2024 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA) | 979-8-3503-9463-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/HORA61326.2024.10550497

FUNSC: A GUI Software for Stochastic Computing

Jackson P. Huse
Department of Electrical and Computer Engineering
University of Kentucky
Lexington, Kentucky, USA
jp.huse@uky.edu

Sayed A. Salehi
Department of Electrical and Computer Engineering
University of Kentucky
Lexington, Kentucky, USA
sayedsalehi @uky.edu

***Abstract*— Stochastic computing (SC) is an unconventional approach for performing computation by logic circuits, where data is represented and processed in the form of pseudorandom bit-streams. Due to its main advantages, i.e., low-area hardware and fault tolerance, SC is emerging in several applications including image processing and neural networks. The design process for SC, however, is challenging. This paper introduces FUNSC, a design automation software tool, which efficiently generates SC circuits for computing mathematical functions. The tool features a user-friendly graphical user interface (GUI) that accepts input functions in their mathematical expressions and allows users to adjust the hardware complexity versus computational accuracy trade-off. To generate SC circuits for a desired function, FUNSC employs a two-step process: first, it approximates the function using a polynomial derived from Maclaurin series expansion, and then it applies Horner's Rule or Double-NAND Expansion rearrangements to map the polynomial to an SC circuit and displays generated schematic in the GUI. Our results demonstrate that the generated SC circuits achieve low error rates while successfully computing eligible functions.**

Keywords— Stochastic computing, design automation software tool, Horner's Rule, Double-NAND expansion

I. INTRODUCTION

Stochastic computing (SC) is an alternative to traditional binary computation where real values are represented as random (pseudorandom) bit-streams [1,2]. The value for a variable is encoded by the percentage of 1s in its representing bit stream. Computation under this framework provides advantages such as: 1) implementing certain computations with much lower cost circuits than their conventional counterparts, 2) built-in fault tolerance, and 3) dynamic configuration of accuracy by adjusting the length of the bit-streams. Due to these advantages, SC has been considered as an alternative to binary computation for various applications such as neural networks [3,4], image processing [5], digital filters [6], and control systems [7].

Multiplication is the most common computation to demonstrate the area efficiency of SC circuits. While a conventional 8-bit binary multiplier requires more than 100 logic gates, a single 2-input AND gate can implement multiplication in the SC context. As depicted in Fig. 1(a), when the two random bit-streams S_A and S_B are fed as inputs into the AND gate and the bit-stream S_C is produced as output, a bit of S_C is 1 if and only if the corresponding bits in S_A and S_B are also 1. Thus, the probability of a 1 occurring in S_C is equal to the multiplication of the probabilities of 1's occurring in S_A and S_B, i.e., $p_C = p_A \times p_B$ provided S_A and S_B are not correlated. For the example shown in Fig. 1(a), since $p_A = 0.5$ and $p_B = 0.8$, for the output $p_C = 0.4$.

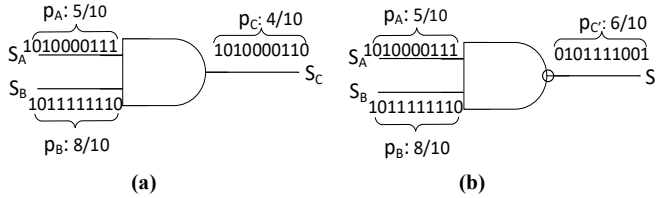


Fig 1. In the SC context: **(a)** a simple AND gate computes multiplication, i.e., $p_C = p_A \times p_B$, **(b)** a simple NAND gate computes one minus multiplication, i.e., $p_C = 1 - (p_A \times p_B)$.

Similarly, as shown in Fig. 1(b), a NAND gate computes one minus the multiplication of its inputs, i.e., $p_C = 1 - (p_A \times p_B)$. For the example shown in Fig. 1(b), since $p_A = 0.5$ and $p_B = 0.8$, the NAND's output calculates $p_C = 1 - (0.5 \times 0.8) = 0.6$.

Because of the benefits and applications of SC, a number of prior works have developed systematic design methods for SC circuits targeting particular class of functions or circuit properties. Reference [8] is the first attempt at the systematic synthesis of SC circuits for the computation of mathematical functions. The approach, which was subsequently expanded upon in [9] as ReSC, generates SC circuits that rely on Bernstein polynomials. However, functions which yield either 0 or 1 within the open interval (0,1) are not suitable for this method. As a different approach, STRAUSS [10] uses spectral transforms for designing SC circuits for the computation of mathematical functions. This technique is only applicable for functions with elements of the Fourier transform within the unit interval and needs external polynomial fitting tools to approximate other functions. In addition to combinational circuits, there are proposed techniques [11,12] for the design of sequential SC circuits. These techniques can design sequential SC circuits specifically limited to the computation of rational functions. Reference [13] introduces a method based on program synthesis. This approach involves iterative exploration of a high-dimensional design space, which imposes a limitation on synthesizing SC circuits with a small number of gates.

In this paper, we present *FUNSC*, an implemented design automation tool that generates SC circuits for mathematical function computation. FUNSC offers a user-friendly graphical user interface (GUI) where users can input information for their desired function. The tool then automatically generates and

979-8-3503-9463-4/24/$31.00 ©2024 IEEE

visually presents the schematic of an SC circuit capable of computing the input function to the specified accuracy level.

This paper is organized as follows. In the next section, Section II, we briefly explain two rearrangement methods used in the algorithm of our software. In Section III, we introduce the algorithm, technical aspects, and usage of FUNSC. In Section IV, we present results used for the evaluation of the software. Finally, we conclude the paper in Section V.

II. Background

The algorithm of the current version of FUNSC software works based on two polynomial rearrangement methods: Horner's Rule [14] and Double-NAND Expansion [15]. Therefore, in this section, we introduce these two rearrangements before using them in the next section. In the following two subsections, we separately explain both methods for a general polynomial, $P(x)$, of degree *n*, represented by its power form in (1).

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \tag{1}$$

A. Horner's Rule

In mathematics and computer science, Horner's Rule is used for polynomial evaluation[16]. Based on Horner's Rule, $P(x)$ in (1) can be rewritten as:

$$P(x) = b_0 \left(1 - b_1 x \left(1 - b_2 x \left(\dots (1 - b_n x)\right)\right)\right) \tag{2}$$

where $b_0 = a_0$ and $b_i = -\frac{a_i}{a_{i-1}}$ for $0 < i \leq n$. To better demonstrate this rearrangement, an example polynomial, $P_1(x) = \frac{2}{3} - \frac{1}{2}x + \frac{1}{4}x^2$, and corresponding rearrangement are shown in Fig. 2.

B. Double-NAND Expansion

The Double-NAND Expansion is another rearrangement method for polynomials, first proposed in [15]. Again, consider the polynomial, $P(x)$, represented in (1). Based on Double-NAND Expansion, (1) $P(x)$ can be rewritten as:

$$P(x) = 1 - b_0 \left(1 - x \left(1 - b_1 \left(\dots (1 - b_n x)\right)\right)\right) \tag{3}$$

where $b_i = \frac{1 - \sum_{k=0}^{i} a_k}{1 - \sum_{k=0}^{i-1} a_k}$ for $0 \leq i < n$ and $b_n = \frac{a_n}{1 - \sum_{k=0}^{n-1} a_k}$. An example polynomial, $P_2(x) = \frac{4}{5} + \frac{1}{15}x + \frac{1}{10}x^2$, and its corresponding rearrangement based on this approach are shown in Fig. 3.

III. FUNSC: The Proposed Software

In this section, we explain the algorithm and technical aspects of the FUNSC software for generating SC circuits to compute mathematical functions. One should note that, in general, only functions that map unit interval [0,1] to itself can be implemented by SC circuits.

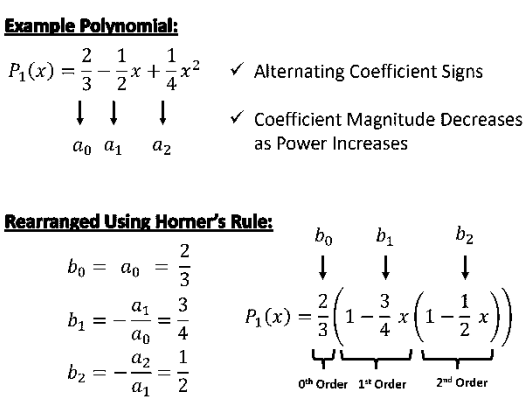


Fig 2. Example of Horner's Rule Rearrangement.

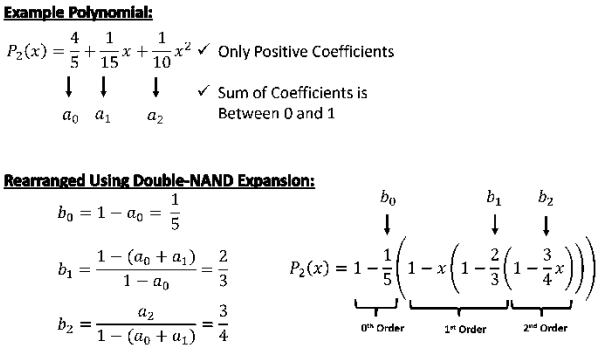


Fig 3. Example of Double-NAND Expansion rearrangement.

A. Algorithm

As the first step, the algorithm utilizes the Maclaurin Series approximation of a function, $f(x)$, to convert it into a polynomial. The Maclaurin Series of $f(x)$ is described in (4).

$$f(x) = \sum_{m=0}^{n} \frac{f^{(n)}(0)}{n!} x^n \tag{4}$$

where *n* is the degree of the approximating polynomial. The variable *n* should be increased if a more accurate approximation is needed as this will estimate $f(x)$ with a polynomial with a higher degree. Note that if the desired function is a polynomial, it is used directly with no approximation.

Next, the algorithm verifies whether the generated polynomial meets requirements for using Horner's Rule or the Double-NAND Expansion rearrangement. The requirements depend on the coefficients of the polynomial. For using Horner's rule, the coefficients of the polynomial must alternate signs and decrease in magnitude as the power increases. On the other hand, for implementing Double-NAND Expansion, all coefficients must be positive, and their sum should be less than or equal to one. FUNSC selects a specific rearrangement based on which set of requirements is satisfied.

Once the appropriate rearrangements are determined and its coefficients are calculated, the corresponding circuit diagram is generated. We explain this process by continuing the examples discussed in Figs. 2 and 3, with the result presented in Figs. 4(a) and 4(b). Note that the process only uses 2-input AND and NAND gates with SC equations described in Fig. 1.

To convert a rearrangement to a circuit diagram, the algorithm begins with the innermost parenthesis of a rearrangement expression. For the example rearrangement in Fig. 2, $P_1(x) = \frac{2}{3}\left(1 - \frac{3}{4}x\left(1 - \frac{1}{2}x\right)\right)$, the innermost parenthesis can be mapped to a *NAND* gate because its format is the same as the equation of the *NAND* gate presented in Fig. 1(b), i.e., $1 - a \times b$, where $a = \frac{1}{2}$ and $b = x$ (which is the input variable). This NAND gate is shown as G1 in Fig. 4(a) with its inputs connected to x and $\frac{1}{2}$. Gate G1 can now compute the value of $1 - \frac{1}{2}x$. The next innermost set of parentheses of $P_1(x)$ also is in the format of a *NAND* gate, with the inputs being three values multiplied: $\frac{3}{4}, x,$ and the output of gate G1. As three values are multiplied together, the process involves taking two of the inputs and passing them through an intermediate AND gate to calculate the product of two of the three inputs. As shown in Fig. 4(a), gate G1 inputs into an *AND* gate, G2, with another input of $\frac{3}{4}$. The output of gate G2 is then used as an input for gate G3, which is a *NAND* gate with another input value of x, resulting in the output of gate G3 being equal to the expression $1 - \frac{3}{4}x\left(1 - \frac{1}{2}x\right)$. The next innermost set of parentheses will complete the computation of $P_1(x)$, where the equation represented by the output of gate G3 is being multiplied by $\frac{2}{3}$. Multiplication is performed by an *AND* gate, so the final gate, G4, will be an *AND* gate with inputs of $\frac{2}{3}$ and the output of gate G3. With this final gate, the output of G4 represents the computation of $P_1(x)$. As shown with this example, the gates cascade into each other to derive an output computation.

Similar to Fig. 2, the rearrangement depicted in Fig. 3 can be translated into a SC circuit. Fig. 4(b) shows the circuit and how parentheses of $P_2(x)$ are mapped to the gates in the circuit. In this example, as with most Double-NAND Expansion rearrangements, all parentheses have the format of $1 - ab$ and multiply only two values. As a result, encapsulation and cascading become simpler, requiring only NAND gates. The exception is when a polynomial comprises terms with exclusively even or odd powers of x. In such cases, the algorithm employs an AND gate to generate x^2, which is subsequently utilized to generate other terms. We will demonstrate this in a later example using the function of $sec(x) - 1$.

B. Technical Aspects and Usage

The FUNSC software is an open-source software created in Python 3, and its source code can be found in the corresponding GitHub repository of our lab [17]. As a design

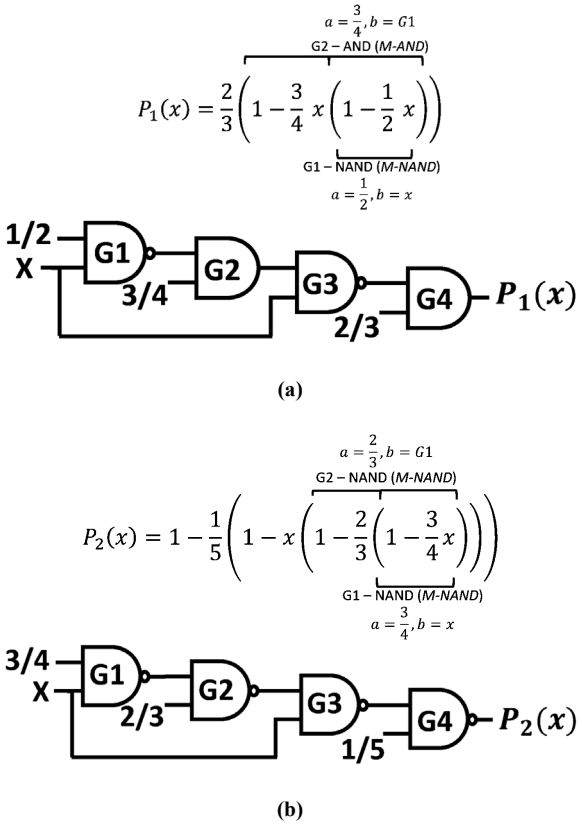


Fig 4. Mapping of Rearranged Polynomials into a Circuit Diagram: (a) Horner's Rule Example from Fig. 2, (b) Double-NAND Expansion Example from Fig. 3.

automation software tool, it has the capability to generate circuit diagrams for desired mathematical functions automatically, based on the basic information provided by its user.

To utilize the software, it is necessary to have the latest versions of LaTeX and Python installed, along with the required Python libraries imported: Scipy, NetworkX, MatPlotLib, Tkinter, Pathlib, SymPy, Pillow, MPMath, Schemdraw, and CairoSVG.

Once the software is downloaded and unzipped, the user can simply run the program by opening the executable file (Windows only), or by running the file "~/build/gui.py". Upon execution, the GUI will appear to the user with several fields, information boxes, and buttons as shown in Fig. 5.

To begin using FUNSC, the user is to enter the following information that the software will then use in its algorithm to generate the computing circuit diagram:

- A single-variable mathematical function in the blue text box in the middle of the GUI. This function must be an eligible function and written as a python expression: i.e., e^{-x} is written as $exp(-x)$.
- The independent variable of the function being used in the "Variable," field. By default, we utilize the

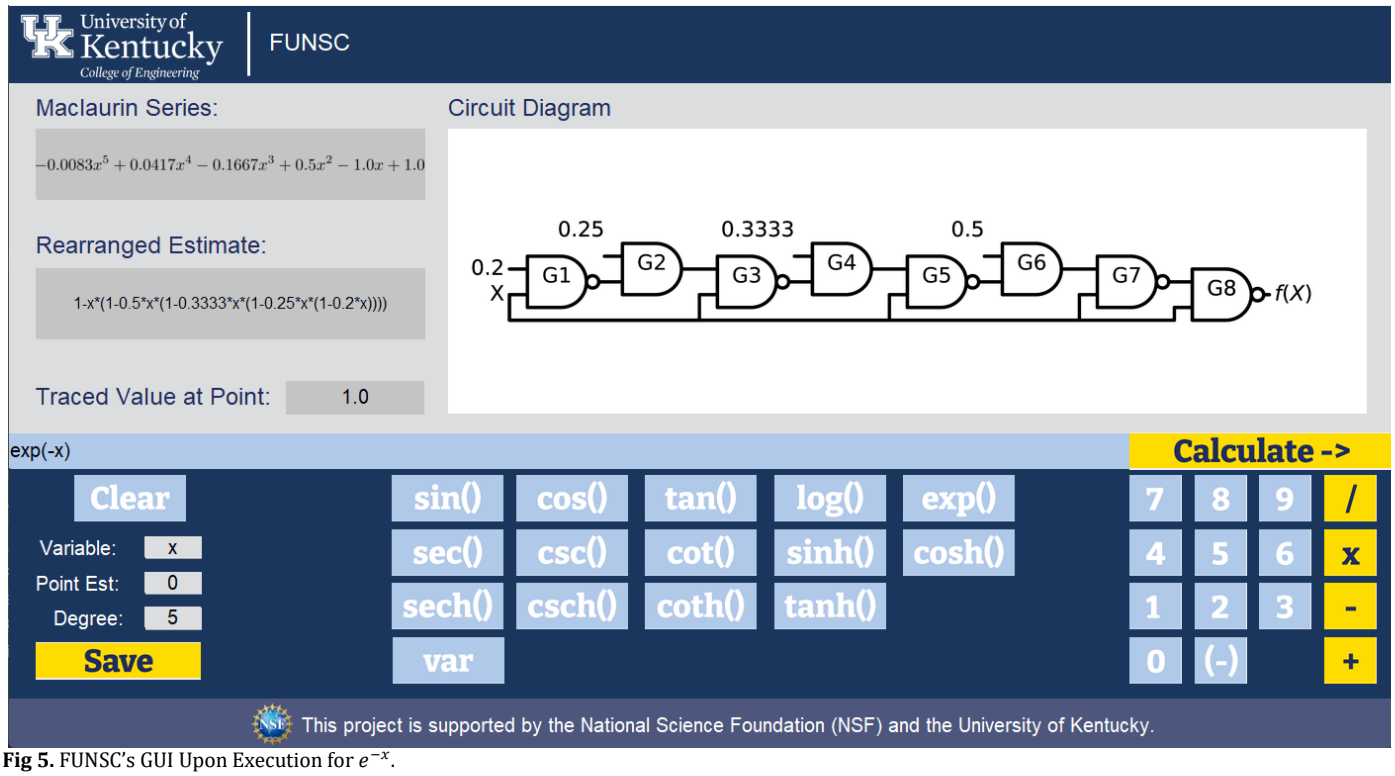


Fig 5. FUNSC's GUI Upon Execution for e^{-x}.

variable "x", but it is also possible to use other letters as variables.

- The point at which the function is being computed is defined in the "Point Est.," field.
- And the degree of polynomial estimation in the "Degree," field. This is equal to the value of n, in (4).

To enhance user convenience, some of the known supported functions are shown as blue buttons on the GUI. Along with these supported functions, buttons for numbers and operations presented in the bottom right of the GUI, give the appearance of a calculator. Users can press these buttons for the selected button to appear in the function textbox in the middle of the GUI.

Once the required information is added and ready to be submitted, the user can press the "Calculate ->" button as shown in Fig. 5 and the FUNSC software will begin running its algorithm for converting the function. Once the algorithm has finished, the software will populate the screen with the generated Maclaurin Series, rearranged estimate, value of the function at input point, and circuit diagram.

If the software cannot approximate a given function, an error will appear to the user with the approximated information.

The algorithm's execution flow involves incorporating various libraries and codes. When the user clicks the "Calculate ->" button, the execution process commences by taking the equation being approximated and converting it into a Python lambda expression. Subsequently, the Maclaurin Series Expansion is determined by implementing (4). To assist this implementation, the SciPy library is employed to calculate the derivative and factorial used in the summation of (4). Each coefficient is stored in a Python dictionary, with the corresponding term's degree assigned as the key for each coefficient.

Once the Maclaurin Series Expansion is determined, the algorithm verifies its coefficients pattern to determine which rearrangement type fits the expansion. This is where an error appears for functions that do not fit any rearrangement type that is not currently supported. If the expansion fits a supported rearrangement type (currently only Horner's Rule or Double-NAND Expansion), the rearrangement is then calculated using the conversions to the proper rearrangement. These conversions are the ones presented in Equations (2) and (3). Similar to the coefficients of Maclaurin Series Expansion, the rearrangement coefficients are also calculated and stored in a Python dictionary with the key being the polynomial term's degrees assigned to the coefficients.

After the rearrangement coefficients are determined, they are mapped to a circuit diagram using Schemdraw and the process explained in Section III.A. The algorithm will take the rearrangement type of a function and its calculated coefficients to determine the type of gate the coefficient will be input into as well as the order. For example, in a Horner's Rule rearrangement, the innermost parentheses will always be a *NAND* gate and outermost gate will change depending on if the coefficient b_0 equals one or another value. The rest of the gates between the innermost and outermost gates will alternate between *AND* and *NAND* gates when conforming to the fan-in constraint of two. The code uses this to predict which gate will

TABLE I. THEORETICAL VS. GENERATED VALUES OF MACLAURIN SERIES AND REARRANGED POLYNOMIALS FOR FUNSC FUNCTIONS.

Function	Replacement Type	Maclaurin Series Polynomial		Rearrangement Equation
e^{-x}	Horner's Expansion	*Theoretical:*	$-\frac{x^5}{5!}+\frac{x^4}{4!}-\frac{x^3}{3!}+\frac{x^2}{2!}-x+1$	$1-x\left(1-\frac{x}{2}\left(1-\frac{x}{3}\left(1-\frac{x}{4}\left(1-\frac{x}{5}\right)\right)\right)\right)$
		FUNSC:	$-0.0083x^5+0.0417x^4-0.1667x^3+0.5x^2-x+1.0$	$1-x\left(1-0.5x\left(1-0.3333x(1-0.25x(1-0.2x))\right)\right)$
$\tan \mathrm{h}(x)$	Horner's Expansion	*Theoretical:*	$-\frac{17x^7}{315}+\frac{2x^5}{15}-\frac{x^3}{3}+x$	$x\left(1-\frac{x^2}{3}\left(1-\frac{2x^2}{5}\left(1-\frac{17x^2}{42}\right)\right)\right)$
		FUNSC:	$-0.0539x^7+0.1333x^5+0.3333x^3+x$	$x\left(1-0.3333x^2(1-0.3999x^2(1-0.4045x^2))\right)$
$\frac{x^2}{10}+\frac{x}{15}+\frac{4}{5}$	Double-NAND Expansion	*Theoretical:*	$\frac{x^2}{10}+\frac{x}{15}+\frac{4}{5}$	$1-\frac{1}{5}\left(1-x\left(1-\frac{2}{3}\left(1-\frac{3x}{4}\right)\right)\right)$
		FUNSC:	$0.1x^2+0.0667x+0.8$	$1-0.2\left(1-x(1-0.6667(1-0.75x))\right)$
$\sec(x)-1$	Double-NAND Replacement	*Theoretical:*	$\frac{61x^6}{720}+\frac{5x^4}{24}+\frac{x^2}{2}$	$x^2\left(1-\frac{1}{2}\left(1-x^2\left(1-\frac{7}{12}\left(1-\frac{61x^2}{210}\right)\right)\right)\right)$
		FUNSC:	$0.0848x^6+0.2084x^4+0.5x^2$	$x^2\left(1-0.5\left(1-x^2(1-0.5832(1-0.2907x^2))\right)\right)$

be used for each coefficient and then utilizes Schemdraw to generate a circuit diagram by adding gates in order.

Finally, once a circuit diagram is generated by Schemdraw, it is exported as an SVG file and then placed on the corresponding box of the GUI for the user's viewing. To ensure future accessibility, both the Maclaurin Series Expansion and the circuit diagram are saved as PNG and SVG files in the "~/assets" folder within the software installation directory.

IV. RESULTS

We evaluated the performance of SC circuits generated by the FUNSC software for various mathematical functions, including those listed in the GUI. In this section, we present the result for four specific functions: e^{-x}, $\frac{x^2}{10}+\frac{x}{15}+\frac{4}{5}$, $\tanh(x)$, and $\sec(x)-1$. The first two functions use Horner's Rule while the next two use Double-NAND Expansion.

Table I provides a comparison of the Maclaurin Series and rearranged polynomials obtained through both theory and the FUNSC tool for these functions. The results in Table I demonstrate that the outputs from FUNSC align with their respective theoretical expressions. However, it is important to note that the precision of the coefficients is not exact. The FUNSC software employs a precision of 0.0001 (one hundred thousandth) for its coefficients, which may explain slight differences observed in some coefficients compared to their theoretical counterparts. For instance, the coefficient inputted into gate G4 for $\tanh(x)$ should ideally be 0.4, but it is presented as 0.3999. Table II illustrates the generated circuit diagram with the calculated coefficients for all four example functions.

Table III shows the mean absolute error of the example SC circuits obtained from Montecarlo simulation. For each function, the error is computed by averaging the absolute computational errors when input x changes from 0 to 1 in 0.01 steps, and the length of bit-streams is 1024 bit. Although the error is low, it can be further decreased by increasing the degree of approximating McLaurin series or the length of bit-streams.

TABLE II. GENERATED CIRCUIT DIAGRAMS FOR FUNSC FUNCTIONS.

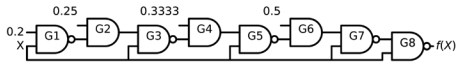
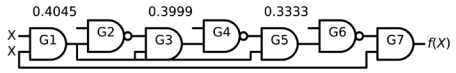
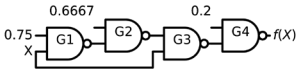
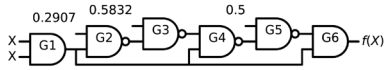
Function	Circuit Diagram
e^{-x}	
$\tan \mathrm{h}(x)$	
$\frac{x^2}{10}+\frac{x}{15}+\frac{4}{5}$	
$\sec(x)-1$	

TABLE III. MEAN ABSOLUTE ERROR (MAE) FOR COMPUTING EXAMPLE FUNCTIONS USING SC CIRCUITS GENERATED BY FUNSC SOFTWARE.

Function	e^{-x}	tanh(x)	$\frac{x^2}{10}+\frac{x}{15}+\frac{4}{5}$	sec(x) − 1
Error	0.0008	0.0140	0. 0040	0.0053

Additionally, for evaluating the computational performance of FUNSC for the example circuits, we collected the values of functions computed by the circuits at eleven equally separated points in the unit interval [0,1]. To visualize the computational accuracy of the generated circuits, Fig. 6 presents a side-by-side representation of the exact values of the functions (depicted by the blue line) alongside their computed values (represented by the red points).

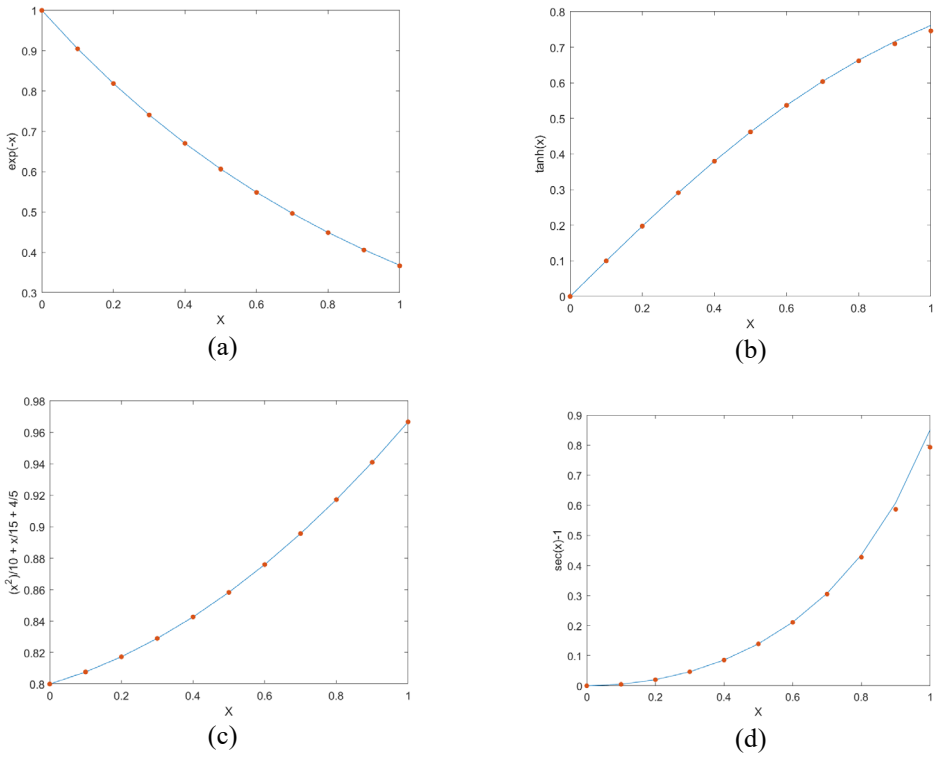


Fig 6. MATLAB Simulation Results. Theoretical function values (blue line) are compared to computed values (red dots).

V. Conclusion

Systematic approaches for synthesis of SC circuits have been proposed in prior work. In this paper, we take a step further by presenting an implemented design automation software tool that generates SC circuits for mathematical functions. The software features a user-friendly GUI that allows users to specify an input function and adjust the trade-off between computational accuracy and hardware complexity of the generated circuits.

The current version of FUNSC supports functions that satisfy the introduced requirements of Horner's Rule and Double-NAND Expansion, as specified by their Maclaurin Series Expansion. In our future work, we plan to enhance FUNSC by integrating additional synthesis methods into the software. This expansion will allow a wider range of functions to be eligible for automatic circuit generation. By incorporating these new methods, we aim to further improve the capabilities and versatility of the software in generating SC circuits for various mathematical functions.

References

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

[2] A. Alaghi and J.P. Hayes, "Survey of stochastic computing," in *ACM Trans. Embed. Comp. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013.

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

[4] A. Ren et al., "SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing," *Proc. Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 405-418, 2017.

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

[6] H. Ichihara and T. Sugino, "Compact and accurate digital filters based on stochastic computing," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[7] D. Zhang and H. Li, "A stochastic-based fpga controller for an induction motor drive with integrated neural network algorithms," IEEE Trans. Ind. Electron., vol. 55, no. 2, pp. 551–561, 2008.

[8] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in Proc. DAC, Anaheim, CA, USA, pp. 648–653, 2008,

[9] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," IEEE Trans. Comput., vol. 60, no. 1, pp. 93–105, Jan. 2011.

[10] A. Alaghi and J. P. Hayes, "STRAUSS: spectral transform use in stochastic circuit synthesis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34, 11, pp. 1770–1783, 2015.

[11] P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," in Proc. Int. Conf. Comput. Design, Montreal, QC, Canada, pp. 303–308, 2012.

[12] N. Saraf, K. Bazargan, D. J. Lilja, and M. D. Riedel, "Stochastic functions using sequential logic," in Proc. Int. Conf. Comput. Design, Asheville, NC, USA, pp. 507–510, 2013.

[13] V. T. Lee, A. Alaghi, L. Ceze, and M. Oskin, "Stochastic synthesis for stochastic computing," arXiv:1810.04756, 2018.

[14] K. K. Parhi, "VLSI digital signal processing systems: design and implementation." Hoboken, NJ, USA: Wiley, 1999.

[15] S. A. Salehi, Y. Liu, M. D. Riedel, and K. K. Parhi, "Computing polynomials with positive coefficients using stochastic logic by double-nand expansion," *Proc. Great Lakes Symp. VLSI*, pp. 471-474, 2017.

[16] Parhi, K. K. & Liu, Y. "Computing arithmetic functions using stochastic logic by series expansion." IEEE Transactions on Emerging Technologies in Computing (TETC), 2016.

[17] J.P. Huse and S.A. Salehi, "Function to stochastic circuit" https://github.com/CUT-Labs/FUNSC, 2024.