High-Accuracy Multiply-Accumulate (MAC) Technique for Unary Stochastic Computing

Peter Schober, M. Hassan Najafi, *Member, IEEE* and Nima TaheriNejad, *Member, IEEE*

***Abstract*—Multiply-accumulate (MAC) operations are common in data processing and machine learning but costly in terms of hardware usage. Stochastic Computing (SC) is a promising approach for low-cost hardware design of complex arithmetic operations such as multiplication. Computing with deterministic unary bit-streams (defined as bit-streams with all 1s grouped together at the beginning or end of a bit-stream) has been recently suggested to improve the accuracy of SC. Conventionally, SC designs use multiplexer (MUX) units or OR gates to accumulate data in the stochastic domain. MUX-based addition suffers from scaling of data and OR-based addition from inaccuracy. This work proposes a novel technique for MAC operation on unary bit-streamsthat allows exact, non-scaled addition of multiplication results. By introducing a relative delay between the products, we control correlation between bit-streams and eliminate OR-based addition error. We evaluate the accuracy of the proposed technique compared to the state-of-the-art MAC designs. After quantization, the proposed technique demonstrates at least 37% and up to 100% decrease of the mean absolute error for uniformly distributed random input values, compared to traditional OR-based MAC designs. Further, we demonstrate that the proposed technique is practical and evaluate area, power and energy of three possible implementations.**

I. Introduction

STOCHASTIC computing (SC) [1]–[3] is an unconventional computing paradigm providing low-cost and noise-tolerant design for complex arithmetic functions such as multiplication. In contrast to common positional binary representation, in SC, data is represented using non-positional uniform bit-streams. The bit-streams can be random with interleaved bits of 0s and 1s or predictable (deterministic) with uniform *unary* bit-streams having first all 1s and then all 0s (or vice versa) [4]–[6].

Stochastic Computing (SC) can be realized in both digital and analog domain. In digital domain, the binary to bit-stream conversion is often performed using a stochastic number generator (SNG) unit built from a random number generator (RNG) (or a counter for the unary case) and a comparator [5]. Alternatively, in the analog domain where the input is given in analog voltage or current format, an analog-to-time converter such as a pulse-width modulator can be used to convert the data into a time-encoded stochastic number [7]. The important factor in generating stochastic numbers is the ratio of the number of 1s to the length of bit-stream, or the fraction of the time that the signal is high (i.e., logic-1). For example, if a signal is high 20% of the time, or equivalently, if 20% of the bits in a bit-stream are 1, the signal/bit-stream represents 0.20 in the so-called *unipolar* representation [3]. In the unipolar format, the probability of observing a 1 in the bit-stream is equal to the represented value[1]. Unless otherwise stated, the bit-streams discussed in this paper are in the unipolar format. The outputs of stochastic operations are again two-level signals, which can be used as the input(s) to other stochastic circuits or converted back to positional binary representation for further processing using conventional binary designs or storing in memory.

Peter Schober and Nima TaheriNejad are with the Institute of Computer Technology (ICT), Technische Universität Wien (TU Wien), Vienna, Austria. M. Hassan Najafi is with the School of Computing and Informatics, University of Louisiana, LA, 70504, USA. Email: peter-schober@gmx.at, najafi@louisiana.edu, nima.taherinejad@tuwien.ac.at

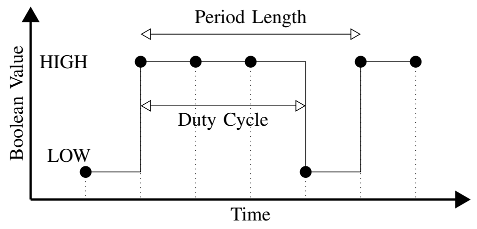


Fig. 1: Encoding the value 0.75 into the duty-cycle of a time-encoded pulse-width-modulation (PWM) signal. SC works with time-continuous PWM signals as well as with their time-discrete represented which we call periodic unary bit-streams.

While (pseudo) random bit-streams have been the common form of representing data in SC [2], [3], unary bit-streams recently attracted attention due to their efficient and low-cost generation, and their potential for deterministic and accurate computation using SC logic [6]–[11]. For example, 1100, 0011, and 1111000 are all examples of unary bit-streams representing 0.5. Unary bit-streams in the digital domain are interpreted as PWM signals in the analog domain [7]. A PWM signal is defined by a duty cycle (D) and a frequency (or period where frequency=1/period). The duty cycle is the fraction of time in which the signal is high. Hence, the duty cycle determines the represented value. Fig. 1 shows a PWM signal with $D = \frac{3}{4}$, which can also be sampled as a discrete unary bit-stream and represented by 0111. Continuous PWM signals can work with significantly higher speed [7], but are more susceptible to environmental conditions and noise compared to discrete bit-streams. While processing of discrete bit-streams is also limited by quantization noise, digital bit-streams are easier to buffer and process compared to continuous signals.

Accumulation (addition) is an essential operation for many computing systems. In unipolar SC, numbers are limited to the [0,1] interval [2]. Hence, scaled addition, instead of normal addition, is natural as the maximum output from normal addition will be above the upper bound. A multiplexer (MUX) implements scaled addition in SC, when *correlated*

[1]A stochastic value is said to be unipolar, if $x = M\rho_x$, where x denotes the represented value, ρ denotes the probability of observing high (logic 1), and M is a positive scaling factor. In this paper, we assume $M = 1$.

(or *uncorrelated*) bit-streams are connected to the main and an *uncorrelated* bit-stream, representing 0.5, is connected to the select input [5], [7]. In this paper, we use the stochastic cross correlation (SCC) [12] as a measure for correlation. Two bit-streams are called positively correlated ($SCC = +1$) when they have maximum overlap between 1s, and negatively correlated ($SCC = -1$) when they have minimum overlap between 1s. Further, the term uncorrelated ($SCC = 0$) is used interchangeable with *independent*. The correlation of bit-streams can be controlled during generation of them or manipulated [13] when receiving from other stochastic circuits. It is also possible to generate approximately uncorrelated or correlated bit-streams from existing bit-streams using additional circuitry. For example, the uncorrelated select input of the MUX can be generated by using an additional XOR gate and a (toggle) flipflop [14], [15].

The processing time increases exponentially with the bit-stream lengths as the output length equals the product of the periods of the input bit-streams [5]. When the input values are small and the result stays in the representable range, i.e., in the [0,1] interval, non-scaled addition is preferred. As an alternative to the MUX-based scaled addition, OR gate has been suggested for fast and non-scaled addition of data [16]. A requirement for OR-based addition, however, is that the input bit-streams must be negatively correlated to produce accurate output. If a bit in a bit-stream is 1, the same bit position in the other bit-stream(s) must be 0. Any overlap between 1s results in inaccuracy in the OR-based addition.

Besides the traditional summation methods using MUX unit and OR gate, several modifications and alternatives have been proposed in the literature. One approach for non-scaled addition combines an OR-based adder with additional circuitry that includes a shift-register [17]. When both inputs of the OR gate are 1, the circuit forwards logic-1 to the output and stores a 1 bit in the shift register. When both inputs are 0, a previously stored 1 bit is added to the inputs. Assuming that the shift-register is large enough to store enough 1s and the sum is in the valid interval, this method also allows exact summation. If the sum should be converted back to a binary number, an accumulative parallel counter (APC) can be used to implement the exact non-scaled summation and the conversion to binary in a single circuit. APCs function as multi-input stochastic-to-binary converters that increase an internal counter for each 1 at their inputs [18]. Finally, scaled summation can be mapped to mealy finite state machines [15]. The number of inputs is equal to the number of states and the current state in conjunction with the current input determine the output. Despite the fact that in SC, multiplication can be accurate and efficient, it has been shown in [19] that multiply-accumulate (MAC) circuits can be implemented without conventional stochastic multiplication and addition. In [19], the authors use counters to compute multiplication as well as accumulation and the inputs and outputs are in conventional binary format.

In this work, we propose a novel summation theory that builds on the already existing work on exact multiplication operation (with AND gates) using deterministic unary bit-streams [5], [7], [8]. Combined, this work performs fast, efficient and accurate MAC operation that can be used in many applications. We propose an OR-based MAC unit that accurately accumulates the output of multiplication operations performed on unary bit-streams.

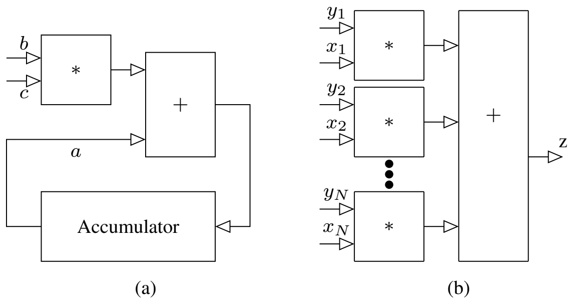


Fig. 2: (a) MAC architecture with accumulator a and factors b and c for sequential data. (b) Parallel MAC architecture with factors x_i, y_i and result z.

The rest of this paper is organized as follows. Section II provides a brief overview of unary SC and MAC operation. In Section III, we present our claims for the proposed theorem as well as the theoretical limits on summation of multiplication results for our technique. In Sections IV and V, mathematical proofs are derived for the proposed theory and its upper error bound, respectively. Section VI presents experimental results of the proposed technique. In this section, we also provide accuracy comparisons with the state-of-the-art MAC designs and evaluate gray-scaling as a practical case study. In Section VII, we provide an analysis of the resource consumption between different MAC implementations. We further discuss constraints and latency of our technique in Section VIII. Finally, we draw conclusions in Section IX.

II. Basics of MAC Operation using Time-Encoded Stochastic Computing

The MAC operation is defined as

$$a \leftarrow a + (b \cdot c). \tag{1}$$

with a as accumulator, and b and c as factors. A block diagram of Equation (1) is illustrated in Fig. 2(a). It is worth pointing out that using multiple multipliers in parallel combined with a multi-input adder produces the same output value z. That is,

$$z = \sum_{i=1}^{N} x_i y_i \tag{2}$$

with inputs x_i, y_i. The corresponding architecture is shown in Fig. 2(b). The advantage of (2) lies in a faster computation speed at the cost of additional hardware resources.

A unary bit-stream is mathematically describable through a length (or period), n, and number of 1s, v. The value represented by a unary bit-stream is therefore $\frac{v}{n}$. For example, if $n = 16$ and $v = 4$, the bit-stream represents $\frac{v}{n} = \frac{4}{16} = 0.25$. We will show that factors with relatively prime lengths of $k = n - 1$ always have a centered interval of uninterrupted 0s in the products. A relative delay (i.e., a unique lag between bit-streams) can position 1s of other summands in an interval where all other products exclusively have 0s. This allows accurate accumulation of stochastic-products through logic-`OR`. The proposed technique has no restrictions concerning the cause of the relative delays between summands. As examples, we provide three possible implementations in Section VI-B.

Throughout the paper, whenever we refer to multiplication, the underlying operation is logic-`AND`, and addition stands for applying logic-`OR` to stochastic bit-streams. When we use the terms summands, products or the inputs of the `OR` gate, we refer to the intermediate results between multiplication and summation. We emphasize that the focus of this paper lies in the summation of unary bit-streams. For more detail on why and how multiplication of unary bit-streams is performed accurately, we refer the reader to [5]. In Section III, we will take a closer look at the limits on summation of products using logic-`OR` and upper-error-bound, if these limits are exceeded.

III. Theorem

A. Claim

Let v denote the maximum allowed number of 1s in the input bit-streams. If binary numbers, represented by unary bit-streams with relatively prime lengths of n, k (where $k = n-1$ and both inputs are less than or equal to $\frac{v}{k}$) are multiplied using logic-conjunction, then N products (results of multiplication) can be accurately summed using logic-disjunctions, when each summand is processed with a predefined lag. The upper-bound of N and v is given by

$$N \leq \left\lfloor \frac{n-v}{v} \right\rfloor \cdot \left\lfloor \frac{n}{v} \right\rfloor \tag{3}$$

$$v \leq \left\lfloor \frac{n}{\lceil \frac{\sqrt{4N+1}-1}{2} \rceil + 1} \right\rfloor, \tag{4}$$

where $\{N, n, v \in \mathbb{N}\}$.

B. Expanded Explanation and Examples

Note that N is the number of inputs to the `OR` gate (not to the MAC unit). A pairwise multiplication of $2N$ inputs with `AND` gates results in N products, which are then summed by `OR` gates. For example, if $n = k+1 = 8$ and $v = 4$, at most $N = 2$ products can be summed by an `OR` gate without producing any error. The four inputs of the MAC unit must be less than or equal to $\frac{v}{k} = 0.5714$. As some additional examples, Table I lists the maximum allowed input thresholds for different $n = k+1$ and N for exact MAC.

Note that, Equation (4) can be used when the application determines the number of inputs. In that case, the input range is $[0, \frac{v}{k}]$. We also provide (3), which is more applicable when the input ranges are known in advance or application does not imply a specific number of inputs.

The delays mentioned above need to be applied in the computation-chain before performing logic-`OR`. This is done by either inserting registers, delaying the bit-stream generation, or using the intrinsic delay of sequential arriving data. The important point in performing exact addition is that all summands are delayed differently and therefore, N unique delays are required to sum N products. In Section IV, we will derive two types of delays. Long or *major*, and intermediate or *minor* delays. We will show N_{major}=$\lfloor \frac{n-2v}{v} \rfloor$ ($N_{major} \in \mathbb{N}$) as the number of different long and N_{minor}=$N_{major}+1$=$\lfloor \frac{n-v}{v} \rfloor$ ($N_{minor} \in \mathbb{N}$) as the number of different minor delays. Having defined the required variables, we will now present the algorithm that computes the delays. Algorithm 1 gets two inputs n, v and returns a vector with N=$(N_{major}+1) \cdot (N_{minor}+1)$ unique delays, which guarantees exact MAC for input values less than or equal $\frac{v}{k}$. As an example, we calculate N_{major} = 1 and N_{minor} = 2, when n=16 and v=5. We compute N=6 different delays with Algorithm 1 and get $Delays := \{0, 5, 10, 80, 85, 90\}$. Fig. 3 shows the summation for this example. The first six sequences are the delayed products of two factors ($\frac{5}{16} \cdot \frac{5}{15}$). The last sequence represents the sum, produced using a 6-input `OR` gate.

TABLE I: Examples for the accuracy threshold $\frac{v}{k}$ for different period lengths $n = k+1$ and number of MAC inputs, see Equation (92)

Calculation with input values below the thresholds is exact.

	Period length $n = k+1$				
	16	32	64	128	256
4 Inputs (N=2)	0.53	0.52	0.51	0.50	0.50
12 Inputs (N=6)	0.33	0.32	0.33	0.33	0.33
24 Inputs (N=12)	0.27	0.26	0.25	0.25	0.25
40 Inputs (N=20)	0.20	0.19	0.19	0.20	0.20
60 Inputs (N=30)	0.13	0.16	0.16	0.17	0.16

Algorithm 1 Computing delays for N summands

```
Input: $n, v$
Output: $Delays$
1: $N_{major} = \lfloor \frac{n-2v}{v} \rfloor$, $N_{minor} = \lfloor \frac{n-v}{v} \rfloor$, $i = 1$
2: for $q = 0, 1, 2,$ to $N_{major}$ do
3:     for $p = 0, 1, 2,$ to $N_{minor}$ do
4:         $Delays(i) = q \cdot vn + p \cdot v$
5:         $i = i + 1$
6:     end for
7: end for
8: return $Delays$
```

If input bit-streams have more 1s than allowed by (4), exact MAC operation cannot be guaranteed. In that case, inputs represent values that are greater than $\frac{v}{k}$. Pairwise multiplication of these inputs is exact, but the accumulation of their products causes error. In that circumstance, the upper error bound, E_{upper}, gives the maximum summation error:

$$E_{upper} = \frac{\frac{3}{2}c(c+1) + c(v-1)}{nk} \cdot L \tag{5}$$

where c is the number of extra (and potentially overlapping) 1s per period in the input bit-streams and $v+c$ is the total number of 1s in them. Further, $L (L \leq N | L \in \mathbb{N})$ is the number of products, where inputs represent values in $[0, \frac{v+c}{n}]$. We highlight that L is the number of products, i.e., the number of inputs to the `OR` gate.

Subsequently, we give two examples for the usage of (5). In the first one, the application guarantees that half of the inputs satisfy (4), but no assumptions are made for the remaining inputs. This example is meant to point out how to use variable L to consider correct summands during the error estimation. Fig. 4 shows the proposed MAC circuit with two of four input

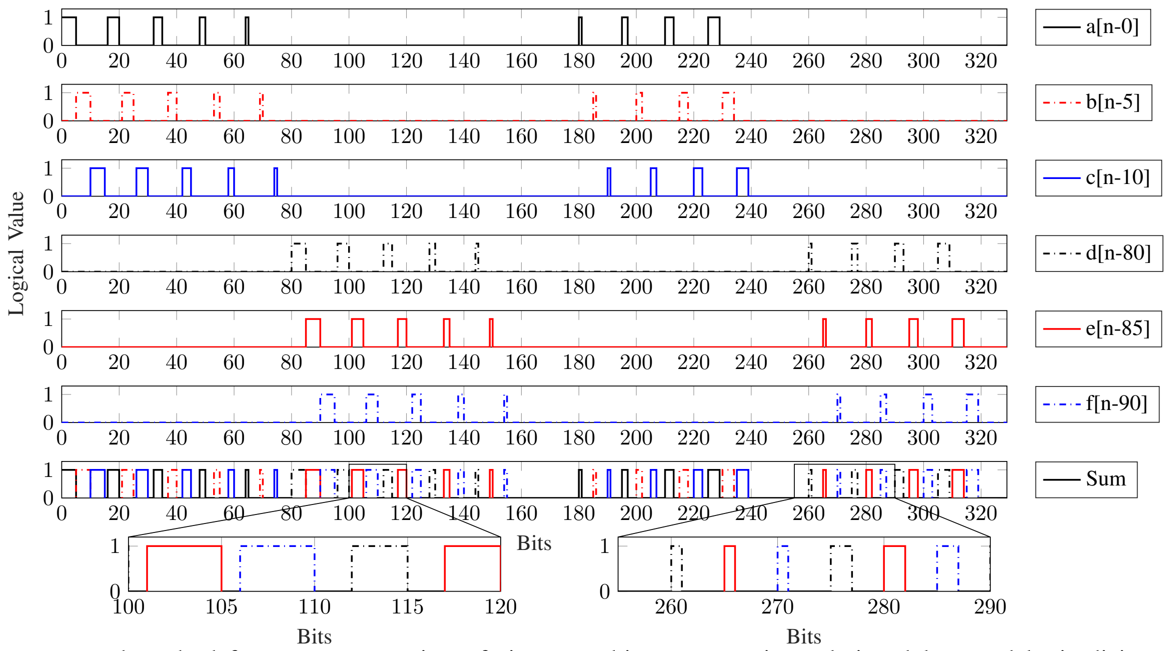


Fig. 3: The proposed method for exact summation of six unary bit-streams using relative delays and logic-disjunction. The summands a, b, c, d, e, f are delayed by $0, 5, 10, 80, 85, 90$ bits, respectively. All summands are equal besides of the different delay ($a = b = c = d = e = \frac{v}{n}\frac{v}{k}$) and sum to $a + b + c + d + e = 6(\frac{v}{n}\frac{v}{k}) = 0.625$, with $n = 16$, $k = 15$ and $v = 5$. The bottom plot shows the logic-OR of upper sequences and represents the exact sum.

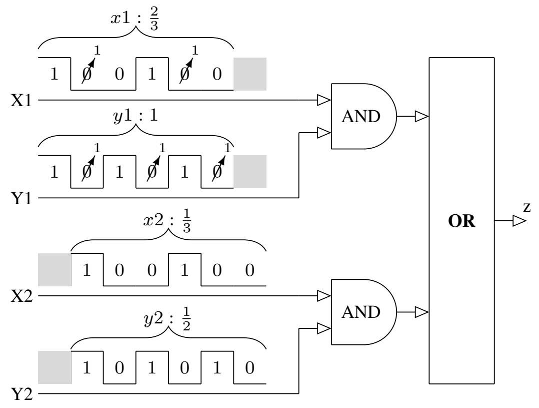


Fig. 4: The proposed MAC technique for four inputs, relative prime lengths $n = k + 1 = 3$ and a relative delay between summands of 1 bit (marked by gray squares). The circuit is exact for inputs with at most 1 bit logic-1 per period (input values less than or equal $\frac{v}{k} = \frac{1}{2}$). Because $X1$ and $Y1$ exceed this limit by $c = 1$ bit, the result z has a maximum error of $E_{upper} = 0.5$ (5).

bit-streams ($X1$ and $Y1$) not satisfying (4), having one extra logic-1 ($c = 1$) per period. Because both are connected to the same AND gate, one product ($L = 1$) has too many 1s for exact summation. We use short relative prime periods $n = k + 1 = 3$ for better graphic display. The circuit computes 1101100 which represents $\frac{4}{6}$, the number of 1s divided by the least common multiple (LCM) of the input periods. The MAC error is $\frac{5}{6}$-$\frac{4}{6}$ $= \frac{1}{6}$, which is less than the upper error bound $E_{upper} = \frac{\frac{3}{2}\cdot 1\cdot(1+1)+1\cdot(1-1)}{3\cdot 2} \cdot 1 = \frac{1}{2}$.

In the second example, all input values stay in the allowed $[0, \frac{v}{k}]$ interval. The results are exact, when all input values are less than or equal $\frac{v}{k} = 0.315$, with $n = k+1 = 16$, $v = 5$ and $N = 6$. The maximum exact result is $z = \sum_1^6 \frac{5}{16} \cdot \frac{5}{15} = 0.625$, when all input bit-streams have $v = 5$ 1s per period. In contrast to the first example, all input values exceed the threshold at the same time. Now assume that all 12 MAC inputs ($L = 6$) have one additional bit toggled from 0 to 1 ($c = 1$). The output increases to $z = \sum_{i=1}^{N} \frac{6}{16} \cdot \frac{6}{15} = 0.9$, whereas the proposed method computes 0.8625. The error $0.9 - 0.8625 = 0.0375$ is less than the upper-error estimation in (5):

$$E_{upper} = [\frac{3}{2} \cdot 1(1+1) + 1 \cdot (5-1)]/(16 \times 15) \times 6 = 0.175 \quad (6)$$

In the next section, we prove that the proposed method is exact when condition (3) or (4) is satisfied.

IV. Proof of the Theorem

Assume S_n and S_k are two unary bit-streams with v 1s, followed by $n - v$ and $k - v$ 0s. Their product is computed by feeding k repetitions of S_n and n repetitions of S_k to an AND gate as elaborated in [7]. The n times repetition of S_k will be denoted as $S_{k,n}$. Similarly, $S_{n,k}$ is the k times repeated sequence of S_n. $S_{n,k}$, $S_{k,n}$ and their logic-conjunction S_{AND} are piecewise defined as

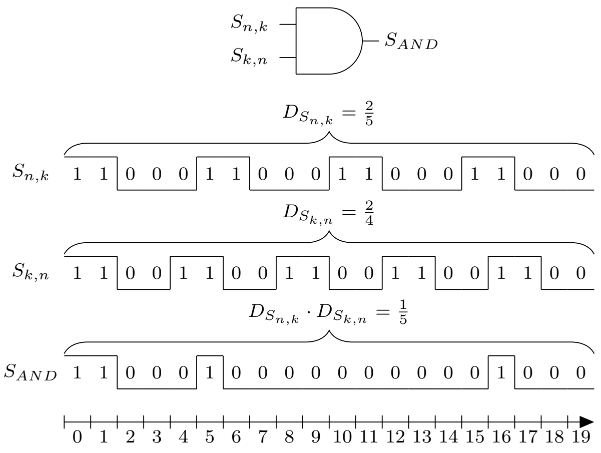
$$S_{n,k}[i] = \begin{cases} 1 & \{nx \le i < nx + v\} \quad \text{(a)} \\ 0 & \{nx + v \le i < n(x+1)\} \quad \text{(b)} \end{cases} \quad (7)$$

$$S_{k,n}[i] = \begin{cases} 1 & \{ky \le i < ky + v\} \quad \text{(a)} \\ 0 & \{ky + v \le i < k(y+1)\} \quad \text{(b)} \end{cases} \quad (8)$$

$$S_{\text{AND}}[i] = \text{AND}(S_{n,k}, S_{k,n}) \quad (9)$$

$$\text{with } \{0 \le x < k, 0 \le y < n | x, y \in \mathbb{N}\}. \quad (10)$$

Note that, in the binary domain, (7) and (8) are factors of the product represented by (9). Fig. 5 shows these sequences for $n = 5$, $k = 4$ and $v = 2$. The subsequent proof of (3) is divided into two parts. We begin by deriving positions of 1s in S_{AND}. We will then use this information to derive the relative delays that lead to no overlap between 1s. Using these delays will guarantee exact summation through logic-disjunction.



$S_{n,k}$
$S_{k,n}$
S_{AND}
$D_{S_{n,k}} = \frac{2}{5}$
$S_{n,k}$ 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0
$D_{S_{k,n}} = \frac{2}{4}$
$S_{k,n}$ 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
$D_{S_{n,k}} \cdot D_{S_{k,n}} = \frac{1}{5}$
S_{AND} 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Fig. 5: Stochastic multiplication of two unary bit-streams using AND gate. $S_{n,k}$ represents $\frac{v}{n} = \frac{2}{5}$ with $v = 2$ 1s and a period of $n = 5$ bits (Duty Cycle $D_{S_{n,k}=0.4}$). $S_{k,n}$ represents $\frac{v}{k} = \frac{2}{4}$ with a period of $k = 4$ bits (Duty Cycle $D_{S_{k,n}=0.5}$). The output S_{AND} represents $\frac{1}{5}$.

By definition $\text{AND}(S_{n,k}[i], S_{k,n}[i])$ produces a 1 when

$$\{S_{n,k}[i] = 1 \quad \text{AND} \quad S_{k,n}[i] = 1\}. \tag{11}$$

It follows that both inequations, (7,a) and (8,a) are required to be fulfilled to get a 1 in S_{AND}.

$$\{nx \le i < nx + v \quad \text{and} \quad ky \le i < ky + v\} \tag{12}$$

We use natural numbers m, t to rewrite inequations (12) as

$$\{nx \le i < nx + v\} = \{i = nx + m\} \tag{13}$$
$$\{ky \le i < ky + v\} = \{i = ky + t\} \tag{14}$$
$$\text{with } \{0 \le m < v, 0 \le t < v | m, t \in \mathbb{N}\}.$$

Equations (12) to (14) lead to two possible substitutions:

$$\begin{aligned}
&\{\{nx \le i < nx + v \quad \text{and} \quad i = ky + t\} \quad \text{or} \\
&\{ky \le i < ky + v \quad \text{and} \quad i = nx + m\}\} &(15)\\
&= \{\{nx \le ky + t < nx + v\} \quad \text{or} \\
&\{ky \le nx + m < ky + v\}\} &(16)\\
&= \{\{-t \le ky - nx < v - t\} \quad \text{or} \\
&\{-m \le nx - ky < v - m\}\} &(17)\\
&= \{\{v - t > ky - nx \ge -t\} \quad \text{or} \\
&\{m \ge ky - nx > m - v\}\} &(18)
\end{aligned}$$

The disjunction in (18) has two maximum solutions ($m = t = 0$ and $m = t = v - 1$) that both can be simplified to (19).

$$\{v > ky - nx \ge 0 \quad \text{or} \quad 0 \ge ky - nx > -v\} \tag{19}$$
$$v > |ky - nx| \tag{20}$$

For given k and n there are different solutions to (20). We set $k = n - 1$ for the reasons mentioned in Section I. Hence,

$$v > |n(y - x) - y| \tag{21}$$

A closer look at (21) and constraint $n > v$ reveals that no solution is possible for $y \le x - 1$ and $y \ge x + 2$ because both equations result in a contradiction:

$$\begin{aligned}
&\text{for } y \le x - 1 \rightarrow y = x - 1 - m \\
&\quad v > |n(y - x) - y| \overset{y=x-1-m}{=} |-n - mn - y| \not< v &(22)\\
&\text{for } y \ge x + 2 \rightarrow y = x + 2 + m \\
&\quad v > |n(y - x) - y| \overset{y=x+2+m}{=} |2n + mn - y| \not< v &(23)\\
&\quad \text{with } \{m \ge 0 | m \in \mathbb{N}\}
\end{aligned}$$

Next, we find explicit solutions for x, knowing that the only possible values for y in (21) are (I.) $y = x$ and (II.) $y = x + 1$.

$$\begin{aligned}
v > |ky - nx| = |ny - y - nx| &\quad (24)\\
\text{for (I)} \quad y = x &\\
v > |nx - x - nx| = x \rightarrow x < v &\quad (25)\\
\text{for (II)} \quad y = x + 1 &\\
v > |nx + n - x - 1 - nx| = |n - x - 1| &\quad (26)\\
\overset{x \le n}{=} n - 1 - x \overset{k=n-1}{=} k - x \rightarrow x > k - v &\quad (27)
\end{aligned}$$

Therefore, 1s in S_{AND} only appear in the beginning and end ($x < v$ and $x > k - v$). Substituting for y in (12) leads to (30) and (33) that describe 1s of S_{AND}, if both factors have v 1s.

$$\begin{aligned}
&\overset{y=x}{\rightarrow} \{nx \le i < nx + v \text{ and } kx \le i < kx + v\} &(28)\\
&= \{\max(nx, kx) \le i < \min(nx + v, kx + v)\} &(29)\\
&= \{nx \le i < kx + v\} \text{ for } x < v &(30)
\end{aligned}$$

$$\begin{aligned}
&\overset{y=x+1}{\rightarrow} \{nx \le i < nx + v \text{ and } kx + k \le i < kx + k + v\} &(31)\\
&= \{\max(nx, kx + k) \le i < \min(nx + v, kx + k + v)\} &(32)\\
&= \{k(x + 1) \le i < nx + v\} \text{ for } x > k - v. &(33)
\end{aligned}$$

$$S_{\text{AND}}[i] = \begin{cases} 1 & \{nx \le i < kx + v\} \quad \text{(a)} \\ & \text{for } 0 \le x < v \\ 1 & \{k(x + 1) \le i < nx + v\} \quad \text{(b)} \\ & \text{for } k > x > k - v \\ 0 & \text{else} \quad \text{(c)} \end{cases} \tag{34}$$

We will follow the common notation $|I|$ for the length of interval I. The length of an interval is the absolute value of the difference between the two endpoints. The minimal interval with property $\{\{nx \le i < kx + v\} \text{ for } x < v\} \subset I_{begin}$ that includes all sections of (34,a) is

$$I_{begin} = [0, k(v - 1) + v) \tag{35}$$
$$|I_{begin}| = kv + v - k = nv - k. \tag{36}$$

In the same manner, we define I_{end} for (34,b) and get

$$I_{end} = [k(k - v + 2), n(k - 1) + v) \tag{37}$$
$$|I_{end}| = nv - n - k. \tag{38}$$

Observe that $|I_{begin}|$ is greater than $|I_{end}|$ regardless of v.

$$\max(|I_{begin}|, |I_{end}|) = |I_{begin}| = nv - k \tag{39}$$

Both I_{begin} and I_{end} contain a certain percentage of 1s proportional to the values of factors. On the contrary, $I_{F,major}$ contains 0s only and is between I_{begin} and I_{end}.

$$\begin{aligned} I_{F,major} &= [nv, n(k-v+1)) \\ &= [nv, n(n-v)) \end{aligned} \tag{40}$$

$$\begin{aligned} |I_{F,major}| &= n(n-v) - (nv) \\ &= n^2 - 2nv. \end{aligned} \tag{41}$$

Knowing $q \cdot vn \geq |I_{begin}|$ $(1 \leq q | q \in \mathbb{N})$, a relative delay of $q \cdot vn$ bits between two summands avoids overlap between their I_{begin}. As a result of $|I_{begin}| > |I_{end}|$ it also avoids overlap between their I_{end}. Delays of form $q \cdot vn$ will be called major delays. We write $S_{\texttt{AND},q}$ for major-delayed products and $I_{begin,q}$, $I_{end,q}$, $I_{F,major,q}$ for their right-shifted intervals:

$$S_{\texttt{AND},q}[i] = S_{\texttt{AND}}[i - q \cdot vn] \tag{42}$$

The task is now to find maximum $q_{max} = N_{major}$ that $I_{begin,N_{major}}$ does not intersect with I_{end} of other products.

$$|I_{F,major}| - (vn)N_{major} \geq 0 \tag{43}$$

$$N_{major} \leq \frac{|I_{F,major}|}{vn} \tag{44}$$

$$N_{major} \leq \frac{n-2v}{v} \tag{45}$$

It follows that interval $I_{begin,q}$ $(1 \leq q \leq N_{major})$ is in $I_{F,major,q-1}$ and $I_{end,q}$ is right-shifted out of $S_{\texttt{AND},q-1}$. Therefore, $N_{major} + 1 = \lfloor \frac{n-v}{v} \rfloor$ major-delayed products do not create overlap in logic-disjunction, when each of them is uniquely delayed.

We can now consider the derivation of N_{minor} and factor $\lfloor \frac{v}{n} \rfloor$ in (3). By definition of logic-AND, 0s in $S_{n,k}$ (and $S_{k,n}$) also appear in $S_{\texttt{AND}}$. Substituting $i \to i - q \cdot vn$ into (7,b) shows that applying major delays extend and do not disturb the periodicity of 0s. Therefore, $S_{\texttt{AND},q}$ and the logic-disjunction of them share a regular occurring pattern of 0s, denoted $I_{F,minor}$.

$$S_{\texttt{AND},q}[i] = 0 \text{ for } \{n(x+qv) + v \leq i < n(x+1+qv)\} \tag{46}$$

$$I_{F,minor} = [n(x+qv) + v, n(x+1+qv)) \tag{47}$$

$$|I_{F,minor}| = n(x+qv+1) - (n(x+qv) + v) \tag{48}$$

$$= n - v \tag{49}$$

Let $I_{two-level}$ be a recurring interval that does not intersect with $I_{F,minor}$. $I_{two-level}$ has 1s proportional to the input values and contains all 1s of $S_{\texttt{AND},q}$. The exact positions of 1s are not required for this proof.

$$\begin{aligned} I_{two-level} &= [n(x+qv), n(x+qv) + v) \\ &\quad \text{for } 0 \leq x < v \text{ and } k > x > k - v \\ &\quad \text{with } I_{two-level} \cap I_{F,minor} = \{\} \end{aligned} \tag{50}$$

$$|I_{two-level}| = v \tag{51}$$

According to (49) and (51), $S_{\sum \texttt{AND},q}$ consists of an alternating pattern of v logical-undetermined bits, followed by $n - v$ bits being guaranteed 0s. Assuming $|I_{F,minor}| \geq |I_{two-level}|$, a relative minor-delay of $|I_{two-level}| = v$ between $S_{\texttt{AND},q}$ avoids overlap at logic-disjunction, because $I_{F,minor}$ exclusively has 0s. We now proceed similar to (45) and compute the number of unique minor-delays N_{minor}.

$$|I_{F,minor}| - |I_{two-level}| \cdot N_{minor} \geq 0 \tag{52}$$

$$N_{minor} \leq \frac{|I_{F,minor}|}{|I_{two-level}|} = \frac{n-v}{v}. \tag{53}$$

The delay for each summand is computed with two variables $0 \leq q \leq N_{major}$ and $0 \leq p < N_{minor}$ with $\text{Delay}(q, v) = q \cdot vn + p \cdot v$. In its final form $S_{\texttt{AND},q,p}[i] = S_{\texttt{AND},q}[i - p \cdot v] = S_{\texttt{AND}}[i - q \cdot vn - p \cdot v]$ is defined as

$$S_{\texttt{AND, q, p}}[i] = \begin{cases} 1 \ \{nx + q \cdot vn + p \cdot v \leq i < kx + q \cdot vn + (p+1) \cdot v\} \\ \quad \text{for } x < v \\ 1 \ \{k(x+1) + q \cdot vn + p \cdot v \leq i < nx + q \cdot vn + (p+1) \cdot v\} \\ \quad \text{for } x > k - v \\ 0 \ \text{else} \end{cases} \tag{54}$$

The result of MAC operation is

$$S_{SUM}[i] = \sum_{q=0,p=0}^{N_{major},N_{minor}} S_{\texttt{AND}}[i - q \cdot vn - p \cdot v] \tag{55}$$

Counting the number of different available delays $(N \in \mathbb{N})$, that is the possibilities for q and p leads to the claim in (3):

$$N \leq (N_{major} + 1)(N_{minor} + 1) \tag{56}$$

$$N \leq \left\lfloor \frac{n-2v}{v} + 1 \right\rfloor \left\lfloor \frac{n-v}{v} + 1 \right\rfloor = \left\lfloor \frac{n-v}{v} \right\rfloor \left\lfloor \frac{n}{v} \right\rfloor \tag{57}$$

The maximum integer function in (57) is required because N_{minor} and N_{major} are naturals. Solving (57) for v is possible because of the property $N_{minor} = N_{major} + 1$. Since N is the product of two consecutive integers we call N a pronic number ($N = x(x+1)$ with $x = N_{major}+1$). By analogy with the square root of N, the pronic-root of N is $x = \frac{\sqrt{4N+1}-1}{2}$. Knowing N we can calculate its N_{major} with

$$N_{major} + 1 = \left\lceil \frac{\sqrt{4N+1} - 1}{2} \right\rceil \tag{58}$$

Next, we solve (45) for natural v and substitute for $N_{major} + 1$. The result is equal to claim (4).

$$v \leq \left\lfloor \frac{n}{N_{major} + 2} \right\rfloor \tag{59}$$

$$v \leq \left\lfloor \frac{n}{\lceil \frac{\sqrt{4N+1}-1}{2} \rceil + 1} \right\rfloor \tag{60}$$

In the following section, we discuss the behavior of the proposed technique when the maximum input constraint (4) is not satisfied. We derive a maximum error-bound and prove that the error is less than E_{upper} from (5).

V. Proof of the Upper Error Bound

This section proves that inputs representing greater values than (4) cause a maximum error given in (5). So far, we have been working under the assumption that the system was designed for N summands and input values are in the $[0, \frac{v}{k}]$ interval. Now suppose that the allowed input interval expands to $[0, \frac{v+c}{n}]$ and N is not reduced. To account for greater input values, we substitute $v \to v+c$ in (34,a) and (34,b), and write S'_{AND} for products (summands) with increased input values.

$$S'_{\text{AND}} = \begin{cases} 1 & \{nx \le i < kx + (v+c)\} \quad \text{(a)} \\ & \text{for } 0 \le x < (v{+}c) \leftrightarrow x \in \{x_0, x_1\} \\ 1 & \{k(x+1) \le i < nx + (v+c)\} \quad \text{(b)} \\ & \text{for } k > x > k - (v{+}c) \leftrightarrow x \in \{x_3, x_4\} \\ 0 & \text{else} \quad \text{(c)} \end{cases} \tag{61}$$

Partitioning index x ($0 \le x < k$) into five shorter indexes ($x = x_0 \cup x_1 \cup x_2 \cup x_3 \cup x_4$) simplifies the subsequent error analysis.

$$0 \le x_0 < v \tag{62}$$
$$v \le x_1 < (v+c) \tag{63}$$
$$(v+c) \le x_2 < k - (v+c) + 1 \tag{64}$$
$$k - (v+c) + 1 \le x_3 < k - v + 1 \tag{65}$$
$$k - v + 1 \le x_4 < k \tag{66}$$

Whenever we analyze one specific section of S_{AND}, we abbreviate intervals of S_{AND} with side condition $x = x_i \in \{x_0, x_1 \dots x_6\}$ to S'_{AND,x_i}:

$$S'_{\text{AND},x_i} = S_{\text{AND}} \text{ and } x = x_i \tag{67}$$

Logic-1s within previously assumed logic-0 intervals produce error at logic-OR, because they can overlap with 1s of other delayed summands. The proposed upper-error bound assumes all extra 1s to cause error and sums the individual components. In what follows, the error is calculated by intersecting both intervals ((47) and (40)) with 1s in S'_{AND} (61,a) and (61,b). The intersection between intervals is defined as $I_i \cap I_j$. We use variable $M_{i,j}$ for the number of bits in the intersections.

$$I_1 \cap I_2 = [a, b) \cap [c, d) \tag{68}$$
$$= [\max(a, c), \min(b, d)) \tag{69}$$
$$M_{1,2} = |I_1 \cap I_2| \tag{70}$$
$$= \min(b, d) - \max(a, c) \tag{71}$$

There is no intersection between $I_{F,major}$ and $S_{\text{AND},x0}$, because $max(S_{\text{AND},x_0}) < min(I_{F,major})$. Same technique for x_1 shows that in the worst case, $S'_{\text{AND},x1}$ is in $I_{F,major}$, so all 1s of $S'_{\text{AND},x1}$ could cause error. We sum all bits of $|S'_{\text{AND},x1}|$ using Gauss-Sum for natural numbers:

$$M_{x_1,F_{major}} = |S'_{\text{AND},x_1} \cap I_{F,major}| \tag{72}$$
$$= \sum_{x=x_1} \min(kx + (v+c), n(n-v))$$
$$- \max(nx, nv) \le \sum |S'_{\text{AND},x_1}| \tag{73}$$
$$= \sum_{x=x_1} |[nx, kx + v + c)| \tag{74}$$
$$= \sum_{x=v}^{v+c-1} v + c - x = \frac{c(c+1)}{2} \tag{75}$$

By definition $S'_{\text{AND},x_2}[i]$=0 regardless of inputs and $M_{x_2,F_{major}}$ is zero as a result. In worst case, $S'_{\text{AND},x3}$ is completely in $I_{F,major}$, which can be seen when comparing their maximum and minimum. Hence, we again sum bits in S'_{AND,x_3}.

$$M_{x_3,F_{major}} = |S'_{\text{AND},x_3} \cap I_{F,major}| \le \sum |S'_{\text{AND},x_3}| \tag{76}$$
$$= \sum_{x=x_3} |[k(x+1), (k+1)x + (v+c))| \tag{77}$$
$$= \sum_{x=k-(v+c)+1}^{k-(v)} (v+c) - k + x = \frac{c(c+1)}{2} \tag{78}$$

Similar to x_0 there is no error for x_4, because $\max(I_{F,major}) < min(S'_{\text{AND},x_4})$. In (47) we showed that $I_{F,minor}$ does not intersect with 1s in S_{AND} and contains 0s only. For x_1, x_2, x_3, $I_{F,minor}$ is in $I_{F,major}$, so error within this interval are already covered. Substituting $v \to (v+c)$ in (50) leads additional error for x_0 and x_4:

$$M_{x_0,F_{minor}} = |I_{F,minor} \cap |S'_{\text{AND},x_0}| \tag{79}$$
$$= \sum_{x=x_0} [nx + v, n(x+1)) \cap [nx, kx + (v+c)) \tag{80}$$
$$= \sum_{x=x_0} \min(n(x+1), kx + (v+c))$$
$$- \max(nx + v, n(x+1)) \tag{81}$$
$$= \sum_{x=x_0} kx + (v+c) - nx - v \tag{82}$$
$$= \sum_{x=0}^{v-1} c - x \le \frac{c(c+1)}{2} \tag{83}$$

Repeating the same concept for $I_{F,minor} \cap S'_{\text{AND},x_4}$ with $\max(nx + v, k(x+1)) = nx + v$ gives $M_{x_4,F_{minor}}$.

$$I_{F,minor} \cap S'_{\text{AND},x_4} = [nx + v, n(x+1))$$
$$\cap \{[k(x+1), nx + (v+c)) \text{ for } k > x > k - v\} \tag{84}$$
$$= \min(n(x+1), nx + (v+c))$$
$$-\max(nx + v, k(x+1)). \tag{85}$$

$$M_{x_4,F_{minor}} = |I_{F,minor} \cap S'_{\text{AND},x_4}| \tag{86}$$
$$= \sum_{x=x_4} (nx + (v+c)) - (nx + v) \tag{87}$$
$$= \sum_{x=k-v+1}^{k-1} c = (v-1)c \tag{88}$$

M_{total} is the sum of individual error contributing intersections.

$$M_{total} = \sum M_{x_i} = \frac{3}{2}c(c+1) + c(v-1) \tag{89}$$

Up to this point we considered only one summand having extra 1s that cause error. This is the result of two MAC inputs having $v+c$ instead v 1s. When L ($L \in \mathbb{N}$) summands exceed the allowed input range, the error increases proportional to L. The number of potential error bits is converted to the positional binary representation E_{upper} by scaling with $\frac{1}{nk}$.

$$E_{upper} = \frac{\frac{3}{2}c(c+1) + c(v-1))}{nk} \cdot L \tag{90}$$

The upper error limit E_{upper} increases linear w.r.t. v and L, and quadratic with c. Note that $E_{upper} = 0$ (the computation is accurate), when $c = 0$ (Equation (4) is satisfied).

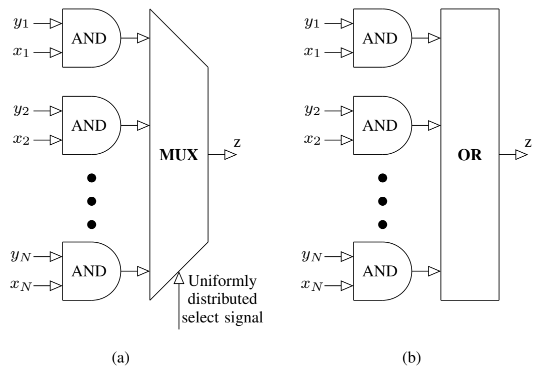


Fig. 6: Two common SC MAC: (a) AND gate followed by a multi-input MUX, (b) AND gate followed by a multi-input OR.

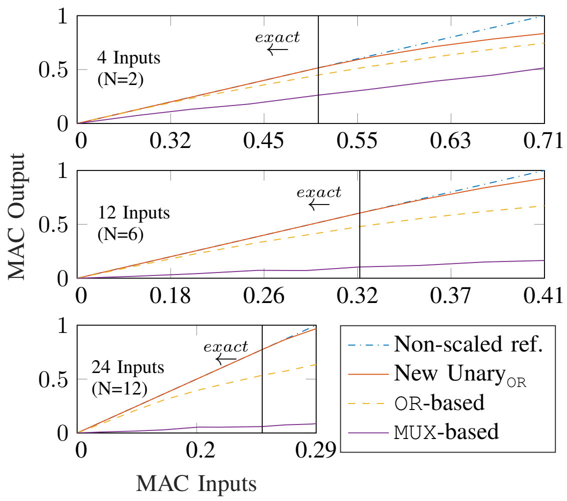


Fig. 7: MAC outputs for OR-based and MUX-based as well as the proposed Unary$_{\mathrm{OR}}$ as a function of input value and number of inputs. Up to the vertical line (thresholds from Table I), the proposed Unary$_{\mathrm{OR}}$ is exact and provides the same output as a non-scaled MAC, shown as diagonal line.

VI. Experimental Result

In this section, we verify our theory by using circuit implementations of the proposed technique. We compare the proposed unary bit-stream-based MAC with three state-of-the-art designs which represent data using Sobol-based low-discrepancy (LD) bit-streams [20] while perform the summation using either MUX or OR gates. Since all four designs use AND gates for multiplication, we distinguish them by their data representation and summation method, and call them Sobol$_{\mathrm{MUX}}$, Sobol$_{\mathrm{MUX/TFF}}$, Sobol$_{\mathrm{OR}}$ and Unary$_{\mathrm{OR}}$. Below, we discuss the MUX-based methods (Sobol$_{\mathrm{MUX}}$ and Sobol$_{\mathrm{MUX/TFF}}$) and OR-based method (Sobol$_{\mathrm{OR}}$) and their corresponding equations and compare them to our proposed technique.

A. Traditional summation

Fig. 6(a) shows the most common SC design for MAC units. Multiplications are performed using AND gates and summation is implemented with MUX unit [3] [21] [22]. A MUX-based adder divides the sum by the number of data inputs N. The MAC function is described as follows:

$$z_{\mathrm{MUX}} = \frac{\sum_{1\le i\le N} x_i y_i}{N} \tag{91}$$

where x_i and y_i are inputs, and z_{MUX} is the output. OR gates replace the MUX unit in an alternative MAC circuit [23] [16] (see Fig. 6(b)). OR gates find the union of input bit-streams. For example, the result of a three-input OR is $z_{OR} = a_1 \cup a_2 \cup a_3 = a_1 + a_2 + a_3 - a_1a_2 - a_1a_3 - a_2a_3 + a_1a_2a_3$. In a two-stage stochastic MAC, a_is are the outputs of AND gates ($a_i = x_iy_i$ and $i = 1, 2, \ldots N$).

Recent works on deterministic methods of SC [8] [7] [5] showed that completely accurate computations can be done using SC designs. Different deterministic approaches were proposed based on LD [24], pseudo-random [5], and unary bit-streams [8]. All these methods produce completely accurate output when processing bit-streams with a specific length (i.e., $2^{N\times M}$ bits where N is the number of inputs and M is the precision of data) and decrease in accuracy when shorter bit-streams are processed. Among these, Sobol-based LD methods [20] [25] have shown minimum random fluctuations and fastest convergence to the expected output [5]. The authors in [5] showed exact and fast converging multiplication with AND gates and scaled addition with MUX when processing Sobol-based bit-streams. The required independence between Sobol bit-streams is provided by generating each bit-stream based on a different Sobol sequence. When used with OR gates, Sobol bit-streams produce the union of the inputs with improved latency compared to pseudo-random and unary bit-streams. We compare our proposed MAC design with three baseline designs processing bit-streams of this form.

Fig. 7 presents the outputs for approximate OR-based MAC, scaled MUX-based MAC and the proposed Unary$_{\mathrm{OR}}$ to show the input-ouput behavior of each method. Note that the number of MAC inputs is twice that of N due to the pairwise multiplication before summation. The 5-bit precision ($n = 32, k = 31$) input values are marked on the X-axis and are equally increased until the reference MAC output, shown on the y-axis, reaches 1 ($z = \sum_{i=1}^{N} x_i y_i \overset{!}{=} 1$). The non-scaled reference MAC output is a diagonal line because MAC is a linear operation. While each of the implemented methods exhibits some degree of deviation compared to the reference, MUX-based methods diverge the most from the non-scaled reference, because a MUX unit performs scaled rather than normal addition. Traditional OR-based methods perform comparably to the proposed Unary$_{\mathrm{OR}}$ design for small input values but falls behind at larger ones. Unary$_{\mathrm{OR}}$ is closer to the non-scaled reference MAC than MUX-based and OR-based designs regardless of the number of inputs (e.g., 4, 12, and 24) and is the only method that can provide completely accurate results for small input values. The thresholds in which the proposed method stops producing exact result is marked by vertical lines in Fig. 7. These match the thresholds from Table I after quantization. For example, the computed threshold for $n = 32$ and $N = 2$ is 0.52 that, after rounding to the next representable value, becomes 0.5. A 32-bit unary bit-stream represents 0.5 with 16 1s followed by 16 0s. For input values greater than the threshold, the proposed method guarantees the upper-error bound (5) discussed in Section V.

B. Practical Implementation

So far we distinguished the individual methods by their summation technique (OR-based or MUX-based) and the SNGs used. In this section, we further refine our MAC techniques and introduce Sobol$_{\texttt{MUX/TFF}}$, Unary$_{\texttt{OR/REG}}$ and Unary$_{\texttt{OR/SEQ}}$. The latter two are alternative designs for the proposed MAC technique, which achieve the same accuracy results as Unary$_{\texttt{OR}}$, but require different resources. Hence, we will not discuss them in accuracy comparisons. However, their differences will be discussed in the resource comparison section (Section VII). Sobol$_{\texttt{MUX/TFF}}$ is a specific implementation of MUX-based addition that does not require an additional select input. The method is extensively discussed in [14]. Both, the accuracy and the resource requirements differ from Sobol$_{\texttt{MUX}}$, so it will be separately discussed in our accuracy and resource comparisons. The three Unary$_{\texttt{OR}}$ methods reflect three examples of making relative delays between summands:

1) Unary$_{\texttt{OR}}$ uses a separate SNG (consists of a counter and a comparator) for each MAC input. The SNGs are enabled pairwise at the clock cycles calculated with Algorithm 1 for nk clock cycles. Two SNGs that are connected to the same AND gate form one pair. The counters have relative prime periods n and $k = n - 1$. The bit-streams are 0, when the SNGs are disabled. The computation architecture is equal to Fig. 2(b). Fig. 3 visualizes the concept of summing *delayed* bit-streams. Each SNG is enabled for 240 clock cycles in unique intervals and disabled (logic 0) for the remaining time.

2) Unary$_{\texttt{OR/REG}}$ differs in both bit-stream generation and cause of relative delays. It uses a pair of two counters instead of one per MAC input (i.e., a total of $2N$ counters). The values of the two counters are compared with each pair of factors. The relative delays are caused by bit-shift registers of different lengths between the outputs of the AND gates and the inputs of the OR gates. The length of the shift-register for summand i is equal to $Delays(i)$ computed with Algorithm 1. In Fig. 2(b), the length of the shift-register after AND gate with inputs x_1, y_1 is $Delays(1)$ (always 0), after AND gate with inputs x_2, y_2 is $Delays(2)$, and so on. It is important that the output bit stream of a SNG is 0, when the bit-stream generation is finished.

3) Unary$_{\texttt{OR/SEQ}}$ uses one SNG pair (one for each relatively prime length bit-stream) and the architecture in Fig. 2(a). The MAC works sequentially and the summands are added to the accumulator one after another. To achieve the same accuracy and output sequence the following two properties are required. First, the inputs to the SNG pair must change N times during MAC operation (once for each summand). Second, the SNG pair must be disabled for the correct number of cycles between generating bit-streams for different inputs, so that the new summand is added to the accumulator at the right time. The accumulator is a shift register of length $nk + N_{major} \cdot vn + N_{minor} \cdot v$ (the LCM of inputs plus the maximum delay from Algorithm 1). The SNGs must be stalled for $N_{major} \cdot vn + N_{minor} \cdot v$ cycles between generating the bit-streams of two subsequent input pairs. Simulations show that it is possible to use a different buffer size and avoid stalling the bit-stream generation (i.e., immediately generate the next bit-stream when the previous is done). However, finding a formula for optimal buffer sizes or proofing 100% accuracy for sequential computation without separately controlling the bit-stream generation requires further investigation and is part of our future work.

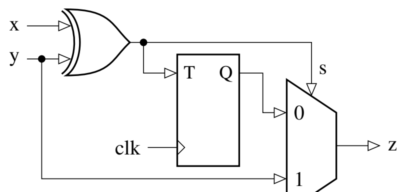


Fig. 8: Toggle flipflop based scaled adder implementation (Sobol$_{\texttt{MUX/TFF}}$) that does not need an additional RNG for the select input of the MUX [14].

All Sobol-based methods use the architecture in Fig. 2(b), but could also be adapted to use the sequential architecture of Fig. 2(a).

1) Sobol$_{\texttt{OR}}$ approximates non-scaled summation with a multi-input OR gate and requires N SNGs, because the AND gates as well as the OR gate require uncorrelated inputs.

2) Sobol$_{\texttt{MUX}}$ uses a multi-input MUX unit for scaled summation. The select input comes from a separate RNG that produces uniformly distributed random integer variables in the $[0, N - 1]$ interval. The inputs of each multiplication operation need to be uncorrelated, but the MUX inputs can be correlated [3]. Hence, two RNGs can be shared to generate all inputs of multiplication operations [5]. This method therefore requires three uncorrelated streams of random numbers. Two, to make the factor bit-streams (connected to AND gates) uncorrelated and one for the select input of the MUX.

3) Sobol$_{\texttt{MUX/TFF}}$ uses multiple 2-to-1 multiplexer stages shown in Fig. 8 for scaled summation without requiring an additional input (and RNG) for the select bit-stream. The circuit consists of a toggle flipflop and an XOR gate. Sobol$_{\texttt{MUX/TFF}}$ requires two streams of random numbers, which is the least of all methods in the parallel architecture.

C. Accuracy Comparison

a) Sobol$_{MUX}$ and Sobol$_{MUX/TFF}$: The accuracy of a stochastic MUX-adder decreases when the length of bit-stream is fixed but the number of inputs increases [26]. For input values close to 1, the $\frac{1}{N}$ factor in (91) helps keeping the result in the $[0, 1]$ interval. For input values close to zero, however, the result tends to be too small for accurate representation. Sobol$_{\texttt{MUX}}$ produces additional occurs if the select input of the MUX is correlated to its inputs. As argued in [24], different Sobol sequences should be used to generate independent bit-streams and avoid the correlation error.

b) Sobol$_{OR}$: The accuracy of the OR-based adder is low when input values are close to one ($x_i \approx x_i^2$) and is high when input values are close to zero ($x_i >> x_i^2$). Additional error occurs if input bit-streams are correlated; any overlap between the location of 1s in the input bit-streams decreases the accuracy.

TABLE II: Mean Absolute Error (MAE) (%) comparison of the proposed Unary$_{\texttt{OR}}$ with state-of-the-art design approaches for different number of summands N and different ranges of (5 bit precision) uniformly distributed random input values. The result of MUX-based methods is rescaled by constant N in a second processing step to compensate the MUX scaling

Mean Reference MAC Output		0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
		Range of Input Values									
		[0,0.31]	[0,0.54]	[0,0.71]	[0,0.84]	[0,0.95]	[0,1.05]	[0,1.14]	[0,1.22]	[0,1.31]	[0,1.38]
4 Inputs (N=2)	Unary$_{\texttt{OR}}$	**0.0**	**0.0**	0.3	1.4	3.2	5.5	8.1	10.0	12.4	14.4
	Sobol$_{\texttt{OR}}$	0.1	0.5	1.5	3.1	5.1	7.5	10.1	12.1	14.4	16.4
	Sobol$_{\texttt{MUX/TFF}}$-rescaled	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Sobol$_{\texttt{MUX}}$-rescaled	0.4	0.7	0.9	1.0	1.2	1.3	1.4	1.4	1.5	1.5
		Range of Input Values									
		[0,0.18]	[0,0.31]	[0,0.41]	[0,0.48]	[0,0.55]	[0,0.60]	[0,0.66]	[0,0.71]	[0,0.75]	[0,0.80]
12 Inputs (N=6)	Unary$_{\texttt{OR}}$	**0.0**	**0.0**	0.2	1.0	2.6	4.6	7.2	10.5	14.8	19.6
	Sobol$_{\texttt{OR}}$	0.4	1.1	2.5	4.7	7.7	10.9	15.1	19.6	25.1	30.9
	Sobol$_{\texttt{MUX/TFF}}$-rescaled	0.5	1.5	2.6	3.6	4.6	5.5	6.6	7.6	8.7	9.8
	Sobol$_{\texttt{MUX}}$-rescaled	0.9	1.6	2.1	2.4	2.8	3.1	3.3	3.5	3.8	4.0
		Range of Input Values									
		[0,0.13]	[0,0.22]	[0,0.29]	[0,0.34]	[0,0.39]	[0,0.43]	[0,0.46]	[0,0.50]	[0,0.53]	[0,0.56]
24 Inputs (N=12)	Unary$_{\texttt{OR}}$	**0.0**	**0.0**	**0.0**	0.6	1.8	3.6	6.1	9.0	12.5	16.5
	Sobol$_{\texttt{OR}}$	0.8	1.6	3.3	5.6	8.7	12.1	16.6	21.3	26.7	32.5
	Sobol$_{\texttt{MUX/TFF}}$-rescaled	0.8	1.4	2.0	2.7	3.4	4.1	4.8	5.5	6.3	6.8
	Sobol$_{\texttt{MUX}}$-rescaled	1.3	2.4	3.1	3.7	4.1	4.6	4.9	5.3	5.7	5.9

c) Unary$_{OR}$: The proposed method is accurate (4) for input values less than

$$\frac{v}{k} \leq \left\lfloor \frac{n}{\lceil \frac{\sqrt{4N+1}-1}{2} \rceil + 1} \right\rfloor \frac{1}{k}. \tag{92}$$

The accuracy decreases for input values greater than this threshold. A requirement for exact computation is applying the relative delays from Algorithm 1 before summation. The proposed method has a maximum error of (5) for input values greater than (92). As shown in Table I, this threshold highly depends on N, the number of summands. Note that Algorithm 1 must be modified for input values closer to 1. We restrict our modifications to decreasing v until Algorithm 1 returns enough delays. It is likely that a better adaption exists, but we leave the study of this aspect for future works.

D. Evaluation Results

In this section, we use 5-bit precision (n=32, k=31) unary bit-streams of length nk=992 and 10-bit precision Sobol-based bit-streams of length $2^{10} = 1024$, unless stated otherwise. We stop processing of bit-streams in the MUX-based and Sobol$_{\texttt{OR}}$ designs after n^2 cycles (here, 1024 cycles). Although this introduces some truncation inaccuracy [5] in the computation, the required number of processing cycles to produce high accuracy results with these MAC designs exponentially increase with increasing the number of inputs, which is not feasible in practice. We discuss the differences in the latency of different SC MAC designs in more detail in Section VIII.

Table II compares the mean absolute error (MAE) (in percent) of the implemented MAC designs for different ranges of input values. We stop increasing the range of input values when the reference output reaches 1. The reference computes on double precision and is listed in the first row. Sobol$_{\texttt{MUX}}$-rescaled is equivalent to Sobol$_{\texttt{MUX}}$ but with compensation of the scaling inherent to the MUX-based adder. In a second processing step, after converting back to positional binary representation, the result of the scaled Sobol$_{\texttt{MUX}}$ is multiplied by N to get the correct order of magnitude (Sobol$_{\texttt{MUX}}$-rescaled = Sobol$_{\texttt{MUX}}$ $\cdot N$). The same is true for Sobol$_{\texttt{MUX/TFF}}$-rescaled. All listed designs perform the multiplication part of the MAC operation accurately. Their summation part, however, can cause error. Sobol$_{\texttt{OR}}$ calculates the union of inputs and suffers from a systematic error when compared to reference sums. The MUX-based designs implement scaled MAC operations and hence have a systematic deviation to non-scaled MAC results. Random fluctuations in the select input [15] of Sobol$_{\texttt{MUX}}$ and the fact that the MUX unit discards $N-1$ bits (through multiplexing) each clock cycle leads to further accuracy loss. In general, Sobol$_{\texttt{MUX}}$-rescaled has the potential to produce accurate results for all input values and number of inputs as the systematic deviation of the Sobol$_{\texttt{MUX}}$ gets compensated when multiplying with N. However, it would take more than 1024 processing cycles to converge to the correct result.

As can be seen in Table II, the proposed Unary$_{\texttt{OR}}$ design is the only MAC design that can compute completely accurate results. It is significantly more accurate than the traditional $\texttt{OR}$-based approaches. When compared to Sobol$_{\texttt{OR}}$, the error decreases between 10% to 100% depending on the range and number of input values. The proposed design achieves a minimum error decrease of 37% for the case of processing four inputs in the $[0, 0.95]$ interval ($\frac{5.1-3.2}{5.1} \times 100 \approx 37\%$) and 100% decrease when the proposed method is exact.

When comparing the Unary$_{\texttt{OR}}$ technique to rescaled MUX-based methods the table shows two trends. The proposed Unary$_{\texttt{OR}}$ design performs better for small input ranges, and for medium input ranges with large number of inputs. For 24-input MAC the proposed method has lower error for input values in the $[0, 0.43]$ interval. The MAC error decreases by 100% for input values in the $[0, 0.29]$ interval and decreases by a minimum of $\frac{4.1-3.6}{4.1} \times 100 = 12\%$ for input values in range $[0.0.43]$. In contrast, higher error is observed for input values greater than 0.45. For 12 inputs, Unary$_{\texttt{OR}}$ is more accurate than both MUX-based MAC designs for input values less than 0.55. For four inputs, Sobol$_{\texttt{MUX/TFF}}$-rescaled, in most cases, computes more accurate results than Unary$_{\texttt{OR}}$ and Sobol$_{\texttt{MUX}}$-rescaled. The proposed Unary$_{\texttt{OR}}$ is more accurate than Sobol$_{\texttt{MUX}}$-rescaled for inputs values less than 0.71. Nevertheless, considering the first row of Table II, if the mean reference result is below 0.25, our proposed technique achieves either exact results or a mean absolute error close to zero.

To give an example for one table entry, a mean reference result of 0.125 with two summands ($N = 2$) requires uniformly distributed random variables in range of 0 to 0.5 as input. In that case, the inputs have an expected value of $E(x) = 0.25$. Since the maximum input for this case is 0.5, the proposed method will be exact as shown in Table II and Fig 7.

$$z = E(x_1)E(y_1) + E(x_2)E(y_2) \overset{!}{=} 0.125 \tag{93}$$

$$\rightarrow x_1, y_1, x_2, y_2 \sim \mathcal{U}(0, 0.5) \tag{94}$$

$$z = 2 \cdot 0.25^2 = 0.125 \tag{95}$$

In the last four columns of Table II, the range of inputs for $N = 2$ exceeds the $[0, 1]$ interval. The upper bound (i.e., 1) gives a maximum input mean of $E(x) = 0.5$, when the input values are uniformly distributed. The maximum mean reference of the MAC output is therefore $z = \sum_{i=1}^{N} x_i y_i = 0.5^2 + 0.5^2 = 0.5$. However, we need outputs of greater than 0.5 to evaluate the accuracy for full possible range of the results. To provide inputs with mean value greater than 0.5, we generate uniformly distributed random values in the $[0, 1.05]$, $[0, 1.14]$, $[0, 1.22]$,$[0, 1.31]$ and $[0, 1.38]$ intervals, and round the values down when the generated input is greater than 1. This way, the mean of inputs increases to values greater than 0.5 as input values close or equal to 1 occur more often. For example, when we generate 13 uniformly distributed random values within the $[0, 1.2]$ interval we get a mean of 0.6, with on average two values greater than 1.0. If we clip the input values to 1.0, the mean value becomes 0.57. The disadvantage of clipping is a deviation from equal spreading of input values. However, it allows using the same RNG in all evaluations.

Fig. 9 shows the MAE of the proposed technique for different period lengths (n) and number of summands (N) over the expected reference result (marked on the X-axis). The inputs are scaled, uniformly distributed random values equal to the numbers in Table II. As it can be seen, the period length and the number of summands have negligible impact on the accuracy. The MAC error primarily depends on the value of the reference result. Note that, the input-output relation of MAC is linear, due to the linearity of both multiplication and addition. We recall the threshold (92) for input values and accurate computation derived in Section IV. Fig. 9 shows that the accuracy threshold for the MAC result is approximately 0.25. This can also be seen in the third column of Table II, with the mean reference MAC output listed in the first row. Beyond this exact threshold the error increases quadratically. This matches the quadratic increase of E_{upper} derived in Section V. When the number of inputs increases, the input ranges need to be decreased to keep the mean reference result in the same interval and achieve a similar accuracy with the case of fewer inputs. We show the impact of increasing the number of inputs on accuracy in Table III, where we compare Unary$_{\texttt{OR}}$, Sobol$_{\texttt{OR}}$, Sobol$_{\texttt{MUX/TFF}}$-rescaled and Sobol$_{\texttt{MUX}}$-rescaled for different bit-stream lengths and number of inputs. For the purpose of not exceeding an output of 1 we limit the input range for $2N-2$ MAC inputs to the $[0, 0.25]$ interval (the total number of MAC inputs is $2N$). The remaining two inputs have values in the $[0, 1]$ interval. As a result, the MAC circuit needs to sum one summand with average of 0.5^2=0.25 with N-1 summands with average of 0.125^2. As can be seen in Table III, all implementations except Sobol$_{\texttt{MUX/TFF}}$-rescaled achieve a comparable MAE as in Table II. Sobol$_{\texttt{MUX/TFF}}$-rescaled produces error if the input values are significantly different ($x >> y$, or $y << x$). For example, in Fig. 8, assume that x is close to 0 and y close to 1. Since the select input is 0 most of the time, the result is $z \approx y$ rather than $z = \frac{x+y}{2}$. Unary$_{\texttt{OR}}$ halves the MAE of conventional $\texttt{OR}$-based approaches and in most cases has a higher error than Sobol$_{\texttt{MUX}}$-rescaled. However, the MUX-based approaches require adjustment of the scaling factor, whereas Unary$_{\texttt{OR}}$ already provides a non-scaled result. Our evaluations confirm that the proposed method is accurate, even with large number of inputs, as long as the input values are small enough.

TABLE III: MAE (%) comparison for different number of summands (N) and input bit-stream lengths (n). One summand is in [0,1] interval and N-1 summands are below 0.25^2.

Reference MAC Output	0.26	0.33	0.33	0.42	0.43	0.55	0.55	0.70
n	16		32		64		128	
# Inputs ($2N$)	4	12	12	24	24	40	40	60
Unary$_{\texttt{OR}}$	0.5	1.5	1.3	2.4	2.4	4.1	4.3	7.6
Sobol$_{\texttt{OR}}$	0.5	2.9	2.3	5.7	5.3	10.3	10.3	17.7
Sobol$_{\texttt{MUX/TFF}}$-rescaled	0.5	7.9	8.0	8.0	8.0	14.9	15.0	3.4
Sobol$_{\texttt{MUX}}$-rescaled	2.2	5.3	2.7	4.5	2.2	3.3	1.6	2.3

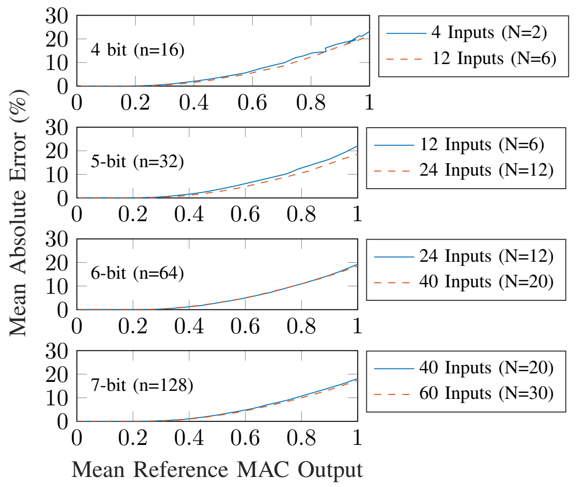


Fig. 9: The MAE (%) of Unary$_{\texttt{OR}}$ versus the mean reference output for different relative prime period lengths (n and $k = n - 1$) and number of MAC inputs. The inputs of the MAC are scaled, uniformly distributed random variables.

E. A Case Study: Gray-scale

To evaluate the proposed technique in an end-application, we perform gray scaling on true color images. Gray scaling is an image processing task that requires a MAC operation for each pixel. RGB values are converted to gray scale values by

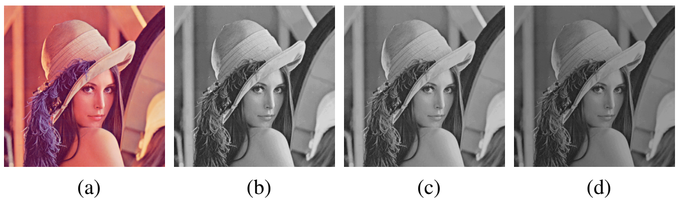


Fig. 10: (a) Original RGB image. Gray-scale using (b) reference design (c) $\text{Unary}_{\texttt{OR}}$ and (d) $\text{Sobol}_{\texttt{OR}}$

forming a weighed sum of the R, G and B components as in

$$Gray = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B. \quad (96)$$

Grayscale is well-suited for unipolar SC. All inputs and results are positive values in the [0,255] interval that be scaled to the [0,1] interval (as Q8 number format with eight fractional bits). The MAC circuit has six inputs ($N = 3$ summands), with three of them being constant and the other three changing with each pixel. The images are shown in Fig. 10 with sizes 512×512×3 for RGB and 512×512×1 for gray-scale. Fig. 10(a) shows the original image and Fig. 10(b) is the reference gray-scale result. The reference uses 5-bit precision inputs and exact arithmetic. Fig. 10(c) and (d) show the gray scale results for $\text{Unary}_{\texttt{OR}}$, and $\text{Sobol}_{\texttt{OR}}$ that also use 5-bit precision inputs and therefore ignore the three least significant bits of the 8-bit color values. Fig. 10(c) has a MAE of 1.6% and Fig. 10(d) has a MAE of 10.8% when compared to the output of the reference. The image in Fig. 10(d) appears darker than Fig. 10(c) as OR-based MAC result values are smaller than the results of the new $\text{Unary}_{\texttt{OR}}$ (as shown in Fig. 7 too) and smaller values mean less brightness. The average value of the pixels in the reference gray-scale image is 125 (=0.49 in Q8 number format). Table II lists a MAE of 2.6% for the proposed method in the case of six summands at a mean reference MAC output of 0.45. Since gray-scale only uses three summands instead of six, we can halve this table entry 2.6/2=1.3%. A MAE of 1.3% is close to the 1.6% error of the proposed $\text{Unary}_{\texttt{OR}}$ in this case study.

$\text{Unary}_{\texttt{OR}}$ and $\text{Sobol}_{\texttt{OR}}$ require almost the same number of clock cycles per pixel. $\text{Unary}_{\texttt{OR}}$ finishes after 1012 cycles while $\text{Sobol}_{\texttt{OR}}$ stops after $n^2 = 1024$ cycles. The results in Fig. 10(c) can be computed efficiently with $\text{Unary}_{\texttt{OR}}$ and $\text{Unary}_{\texttt{OR/REG}}$. We refer the readers to Section VI-B and Section VII for an analysis of both implementations and a resource consumption comparison.

VII. Resource Comparison

In this section, we compare area and power consumption of the state-of-the art methods with different implementations of the proposed MAC technique. A SC system often consists of some SNGs, a stochastic circuit that does the actual computation and a probability estimator (for bit-stream to binary conversion). We exclude the overhead of SNGs in our evaluations as they are already discussed in the literature extensively [2], [3], [5]. We also exclude the overhead cost of compensating the scaling for the MUX-based methods (multiplying by N). Therefore, we don't differentiate between $\text{Sobol}_{\texttt{MUX}}$ and $\text{Sobol}_{\texttt{MUX}}$-rescaled (same for $\text{Sobol}_{\texttt{MUX/TFF}}$ and $\text{Sobol}_{\texttt{MUX/TFF}}$-rescaled). The resource comparison includes the stochastic circuit, a minimal control logic (receives start signals and transmits a done signal after completion) and the probability estimator unit. All flipflops are synchronous with synchronous resets. We synthesized the designs using the Synopsys Design Compiler v2018.06 with a 45nm gate library. The designs were synthesized for 100MHz frequency.

Table IV lists the area and power consumption of all six implementations, with three of them being based on the proposed MAC, for different number of inputs and input precisions. The comparison shows that the implementations without delay registers are significantly more efficient in both area and power than implementations with registers ($\text{Unary}_{\texttt{OR/REG}}$ and $\text{Unary}_{\texttt{OR/SEQ}}$). The exception is $\text{Unary}_{\texttt{OR/REG}}$ with four inputs, as it requires comparable area and power but can be implemented with fewer RNGs (see Section VI-B). Note that, $\text{Unary}_{\texttt{OR/REG}}$ is also viable for six inputs, because for four and six inputs the proposed method only requires *minor* relative delays. We refer to Section III-B for the definition of minor delays and to Fig. 3 (first tree subplots) for an example of minor delayed summands. All implementations except $\text{Unary}_{\texttt{OR/SEQ}}$ require approximately the same computation time. So, the ratio of the required energy is similar to the ratio of the required power. However, this is not true for $\text{Unary}_{\texttt{OR/SEQ}}$. HDL simulations show that the sequential implementation requires 2×, 7.8×, and 17.9× more computation time than the Sobol-based methods for 4, 12 and 24 inputs, respectively. This is primary due to the sequential processing of inputs and secondary due to the stalled bit-stream generation, which is required to achieve the same accuracy as $\text{Unary}_{\texttt{OR/REG}}$ and $\text{Unary}_{\texttt{OR}}$. As a result of the longer processing time, the energy consumption is significantly higher for the $\text{Unary}_{\texttt{OR/SEQ}}$ design than for the other implementations. The resource consumption of $\text{Sobol}_{\texttt{OR}}$, $\text{Sobol}_{\texttt{MUX}}$, and $\text{Unary}_{\texttt{OR}}$ hardly changes for different accumulator sizes and bit-stream lengths. The reason is that the reported numbers does not include the cost of the SNGs, and the changes only show different counter bitwidths, additional parallel AND gates for multiplication, and more inputs to the OR gate (or the MUX unit) for summation. $\text{Sobol}_{\texttt{MUX/TFF}}$ shows more resource consumption with increasing MAC size. This is because it needs additional toggle flipflops and XOR gates to compute the MUX's select bit-stream (see Fig. 8). The required delay registers of $\text{Unary}_{\texttt{OR/REG}}$ increases heavily with the input count as each additional summand requires a larger delay register. In contrast, $\text{Unary}_{\texttt{OR/SEQ}}$ has a high base resource consumption which barely changes with the number of inputs as the overall feedback buffer size is similar for all input counts. Since the number of delay elements heavily depends on the bit-stream lengths, both $\text{Unary}_{\texttt{OR/SEQ}}$ and $\text{Unary}_{\texttt{OR/REG}}$, require significantly less resources for $n = 16$ than for $n = 32$.

Between the three implementations of the proposed technique (i.e., $\text{Unary}_{\texttt{OR}}$, $\text{Unary}_{\texttt{OR/SEQ}}$, $\text{Unary}_{\texttt{OR/REG}}$), $\text{Unary}_{\texttt{OR}}$ is selected if the number of MAC inputs is greater than six and counters for bit-stream generation can be shared between multiple parallel MAC units. If counters can not be shared, or the number of MAC inputs is less than or equal six, $\text{Unary}_{\texttt{OR/REG}}$ could be more efficient, when taking bit-stream generation into account. $\text{Unary}_{\texttt{OR/SEQ}}$ uses the MAC architec-

TABLE IV: Area (μm^2) and Power consumption (μW) for three state-of-the-art SC MAC designs and for three implementations of the proposed MAC technique.

		4 Inputs		12 Inputs		24 Inputs		40 Inputs		60 Inputs	
		Area	Power	Area	Power	Area	Power	Area	Power	Area	Power
n=32	$\text{Sobol}_{\texttt{OR}}$	413	1.24	429	1.67	450	2.16	487	2.74	521	3.52
	$\text{Sobol}_{\texttt{MUX/TFF}}$	434	1.52	525	3.15	660	5.55	840	8.77	1069	12.7
	$\text{Sobol}_{\texttt{MUX}}$	428	1.46	461	1.85	528	2.82	574	3.43	643	4.65
	$\text{Unary}_{\texttt{OR}}$	451	1.44	520	1.75	542	1.71	664	2.07	743	2.01
	$\text{Unary}_{\texttt{OR/REG}}$	596	1.48	11026	15.5	33755	47.9	62586	89.5	103730	0.147
	$\text{Unary}_{\texttt{OR/SEQ}}$	10891	14.7	14355	19.6	16435	22.6	17154	23.4	17878	24.6
n=16	$\text{Sobol}_{\texttt{OR}}$	385	1.21	389	1.50	403	2.32	426	2.75	481	3.92
	$\text{Sobol}_{\texttt{MUX/TFF}}$	405	1.62	495	4.11	585	4.93	765	8.16	993	12.4
	$\text{Sobol}_{\texttt{MUX}}$	389	1.20	416	1.64	439	2.25	506	3.26	581	4.59
	$\text{Unary}_{\texttt{OR}}$	444	1.40	502	1.76	502	1.72	557	1.94	640	2.49
	$\text{Unary}_{\texttt{OR/REG}}$	490	1.39	3246	5.26	9114	14.1	16606	27.9	16470	28.9
	$\text{Unary}_{\texttt{OR/SEQ}}$	3025	4.09	4012	6.35	4465	5.31	4717	6.87	4572	6.70

ture in Fig. 2(a) and is not efficient enough to compete with the parallel architecture in Fig. 2(b), as the cost for large shift-registers outweighs potential savings. The proposed $\text{Unary}_{\texttt{OR}}$ design requires, on average, 21% more area than $\text{Sobol}_{\texttt{OR}}$ due to the overhead of stalling the SNGs to make relative delays between bit-streams. Further, $\text{Unary}_{\texttt{OR}}$ and $\text{Sobol}_{\texttt{OR}}$ require similar power and energy due to similar processing time. The main difference is their distinct bit-stream generation. Both $\text{Unary}_{\texttt{OR}}$ and $\text{Sobol}_{\texttt{OR}}$ require N SNGs, but $\text{Unary}_{\texttt{OR}}$ requires counters as RNG while $\text{Sobol}_{\texttt{OR}}$ requires costly Sobol sequence generators [20]. Without the costs for SNGs, $\text{Sobol}_{\texttt{MUX}}$ consumes slightly less and $\text{Sobol}_{\texttt{MUX/TFF}}$ more resources than $\text{Unary}_{\texttt{OR}}$. However, we emphasize that MUX-based circuits compute scaled results. Rescaling by N is not possible in unipolar SC and the resource consumption of the weighed binary multiplier is not included in this analysis.

VIII. Further Discussions

In this section, we discuss three properties of the proposed technique in more detail. First, we examine the implications of N (the number of possible summands) being pronic. Second, we discuss the consequences of requiring bit-streams with relative prime period lengths as inputs. Finally, we take a closer look at the convergence behavior of the proposed technique compared to the state-of-the-art methods.

1) In Section IV, we derived that the maximum allowed number of summands is a pronic number ($N = 2, 6, 12, 20, 30 \ldots$). The maximum allowed number of inputs for the MAC unit is twice of that ($2N = 4, 12, 24, 40, 60 \ldots$). The number of inputs is often given by the application. The proposed method also works for the number of inputs that are not pronic. In that case, it is possible to use parts of the delays from Algorithm 1 and leave the other unused. Then, the accuracy threshold (92) and error (Table II) is similar as if all possible inputs are used. Compared to prior methods, the proposed design performs best when the number of inputs equals the maximum allowed from (3).

2) The proposed MAC technique of this work guarantees uncorrelated inputs by using relatively prime bit-stream lengths. The input bit-streams to the `AND` gates use relative prime period lengths of n and k ($n = k + 1$). Thus, one input value needs to be converted to a unary bit-stream with a slightly higher resolution than the other one. This difference in the representation introduces a systematic quantization inaccuracy during bit-stream generation particularly for small values of n and k. For example, the bit-stream representation for 0.5 with period lengths $n = 8$ and $k = 7$ is 11110000 and 1111000, respectively. When converting back to positional binary the values are $\frac{v}{n} = \frac{4}{8} = 0.5$ and $\frac{v}{k} = \frac{4}{7} = 0.5714$. We claim that the proposed MAC design is deterministic and accurate because its inaccuracy is predictable, systematic and occurs at the bit-stream generation and not in the computation circuit.

The maximum number of 1s in a stochastic bit-stream is limited by the length of bit-stream. A relative delay between input bit-streams increases the length of the output bit-stream. In the proposed MAC design, the output bit-stream has up to $n \cdot k$ (the LCM of input lengths) plus $N_{major} \cdot vn + N_{minor} \cdot v$ (the maximum delay from Algorithm 1) bits. Assuming that the maximum value (i.e., 1.0) is represented by $n \cdot k$ bits of 1s, the expanded bit-stream result may represent a value greater than 1, which needs to be considered when converting the output back to the conventional weighted binary representation.

3) The number of processing cycles (i.e., latency) is different for different stochastic MAC designs. The first stage of MAC operation is multiplication which is similarly implemented in all MAC designs using standard `AND` gates. Different MAC designs, however, are different in the second stage of MAC operation which accumulates the multiplication results. The inputs of each `AND` gate are two uncorrelated bit-streams. For highest accuracy the inputs are 2^{2B} bit long when multiplying two B-bit precision input values [5]. In Section VI, we choose $B = 5$, so the multiplication latency is $2^{10} = 1024$ cycles (assuming each bit of the input bit-streams are processed in one cycle) for the Sobol-based designs and $n \cdot k = 992$ cycles for the proposed design.

In $\text{Sobol}_{\texttt{OR}}$, the `OR`-based addition converges to the union of its inputs after $2^{N \cdot 2B} = 2^{N \cdot 10}$ cycles where N is the number of summands. MUX-based MAC converges to the scaled addition of the summands after $2^{2 \times 2B} = 2^{2 \times 10}$ cycles because the select input of the MUX must be uncorrelated to the outputs of multiplications. The summation stage of the proposed $\text{Unary}_{\texttt{OR}}$ MAC technique does not introduce any additional latency besides the extra cycles from the relative delays. $\text{Unary}_{\texttt{OR}}$ requires exactly nk, the LCM of inputs, plus $N_{major} \cdot vn + N_{minor} \cdot v$ cycles, the maximum delay from Algorithm 1 to reach its maximum accuracy (the maximum latency is less than $2nk$ cycles). The MAE results of the MUX-based methods in Section VI improve with longer processing times. In contrast, running the proposed MAC technique for the exact number of cycles is important, because the accuracy decreases if the computation is stopped too early or too late.

IX. Conclusions

In this work, we proposed a novel SC MAC technique based on deterministic unary bit-streams. We showed that the proposed design can compute completely accurate non-scaled MAC and calculates overall more accurate results than the `OR`-based MAC design fed with Sobol-based LD bit-stremas. It also achieves lower error compared to the rescaled MUX-based MAC designs, except for the case of processing a small number of large inputs where the inherent scaling of MUX units is beneficial. We provided practical implementations that show the proposed technique is suitable for applications with low input counts that require exact computation as well as large accumulator sizes that can tolerate small errors. Modification of the proposed algorithm and using additional logic gates are potential solutions to further increase the threshold for exact MAC operation and to enhance the accuracy for large input values. Such solutions however require further investigation and are part of our future work.

Acknowledgements

This work was supported in part by the Louisiana Board of Regents Support Fund no. LEQSF(2020-23)-RD-A-26 and National Science Foundation grant no. 2019511.

References

[1] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pp. 37–172. Springer US, 1969.

[2] A. Alaghi *et al.* The Promise and Challenge of Stochastic Computing. *IEEE Trans. on Computer-Aided Design of Integ. Circuits and Systems*, 37(8):1515–1531, Aug 2018.

[3] W. Qian *et al.* An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, Jan 2011.

[4] W. Poppelbaum *et al.* Unary processing. In *Advances in Computers*, volume 26, pp. 47 – 92. Elsevier, 1987.

[5] M. H. Najafi *et al.* Performing Stochastic Computation Deterministically. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, 2019.

[6] M. H. Najafi *et al.* Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Trans. on VLSI Systems*, 26(8):1471–1480, Aug 2018.

[7] M. H. Najafi *et al.* Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[8] D. Jenson and M. Riedel. A deterministic approach to stochastic computation. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[9] S. R. Faraji and K. Bazargan. Hybrid Binary-Unary Hardware Accelerator. *IEEE Trans. on Computers*, 69(9):1308–1319, 2020.

[10] S. A. Faraji *et al.* HBUNN - Hybrid Binary-Unary Neural Network: Realizing a Complete CNN on an FPGA. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pp. 156–163, 2019.

[11] S. Mohajer *et al.* Parallel Unary Computing Based on Function Derivatives. *ACM Trans. Reconfigurable Technol. Syst.*, 14(1), October 2020.

[12] A. Alaghi and J. Hayes. Exploiting correlation in stochastic circuit design. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pp. 39–46, Oct 2013.

[13] V. T. Lee *et al.* Correlation manipulating circuits for stochastic computing. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1417–1422, 2018.

[14] V. T. Lee *et al.* Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 13–18, March 2017.

[15] P. Ting and J. P. Hayes. Eliminating a hidden error source in stochastic circuits. In *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 1–6, 2017.

[16] B. Li *et al.* Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pp. 36–41, New York, NY, USA, 2016. ACM.

[17] B. Yuan and Y. Wang. High-accuracy fir filter design using stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 128–133, 2016.

[18] P. Ting and J. P. Hayes. Stochastic logic realization of matrix operations. In *2014 17th Euromicro Conference on Digital System Design*, pp. 356–364, 2014.

[19] H. Sim and J. Lee. Cost-effective stochastic mac circuits for deep neural networks. *Neural Networks*, 117:152–162, 2019.

[20] S. Liu and J. Han. Energy efficient stochastic computing with sobol sequences. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 650–653, 2017.

[21] Wong, Ming Ming *et al.* A new stochastic inner product core design for digital fir filters. *MATEC Web Conf.*, 125:05006, 2017.

[22] Y. Chang and K. K. Parhi. Architectures for digital filters using stochastic computing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2697–2701, 2013.

[23] J. A. Dickson *et al.* Stochastic arithmetic implementations of neural networks with in situ learning. In *IEEE International Conference on Neural Networks*, pp. 711–716 vol.2, 1993.

[24] M. H. Najafi *et al.* Deterministic Methods for Stochastic Computing using Low-Discrepancy Sequences. In *Proceedings of the 37th International Conference on Computer-Aided Design*, ICCAD '18, 2018.

[25] S. Liu and J. Han. Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1326–1339, July 2018.

[26] B. Moons and M. Verhelst. Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(4):475–486, Dec 2014.

Peter Schober received his B.Sc. degree in Electronics and Information Technology from Johannes Keppler University, Linz, Austria, in 2017. He is currently a Master's student in Embedded Systems at the TU Wien (formerly known as Vienna University of Technology as well), Vienna, Austria. From September to December 2019, he was a visiting research scholar at the School of Computing and Informatics, University of Louisiana, LA, USA.

M. Hassan Najafi received the B.Sc. degree in Computer Engineering from the University of Isfahan, Iran, the M.Sc. degree in Computer Architecture from the University of Tehran, Iran, and the Ph.D. degree in Electrical Engineering from the University of Minnesota, Twin Cities, USA, in 2011, 2014, and 2018, respectively. He is currently an Assistant Professor with the School of Computing and Informatics, University of Louisiana, LA, USA. His research interests include stochastic and approximate computing, unary processing, in-memory computing, and machine-learning. In recognition of his research, he received the 2018 EDAA Outstanding Dissertation Award, the Doctoral Dissertation Fellowship from the University of Minnesota, and the Best Paper Award at the ICCD'17.

Nima TaheriNejad (S'08-M'15) received his Ph.D. degree in electrical and computer engineering from The University of British Columbia (UBC), Vancouver, Canada, in 2015. He is currently a "Universitätsassistent" at the TU Wien, Vienna, Austria, where his areas of work include cyber-physical embedded systems, computer architecture, computational self-awareness, in-memory computing, systems on chip, and health-care. He has published two books and more than 60 peer-reviewed articles. He has also served as a reviewer, an editor, an organizer, and the chair for various journals, conferences, and workshops. Dr. Taherinejad has received several awards and scholarships from universities and conferences he has attended.