electronics

MDPI

Article

Optimal Stochastic Computing Randomization

Christiam F. Frasser [1], Miquel Roca [1,2] and Josep L. Rosselló [1,2,*]

[1] Electronic Engineering Group, Industrial Engineering Department, Universitat de les Illes Balears, 07122 Palma de Mallorca, Spain; christian.franco@uib.es (C.F.F.); miquel.roca@uib.es (M.R.)
[2] Health Research Institute of the Balearic Islands (IdISBa), 07010 Palma de Mallorca, Spain
* Correspondence: j.rossello@uib.es

Abstract: Stochastic computing (SC) is a probabilistic-based processing methodology that has emerged as an energy-efficient solution for implementing image processing and deep learning in hardware. The core of these systems relies on the selection of appropriate Random Number Generators (RNGs) to guarantee an acceptable accuracy. In this work, we demonstrate that classical Linear Feedback Shift Registers (LFSR) can be efficiently used for correlation-sensitive circuits if an appropriate seed selection is followed. For this purpose, we implement some basic SC operations along with a real image processing application, an edge detection circuit. Compared with the literature, the results show that the use of a single LFSR architecture with an appropriate seeding has the best accuracy. Compared to the second best method (Sobol) for 8-bit precision, our work performs 7.3 times better for the quadratic function; a 1.5 improvement factor is observed for the scaled addition; a 1.1 improvement for the multiplication; and a 1.3 factor for edge detection. Finally, we supply the polynomials and seeds that must be employed for different use cases, allowing the SC circuit designer to have a solid base for generating reliable bit-streams.

Keywords: stochastic computing; LFSR; seeding; correlation

Citation: Frasser, C.F.; Roca, M.; Rosselló, J.L. Optimal Stochastic Computing Randomization. *Electronics* **2021**, *10*, 2985. https://doi.org/10.3390/electronics10232985

Academic Editor: Akash Kumar

Received: 24 October 2021
Accepted: 22 November 2021
Published: 1 December 2021

1. Introduction

Stochastic computing (SC) has emerged as a possible solution for Neural Network hardware implementation [1,2] and also as a way to accelerate the computation in different applications such as image processing [3] or Deep Learning (DL) for inference [4,5] and training [6,7]. It presents several advantages compared to traditional computing, such as noise resiliency [8], low signal transmission delay [9], low power consumption, and a small footprint area [10].

Two main SC-based codifications exist for representing variables: unipolar and bipolar coding. In unipolar coding, the bit-stream (BS) value x is seen as the probability of obtaining a 1 at a random position in x. This corresponds to counting the number of 1's N_1 and the number of 0's N_0 along the BS and computing $x = P(x = 1) = N_1/(N_1 + N_0)$. This value lies in the interval [0,1]. On the other hand, bipolar coding represents the BS values in the interval [−1,1]. To accomplish this, zeroes are weighted as -1, while ones are weighted with a $+1$, so that $x^* = (N_1 - N_0)/(N_1 + N_0)$, where the upper index $*$ denotes bipolar coding. Operating over single BSs allows cost operations such as multiplication, to be implemented with single logic gates. For instance, Figure 1a shows how two unipolar BSs $x = 4/8$ and $y = 6/8$ are multiplied using a single AND gate.

In order to operate in the SC domain, a Stochastic Number Generator (SNG) circuit must be employed. The most commonly used circuit in the literature is based on a RNG circuit and a single comparator [11,12], as shown in Figure 1b. Whenever the digital input value X is greater than the RNG value R, the stochastic output x is set to '1', otherwise it is set to '0'. If the RNG sequence is uniformly distributed in the interval of all possible values of X, the probability $P(x)$ is proportional to the number X.

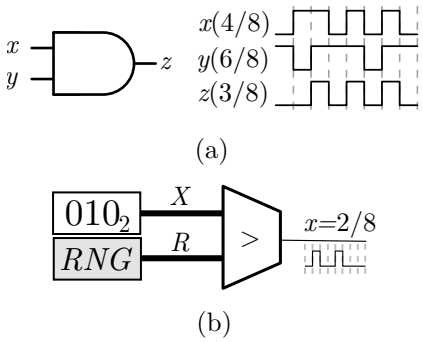


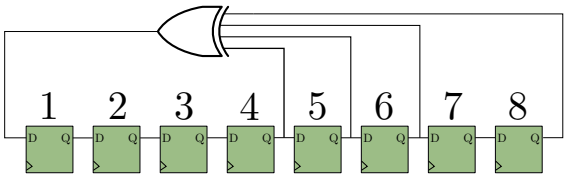
Figure 1. (**a**) Stochastic multiplication using an AND gate. (**b**) Stochastic number generator for an 8 BS length.

The overall SC system performance is highly dependent on the RNG quality. The reasons are twofold. Firstly, the area employed for this part of the system is much higher than that of the computational part, occupying around 80% or even 90% of the total footprint of the system [13]. This is especially critical for applications requiring a high fan-in. However, secondly, the randomness quality can significantly affect the precision of those operations requiring non-correlated signals, as in the case of stochastic multiplication, for instance. For these reasons, finding the best RNG in terms of area and low correlation is a major concern to address when designing real SC applications.

Different approaches have been proposed to tackle these issues. Low-discrepancy sequences such as Halton [14] or Sobol sequences [15] deal with the low-correlation matter. This sort of sequences produce pulse signals with ones and zeroes uniformly spaced; this mitigates the random fluctuations in the BSs generated. The problem they face is the area employed for generating such sequences. Normally, different base-counters [14] or least significant zero detectors plus storage arrays [16,17] are utilized, thus increasing the hardware overhead. On the other hand, area saving mainly comes in one mode: the Linear-Feedback-Shift-Register (LFSR) circuit.

An LFSR is a circuit based on a shift register and a linear function of its previous state connected to its input. Normally, the linear function is produced by connecting exclusive OR gates to different points (known as taps) in the state registers. The way of connecting these taps in the circuit is known as *primitive polynomials*, which can be expressed using two different notations: polynomial or binary notation. In polynomial notation, the tap connections are expressed as $1 + \sum_{i \in [T]} x^i$, where T are the register taps selected. In binary notation, the taps connected are expressed as $\sum_{i \in [T]} 2^{i-1}$. Figure 2 shows an LFSR circuit where the inputs of the XOR gate are connected to registers 8, 6, 5, and 4; its polynomial is thereby expressed as $x^8 + x^6 + x^5 + x^4 + 1$ or 184 in binary notation. Different polynomials allow different deterministic streams with different lengths to be produced. There exists a finite number of polynomial configurations (depending on the LFSR resolution) that produce a maximal length sequence of $N = 2^b - 1$ cycles, where b is the number of registers used in the circuit (bit-resolution). For instance, a 10-bit LFSR has at least 60 different polynomials that produce maximal length sequences [18]. Those polynomials which generate a maximal stream length are known as primitive polynomials. The reason the LFSR sequence has one cycle missed is because there is a prohibitive state, where the LFSR is locked-up in case it enters (when all bit registers are zeros in case the XOR gate is used).

The starting value of the LFSR is named as the *seed*. Figure 2 shows an LFSR sequence with a seed set to 255. After N clock cycles, the sequence will roll over again from the starting seed point. This is one of the reasons LFSRs are not truly RNGs; they are, rather, pseudorandom numbers, since we can predict the next state of the sequence if we know the current state (something uncertain in a true RNG).



clock cycles	LFSR value
0	255
1	254
2	252
...	...
254	127
255	255
256	254
257	252

Figure 2. 8-bit LFSR circuit with a primitive polynomial configuration equal to $x^8 + x^6 + x^5 + x^4 + 1$ (184 in binary notation). Every $N = 255$ cycles, the sequence is repeated periodically, starting from the seed value (blue color).

LFSR is the easiest and smallest circuit to produce a pseudo-RNG (from now referred as RNG; we explicitly use the adjective *True* when meaning True random), albeit presenting a high correlation behavior [19]. Different works have been proposed in the literature for exploiting the advantages of LFSR while mitigating its shortcomings. Basically, most of the works presented focus on sharing the same LFSR between different SNGs but alleviating the correlation effects that this method raises. The work carried out by Z. Li et al. [20] is a good example of this; their approach is based on a DFF insertion technique (adding a DFF to the line to uncorrelate the BS with itself), where the DFF circuit aims to uncorrelate one of the BS from the other. This technique performs successfully when employing a True Random Number Generator (TRNG), but for stand-alone LFSRs this is not the case. The LFSR shows a high level of autocorrelation only if isolated with a single DFF, as demonstrated in [19]; therefore, this approach is not efficient. Hideyuki Ichihara et al. [21] suggested a technique for sharing as many LFSRs as possible by employing a circular shift at the LFSR output. This method allows generation of two non-correlated signals with no hardware overhead; therefore, it reduces the area employed and deals with the error at the same time. Along the same line, another relevant study was proposed by H. Joe and Y. Kim [22]. They dug deeper into the different ways of connecting the same LFSR to produce the smallest correlation impact between signals using a wire exchanging technique. Their results showed better performance in comparison with other LFSR sharing methods. Another approach, introduced by F. Neugebauer et al. [19], tried to increase the randomness of the LFSR outcome by adding a nonlinear boolean circuit. This method decreased the correlation impact at the cost of hardware overhead. J. Anderson et al. [23] explored an interesting approach based on the effect of the seeds on the accuracy in SC systems when employing LFSRs. They demonstrated that an efficient selection of seeds in the circuit improved the accuracy. For this, the authors explored the whole space to find the best seeding set. This is affordable as long as the exploration space is bounded (small bit-precision). However, for more complex analysis cases (as more complex systems or higher bit-precision), a higher-level procedure must be employed (such as metaheuristic techniques [24,25]). Nevertheless, as previously introduced, SC advantages are displayed when small bit-precision is used ($\leq$8 bits, for the case of multiplication). Therefore, the whole exploration process is reasonable and there is need for a higher-level heuristic.

Despite the fact that some solutions have been produced, none of them guarantees good accuracy and a small footprint area at the same time for correlation-sensitive circuits

such as the SC quadratic function, the scaled addition, and the multiplication. These circuits are the driving force in the SC realm, and high demand applications such as image processing or DL employ them.

In this work, we explore in more detail how the LFSR circuit could be better exploited as a RNG source in SC by making a careful selection of the seeds employed, with the purpose of finding the best BS generator technique for different application requirements. Our contribution is four-fold:

1. We show the LFSR seeding impact over different correlation-sensitive circuits. We demonstrate that, if seeded properly, the LFSR circuit is the most accurate RNG compared to other methods in the literature, as long as it is computed for a complete sequence period. This, in fact, comes with the advantage of using the cheapest RNG found since the early beginnings of SC [11].
2. We demonstrate that the LFSR may achieve low autocorrelation behavior when isolated properly, something that has been totally overlooked in the literature up to now. This fact has an impact on the design of commonly used operations such as the stochastic square function x^2.
3. We prove our claims in real hardware implementations. Using an FPGA device, we perform a real case application for image processing, implementing an edge detection circuit.
4. We provide the seeds that must be employed when using LFSRs for different use cases, offering SC designers a direct RNG setting.

2. Seeding Impact on Correlation

LFSRs are valuable in different applications such as fast digital counters [26], whitening sequences [27], cryptography [28], circuit testing [29], and, indeed, SC circuits. Despite its advantages, the use of LFSR sequences for SNG raises some difficulties. Unlike in the case of TRNGs, in that of LFSRs, the premise that all bits in the stochastic stream are independent of each other does not apply anymore. Consecutive values of the LFSR sequence are highly dependent, as they are a shifted version of past states. Each bit in the sequence, if taken separately, possesses good randomness characteristics, but when it is seen as a whole binary number (as it is normally used in SC), the ideal randomness quality disappears. This is a real issue for commonly used stochastic functions such as the quadratic function x^2, which is carried out with a single AND gate (*unipolar* coding) and a single D-FF register. The purpose of the D-FF is to uncorrelate the stochastic signal from itself by adding a delay cycle. This is true for stochastic signals generated by TRNGs, but it is not fulfilled for the LFSR case. Moreover, when operating multiple BS, different seed combinations produce different results, and since the sequence is periodic, the same error is observed in each cycle. This contrasts with the TRNG, in which the error fluctuates, converging to zero as the integration time increases. For these reasons, finding optimum seed combinations is a major issue when employing LFSRs as the random number source of the circuit.

Despite the fact that LFSR has been the preferred RNG in SC real implementations, previous works do not provide a careful analysis of the LFSR seeding; still less do they offer a method to efficiently choose the best seeds to operate. The work that comes closest to doing so was that carried out by J.Anderson et al. [23]. They explored whether there existed a suitable set of seeds to improve the accuracy of stochastic computing systems. They demonstrated that a good selection of seeds increased the accuracy of SC circuits for the same bit precision, and that shorter streams with optimum seeding had the same or better accuracy than longer bit streams with random seeding. Nevertheless, although they present empirical evidence for their results, the authors do not provide what those seed combinations are and how to select them a priori, i.e., without iterating the whole *seed sweeping* computation (Monte-Carlo way) for the application. One of the problems with their method is that we need to perform a trial and error procedure: a quick task if small circuits are evaluated but a heavy task for a large implementation. In this

section, an analysis of the seeds of the LFSR is presented, the aim being to overcome the aforementioned shortcomings.

Suppose two BSs are generated from two independent LFSRs with the same polynomial but having different seeds. Suppose these two BSs are multiplied. The question that arises is: does any different couple of seeds generate the same result? If not, how does the seeding affect the overall outcome? Take for instance the operation shown in Figure 3. The x signal represents the value $4/15$, while y represents $5/15$. The same LFSR polynomial is used to generate each signal but with different seeds. Two versions of y (y_1 and y_2) are generated for comparison purposes. For the y_1, we picked the next value in the sequence of the LFSR$_x$ (see LFSR$_{y1}$ with the seed highlighted in red) as the seed. For y_2, we picked the fourth value in the sequence of the LFSR$_x$ (see LFSR$_{y2}$ with the seed highlighted in green). As seen, due to the fact that the LFSRs have the same polynomial, the y_1 stream is a shifted version of y_2. This makes the x signal 1's match the y_1 and y_2 1's in different times, producing different outcomes in the final operation. In order to see the impact of seeding, the AND operation between x and y_1, y_2 is presented in z_1, z_2, respectively. For the case of $z_1 = x \wedge y_1$, the result is $2/15 \approx 0.13$, whereas for $z_2 = x \wedge y_2$, the result is $1/15 \approx 0.06$. As shown, z_2 represents more accurately the ideal value ($\approx$0.08), with an absolute error of $\approx$0.02. This fact shows that y_2 is less correlated than y_1, respect to x. This short and simple example demonstrates how seeding has an impact on the overall results in SC correlation sensitive circuits (as is the case of multiplication), so careful selection of seeds must be carried out to obtain the most accurate results.

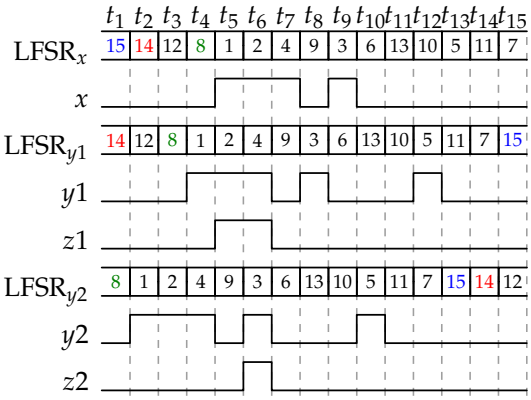


Figure 3. Impact of seeding for stochastic multiplication ($x \times y$, where $x = 4/15$ and $y = 5/15$). Depending on the seed used to generate the stochastic BS (y_1 or y_2), different results are produced (z_1 or z_2).

Expanding the preceding example, Figure 4 shows the Mean Absolute Error (MAE) for different seeds when x and y are multiplied considering all possible values. Once again, the same polynomial is employed for the two LFSRs. Two different bit resolutions are taken into account in the analysis: 6-bit and 8-bit. Instead of performing the analysis by taking as reference the seed values as in the previous example, this time we took the difference between the seed position (*seed index*) in the sequence $\Delta idx = idx_{\text{SEEDy}} - idx_{\text{SEEDx}}$. Taking Figure 3 as an example, x is generated with the seed index 0, y_1 is generated with the seed index 1, and y_2 is generated with the seed index 3. In essence, the seed index corresponds to the value taken by the LFSR sequence at time $t_{index+1}$. Formally, the MAE for every seed index is calculated as:

$$MAE = \frac{1}{N}\sum_j |y_j - \hat{y}_j| = \frac{\sum_{X=0}^{2^b-1}\sum_{Y=0}^{2^b-1}|x \wedge y - \bar{x}\bar{y}|}{2^{2b}} \quad (1)$$

where b is the bit-precision, x and y are the BS generated when converting X and Y to SC domain, and $\bar{x}$, $\bar{y}$ the expected value of x and y, respectively.

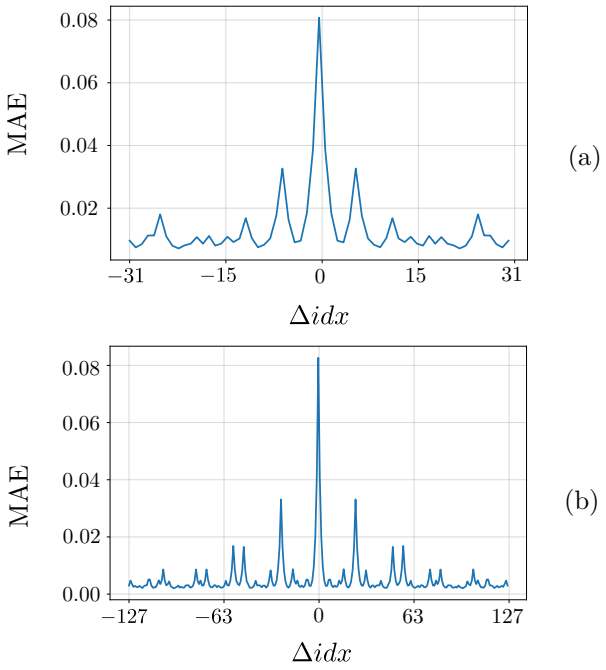


Figure 4. MAE for different seed indexes measured from a 6-bit LFSR (**a**) and an 8-bit LFSR (**b**) for the stochastic multiplication. LFSR polynomials are shown in Appendix A Table A2.

As shown in both plots, the maximum error occurs when x and y are generated with the same seed ($\Delta idx = 0$). However, as the $idx_{\mathrm{SEED_y}}$ moves away from the $idx_{\mathrm{SEED_x}}$, the error tends to diminish (with some resonant error peaks throughout), until we move to the *further* seed index $\Delta idx = \pm(2^{b-1} - 1)$. The behaviour can be seen as a mirror if we take the center as the point of reference. The takeaway from these figures is that *the difference between both seed indexes* $|\Delta idx|$ *is the real issue, not the LFSR seed values*.

It is worth noting that as we move away from $\Delta index = 0$, some seed indexes present an error resonant behaviour (high peaks in the plots); however, as we move closer to the further index, the peaks are mitigated in an exponential way. The reason for this phenomenon can be better understood by observing Figure 5. The blue line shows the normalized value of the 8-bit LFSR sequence for the first 127 cycles. The orange, in turn, shows the MAE for the first 127 seed indexes. Since both variables share the same axis ($idx = cycles$), we can plot them in the same figure. As shown, the LFSR sequence presents similar patterns periodically (see arrows in the plot). Considering that the LFSRx seed is at $idx = 0$, if the LFSRy seed coincides with one of these initial-pattern values, then a resonant error occurs, indicating that both sequences (the one starting with seed index 0 and the one starting with the initial-pattern) have a high degree of correlation, i.e., they are similar. Therefore, if noncorrelated operations are to take place (as is the case of the multiplication), it is mandatory to avoid these seed values for the generation of the second BS.

Figure 6 shows the MAE histogram for an 8-bit LFSR and a 10-bit LFSR implementation. As can be appreciated from the LFSR-8 instance, selecting random seeds can lead us to have more than twice the error than if the seeds are selected intentionally. According to the measurements, there exists a 79% probability of choosing an inaccurate seed (seeds with a MAE greater than 0.002 in Figure 6 for the LFSR-8 case). Moreover, 90% of the LFSR-10 seeds produce an MAE of less than 0.002, which is the same MAE we obtain when the seeds are efficiently chosen for the LFSR-8. We can thereby achieve similar accuracy if the pairing seed selection is carried out deliberately for lower resolution LFSR, instead of doing it in a random way for higher resolution LFSR, saving hardware resources, latency, and power.

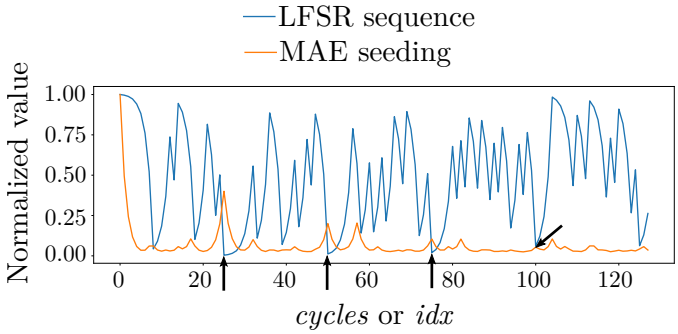


Figure 5. Resonant seeds (marked by arrows) produced by values which are more correlated with respect to the first seed index. There exist some correlated patterns in the LFSR sequence (blue line), which produce high correlation between stochastic signals, thereby inducing higher MAE (orange line).

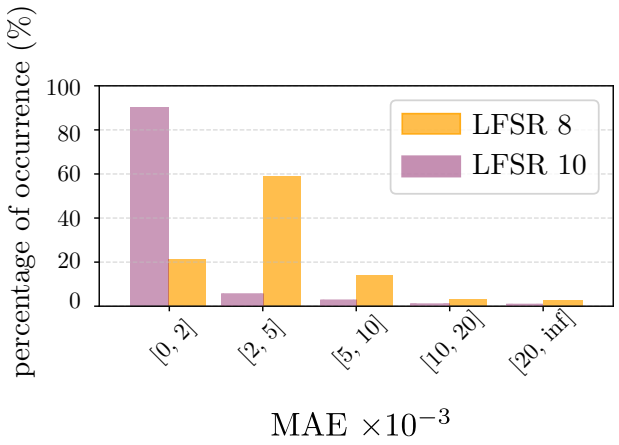


Figure 6. MAE histogram for the 8-bit and 10-bit LFSR examples.

The Absolute Errors (AE) for different seed indexes when varying x and y are presented in Figure 7. The worst case (Figure 7a) occurs when we generate the y BS with seed index 0, producing maximum correlation between both input BSs. As seen, the maximum error is produced around the center, when the variance of the signal is maximum ($x = y = 0.5$), taking maximum error values of up to 0.25. The second plot (Figure 7b), shows the error behaviour for the first resonant peak of Figure 5 (seed index 25). On this occasion, the maximum error is spotted when one of the signals is 0.5 and the other is at 0.25 and 0.75, raising error values of up to 0.12, almost half of the worst case. The best seed (index = 97) is presented in Figure 7c, where the maximum error rises to no more than 0.011, an order of magnitude less than the first resonant peak. Finally, the further seed index (idx = 127) is shown in Figure 7d. As can be seen, its behavior is very similar to the best seed index case (Figure 7c), presenting a maximum value of 0.016; 0.005 higher than the best seed case. As shown, the outcome pattern varies depending on the seed employed; the seeding effect is therefore a major concern when utilizing LFSRs as the random source for generating stochastic BSs.

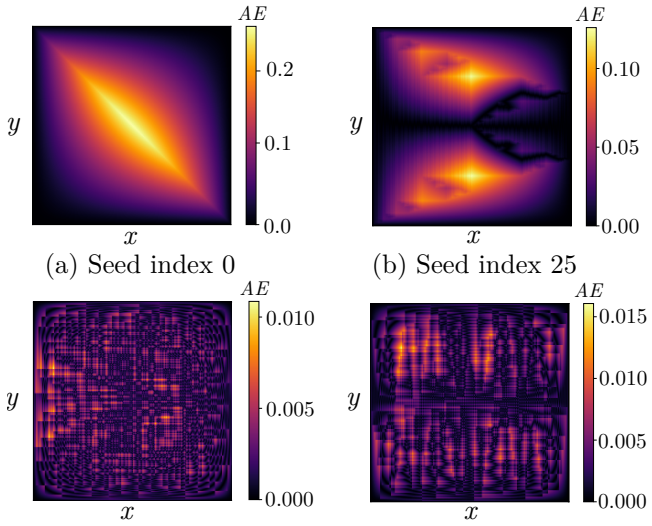


Figure 7. Absolute Error (AE) in the multiplication when varying x, y (from 0 to 1) using different seeds in the 8-bit LFSR example. Seed index = 0, which is the worst case, is shown in (**a**). Seed index = 25, which is one of the resonant error peaks, is shown in (**b**). Seed index = 97, which is the best case, is shown in (**c**). Furthermore, seed index = 127, which is the further seed index is shown in (**d**).

3. Seeding Impact on Autocorrelation

The study of LFSR seeding can be extended to a very important area of research in SC: autocorrelation. A BS is said to be non-autocorrelated when there exists a low dependency between its consequent bits. When autocorrelation occurs, an isolator circuit could be used to uncorrelate the BS with itself, allowing operations such as the stochastic square function (x^2) to be performed. In other words, the autocorrelation measures how well an isolator circuit is able to uncorrelate the BS with a delayed version of itself [19]. The most common circuit employed in the literature as an isolator is the D-Flip-Flop (DFF) [11,30]. Inserting a DFF in the BS line uncorrelates the BS with itself or with another BS *generated with the same RNG*, as explained by Z. Li et al. [20]. This is something especially exploited in their work in the quest to reduce the area overhead produced by the RNG circuits, since inserting a single DFF in the line is much simpler than inserting a complete independent RNG. Nevertheless, although they claim that their technique works for any BS generated with a circuit generator structure made of any RNG (LFSR method included) and a comparator, the truth is that for the LFSR case, this property does not apply, as will be analyzed in this section. The LFSR circuit has a high degree of autocorrelation, as demonstrated in [19], and a single DFF isolator insertion is insufficient to uncorrelate the BS with itself.

Let us see firstly how the LFSR behaves compared to other RNG found in the literature using an autocorrelation metric. F. Neugebauer et al. [19] define an autocorrelation metric based on the Box-Jenkins function [31], which is employed in the NIST Engineering Statistics Handbook [32]. The definition they provide is as follows: Let x be a BS with a sequence : $x_1, x_2, \ldots, x_N$, where N is the BS length, and let $\bar{x}$ be the expected value of x. The autocorrelation A_k of x will be:

$$A_k = \frac{\sum_{i=1}^{N-k}(x[i]-\bar{x})(x[i+k]-\bar{x})}{\sum_{i=1}^{N}(x[i]-\bar{x})^2}, \tag{2}$$

where k is the number of cycles delayed (the number of DFF inserted in the line). Autocorrelation values close to 0 indicate a good independence of the BS with its k delayed version, whereas a high absolute value indicates a bad independence.

Figure 8 shows a comparison of the autocorrelation between the LFSR for different k values and other RNG methods. The measure is taken for different x values using 8-bit

precision. As seen, LFSR for $k = [1,2]$ presents an autocorrelation value higher than that of the TRNG (TRandom). The average value of the LFSR A_1 (LFSR k = 1, blue line) is 0.29, showing an increase in A as x moves away from the middle point. That is why inserting a single DFF when using an LFSR produces poor precision results when the BS value moves away from the center point ($x = 0.5$). For the case of inserting 2 DFF in the line (LFSR k = 2), the average value decreases to 0.12, but still shows high peaks of more than 0.2, still performing poorly for most applications. The ideal case, which is the TRandom, presents an average A value of -0.038, measured from 1000 different random samples (in the figure, one of the samples chosen arbitrarily is plotted), having better autocorrelation value than the LFSR for $k = 1,2$. This conclusion is supported by [19] in their work, discarding the LFSR standalone circuit for stochastic operations such as the quadratic function, and proposing the SBoNG method as an approach to circumvent the problem. The SBoNG method is based on connecting a nonlinear Boolean function to the output of the LFSR. The function is performed by a combinational circuit called SBox [33]. The SBox circuit is normally implemented as a LUT (although it can be implemented using logic gates) and is constrained to 4-bit inputs, limiting its use to 4-bit-multiple RNGs. In their paper, the authors compare the SBoNG method with the LFSR circuit, but only for $k = 1,2,3$; concluding that SBoNG performs better for stochastic operations demanding low autocorrelated BSs. Nevertheless, as Figure 8 shows, the LFSR with the precise number of delay elements ($k = 5$) performs much better than the TRandom and the SBoNG implementations, with an average A value of 0.007; 5.1, and 2.2 times better than the TRandom and the SBoNG methods, respectively. It must be said that for the case of the SBoNG measures, our results differ from the ones presented in the original paper [19]. This is because they evaluated the SBoNG circuit by varying randomly, for every iteration of the test, the LFSR seed and the initial state of the circuit (see details in original paper). However, to conduct real digital circuit implementations without increasing the amount of resources to generate random values for every iteration, we fixed the seed and initial-state values. These values were found by running 1000 tests and selecting the best case result, i.e., the LFSR-seed and initial-state couple, which performed the lowest autocorrelation average value.

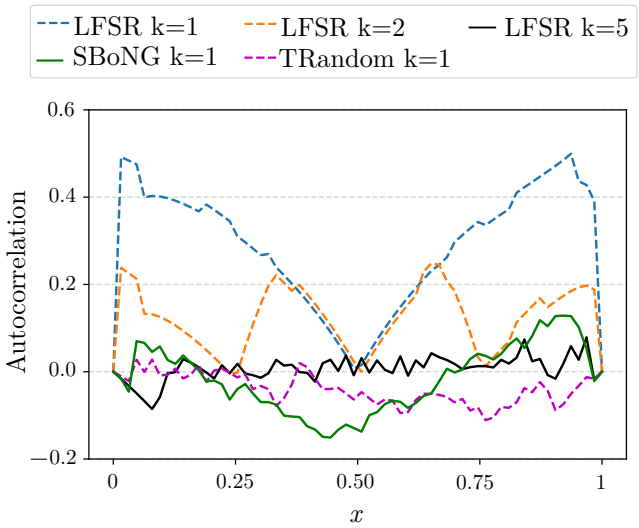


Figure 8. Autocorrelation comparison for the LFSR, TRNG (TRandom), and SBoNG methods when initial input x value has different quantities.

The reason LFSR with 5 DFF performs better can be understood by analyzing the MAE values of the first seed indexes in the 8-bit LFSR multiplier. Table 1 shows the numerical values. A k delayed version of x is equivalent to taking the seed index k (see the example of Figure 3). That is why the $k = 5$ version has a low autocorrelation value, because the seed index 5 is the first minimum MAE, as can be observed in the table. Additionally, the first

two seed indexes, which represent $k = 1, 2$ in Figure 8, have a high MAE, corresponding to a high autocorrelation level. It is therefore expected that if we employ only one or two isolators, the LFSR will perform poorly, since it corresponds to employing the two first seed indexes for operating. However, if we embed the correct number of DFF, we can have accurate results.

Table 1. MAE for the first 9 seed indexes of the 8-bit LFSR circuit. The minor value is highlighted in bold.

Seed Index	MAE
1	0.04089
2	0.02006
3	0.00972
4	0.00479
5	**0.00287**
6	0.00290
7	0.00503
8	0.00507
9	0.00299

4. Experimental Results

In this section, we conduct different experiments to test the LFSR seeding impact in SC applications. We first test three important operations employed in the SC realm that are correlation sensitive: the quadratic function, the scaled addition, and the multiplication. Finally, we implement a real image processing application over an FPGA device: an edge detection circuit.

Under otherwise noted, the polynomials employed for the different bit-precision LFSR implementations are the ones described in Table A2. We have selected these primitive polynomials arbitrarily from all possible ones, since the variation of the best seed MAE for each of them is negligible (see the MAE std column in Table 2).

Table 2. MAE statistics for the best seeds of each polynomial sequence considering 4 to 8 bit precision. MAE was calculated for the multiplication operation.

Bit Precision	Possible Primitive Polynomials	MAE Avg	MAE Std ($\times 10^{-3}$)
4	[9, 12]	0.021	2.971
5	[18, 20, 23, 27, 29, 30]	0.012	0.873
6	[33, 45, 48, 51, 54, 57]	0.007	0.238
7	[65, 68, 71, 72, 78, 83, 85, 92, 95, 96, 101, 105, 106, 114, 119, 120, 123, 126]	0.004	0.141
8	[142, 149, 150, 166, 175, 177, 178, 180, 184, 195, 198, 212, 225, 231, 243, 250]	0.002	0.057

4.1. Quadratic Function

For comparison purposes, we evaluated an important operation in SC, namely, that of the Stochastic Quadratic Function (SQF). Its use is extended to very interesting applications such as the implementation of the SC-Gaussian function employed by V. Canals [34] for describing the probability density of a continuous random variable. Additionally, it has recently been used by J. Li et al. in the normalization block circuit for a SC neural network implementation [35]. We evaluated this function circuit, as an example, in an effort to see the effect of correctly isolating the BS generated by the LFSR and compare it with other RNG approaches. The SQF is built by a multiplication gate (AND gate for unipolar coding and XNOR for bipolar coding), and an isolation block to uncorrelate the input signal with itself, as shown in Figure 9a.

Table 3 presents the MAE for the *bipolar* SQF using different RNG methods proposed in the literature. We vary the number of isolation elements for different bit widths, where the number next to the D termination corresponds to the number of elements employed (DFFs). The RNG methods evaluated were: the LFSR method, the Sobol sequence method [16], the SBoNG method [19], and the TRNG method (TRandom). For a fair comparison, the

table presents the same number of isolation circuits for the Sobol and LFSR methods; whereas, for the SBoNG and TRandom methods, only one isolator is employed, since their best results are obtained in this manner [19]. We measured the MAE for a complete LFSR period ($2^b - 1$ cycles), where b is the bit-precision. The best result of each column is highlighted in bold. For the case of the SBoNG and TRandom methods, we followed the same experimental setting described in Section 3 for measuring the autocorrelation. A graphical comparison for the 8-bit precision case is shown in Figure 9b.

Table 3. MAE comparison for the bipolar SQF using different RNGs. The best result for each column is highlighted in bold.

RNG Method	Bit Precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D	0.111	0.134	0.149	0.158	0.162
LFSR 2D	**0.084**	0.067	0.068	0.074	0.079
LFSR 3D	0.089	**0.047**	0.036	0.036	0.038
LFSR 4D	0.089	0.054	**0.030**	**0.022**	0.019
LFSR 5D	0.124	0.080	0.083	0.030	**0.012**
LFSR 6D	0.116	0.068	0.112	0.085	0.012
LFSR 7D	0.122	0.053	0.067	0.118	0.016
LFSR 8D	0.122	0.064	0.046	0.061	0.019
Sobol 1D [16]	0.091	0.080	0.083	0.086	0.088
Sobol 2D [16]	0.311	0.323	0.328	0.331	0.332
Sobol 3D [16]	0.100	0.084	0.083	0.085	0.087
Sobol 4D [16]	0.222	0.256	0.265	0.269	0.270
Sobol 5D [16]	0.127	0.105	0.098	0.094	0.092
Sobol 6D [16]	0.244	0.290	0.312	0.323	0.328
Sobol 7D [16]	0.202	0.121	0.096	0.090	0.089
Sobol 8D [16]	0.202	0.318	0.373	0.401	0.415
SBoNG 1D [19]	0.162	–	–	–	0.079
TRandom	0.360	0.126	0.106	0.047	0.025

Observing the table and Figure 9b, the one with the lowest precision is that of the Sobol sequence. For 8-bit precision, its MAE rose to 0.415 (Sobol 8D), which was 2.5 times worse than the LFSR with only one DFF (LFSR 1D) and 36 times worse than the best LFSR case (LFSR 5D). Moreover, its best case (Sobol 3D), was worse than all of the other methods, excluding the one delay instance of the LFSR (LFSR 1D). Observing the plot, we see that for negative input values the behaviour was rather imprecise, tying the output to 0 for input values between -0.5 and 0 and behaving far from ideally even for the positive range. The reason for this is that the Sobol sequence is a deterministic low-discrepancy sequence for which every bit in the stream is highly dependent on the past bits, leading to high autocorrelation values.

For the SBoNG method, we could measure only the MAE for 4-bit and 8-bit precision, as the method is restricted to 4-bit multiples. As regards 8-bit precision, SBoNG had a similar performance to LFSR when two isolators were employed (LFSR 2D). This is an interesting observation which can help designers to save a good deal of resources, as one extra DFF (for the LFSR case), is cheaper than the whole SBoNG circuit with fixed seeds. Observing its behaviour in Figure 9b, the output can take negative values when the input value is near 0. This is caused by the imprecision of the BS generated by this RNG, because the sequence, for a complete period of 2^b, does not cover all the possible values of x.

Finally, the LFSR method had the best performance overall when selecting the correct number of isolation elements. It was on average more than twice as good as the second best method for all bit-widths. As far as 8-bit precision was concerned, it performed

7.3, 6.6, and 2.2 times better than the best Sobol, SBoNG, and TRandom methods, respectively.

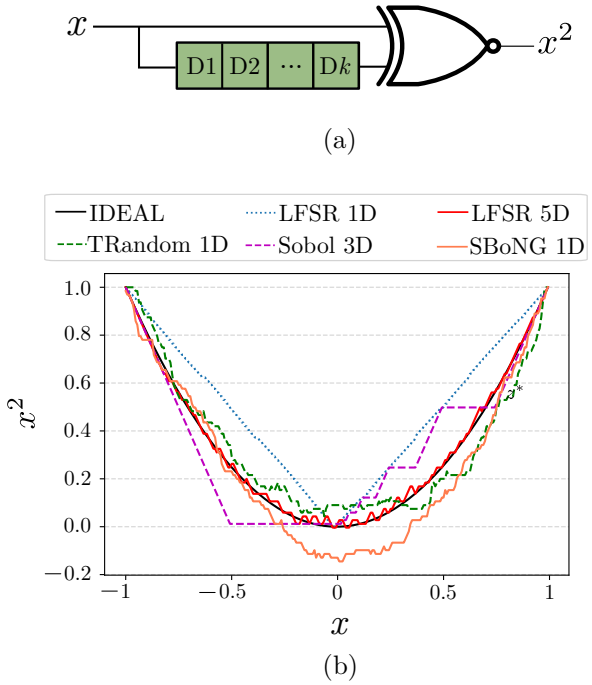


Figure 9. Bipolar Stochastic Quadratic Function (SQF). The circuit implementation is shown in (**a**) for different k DFF. The operation outcome comparison is shown in (**b**) for different RNGs: LFSR, Sobol, SBoNG, and TRNG (TRandom).

Figure 10 shows the best outcomes of each method. As seen, the Sobol method barely changed its MAE for different bit widths. When $b > 6$ TRandom became better than Sobol, approaching LFSR performance. Therefore, if accuracy is the main requirement of our application, the LFSR remains the best option if we analyze the data for a complete BS period.

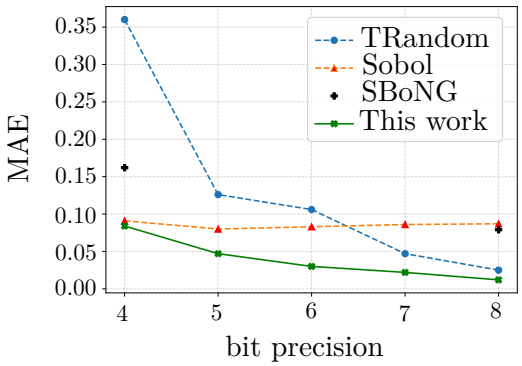


Figure 10. MAE for the best outcomes of each method employing the SQF.

4.2. Scaled Addition

The stochastic addition is essential for almost any SC application. The most commonly used circuit is based on a MUX [11]. This circuit needs the input signals to be uncorrelated with the selector signal to have accurate results. We analysed the performance of different RNGs proposed in the literature to mitigate the correlation error of the scaled addition and compared them with the LFSR seeding method put forward in this work.

We examined seven different RNG methods. The first one attempted to tackle the correlation problem effect using LFSR with different polynomials (different taps, defined here as DTaps). Ideally, we would expect that LFSR with different polynomials would be

totally uncorrelated, and thus a precise outcome would be guaranteed. The second one was based on the DFF technique insertion proposed by Z. Li et al. [20]. In order to reduce both the computation latency and area overhead, they proposed using a single LFSR to generate multiple BSs, but inserted DFFs in their lines, assuming that a single DFF would do the task of uncorrelating. The third one was the circuit suggested by H. Ichihara et al. [21]; again, it was based on the LFSR sharing technique. However, in this method, the authors suggested making a circular shift to the LFSR bit lines that were connected to one of the converters. In their experiments, they concluded that the best shuffle choice was when a circular shift of $k = b/2$ was chosen, where b was the bit precision. The next method was an improvement of the circular shift method. H. Joe and Y. Kim [22] proposed an efficient technique to permute the lines in the LFSR; they called it the Wire Exchanging Technique (WET). They demonstrated that a fixed setting of permutation reduced the relative error compared with other LFSR sharing techniques, saving area and power. The WET swaps the bits by pairs and then exchanges the wires symmetrically as follows. Suppose the LFSR output bit order is $[b_{n-1}, b_{n-2}, ..., b_1, b_0]$, where b_n is the bit at position n, being n the bit-precision. The WET is defined as:

$$Swap(b_{n-1}, b_{n-2}, \ldots, b_1, b_0) = b_{n-2}, b_{n-1}, \ldots, b_0, b_1$$
$$Exchange(b_{n-2}, b_{n-1}, \ldots, b_0, b_1) = b_1, b_0, \ldots, b_{n-1}, b_{n-2}.$$

In this manner, only using one LFSR but permuting the wires, the correlation was mitigated. The fifth method taken into account was an approach that has aroused the interest of the SC community lately: the low-discrepancy sequence, especially the Sobol sequence [16]. Low discrepancy (LD) sequences have been commonly used in speeding-up Monte-Carlo simulations [17], but S. Liu and J. Han [16] employed it for the purpose of lowering the computing latency on SC. The LD sequences distribute the highs and lows along the BS uniformly, allowing the outcome to converge faster to the expected result, thus lowering the latency. The Sobol sequence has been employed in different studies in the SC realm, such as in SC edge detection circuits [36], as well as in SC convolutional neural networks [37]. It is based on an address generator circuit formed by a Least Significant Zero (LSZ) detector circuit and a storage array, normally implemented as a Look-Up-Table (LUT). As a sixth method, we introduce the proposed work, which consists of the stand-alone LFSR circuit with the best couple of seeds for addition (see Appendix A Table A1). Finally, a groundtruth instance was added using a TRNG. Similar to the previous experiment, this method was evaluated running 1000 different iterations.

The circuit employed for the experiment is set out in Figure 11a. All of the tests were done varying all possible binary inputs X, Y and the MAE was calculated with respect to the ideal outcome (similar to Equation (1) but with the scaled addition operation).

Table 4 shows the MAE results for the different RNG evaluated with different bit-precision. We measured the MAE for a complete LFSR period ($2^b - 1$ cycles). The polynomials employed for the second LFSR (RNG2 in Figure 11a) were [9, 18, 33, 83, and 175] for the 4, 5, 6, 7, and 8 bit precision, respectively.

As shown, the 1-DFF insertion method [20] (LFSR 1D) had the worst performance for bit widths greater than 6. The correlation introduced by inserting only one DFF impacted the scaled addition to the extent of producing an approximate constant MAE value of 0.04 for all bit-widths. As the bit precision increased, the error gap in relation to other methods grew exponentially, with a factor of 42 with respect to the best method for 8 bits. This was expected, as only one DFF insertion is insufficient to decorrelate two BS generated with the same LFSR, as discussed in Section 3 and verified in Section 4.1.

The TRNG (TRandom) method presented the second-worst performance. However, as in Section 4.1, when the bit width increased, the MAE decreased exponentially.

It should be noted that the gap between the LFSR-WET and Sobol methods decreased as the bit-precision became higher.

Finally, the best-seed LFSR method (this work) outstripped all other methods for all bit-precision cases; a 1.5 factor was observed with respect to the second best method

(Sobol [16]) for all the bit-widths. Contrary to intuition, using the same LFSR taps produced better results than using different taps.

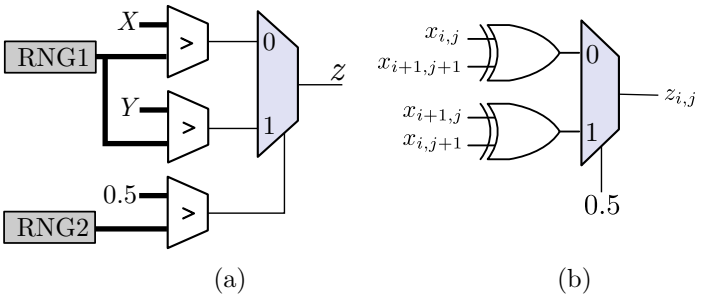


Figure 11. (**a**) Scaled addition test setting. (**b**) SC Roberts edge detection circuit exploiting correlation.

Table 4. MAE for the scaled addition circuit using different RNGs. The best result for each column is highlighted in bold.

RNG Method	Bit Precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D [20]	0.0458	0.0433	0.0424	0.0420	0.0418
TRandom	0.0964	0.0654	0.0461	0.0321	0.0226
LFSR CShift [21]	0.0250	0.0222	0.0109	0.0106	0.0053
LFSR DTaps	0.0307	0.0163	0.0102	0.0062	0.0039
LFSR WET [22]	0.0297	0.0134	0.0063	0.0031	0.0015
Sobol [16]	0.0250	0.0121	0.0060	0.0030	0.0015
This work	**0.0167**	**0.0081**	**0.0040**	**0.0020**	**0.0010**

4.3. Multiplication

In this experiment, we evaluated how the different RNGs performed for stochastic multiplication using the same RNG conditions used in Section 4.2. Table 5 shows the MAE comparison. The test circuit is displayed in Figure 12, calculating the MAE as in Equation (1). For the proposed LFSR method, the seeds employed were the best found for the multiplication operation (see Table A1 in Appendix A).

Similar behavior to that obtained in Section 4.2 was observed for the different methods (being ordered similarly to Table 4). However, for the multiplication the Sobol method outperformed LFSR-WET in all bit-precision. Moreover, it did slightly better than this work for the 6 and 7 bit-widths. Nevertheless, it only performed better by a factor of 1.01 and 1.05, respectively; whereas ours performed better by a factor of 1.3, 1.2, and 1.1 for the 4, 5, and 8 bit cases, respectively.

Stochastic multiplication is the core operation for measuring the correlation between two BSs [38]. We can therefore conclude that our method produced less correlated signals than state-of-the-art RNG when analyzing a complete integration period. This claim is confirmed when implementing a real use case application where SC has been widely used, as detailed in the next experiment.

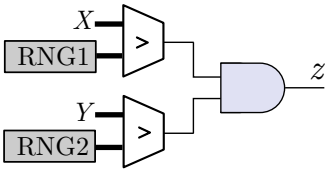


Figure 12. Multiplication test setting.

4.4. Edge Detection

Different image processing algorithms have been proposed in the literature where SC has been demonstrated to perform better than traditional computing with an imperceptible error increase [39,40]. One of them is the Roberts' cross algorithm for edge detection. The algorithm computes the input image in a 2×2 moving pixel window by the following formula:

$$z_{i,j} = 0.5(|x_{i,j} - x_{i+1,j+1}| + |x_{i+1,j} - x_{i,j+1}|),$$

where $x_{i,j}$ represents the pixel value at location (i, j), and $z_{i,j}$ represents the outcome pixel value.

Table 5. MAE for different RNG circuits evaluated for different bit-precision in the multiplication operation. The best result of each column is highlighted in bold.

RNG Method	Bit Precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D [20]	0.0330	0.0365	0.0388	0.0401	0.0409
TRandom	0.0762	0.0514	0.0356	0.0252	0.0174
LFSR CShift [21]	0.0284	0.0243	0.0172	0.0138	0.0095
LFSR DTaps	0.0217	0.0136	0.0102	0.0071	0.0046
LFSR WET [22]	0.0284	0.0177	0.0115	0.0066	0.0040
Sobol [16]	0.0255	0.0129	**0.0071**	**0.0038**	0.0022
This work	**0.0190**	**0.0108**	0.0072	0.0040	**0.0020**

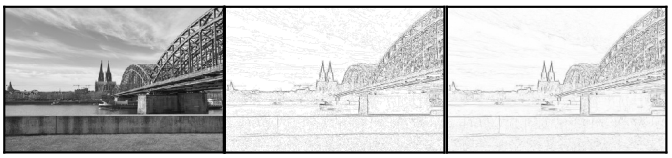
Figure 11b shows the SC circuit, considering that all input BSs are correlated, and the selector signal is uncorrelated with respect to the inputs [3]. Using this circuit, we measured the MAE for the different RNG methods introduced in the preceding Sections 4.2 and 4.3, except for the TRNG, since a real implementation on an FPGA was carried out. The FPGA employed was a Cyclone V 5CSEBA6U23I7 included on the DE10-Nano board from Terasic, running at 50 MHz. The input BSs was generated with the first RNG of each method and the selector with the second RNG (see Figure 11b).

The MAE for each method is shown in Table 6. A noteworthy pattern displayed is that as the bit precision became higher, the proposed work presented a higher improvement factor than the others. For instance, for the 4-bit precision column, an improvement of 1.04, and 1.06 times was observed compared to the Sobol and LFSR-WET methods, respectively, while a 1.3 times increase was observed for the 8-bit precision.

Figure 13 shows the edge detection results for the 4-bit and 8-bit precision using the proposed LFSR seeding method. These results demonstrate that a good seeding in the LFSR presented the most accurate results for real image processing implementations.

Table 6. MAE for different RNG circuits evaluated for different bit-precision in the Roberts edge detection circuit. The best result of each column is highlighted in bold.

RNG Method	Bit Precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D [20]	0.0262	0.0152	0.0089	0.0055	0.0034
LFSR CShift [21]	0.0261	0.0174	0.0105	0.0088	0.0053
LFSR DTaps	0.0261	0.0169	0.0100	0.0063	0.0043
LFSR WET [22]	0.0259	0.0149	0.0082	0.0043	0.0020
Sobol [16]	0.0256	0.0146	0.0080	0.0042	0.0020
This work	**0.0245**	**0.0134**	**0.0069**	**0.0035**	**0.0015**



(a) (b) (c)

Figure 13. Edge detection outcome using SC Roberts edge detection circuit for different bit-precision: Input image (**a**); and output image for 4-bit precision (**b**) and 8-bit precision (**c**), using the proposed LFSR seeding method.

5. Conclusions

We have presented a solid base for the LFSR as the best RNG circuit in the SC domain, if computed for a complete sequence period. Considering different cases of study, we demonstrated that LFSR block is an appropriate circuitry for improving accuracy in key SC operations. Compared with other RNG methodologies applied to SC, the proposed method showed better results for different SC operations such as the quadratic function, the scaled addition, and the multiplication (for this last case, it presented a similar performance to the Sobol method [16]). Furthermore, the proposed method did better than other RNG performances in real case applications such as an edge detection circuit. We obtained these results using both simulations and FPGA implementation. To conclude, we offer to SC designers the guidelines for setting their LFSRs for different case applications, in this way, saving them time, while simultaneously assuring good results.

Author Contributions: Conceptualization, C.F.F. and J.L.R.; methodology, C.F.F. and J.L.R.; software, C.F.F., M.R. and J.L.R.; validation, C.F.F. and M.R.; formal analysis, C.F.F., M.R. and J.L.R.; investigation, C.F.F. and J.L.R.; resources, C.F.F., M.R. and J.L.R.; data curation, C.F.F. and J.L.R.; writing—original draft preparation, C.F.F. and J.L.R.; writing—review and editing, all authors; visualization, C.F.F., M.R. and J.L.R.; supervision, J.L.R.; project administration, M.R. and J.L.R.; funding acquisition, M.R. and J.L.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Ministerio de Ciencia e Innovación and the Regional European Development Funds (FEDER) under grant contracts TEC2017-84877-R, PID2020-120075RB-I00 and PDC2021-121847-I00. Grant TEC2017-84877-R funded by MCIN/AEI/10.13039/501100011033 and by ERDF A way of making Europe. Grant PID2020-120075RB-I00 funded by MCIN/AEI/10.13039/501100011033. Grant PDC2021-121847-I00 funded by MCIN/AEI/10.13039/501100011033 by the European Union NextGenerationEU/PRTR.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. LFSR Tables

Table A1 presents the seed indexes and values that must be employed in the second LFSR considering the first LFSR seed index is 0 for different bit-precisions. The "ADD" column shows the best seed index for the scale addition operation. The "MUL" column shows the best seed index for the multiplication operation.

Table A2 shows the tap configuration used in this work for the LFSR at different bit-precisions.

Table A1. Best relative seed index for different use cases depending on the bit-precision.

Bit-Precision	Best ADD Index	Best MULT Index
4	3	2
5	4	3
6	5	23
7	6	52
8	7	97

Table A2. Tap configuration used in the estimation of the best seeds.

Bit-Precision	Polynomial	Binary Notation
4	$x^4 + x^3 + 1$	12
5	$x^5 + x^3 + 1$	20
6	$x^6 + x^5 + 1$	48
7	$x^7 + x^6 + 1$	96
8	$x^8 + x^6 + x^5 + x^4 + 1$	184

Appendix B. LFSR RTL Code

Listing A1 shows the VHDL code for the LFSR implementation of this work. Different bit-precisions can be synthesized modifying the generic parameter. The LFSR polynomials are set according to Table A2.

Listing A1. VHDL code for the LFSR implementation of this work.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY lfsr_nbits IS
generic (
BIT_PRESICION : integer:=8 -- Num. within [4,8]
);
PORT(
-- Inputs
clk, rst : IN STD_LOGIC;
i_en : IN STD_LOGIC;
i_seed: in STD_LOGIC_VECTOR(BIT_PRESICION-1 downto 0);
--Outputs
o_sequence : OUT STD_LOGIC_VECTOR(BIT_PRECISION-1 downto 0)
);
END lfsr_nbits;

ARCHITECTURE arch OF lfsr_nbits IS
SIGNAL lfsr: STD_LOGIC_VECTOR(BIT_PRECISION downto 1);
BEGIN
PROCESS (clk, rst, i_seed, i_en)
BEGIN
IF rst ='1' THEN
lfsr <= i_seed;
ELSIF (i_en='1') THEN
IF (clk'EVENT AND clk='1') THEN
-- Shift Register
for i in 1 to BIT_PRECISION-1 loop
lfsr(i+1) <= lfsr(i);
end loop;
if BIT_PRECISION = 4 THEN
lfsr(1) <= lfsr(4) XOR lfsr(3);
elsif BIT_PRECISION = 5 THEN
lfsr(1) <= lfsr(5) XOR lfsr(3);
elsif BIT_PRECISION = 6 THEN
lfsr(1) <= lfsr(6) XOR lfsr(5);
elsif BIT_PRECISION = 7 THEN
lfsr(1) <= lfsr(7) XOR lfsr(6);
elsif BIT_PRECISION = 8 THEN
lfsr(1) <= lfsr(8) XOR lfsr(6) XOR lfsr(5) XOR lfsr(4);
end if;
END IF;
END IF;
END PROCESS;
o_sequence <= lfsr;
END arch;
```

Abbreviations

The following abbreviations are used in this manuscript:

AE	Absolute Error
BS	Bit Stream
DFF	D-type Flip Flop
DL	Deep Learning
IPE	Information Processing Efficiency
LD	Low Discrepancy
LFSR	Linear Feedback Shift Register
LSZ	Least Significant Zero
LUT	Luck-UP Table
MAE	Mean Absolute Error
RNG	Random Number Generator
SC	Stochastic Computing
SNG	Stochastic Number Generator
SQF	Stochastic Quadratic Function
TC	Traditional Computing
TRNG	True Random Number Generator
WET	Wire Exchange Technique

References

1. Rosselló, J.L.; Canals, V.; Morro, A. Hardware implementation of stochastic-based Neural Networks. In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July 2010; pp. 1–4. [CrossRef]
2. Brown, B.; Card, H. Stochastic neural computation. I. Computational elements. *IEEE Trans. Comput.* **2001**, *50*, 891–905. [CrossRef]
3. Alaghi, A.; Li, C.; Hayes, J.P. Stochastic circuits for real-time image-processing applications. In Proceedings of the 50th Annual Design Automation Conference on-DAC '13, Austin, TX, USA, 29 May–7 June 2013; ACM Press: New York, NY, USA, 2013; p. 1.
4. Xie, Y.; Liao, S.; Yuan, B.; Wang, Y.; Wang, Z. Fully-parallel area-efficient deep neural network design using stochastic computing. *IEEE Trans. Circuits Syst. II Express Briefs* **2017**, *64*, 1382–1386. [CrossRef]
5. Xia, Z.; Chen, J.; Huang, Q.; Luo, J.; Hu, J. Neural Synaptic Plasticity-Inspired Computing: A High Computing Efficient Deep Convolutional Neural Network Accelerator. *IEEE Trans. Circuits Syst. Regul. Pap.* **2021**, *68*, 728–740. [CrossRef]
6. Onizawa, N.; Smithson, S.; Meyer, B.; Gross, W.; Hanyu, T. In-Hardware Training Chip Based on CMOS Invertible Logic for Machine Learning. *IEEE Trans. Circuits Syst. Regul. Pap.* **2020**, *67*, 1541–1550. [CrossRef]
7. Liu, Y.; Liu, S.; Wang, Y.; Lombardi, F.; Han, J. A stochastic computational multi-layer perceptron with backward propagation. *IEEE Trans. Comput.* **2018**, *67*, 1273–1286. [CrossRef]
8. Qian, W.; Li, X.; Riedel, M.D.; Bazargan, K.; Lilja, D.J. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.* **2010**, *60*, 93–105. [CrossRef]
9. Chen, K.C.J.; Wu, C.H. High-Accurate Stochastic Computing for Artificial Neural Network by Using Extended Stochastic Logic. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Korea, 22–28 May 2021; pp. 1–4.
10. Smithson, S.C.; Onizawa, N.; Meyer, B.H.; Gross, W.J.; Hanyu, T. Efficient CMOS Invertible Logic Using Stochastic Computing. *IEEE Trans. Circuits Syst. Regul. Pap.* **2019**, *66*, 2263–2274. [CrossRef]
11. Gaines, B.R. Stochastic computing systems. *Adv. Inf. Syst. Sci.* **1969**, *2*, 37–172.
12. Canals, V.; Morro, A.; Rosselló, J.L. Stochastic-based pattern-recognition analysis. *Pattern Recognit. Lett.* **2010**, *31*, 2353–2356. [CrossRef]
13. Zhang, Y.; Wang, R.; Zhang, X.; Zhang, Z.; Song, J.; Zhang, Z.; Wang, Y.; Huang, R. A parallel bitstream generator for stochastic computing. In Proceedings of the 2019 Silicon Nanoelectronics Workshop (SNW), Kyoto, Japan, 9–10 June 2019; pp. 1–2.
14. Alaghi, A.; Hayes, J.P. Fast and accurate computation using stochastic circuits. In Proceedings of the 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–4.
15. Schober, P.; Najafi, M.H.; Taherinejad, N. High-Accuracy Multiply-Accumulate (MAC) Technique for Unary Stochastic Computing. *IEEE Trans. Comput.* **2021**, 1. [CrossRef]
16. Liu, S.; Han, J. Energy efficient stochastic computing with Sobol sequences. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 650–653.
17. Dalal, I.L.; Stefan, D.; Harwayne-Gidansky, J. Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms. In Proceedings of the 2008 International Conference on Application-Specific Systems, Architectures and Processors, Leuven, Belgium, 2–4 July 2008; pp. 108–113.
18. Koopman, P. Maximal Length LFSR Feedback Terms. Available online: https://users.ece.cmu.edu/~{}koopman/lfsr/ (accessed on 15 January 2021).

19. Neugebauer, F.; Polian, I.; Hayes, J.P. Building a better random number generator for stochastic computing. In Proceedings of the 2017 Euromicro Conference on Digital System Design (DSD), Vienna, Austria, 30 August–1 September 2017; pp. 1–8.
20. Li, Z.; Chen, Z.; Zhang, Y.; Huang, Z.; Qian, W. Simultaneous area and latency optimization for stochastic circuits by D flip-flop insertion. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *38*, 1251–1264. [CrossRef]
21. Ichihara, H.; Ishii, S.; Sunamori, D.; Iwagaki, T.; Inoue, T. Compact and accurate stochastic circuits with shared random number sources. In Proceedings of the 2014 IEEE 32nd International Conference on Computer Design (ICCD), Seoul, Korea, 19–22 October 2014; pp. 361–366.
22. Joe, H.; Kim, Y. Novel stochastic computing for energy-efficient image processors. *Electronics* **2019**, *8*, 720. [CrossRef]
23. Anderson, J.H.; Hara-Azumi, Y.; Yamashita, S. Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy. In Proceedings of the 2016 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1550–1555.
24. Abd Elaziz, M.; Elsheikh, A.H.; Oliva, D.; Abualigah, L.; Lu, S.; Ewees, A.A. Advanced Metaheuristic Techniques for Mechanical Design Problems. *Arch. Comput. Methods Eng.* **2021**, 1–22. Available online: https://link.springer.com/article/10.1007/s11831-021-09589-4 (accessed on 15 November 2021). [CrossRef]
25. Oliva, D.; Abd Elaziz, M.; Elsheikh, A.H.; Ewees, A.A. A review on meta-heuristics methods for estimating parameters of solar cells. *J. Power Sources* **2019**, *435*, 126683. [CrossRef]
26. Ajane, A.; Furth, P.M.; Johnson, E.E.; Subramanyam, R.L. Comparison of binary and LFSR counters and efficient LFSR decoding algorithm. In Proceedings of the 2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), Seoul, Korea, 7–10 August 2011; pp. 1–4.
27. Baldi, M.; Bianchi, M.; Maturo, N.; Chiaraluce, F. A Physical Layer Secured Key Distribution Technique for IEEE 802.11g Wireless Networks. *IEEE Wirel. Commun. Lett.* **2013**, *2*, 183–186. [CrossRef]
28. Liang, W.; Long, J. A cryptographic algorithm based on Linear Feedback Shift Register. In Proceedings of the 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Taiyuan, China, 22–24 October 2010; Volume 15, pp. 15–529. [CrossRef]
29. Jone, W.B.; Rau, J.C.; Chang, S.C.; Wu, Y.L. A tree-structured LFSR synthesis scheme for pseudo-exhaustive testing of VLSI circuits. In Proceedings of the International Test Conference 1998 (IEEE Cat. No.98CH36270), Washington, DC, USA, 18–23 October 1998; pp. 322–330. [CrossRef]
30. Ting, P.S.; Hayes, J.P. Isolation-based decorrelation of stochastic circuits. In Proceedings of the 2016 IEEE 34th International Conference on Computer Design (ICCD), Phoenix, AZ, USA, 3–5 October 2016; pp. 88–95.
31. Box, G.E.; Jenkins, G.M.; Reinsel, G.C.; Ljung, G.M. *Time Series Analysis: Forecasting and Control*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
32. Heckert, N.A.; Filliben, J.J.; Croarkin, C.M.; Hembree, B.; Guthrie, W.F.; Tobias, P.; Prinz, J. Handbook 151: NIST/SEMATECH e-Handbook of Statistical Methods. 2002. Available online: https://www.nist.gov/publications/handbook-151-nistsematech-e-handbook-statistical-methods (accessed on 15 October 2021).
33. Gay, M.; Burchard, J.; Horácek, J.; Messeng Ekossono, A.S.; Schubert, T.; Becker, B.; Kreuzer, M.; Polian, I. Small scale AES toolbox: Algebraic and Propositional Formulas, Circuit-Implementations and Fault Equations. 2016. Available online: https://upcommons.upc.edu/handle/2117/99210 (accessed on 25 November 2021).
34. Guinand, C.; José, V. Implementación en Hardware de Sistemas de Alta Fiabilidad Basados en Metodologías Estocásticas. Ph.D. Thesis, Universitat de les Illes Balears, Palma de Mallorca, Spain, 2017.
35. Li, J.; Yuan, Z.; Li, Z.; Ren, A.; Ding, C.; Draper, J.; Nazarian, S.; Qiu, Q.; Yuan, B.; Wang, Y. Normalization and dropout for stochastic computing-based deep convolutional neural networks. *Integration* **2019**, *65*, 395–403. [CrossRef]
36. Metku, P.; Seva, R.; Choi, M. Energy-Performance Scalability Analysis of a Novel Quasi-Stochastic Computing Approach. *J. Low Power Electron. Appl.* **2019**, *9*, 30. [CrossRef]
37. Faraji, S.R.; Najafi, M.H.; Li, B.; Lilja, D.J.; Bazargan, K. Energy-efficient convolutional neural networks with deterministic bit-stream processing. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1757–1762.
38. Hsiao, H.; Miguel, J.S.; Hara-Azumi, Y.; Anderson, J. Zero Correlation Error: A Metric for Finite-Length Bitstream Independence in Stochastic Computing. In Proceedings of the 26th Asia and South Pacific Design Automation Conference, Tokyo, Japan, 18–21 January 2021; pp. 260–265.
39. Ting, P.; Hayes, J.P. Exploiting Randomness in Stochastic Computing. In Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 4–7 November 2019; pp. 1–6.
40. Budhwani, R.K.; Ragavan, R.; Sentieys, O. Taking advantage of correlation in stochastic computing. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.