A Delta Sigma Modulator-Based Stochastic Divider

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

***Abstract*— The divider is one of the most complex hardware units in Stochastic Computing (SC); even though several new designs have been presented to reduce the computation latency of the conventional divider, all of them still require a considerable number of clock cycles. Moreover, they incur in low performance due to the employed arithmetic computational scheme. In this paper, a Delta Sigma Modulator (DSM) based stochastic divider is proposed. As an entirely digital circuit, the proposed divider offers the best computation latency and accuracy over all existing stochastic dividers found in the technical literature (with a typical reduction between 66.8% and 96.9% in the number of clock cycles and a reduction from $10^{-3.4}$ to $10^{-3.9}$ in the average mean square error for a 10-bit resolution). An SC-based Neural Network (NN) is considered as an initial case study to evaluate the advantages of the proposed design in an emerging application; results show that the proposed divider enables an SC-based NN to achieve a higher classification accuracy and hardware efficiency than existing designs. To show the flexibility of the proposed divider design, its application to Sobol-based sequences is also presented; also in this case, its superiority over other designs is confirmed. These features make the proposed design very attractive for hardware-constrained platforms; moreover, such a novel design approach that incorporates ideas from analog/mixed signal circuit design into a digital circuit design, can motivate other researchers to design efficient SC designs using similar schemes.**

***Index Terms*— Stochastic computing, divider, Delta Sigma modulator, neural network.**

Manuscript received 7 January 2022; revised 15 March 2022; accepted 14 April 2022. Date of publication 27 April 2022; date of current version 28 July 2022. The work of Shanshan Liu and Wei Tang was supported by the U.S. Department of Defense under Contract W52P1J2093009. The work of Farzad Niknia, Ziheng Wang, and Fabrizio Lombardi was supported by the NSF under Grant CCF-1953961 and Grant 1812467. The work of Pedro Reviriego was supported by the Spanish Agecia Estatal de Inves-tigation under Grant PID2019-104207RB-I00 and Grant TSI-063000-2021-127, and in part by the Madrid Community Research Project under Grant TAPIR-CM P2018/TCS-4496. The work of Ahmed Louri was supported by the NSF under Grant CCF-1812495 and Grant 1953980. This article was recommended by Associate Editor F. Z. Z. Rokhani. *(Xiaochen Tang and Shanshan Liu contributed equally to this work.) (Corresponding author: Shanshan Liu.)*

Xiaochen Tang, Shanshan Liu, and Wei Tang are with the Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88001 USA (e-mail: ssliu@ece.neu.edu).

Farzad Niknia, Ziheng Wang, and Fabrizio Lombardi are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA.

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

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.2022.3168286.

Digital Object Identifier 10.1109/TCSI.2022.3168286

I. Introduction

TODAY'S computing systems involve large volume of data (to be stored or computed). However, their operations face major challenges in terms of hardware complexity; for example, for Machine Learning (ML) applications, a Neural Network (NN) may require millions of neurons [1], so computing a very large number of arithmetic operations and thus incurring in a large overhead for implementation. Therefore, better designs are pursued for reducing the computation complexity (e.g., approximated or quantized schemes for NNs [2]) and meeting the need of low hardware overhead when used in hardware-constrained platforms (e.g., Internet of Things devices). An alternative solution targets the hardware implementation directly by utilizing emerging computing circuits; this approach does not need to modify the algorithm and is able to be integrated with algorithmic improvements to further reduce the hardware overhead [3], thus making it more attractive for efficient implementations.

A promising computing paradigm is Stochastic Computing (SC); it was firstly proposed in the 1960s [4] and recently has attracted significant attention for implementing power-ful computing systems (which require a large number of arithmetic operations) like NNs [5], image processing [6], [7], digital filters [8], [9] and decoding of complex error correction codes [10], [11] due to its low computational complexity. SC utilizes the probability of occurrence of "1" in a random binary bit stream (typically generated by using a linear feedback shift register (LFSR)) to represent a real value. Therefore, different from conventional binary arithmetic implementations, SC arithmetic units perform arithmetic computation by utilizing very simple logic schemes; for example, a single AND (XNOR) gate performs multiplication in the unipolar (bipolar) SC representation [12]. Moreover, SC also offers high error tolerance against bit-flip errors, making it attractive for safety-critical applications. Since arithmetic operations are performed on several SC sequences, these sequences are usually constrained with zero or low correlation for retaining an acceptable computation accuracy [12]; although some designs exploit the correlation of different sequences for SC [13], [14], they only focus on few arithmetic units. Therefore, they cannot be utilized to implement a system that involves many other arithmetic operations, limiting their application potential; such designs are not considered in this paper.

SC values are analyzed based on probability, so the possible range of values represented by SC is [0, 1] for unipolar

computation or [-1, 1] for bipolar computation. This may cause an accuracy loss in some applications that perform computation over the entire real number range (e.g., NNs); to address this issue, extended stochastic logic (ESL) [15] dividing two SC sequences is often employed for extending the computation range. As the size of ESL is nearly double compared to the standard SC units, a hybrid design combining both is usually employed to reduce the hardware overhead (e.g., such as the hybrid SC-based NN of [16]). In such case, a divider is required for converting ESL to a traditional SC sequence.

The divider is probably the most challenging block among basic SC units. Unlike a multiplier or an adder that is implemented by applying simple combinational logic gates, the design of a conventional divider is usually more complicated and realized using a feedback loop for providing an acceptable computational accuracy [12]. By feeding the temporarily calculated quotient into the input of the loop, an up/down counter in the loop adjusts the number of "1" in the SC sequence in a one-by-one fashion to approach the correct result; therefore, once the loop is stable, the calculated quotient is considered accurate and used as output. However, the computation latency of such conventional divider is extremely slow because the loop requires a significant number of clock cycles (each bit is processed in a clock cycle) to converge; when the SC sequence represents a small value (i.e., most bits in the sequence are "0"), it is difficult to capture the difference between the input and output of the loop, so requiring more bits to stabilize the entire process.

To reduce the computation latency, a binary searching (BS) scheme [12] has been utilized to adjust the intermediate result by progressive precision and determining each bit of the N-bit SC probability associated with the quotient, i.e., identify the possible range of the result in half-by-half fashion (this process is discussed in detail in the next section). This makes the loop to converge faster; this scheme is then improved by utilizing triple modular redundancy (TMR) for further reducing the number of clock cycles, while maintaining the same accuracy [16]. However, such BS-TMR based divider still requires N iterations to calculate the quotient. To address this issue, a decimal searching and TMR (DS-TMR) based divider has been recently designed [17] to adjust the intermediate result with progressive precision by determining each bit of the real value, so in a region-by-region fashion. In this case, only two iterations are required for the loop to be stable with no loss in computation accuracy; however, this divider still requires significantly more than 2^N clock cycles. Moreover, the fast convergence in the DS-TMR based divider requires a larger circuit area; this has limited impact on the entire system if the divider is rarely used, but it is not acceptable if it accounts for a significant part.

In addition to latency, another issue of all current dividers is the limited accuracy; this is a disadvantage of the up/down counter utilized in the loop. This unit adjusts the SC sequence as per the difference between each input and output bits of the loop; therefore, no refined adjustment due to the accumulated difference between several input and output bits is available, and thus the result may not be very accurate. Hence, it is of interest to investigate other feedback loop schemes for a stochastic divider.

Recently, as a widely used mixed-signal circuit, a Delta Sigma Modulator (DSM) has been utilized for SC to generate SC sequences [18], [19]; however, the DSM-based sequences may suffer from high correlation that degrades the computation accuracy. The disadvantages of existing dividers (as previously discussed) and the interest of utilizing a design with structures traditionally used in non-digital circuits (like DSM) have motivated this paper to propose a fast stochastic divider relying on DSM. The main contributions of this paper are as follows:

- A DSM-based negative feedback loop is utilized to design a fast and accurate stochastic divider (which is entirely a digital circuit).
- The proposed stochastic divider requires the smallest number of clock cycles for calculation among all current dividers (e.g., at a reduction between 66.8% and 96.9% when $N = 10$) and thus, having the lowest computation latency;
- The proposed stochastic divider also improves computation accuracy over all stochastic dividers found in the technical literature (e.g., it reduces the average mean square error from $10^{-3.4}$ to $10^{-3.9}$ when $N = 10$);
- An SC-based NN is studied as a first case study to evaluate the advantages of using the proposed divider in emerging applications; results show that for the considered datasets, the SC-based NN with the proposed divider provides the best classification accuracy and PALPC (the product of area, latency, power and number of clock cycles) results.
- In addition to LFSR-based sequences, the divider is also implemented and evaluated by considering Sobol-based sequences; the results show that the proposed design offers the best figure of metric in both computational accuracy and hardware overhead compared to current dividers when using such sequences.

The rest of the paper is organized as follows. In Section II, DSM and SC design are briefly reviewed. Section III presents the proposed DSM-based divider; it also analyzes its convergence and the theoretical number of clock cycles required for performing a calculation. Then, the performance of the proposed design is evaluated and compared with current dividers in Section IV. In Section V, an SC-based NN implemented with different dividers is investigated and assessed as the first case study to evaluate the advantages of the proposed design. To further show the flexibility of the proposed design, its Sobol version is discussed in Section VI as a second case study. Finally, the paper ends in Section VII with the conclusion.

II. Preliminaries

A. Delta Sigma Modulator (DSM)

A Delta Sigma Modulator (DSM) is usually designed as part of a Delta-Sigma Analog to Digital Converter ($\Delta\Sigma$ ADC) for analog/mixed-signal circuits [20] and signal processing [21]. The block diagram of a standard first-order discrete time DSM is shown in Figure 1 (a); it is made of a one-bit Digital to Analog Converter (DAC) (that converts the digital signal back to an analog signal to establish a negative feedback loop), an integrator (that checks the accumulated differences between the original and the converted digital signals), and a one-bit ADC (that is usually implemented by a comparator and

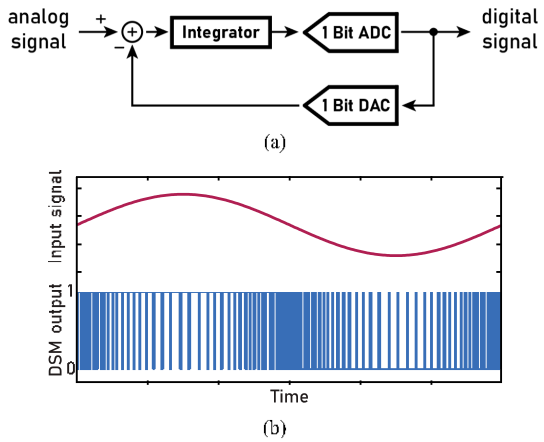

Fig. 1. A first-order DSM: (a) its simplified block diagram; (b) an example of signal conversion.

checks the integrator result with a reference voltage (e.g., 0)). By using such scheme, a DSM correctly converts an input analog signal to a digital bit-stream composed of binary bits (i.e. '1' and '0'); the pulse density is related to the input signal amplitude (Figure 1 (b)), i.e., a higher density of "1"s is generated in the output bit-stream for an input signal with a higher amplitude/value (so similar to the case of SC).

Therefore, DSM has been considered to directly operate as a stochastic number generator (SNG) [18]. However, due to the conversion mechanism of DSM [20], its use as an SNG tends to generate a sequence consisting of repeated segments; moreover, a segment tends to be fixed for a given input/sampled value. So, such type of sequences has two features: i) the "1"s in the entire sequence are not randomly distributed; ii) correlation exists between different sequences. Therefore, when using DSM as an SNG, the generated sequences face the issue of degrading the computation accuracy in subsequent arithmetic computations; moreover, the applicability of such SNG is limited, because it can only be utilized in sensor's interfaces that convert an analog signal (sampled from a sensor) to a stochastic sequence (for subsequent processing by an SC core).

In the next sections of this paper, the structure of DSM is utilized in the design of a stochastic divider; as windowed stochastic sequences keep changing when calculating a quotient sequence, then the issue of using DSM (as an SNG discussed previously) is not encountered in the proposed divider.

B. Stochastic Computing and Current Stochastic Dividers

Prior to introducing SC-related works, the terminology used in this paper is presented for clarification:

- A ***real value*** refers to the value considered in traditional binary computation (represented in the format of integer, floating-point, etc.);
- A ***stochastic sequence*** refers to the bit-stream of "1" and "0";
- The ***probability*** in SC is represented by an N-bit binary sequence and refers to the probability of the occurrence of "1" in the stochastic sequence. SC utilizes such probability to represent a real value; for unipolar computation, the real value associated to a stochastic sequence is equal to the value of its probability p while for bipolar computation, it is equal to $2 \cdot p$-1;

Figure 2 illustrates the SC process; note that bipolar computation is considered in the remaining of this paper, because both positive and negative values are involved in the computation for most applications (like NNs, as first case study in Section V). The probability (e.g., $p(x)$ or $p(y)$) is related to a given real value and is input to a stochastic number generator (SNG) for generating a stochastic sequence (e.g., x or y) prior to all arithmetic computations; in such SNG, a random number generator (RNG) generates an N-bit binary sequence that is compared with the input probability p to generate a stochastic bit ("1" if it is larger than p and "0" otherwise) per clock cycle. After 2^N clock cycles, a stochastic sequence related to the given real value is obtained (in which the probability of the occurrence of "1" is equal to p). Typically, the RNG is implemented using a LFSR, but recent works [22], [23] have shown that low-discrepancy sequences (e.g., Sobol) offer either a higher computation accuracy for some SC circuits (when compared with LFSR-based sequences), or a faster speed at the same accuracy; how-ever, they cannot provide satisfactory accuracy for finite-state machine (FSM)-based units. As introduced in the previous subsection, DSM can also be used to realize an SNG, but the generation of sequences is problematic due to unsatisfactory distributions (as introduced previously) and hence, its limited applicability.

The basic SC arithmetic units mostly include two categories: combinational units (e.g., multiplier, adder, etc) and sequential units (e.g., divider, FSM-based units like the tanh function, etc.); more details of these units are given in [12]. The divider is one of the most complex SC units; it is required for conversion between ESL sequences (that utilize the division of two SC sequences to extend the possible range of bipolar computations from [-1, 1] to $(-2^{N-1}, 2^{N-1})$) and standard SC sequences when implementing a hybrid SC-based NN [16]. In such applications, the divider is in the critical computation path of the NN that determines the time required to complete an inference (i.e., its complexity affects the latency of the entire system), so some designs have been proposed to reduce the computation latency of the divider, because a conventional design requires a significant number of clock cycles to perform this calculation. Next, stochastic dividers (with uncorrelated sequences as considered in this paper) found in the technical literature are briefly reviewed.

1) Conventional Stochastic Divider: The conventional stochastic divider with the input dividend sequence y and the divisor sequence x is also shown in Figure 2; it mainly includes the following logic:

- *AD flip-flop* (DFF) that generates a sequence having the same probability as the input sequence x but it is uncorrelated to x;
- *Three SC multipliers* (i.e., XNOR gates) that perform $p(y) \cdot p(x)$, $p(x) \cdot p(x)$ and $p(x)^2 \cdot p(z)$, where p is the probability of an SC sequence and z is the current calculated division sequence (i.e., $p(z) = p(y)/p(x)$);

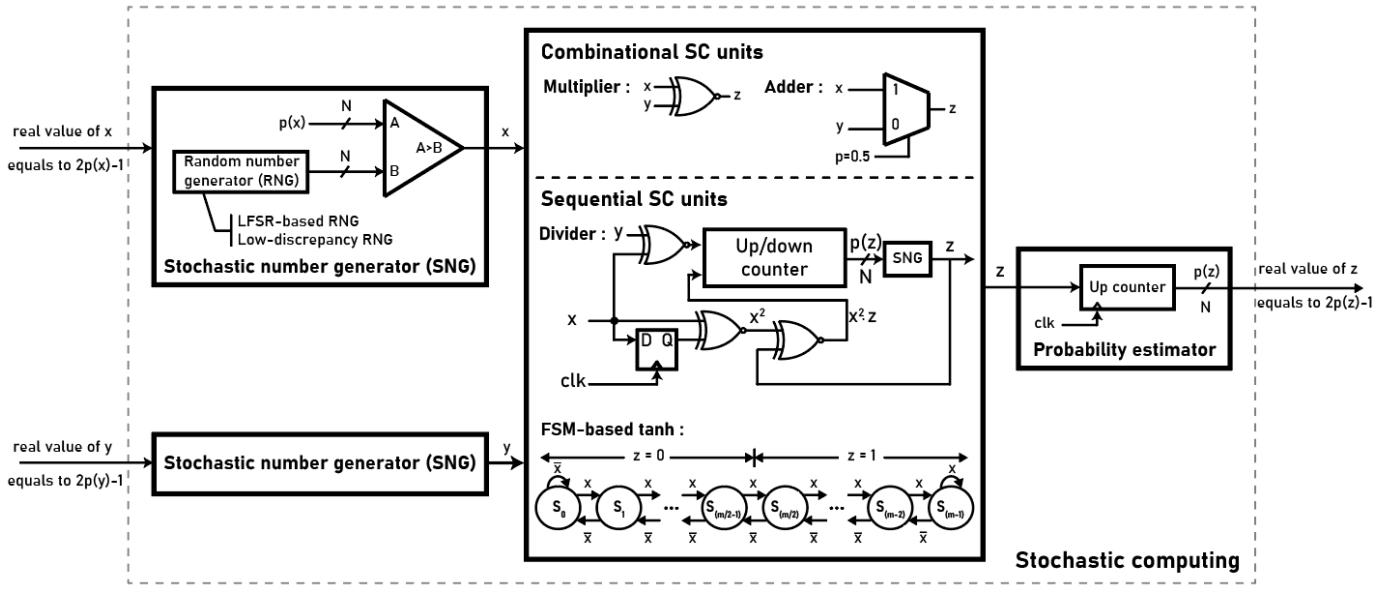

Fig. 2. Illustration of stochastic computing showing its basic arithmetic units.

- *An up/down counter* that initially generates a SC probability p of 0 and increases/decreases it as per the comparison between its input bits for $p(y) \cdot p(x)$ and $p(x)^2 \cdot p(z)$;
- *A SNG* that generates SC bits as per the updated probability of the counter.

The calculated division sequence z is fed back into the divider, so a negative feedback loop is formed, in which $p(x)^2 \cdot p(z)$ keeps track of $p(y) \cdot p(x)$. Hence, once the loop achieves stability, the result of $p(x)^2 \cdot p(z)$ is considered as equal to the one for $p(y) \cdot p(x)$. Therefore, the current $p(z)$ is the final calculated quotient and z consists of the last 2^N bits (with $p(z)$) as division sequence.

However, since the probability related to the quotient is adjusted by increasing/decreasing a "1" in each clock cycle (i.e., in a one-by-one fashion), the stabilization of the conventional stochastic divider requires considerable time (e.g., 46341 clock cycles on average when $N = 10$ [17]), especially for inputs with a small value (i.e., there are many "0" in the sequence, so it is hard for the counter to capture the difference between $p(x)^2 \cdot p(z)$ and $p(y) \cdot p(x)$). Thus, different approaches (as briefly described next) have been presented to improve the convergence process.

2) Binary Searching and Triple Modular Redundancy (BS-TMR)-Based Stochastic Divider: To achieve a reduction in latency, a binary searching (BS) method has been proposed to determine the probability in the up/down counter in a half-by-half fashion [12]. In such case, each bit of the N-bit probability is obtained by utilizing an increase/decrease step. This is 2^{N-2} in the first iteration, which is then scaled down by a factor of two (i.e., 2^{N-3}, 2^{N-4}, …1) in each of the next N-1 iterations. Therefore, partial bits can be used for a decision in the counter, because in each iteration, the goal is to estimate if the probability belongs to one half of the possible range. After this process, the divider works in the conventional fashion (a so-called stabilization phase) and continues to further adjust the estimated results with a step of 1 (i.e., in the conventional fashion) to achieve stability.

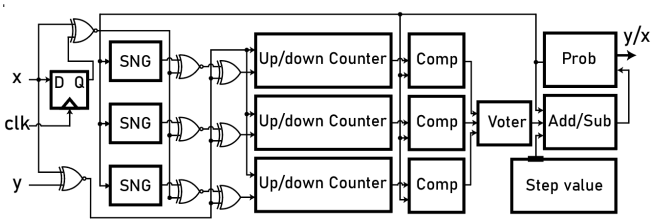

Fig. 3. A BS-TMR based bipolar stochastic divider [16].

However, such method still requires a significant number of cycles to achieve the final result, because the partial bits for the estimate may introduce an accuracy loss due to fluctuation errors (these errors need to be compensated during the operation in the conventional fashion). To address this issue and shown in Figure 3, a triple modular redundancy (TMR) scheme is used to generate three different and independent sequences for each probability; a majority voting among the three paths is then used to generate a correct estimate result. Even though such BS-TMR-based stochastic divider [16] can achieve the same accuracy as the conventional design at a shorter convergence process (e.g., 9214 clock cycles on average when $N = 10$ [16], [17]), it still requires N iterations to perform the calculation, so proportional to the SC resolution.

3) Decimal Searching and Triple Modular Redundancy (DS-TMR)-Based Stochastic Divider: To further decrease the converging time of the stochastic divider, a recently proposed DS-TMR-based design (as shown in Figure 4) overcomes the problem of the BS-TMR-based divider and reduces the required iterations to avoid the relation with N [17]. In such divider, the adjustment step for the up/down counter depends on the SC resolution. For example, for $N = 10$, the real value represented by SC has a resolution of 0.002 (i.e., with three decimal bits); in this case, by setting nine target probabilities (in nine counters) to separate the value range associated with each decimal bit into ten regions, then this divider estimates the number for a decimal bit during an iteration. This searching

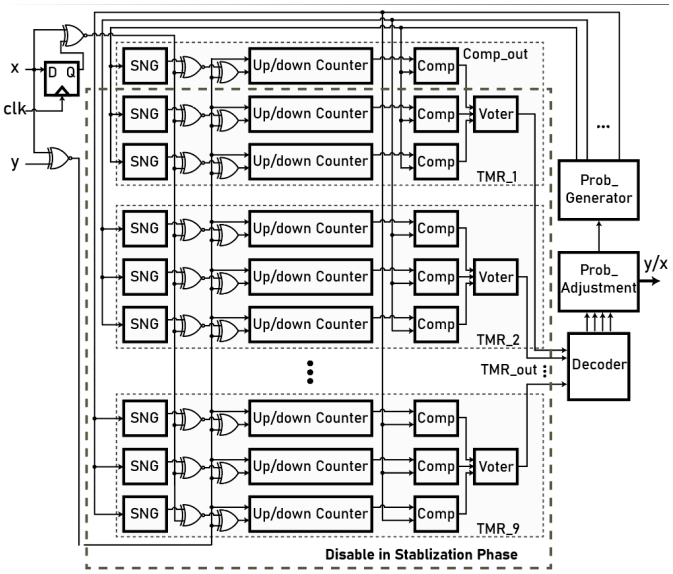

Fig. 4. A DS-TMR based bipolar stochastic divider [17].

approach operates in a region-by-region fashion, because in each iteration a progressive region of the possible range is determined. Note that the divider requires only two iterations to estimate the upper two decimal bits of the quotient; after that, it works in the conventional fashion (i.e., the stabilization phase, which is the same as for the BS-TMR-based divider) to further adjust the calculated result and obtain the last decimal bit. Like the BS-TMR-based divider, a small number of bits are used in each iteration and a TMR-based scheme is employed to deal with fluctuation errors for a satisfactory computation accuracy.

Even though the DS-TMR-based divider significantly reduces the computation latency (e.g., 4300 clock cycles on average when N = 10 [17]) and permits a constant output flow for pipelining at architectural level (it only requires two iterations for computation, so independent of N), the number of clock cycles is still significantly higher than other basic units (like multiplication or addition) that perform a calculation in 2^N cycles. Therefore, when using such a divider in a given implementation involving different arithmetic operations (as in an SC-based NN), the critical computation path of the NN (as well as the total computation energy) is increased. Moreover, the DS-TMR-based divider requires nine TMR blocks, so the hardware area is also significantly larger than that for the conventional and BS-TMR-based dividers.

Therefore, to efficiently implement SC, there is a need for a better stochastic divider with lower area and latency, so overcoming the issues exhibited in current designs.

III. Proposed Stochastic Divider

Next, a novel stochastic divider is presented; it exploits the design principle of DSM, but it is an entirely digital circuit. This divider converges fast and offers a high accuracy.

A. Design of Proposed DSM-Based Stochastic Divider

The proposed divider design is illustrated in Figure 5. Unlike the other bipolar stochastic dividers described in

Algorithm 1 The Calculation Process of Counter in the Feedback Generation Block

```
if z = 1
    Counter = Counter + ABit - DBit;
else
    Counter = WinLen - Counter + ABit - DBit;
end
feedback = Counter;
Dacc = Dacc+ Input - feedback;
```

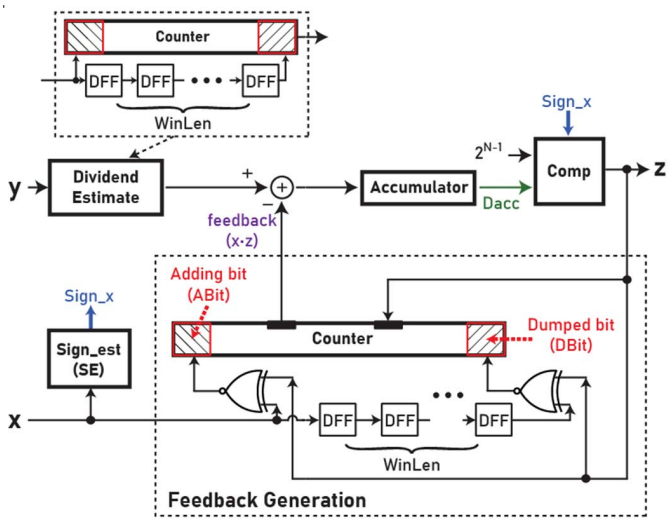

Fig. 5. The proposed DSM-based divider.

section II.B, the proposed bipolar DSM-based divider follows the methodology of designing an unipolar divider; it is realized by forming a negative feedback between $p(y)$ and $p(x)\cdot p(z)$, instead of between $p(y)\cdot p(x)$ and $p(x)^2\cdot p(z)$. This ensures that the loop works correctly when a sequence x is related to a negative dividend value in the bipolar computation [12]. The reason for establishing such feedback is to avoid the so-called dead zone of the DSM [20]; this refers to the scenario when the input amplitude is extremely low, and the DSM tends to incorrectly converge to it. This occurs when the value of $p(y)\cdot p(x)$ is closer to a stochastic probability of 0 than $p(x)$, so such input with a small amplitude is highly likely to make the circuit enter the dead zone (especially when both $p(y)$ and $p(x)$ are close to 0); thus, it severely impacts the performance of the divider. Moreover, stochastic computation for very small values tends to have a limited accuracy.

As per Figure 5, the proposed divider circuit includes the following blocks:

- ***Sign estimate (SE) block***: since the proposed loop is based on a unipolar computation, the sign of the dividend value (i.e., represented by the sequence x) needs to be determined for performing a bipolar computation. This is realized in the SE block that continues to monitor the sequence x in a timing window (this refers to a few continuous sequence bits); the sign of sequence x ($sign\_x$) is estimated as positive if more than half the bits observed in the window is "1", and negative if less than half; for the case in which the same number of "1" and "0" is observed, this SE block operates for one more window

Algorithm 2 The Process of Generating z

```
1:if sign_x is positive
2:    if Dacc >= 2^(N-1)
3:        z= 1;
4:    else
5:        z = 0;
6:    end
7:elseif sign_x is negative
8:    if Dacc >= 2^(N-1)
9:        z= 0;
10:   else
11:       z = 1;
12:   end
13:else
14:   Re-compute sign_x for another WinLen;
15:End
```

to make a decision. Once the sign is determined, this block is disabled. The length of the window *WinLen* to meet a satisfactory computation accuracy is analyzed in section IV.

- ***Dividend estimate block***: this block consists of *WinLen* DFFs and a counter that checks the number of "1" in the divisor sequence y per window (i.e., related to $p(y)$).
- ***A feedback generation block***: this block consists of *WinLen* DFFs that process the sequence x per window, two XNOR gates for multiplying x and z, and a counter that generates the feedback result. This counter checks the number of "1" in the multiplication result of *WinLen* x and z bits; once the *WinLen*+1^{th}x bit arrives, the counter is updated by just processing the ABit (the multiplication of *WinLen*+1^{th}x and *WinLen*+1^{th}z) and subtracting the DBit (the multiplication of the 1^{st}x and *WinLen*+1^{th}z) according to the *sign_x* found by the SE block (this process is given in Algorithm 1).
- ***Subtractor, accumulator and comparator***: the subtractor checks the difference between $p(y)$ (which is related to the result obtained in the dividend estimate block) and $p(x) \cdot p(z)$ (which is related to the result obtained in the feedback generation block). The accumulator (so acting as an integrator of the DSM circuit shown in Figure 1 (a)) accumulates the detected difference in each window and uses the accumulation result as the outcome *Dacc*; this is then compared in a comparator c with 2^N-1 (refers to a stochastic probability of 0) to identify whether $p(y)$ is larger than the estimated $p(x) \cdot p(z)$. For the different cases of *sign_x*, the comparator generates the next z bit (using Algorithm 2).

As discussed above, the negative feedback loop is finally established and as per the accumulated differences, the output sequence is adjusted at each window/clock such that $p(x) \cdot p(z)$ approaches $p(y)$. Once the feedback loop is stable (in 2^N+*WinLen*-1 clock cycles, as proved in the next subsection), the quotient sequence is also obtained.

Note that once the first *WinLen* input bits are fed into the circuit, the window updates at each clock cycle (e.g., the second window consists of the 2^{nd}x bit up to *WinLen*+1^{th}x bit); so, the counter in the feedback generation block operates by only processing the previous counter result with Abit and Dbit (instead of calculating the multiplication of *WinLen* x bits and z bits) starting from the 2nd window; this significantly reduces the computation complexity (and thus, the hardware overhead). During this calculation process, the proposed divider does not require a SNG (which accounts for a considerable hardware overhead); this compensates for the overhead introduced by the additional blocks compared to the conventional divider. Overall, the circuit area of the proposed DSM-based divider is slightly larger than the conventional divider, but significantly smaller than the BS-TMR-based and DS-TMR-based dividers. This will be evaluated in section IV.

Due to the convergence analysis of a DSM in the design of the proposed divider, the estimate calculation result can be compared with the target result in the loop in multiple bits (i.e., *WinLen*) and when adjusted it as per the accumulated differences (instead of only based on the comparison between two single bits each time in the existing dividers), the proposed divider offers a faster convergence process (i.e., a smaller number of clock cycles to perform a calculation) and a higher computation accuracy.

B. Proof of Convergence

Next, the convergence of the proposed DSM-based stochastic divider is proved, and the required number of clock cycles is analyzed.

Define $p(z)_i$ as the stochastic probability of the output sequence z obtained in i bits; the $i+1$ bit of sequence z is given by (refer to Algorithms 1 and 2):

$$z_{i+1} = sgn(((i p(y)_{WinLen\_i} - p(x)_{WinLen\_i} \cdot p(z)_i)) \quad (1)$$

where $p(y)_{WinLen\_i}$ ($p(x)_{WinLen\_i}$) is the estimated probability of the dividend sequence y (divisor sequence x) in the i^{th} window, and the function *sgn* is 0 when its input is negative, and 1 otherwise. Since the window sequence is used to estimate the entire sequence in the proposed design, Eq. (1) can be approximately updated as follows when i is up to n, where $\hat{n}$ denotes the total number of "1" in a n-bit output sequence z:

$$z_{n+1} = sgn\, n \cdot p(y) - \hat{n} \cdot p(x) + n - \hat{n} \cdot p(x) \quad (2)$$

In the bipolar representation for SC, the probability of a n-bit output sequence $p(z)$ is calculated as:

$$p(z)_n = \frac{2\hat{n}}{n} - 1 \quad (3)$$

Therefore, the probability of the output sequence with $n+1$ bits is calculated as per Eqs. (2) and (3): when $p(z)_n < \frac{p(y)}{p(x)}$ (i.e., $\frac{2\hat{n}}{n} - 1 < \frac{p(y)}{p(x)}$), the result of Eq. (2) is equal to 1 (which increases $p(z)_n$ and makes it to approach $\frac{p(y)}{p(x)}$). So, the stochastic probability of the output with $n+1$ bits is given by Eq. (4). In this case, the difference between $p(z)_{n+1}$ and

$p(z)_n$ is calculated by Eq. (5).

$$p(z)_{n+1} = \frac{2(\hat{n}+1)}{n+1} - 1 \tag{4}$$

$$p(z)_{n+1} - p(z)_n = \frac{2(n-\hat{n})}{n(n+1)} \leq \frac{2}{n+1} \tag{5}$$

Otherwise, when $\frac{Y}{X} < Z_n$ and the result of Eq. (2) is equal to 0, Eqs. (4) and (5) must be modified as (6) and (7) respectively.

$$p(z)_{n+1} = \frac{2\hat{n}}{n+1} - 1 \tag{6}$$

$$p(z)_{n+1} - p(z)_n = \frac{2\hat{n}}{n(n+1)} \leq \frac{2}{n+1} \tag{7}$$

Therefore, as the length of the output sequence n increases, both Eqs. (5) and (7) approach 0, i.e., the output result of the divider converges. Moreover, the number of clock cycles can be determined; since the data resolution of an n-bit stochastic sequence is $2/n$ (e.g., 0.002 for an n-bit of 2^{10} sequence), once the result of Eq. (5) or (7) is smaller than $2/n$, the calculation process of the divider is completed. Therefore, the DSM-based divider only needs up to n+*WinLen*-1 clock cycles to calculate the result for n-bit sequences, because the results of Eq. (5) or (7) are always smaller than $2/n$.

IV. Evaluation

In this section, the proposed DSM-based stochastic divider is implemented; its performance is evaluated and compared with the designs discussed previously (including the conventional divider, BS-TMR-based divider, and DS-TMR-based divider).

A. Convergence and Accuracy

Figure 6 shows the convergence and computation accuracy of different dividers by considering the dividend sequence y with $p(y) = 0.8$ and the divisor sequence x with $p(x) = 0.9$ as an example, for the SC resolution with $N = 10$; the simulation results (details are presented in section VI) show that a *WinLen* of 2^7 is sufficient to provide a satisfactory accuracy for the proposed divider, so this value is selected in the evaluation work in this subsection. As shown in this figure, the proposed divider starts outputting the quotient SC sequence after *WinLen* clock cycles; it converges fast and after 2^N+*WinLen*-1 (1151 in this case) clock cycles, the quotient reaches the target value and does not significantly vary anymore. The conventional divider requires the largest number (4096) of clock cycles to generate the quotient sequence; then the BS-TMR-based and the DS-TMR-based dividers follow as requiring 3072 and 1434 clock cycles, respectively. Note that only the conventional divider and the proposed divider generate a stochastic sequence directly, while the other dividers generate a stochastic probability for the calculated quotient. Thus, they require an additional 2^N (i.e., 1024 in this example) clock cycles starting from the stabilization phase to generate the corresponding stochastic sequence when the quotient is used for any other subsequent computation [17].

In the second experiment, the evaluation of the average case with $N = 10$ for 10000 random input pairs is pursued; the mean square error (MSE) is used as a metric to compare the accuracy of the different dividers. Table I shows the average number of clock cycles to perform the calculation and the MSE of the different designs; all current dividers have the same MSE while the proposed divider achieves the highest accuracy with the smallest number of clock cycles (a reduction between 66.8% and 96.9%). This is expected to account for DSM as employed in the proposed divider design and per previous discussion.

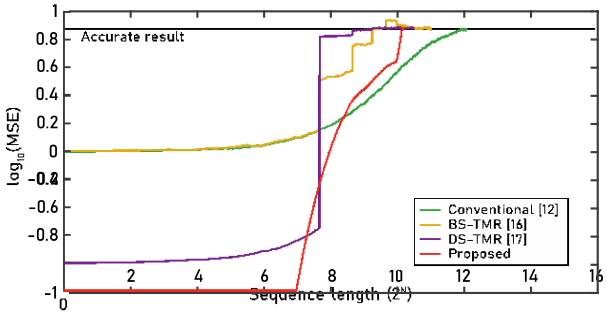

Fig. 6. Quotient estimate by using different stochastic dividers for 0.8/0.9 (the conventional, BS-TMR-based, DS-TMR-based, and the proposed divider require 4096, 3072, 1434, and 1151 bits respectively to obtain the 1024-bit quotient sequence).

TABLE I
Performance Metrics of Different Stochastic Dividers ($N = 10$)

Divider	Average # of clock cycles	MSE
Conventional [12]	46341	$10^{-3.4}$
BS-TMR [16]	9214	$10^{-3.4}$
DS-TMR [17]	4300	$10^{-3.4}$
Proposed	**1428**	$\mathbf{10^{-3.9}}$

Next, the comparison of the MSE results for different dividers when the input dividend and divisor have different value ranges, is pursued. The results for different stochastic dividers are plotted in Figure 7; Figure 7 shows that the MSE is larger when the inputs are near the center of the entire possible range for all dividers. This is expected because the fundamental operation of a divider is based on multiplication, which has a lower computational accuracy for inputs centered in the possible range (as indicated also in [24]). Moreover, the proposed design is also shown to achieve better MSE results than other designs.

In the last experiment, the convergence process and MSE of the proposed DSM-based divider for SC resolution under different values of N (from 8 to 12) with 10000 random inputs pairs is evaluated. The estimated convergence point (ECP) is defined as the point of the MSE curve starting to approach a stable value, so it can be considered as the worst accuracy performance of the divider (i.e., only a slightly better MSE can be achieved after ECP). As shown in Figure 8, the proposed divider calculates the quotient with an ECP slightly larger than 2^N (the theoretical value discussed in section III.B) for all cases; this is due to the distribution of LFSR-based stochastic sequences that cannot be fully uniform (the theoretical analysis is applicable only to the ideal scenario). In all cases, the proposed divider is faster and more accurate compared to all current dividers (the work of [17] shows that the DS-TMR-based divider, with the best computation performance among

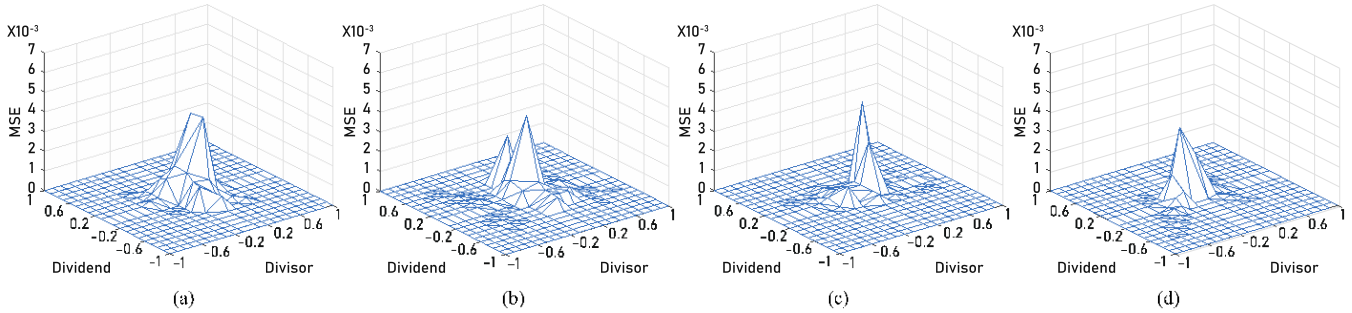

Fig. 7. MSE of different dividers for inputs in different value ranges: (a) the conventional divider; (b) the BS-TMR-based divider; (c) the DS-TMR-based divider; (d) the proposed DSM-based divider.

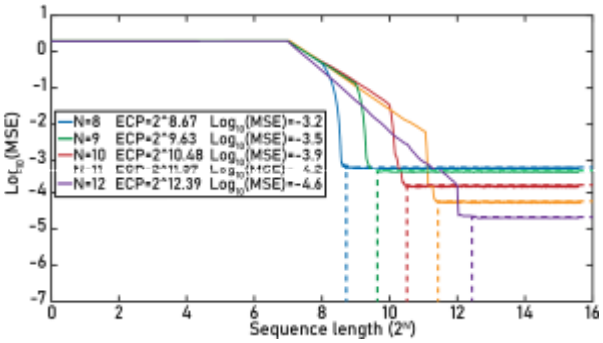

Fig. 8. Convergence of the proposed divider (with *WinLen* = 7) at different values of *N*.

TABLE II

SYNTHESIS RESULTS OF DIFFERENT DIVIDERS

Divider		Area (μm^2)	Latency (ns)	Power (μW)	# clock cycles
FP divider [26]		8248.8	**100.0**	10.0	**50**
SC divider ($N = 10$)	Conventional [12]	**653.6**	10380.3	**1.2**	46341
	BS-TMR [16]	2770.6	1233.7	4.0	9214
	DS-TMR [17]	13900.5	463.4	19.7	4300
	Proposed	1150.7	377.6	1.8	1428

all these three dividers, cannot achieve a MSE of 10^{-3} until $N = 9$ with more than 2^{11}clock cycles, and an MSE of $10^{-4.1}$ until $N = 12$ with more than 2^{14} clock cycles).

B. Hardware Overhead

Different stochastic dividers with $N = 10$ are implemented in HDL; the designs are synthesized using the Synopsys Design Compiler by mapping to an ASAP 7 nm library [25] for evaluating and comparing the hardware overhead. Note that the proposed design is fully synthesized, because it is an entirely digital circuit. The clock frequency of the circuits is set to 200 MHz as an example, but higher frequencies can also be selected (this has no impact on the qualitative comparison results).

Table II shows the synthesis results for different stochastic dividers in terms of circuit area, computation latency and number of clock cycles for performing an entire calculation, and power dissipation. For comparison purposes, a Newton-Raphson based single-precision floating-point (FP) divider [26] (as one of the most used traditional binary/non-SC dividers) is also implemented and evaluated.

As per Table II, the conventional stochastic divider requires the smallest area due to its simple circuity, but it has the largest latency because it requires a significant number of clock cycles to perform the computation. Among all existing stochastic divider designs that strive to improve the computation latency evaluated in Table II, the proposed divider offers the smallest latency and number of clock cycles (with a reduction of 96.4% in latency and 96.9% in clocks compared to the conventional design). Next there is the DS-TMR-based divider (with a reduction of 95.5% in latency and 90.7% in clocks compared to the conventional design) and then the BS-TMR-based divider (with a reduction of 88.1% in latency and 80.1% in clocks compared to the conventional design). The proposed divider also requires the smallest hardware area and power dissipation compared to the BS-TMR-based and DS-TMR-based dividers (with a reduction of 58.5% and 91.7% in area and 55% and 90.9% in power, respectively).

Compared to the FP divider, the proposed design achieves a reduction of 86.1% in area and 82.0% in power dissipation; this confirms the advantage of SC in terms of hardware overhead. Even though it requires a longer computation latency and larger number of clock cycles (which is an inherent issue of SC), the proposed design also significantly reduces the difference between the FP divider and existing stochastic dividers (as shown in Table II) under these metrics.

Overall, the proposed stochastic divider is superior to all SC designs found in the technical literature because it requires the lowest hardware overhead in a comprehensive manner, while providing the highest computation accuracy due to its DSM-based scheme. These advantages make the proposed divider very attractive for SC based high-performance implementation in hardware-constrained applications. Moreover, this novel design addresses computation accuracy and latency (compared to the conventional computing scheme) when division is required in an SC implementation. The implications of using the proposed as well as other dividers in an SC-based NN as a first case study, are presented next.

V. FIRST CASE STUDY: SC-BASED NEURAL NETWORKS

As discussed previously, the advantages in terms of error-tolerance and hardware computational complexity make stochastic computing (SC) very attractive when implementing systems that require many arithmetic operations such as Neural Networks (NNs) [5]. By introducing a limited computational accuracy loss (in many cases no loss), a SC implementation has been shown to significantly reduce hardware overhead for different types of NNs. As one of the most widely used type of

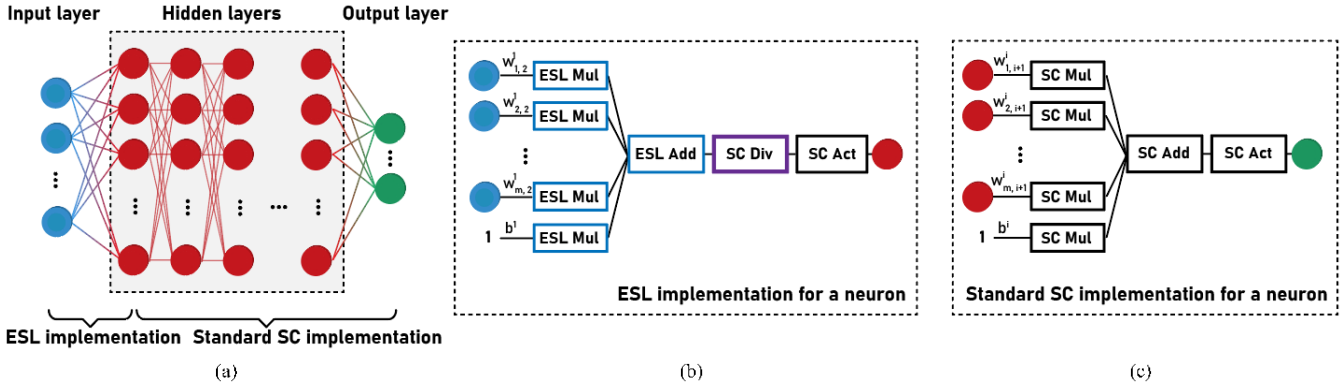

Fig. 9. A Multi-Layer Perceptron (MLP): (a) the network; (b) one neuron computation with ESL implementation (for the first hidden layer in a hybrid SC-based MLP design); (c) one neuron computation with standard SC implementation (for the 2nd to the last hidden layers and the output layer in a hybrid SC-based MLP design).

NNs, the Multi-Layer Perceptron (MLP) offers high flexibility when learning the mapping from inputs to outputs, so it is capable of dealing with nearly all types of data. In addition, an MLP can also be combined with other types of NNs to provide significant performance improvements in ML [27]. Therefore, the MLP is considered in this paper as case study, and an existing SC-MLP design [16] is utilized to evaluate the advantages of the proposed divider.

The block diagram of an MLP is shown in Figure 9 (a); it consists of one input layer, one or more hidden layers and one output layer, among which each two neighbor layers are fully connected. As per Eq. (8), to obtain the value of the n^{th} neuron in the $i+1^{th}$ layer (that is a hidden or output layer), every neuron (m in total) in its previous layer (i.e., the i^{th} layer) is assigned with a specific weight w and its multiply-accumulate result is combined with an assigned bias value b of the i^{th} layer. This is then activated by a function δ; for calculating the output neuron (s), the activation result is utilized to provide a learning result (e.g., a predicted class if the NN performs a classification task).

$$Neuron_{n,i+1} = \delta\left(\sum_{j=1}^{m} w_{j,n}^{i} \cdot Neuron_{j,i} + b^{i}\right) \quad (8)$$

Recently, an efficient hybrid SC design has been proposed for implementing MLPs [16]. To incur a limited accuracy loss in the computation result of an SC-based MLP, extended stochastic logic (ESL, implementation details are provided in [16]) is employed in the mapping between the input layer and the first hidden layer to extend the computational range at an acceptable hardware overhead. Therefore, as shown in Figure 9 (b), a stochastic divider is required for converting the ESL sequences back to standard SC sequences for computation in subsequent layers (Figure 9 (c)). In such scenario, the divider accounts for a small fraction of the entire network if the network is large (because it is only used for one layer); however, it is in the critical computation path, so the latency of the divider impacts the latency and power dissipation of the NN.

Next, the inference process of three pre-trained MLPs (associated to three widely-used datasets [28]–[30] given in Table III) is implemented based on the hybrid SC design (with $N = 10$) by utilizing different dividers. The classification accuracy and hardware overhead of the SC-based MLP is evaluated and compared next.

TABLE III
MLP FEATURES FOR DIFFERENT DATASETS (TEN CLASSES)

Dataset	# layers	# neurons per layer	Activiation function		Classification accuracy
			Hidden layer	Output Layer	
MNIST	4	784, 256, 128, 10	Clamped Relu	Tanh	98.3%
Fashion-MNIST	5	784, 256, 128, 128, 10	Clamped Relu	Tanh	88.9%
SVHN	5	704, 512, 512, 512, 10	Clamped Relu	Tanh	85.5%

TABLE IV
CLASSIFICATION ACCURACY OF DIFFERENT SC-BASED MLP IMPLEMENTATIONS

SC-based MLP with different dividers	MNIST	Fashion-MNIST	SVHN
Conventional [12]	97.0%	87.9%	83.8%
BS-TMR [16]	97.0%	87.9%	83.8%
DS-TMR [17]	97.0%	87.9%	83.8%
Proposed	**97.8%**	**88.5%**	**85.0%**

A. Classification Accuracy

Table IV reports the classification accuracy of SC-based MLPs with different dividers; results show that the MLPs with the conventional divider, BS-TMR-based divider and DS-TMR-based divider achieve the same classification accuracy in all cases, because these three types of dividers have the same computation accuracy (Table I). As the proposed divider has a higher accuracy, then the MLP with such divider achieves the best accuracy (an increase in a range of 0.6% to 1.2% for the considered networks); it also reduces the accuracy loss between the FP version (in Table III) and the SC version (e.g., from 1.3% down to 0.5%). The SC circuits are used to implement the NNs that in this case study they are pre-trained by FP numbers in Matlab; the accuracy of all SC-based MLPs can be further improved by using SC to also implement the training process. This is not further considered in this paper, because we target the effectiveness of the proposed divider compared to other stochastic dividers found in the technical literature.

TABLE V
SYNTHESIS RESULTS OF DIFFERENT MLP IMPLEMENTATIONS

MLP			Area		Latency		Power		# Clock Cycles	PALPC (%)
			mm^2	%	*ns*	%	*mW*	%		
MNIST	FP-based MLP		36.8	100	6363.7	100	1325.2	100	3376	100
	SC-based MLP with different dividers	**Conventional [12]**	**15.7**	**42.7**	34269.1	538.5	**207.4**	**15.7**	46341	493.6
		BS-TMR [16]	16.1	43.8	6134.9	96.4	208.1	15.7	9214	18.1
		DS-TMR [17]	18.8	51.1	2851.5	44.8	212.1	16.0	4300	4.7
		Proposed	15.8	42.9	**1452.1**	**22.8**	207.6	15.7	**1428**	**0.6**
Fashion-MNIST	FP-based MLP		36.8	100	7343.9	100	1923.6	100	3896	100
	SC-based MLP with different dividers	**Conventional [12]**	**16.6**	**45.1**	43934.5	598.2	**218.2**	**11.3**	46341	364.1
		BS-TMR [16]	17.1	46.5	6205.8	84.5	218.9	11.4	9214	10.6
		DS-TMR [17]	19.9	54.1	2923.1	39.8	222.9	11.6	4300	2.8
		Proposed	16.7	45.5	**1523.8**	**20.7**	218.4	11.4	**1428**	**0.4**
SVHN	FP-based MLP		125.4	100	8045.1	100	6421.2	100	4208	100
	SC-based MLP with different dividers	**Conventional [12]**	**54.3**	**43.3**	31605.5	392.6	**692.7**	**10.8**	46341	202.1
		BS-TMR [16]	55.3	44.1	5799.9	72.1	694.1	10.8	9214	7.5
		DS-TMR [17]	61.0	48.6	2824.7	35.1	702.2	10.9	4300	1.9
		Proposed	54.5	43.5	**1605.0**	**20.0**	693.0	10.8	**1428**	**0.3**

B. Hardware Overhead

Next, the SC-based MLPs are implemented in HDL and synthesized (using the same method as for all dividers and at a 200 MHz clock frequency) for evaluating the hardware overhead. Table V shows the circuit overhead, the number of clock cycles (which depends on the divider because it is the most complex unit on the critical computation path of the MLP) and a system-level combined metric PALPC (the product of area, latency, power and number of clock cycles) for each SC-based MLP design, as well as those for the traditional FP version for comparison purposes (these results are from [17]). As per Table V, all SC-based MLPs are shown to be superior to the FP version in each evaluation metric except for the SC version with the conventional divider for the computation latency, and thus the PALPC (because this divider's calculation is rather time consuming). Then among all SC-based MLPs, the NN with the proposed divider achieves the best result in latency, power and number of clock cycles, by incurring in an extremely small additional area (up to 0.6%) and power per cycle (up to 0.1%) compared to the MLP with a conventional stochastic divider that has the best results in these two metrics. Hence it has the lowest PALPC for the entire inference; it is only up to 0.6% of the FP version, 0.1% of the SC version with the conventional divider, 4.0% of the SC version with the BS-TMR-based divider, and 15.8% of the SC version with the DS-TMR-based divider.

Overall, the proposed divider permits an SC-MLPs to have the best accuracy and hardware efficiency compared to existing designs. Note that even though the proposed stochastic divider incurs in larger area and power dissipation than the conventional stochastic divider (as evaluated in section IV.B), its impact on the entire area/power of SC-based MLPs is rather negligible as discussed above. Therefore, the increased area/power does not limit the applicability of the proposed design for such implementation in area/power-constrained platforms; conversely, it makes the SC implementation more attractive for power-constrained designs due to the significantly reduced number of clock cycles (substantially saving energy consumption for performing a classification task, as an example).

VI. SECOND CASE STUDY: SOBOL-BASED STOCHASTIC DIVIDER

The proposed stochastic divider is flexible to be utilized for other types of SC sequences such as the Sobol-based sequence; even though this type of sequence offers better performance for some SC units, it cannot provide a satisfactory accuracy for some FSM-based units (e.g., a *tanh* function) [23]), and thus, it is not acceptable in SC implementation for NN). Therefore, only the stochastic divider itself using Sobol-based sequences is discussed in this section to show the effectiveness of the proposed design for other potential applications in which a Sobol-based SC implementation is attractive.

Different from an LFSR-based sequence, the Sobol-based sequence has a unique nature due to its recursive generation method [22], in which each pseudo-random number is obtained as per the previous number. This makes the distribution of Sobol-based random numbers more uniform than the LFSR version. Therefore, Sobol-based sequences require a shorter length to achieve a similar accuracy as the LFSR version; as indicated in [22], [23], a higher computation accuracy can be finally offered when implementing some basic SC units (e.g., the adder, multiplier, and divider). However, as shown in Figure 10 (a), the Sobol-based stochastic generation gate (SNG) is more complex than the LFSR version; it consists of an address generator (used to detect the least significant zeros), a storage array (that stores many intermediate variables for Sobol sequence generation), an XOR gate and D flip-flop (for recursively generating quasi-random numbers), and a comparator (for SC sequence generation). Therefore, such SNG incurs in a larger hardware overhead.

The computation accuracy of the proposed divider using both types of sequences (LFSR and Sobol) is compared when selecting different lengths for the window. Consider Figure 11, the MSE of the Sobol version performs better due to the better distribution property of Sobol-based sequences, especially when the window length is smaller. Figure 12 shows the convergence process and the MSE results of the proposed divider (with *WinLen* of 7) using Sobol-based sequences with a different value of *N*. Compared to the LFSR version (Figure 8), a higher accuracy (i.e., smaller MSE) is achieved in all cases as expected, and a faster convergence ECP (a more

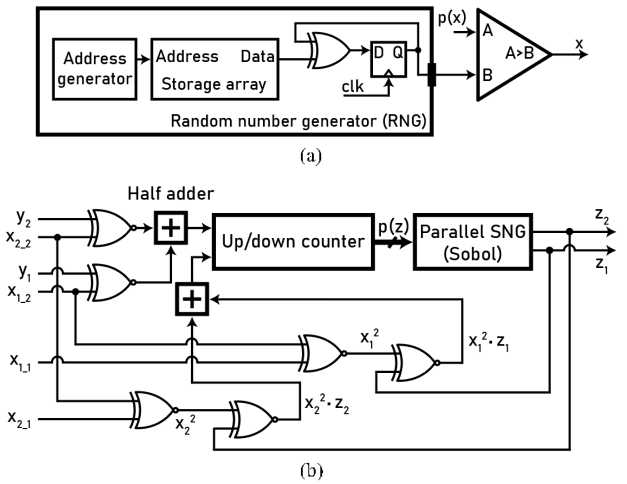

Fig. 10. Sobol-based stochastic computing: (a) a stochastic generator gate; (b) a stochastic divider of [22] with 2x parallelization.

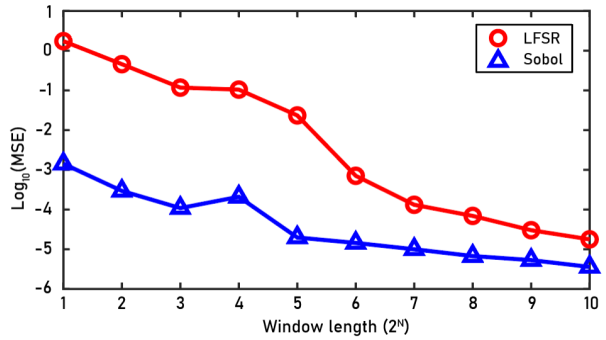

Fig. 11. MSE at different values of window length when N=10.

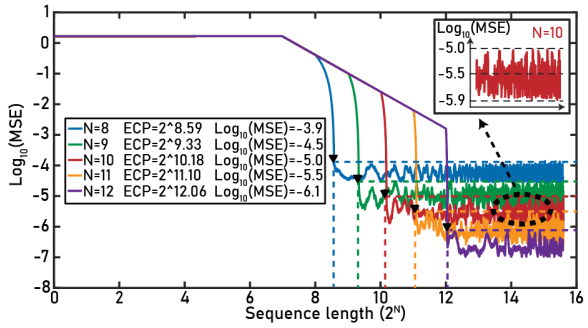

Fig. 12. Convergence of the proposed divider (using Sobol-based sequence, with *Winlen* of 7) at different values of *N*.

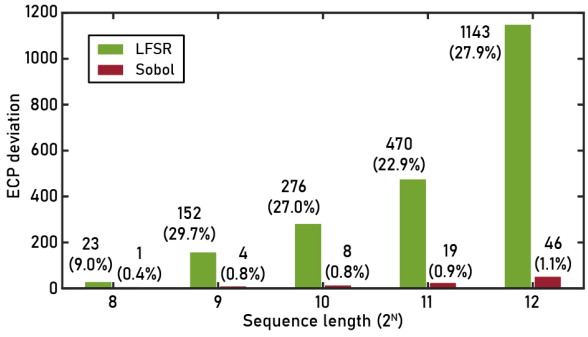

Fig. 13. ECP deviation of the proposed divider from the theoretical value when using different types of stochastic sequence.

detailed comparison is given in Figure 13) that better fits the theoretical analysis provided in section III.B, is obtained.

TABLE VI
COMPARISON OF STOCHASTIC DIVIDERS WHEN USING SOBOL-BASED SEQUENCE(N=10)

Divider	MSE	Hardware overheads				
		Area (μm^2)	Latency (ns)	Power (μW)	# clock cycles	PALPC (%)
Conventional [12]	$10^{-5.4}$	1210.6	7286.1	2.7	23171	100
Parallel scheme [23]	$10^{-5.4}$	1566.3	4194.1	3.5	11586	48.3
Proposed	$\mathbf{10^{-5.9}}$	**1150.7**	**304.4**	**1.8**	**1159**	**0.1**

Since the Sobol-based sequence generation is inherently parallelizable, a parallel stochastic divider that maintains the same accuracy but with a faster convergence process compared to the non-parallelization scheme, has been designed in [22]. As shown in Figure 10 (b), a 2x parallelization configuration additionally requires a parallel SNG (instead of a standard one) for generating two parallel sequences, two half adders for encoding these sequences and three extra XNOR gates for performing multiplication on different sequences, compared to the standard divider design (as shown in Figure 2).

Next, the performance of different dividers using Sobol-based sequences (with N = 10), including the conventional scheme (i.e., with non-parallelization), the parallel scheme (with 2x parallelization) of [23], and the proposed DSM-based design, are evaluated and compared in terms of MSE and hardware overhead. The results are given in Table VI; compared to the conventional design, the parallel scheme keeps the same accuracy, while reducing the number of clock cycles by half at the cost of a very small additional area and power. The proposed divider achieves the best value in each metric; this again is due the DSM-based scheme that enables a faster convergence and a higher accuracy as discussed previously. Note that all dividers using Sobol-based sequences offer better MSE results than LFSR-based sequences due to their better distribution property of the sequences; therefore, such sequences seem to have good potential for implementing high-accuracy SC systems that do not require any FSM-based units.

In addition to LFSR/Sobol-based stochastic sequences, there are also few other types of sequences like the deterministic sequences [31], [32]. Such sequences are generated by deterministically arranging the position of "1"s following a specific rule; then, a completely accurate result can be achieved when using these sequences to perform arithmetic operations such as multiplication or addition. Hence, deterministic sequences are expected to offer a better MSE than LFSR/Sobol-based sequences for each divider design, because the fundamental operation of a divider is based on multiplication as introduced previously. However, such approach requires long sequences to achieve a complete accuracy and thus, a larger latency and energy consumption for performing each arithmetic computation. Therefore, the deterministic SC approach tends to be more attractive for implementing a system that does not have error resilience and requires an accurate result; for a system with high performance and that can tolerate small errors (like ML inference using NNs), its use needs to be further investigated.

VII. CONCLUSION

In this paper, a fast and accurate stochastic divider has been proposed. By employing a DSM-based negative feedback loop to design a novel scheme, the proposed divider requires

the smallest number of clock cycles to perform a division calculation compared to all stochastic dividers found in the technical literature; the convergence and theoretical number of clock cycles required by the proposed design has been analyzed. Moreover, since the DSM-based loop enables a more accurate adjustment on an intermediate result during the computation process, the proposed divider also offers a higher computational accuracy.

The DSM-based divider has been then firstly implemented and evaluated by comparison with LFSR-based sequences (as the widely used sequence type); the results (e.g., for $N = 10$) have shown that compared to an MSE of $10^{-3.4}$ achieved by existing dividers, the proposed divider offers a better result of $10^{-3.9}$ while requiring the smallest number of clock cycles (i.e., a reduction in a range of 66.8% to 96.9%). Hardware synthesis results have shown that even though the proposed divider requires additional area and power compared to the conventional design (which has the smallest circuit size but is considerably slower), it is significantly faster and also better than the other existing dividers in all figures of merit. An SC-based neural network has also been presented as a case study to evaluate the advantages of the proposed divider in an ML application. The evaluation results have shown that the network with the proposed divider is superior to all other versions (utilizing with current dividers) in both classification accuracy and hardware overhead. To show the flexibility of the proposed design, other types of stochastic sequences like Sobol have also been considered; it also provides the best performance by all figures of merit considered compared to existing Sobol-based dividers.

This paper has exploited analog and/or mixed-signal circuit structures and designs to realize a purely digital SC unit; this would likely trigger further research in SC also along this direction.

REFERENCES

[1] A. Subasi, *Practical Machine Learning for Data Analysis Using Python*. New York, NY, USA: Academic, 2020.

[2] 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.

[3] B. Li, M. H. Najafi, B. Yuan, and D. J. Lilja, "Quantized neural networks with new stochastic multipliers," in *Proc. 19th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2018, pp. 1–7.

[4] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA, USA: Springer, 1969, pp. 37–172.

[5] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 2809–2824, Aug. 2020.

[6] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[7] M. H. Najafi and M. E. Salehi, "A fast fault-tolerant architecture for Sauvola local image thresholding algorithm using stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 808–812, Feb. 2016.

[8] N. Onizawa, S. Koshita, and T. Hanyu, "Scaled IIR filter based on stochastic computation," in *Proc. IEEE 58th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2015, pp. 1–4.

[9] H. Ichihara, T. Sugino, S. Ishii, T. Iwagaki, and T. Inoue, "Compact and accurate digital filters based on stochastic computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 31–43, Jan. 2019.

[10] B. Yuan and K. K. Parhi, "Belief propagation decoding of polar codes using stochastic computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 157–160.

[11] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, "Sotchastic decoding of LDPC codes over GF(q)," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 939–950, Mar. 2013.

[12] 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.

[13] 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.

[14] S. Yu, Y. Liu, and S. X.-D. Tan, "Approximate divider design based on counting-based stochastic computing division," in *Proc. ACM/IEEE 3rd Workshop Mach. Learn. CAD (MLCAD)*, Aug. 2021, pp. 1–6.

[15] 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.

[16] 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.

[17] S. Liu *et al.*, "Stochastic dividers for low latency neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 10, pp. 4102–4115, Oct. 2021.

[18] P. Gonzalez-Guerrero, X. Guo, and M. Stan, "SC-SD: Towards low power stochastic computing using sigma delta streams," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Nov. 2018, pp. 1–8.

[19] P. Gonzalez-Guerrero, S. G. Wilson, and M. R. Stan, "Error-latency trade-off for asynchronous stochastic computing with $6\,1$ streams for the IoT," in *Proc. 32nd IEEE Int. Syst. Chip Conf. (SOCC)*, Sep. 2019, pp. 97–102.

[20] R. Schreier and G. C. Temes, *Understanding Delta Sigma Data Converters*. Piscataway, NJ, USA: IEEE Press, 2005.

[21] X. Tang, Q. Hu, and W. Tang, "Delta-sigma encoder for low-power wireless bio-sensors using ultrawideband impulse radio," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 7, pp. 747–751, Jul. 2017.

[22] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 650–653.

[23] S. T. 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. 1226–1339, 2018.

[24] T. J. Baker and J. P. Hayes, "The hypergeometric distribution as a more accurate model for stochastic computing," in *Proc. 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. Li, *Computer Principles and Design in Verilog HDL: Floating-Point Algorithms and FPU Design in Verilog HDL*. Hoboken, NJ, USA: Wiley, 2015.

[27] J. Brownlee, *When to Use MLP, CNN, and RNN Neural Networks*. Montpelier, VT, Australia: Machine Learning Mastery, 2018. [Online]. Available: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

[28] 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.

[29] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

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

[31] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–8.

[32] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.