From Stochastic to Bit Stream Computing: Accurate Implementation of Arithmetic Circuits and Applications in Neural Networks

Ensar Vahapoglu and Mustafa Altun

***Abstract*—In this study, we propose a novel computing paradigm "Bit Stream Computing" that is constructed on the logic used in stochastic computing, but does not necessarily employ randomly or Binomially distributed bit streams as stochastic computing does. Any type of streams can be used either stochastic or deterministic. The proposed paradigm benefits from the area advantage of stochastic logic and the accuracy advantage of conventional binary logic. We implement accurate arithmetic multiplier and adder circuits, classified as asynchronous or synchronous; we also consider their suitability of processing successive streams. The proposed circuits are simulated both in gate level and in transistor level with AMS 0.35µm CMOS technology to show the circuits' potential for practical use. We thoroughly compare the proposed adders and multipliers with their predecessors in the literature, individually and in a neural network application. Comparisons made in terms of area and accuracy clearly favor the proposed designs. We believe that this study opens up new horizons for computing that enables us to implement much smaller yet accurate arithmetic circuits compared to the conventional binary and stochastic ones.**

***Index Terms*—Stochastic computing, bit stream computing, arithmetic circuits, neural network.**

I. Introduction

STOCHASTIC computing (SC), first brought forward in 1960s [2], [3], performs serial data processing with Binomially distributed bit streams. Each stream represents a probability value, obtained as the number of 1 valued bits over the total number of bits. Thus, it is possible to use $n+1$ different states with a single input/output, corresponding to $n+1$ different values ranging from $0/n$ to n/n where n is the total number of bits in a stream. On the other hand, conventionally a binary input/output has two states that are logic 0 or logic 1. This feature offers an important area advantage for SC, especially for arithmetic operations. For example, a single AND gate is used for stochastic multiplication. This is illustrated in Fig. 1. Here, input streams have values of 1/2, so an output value of 1/4 is expected. Although the correct result can be achieved as in Fig. 1 a), it is not guaranteed for different cases since 1's and 0's in streams are randomly positioned in SC. Fig. 1 b) shows an erroneous result with an output value

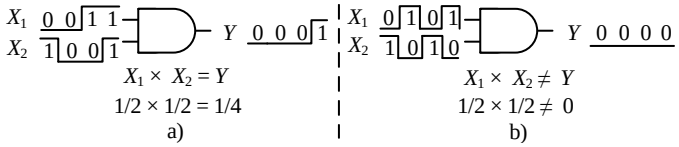

Fig. 1: Stochastic multiplication with an AND gate having: a) accurate results, and b) inaccurate results.

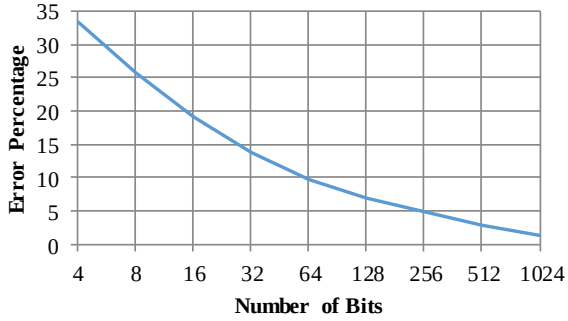

Fig. 2: Average error percentage for an AND gate with respect to the number of bits (n) in a stream; inputs probability values are both $1/2$.

of 0/4. Here, the relative standard error is 100%. Note that we represent streams such that the bit on the leftmost is the first to be processed.

Indeed, accurate (error-free) computation is impossible for SC. Since streams should be always Binomially distributed, to achieve zero error or zero standard deviation, infinite number of bits are needed. Fig. 2 shows how the average error changes with the number of bits for an AND gate having input values of 1/2. Here, to achieve 10% and 1% errors, streams having more than 100 and 1000 bits are needed that is not practical in terms of the computing time. This explains why SC could not become a real competitor to conventional computing although it offers significant area advantage [4]. Low accuracy or long computing times is the main obstacle in front of SC and the main motivation of this study.

In this paper, we propose a novel computing paradigm "Bit Stream Computing (BSC)" that is constructed on the logic used in SC, but does not necessarily employ randomly or Binomially distributed input/output bit streams as SC does. Any type of streams can be used either stochastic or deterministic. The proposed paradigm benefits from the area advantage of stochastic logic and the accuracy advantage of conventional

* This work is supported by the TUBITAK-1001 project #116E250 and Istanbul Technical University BAP (ITU-BAP) project #40781.

* E. Vahapoglu and M. Altun are with the Department of Electronics and Communication Eng., Istanbul Technical University, Istanbul, Turkey, 34469.

* E-mails: {vahapoglu, altunmus}@itu.edu.tr

* A preliminary version of this paper appeared in [1]

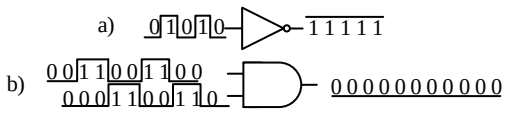

Fig. 3: Examples of timing problems for a bit duration of X: a) inverter has a delay of X, b) misalignment of inputs by X.

binary logic. With BSC, we successfully implement accurate arithmetic multiplier and adder circuits.

Along with the accuracy issue that is kept on the agenda since the birth of SC, there is another annoying difficulty for SC as well as for BSC: timing problems. They happen mainly due to undesirable changes in the duration of 1's and 0's in a bit stream. Two examples are shown in Fig. 3. Here, time duration of a bit is X. The problem in Fig. 3 a) is a result of the fact that a logic gate can not have a delay equal and larger than X. Indeed, depending on the used technology the delay should generally be much smaller than X. Fig. 3 b) shows a misalignment problem; if inputs were perfectly aligned, then the output stream would have two 1 valued bits, but it does not have any. Considering the severity of these and similar types of timing problems, any circuit design technique developed for SC or BSC should be justified with timing considerations. For this purpose, we test the proposed circuits with transistor level simulations.

A. Previous Works and Contributions

The mainstream solution to improve accuracy in SC is manipulating input bit streams by either decreasing their randomness or making them dependent/correlated. For this purpose, pseudo-random and quasi-random number generators are proposed [5], [6]. Pseudo-random generators generally use LFSR's (Linear Feedback Shift Register) that even allow to produce desired orders of 0's and 1's resulting in perfect accuracy [6]. Additionally, quasi-random generators producing low-discrepancy bit streams can decrease error rates [7]. There are also recent studies exploiting correlation for accuracy [8], as well as using fully deterministic generators [9], [10]. In [9] and [10], accurate arithmetic operations are achieved. Nevertheless, for all of these studies an input bit stream generator is needed for each input. Therefore, the number of generators are linearly dependent with the number of inputs, and these generators generally consume a majority of the circuit area. In [10], smaller deterministic generators than random ones are used, but still each input should have its own generator including a clock generation circuitry.

Another important drawback of the mentioned studies is that they are not suitable for multi-level designs. Outputs of one level can not be directly used as inputs of another level. For example, outputs of two AND gates can not be directly used as inputs of another AND gate. Outputs should be recreated to fit the desired format, and this is quite costly. In [10] the authors discuss this problem. The generated input signals, called as PWM signals in the paper, loose their formation at the output. They propose a solution for this, but it requires extra control inputs, so new streams are needed to be generated. This does not just worsens the design complexity, but it also decreases the speed dramatically.

As opposed to the studies focusing on the generation of bit streams in desired formats, our treatment BSC does only care about the values carried by the streams, so any type of input bit streams can be directly used. This eliminates the need of specific stream generators. Furthermore, there is no extra cost for multi-level designs; output streams of one level can be directly used as inputs. In the literature, using the same logic, an accurate adder is proposed in [11], called as Alaghi adder in the paper. One of our two proposed synchronous adders is quite similar to the Alaghi adder with additional considerations for timing. Also we show that our adder can be generalized for any number of inputs. Furthermore, along with the adders, we propose two synchronous multipliers.

All of the above mentioned designs with an aim of improving accuracy, use clock signals so they are synchronous. To eliminate the cost of synchronization, we also propose asynchronous adders and multipliers mainly constructed on delay elements. Another shortcoming of these designs is their inability to process successive input bit streams; they are assumed to perform one-time operations. To overcome this shortcoming, we propose an adder and a multiplier that can successively process input bit streams.

Apart from the mentioned shortcomings, underestimating the timing problems is a general tendency in the literature. These problems are indigenous to bit stream operations in SC and BSC, and without solving them it is hard to claim the feasibility of the proposed study. For example, in [10], the authors claim to work with 1GHz input signals. Suppose that 8 bit binary equivalent operations are performed, so there should be at least 256 different values for streams. Therefore, for the worst-case scenario to represent the value of 1/256, a bit stream has a 1 valued bit with a duration of $1/(256 \times 10^9)$ seconds. It means that the proposed circuits should safely process 0.256 THz signals that does not seem to be possible (recall the case in Fig. 3 a)). Therefore, much slower operations should be used that also causes dramatic area increase for this study (justified in the experimental results section). Similar problems exist in almost all studies in the literature; unless using bit streams with very low frequencies such that duration a bit in a stream is much larger than the circuit's delay, the proposed designs do not work properly. For this reason, we have designed all of the proposed circuits in transistor level with timing simulations.

B. Overview

Total of three adders and three multipliers performing BSC are proposed. While one adder and one multiplier are asynchronous, the rest are synchronous. Among four synchronous adders and multipliers, one adder and one multiplier are able to process successive input bit streams. We evaluate all of the proposed designs with their predecessors by performing transistor level simulations with AMS 0.35μm CMOS technology. The proposed circuits are also tested in a neural network application.

The rest of paper is lined up as follows. Section II is composed of definitions, explanations, and limitations for BSC. In Section III and Section IV, we respectively introduce our

asynchronous and synchronous circuits performing accurate arithmetic operations. In Section V, we give experimental results to evaluate the proposed circuits. Section VI concludes this work with future directions.

II. Preliminaries

We start with yellow a few definitions. We define bit width as the time duration of a single bit in a stream, and stream length as the number of bits in the stream.

Improving accuracy in SC has a fundamental limit as explained in the following theorem.

Theorem 1. *Consider a system with ideal elements performing ideal SC. Accuracy of the system only depends on the expected output values z_e's and the output stream lengths n.*

Proof. In SC, z_e can also be defined as the probability that each output bit takes a logic 1 value. Therefore, each output bit has a Bernoulli distribution and the output stream has Binomial distribution ($p = z_e$). The standard error (standard deviation) and the relative error can be calculated as $\sqrt{\frac{p\times(1-p)}{n}}$ and $\sqrt{\frac{(1-p)}{p\times n}}$, respectively; both only depend on $p = z_e$ and n. □

This relatively simple theorem tells us that 1) accurate computation is impossible with SC that needs infinite stream lengths; 2) increasing stream lengths, accordingly computing times, X times results in a decrease in error values by only $\sqrt{X}$ times that is not efficient; and 3) in order to achieve high accuracy, randomness in output bit streams should be sacrificed.

Motivated by these inferences, we introduce a novel computing paradigm "Bit Stream Computing (BSC)". It benefits the logic used in SC, but does not necessarily employ randomly or Binomially distributed input/output bit streams as SC does. Any type of streams can be used either stochastic or deterministic. We perform accurate arithmetic addition and multiplication operations with BSC by considering the constraints given below. Note that since values of bit streams are in the range of 0-1, addition should be scaled by averaging the values. Throughout the paper we simply use the word "addition" to refer "scaled addition" .

Lemma 1. *Consider two input bit streams with lengths of n. Suppose that the streams take $n+1$ values between $0/n$ and n/n. Accurate addition of the inputs with BSC requires an output bit stream with a minimum length of $2 \times n$.*

Proof. Consider a worst case scenario for which one input takes the value of $1/n$ and the other one takes $0/n$. The output value should be $1/(2 \times n)$ that requires a length of $2 \times n$. □

Lemma 2. *Consider two input bit streams with lengths of n. Suppose that the streams take $n+1$ values between $0/n$ and n/n. Accurate multiplication of the inputs with BSC requires an output bit stream with a minimum length of n^2.*

Proof. Consider a worst case scenario for which both inputs take the value of $1/n$. The output value should be $1/n^2$ that requires a length of n^2. □

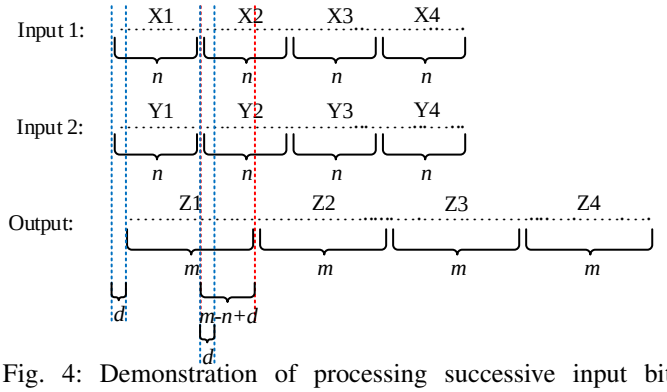

Fig. 4: Demonstration of processing successive input bit streams.

Theorem 2. *Consider a system performing BSC such that input and output stream lengths are n and m, respectively where $n < m$. If the system's current reaction time or delay is independent of the past, it cannot correctly process successive input streams.*

Proof. Suppose that the system has a delay of d bits (d may be fractional). It means that after applying input bit streams, the system needs to wait for a time duration of d bits to have the first output bit. Consider two sets of successive input bit streams. After the completion of the first set, the system needs more time equivalent to $m - n + d$ bits to have the output in full. However, we know that after the time duration of d bits, the output starts to have the results for the second input set. This is illustrated in Fig. 4. As a result, $m - n + d = d$, and $m = n$ should be satisfied to obtain correct results. □

Theorem 2 leads to two solutions for successive processing of bit streams. The first one is controlling the system's delay sequentially. For example in Fig. 4, for the first, the second, and the third set of input streams, the delay should be d, $d + m - n$, and $d + (2m - n)$ bits, respectively. Implementing such a complex and sequential system certainly kills the area advantage of BSC. The second solution is having same stream lengths for the inputs and outputs. This solution is much better not just for the area, but also for its suitability for multi-level designs. In this study, we use the second solution.

Using same stream lengths might result in inaccurate outputs. From Lemma 1 and Lemma 2, we know that we cannot achieve accurate addition and multiplication by using the same lengths if input bit streams are in full resolution meaning that they can take all possible values. For example, suppose that input and output stream lengths are 16, and multiplication is performed. If both inputs have values of 3/16, the correct result should be 9/256 or 0.5625/16, but we can only get either 0/16 or 1/16 from the output, so there is an error. To minimize the error, we round the output value to the nearest integer. In this example, the rounded result is 1/16. Considering this accuracy issue, we classify the proposed asynchronous and synchronous circuits as "Semi-accurate" that achieves successive stream processing and "Fully-accurate" that always give correct results. This is illustrated in Fig. 5. Note that fully-accurate ones do not apply aforementioned solution, i.e. they are not proper for successive processing.

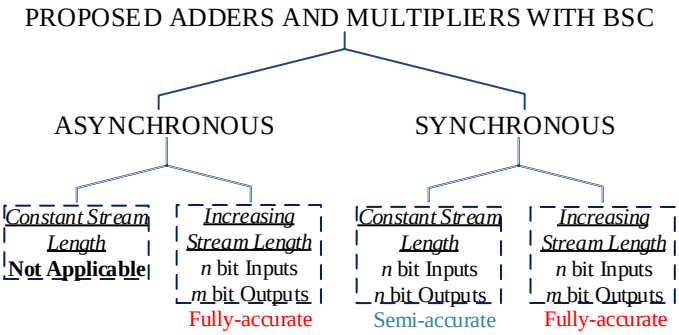

Fig. 5: Summary of the proposed adder and multiplier designs.

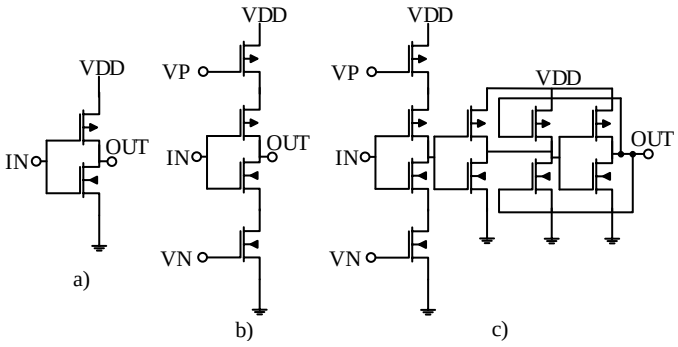

Fig. 6: Inverter as a delay element: a) conventional inverter, b) NP-voltage controlled inverter, and c) its cascaded version with a Schmitt trigger.

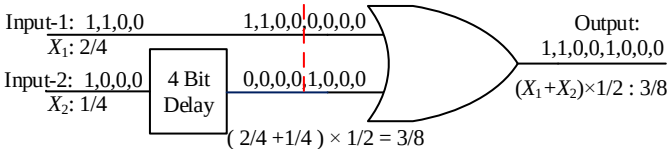

Fig. 7: The proposed asynchronous adder; $n = 4$.

III. Asynchronous Adders and Multipliers

First, we clarify why we do not use constant stream lengths for asynchronous circuits as stated in Fig. 5.

Lemma 3. *Consider a system performing BSC with multiple input streams being able take all different values from 0 to 1. If output stream lengths should be larger than those of inputs for full accuracy, there should be at least one case such that a bit change in the input stream causes multiple bit changes at the output stream.*

Proof. The proof is by contradiction. Suppose that a bit change in the input always results in at most one bit change at the output. Then output and input stream lengths should be equal for full accuracy that result in a contradiction. □

A direct result of Lemma 3 is that in order to use same input and output stream lengths for addition and multiplication operations, information of input bits should be stored to be used in the future. Since this sequential operation is costly for asynchronous circuits, we do not design circuits with constant stream lengths.

For our designs we should consider the following limitation.

Lemma 4. *Consider a fully-accurate asynchronous system performing BSC such that input and output stream lengths are n and m, respectively where $n < m$. The system should include at least $m-n$ input/output nodes of its internal circuit elements.*

Proof. At the time the output has its nth bit, the remaining $m-n$ bits should be kept in the system that necessitates $m-n$ nodes. An asynchronous system does not allow to store and reprocess the remaining information with less than $m-n$ nodes. □

Considering arithmetic operations and their stream length specifications previously given in Lemma 1 and Lemma 2, we conclude that at least n and n^2-n nodes are needed for fully-accurate adders and multipliers, respectively. Since logic gates are used as circuit elements and inverters are the most-area efficient ones, we use inverters as delay elements to keep $m-n$ bits in the system as stated in Lemma 4. By cascading even number of inverters, we can achieve a desired delay. To select a proper inverter structure, we consider three criteria: 1) its circuit area should be small in harmony with the area advantage of BSC; 2) its rise and fall times should be small to preserve signal integrity; and 3) it should be controllable to compensate for changes in delay values. Considering different options, three inverter structures come forward, shown in Fig. 6. Delay control of the conventional inverter, shown in Fig. 6 a), can be achieved by VDD scaling. For better control, VN and VP analog voltage inputs can be used as shown in Fig. 6 b). Additionally, to improve signal integrity, the NP controlled inverter can be cascaded with a Schmitt trigger as shown in Fig. 6 c) [12]. Among these three options, we prefer the first conventional one for the sake of simplicity and area efficiency.

A. Increasing Stream Length: Fully-accurate Addition

The proposed adder includes a delay block and an OR gate as shown in Fig. 7. The delay block is used to postpone one of the inputs with a delay amount of the time duration of the input stream that can be calculated as (input stream length n)×(bit width). To satisfy this amount and Lemma 4, we need to use at least n inverters if n is an even number, and $n+1$ inverters is n is an odd number. As a result the area complexity is $O(n)$.

B. Increasing Stream Length: Fully-accurate Multiplication

Accurate multiplication cannot be achieved unless each bit in one of the input streams is multiplied with each and every bit in the other one. Therefore, total of n^2 operations are needed, given that the inputs have n bits. We satisfy this by applying delays to the input streams for $2n-1$ different cases; 1 case for no delay, $n-1$ cases for delaying one of the inputs more than the other one, and $n-1$ cases for the opposite. After making multiplications with AND gates, an OR gate is used to combine the results. This is illustrated in Fig. 8 for $n = 3$. Here, there are total of 5 cases corresponding to 5 AND gates. Fig. 9 shows the circuit structure.

In general for n bit inputs, the circuit is constructed in four steps:

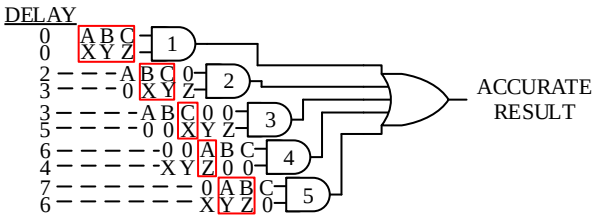

Fig. 8: Elucidation of the proposed asynchronous multiplier for 3 bit inputs.

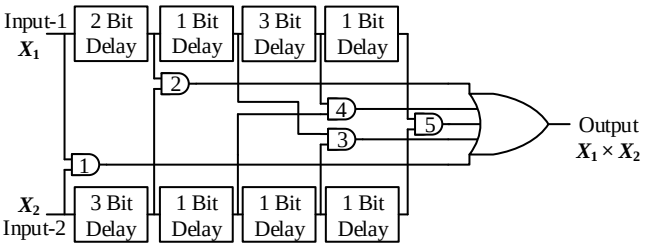

Fig. 9: The circuit structure of the proposed asynchronous multiplier for 3 bit inputs.

1) The inputs are ANDed without any delay (corresponding to the AND gate numbered 1 in Fig. 8).
2) Last $n-i$ bits of Input-1 and first $n-i$ bits of Input-2 are ANDed successively for $i = 1, 2, ..., n-1$, corresponding to the AND gates numbered 2 and 3 in Fig. 8. Total of $n-1$ AND gates are used for these operations with $n-1$ delay blocks for Input-1 and $n-1$ delay blocks for Input-2, corresponding to the 2 delay blocks in the upper part and the 2 delay blocks in the lower part of the circuit in Fig. 9.
3) First $n-i$ bits of Input-1 and last $n-i$ bits of Input-2 are ANDed successively for $i = n-1, n-2, ..., 1$, corresponding to the AND gates numbered 4 and 5 in Fig. 8. Total of $n-1$ AND gates are used for these operations with $n-1$ delay blocks for Input-1 and $n-1$ delay blocks for Input-2, corresponding to the 2 delay blocks in the upper part and the 2 delay blocks in the lower part of the circuit in Fig. 9.
4) Outputs of the all $2n-1$ AND gates are ORed with a $2n-1$ fan-in OR gate. The output of this OR gate gives the accurate result.

Delay difference between the inputs of ith and $i-1$th AND gates, representing the delay of the corresponding block, can be generalized as follows:

$$\text{For Input-1} \begin{cases} 0 & i = 1 \\ n-(i-1) & i = 2, 3, \ldots, n \\ n & i = n+1 \\ i-(n+1) & i = n+2, n+3, \ldots, 2n-1 \end{cases}$$

$$\text{For Input-2} \begin{cases} 0 & i = 1 \\ n-i & i = 2, 3, \ldots, n \\ -(n-2) & i = n+1 \\ i-n & i = n+2, n+3, \ldots, 2n-1 \end{cases}$$

Note that $(n+1)$th case for the second input has a negative value meaning that it needs less delay than that of nth case (see Fig. and Fig. 9).

As a result, the delay blocks offer total of $n^2-(n-1)$ and $n^2-(n)$ bit delays for Input-1 and Input-2, respectively. Therefore, we need at least $\approx 2n^2$ inverters to realize the delay blocks, so the area complexity is $O(n^2)$.

IV. Synchronous Adders and Multipliers

The proposed asynchronous circuits are easy to design with delay controllability features. However, their area quickly grows with the input stream length n; for high n values the circuits become inefficiently large. Also, they cannot process successive input streams. To solve these problems, we proceed to synchronous designs. The existence of auxiliary signals allows to keep and process the information carried by streams with binary digits. Indeed, the resulting circuits are hybrid with processing both streams and binary digits.

We have two classes for the proposed synchronous designs that have increasing and constant stream lengths. While the former one is fully-accurate, similar to the proposed asynchronous ones, the latter one concedes slight errors with an important plus of being able to process successive input streams. As a result, we propose four adders and multipliers that are thoroughly explained in the following four subsections.

A. Increasing Stream Length: Fully-accurate Addition

We mainly use the same approach as we previously use for our asynchronous adder: one of the input streams waits until all bits of the other one is processed. Instead of using an asynchronous delay block as in Fig. 7, we use synchronous blocks to store the input information in binary format that is more area efficient especially for large stream lengths.

Fig. 10 shows the proposed adder for an input stream length $n = 8$; X_1 and X_2 represent the input values ranging between $0/8$ and $8/8$. Note that the stream length of the output should be 16 for accurate operation as previously stated in Lemma 1. The proposed adder first turns X_1 into a binary format via the counter. After the completion of the counting process, the register saves the information in the output of the counter. Then, binary to stream converting is done by the multiplexer. Finally, addition is performed with an OR gate.

Even though the 8 bit stream corresponds to 3 bit binary resolution, the counter and the register are both selected 4 bit to be able to represent all 9 values between $0/8$ and $8/8$. Also, that is the reason why OR gates are used before the multiplexer. The largest binary value coming from the register is 1000 ($R_3 = 1$) that should produce 1's for all bits in the stream at the output of the multiplexer. To do so, R_3 is ORed with other R's.

For the multiplexer, along with the 4 inputs I_0, ..., I_3 coming from the register, there is one more input I_4 used to make the output of the multiplexer logic 0 for a time duration of n bits, needed to process the input stream coming from Input-2. Table I shows the relation between the data inputs, the selection inputs, and the output of the multiplexer. All selection inputs

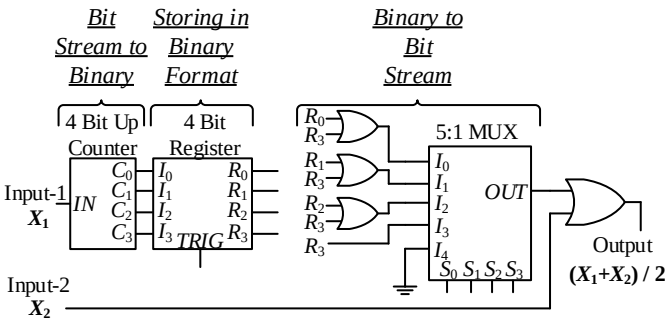

Fig. 10: The proposed synchronous fully-accurate adder for 8 bit inputs.

TABLE I: Relation between the selection inputs and the output of the 5:1 multiplexer in Fig. 10

SELECTION INPUTS				OUTPUT	
S_3	S_2	S_1	S_0	Equivalence	Duration
0	0	0	0	I_0	1 bit
0	0	0	1	I_3	1 bit
0	0	1	X	I_1	2 bit
0	1	X	X	I_2	4 bit
1	X	X	X	I_4	8 bit

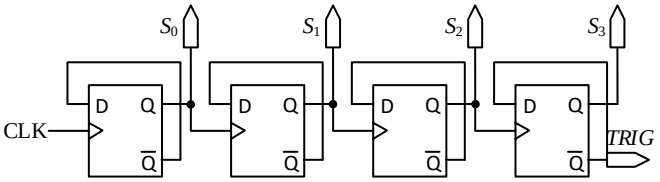

Fig. 11: The generation circuitry of the auxiliary signals in Fig. 10

are actually clock signals with 50% duty cycles. They can be generated from a single CLK input via frequency division with flip-flops, as shown in Fig. 11. Additionally, the $TRIG$ input of the register is selected as $\overline{S_3}$.

If the input streams have n bits, the counter and the register should be $\log_2 n + 1$ bit, and the multiplexer should have $\log_2 n+2$ data inputs and $\log_2 n+1$ selection inputs. Also, all auxiliary signals could be generated from a frequency divider circuit consisting of $\log_2 n+1$ successive flip-flops. As a result, the area complexity is $O(\log n)$.

B. Increasing Stream Length: Fully-accurate Multiplication

Consider two input bit streams with lengths of n. As mentioned earlier, accurate multiplication requires n^2 bitwise operations that is in compatible with Lemma 2. We satisfy this by repeating one of the streams n times, and by repeating each bit of the other stream n times. An example for $n = 4$ is shown in Fig. 12. Note that the orders of 0's and 1's in the input streams are not reflected to the repeated streams; after the counting process, we only have the information of the input values X_1 and X_2, not the orderings. In the example, both of the input streams are treated as (0,1,1,1) since $X_1 = X_2$.

The circuit implementation of the proposed multiplier for $n = 4$ is given in Fig. 13. The counting and reconversion circuitry is nearly same with the one in Fig. 10. The only difference is the number of inputs in multiplexers (one less), because there is no need to wait for one of the streams as we

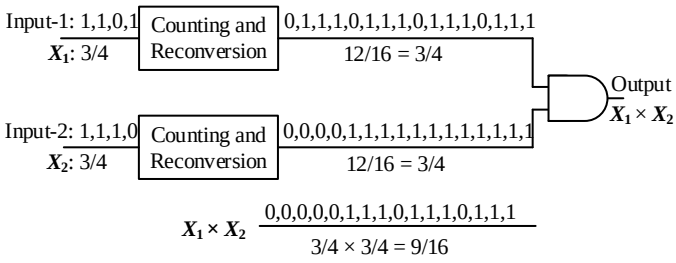

Fig. 12: The proposed multiplication scheme for 4 bit inputs.

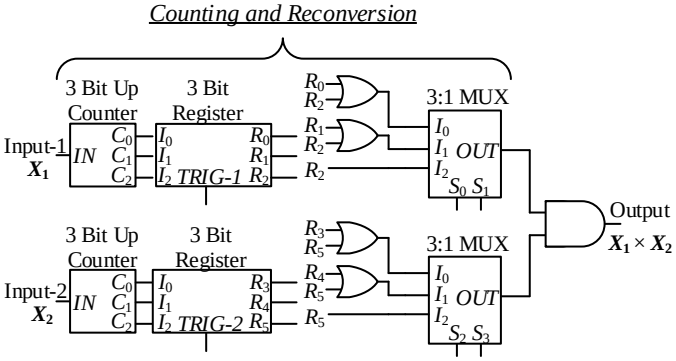

Fig. 13: The proposed fully-accurate synchronous multiplier for 4 bit inputs.

do for the adder. The selection inputs of the upper multiplexer (S_0, S_1) are 4 times faster than those of the lower ones (S_2, S_3). Additionally, the *TRIG-1* input can be selected as the negated form of S_2, and the *TRIG-2* input is the negated form of the two times slowed version of S_3. Therefore, all auxiliary inputs can be generated from a single clock signal by using a frequency divider circuit having 5 successive flip-flops.

Analyzing the scalability of the proposed multiplier, we see that the counters and registers should be $\log_2 n + 1$ bit, while the multiplexer should have $\log_2 n + 1$ inputs. Furthermore, $\log_2 n+2$ successive flip-flops needed to generate all required auxiliary signals. Therefore, the area complexity is $O(\log n)$.

C. Constant Stream Length: Semi-accurate Addition

Addition in BSC is having an average of the input values X_1 and X_2, and for the constant stream length, this can be performed with bit-by-bit averaging of the input stream bits. However, since the average of 1 valued and 0 valued bits results in 0.5 and it can not be represented with a single output bit, carry is needed to store the information. Table II shows the truth table for such a solution. A circuit suiting Table II should operate as desired. Fig. 14 shows the circuit implementation of the proposed adder. It works as follows: if the input values X_1 and X_2 are both even or both odd, then the result is correct; otherwise the result is the rounded version of the correct result with an error distance of $0.5/n$ where n is the input stream length. Fig. 15 shows two examples for the proposed addition operation giving erroneous and accurate results. Since the circuit area is constant, independent of n, the area complexity is $O(1)$.

Note that our circuit in Fig. 14 has a quite similar performance compared to the scaled adder in [11]. However, with our

TABLE II: Transition table of the proposed adder

Carry	Input-1	Input-2	Output	Carry-new
X	0	0	0	Carry
X	0	1	Carry	$\overline{\text{Carry}}$
X	1	0	Carry	$\overline{\text{Carry}}$
X	1	1	1	Carry

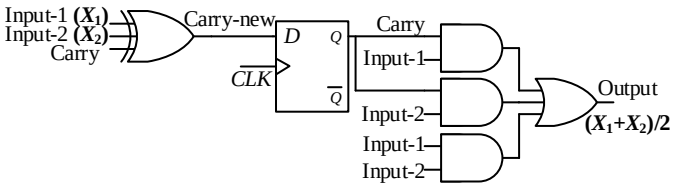

Fig. 14: The proposed semi-accurate synchronous adder for two inputs.

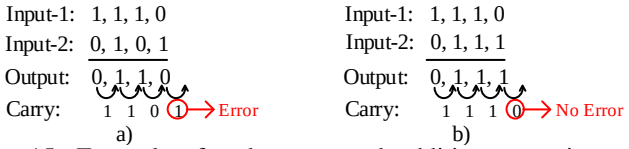

Fig. 15: Examples for the proposed addition operation: a) $X_1 = 3/4$ and $X_2 = 2/4$, and b) $X_1 = 3/4$ and $X_2 = 3/4$.

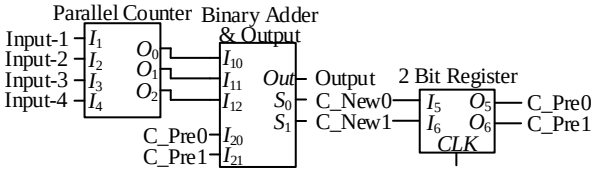

Fig. 16: The proposed semi-accurate synchronous adder for four inputs.

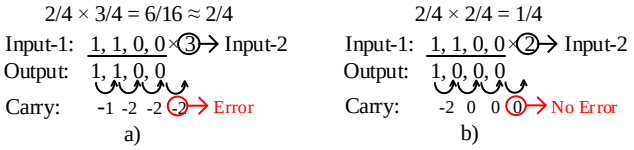

Fig. 17: Examples for the proposed multiplication approach: a) $X_1 = 2/4$ and $X_2 = 3/4$, and b) $X_1 = 2/4$ and $X_2 = 2/4$.

point of view, we can generalize our adder for any i number of inputs. In our design, at first the input bits are counted in parallel, and the result is added to the carry which has an initial value of $i/2$ to eliminate probable negative carry values. If the carry is larger than i, the output becomes 1, and i is subtracted from the carry. Otherwise, the output is 0. Fig. 16 shows $i = 4$ version where "Parallel Counter" simply counts 1's in the input streams, and "Binary Adder & Output" first adds the current carry value to the output of the counter, then determines the output and updates the carry value with aforementioned process steps.

D. Constant Stream Length: Semi-accurate Multiplication

We fundamentally use the same approach as we use for addition considering that multiplication is the repeated version of addition. Suppose that input bit streams represent values $X_1 = a/n$ and $X_2 = b/n$ where n is the length of the streams. If we add b copies of the first stream, or a copies of the second, we can achieve multiplication by using bit-by-bit averaging

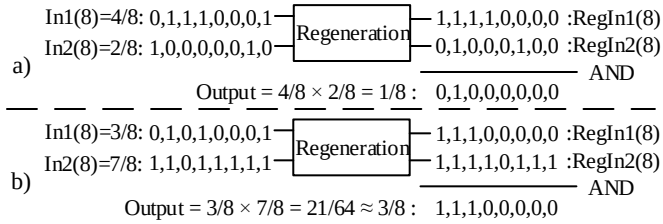

Fig. 18: The regeneration of input streams with Algorithm 1 with outputs giving a) no error, and b) optimal error.

with a carry. Different from the addition operation for which rounding to the nearest integer can be always satisfied with positive carry values, the multiplication operation with optimal error performance should have both positive and negative carry values between $-0.5n$ and $+0.5n$. Thus, we make the error distance upper bounded by $0.5/n$. Note that if we only used positive carries, this upper bound would be $1/n$.

Fig. 17 elucidates our multiplication scheme for 4 bit inputs. For example in Fig. 17 a), the first operation is adding three 1's with a result of 3; then $3/4$ is rounded to the nearest integer that is 1 as the output, and the carry with a value of -1 is transferred to the next bit operation for which three 1's and the carry results in $2/4$ that is rounded to 1 (it could have been rounded to 0 also) with a carry of -2 as an error.

Instead of directly implementing the flow in Fig. 17 that requires to first process Input-2, and then depending on the value of it, process Input-1, we process Input-1 and Input-2 separately. Thus, we can treat the inputs simultaneously by regenerating them with independent circuitries. We still satisfy the overall flow in Fig. 17 by achieving a faster, less complex, and smaller multiplier circuit. In fact, regeneration of input signals to achieve better accuracy has been previously used in[1] and [9]. They manipulate the input streams to achieve 100% accuracy at the output. However, these works produce larger output stream lengths than input ones that results in serious problems, as discussed in Section II.

For the proposed multiplication scheme, we regenerate input streams such that one of them is lined up in a way to process all 1's first and then all 0's, and the other one is used as a multiplicand. In every bit-by-bit operation, the multiplicand is added to the carry. Algorithm 1 demonstrates the steps regenerating input streams of $In1(n)$ and $In2(n)$ as $RegIn1(n)$ and $RegIn2(n)$, respectively. The generation of $RegIn1(n)$ is quite simple: first all 1's, then all 0's. However, $RegIn2(n)$ is generated with a more complex way. For each iteration, number of 1's in $In2(n)$ is added to $Carry$. If new $Carry$ is larger than or equal to $n/2$, it is subtracted by n and RegIn2(i) becomes 1, otherwise $RegIn2(i)$ becomes 0. Note that for the first iteration $Carry = 0$.

Fig. 18 shows two examples for the regenerations of input bit streams. The multiplication operation is completed by AND operation of the regenerated streams. In Fig. 18 a), since the expected output value of $1/8$ can be represented with 8 bits, the accurate output value of $1/8$ is obtained. On the other hand, in Fig. 18 b), the expected output value of $21/64$ cannot be represented accurately with 8 bits. Therefore, the output gives the nearest possible value of $3/8$.

Fig. 19 shows a circuit to realize Algorithm 1. It has three exactly same 4-bit up counters. Each has an input IN_i;

Algorithm 1 Regeneration of Inputs for Optimal Error Performance in Constant Stream Multiplication

1: **procedure** REGIN(In1(n),In2(n))=(RegIn1(n),RegIn2(n)) ▷ Regeneration of n-bit Input Streams
2: $SumIn1 \leftarrow sum(In1)$
3: $SumIn2 \leftarrow sum(In2)$
4: $Carry \leftarrow 0$
5: **for** $i \leftarrow 1, n$ **do**
6: **if** $i \leq SumIn1$ **then** ▷ Determining ith bit of the 1st Regenerated Input
7: $RegIn1(i) \leftarrow 1$
8: **else**
9: $RegIn1(i) \leftarrow 0$
10: **end if**
11: $Carry \leftarrow Carry + SumIn2$ ▷ Addition of Multiplicand to the Carry
12: **if** $Carry \geq n/2$ **then** ▷ Determining ith bit of the 2nd Regenerated Input
13: $RegIn2(i) \leftarrow 1$
14: $Carry \leftarrow Carry - n$
15: **else**
16: $RegIn2(i) \leftarrow 0$
17: **end if**
18: **end for**
19: **end procedure**

four output ports C_{ij} and their negates C_{ijB}; and four clear and four preset inputs CLR_i and PRE_i, respectively. For simplicity, unused ports are not generally shown in the circuit. That is why counters look different though they are exactly same. Similarly, each of the three identical registers has four inputs I_{ij}; four outputs R_{ij} and their negates R_{ijB}; and clear and clock inputs CLR and CLK (CLK or $TRIG$), respectively. Again unused ports are not shown.

Inputs IN_1 and IN_2 are first counted by up counters. The information of IN_1 is saved in the 4-bit register and then inversely loaded to the next 4-bit up counter via the CLR_i and PRE_i inputs generated from $TRIG$, R_{1j}, and R_{1jB}. The signals CLR_i and PRE_i are connected to the $CLEAR$ and $PRESET$ inputs of the corresponding D-FF in the counter to transfer the inversion of the saved information in the register to the counter after the counting in the first counter is completed. This means that the counter starts to count from inversion of saved information instead of from "0000". Then the most significant bit (MSB) of the counter C_{23} and negated version of other bits, C_{2iB} for $i = 0, 1, 2$, determine REG_IN_1. If the counter output is between 0111 and 1110, REG_IN_1 becomes 1; otherwise it becomes 0. For instance, for the case of $X_1 = 5/8$, the saved information is 0101 and the transferred information is 1010. So, the counter starts from 1010 and arrives at 1111 after 5 clock cycles, which means first 5 bits of REG_IN_1 are 1 and the rest are 0. Thus, we generate REG_IN_1 with an up counter and some logic circuit instead of a costly digital comparator.

The saved information of IN_2 in the 4 bit register is summed with the previous sum in each clock cycle by a binary addition block so-called "No Carry Adder" which excludes the MSB (or final carry) of a classical binary adder. This summation performs the operation $Carry \leftarrow Carry + SumIn2$ in the line 11 of Algorithm 1. In other words, the inputs of the undermost register corresponds to $Carry$. Note that $Carry$ varies between $[-n/2, n/2]$. To get rid of negative numbers, we start $Carry$ from $n/2$, instead of 0, and shift the interval to $[0, n]$ by starting the undermost register from 01..0, instead of from 00...0. Thus, if the MSB of the output of No Carry Adder is 1, which means the unshifted and shifted carries are larger than $n/2$ and n, respectively, REG_IN_2 becomes 1. Otherwise, $Carry$ is not large enough to produce 1, so REG_IN_2 becomes 0. Furthermore, the MSB of the input of the undermost register is always zero to implement the subtraction in the line 14 of Algorithm 1. If REG_IN_2 is 1 then it means that $Carry \geq n$, so $Carry$ should be subtracted by n, i.e. MSB should be turned into 0. On the other hand, $REG\_IN_2 = 0$ does not require any change in $Carry$, so the MSB should be again 0.

The circuitry to generate auxiliary inputs $TRIG$ and CLR is also added in Fig. 19. Essentially, CLR is the slightly delayed version of $TRIG$, because registers using $TRIG$ signal need to save the outputs of the leftmost counters before they are cleared. The proposed design in Fig. 19 can be generalized for n bit inputs. Counters, registers, and binary adders should be $\log_2 n + 1$ bit. Additionally, $2 \times (\log_2 n + 1)$ AND gates are needed for producing CLR and PRE signals, and $log_2 n$ successive flip-flops are used in auxiliary signal generation. As a result, the area complexity becomes $O(\log n)$.

V. Experimental Results

In this section, we evaluate the proposed six circuits:

- **A**synchronous **I**ncreasing **S**tream-length **A**dder (**AISA**),
- **A**synchronous **I**ncreasing **S**tream-length **M**ultiplier (**AISM**),
- **S**ynchronous **I**ncreasing **S**tream-length **A**dder (**SISA**),
- **S**ynchronous **I**ncreasing **S**tream-length **M**ultiplier (**SISM**),
- **S**ynchronous **C**onstant **S**tream-length **A**dder (**SCSA**), and
- **S**ynchronous **C**onstant **S**tream-length **M**ultiplier (**SCSM**).

The results are grouped in the following four subsections. In the first one, we present transistor level simulation results with an aim of showing that the proposed circuits work properly and accurately in practice. Second subsection includes transistor level delay results for the proposed design and their counterparts in the literature. In the third and the fourth subsections, we thoroughly compare the proposed adders and multipliers with those in the literature, individually and in a neural network application, respectively. In these two subsections, we perform gate level simulations by considering area and accuracy.

A. Transistor Level Evaluations for Accuracy

We perform simulations with the AMS 0.35μm CMOS technology in Cadence environment. We test the proposed adders and multipliers for different input bit widths ranging between 0.5ns and 10ns. Input values are selected such that the expected output value is around $1/2$ that can be considered as the worst-case scenario for accuracy. Input stream lengths are selected as 8 for all simulations. Two performance metrics are considered: 1) integrity of the output signal, and 2) correctness of the obtained output value. To elaborate, consider an expected output stream $1, 0, 1, 1, 0, 0, 0, 1$ with a bit width is 1ns. Also suppose that from simulations, we obtain 1 valued bit widths of 1.1ns, 2.1ns, and 0.4ns; ideally it should be 1ns, 2ns, and 1ns, respectively. For the first metric, we first

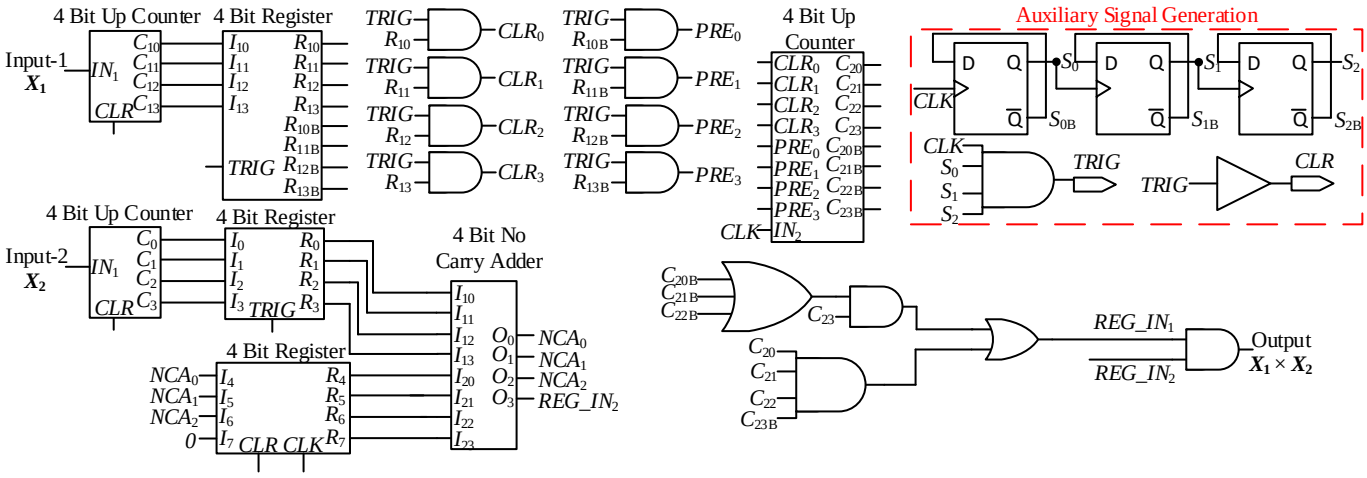

Fig. 19: The proposed semi-accurate synchronous multiplier for 8 bit inputs.

TABLE III: Integrity and correctness results of the proposed adders and multipliers (INT:Integrity, COR:Correctness)

Bit Width	AISA		AISM		SISA		SISM		SCSA		SCSM	
	INT	COR	INT	COR	INT	COR	INT	COR	INT	COR	INT	COR
0.5ns	87.8%	100%	91.8%	100%	31.8%	37.5%	0%	0%	56.9%	50%	0%	0%
0.75ns	95.3%	100%	85.2%	97.1%	87.5%	100%	85.7%	100%	95.6%	100%	0%	0%
1ns	92.5%	100%	58.4%	60%	90.5%	100%	89.2%	100%	95.5%	100%	0%	0%
2ns	N/A	N/A	N/A	N/A	95.4%	100%	94.6%	100%	97.8%	100%	66.9%	75%
10ns	N/A	N/A	N/A	N/A	99.1%	100%	98.9%	100%	98.2%	100%	94.8%	100%

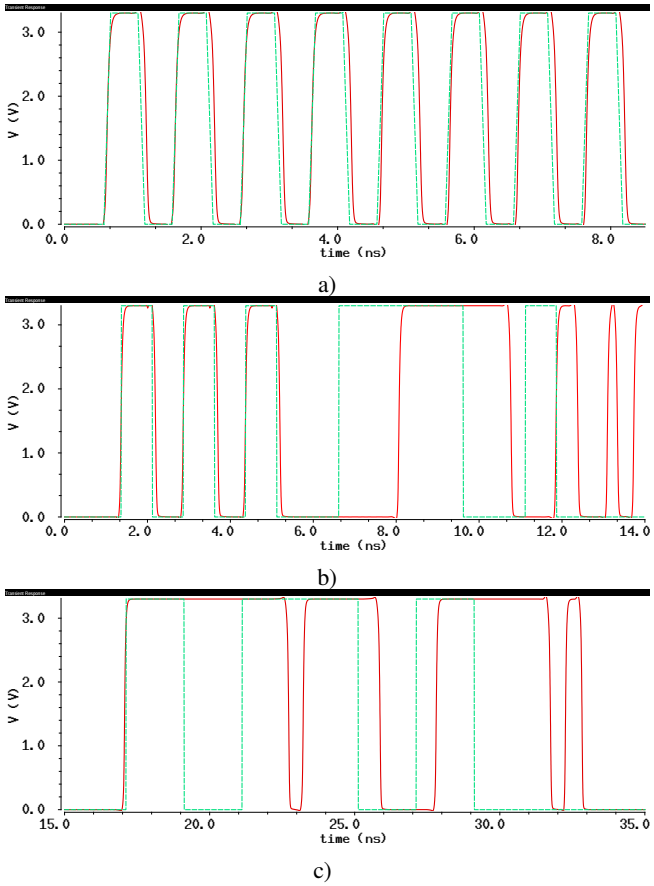

Fig. 20: Transistor level simulation results for a) AISA with 0.5ns bit width, b) SISA with 0.750ns bit width, and c) SCSM with 2ns bit width. Red solid lines and green dashed lines represent real and expected outputs, respectively.

calculate absolute deviations of 0.1ns, 0.1ns, and 0.6ns, then relative deviations 0.1ns/1ns, 0.1ns/2ns, and 0.6/1ns, and finally the average deviation of 25% is obtained that results in 75% signal integrity. For the second metric, we first obtain the ratios of the obtained output bit widths over the given bit width. For this example, the ratios are 1.1ns/1ns, 2.1ns/2ns, and 0.4/1ns. Then we round them to the nearest integers, as 1, 2, and 0 for the example. Finally we have the obtained output value of 3/8 with 75% correctness.

Table III shows the transistor level accuracy results of the proposed circuits for various bit width values. Except for SCSM, all of the proposed circuits can work for a bit width equal to or smaller than 1ns, so the operating frequency of 1GHz can be achievable. Actually this is quite expected regarding the complex connections and feedbacks of the architecture of SCSM. To achieve so, it needs to be thoughtfully optimized for the given operating frequency to get rid of any possible timing errors and skews. Note that the proposed asynchronous circuits do a better job in small bit widths, thanks to their simple and delay block based structures. However, since increase in the bit width requires more delay, and after a certain point it cannot be achieved with the controlling mechanism (in our case, VDD scaling), additional hardware in terms of extra inverters is needed. That is why for bit widths of 2ns and 10ns, the proposed circuits fail.

Fig. 20 shows the expected and the real output signal forms for three different cases. While the output signal has a good alignment in Fig. 20 a), Fig. 20 b) and c) show undesirable glitches and shifts.

Note that we evaluate the circuits for only 8 bit inputs as a simple case to show the potential of the proposed circuits for practical use. Larger input stream lengths could be considered, but since larger lengths make our circuits more complicated

TABLE IV: Delay Comparison of Adders

Adders	ICD (ns) for 16 bits	ICD (ns) for 8 bits	SD	SD (ns) for $n = 8$ and $W = 2$ns
Ripple Carry Adder	1.80	1.44	NA	NA
[9]	4.86	4.08	$(n^2+n)\times W$	144
[6]	3.56	2.94	$(n^2+n)\times W$	144
[11]	1.17	1.17	$n \times W$	16
AISA	0.15	0.15	$2 \times n \times W$	32
SISA	3.85	3.31	$3 \times n \times W$	48
SCSA	1.17	1.17	$n \times W$	16

TABLE V: Delay Comparison of Multipliers

Multipliers	ICD (ns) for 16 bits	ICD (ns) for 8 bits	SD	SD (ns) for $n = 8$ and $W = 2$ns
Array Multiplier	3.39	2.67	NA	NA
[9]	4.70	3.92	$(n^2+n)\times W$	144
[6]	3.40	2.78	$(n^2+n)\times W$	144
AISM	0.15	0.15	$n^2 \times W$	128
SISM	3.85	3.31	$(n^2+n)\times W$	144
SCSM	5.19	4.29	$2 \times n \times W$	32

with systematic timing design strategies needed to be followed, we consider this as a future work. However, it is not hard to predict that for larger input stream lengths our asynchronous designs would have again quite successful outcomes, mainly because the architectures are quite straightforward. This is also true for SCSA, due to its scalable architecture. However, other synchronous designs, especially SCSM, circuits are getting more complex for larger stream lengths, so they may become more prone to timing errors.

B. Transistor Level Evaluations for Delay

We carry out delay simulations for our designs along with their stochastic and binary counterparts in the literature. Again we use the AMS 0.35μm CMOS technology in Cadence environment. To evaluate the results more efficiently, we separate the results into two forms: "Inherent Circuit Delay (ICD)" and "Stream Delay (SD)". Let bit width and stream length defined in Section II be represented as W and n, respectively. As a conventional measure, ICD is the worst case total propagation delay of a given circuit. It does not depend on W, however it may grow with n for non-scalable designs. On the other hand, SD is a delay measure specific for BSC/SC designs, not applicable (NA) for binary designs. It is the time difference between the beginning of input streams and the end of the output stream with an assumption of zero propagation delay. As a result, the total delay of the system can be calculated as the sum of total ICD and SD.

Table IV and V show the delay simulation results of the proposed adder and multiplier designs and their rivals in the literature. They include ICD's for input stream lengths of 8 and 16 (4 and 5 binary bits). Examining the results, we see that our asynchronous designs have the best performance for ICD. Furthermore, SD formulas are given as functions of n and W. Finally, the rightmost columns show the exact amounts of SD's where $n = 8$ and $W = 2$ns. Examining the results, we see that our constant stream designs have the best performance for SD. Note that for any BSC/SC design, SD's overwhelmingly dominate the total delay. In fact, this domination is valid for any meaningful values of n and W, which means that considering only SD's is a convenient and fair enough way to analyze delay behavior of BSC/SC designs.

TABLE VI: Transistor counts (TC) of the used logic components

	INV	NAND	NOR	XOR	D Flip-flops	Half Adder	Full Adder
TC	2	4	4	12	12	18	28

C. Gate Level Evaluations for Area

In comparisons, we consider three studies offering accurate stochastic operations that are based on clock division [9], using LFSR's [6], and PWM signals [10]. We also consider conventional binary ripple carry adder and array multiplier circuits. In order to make fair comparisons, we take into account the signal forms at the inputs and the outputs. All of the proposed six circuits with BSC use streams as inputs and outputs. However, since the proposed synchronous designs do already make stream-to-binary conversion via counters and registers, they can be directly used for binary-to-stream computing with even smaller circuit sizes. There is an exception for Input-2 of SISA, whose overhead is also considered. Moreover, the studies [9] and [6] use binary inputs and stream outputs; to make them perform stream-to-stream computing, counters and registers can be added to the inputs, that is why their stream-to-stream cases are more costly than their binary-to-stream counterparts. On the other hand, since the study [10] uses analog inputs and it is not straightforward to make such conversions, we separately evaluate it.

Obtained by using the values in Table VI, in Table VII and Table VIII, we compare transistor counts of the circuits. We consider different input levels; for example, the input level of 32 corresponds to 5 binary inputs or a stream having a length of 32. Examining the numbers in Table VII, we see that the proposed adders overwhelm the others in their categories "binary-to-stream" and "stream-to-stream". In overall comparisons binary ripple carry adder, AISA, and SCSA come forward; AISA and SCSA gives the best results for small and large input levels, respectively. However, considering that SCSA is not fully-accurate, the conventional ripple carry adder is still an important contender. Similar derivations can be observed from Table VIII for the multipliers that even gives more favorable results for the proposed circuits. One thing worth to mention is that the transistor counts are excessively high for the analog-to-stream computing. This happens due to the high area cost of the signal generators or ring oscillators used in the paper [10]; they are implemented with inverter chains. In our calculations, we select a bit width of 1ns that requires $33 \times n$ inverters in generators where n is the stream length and the number 33 is obtained by considering the inverter delay values for the AMS 0.35μm CMOS technology in Cadence environment. Note that the area cost can be reduced by decreasing the bit width. However, there is a lower bound for this, and it should be determined by transistor level simulations and detailed timing analysis. We believe that our assumption of using a bit width of 1ns is quite fair, even more than fair. Another way of decreasing the transistor counts would be using a different oscillator topology.

TABLE VII: Transistor count comparison of adders

	Binary-to-Binary	Binary-to-Stream				Stream-to-Stream					Analog-to-Stream
Input Levels / Stream Length	**Ripple Carry Adder**	**[9]**	**[6]**	**SISA**	**SCSA**	**[9]**	**[6]**	**AISA**	**SISA**	**SCSA**	**[10]**
8	84	446	250	184	186	638	394	22	224	66	1637
16	112	602	346	230	226	842	538	38	280	66	3221
32	140	770	454	276	266	1058	694	70	336	66	6389
64	168	950	574	322	306	1286	862	134	392	66	12725
128	196	1142	706	368	346	1526	1042	262	448	66	25397
256	224	1346	850	414	386	1778	1234	518	504	66	50741

TABLE VIII: Transistor count comparison of multipliers

	Binary-to-Binary	Binary-to-Stream				Stream-to-Stream					Analog-to-Stream
Input Levels / Stream Length	**Array Multiplier**	**[9]**	**[6]**	**SISM**	**SCSM**	**[9]**	**[6]**	**AISM**	**SISM**	**SCSM**	**[10]**
8	408	318	226	198	306	510	370	376	390	498	4784
16	688	430	314	262	394	670	506	1272	502	634	17984
32	1040	550	414	326	482	838	654	4600	614	770	69728
64	1464	678	526	390	570	1014	814	17400	726	906	274592
128	1960	814	650	454	658	1198	986	67576	838	1042	1089824
256	2528	958	786	518	746	1390	1170	266232	950	1178	4342304

TABLE IX: Qualitative comparison of adders and multipliers

	Delay	**Accuracy**	**Area**	**Successive Processing**	**Multi-level Design**
Conventional Binary	Excellent	Excellent	Moderate	Good	Excellent
[9]	Moderate	Excellent	Poor	Poor	Moderate
[6]	Moderate	Excellent	Moderate	Poor	Moderate
[10]	Moderate	Excellent	Poor	Poor	Poor
AISA/AISM	Moderate	Excellent	Moderate	Poor	Excellent
SISA/SISM	Moderate	Excellent	Good	Poor	Excellent
SCSA/SCSM	Moderate	Good	Good	Excellent	Excellent
Conventional Stochastic	Poor	Poor	Excellent	Excellent	Excellent

We also evaluate the aforementioned methods qualitatively in Table IX. We consider delay as the total computing time including the time duration of the output stream and the circuit's internal delay. We also consider area and accuracy as well as compatibility for processing successive input streams and multi-level design. Examining the results, we see that certain attributes need to be considered before choosing a suitable adder/multiplier. When all criteria are equally important, the conventional binary and the proposed SCSA/SCSM are fairly competent. However, when accuracy and area are the most important factors, the proposed SISA/SISM comes forward. The conventional stochastic seems to be the best in terms of the area since it uses a 2-to-1 multiplexer for addition and an AND gate for multiplication. However if we considered the costly random number generators, needed for SC, then the area cost of SC would even become the worst. Additionally, the proposed circuits are not satisfactory for the delay criterion; indeed this is valid for any circuit processing bit streams.

D. Evaluations within Neural Networks

To further evaluate the proposed circuits, we choose a neural network because it mainly consists of adders and multipliers, and it does not require perfect accuracy. We use a trained database PENDIGIT which is a set of handwritten digits [13]. It has 16 different input features corresponding to 16 perceptrons in the input layer of the neural network. There is also one hidden layer having 100 perceptrons. Obviously the output layer has 10 perceptrons to represent 10 digits. Except for the conventional implementation of the network with binary multipliers and adders, in each layer binary input values and their weights are multiplied with the binary-to-stream multipliers, and then they are summed in pairs with stream-to-stream adders. After that by first using stream-to-binary converters (just counters), all processes up to the next layer are implemented with conventional binary circuits. Because this part is same for all of the compared studies/techniques, we do not consider its area cost.

In comparisons, we consider four different implementation techniques of adders and multipliers. The first one offers the most area efficient accurate adders and multipliers among the studies considered in Table VII and Table VIII [6]. The second one uses adders very similar to the proposed SCSA, and conventional stochastic multipliers (AND gates) [11]. The third one uses conventional binary ripple carry adders and array multipliers, and finally the fourth technique employs conventional stochastic adders (2-to-1 multiplexers) and multipliers (AND gates). For the conventional stochastic circuits, randomly distributed input streams are needed. Therefore, binary-to-stream multipliers need one LFSR and digital comparators as twice as the multipliers. As stated in [3], one LFSR is enough for input streams, thanks to the low correlations between shifted streams. Additionally, each stream-to-stream adder needs a digital comparator for the generation of 0.5 valued stream for the select input of the multiplexer; again one LFSR is adequate for all adders.

Table X gives the results. The proposed implementations are clearly the best ones in terms of accuracy and circuit area. Of course, if we did not consider the costs of stochastic number generators, [11] and the conventional stochastic would have much smaller transistor counts. However, this would not be a fair comparison.

VI. Conclusion

We introduce a novel computing paradigm "Bit Stream Computing (BSC)" that benefits from the area advantage of stochastic logic and the accuracy advantage of conventional binary logic. The experimental results to evaluate the proposed adders and multiplier approve the efficiency of BSC in terms of area and accuracy. As a future work, we will perform more detailed simulations followed by fabrications by considering area, delay, power, and accuracy with timing variations. In our transistor-level simulations, we see that the proposed circuits properly work up to around a bit width of 1ns; we aim to

TABLE X: Transistor counts of all adders and multipliers used in a neural network with PENDIGIT database (TC:Transistor Count, MR:Misclassification Rate)

Input Levels	[11]		[6]		SISA-SISM		SCSA-SCSM		Conventional Binary		Conventional Stochastic	
	TC	MR	TC	MR	TC	MR	TC	MR	TC	MR	TC	MR
8	0.92M	56.77%	1.71M	7.64%	1.03M	7.64%	0.88M	31.3%	1.27M	7.64%	0.95M	69.4%
16	1.18M	24.84%	2.43M	2.89%	1.34M	2.89%	1.11M	7.60%	2.08M	2.89%	1.26M	47.9%
32	1.46M	9.63%	3.23M	2.60%	1.66M	2.60%	1.34M	3.00%	3.06M	2.60%	1.60M	25.6%
64	1.76M	4.04%	4.14M	2.63%	1.97M	2.63%	1.57M	2.80}	4.24M	2.63%	1.97M	11.8%
128	2.08M	3.12%	5.13M	2.54%	2.28M	2.54%	1.80M	2.60%	5.60M	2.54%	2.36M	6.00%
256	2.43M	2.84%	6.22M	2.57%	2.60M	2.57%	2.03M	2.54%	7.15M	2.57%	2.78M	3.87%

improve this. More generally, we aim to introduce a new circuit design methodology specifically developed for BSC.

Another direction is testing the proposed circuits in large area electronics including organic and flexible circuits that should have relatively small number of transistors. The conventional binary logic is not suitable for this; we comment that the proposed circuits performing BSC can be.

REFERENCES

[1] E. Vahapoglu and M. Altun, "Accurate synthesis of arithmetic operations with stochastic logic," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 2016, pp. 415–420.

[2] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[3] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 149–156.

[4] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, p. 92, 2013.

[5] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*. IEEE, 2014, pp. 361–366.

[6] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[7] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–4.

[8] ——, "Exploiting correlation in stochastic circuit design," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 39–46.

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

[10] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, 2017.

[11] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 13–18.

[12] N. R. Mahapatra, A. Tareen, and S. V. Garimella, "Comparison and analysis of delay elements," in *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*, vol. 2. IEEE, 2002, pp. II–II.

[13] E. Alpaydin and F. Alimoglu, "Pen-based recognition of handwritten digits data set," *University of California, Irvine, Machine Learning Repository. Irvine: University of California*, 1998.

Ensar Vahapoglu received the B.Sc. degree from the Department of Electronics and Communication Engineering Istanbul Technical University, Istanbul, Turkey in 2015. He is currently a M.Sc. student and works as a research assistant in the same department. His main research areas are analog/digital circuits design, stochastic computing and quantum computing.

Mustafa Altun received his BSc and MSc degrees in electronics engineering at Istanbul Technical University in 2004 and 2007, respectively. He received his PhD degree in electrical engineering with a PhD minor in mathematics at the University of Minnesota in 2012. Since 2013, he has served as an assistant professor at Istanbul Technical University and runs the Emerging Circuits and Computation (ECC) Group. Dr. Altun has been served as a principal investigator/researcher of various projects including EU H2020 RISE, National Science Foundation of USA (NSF) and TUBITAK projects. He is an author of more than 50 peer reviewed papers and a book chapter, and the recipient of the TUBITAK Success, TUBITAK Career, and Werner von Siemens Excellence awards.