Stochastic Arithmetic Complex Number Operators

Jakub Šťastný
Department of Circuit Theory
Faculty of Electrotechnical Engineering, Czech Technical University in Prague
Technická 2, Prague 6, 166 27, Czech Republic
stastnj1@feld.cvut.cz

***Abstract*—This work presents a simple study of stochastic arithmetic complex number operators for addition and multiplication. Their usage is demonstrated by design of a sum of product circuit. As the stochastic complex number operators need more control random streams than stochastic rational number operators, we optimized the number of random generators used in the real circuit. In the end our sum of product circuit contains two LFSRs thus we analyzed the impact of the choice of the seeds for the LFSRs on the quality of the calculated results. By using exhaustive search over the LFSR state space we were able to reduce the output RMSE by 34% in comparison to choice of the equally spaced seeds over the LFSR state space.**

***Keywords*—Stochastic arithmetic, LFSR, LFSR initial state**

I. Introduction

Stochastic arithmetic circuits work with numbers represented by streams of random bits over which all the computations are performed. In such circuits number is represented as the mean value over a stochastic single bit data stream instead of its binary representation as with conventional parallel arithmetic. This brings the following advantages:

low-cost implementation in terms of occupied silicon area. Logic circuits calculating arithmetic operations are simplified up to single gates performing bit-wise logic operations over the stochastic bit streams [1].

tolerance to soft errors as a single bit flip in the stochastic stream does not ruin the calculations; an occasional bit flip is not statistically significant in the long stream of bits [2].

reduced design and verification effort and more reusable hardware compared with parallel arithmetic datapaths [2]. This is because of the implementation of the operators nearly does not depend on the precision of the processed numbers and of the simplicity of arithmetic operators design. Further, accuracy can be traded off with computation time [1].

On the other hand, this concept has also some disadvantages. The most important one is the low bandwidth: increase in precision by one bit requires exponential increase in the processing time [1].

Thanks to its advantages, stochastic arithmetic is an interesting concept. Indeed, there are many publications available presenting lots of applications and various building blocks – stochastic number generators [1], [3], arithmetic operators for rational numbers [1], [4], digital filtering [5], [6], more complicated functions [7], [8], [9], [10], and others.

The objective of this work is to present design of basic complex number stochastic arithmetic operators. A demonstrational design calculating sum of products is then implemented as Matlab model as well as in VHDL on the RTL level on Spartan6 FPGA platform. Since the final implementation contains two LFSR-based random number generators, we also analyzed the influence of the choice of the generator seed on the precision of the calculated results. By using exhaustive search over the LFSR state space we were able to reduce the output RMSE by 34% in comparison to choice of the equally spaced seeds over the LFSR state space.

II. Methods

A. Complex Number Representation

A complex number $z = \Re(z) + i\Im(z)$, where $\Re(z), \Im(z) \in \langle -V; V \rangle$ is represented as stochastic complex number (SCN) by two stochastic bit streams (SBS) carried by two lines w_r (real part) and w_i (imaginary part). The real and imaginary part of the number are then represented by the probabilities p_r and p_i that the respective line is at logic 1. Each line is encoded using single-line bipolar representation (form III in [1]), $\Re(z) = 2Vp_r - V$,$\Im(z) = 2Vp_i - V$. By $SN(p)$ we denote an SBS where probability of logic 1 is equal to p, $\Re(z)$ and $\Im(z)$ denote also stochastic number representing real and imaginary part of z.

B. Stochastic Complex Number Generation

The device used to generate stochastic numbers is called the Stochastic Number Generator (SNG), see Fig. 1. The SNG generating the stochastic equivalent of an N bit number is composed of the random number generator (RNG) and the comparator [1], [11] both of N bits. As the RNG, Linear Feedback Shift Register (LFSR) is widely used [1], [11], [8]. Complex number SNG (CSNG) can be built by doubling a rational number SNG, see Fig. 1a.

C. Summer and Multiplier

For the rational numbers a stochastic weighted summer operator is presented in [1]. This operator can be easily extended to complex number case by doubling it for real and imaginary parts, see Fig. 2a. The summer

ISBN 978-80-261-0602-9, © University of West Bohemia, 2016

Fig. 1. Complex stochastic number generator; a – naive implementation, b – shared design.

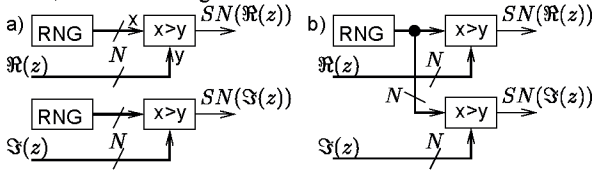
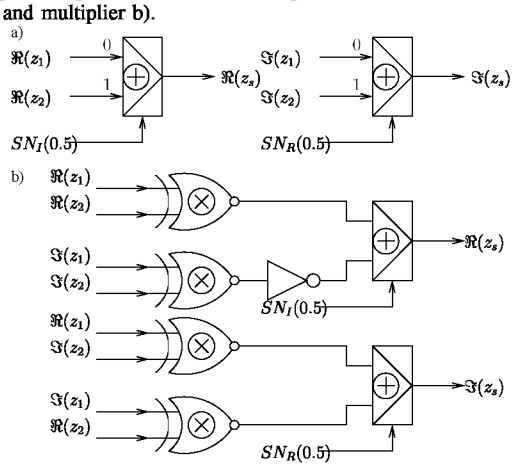


Fig. 2. Implementation of the complex number stochastic summer a) and multiplier b).



calculates $z_s = 0.5z_1 + 0.5z_2$. The range of the input number is $\langle -V; +V \rangle$, then after summing the range of the output is $\langle -2V; +2V \rangle$ so we need to multiply by 0.5 to prevent overflow and thus need here two SBS: $SN_R(0.5)$ and $SN_I(0.5)$.

Complex number multiplication $z_m = 0.5z_1z_2$ is performed using the well-known relationships

$$\Re(z_n) = 0.5\Re(z_1)\Re(z_2) - 0.5\Im(z_1)\Im(z_2), \quad (1)$$

$$\Im(z_n) = 0.5\Re(z_1)\Im(z_2) + 0.5\Im(z_1)\Re(z_2). \quad (2)$$

Complex multiplier is built using two rational number summers and four multipliers presented in [1], see Fig. 2b. To prevent overflow at the output summers we again have to multiply by 0.5 and need two additional random streams $SN_R(0.5)$ and $SN_I(0.5)$, while the rational number stochastic multiplier does not need any. In the summer as well as the multiplier, the multiplexer controlling streams shall be mutually independent of the stochastic streams at the inputs of the summing multiplexers [11].

D. Parallel Complex Number Decoding

For conversion of the CSN back to the binary format, two ADaptive DIgital Elements ADDIEs [1] in parallel (one for real, one for imaginary part) can be used. One N bit RNG is needed for both ADDIEs for their operation.

E. Test Circuit And Number Of RNGs

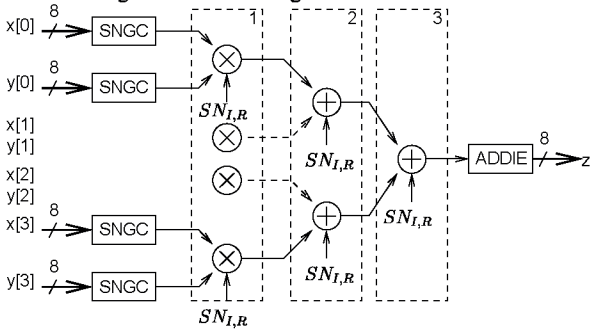
A simple circuit calculating complex sum of products (see Figure 3),

$$f = \sum_{n=0}^{3} x[n]y[n] \quad (3)$$

TABLE I
NUMBER OF RNGS FOR ALL THE DISCUSSED OPTIONS (NOP = NOT OPTIMIZED IN THIS OPTION).

Option	CNGs	MULTs	SUMs	ADDIE
naive	16	8	6	1
y shifted	1	nop	nop	nop
mux RI shared	nop	4	3	nop
mux levels	nop	1	2	nop
mux csng	nop	1	0	nop
all shared	1	0	0	nop
addie shared	1	nop	nop	0

Fig. 3. Data flow diagram of the test circuit.



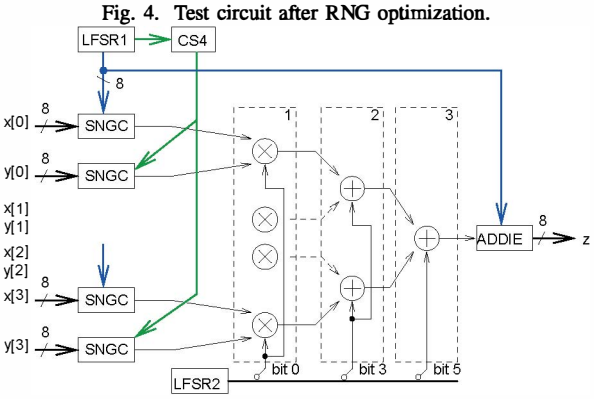
was implemented and its parameters analyzed. To generate $N = 8$ *bit* random numbers LFSR with polynomial of $x^8 + x^6 + x^5 + x^4$ was used.

While the implementation of the complex arithmetic operators is simple, they require more stochastic streams than the corresponding stochastic rational number processing. The *naive* implementation in Fig. 3 would need 31 RNGs, see Table I. This would need a lot of silicon area in the final implementation. To reduce the numbers of the RNGs we applied transformations based on Theorem 1 in [11]. First, the circuit requires up to 16 RNGs in the CSNGs. To reduce this number we can

1) share one RNG between real and imaginary part of the CSNG, see Fig 1b. Correlation between real and imaginary parts will be maximal at the CSNG output. This is not an issue until we need to calculate e.g., $\Re(z)\Im(z)$ – which is not used in (3). We would need only 8 RNGs, then.
2) share one RNG among all CSNGs generating $x[i]$, $i = 0 \ldots 3$ and the other one RNG among all CSNGs generating $y[i]$, $i = 0 \ldots 3$. Then all the $x[i]$ will be mutually fully correlated, the same for all the $y[i]$. This does not negatively influence results of the (3) as all multiplication operands ($x[i] \times y[i]$, $i = 0 \ldots 3$) are not mutually correlated. Afterwards we can apply on the LFSR output the circular shift transformation presented in [11] and share one RNG among all CSNGs, see Fig. 4. Circular shift by 4 bits is used since it gives the smallest correlation between generated stochastic numbers [11]. This way we would need only 1 RNG for all the CSNGs, see Table I, *y shifted*.

Second, we need to generate up to 14 stochastic bit streams (SBS) $SN_I(0.5)$ and $SN_R(0.5)$ to control

Fig. 4. Test circuit after RNG optimization.



the multiplexers in the stochastic arithmetic operator. These bit streams shall not be correlated with the data inputs of the multiplexers in the operators, [11]. Here we can

1) use the same SBS to drive both $SN_I(a)$ and $SN_R(a)$ in Fig. 2, to need only 7 SBS, Table I, *mux RI shared.*
2) use one SBS for all the operators in dashed box 1, see Fig. 3, then a different SBS for all operators in box 2, and one more SBS for the adder in box 3, see Table I, *mux levels*. This will reduce number of necessary SBS to 3.
3) derive all the three SBS driving boxes 1, 2, and 3 from LFSR1 by selecting e.g. bits, 0, 3, and 6 of its 8 bit output; the second LFSR2 will drive all the x CSNGs and after circular shift by 4 bits all the y CSNGs. Both 8-bit LFSRs will be the same with different seeds and we would need only 2 RNGs, see Table I, *mux csng*.
4) derive all three streams directly from the main LFSR driving the CSNGs, as in [11] to need only one RNG at all, see line *all shared* in Table I. MUXes will be controlled by inverted bits 0 (box 1), 3 (box 2), and 5 (box 3) of the LFSR output. This configuration is the same as in [11] for the edge detection circuit.

Third, we will need one more RNG for the output ADDIEs. Here we can easily use the RNG driving the CSNGs, see Table I, *addie shared.*

III. Results And Discussion

The architectural options were evaluated using the RMSE measure calculated as

$$RMSE = \sqrt{\frac{\sum_{n=0}^{N_r-1} |z_{calc_n} - z_{ideal_n}|^2}{N_r}}, \quad (4)$$

where z_{calc_n} is the stochastic circuit output in run n, z_{ideal_n} is the ideal output expected from the circuit, and N_r is the number of runs with different input x and y data. The lower the $RMSE$ value is, the better.

A. Number of RNGs

First, we had to evaluate all the options to reduce the number of RNGs needed by the circuit. To do this, we ran the following experiments.

TABLE II
RMSEs for the explored architectonical options. ADDIE rand = ideal averaging used.

Option	CSNG	MUXes	ADDIE	RMSE
naive	rand	rand	rand	1.034
y shifted	LFSR1	rand	rand	0.211
mux R/I shared	rand	rand	rand	1.040
mux levels	rand	rand	rand	1.031
mux csng	LFSR1	LFSR2	rand	0.473
all shared	LFSR1	LFSR1	rand	1.535
addie shared	LFSR1	LFSR2	LFSR1	0.500
seed	LFSR1	LFSR2	LFSR1	0.312

Referential experiment First, we implemented Matlab model of the whole circuit. As a reference we used *naive* implementation using Matlab random number generator `rand` instead of LFSR, implementing all the 31 random generators as independent ones. The output ADDIE was emulated by the SBS mean value calculation. To get statistically reliable data, we used bit streams of 16384 bits and the RMSE was calculated over $N_r = 64$ runs. The $x[i]$ and $y[i]$ inputs were different and randomly generated for each run and all other tests used the same set of 64 x and y vectors to get comparable results. See Table II for the achieved RMSE.

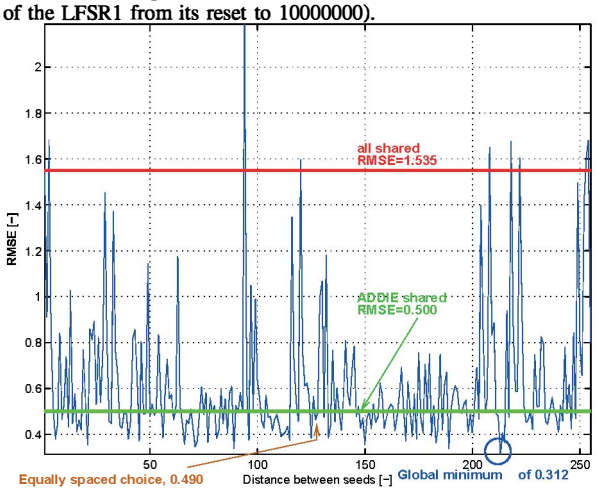
Sharing multiplexer control Two experiments were ran to evaluate impact on the shared MUX control, see Table II, *mux R/I shared* and *mux levels*. No degradation of the RMSE is observed, here. Sharing of random streams between real and imaginary multiplexers in the datapath does not influence overall precision of the circuit.

Sharing RNGs for the CSNG Experiment *y shifted* was ran to check how the sharing of the RNGs between the CSNGs influences result of the stochastic computation. While 8 bit LFSR was used as RNG for all the CSNGs (either with or without circular shift), multiplexers were driven by random streams generated by Matlab `rand` function to be able to directly compare results of this experiment and the *naive* one. The achieved RMSE (see Table II) is better than for the *naive* experiment as we use LFSR instead of pseudorandom numbers.

Sharing LFSR between MUX and CSNGs Possibility of sharing the LFSR between the MUXes and CSNGs was evaluated, see Table II, *mux csng* and *all shared*. All RNGs in the Matlab model are here already implemented as LFSRs, LFSR1 has seed of 10000000, LFSR2 of 11110000. For option *mux csng* better performance was achieved than with *all shared* option. RMSE of *all shared* option is likely hampered by the correlation between the MUX control stream and MUX input data, a corollary of Theorem 1 in [11]. Due to this we chose option *mux csng* with two LFSRs.

Finally, we ran a simulation of the circuit with ADDIE driven by LFSR used to drive all the CSNGs, see Table II, *addie shared.* No significant degradation of performance was observed.

Fig. 5. RMSE dependence on LFSR2 seed plotted over distance computed from LFSR state sequence (e.g., 50 at x axis means that the RMSE corresponds to the seed of LFSR2 which is the 50th state of the LFSR1 from its reset to 10000000).



B. Seeds for the LFSRs

Although two physically different LFSRs (LFSR1, LFSR2) are used for the CSNGs and for the multiplexer control, they both use the same polynomial. The only way how to decorrelate their outputs is by using different seeds. Usage of the seeds to decorrelate outputs of LFSRs is a widely known technique (e.g., [6], [9], [8]) and the usual approach is (e.g., [8]) to choose equally spaced seeds over the full period of the LFSR. Instead of this, we decided to run the exhaustive search over the complete 8-bit LFSR state space to find the best seed for the LFSR2.

We repeated the 64 runs of calculations with different $x[i]$ and $y[i]$ additional 255 times, the LFSR1 had fixed seed of 10000000 and the other LFSR2 seed was varied through all values from 00000001 to 11111111. Final RMSEs are plotted in Fig. 5 sorted according to the LFSR1 sequence.

Looking at the graph we can see that there are even few seeds providing in the configuration with two LFSRs worse results than if only one LFSR is used (x position 0 in the graph – both LFSR have the same output, also marked with the red line for *all shared* option). On the other hand, there are many seeds for which RMSE is lower than for the so far best *addie shared* option (see green horizontal line in Fig. 5). The global minimum of RMSE in Fig. 5 defines the final seed for the LFSR2 (10111110), see Table II, *seed*. In our case, choice of the equally spaced seeds for the LFSR1 and LFSR2 would result into worse RMSE (of 0.49) than we achieved by exhaustive search-based setup (0.312 for *seed*).

In addition to this experiment we repeated all the 255 runs also for different set of 64 x and y vectors to see if the curve in Fig. 5 is data-dependent. Here, obtained RMSE over distance curve was very similar to the one in the Figure thus the shape of the curve is given by the LFSR2 seed itself.

IV. Conclusions

We presented design of complex number stochastic operators as a simple extension of already used rational number operators. Complex number-processing circuits require more RNGs than rational number ones due to need to add in the multiplier and process real and imaginary channels, we thus reduced the number of RNGs using Theorem 1 from the [11]. After optimizations solution utilizing 2 LFSRs was chosen since it achieves three times better performance than circuit with only one LFSR (RMSE of 0.500 vs 1.535).

An analysis aimed to find the best value of the seed for the LFSR2 by exhaustive search was done. The dependency of the output RMSE on the distance between seeds of LFSR1 and LFSR2 in terms of LFSR sequence is not monotonic; by choosing the seed for the LFSR2 as the global minimum of the RMSE we were able to further reduce the output RMSE from 0.473 to 0.312 (by 34%). Exhaustive search approach also outperformed commonly used choice of equally spaced seeds by 36% (RMSE of 0.49 vs the final one of 0.312).

Finally, the testing circuit was implemented in VHDL language on the RTL level according to Fig. 4 and verified against its Matlab model. The design was implemented into xc6slx25-3 Xilinx Spartan 6 FPGA occupying 160 flip-flops, 160 LUTs, with operating frequency of 133 MHz (no pipelining was applied).

References

[1] B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science* (J. Tou, ed.), Advances in Information Systems Science, pp. 37–172, Springer US, 1969.

[2] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, pp. 92:1–92:19, May 2013.

[3] P. Jeavons, D. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *Information Theory, IEEE Transactions on*, vol. 40, pp. 716–720, May 1994.

[4] P. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 36, pp. 603–606, Apr 1988.

[5] Y. Liu and K. Parhi, "Lattice fir digital filter architectures using stochastic computing," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 1027–1031, April 2015.

[6] Y.-N. Chang and K. Parhi, "Architectures for digital filters using stochastic computing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 2697–2701, May 2013.

[7] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computers,*, vol. 63, no. 6, pp. 1473 – 1486, June 2014.

[8] S. L. T. Marin, J. M. Q. Reboul, and L. G. Franquelo, "Digital stochastic realization of complex analog controllers," *IEEE Transactions on Industrial Electronics*, vol. 49, pp. 1101–1109, Oct 2002.

[9] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pp. 154–161, Oct 2011.

[10] B. Brown and H. Card, "Stochastic neural computation. i. computational elements," *Computers, IEEE Transactions on*, vol. 50, pp. 891–905, Sep 2001.

[11] 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*, pp. 361–366, Oct 2014.