each phase. However, it still takes N iterations to obtain the quotient in the computation phase, so it introduces a computation latency proportional to the length of SC sequences.

III. Proposed Stochastic Divider Design

A. Decimal Searching Algorithm

The method to calculate the quotient of two stochastic sequences with progressive precision has been employed in the divider of [11] based on binary searching and TMR (BS-TMR); as discussed previously, this divider refines the precision of the calculated result by comparing it with the center value of the possible range in each iteration (i.e., setting 1 for each bit of the N-bit binary sequence as base value for comparison, from the most significant to the lowest

Algorithm 1 Decimal Searching

1: Split the entire decimal value range into $M+1$ intervals;
2: Set the largest value of the 1st to Mth intervals as the initial base value for each of M blocks;
3: **for** $i = 1$ **to** t–1 (where t is the number of search iterations)
4: Compare the input with the base value in each block to locate the correct interval;
5: Further split the located interval into $M+1$ refined intervals (i.e., with one more decimal bit);
6: Update the base value of each block as per the refined intervals;
7: **end**
8: Compare the input with the base value in each block to identify the correct interval;
9: Output the base value of the located block as the final result;

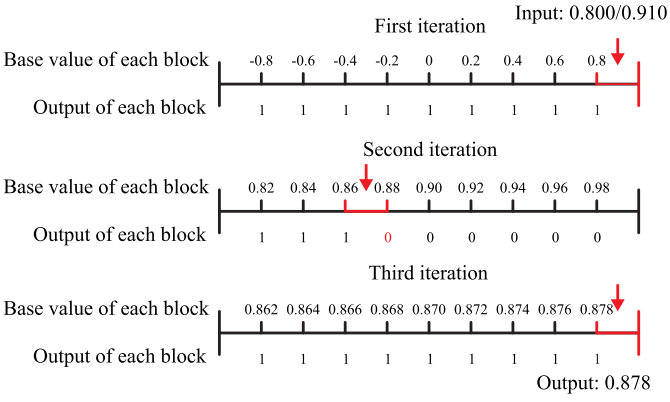

Fig. 7. Illustration of utilizing the decimal searching-based algorithm to find the quotient of 0.800/0.910.

significant bits). Although the BS-TMR-based divider improves computation, it still requires N iterations in the computation phase, incurring in a significant computation latency when N is large. To further reduce the computation latency, a faster searching algorithm, referred to as the decimal searching algorithm, is presented and employed to design new stochastic dividers (that will be discussed in the next subsections).

The so-called "decimal searching" refers to the process for determining the decimal version of a stochastic value with one decimal bit per search iteration. As given by Algorithm 1, M identical blocks operate in parallel to determine the possible value for a decimal bit, and then reaching the correct result after several iterations. The number of blocks M (with different base values) and the required number of iterations can be designed by considering the data resolution. For example, if N (the number of bits in the binary representation for a bipolar stochastic value) is equal to 10, the data resolution should be $\frac{1}{2^{10}} \cdot 2 \approx 0.002$. In this case, M is given by nine, so that the bipolar range of $[-1, +1]$ is divided in ten intervals. Thus, a resolution of 0.2 is achieved in the first iteration, 0.02 in the second iteration, and 0.002 in the third iteration. This is explained in detail by considering the example shown in Figure 7; in Figure 7 with $N = 10$ and $M = 9$, the base value for each block is initially given by $\{-0.8, -0.6, -0.4,$

…, 0.6, 0.8} (i.e., ten intervals of [−1, 1]). In the first iteration, the quotient of two stochastic values is found by comparing the base values stored in each block in parallel; the block generates "1" as comparison result if the quotient is equal to or larger than the associated base value, and "0" otherwise. This allows the correct interval to be determined by locating the first "0" in the nine outcomes or by the last block if all outcomes are "1" (as per Figure 7); then the base value in each block is updated by splitting the determined interval into ten refined intervals (i.e., {0.82, 0.84, 0.86, …, 0.96, 0.98}) and the searching is executed in the second iteration. As per Figure 7, the final result with the same data resolution of the input (i.e., 0.002) is obtained after three iterations. It is also of interest to calculate the result with a smaller number of TMR blocks (that however cannot attain the smallest data resolution), achieving a trade-off between computation accuracy and circuit size. This will be discussed in the following sections.

B. Proposed Stochastic Divider

A stochastic divider is designed as per the decimal searching algorithm (referred to as the decimal searching and TMR (DS-TMR)-based divider); the circuit includes the following elements:

- M TMR blocks, that perform the comparison between the temporary estimated quotient and the interval corresponding to each block.
- A probability generator (Prob_generator) block, which generates the base probability for each TMR block.
- A decoder, that locates the correct interval for the quotient as per the results of the TMR blocks.
- A probability adjustment (Prob_adjustment) block, that updates the probability in the probability generator block according to the results of the decoder.

To achieve the smallest computation resolution (i.e., $\frac{1}{2^N} \cdot 2$) and provide a high accuracy, the value of M can be given by round ($\sqrt[t]{2^N}$) – 1 and t (the number of iterations required) as 3; for example when $N = 10$, M is equal to 9, and the computation resolution of the divider after three search iterations is 0.002 as discussed previously. Note that for different values of N, t is set to 3 and then the value of M is determined; for precision, the configuration with these values of M and t can be appropriately adjusted, i.e. for example, by setting a larger M with a smaller t, or a smaller M with a larger t. In this paper, the first configuration (with t as a constant equal to 3) is utilized as example when discussing the proposed design in more detail, because as introduced previously at architectural level, it permits a constant output flow for pipelining. Moreover, it is also of interest to consider a smaller value for M when keeping the same value for t; even if an accuracy loss is introduced, the hardware is reduced, so saving area. The divider with different number of TMR blocks is evaluated and analyzed in Section IV.

The proposed DS-TMR-based divider requires a small and constant number of iterations (i.e., t) to obtain the quotient, so it significantly reduces the computation latency needed by the conventional or the binary searching-based dividers. Its design and operations are treated in detail next by taking $N = 10$ as an example.

In the case of $N = 10$, the number of TMR blocks M can be set to nine (i.e., the value range is split into ten intervals); this enables the estimated quotient to have the smallest resolution (i.e., 0.002) after three iterations. Figure 8 shows the proposed divider for this case; the TMR blocks have the same structure as found in the BS-TMR-based divider shown in Figure 6. In each TMR, three independent stochastic sequences are generated for the same probability to reduce the fluctuation errors. The voting result indicates that this probability should be increased or decreased in the next search iteration to reach the correct result. The probabilities set for each TMR block (i.e., the probabilities in each iteration shown in Figure 7) are generated by the Prob_generator block in Figure 8. In the example of Figure 7, the outcomes of the TMR blocks (i.e., *TMR_out* in Figure 8 are "111111111" in the first iteration; they are then provided as input to the decoder and decoded as "1001", which indicates that the ninth TMR block generates the correct interval. Next, the Prob_adjustment block is utilized to refine the probability generated in the Prob_generator block in the next search iteration as per the output of the decoder (i.e., *Dec_out* in Figure 8).

Algorithm 2 illustrates the process of generating the quotient by using the proposed DS-TMR-based divider; like the BS-TMR-based divider, two phases (computation and stabilization) are required. In the computation phase, three independent sequences for the same probability are utilized to reduce the fluctuation errors in each TMR block, thus only the partial bits of the input sequences (i.e., generated in clk_1 periods) are utilized. As discussed previously, the entire range of [−1, 1] is split into $M+1$ intervals during the first iteration of the computation phase and the boundary value of each interval is set as the initial base value for the three N-bit counters in each TMR block (i.e., step 1 in Algorithm 2). Then, the counters start increasing/decreasing the base value to iteratively approach the inputs by capturing the difference between the current quotient sequence (i.e., related to the inputs) and the predicted quotient sequence (i.e., related to the base value/ interval) in clk_1 periods; if the updated probability (i.e., approaching the current quotient) is equal to or larger than the base probability, the comparator (Comp) block flags it out, i.e. it generates a 1, and 0 otherwise. A majority voting is performed on the three outcomes of the comparator blocks (i.e., obtained by utilizing three independent sequences) to determine if the interval is covered (i.e., the result of the TMR block is 1 if the interval is covered, and 0 otherwise). Therefore, at the end of the first search iteration, a decoder block decodes the TMR results to identify the correct interval (i.e., step 17 in Algorithm 2) by locating the first 0 in the M outcomes, or by the last interval if all outcomes are "1" (as discussed in Section III-A). Once the correct interval is determined, it is further split into $M + 1$ intervals and the same steps (as in the first iteration) are performed again for the second iteration; this process is repeated by using the refined intervals in the third iteration.

Subsequently, some additional bits (i.e., generated in clk_2 periods) are utilized in the conventional fashion to further adjust the result in the stabilization phase; this is performed by utilizing a single version of the first TMR block and disabling the other two (redundant) versions and all other TMR blocks. Note that when the number of TMR blocks M is smaller than round ($\sqrt[t]{2^N}$) – 1, the stabilization phase is also helpful to refine the estimated result. The number of bits required in

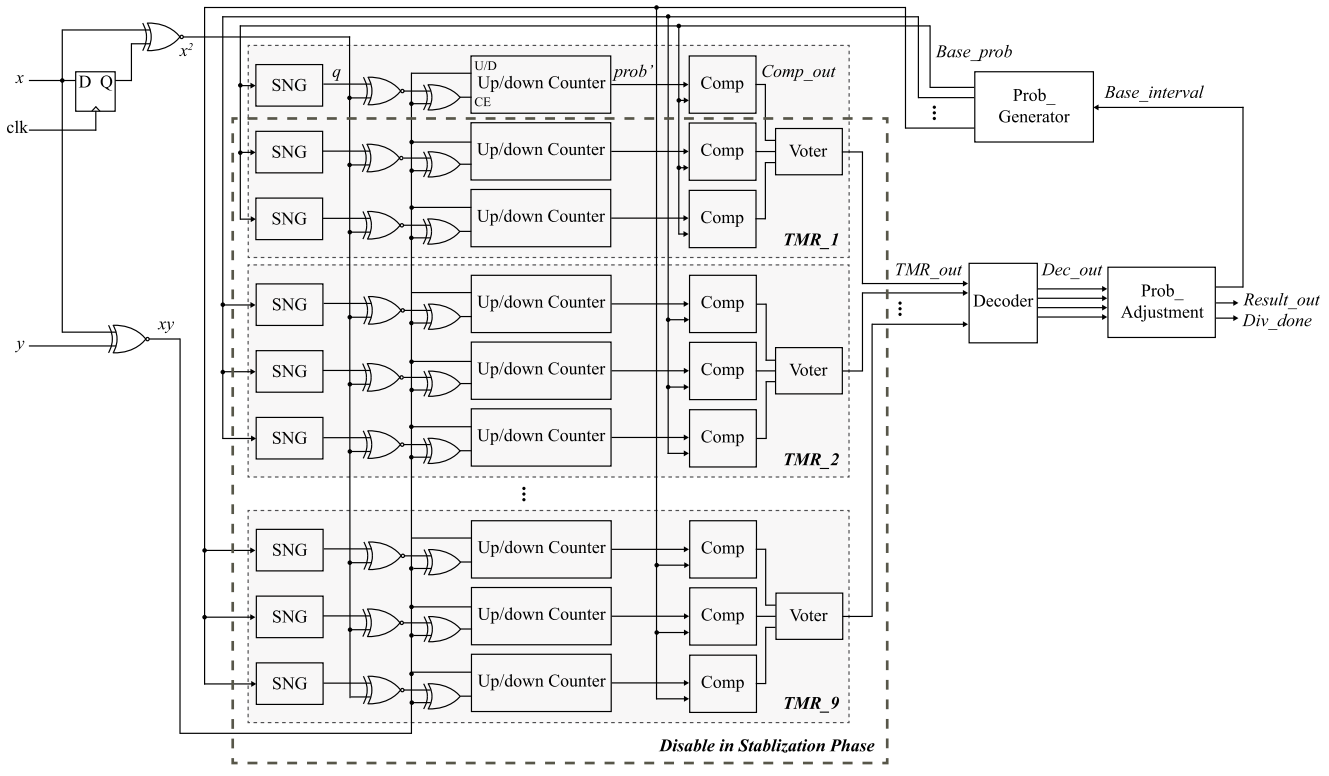

Fig. 8. Design of the proposed divider ($N = 10$, $M = 9$).

each phase (i.e., the values of clk_1 and clk_2) can be carefully selected to achieve a high computation accuracy. Overall, compared with the BS-TMR-based divider that requires N iterations, the proposed divider only requires three iterations (two iterations are crucial as discussed in Section III.D), therefore it is independent of the value of N and taking an advantage in terms of a substantial reduction in computation latency when N is large.

C. Trade-Off Stochastic Design

Although the penalty of the proposed stochastic divider in terms of area is not a primary issue for an SC-based ANN implementation (i.e., the divider is utilized only during the mapping between the input layer and the first hidden layer), it is of interest to consider a trade-off design to reduce power while incurring in a slight decrease in accuracy or increase in latency. A possible solution is to reduce the number of TMR blocks, such that the obtained result is estimated with an inexact data solution. An alternative solution is to utilize single modules (SM) to replace the TMR blocks; by employing the proposed decimal searching-based algorithm, this divider (referred to as the DS-based divider) reduces the circuit size but with a slight accuracy loss; thus, it is attractive for applications in which the ANN can tolerate results with a small deviation from the correct value.

Since fluctuation errors that may occur in some cases, cannot be avoided without voting in the DS-based divider (and causing an accuracy loss), some of the single copies can be utilized in the TMR fashion by introducing an additional iteration to perform the comparison again when fluctuation errors are detected. The operational process is explained in detail by considering $N = 10$ as an example again.

When the trade-off design is implemented, the DS-TMR-based divider (as Figure 8) is modified as shown in Figure 9 and the calculation process is given by Algorithm 3. Similar to the calculation process of the DS-TMR-based divider, the DS-based divider determines the correct interval in each search iteration (i.e., steps 1 to 8 in Algorithm 3); however in this case, only a single copy of each TMR block is required and two voters are kept for the back-up TMR structure. Once the fluctuation errors occur, the output of the nine single modules (i.e., SM blocks in Figure 9) may not consist of several "1" following by several "0". For example, the comparison results of "111000000" in the example of Figure 7 can be "101000000" if a fluctuation error occurs in the second single module. In this case, the modules related to the first "10" are checked again by utilizing the first six SM modules as two TMR blocks (i.e., steps 9 to 19 in Algorithm 3); if the two comparison results are still "10", the fluctuation errors may have occurred in the original third module (i.e., causing "100000000" to become "101000000"), otherwise in the second module (i.e., causing "111000000" to become "101000000"). Subsequently, the correct interval is identified as per the updated comparison result. Therefore, at most twice the number of search iterations required by the DS-TMR-based divider will be needed for the computation phase in this alternative scheme (i.e., adding one more iteration during each search step), but the size of the circuit is significantly reduced. Finally, the estimated quotient is refined in the stabilization phase that is the same as in the DS-TMR-based divider (i.e., step 27 in Algorithm 3).

However, if more than one stochastic sequence is affected by fluctuation errors, the correct interval cannot be located; this is a problem for all dividers based on progressive precision searching (i.e., the BS-TMR-based divider, the proposed

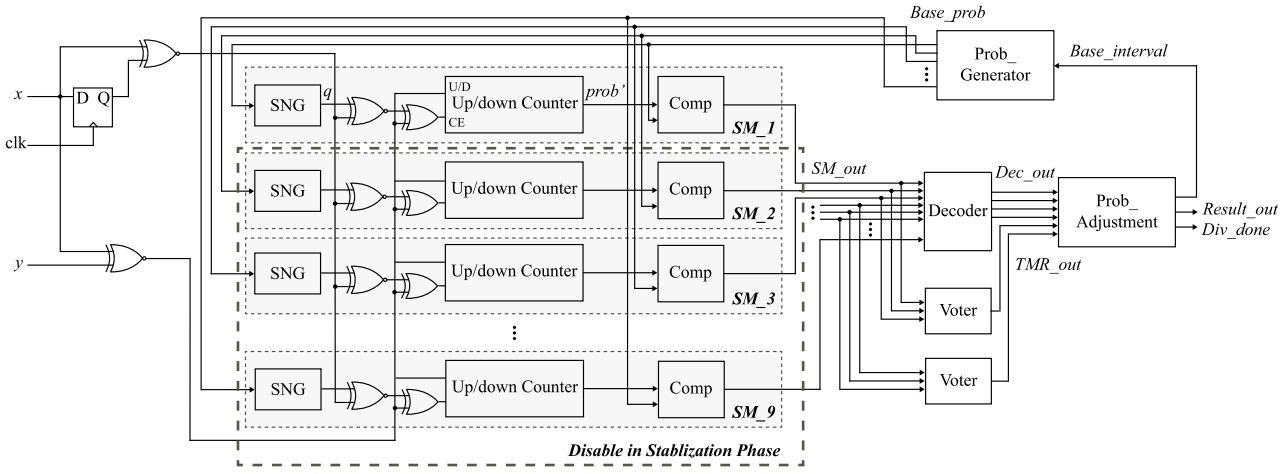

Fig. 9. Design of the proposed divider with trade-off ($N = 10$, $M = 9$).

DS-TMR-based and DS-based dividers). Assume that there are two sequences affected by fluctuation errors; the proposed DS-TMR-based divider fails only when they affect the same TMR block, while the BS-TMR-based divider and the proposed DS-based divider surely fail. Therefore, the trade-off design introduces an accuracy loss in some cases when compared with the DS-TMR-based divider; this is evaluated in the next section.

D. Discussion

As introduced previously, the proposed divider achieves the smallest (or nearly the smallest) computation resolution in the last search iteration of the computation phase (e.g., in the third iteration when $N = 10$, $M = 9$ and $t = 3$ for the previous example); this search process is required for determining at most one additional "1" occurrence in the stochastic sequence. Therefore, when utilizing a small number of bits, it is difficult to capture this small difference (e.g., only a "1") and in some cases, it can even push the result into an incorrect data interval, so making the third search iteration counterproductive for approaching the accurate result. Moreover, since the stabilization phase is also utilized to finely tune the computation result, the third search iteration in the computation phase can be saved to further improve the computation process. Therefore, when the proposed divider is designed by achieving the smallest (or nearly the smallest) computation resolution, only the first t-1 iterations in the computation phase are utilized (i.e., the last iteration (steps 21 to 23 in Algorithm 2 and steps 24 to 26 in Algorithm 3) can be saved), followed immediately by the stabilization phase.

It is of interest to mention that this observation is also applicable to the BS-TMR-based divider [11]; its last few iterations in the computation phase, which are used to identify a small change in the number of "1" in the stochastic sequence, do not contribute to approach the accurate computation result. This will be verified in the following section when evaluating the convergence process of the divider.

IV. Evaluation

A. Computation Accuracy

The computation accuracy achieved by the proposed dividers is first analyzed by considering the example of

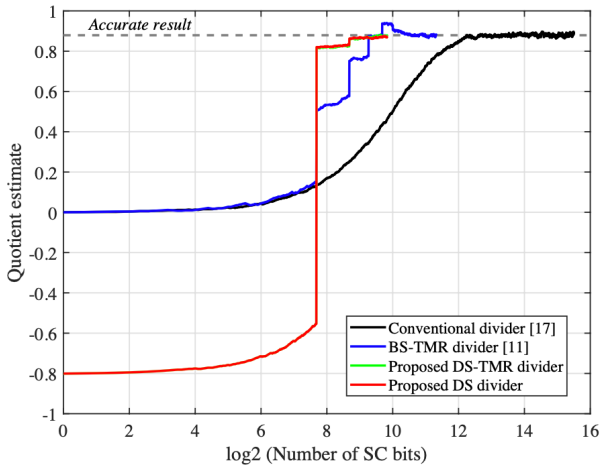

Fig. 10. Quotient for 0.800/0.910 estimated by different stochastic dividers (the green line and the red line are almost overlapped because their values are similar).

$y = 0.800$, $x = 0.910$ and $N = 10$; $M = 9$ TMR (SM) blocks are designed in the proposed DS-TMR-based (DS-based) divider to achieve the data resolution as discussed previously and thus, two iterations are utilized in the computation phase. The conventional divider [17] and the BS-TMR-based divider [11] are also implemented and the generated results are compared. As per the experiments results in [11], 2048 (512) bits are selected in the computation (stabilization) phase of the BS-TMR-based divider to achieve an accurate result, i.e. 205 clock cycles are required in each iteration of the binary searching process, and 512 clock cycles are required during the refined searching in the conventional fashion. Therefore, this configuration is also selected for the proposed two dividers. The convergence processes of the different dividers are compared in Figure 10; since the BS-TMR-based and the proposed dividers search the result in multiple parallel modules (i.e., 3, 27 and 9 respectively) in the computation phase, the results tracked in the first module are utilized to show the convergence process of these dividers.

As per Figure 10, all dividers tend to finally reach the accurate quotient, but the convergence times are different. The conventional divider incurs in the longest convergence process while the proposed dividers are fastest, followed by the BS-TMR-based divider; this is consistent with the

Algorithm 2 Calculation Process of the Proposed DS-TMR-Based Divider*

Input: Stochastic bits x and y;
Output: Quotient *Result_out* (the probability for y/x); signal *Div_done*;
{Computation phase}
{Generate initial base values for the counters in M TMR blocks}
Base_prob = round($i\cdot(2^N/(M+1))$);
prob' = *Base_prob*;
{Estimate the quotient in 1st iteration; each requires clk_1 periods}
{Calculation in each TMR block}
for $i = 1 : clk_1$
  $q(i)$ = **SNG**(*Base_prob*);
  $CE(i)$ = **XOR**($xy(i)$, **XNOR**($x^2(i)$, $q(i)$));
  $UD(i) = xy(i)$;
  if $CE(i) = 1$ and $UD(i) = 1$
    prob' increases by 1, or keeps the same value when reaches the upper boundary;
  elseif $CE(i) = 1$ and $UD(i) = 0$
    prob' decreases by 1, or keeps the same value when reaches the lower boundary;
  else
    prob' = *prob'*;
  end
  Comp_out = 1 if *prob'* ≥ *Base_prob* and 0 otherwise;
end
  Perform majority voting among *Comp_out* to obtain *TMR_out*;
{Find the correct interval as per the results of TMR blocks}
  Dec_out = number of "1" in *Comp_out* + 1, or "1001" when exceeds M;
{Refine the found interval; perform 2nd iteration}
Base_interval = round($i\cdot(2^N/(M+1)^2)$);
Base_prob = *Base_prob* + *Base_interval*;
Repeat steps 2 to 17;
{Refine the found interval; perform 3rd iteration}
Base_interval = round($i\cdot(2^N/(M+1)^3)$);
Base_prob = *Base_prob* + *Base_interval*;
Repeat steps 2 to 17;
{Stablization phase}
{Refine the estimated quotient by using single version in the first TMR block (disable all others) with clk_2 periods}
prob'(1) = *Base_prob* (*Dec_out*);
for $i = 1 : clk_2$
  Repeat steps 4 to 14;
end
Result_out = *Comp_out*;
Div_done = 1;
*Signals are illustrated in Figure 8.

Algorithm 3 Calculation Process of the Proposed DS-Based Divider*

Input: Stochastic bits x and y;
Output: Quotient *Result_out* (the probability for y/x); signal *Div_done*;
{Computation phase}
{Generate initial base values for the counters in M TMR blocks}
Base_prob = round($i\cdot(2^N/(M+1))$);
prob' = *Base_prob*;
{Estimate the quotient in 1st iteration; each requires clk_1 periods}
{Calculation in each SM block}
for $i = 1 : clk_1$
  Perform the same steps 4 to 14 in Algorithm 2;
end
if *Comp_out* = "000…0" or "111…1" or "1…10…0"
  Two voter blocks are disabled;
  Dec_out = number of "1" in *Comp_out* + 1, or "1001" when exceeds M;
else
{A fluctuation error is detected; the first six SM blocks are used as two TMR blocks to compute the result again in the 2nd iteration}
  Error_pos = two positions for the first "10" in *Dec_out*;
  prob'(1:3) = *Base_prob*(*Error_pos*(1));
  prob'(4:6) = *Base_prob*(*Error_pos*(2));
  Repeat steps 3 to 5;
  Perform majority voting among *Comp_out*(1:6) to obtain *TMR_out*;
  if *TMR_out* = "10"
    The correct interval is found as *Error_pos*(2);
  else
    The correct interval is found as *Error_pos*(2)+2 or as the last interval when it exceeds M;
  end
end
{Refine the found interval; perform 2nd or up to 4th iterations}
  Base_interval = round($i\cdot(2^N/(M+1)^2)$);
  Base_prob = *Base_prob* + *Base_interval*;
  Repeat steps 2 to 20;
{Refine the found interval; perform 3rd or up to 6th iterations}
  Base_interval = round($i\cdot(2^N/(M+1)^2)$);
  Base_prob = *Base_prob* + *Base_interval*;
  Repeat steps 2 to 20;
{Stablization phase}
{Refine the estimated quotient by using one SM block (disable all others) with clk_2 periods}
Perform the same steps 24 to 29 in Algorithm 2;
*Signals are illustrated in Figure 9.

analysis presented in the previous section, because the proposed dividers only take a small number of search iterations. For each module of the BS-TMR-based and the proposed dividers, a base probability is set in the first search iteration and its changing direction (i.e., increasing or decreasing) for the next iterations can be identified by the partial bits of the stochastic sequence. Therefore, the plots for these dividers have a stepped shape, starting with the initial base probability

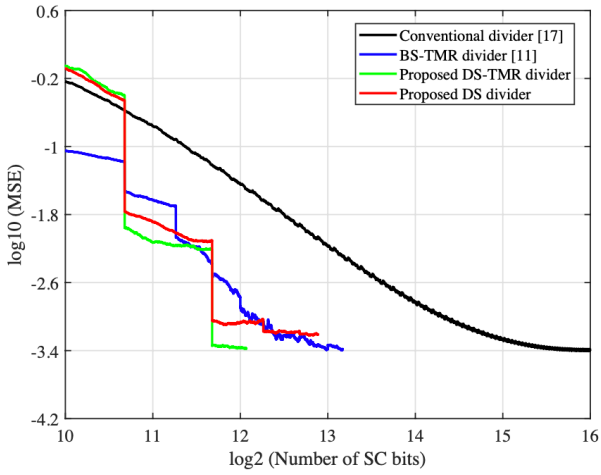

Fig. 11. Accuracy of different stochastic dividers with data resolution of 2^{10} (conventional, BS-TMR-based, proposed DS-TMR and DS -based dividers require 46,341, 9,214, 4,300 and 4,738 (on average as per Table I) SC bits, respectively).

TABLE I

DISTRIBUTION OF THE NUMBER OF ITERATIONS REQUIRED BY THE PROPOSED DS-BASED DIVIDER

Number of iterations	Percentage
2	80.9%
3	11.5%
4	7.6%

TABLE II

MSE OF THE PROPOSED DIVIDERS WITH DIFFERENT NUMBER OF BLOCKS M

Divider		MSE
Conventional [17]		$10^{-3.4}$
BS-TMR [11]		$10^{-3.4}$
Proposed DS-TMR	$M=3$	$10^{-2.7}$
	$M=5$	$10^{-2.8}$
	$M=7$	$10^{-3.1}$
	$M=9$	$10^{-3.4}$
Proposed DS	$M=7$	$10^{-3.0}$
	$M=9$	$10^{-3.2}$

set in the corresponding module (i.e., 0 in the conventional and BS-TMR-based divider and -0.8 in the proposed dividers).

In the second experiment, the accuracy in terms of Mean Square Error (MSE) is evaluated by testing a larger number of random data pairs within the range of $[-1, 1]$ as inputs of the divider (i.e., so for both the divisor and dividend); since it has been found that the results for 10,000 trials were consistent with those with 100,000 trials, 10,000 data pairs are selected in the next experiments. Results for different stochastic dividers are compared in Figure 11; all dividers can finally achieve an MSE of $10^{-3.4}$, except for the proposed DS-based divider that is slightly worse because it is more vulnerable to fluctuation errors (as discussed previously). It is also of interest to study the accuracy of SC dividers for different input values. As indicated in [24], the accuracy of some SC circuits (e.g., the multiplier) is lower when the inputs are around 0.5 for the unipolar representation (i.e., nearly centered in the possible value range), and thus, around 0 for the bipolar computation; this is also the case for the SC dividers (in which the basic units are also multipliers) as per our simulation results (given in Figure 12). The improvement in the computation accuracy of SC arithmetic units in such value ranges is left for future work.

In terms of convergence, the conventional divider requires the largest number of SC bits to reach a stable state (i.e., $2^{15.5} = 46341$), while the other dividers are significantly faster. To achieve the same MSE as the conventional dividers, 819 (1638) bits in each search iteration of the computation phase and 1024 bits in the stabilization phase are required by the BS-TMR-based divider (the proposed DS-TMR-based divider), i.e., 9214 (4300) SC bits in total. Therefore, the proposed DS-TMR-based divider only requires 9.3% (46.7%) clock cycles compared to the conventional divider (the BS-TMR-based divider), providing a faster convergence. As for the proposed DS-based divider, its convergence process presented in Figure 11 is for the worst case (i.e., four search iterations in total in the computation phase); however as per the distribution of the required number of search iterations among the 10,000 data pairs (given in Table I), the DS-based divider requires on average 4738 clock cycles in total. Therefore, compared with the conventional divider (BS-TMR-based divider), the DS-based divider requires on average 10.22% (51.4%) clock cycles. Note that the BS-TMR-based divider does not significantly reduce the MSE after approximately $2^{12.3}$ bits; therefore, this indicates that the last few search iterations in the computation phase do not significantly affect the calculation as discussed previously. Note that if the dividers are connected to other SC units for subsequent computation in a given application (e.g., the case studied in Section V), the stochastic sequence can be represented by the last 2^N bits in the conventional divider and by the 2^N bits starting at the beginning of the stabilization phase of the BS-TMR-based divider and the proposed dividers, because the changes of MSE within these SC bits of all dividers are extremely low.

In addition to the case of $M = 9$, the accuracy of the proposed dividers with different number of blocks is also evaluated and compared; in particular, $M = 3$, 5 and 7 are considered for the DS-TMR-based divider and $M = 7$ is considered for the DS-based divider (that requires at least 6 blocks for two back-up TMRs). Table II presents the MSE results for different cases. As per Table II, the proposed dividers with a smaller number of TMR blocks tend to provide a larger MSE, because when the number of intervals decreases, the computation result is roughly estimated (with an inexact data resolution). The proposed dividers with $M = 7$ and 9 are considered in the evaluation for hardware overhead next, because all of them have good MSE results (i.e., $<10^{-3}$).

In the third experiment, the accuracy and the required number of SC bits are measured in the case of different data resolutions (including $2^N = 2^8$, 2^9, 2^{10}, 2^{11} and 2^{12}). The configuration for the two phases of the BS-TMR-based divider and the proposed dividers are also proportionally scaled. Figure 13 presents the average MSE results for 10,000 random data pairs and the number of SC bits required by different stochastic dividers to achieve such an accuracy, respectively. Compared with the conventional divider, both the BS-TMR-based divider and the proposed DS-TMR-based divider achieve the same MSE, but they significantly reduce the number of clock cycles (i.e., requiring a smaller number of SC bits); thus, the dividers with the progressive precision

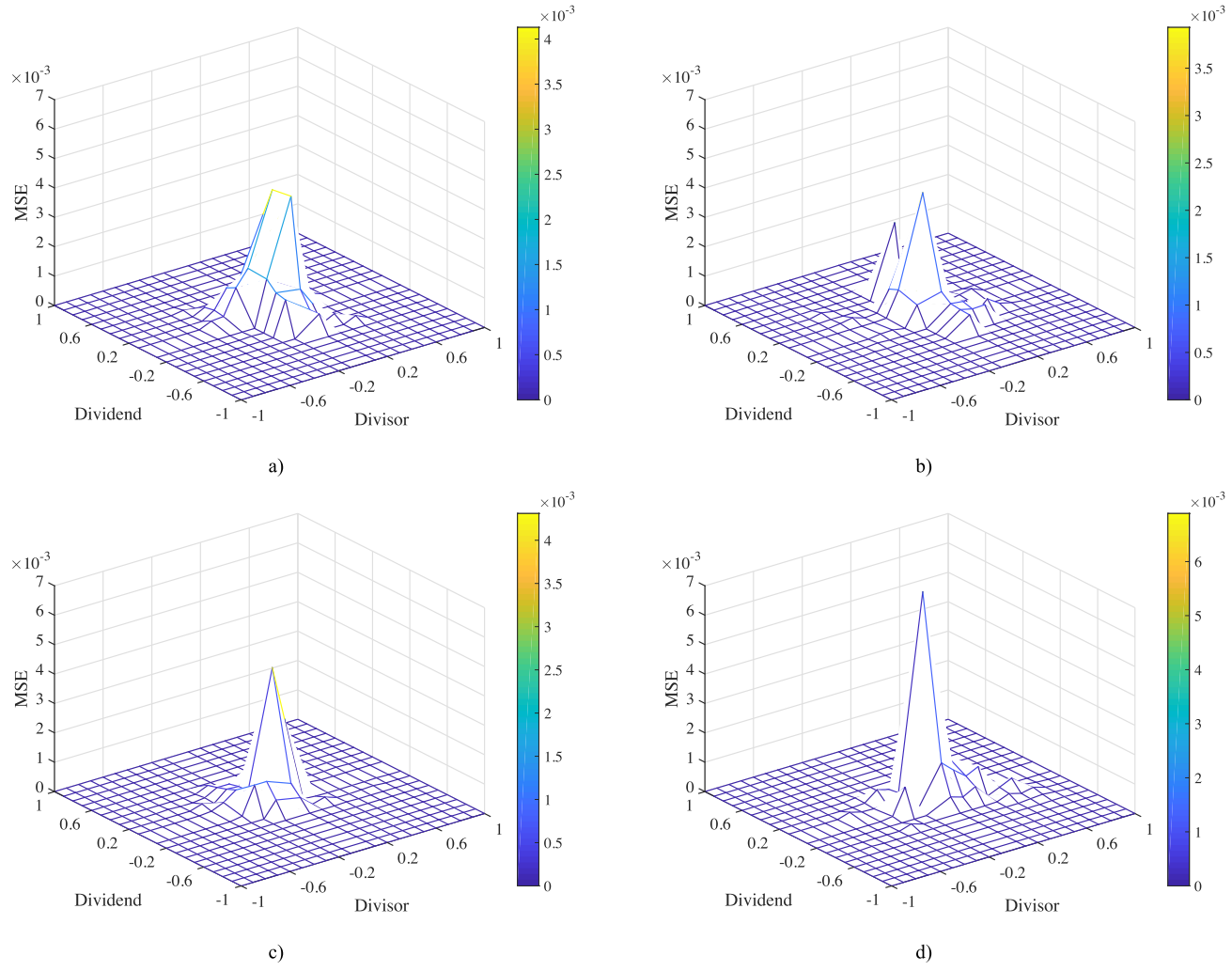

Fig. 12. MSE of different stochastic dividers for different input values: a) Conventional divider [17]; b) BS-TMR divider [11]; c) Proposed DS-TMR divider; d) Proposed DS divider.

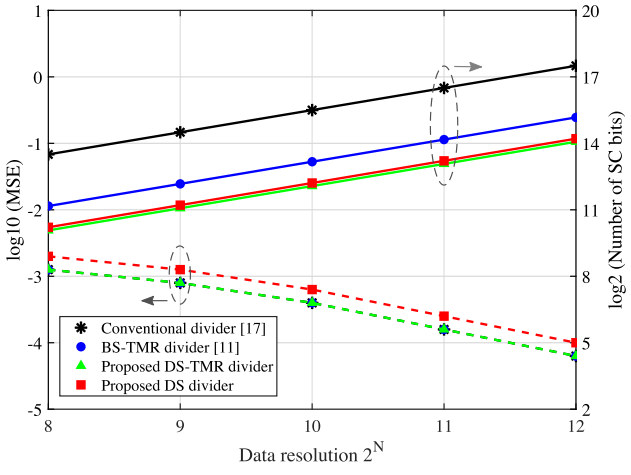

Fig. 13. Accuracy (left y axis) and corresponding number of SC bits (right y axis) for different stochastic dividers with different data resolutions.

are more attractive due to the fast computation process. In this case, the proposed DS-TMR-based divider is a better choice, because it has the same accuracy as the exiting BS-TMR-based divider in all cases, but it reduces the number of clock cycles, and therefore the computation latency.

TABLE III
SYNTHESIS RESULTS OF DIFFERENT STOCHASTIC DIVIDERS

		Area (μm^2)	Latency (ns)	Power (mW)
Conventional [17]		653.6	10380.3	55.6
BS-TMR [11]		2770.6	1233.7	36.5
Proposed DS-TMR	$M = 7$	10955.1	463.4	76.5
	$M = 9$	13900.5	463.4	84.9
Proposed DS	$M = 7$	6320.2	501.0	21.9
	$M = 9$	8577.0	501.0	30.9

Since the proposed DS-TMR-based divider has the fastest convergence speed, it is of interest to evaluate the accuracy that the other dividers can achieve once the DS-TMR-based divider completes the computation. Results are presented in Figure 14; the proposed DS-TMR-based divider provides a significantly smaller MSE than other dividers and such advantageous feature increases for a larger N.

B. Hardware

To evaluate and compare the hardware of different designs, dividers with $N = 10$ have been implemented in HDL and mapped to the ASAP 7 nm library (with typical corners) [25] using the Synopsis Design Compiler. The synthesis tool has

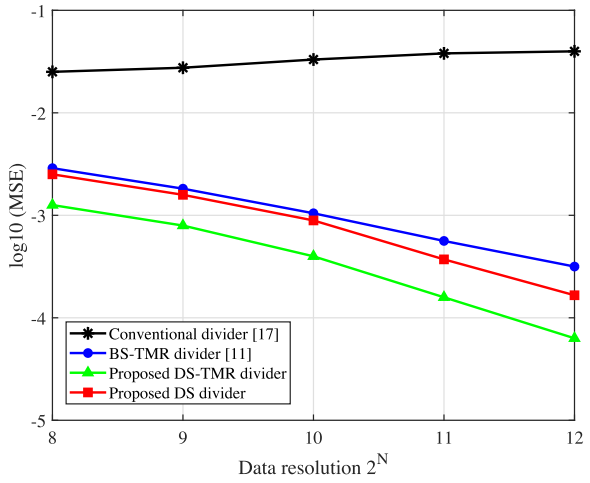

Fig. 14. MSE of different stochastic dividers when the proposed DS-TMR-based divider completes the computation process.

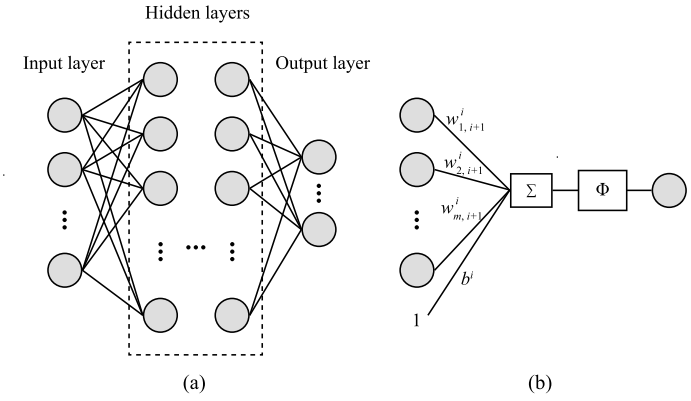

Fig. 15. A Multi-layer perceptron: (a) its structure; (b) one neuron computation in the hidden or output layer.

been set to area, delay and power optimization (with the default toggle rate of 0.5) in the circuitry to obtain the results for these metrics. Table III presents the synthesis results for different dividers, including the area of the circuity, the latency and power dissipation (including dynamic and static powers) to calculate the final result. Note that for the proposed DS-based divider, the average number of required bits as per Table I is considered in this evaluation; the proposed dividers with different numbers of blocks that can achieve similar accuracy, are also considered. Since the blocks operate in parallel, the value of M does not affect the number of clock cycles required by the dividers (that account for the bulk of the computation operations); thus the latency results for the dividers with $M = 7$ and $M = 9$ have an extremely small difference (due to the decoder that is utilized only once during each search iteration), and this is not evidenced in Table III due to the unit. As per Table III, the conventional divider incurs in the smallest area, but it introduces a significant latency; the BS-TMR-based divider reduces the latency by 88.1% and power by 34.4%, at the cost of 3.2 times the additional area. As for the proposed dividers, the DS-TMR-based divider requires 3 to 4 times the additional area and 1.1 to 1.3 times in additional power over the BS-TMR-based divider, but it further reduces the latency by an additional 62.4%; while the proposed DS-based divider requires 1.3 to 2.1 times of additional area over the BS-TMR-based divider, but it reduces the latency by 59.4% and the power dissipation by 15.3% to 40.0%.

Overall, the proposed stochastic dividers operate significantly faster than existing dividers at the cost of area and in some cases power; however, this is not a primary issue when the implementation of an SC-based ANN is considered because it accounts for a small fraction of the entire implementation, so the impact of the increased area is rather low. The overhead of applying a divider in an SC-based Multi-Layer Perceptron is evaluated in the next section.

V. Application: SC-Based Multi-Layer Perceptron

As introduced previously, due to its low complexity and error tolerance, SC is extremely attractive for hardware implementation of different types of ANNs (especially complex networks) that include a very large number of arithmetic operations [11]–[15]. For example, a widely used ANN to perform classification tasks is the Multi-Layer Perceptron (MLP, shown in Figure 15 (a)); it has one input layer, at least one hidden layer and one output layer. In general, to predict the class for an input element, its valid features are normalized and then used as values for the neurons; by performing the mapping between neurons in neighbor layers (this process is given in Figure 15 (b)), the output neurons are finally obtained to predict the class. The calculation performed in each neuron in the hidden and the output layers is given by Eq. (3), where $w^i_{j,i+1}$ denotes the weight of the j^{th} neuron in the i^{th} layer (that has m neurons), b^i is the so-called bias of this layer, and Φ is the activation function to active the calculation result (e.g., *tanh*, *ReLU* and variants such as the clamped *ReLU* are widely used).

$$Neuron_{i+1} = \Phi\left(\sum_{j=1}^{m} w^i_{j,i+1} \cdot Neuron_j + b^i\right) \quad (3)$$

Since ANNs require a large number of neurons and thus, executing many arithmetic operations for neural computation, SC units are very attractive in terms of hardware implementation to replace traditional circuits and save hardware. Recently, an efficient SC framework for MLPs, that uses a hybrid structure of the traditional SC and ESL units, has been introduced in [11]; for mapping between neurons in the input layer and the first hidden layer, ESL units are utilized for a better computation accuracy, because addition among multiple multiplication results can exceed the traditional SC computation range of $[-1, 1]$. ESL units can be employed in each layer of the network; however, the accuracy may not be further increased [11], while the size of the entire circuits is nearly doubled as discussed previously. Therefore, traditional SC units are used for the other layers. In such a hybrid SC-based MLP, in addition to the multiplier, adder and activation function units (as per Eq. (3)), a divider is also required to perform the conversion between the conventional stochastic and the ESL sequences. The divider affects the critical computation path in the MLP implementation, but it is employed only during the mapping between the input and the first hidden layer; therefore, its performance in terms of reduced latency is of primary importance. The hardware overhead of the divider is not such a critical metric, because a low latency stochastic divider is useful to overcome the latency disadvantage of an SC-based ANN.

Next, such an MLP is implemented and the effectiveness of the proposed dividers is evaluated by utilizing three widely used multi-classification datasets (all have ten classes): the MNIST dataset [26], the Fashion-MNIST dataset [27] and the SVHN dataset [28]. The traditional 32-bit floating point (FP)-based MLP is also implemented for comparison. The MLP models for different datasets that have been pre-trained in the FP version in Matlab by using the 10-fold cross-validation method, are as follows.

- For the MNIST dataset, the MLP model has four layers (i.e., two hidden layers), in which the clamped *ReLU* is used as activation function for each hidden layer and *tanh* is used for the output layer; the number of neurons in each layer is 784, 256, 128 and 10, respectively.
- For the Fashion-MNIST dataset, the MLP model has five layers (i.e., three hidden layers), in which the clamped *ReLU* is used as activation function for each hidden layer and *tanh* is used for the output layer; the number of neurons in each layer is 784, 256, 128, 128 and 10, respectively.
- For the SVHN dataset (that is processed by converting RGB into grayscale images to reduce the data size; five left and right pixels of each image are removed to reduce the distraction), the MLP model has five layers (i.e., three hidden layers), in which the clamped *ReLU* is used as activation function for each hidden layer and *tanh* is used for the output layer; the number of neurons in each layer is 704, 512, 512, 512 and 10, respectively.

Next, these models are used to evaluate the performance of different ANN implementations in terms of classification accuracy on the testing set and the hardware for performing the classification for one sample (i.e., the inference process).

The SC activation functions (i.e., the *tanh* and the clamped *ReLU*) are implemented as per the designs presented in [11]. Note that in the SC-based MLP implementation, the model parameters (i.e., the weights and bias) are converted to SC values prior to storing them in memory; once the weights are read out, they can be directly used as input to the stochastic number generators (as probability), i.e., to generate stochastic sequences. Therefore, the weight precision is the same as the SC computation resolution, which is 0.002 when $N = 10$ for this case study. For the weights related to neurons in the first/input layer (i.e., used for ESL units), they are converted to quotients prior to SC value conversion if their absolute value is larger than 1; an additional bit is required as the most significant position of the data representation to indicate whether the stored value is used for the numerator stochastic sequence, or the denominator sequence in the ESL units. For the weights related to the neurons in the other layers (i.e., used for the conventional SC units), they are bounded as 1 or -1 prior to converting to SC values, if their absolute value is larger than 1. Therefore, in the SC implementation, the memory used to store the parameters can also be reduced compared to the FP version, because only $N+1$ bits (N usually takes a value between 8 and 20 in the works found in the technical literature [11], [21]) are used to represent one data instead of 32 bits (when the IEEE 754 format is used for the FP numbers [29]).

Table IV presents the classification accuracy of the FP-based MLP, as well as the SC-based MLP with different dividers; compared to the SC-based MLPs with existing dividers, the MLP with the proposed dividers achieves the same accuracy when $M = 9$, while the DS divider with $M = 7$ (so, a trade-off design) introduces a loss of at most 0.9% in accuracy. Even though all SC-based MLPs slightly reduce the accuracy compared to the FP-based version, this can be addressed by training the model by also utilizing an SC design and/or techniques that specifically address accuracy in SC designs, as introduced previously; this is not further investigated as it is outside of the scope of this paper (which mostly addresses the computational latency of an SC design).

TABLE IV
CLASSIFICATION ACCURACY FOR THE FLOATING POINT-BASED MLP AND SC-BASED MLP WITH DIFFERENT DIVIDERS

MLP			MNIST	Fashion-MNIST	SVHN
FP-based MLP			**98.3%**	**88.9%**	**85.5%**
SC-based MLP with different dividers	**Conventional [17]**		97.0%	87.9%	83.8%
	BS-TMR [11]		97.0%	87.9%	83.8%
	Proposed DS-TMR	$M = 7$	97.0%	87.9%	83.8%
		$M = 9$	97.0%	87.9%	83.8%
	Proposed DS	$M = 7$	96.6%	87.0%	83.5%
		$M = 9$	97.0%	87.9%	83.8%

The hardware design of the MLP in different schemes is implemented in HDL and the overhead is also evaluated by conducting the same synthesis method as the one in the evaluation for the dividers; the clock frequency of the circuits is set to 200 MHz as an example for evaluating the hardware overhead (higher clock frequency can also be selected, but it has no impact on the qualitative comparison results). The area, latency and power dissipation (including dynamic and static powers) of the FP-based MLP and the SC-based MLP with different dividers are compared in Table V; as per these results, the SC-based MLPs significantly reduce the area compared with the FP-based MLP (i.e., by 45.9% to 57.3% for the considered datasets) due to the low complexity of the SC-based computation units. The latency and power dissipation to perform the calculation and then classification for one data sample depends on the number of clock cycles; thus, the latency/power required for the SC-based MLP depends on the number N as well as the effectiveness of the divider (as it is in the critical path). As discussed previously, the stochastic sequence can be represented by the last 2^N bits in the conventional divider and by the 2^N bits starting at the beginning of the stabilization phase of the BS-TMR-based divider and the proposed dividers, no additional SC bits are required for the units following the divider, because the stabilization phase of the BS-TMR-based and the proposed dividers exactly utilize 1024 bits. Therefore, the SC-based MLPs require the same number of clock cycles as the employed dividers in all cases.

To evaluate all MLPs, a combined figure of merit is used in this paper: this is given by the product of area, latency, power, and number of clock cycles (PALPC). PALPC accounts not only for the hardware design (so inclusive of the divider circuit), but also for the total computation complexity (given by the number of clock cycles) needed for the classification task using a specific dataset.

TABLE V

SYNTHESIS RESULTS OF THE FLOATING POINT-BASED MLP AND SC-BASED MLP WITH DIFFERENT DIVIDERS (THE SMALLEST OVERHEAD IN DIFFERENT METRIC IS MARKED IN BOLD)

MLP				Area		Latency		Power		# Clock Cycles	PALPC (%)
				mm^2	%	ns	%	mW	%		
MNIST (784-256-128-10)	**FP-based MLP**			36.8	100	6363.7	100	1325.2	100	**3376**	100
	SC-based MLP with different dividers	**Conventional [17]**		**15.7**	**42.7**	34269.1	538.5	**207.4**	**15.7**	46341	493.6
		BS-TMR [11]		16.1	43.8	6134.9	96.4	208.1	**15.7**	9214	18.1
		Proposed DS-TMR	$M = 7$	18.1	49.2	**2851.5**	**44.8**	211.6	16.0	4300	**4.5**
			$M = 9$	18.8	51.1	**2851.5**	**44.8**	212.1	16.0	4300	4.7
		Proposed DS	$M = 7$	17.0	46.2	3113.1	48.9	208.3	15.7	4738	5.0
			$M = 9$	17.5	47.6	3113.1	48.9	208.8	15.8	4738	5.1
Fashion-MNIST (784-256-128-128-10)	**FP-based MLP**			36.8	100	7343.9	100	1923.6	100	**3896**	100
	SC-based MLP with different dividers	**Conventional [17]**		**16.6**	**45.1**	43934.5	598.2	**218.2**	**11.3**	46341	364.1
		BS-TMR [11]		17.1	46.5	6205.8	84.5	218.9	11.4	9214	10.6
		Proposed DS-TMR	$M = 7$	19.2	52.2	**2923.1**	**39.8**	222.4	11.6	4300	**2.6**
			$M = 9$	19.9	54.1	**2923.1**	**39.8**	222.9	11.6	4300	2.8
		Proposed DS	$M = 7$	18.0	48.9	3180.7	43.3	219.1	11.4	4738	2.9
			$M = 9$	18.6	50.5	3180.7	43.3	219.6	11.4	4738	3.0
SVHN (704-512-512-512-10)	**FP-based MLP**			125.4	100	8045.1	100	6421.2	100	**4208**	100
	SC-based MLP with different dividers	**Conventional [17]**		**54.3**	**43.3**	31605.5	392.6	**692.7**	**10.8**	46341	202.1
		BS-TMR [11]		55.3	44.1	5799.9	72.1	694.1	**10.8**	9214	7.5
		Proposed DS-TMR	$M = 7$	59.5	47.4	**2824.7**	**35.1**	701.2	10.9	4300	**1.9**
			$M = 9$	61.0	48.6	**2824.7**	**35.1**	702.2	10.9	4300	**1.9**
		Proposed DS	$M = 7$	57.2	45.6	3058.8	38.0	694.4	10.8	4738	2.1
			$M = 9$	58.3	46.5	3058.8	38.0	695.4	10.8	4738	2.2

Since the divider is utilized during the computation for only the input layer, the SC implementation is more efficient than the FP implementation in terms of latency/power when the MLP has a larger number of hidden layers and neurons. This has been verified by the datasets considered in this paper; as per Table V, the relative latency/power required for the SC-based MLP over the FP-based MLP for the MNIST datasets tends to be lower than the Fashion-MNIN dataset (followed by the SVHN dataset) when using the same design. The results in Table V also confirm that the advantage of the proposed SC dividers; when compared with existing designs using the conventional divider (BS-TMR-based divider), the SC-based MLP with the proposed dividers increases the area but significantly reduces the required number of clock cycles, so achieving 98.9% to 99.3% (70.7% to 75.5%) saving in terms of PALPC. Moreover, compared with the FP-based MLP, the SC-based MLP with the proposed designs also provides a better PALPC (with 94.9% to 98.1% saving). This trend becomes more obvious as the MLP complexity increases, because the latency/power required for the MLP mostly depends on both the circuit complexity and the number of required clock cycles to perform the classification task. When the network is more complex or larger, the SC circuits are significantly simpler than the FP circuits; moreover, the number of clock cycles required for classification in the SC-based MLP is fixed (it depends on the divider with a complexity that is network independent), while in the FP-based MLP it increases (so, with a complexity that is network dependent).

The results of Table V have verified that even though the proposed SC dividers incur in an increase in area over current dividers (as per Table III), the impact on the entire area overhead of an SC-based ANN is rather low, while their benefits in terms of delay and power reduction (thus reflected also in the PALPC) are significant. Overall, the proposed SC dividers make the SC implementation more attractive for the ANN (e.g., MLP), especially for more complex ANNs, because they provide a significant advantage in terms of all considered figures of merit (i.e., the PALPC) and only introduce a very small classification accuracy loss. Moreover, it is important to note that the delay and power consumption tend to accumulate for each classification in an ANN, so the advantage of the proposed SC dividers is critical at the system level.

VI. CONCLUSION

This paper has proposed novel designs for the divider to significantly reduce the latency encountered in stochastic computations (SC) as mostly related to ANN implementations. This circuit has a significant ramification through the entire operation of an SC-based MLP because it significantly affects its entire performance, so inclusive of power dissipation and computation complexity (given by the number of cycles for performing a classification task using ANNs for a dataset). Initially, a decimal searching algorithm has been first presented for calculating the quotient of a divider with progressive precision. This algorithm has then been employed to design two fast stochastic dividers for use in low latency stochastic Artificial Neural Networks (ANNs). Initially, a decimal searching and TMR (DS-TMR)-based divider has been proposed; this divider only requires two iterations to perform the searching algorithm, so independent of the stochastic sequence length; this advantage leads to a significant low computation latency compared with existing dividers. However, some additional area and power dissipation are introduced by the TMR blocks. Therefore, a DS-based divider that utilizes single modules instead of TMR blocks, has also been presented; such divider achieves a considerable trade-off between area/power and latency, however it introduces a slight computation accuracy loss, so providing an alternative option for different applications.

The divider is utilized in an SC-based ANN to perform conversion between the extended stochastic logic and the standard SC units; it accounts for a small fraction of the

area, but it significantly affects the critical computation path. Even though the proposed dividers incur in an increase in area over existing dividers, such overhead has a miniscule impact on the entire SC-based ANN implementation, because ANN designs using the proposed dividers show substantial benefits in terms of significant reductions in delay and power for the entire network (while retaining the same accuracy as current SC-based ANNs). An SC-based Multi-Layer Perceptron (MLP) has then been analyzed as an application to evaluate the effectiveness of the proposed stochastic dividers; results have shown that the MLP with the proposed dividers require the lowest hardware overhead (with more than 98.9% (70.7%) saving in the PALPC compared to the MLP with the conventional (BS-TMR-based) divider), while achieving the same classification accuracy (or incurring in a loss of up to 0.9% when a trade-off design is employed). The traditional implementation of MLP using 32-bit floating point (FP) is also evaluated and compared to show the advantage/disadvantage of the SC design; results show that the SC-based MLP reduces the PALPC by over 94.9%; moreover, the saving in terms of delay and power at the system level (i.e., the number of clock cycles is involved) tend to accumulate for each classification operation; therefore, the proposed designs are more attractive for SC-based ANNs used in high-speed and resource-constrained applications. Finally, it is also of interest to evaluate the advantages/disadvantages of SC-based ANNs with other types of ANNs and utilize/integrate an SC implementation in them; these topics are left for future work.

ACKNOWLEDGMENT

The authors would like to thank Dr. Jie Han and Dr. Yidong Liu from the University of Alberta, Canada, and Dr. Wei Tang from New Mexico State University, USA for their support and help for this work.

REFERENCES

[1] S. Das, A. Dey, A. Pal, and N. Roy, "Applications of artificial intelligence in machine learning: Review and prospect," *Int. J. Comput. Appl.*, vol. 115, no. 9, pp. 31–41, Apr. 2015.

[2] O. P. Patel, N. Bharill, A. Tiwari, and M. Prasad, "A novel quantum-inspired fuzzy based neural network for data classification," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 1031–1044, Apr. 2021.

[3] S. S. Haykin, *Neural Networks and Learning Machines*. Upper Saddle River, NJ, USA: Pearson, 2009.

[4] F. A. Aoudia and J. Hoydis, "Towards hardware implementation of neural network-based communication algorithms," in *Proc. IEEE 20th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jul. 2019, pp. 1–5.

[5] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 77–84.

[6] Y. Lu, K. Xie, G. Xu, H. Dong, C. Li, and T. Li, "MTFC: A multi-GPU training framework for Cube-CNN-based hyperspectral image classification," *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 17, 2020, doi: 10.1109/TETC.2020.3016978.

[7] H. Zheng and A. Louri, "Agile: A learning-enabled power and performance-efficient network-on-chip design," *IEEE Trans. Emerg. Topics Comput.*, early access, Jun. 18, 2020, doi: 10.1109/TETC.2020.3003496.

[8] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossellè, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 551–564, Mar. 2016.

[9] H. Kaul, M. Anders, S. Mathew, S. Kim, and R. Krishnamurthy, "Optimized fused floating-point many-term dot-product hardware for machine learning accelerators," in *Proc. IEEE 26th Symp. Comput. Arithmetic (ARITH)*, Jun. 2019, pp. 84–87.

[10] H. Kim, M. S. Kim, A. A. Del Barrio, and N. Bagherzadeh, "A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks," in *Proc. IEEE 26th Symp. Comput. Arithmetic (ARITH)*, Jun. 2019, pp. 108–111.

[11] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1273–1286, Sep. 2018.

[12] A. Ren *et al.*, "SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2017, pp. 405–418.

[13] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2688–2699, Oct. 2017.

[14] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-learning stochastic computational deep belief network," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 454–465, Sep. 2018.

[15] Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2213–2221, Sep. 2019.

[16] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Boston, MA, USA: Springer, 1969, pp. 37–172.

[17] B. D. Brown and H. C. Card, "Stochastic neural computation I: Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[18] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, May 2013.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[20] J. H. Anderson, Y. H. Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," *IEEE Design, Autom. Test Eur. Conf. Exhib. (DATE)*, pp. 1550–1555, 2016.

[21] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel Sobol sequences," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1326–1339, Jul. 2018.

[22] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 116–121.

[23] S.-I. Chu, "New divider design for stochastic computing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 1, pp. 147–151, Jan. 2020.

[24] T. J. Baker and J. P. Hayes, "The hypergeometric distribution as a more accurate model for stochastic computing," in *Proc. IEEE Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 592–597.

[25] (2021). *ASAP, ASAP: Arizona State Predictive PDK*. [Online]. Available: http://asap.asu.edu/asap/

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[27] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: http://arxiv.org/abs/1708.07747

[28] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learning. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[29] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic," *Lect. Notes Status IEEE*, vol. 754, no. 1776, p. 11, May 1996.