2019 IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)

Efficient Architectures and Implementation of Arithmetic Functions Approximation Based Stochastic Computing

Tieu-Khanh Luong
Dept. of Electrical & Electronic Engineering, and MCCI [1]
University College Cork
Ireland
118221694@umail.ucc.ie

Van-Tinh Nguyen
School of Information Science
Nara Institute of Science and Technology
Japan
nguyen.van_tinh.np3@is.naist.jp

Anh-Thai Nguyen
Dept. of Microelectronics and Microprocessing
Le Quy Don Technical University
Hanoi, Vietnam
nguyenanhthai77@gmail.com

Emanuel Popovici
Dept. of Electrical & Electronic Engineering, and MCCI
University College Cork
Ireland
E.Popovici@ucc.ie

***Abstract*— Stochastic computing (SC) has emerged as a potential alternative to binary computing for a number of low-power embedded systems, DSP, neural networks and communications applications. In this paper, a new method, associated architectures and implementations of complex arithmetic functions, such as exponential, sigmoid and hyperbolic tangent functions are presented. Our approach is based on a combination of piecewise linear (PWL) approximation as well as a polynomial interpolation based (Lagrange interpolation) methods. The proposed method aims at reducing the number of binary to stochastic converters. This is the most power sensitive module in an SC system. The hardware implementation for each complex arithmetic function is then derived using the 65nm CMOS technology node. In terms of accuracy, the proposed approach outperforms other well-known methods by 2 times on average. The power consumption of the implementations based on our method is decreased on average by 40 % comparing to other previous solutions. Additionally, the hardware complexity of our proposed method is also improved (40 % on average) while the critical path of the proposed method is slightly increased by 2.5% on average when comparing to other methods.**

Keywords*— *stochastic computing, VLSI, arithmetic, low power, efficient architectures, sigmoid function, piecewise linear approximation, Lagrange interpolations

I. INTRODUCTION

Stochastic Computing was firstly introduced in the late 1960s by a research group led by Gaines [1] as an alternative method for efficient arithmetic digital representation circuits, but its origin can be traced back to von Neumann's seminal work on probabilistic logic [2]. This has recently attracted significant interest due to its unique properties which are extremely low-cost arithmetic units [3], fault tolerance, simple hardware implementation allowing very high clock rates [4]. However, apart from these advantages which lead to significant savings of hardware cost and power consumption, SC presents some disadvantages including long latency and degradation of accuracy. Applying massive parallelism in computation driven by the very low-cost hardware area may alleviate some of these disadvantages [4].

In Stochastic Computing (SC), the basic concept is to represent a real number with a random bit stream by comparing it with a uniformly distributed random number. A given binary number (BN) B is converted to a stochastic bit-stream form by a stochastic number generator (SNG). The SNG samples a BN R, compares it with B, and outputs a stochastic number of probability $B/2^k$ at a rate of one bit per clock cycle where k is the number of bits to represent a BN. After N clock cycles, it has produced an N-bit stochastic number (SN) X with $px \approx B/2^k$, where px is the frequency or rate at which ones appear in the sequence. Generally, the estimate's accuracy depends on the randomness of X's bit-pattern and its length N [5].

Using a stochastic representation can map complex arithmetic functions to simple bitwise logical operation. Figure 1 shows fundamental stochastic computational elements. Multiplication which has high hardware cost in terms of area and power consumption can now be implemented in a single AND gate in stochastic computing in unipolar format ($0 \le px \le 1$) or XNOR gate in bipolar format ($-1 \le px \le 1$). NOT gate is used to implement $1 - x$ in unipolar format and $-x$ in bipolar format. NAND gate is used to implement $1 - ax$, where a is in [0, 1]. Addition is implemented by a multiplexer (MUX) and the output form MUX is the scaled result [3]. These simplified operations, along with the ones presented in this paper, have a significant area and power impact on many application in digital signal processing, communications (e.g. LDPC codecs), neural networks as well as the general field of probabilistic computing [6].

With the recent trend of low power wearable devices and Internet of Things (IoTs), there comes the necessity of integrating more artificial intelligence at the edge. SC becomes attractive to integrate neural network onto embedded and portable devices, which require low power and energy consumption and are constrained by a small hardware area. Deep Belief Network and Deep Neural Network based on Stochastic sComputing have been proposed recently in [7] and [8], respectively with low energy consumption, high area efficiency. For many signal processing applications, SC has also been exploited to significantly reduce the complexity and the power consumption (e.g. Lower Upper-Decomposition for

1 MCCI : Microelectronic Circuits Centre Ireland, www.mcci.ie , a Technology Centre supported by Enterprise Ireland.

2160-052X/19/$31.00 ©2019 IEEE
DOI 10.1109/ASAP.2019.00018

MIMO receiver [9], Sparse code multi-access detector [10], digital filters, etc). Recently, some works have shown that SC-based LDPC decoders are competitive in performance and cost with conventional binary designs while providing significant savings in area [5]. Besides, SC has also been applied for computation engine for DRAM based In-stu Accelerator [11], image processing [12].

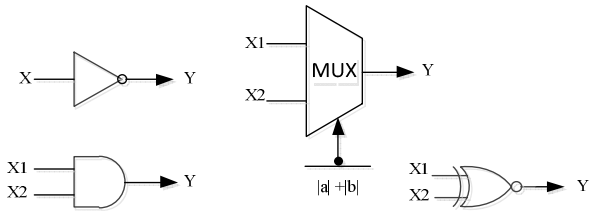

Fig. 1: Fundamental Stochastic Computational Elements

One of the most attractive features of SC is that complex computational functions can be implemented by very low-cost hardware design allowing for massive parallelism. Widely used exponential and hyperbolic tangent functions on stochastic bit streams were proposed in [3] using FSM. But, one drawback of the work is that there were a limited number of functions which can be implemented by using this method. FSM based on Markov chains with a redesign of topology was proposed, so that more complex functions can be synthesized stochastically [4]. In [5] an alternative method based on Bernstein polynomials has been shown in which high complexity functions can be approximated. However, Bernstein polynomials based implementation requires a higher degree of polynomials to reach higher accuracy. This leads to a higher hardware complexity and higher cost. Stochastic logic based implementations of complex arithmetic functions using truncated Maclaurin series polynomials was presented in [3]. This method drastically reduced the critical path and hardware complexity compared to FSM and Bernstein polynomials based implementations.

In our quest to further simplify the implementations of these functions, we propose a hybrid method to implement complex arithmetic trigonometric, exponential, logarithmic and sigmoid functions based on stochastic computing in which both PWL as well as polynomial interpolation methods are used for approximation.

II. Arithmetic Functions Based Piecewise Linear Approximation

In this section, the proposed method for calculating complex arithmetic functions is provided which is based on the combination of piecewise linear approximation and Lagrange interpolation polynomial. The uniform subdivisions and the transformations of each subdivision are selected to increase the accuracy while keeping the hardware complexity as low as possible.

The polynomial approximation is based on the classical Weierstrass theorem [13] in which for any continuous function $f(x)$ in the range [a, b], there exists a polynomial $L(x)$ for approximating the function in that range so the maximum approximation error is below the given limit. Then degree polynomial can be found through $n+1$ fitting points in the interval range $a_i \le x \le b_i$. The coefficients of this polynomial are given by the Lagrange interpolation formula:

$$f(x) = \sum_{i=0}^{n} L_i(x) \tag{1}$$

Where each $L(x)$ term is defined by

$$L_i(x) = f(x_i).\prod_{j=0, i\neq j}^{n} \frac{x - x_i}{x_i - x_j} \tag{2}$$

With each set of fitting points, we can calculate the polynomial $L(x)$ then the approximation error is $\varepsilon = p(x) - L(x)$. However, this approximation may not be optimal. Hence, we can perform an exhaustive search in the range $a \le x \le b$ with all possible pairs of fitting points on $f(x)$ to find the optimal polynomial.

$$c_0 = \cos\left(\frac{\pi}{2n+2}\right), \dots, c_n = \cos(\frac{(2n+1)\pi}{2n+2}) \tag{3}$$

For the approximation interval [-1, 1], the best points which are to minimize the maximum of the product $|(x - x_0)(x - x_1) \dots (x - x_n)|$ are the so-called Chebyshev nodes [14], for which

$$\begin{aligned} &Max|(x - x_0)(x - x_1) \dots (x - x_n)| \\ &-1 < x < 1 \end{aligned} \tag{4}$$

is minimal. In general, for the case with $n+1$ points x_0, x_1, x_n the Chebyshev nodes are:

$$\begin{aligned} &Max|(x - c_0)(x - c_1) \dots (x - c_n)| = \frac{1}{2^n} \\ &-1 < x < 1 \end{aligned} \tag{5}$$

Using the Chebyshev nodes usually provides an interpolating polynomial that is reasonably close to the optimal [14].

Using a high-order polynomial to approximate complex arithmetic functions based on stochastic computing will lead to an increase of hardware complexity as well as longer latency. In general, the shorter the approximation interval, the closer to linear the function then the lower degree polynomial is required while keeping the suitable error approximation. Therefore, reducing approximating interval into subintervals is necessary.

A complex arithmetic function $f(x)$ is approximated by using piecewise-linear (PWL) approximation in which these continuous functions are broken into segments. The domain of $x \in (\alpha, \beta)$ could be divided into s equal segments. Then, each segment is approximated by a linear function. The number of segments is chosen such as $s = 2^k$ with (k = 1,2,3,4 …) [15]. In segment i^{th}, the function f(x) can be written as:

$$\begin{aligned} &f(x) \approx a_i x + b_i, \ \frac{i}{s}(\beta - \alpha) \le x \le \frac{i+1}{s}(\beta - \alpha) \\ &i = 0 \to s - 1 \end{aligned} \tag{6}$$

where i describes i^{th} segment.

The absolute values of coefficients a_i and b_i could be described in (7) where B represents the number of bits of the two coefficients.

TABLE I

THE OPTIMIZED VALUES OF COEFFICIENTS a_i AND b_i ACHIEVED BY USING PIECEWISE-LINEAR APPROXIMATION AND LAGRANGE INTERPOLATION

Segment	$ln(1+x)$		$tanh(x)$		$sigmoid(x)$		sin(x)	
	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
0	964 x 2^{-10}	1 x 2^{-10}	1023 x 2^{-10}	1 x 2^{-10}	256 x 2^{-10}	512 x 2^{-10}	1023 x 2^{-10}	0 x 2^{-10}
1	861 x 2^{-10}	14 x 2^{-10}	988 x 2^{-10}	4 x 2^{-10}	254 x 2^{-10}	512 x 2^{-10}	1023 x 2^{-10}	2 x 2^{-10}
2	780 x 2^{-10}	34 x 2^{-10}	929 x 2^{-10}	19 x 2^{-10}	250 x 2^{-10}	513 x 2^{-10}	1023 x 2^{-10}	2 x 2^{-10}
3	713 x 2^{-10}	60 x 2^{-10}	850 x 2^{-10}	49 x 2^{-10}	244 x 2^{-10}	515 x 2^{-10}	974 x 2^{-10}	10 x 2^{-10}
4	655 x 2^{-10}	88 x 2^{-10}	758 x 2^{-10}	95 x 2^{-10}	237 x 2^{-10}	519 x 2^{-10}	927 x 2^{-10}	28 x 2^{-10}
5	606 x 2^{-10}	118 x 2^{-10}	660 x 2^{-10}	156 x 2^{-10}	228 x 2^{-10}	524 x 2^{-10}	866 x 2^{-10}	58 x 2^{-10}
6	565 x 2^{-10}	150 x 2^{-10}	563 x 2^{-10}	229 x 2^{-10}	218 x 2^{-10}	532 x 2^{-10}	791 x 2^{-10}	105 x 2^{-10}
7	529 x 2^{-10}	181 x 2^{-10}	472 x 2^{-10}	308 x 2^{-10}	207 x 2^{-10}	542 x 2^{-10}	704 x 2^{-10}	170 x 2^{-10}
Segment	e^{-2x}		$cos(x)$		e^{-x}		$\frac{\sin(\pi x)}{\pi}$	
	a_i	b_i	a_i	b_i	a_i	b_i	a_i	b_i
0	-1809 x 2^{-10}	1023 x 2^{-10}	-63 x 2^{-10}	1025 x 2^{-10}	-962 x 2^{-10}	1005 x 2^{-10}	1001 x 2^{-10}	0 x 2^{-10}
1	-1409 x 2^{-10}	970 x 2^{-10}	-190 x 2^{-10}	1041 x 2^{-10}	-849 x 2^{-10}	988 x 2^{-10}	846 x 2^{-10}	20 x 2^{-10}
2	-1097 x 2^{-10}	893 x 2^{-10}	-315 x 2^{-10}	1072 x 2^{-10}	-748 x 2^{-10}	926 x 2^{-10}	567 x 2^{-10}	91 x 2^{-10}
3	-855 x 2^{-10}	802 x 2^{-10}	-433 x 2^{-10}	1116 x 2^{-10}	-661 x 2^{-10}	859 x 2^{-10}	199 x 2^{-10}	229 x 2^{-10}
4	-665 x 2^{-10}	708 x 2^{-10}	-545 x 2^{-10}	1172 x 2^{-10}	-582 x 2^{-10}	773 x 2^{-10}	-199 x 2^{-10}	428 x 2^{-10}
5	-518 x 2^{-10}	616 x 2^{-10}	-649 x 2^{-10}	1237 x 2^{-10}	-517 x 2^{-10}	723 x 2^{-10}	-567 x 2^{-10}	658 x 2^{-10}
6	-403 x 2^{-10}	530 x 2^{-10}	-743 x 2^{-10}	1307 x 2^{-10}	-453 x 2^{-10}	682 x 2^{-10}	-846 x 2^{-10}	868 x 2^{-10}
7	-314 x 2-10	452 x 2^{-10}	-825 x 2^{-10}	1379 x 2^{-10}	-394 x 2^{-10}	611 x 2^{-10}	-1001 x 2^{-10}	1001 x 2^{-10}

$$a_i, b_i = N.2^{-B} \quad N, B \in Z \tag{7}$$

The optimized values of coefficients a_i and b_i can be generated by using Lagrange Interpolation Approximation method by using Chebyshev nodes as presented above to minimize the approximation error. The complex arithmetic functions $f(x)$ that are investigated in this paper include $ln(1+x), e^{-x}, e^{-2x}, tanh(x), sigmoid(x), cos(x), sin(x), \frac{sin(\pi x)}{\pi}$. Theoretically, the accuracy can be improved by two alternatives: either by increasing the degree of approximation polynomials or dividing the approximation interval to smaller subintervals. In this paper, we will investigate the number of segments in two cases, namely s = 8 and s = 16 respectively. The number of bits of the coefficients is chosen B = 10 in order to balance the latency of computing and accuracy.

The Table I describes the approximated values of coefficients a_i and b_i achieved by using PWL approximation combined with Lagrange interpolation approximation.

III. THE PROPOSED HARDWARE ARCHITECTURES.

A. The Hardware Designs of f(x)= $e^{-x}, cos(x)$

By using the proposed method, the two above arithmetic functions can be approximated as follows:

$$f(x) \approx a_i x + b_i \tag{8}$$

Implementing the above approximation function needs a multiplier and an addition. The addition is implemented by using a 2 by 1 multiplexer and the multiplier is a simple two input AND gate. Note that this is the scaled version of addition so it will lead to a reduction of accuracy. However, the coefficients of $e^{-x}, cos(x)$ from the Table I, the values of a_i in i^{th} segment are always negative and their absolute values are smaller than b_i. Hence, the factor $\frac{a_i}{b_i}$ is in the range of [0, 1] and can be represented by a stochastic number. The equation (8) could be rewritten then as follows:

$$f(x) = 1 - \frac{a_i}{b_i}x, \qquad i = 0:7 \tag{9}$$

As a result, the stochastic elements used to implement the equation (9) includes a NAND gate rather than an AND gate and a multiplexer as in (8). The architecture to calculate these functions is illustrated in Fig.2. The factor of $\frac{a_i}{b_i}$ is stored in a look-up table (LUT A). Three MSBs of the input values are the values to select the appropriate factor $\frac{a_i}{b_i}$ corresponding to the i^{th} segment.

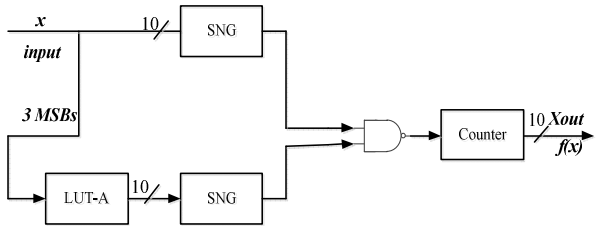

Fig. 2: The architecture of stochastic implementation of the functions $e^{-x}, cos(x)$

The output values of the LUT are then converted to a stochastic number by a stochastic number generator. Afterwards, the two stochastic numbers are inputs to a two input NAND gate to implement equation (9) and the output of the NAND gate will be converted to binary values through a counter.

B. The Hardware Design of $f(x) = ln(1+x),\ tanh(x), sigmoid(x), sin(x)$

From the Table I, it can be clearly seen that the values of both a_i and b_i are in the range [0, 1]. Note that the values of a_i are always greater than that of b_i. If we use $b_i = 1 - c_i$, the approximating function can be represented as:

$$f(x) = 1 - c_i + a_i x = 1 - c_i(1 - \frac{a_i}{c_i}x) \qquad (10)$$

Where $\frac{a_i}{c_i}$ are in the range [0, 1]. With this slight modification, we avoid using an addition in implementing the approximation function. The architecture of these functions is presented in Fig 3 as below.

In this hardware architecture, LUT-A stores the values of $\frac{a_i}{c_i}$ while LUT-B stores the values of c_i. The appropriate values for each segment are selected by three MSB of input. The SNGs convert from binary to stochastic numbers. The NAND gate is used to implement the expression $y = 1 - ax$ Therefore, we need two NAND gates to implement the equation (10). Then the stochastic bit-stream from the second NAND gate will be converted to a binary number by a counter.

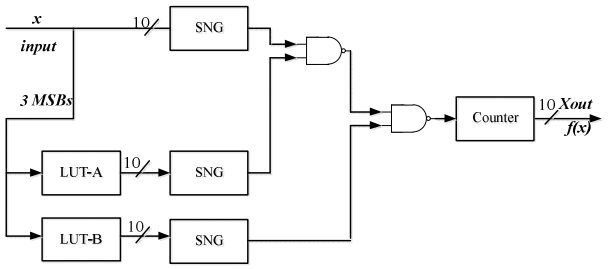

Fig. 3: The architecture of stochastic implementation of the functions $ln(1+x),\ tanh(x),\ sigmoid(x), sin(x)$

C. The Hardware Design of $f(x) = \frac{sin(\pi x)}{\pi}$

The values of a_i of approximation function are in the range [-1, 1], while those of b_i are in the range [0, 1]. Clearly, the domain of the values a_i is divided in two halves in which the domain of the first four values of a_i belongs to [0, 1] and the other four values is in [-1, 0]. Thus, the approximation function can be described by:

$$f(x) = \begin{cases} a_i x + b_i, & i = 0,1,2,3 \\ -a_i x + b_i, & i = 4,5,6,7 \end{cases} \qquad (11)$$

In the first half, the values of a_i and b_i are always positive and b_i's are less than a_i's. Hence the factor $\frac{a_i}{b_i}$ will exceed the range [0, 1] and cannot be represented by stochastic numbers. Hence, the approximation function can be described as follows:

$$f(x) = 1 - c_i(1 - \frac{a_i}{c_i}x) \quad i = 0,1,2,3 \qquad (12)$$

Where $c_i = 1 - b_i$ and c_i is greater than a_i so the factor $\frac{a_i}{c_i}$ is within [0, 1] and can be represented in the stochastic domain.

In the second half, Table I shows that the values of a_i's are negative whereas those of b_i's are positive. Also, the same table shows that the absolute values of b_i are greater than those of a_i's, so the approximation function can be written as below:

$$f(x) = 1 - \frac{a_i}{b_i}x, \qquad i = 4,5,6,7 \qquad (13)$$

A multiplexer is required to select the appropriate output result and the MSB will be fed to the multiplexer for the selection purpose. The hardware architecture of the function is presented in Fig. 4.

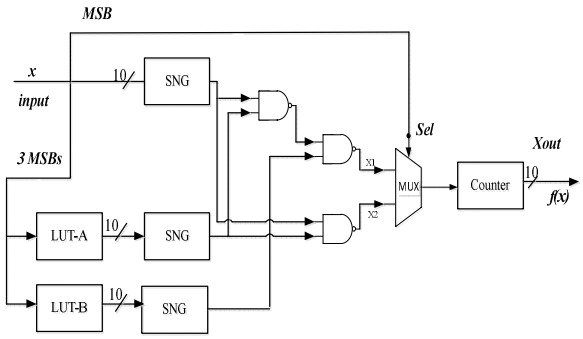

Fig. 4: The architecture of stochastic implementation of the functions $\frac{sin(\pi x)}{\pi}$

In this hardware architecture, LUT-A will be divided into two halves in which the first half stores the values of $\frac{a_i}{c_i}$'s and $\frac{a_i}{b_i}$'s values are stored in the second half. As the values of c_i's are only used for the first half of the approximating function, they will be stored in the first half of LUT-B and the second half of it is filled with zeros. Three MSBs of the input are also used to select the appropriate factors corresponding with each segment. The stochastic bit-stream from the multiplexer will then converted to a binary number by using a counter.

D. The Hardware Design of $f(x) = e^{-2x}$

In Table I, it can be seen that the range of values a_i's are in [-2, 0] whereas those of b_i's are in [0, 1]. Additionally, the absolute first four values of a_i are greater than those of b_i and vice versa for the next four values.

Considering the first four values, the absolute values of a_i are greater than 1 in three cases. And the values of the factor $\frac{a_i}{b_i}$ belong to [1, 2], so they cannot be directly converted to stochastic numbers. Noting these features, the factorization method is considered, namely $\frac{a_i}{b_i}$ is divided by 2. Then the approximation function of the first four values can be written as follows:

$$f(x) = -a_i x + b_i = 1 - \frac{a_i}{b_i}x, \qquad i = 0,1,2,3$$

$$= 1 - \frac{a_i}{2b_i}x - \frac{a_i}{2b_i}x$$

Where $\frac{a_i}{2b_i}$is in the range [0, 1] which can be a stochastic number. Implementing the above formula requires a subtraction.

TABLE II
THE OUTPUT MEAN ABSOLUTE ERROR (MAE) OF STOCHASTIC IMPLEMENTATION FOR DIFFERENT COMPLEX ARITHMETIC FUNCTION USING THE PROPOSED METHOD, MACLAURIN EXPANSION BASED METHOD AND THE FSM-BASED METHOD.

Functions		Proposed (8 Segments)	Proposed (16 Segments)	Horners rule [3]	FSM-based [3]
$\sin(x)$	Order	-	-	7	8 states
	Error	0.0013	0.0012	0.0034	0.0025
$cos(x)$	Order	-	-	6	8 states
	Error	0.0087	0.0063	0.0023	0.0053
$\ln(1+x)$	Order	-	-	7	8 states
	Error	0.0026	0.0011	0.0081	0.0186
$\tanh(x)$	Order	-	-	7	8 states
	Error	0.0012	0.0012	0.0140	0.0351
e^{-x}	Order	-	-	6	8 states
	Error	0.010	0.0064	0.0008	0.0154
e^{-2x}	Order	-	-	6	8 states
	Error	0.0766	0.0678	0.0009	0.0423
$sigmoid(x)$	Order	-	-	5	8 states
	Error	0.0043	0.0012	0.0046	0.0198
$\frac{\sin(\pi x)}{\pi}$	Order	-	-	11	8 states
	Error	0.0288	0.0288	0.0487	0.4716

However, since the value range [0, 1] is unipolar, i.e. does not include negative numbers, it is quite complex to implement the subtraction. One method has been proposed in [16] to implement the subtraction by using an XOR gate.

In the second four values of a_i *and* b_i, the approximation function can be described as follows:

$$f(x) = -a_i x + b_i = 1 - \frac{a_i}{b_i}x, \qquad i = 4,5,6,7 \tag{15}$$

The hardware architecture of the resulting function is shown in Fig. 5. In this hardware architecture, a LUT A is used to store the values $\frac{a_i}{2b_i}$ in the first half and $\frac{a_i}{b_i}$ in the second half. Three MSBs will be feed to the LUT to select the appropriate value in each segment. The selected signal of the multiplexer is given by the MSB bit of the input. The AND gate is used to implement unipolar multiplication. We use a one-bit delay element for the decorrelation of $\frac{a_i}{2b_i}$ [3] and then a XOR gate is implemented for subtraction [16]. The stochastic bit-stream from the multiplexer will then be converted to a binary number by using a counter.

IV. SIMULATION RESULTS

The functions presented in the previous sections are implemented by using the proposed method, the FSM-based method [17] (the 2-dimensional method more specifically as the 1-dimensional FSM-based method is not suited for the implementation of the selected complex functions [17]) and Horner's Rule for Maclaurin expansions [3].

The Monte Carlo simulation method was used to evaluate the Mean Absolute Error (MAE) of different algorithms. We use 1,024 bits to represent a numerical value stochastically. Table II shows the MAE of outputs of selected implementations by using this assumption.

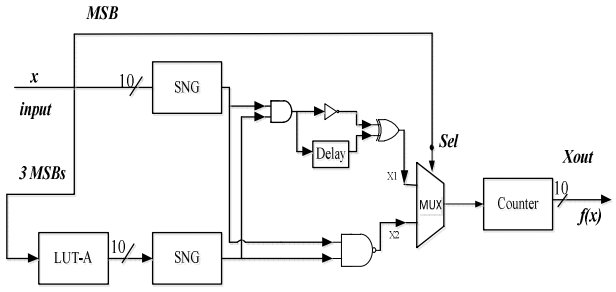

Fig. 5: The architecture of stochastic implementation of the function e^{-2x}.

In terms of accuracy, it is shown in Table II that the MAE of our proposed method outperforms FSM based method by a factor of 8.25 times on average. Additionally, from Table II, we can observe that the proposed method improved the MAE 2.5 times on average comparing to the Maclaurin based expansions method. Figure 6 and 7 show the simulation results of the stochastic implementation of widely used tanh(x) and sigmoid(x) functions.

All presented algorithms and architectures are implemented in hardware by using VHDL and synthesized in 65 nm CMOS library using Synopsys Design Compiler for a fair comparison. Table III shows the hardware implementation results for the implementations. The hardware complexity results are given in terms of equivalents GE (number of 2-input NAND gates).

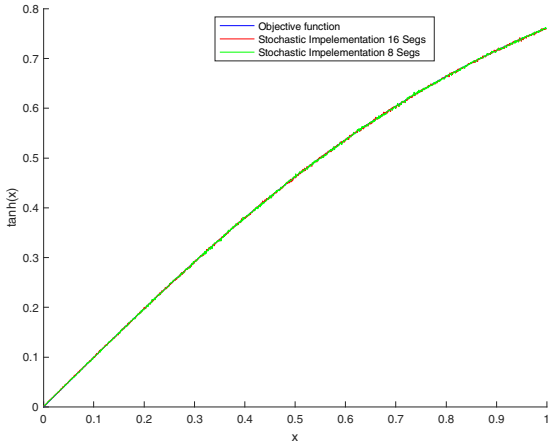

Fig. 6: The simulation result of stochastic implementation of tanh(x)

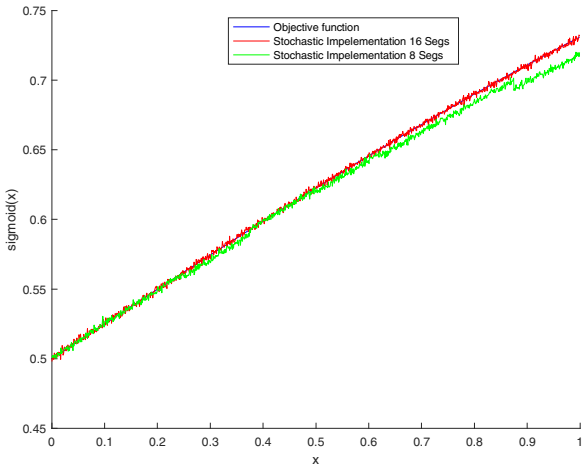

Fig. 7: The simulation result of stochastic implementation of *sigmoid(x)*

TABLE III

THE HARDWARE COMPLEXITY AND CRITICAL PATH DELAY OF STOCHASTIC IMPLEMENTATION OF COMPLEX ARITHMETIC FUNCTIONS WITH INPUT VALUES FROM 0 TO 1

Functions		Proposed (8 Segments)	Proposed (16 Segments)	Horners rule [3]	FSM-based [3]
sin(x)	Total Cell Area	601	650	2671	1269
	Power(µW)	16.5767	17.56	67.65	35.49
	Delay(ns)	2.87	2.99	3.09	2.71
$cos(x)$	Total Cell Area	489	525	769	1268
	Power(µW)	13.6596	13.6781	21.563	36.44
	Delay(ns)	3.07	3.11	2.86	2.96
$\ln(1+x)$	Total Cell Area	607	650	763	1269
	Power(µW)	16.73	17.498	21.42	34.79
	Delay(ns)	2.92	3.01	2.89	2.78
$\tanh(x)$	Total Cell Area	603	657	674	1270
	Power(µW)	16.62	17.644	18.82	35.5213
	Delay(ns)	2.97	3.12	2.89	2.71
e^{-x}	Total Cell Area	491	530	763	2489
	Power(µW)	13.06	13.9	21.48	41.29
	Delay(ns)	2.73	2.97	2.82	3.09
e^{-2x}	Total Cell Area	504	540	1199	1269
	Power(µW)	13.353	14.13	21.783	35.57
	Delay(ns)	3.08	3.32	2.87	2.71
$sigmoid(x)$	Total Cell Area	600	640	758	1489
	Power(µW)	16.542	17.29	21.15	41.23
	Delay(ns)	2.86	3.07	2.79	3.09
$\frac{\sin(\pi x)}{\pi}$	Total Cell Area	600	647	680	1269
	Power(µW)	16.3	17.4924	19.1076	35.52
	Delay(ns)	3.11	3.16	2.81	2.71

Finally, each architecture is evaluated for power consumption. The proposed architectures also required 32% and 58 % times (on average) less power than the Maclaurin based expansions and FSM based method, respectively. The cell area utilization of the proposed method requires 1.3 times and 2 times (on average) less than the Maclaurin based expansions and FSM based method, respectively. Additionally, the delay of the proposed method is 2.5 % more than others. Besides, when the size of LUT is doubled, the MAE is improved 1.64 times while cell area and power consumption are nearly unchanged. These results show that, the proposed hardware architectures are well suited for low-cost, low power applications by achieving a significant improvement over the state of the art.

V. CONCLUSION

Stochastic computing re-emerges as a promising technique in many applications which require low cost, fault tolerance and low power. Complex functions are often a determining cost factor for many hardware implementations. Stochastic implementations of complex arithmetic functions based on the combination of piecewise-linear approximation and Lagrange interpolation have been presented in this paper. Based on the proposed approximation scheme, a hybrid architecture for hardware implementation of complex arithmetic functions has been shown. The ASIC implementations and error analysis results have proven the benefits of the proposed approach with significant savings in area and power consumption. Future work will be towards further multi-objective optimisation of the proposed architectures (accuracy and power consumption) and their use in applications including LDPC decoders as well as neural networks.

VI. REFERENCES

[1] B. R. Gaines, "Stochastic Computing," in *Proc. AFIPS Spring Joint Computer Conf*, 1967.

[2] J. v. Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components,, Automata Studies, Shannon C.E. & McCarthy J., eds., Princeton University Press, pp. 43-98, 1956.

[3] K. Parhi and Y. Liu, "Computing Arithmetic Functions Using Stochastic Logic by Series Expansion," *IEEE Trans. on Emerging Topics in Computing,* pp. 1-13, Oct. 2016.

[4] B. D. Card and H. C. Brown, "Stochastic Neural Computation I: Computational Elements," *IEEE Transactions on Computers,* vol. 50, pp. pp. 891-905, 2001.

[5] A. Alaghi, W. Qian and J. P. Hayes, "The Promise and Challenge of Stochastic Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. vol. 37, pp. 1515-1531, Aug. 2018.

[6] K. Palem and A. Lingamneni, "What to Do About the End of Moore's Law, Probably!," in *DAC Design Automation Conference*, San Francisco, pp924-929 2012.

[7] Y. Liu, Y. Wang, F. Lombardi and J. Han, "An Energy-Efficient Online-Learning Stochastic Computational Deep Belief Network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 8, no. 3, pp. 454 - 465, 2018.

[8] Z. Li , J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu and Y. Wang, "HEIF: Highly Efficient Stochastic Computing based Inference Framework for Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. Early Access, pp. 1-1, 2018.

[9] J. Chen, J. Hu and J. Zhou, "Hardware and Energy-Efficient Stochastic LU Decomposition Scheme for MIMO Receivers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* pp. vol. 24, pp. 1391-1401, Apr. 2016.

[10] K. Han, J. Hu, J. Chen and H. Lu, "A Low Complexity Sparse Code Multiple Access Detector Based on Stochastic Computing," *IEEE Transactions on Circuits and Systems I: Regular Papers,* pp. vol. 65, issue. 2, pp. 769-782, Feb. 2018.

[11] S. Li , A. O. Glova , X. Hu , P. Gu , D. Niu , K. T. Malladi , H. Zheng, B. Brennan and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Fukuoka, 2018.

[12] A. Alaghi, C. Li and J. P. Hayes, "Stochastic Circuits for Real-Time Image-Processing Applications," in *ACM/EDAC/IEEE DAC*, Austin, TX, USA, 2013.

[13] W. Karl, Über die analytische Darstellbarkeit sogenannter willkülicher Funktionen reeller Argumente, Sitzungsber. Akad. Wiss. Berlin, pp. 633-639, 789-805 1885.

[14] S. P. Gordon and Y. Yang, "Approximating exponential and logarithmic functions using polynomial interpolation," *Intl Journal of Mathematical Education in Science and Technology,* pp. pp. 455-473, 2016.

[15] V.-T. Nguyen, V.-P. Hoang, V.-T. Sai, T.-K. Luong, M.-T. Nguyen and D.-H. Le, "A new approach of stochastic computing for arithmetic functions in wideband RF transceivers," in *2017 IEEE MWSCAS*, Boston, MA, USA, pp. 1525 - 1528, 2017.

[16] A. Alaghi and J. P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," in *IEEE ICCD*, Asheville, NC, USA, pp. 39-46, Oct.2013.

[17] P. Li, D. J. Lilja, K. Bazargan and M. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *IEEE/ACM (ICCAD)*, San Jose, CA, USA, pp. 480-487, Dec. 2012.