applied sciences

IMPACT FACTOR 2.5

CITESCORE 5.5

Article

Multi-Memory Approach for Random Number Generators in FPGA

Thiago Campos Acácio Paschoalin, Tiago Motta Quirino and Luciano Manhães de Andrade Filho

Special Issue

Application of Signal Processing and Computational Intelligence in High-Energy Calorimeter Systems

Edited by

Dr. Bernardo Peralva and Dr. Gustavo Barbosa Libotte

MDPI

https://doi.org/10.3390/app16052537

applied sciences

MDPI

Article

Multi-Memory Approach for Random Number Generators in FPGA

Thiago Campos Acácio Paschoalin [1,*], Tiago Motta Quirino [2,*] and Luciano Manhães de Andrade Filho [3,*]

[1] Electroelectronic Department, Federal Center for Technological Education of Minas Gerais, Leopoldina Campus, Leopoldina 36700-000, MG, Brazil
[2] Electrical Engineering Department, Engineering Faculty, Rio de Janeiro State University, Rio de Janeiro 20550-900, RJ, Brazil
[3] Electrical Circuit Department, Federal University of Juiz de Fora, Juiz de Fora 36036-900, MG, Brazil
* Correspondence: thiago.paschoalin@cefetmg.br (T.C.A.P.); tiago.quirino@eng.uerj.br (T.M.Q.); luciano.andrade@ufjf.br (L.M.d.A.F.)

Abstract

Random number generation is essential in many application domains, including high-energy physics simulations. Implementing Monte Carlo methods that generate samples following a desired probability distribution is particularly challenging on hardware platforms such as FPGAs. Direct implementations of analytical distribution functions are often resource-intensive, making them impractical for real-time systems. An efficient alternative is the use of the inverse cumulative distribution function (CDF), which can be implemented using look-up tables (LUTs). In this approach, a uniformly distributed random number—generated by Linear Feedback Shift Registers (LFSRs)—is used as an address to access LUTs containing discretized x-axis values of the CDF, thereby yielding the target random variable. However, this method presents limited accuracy in low-probability regions of the distribution. To address this issue, this paper proposes a segmented CDF implementation based on multiple LUTs, improving resolution in poorly sampled regions. A cascade of decision logic selects the appropriate memory output, increasing resolution only where necessary while optimizing memory usage. The proposed method was validated through Monte Carlo simulations in particle physics applications, achieving close agreement with theoretical distributions while requiring limited FPGA resources and no DSP blocks.

Keywords: Monte Carlo simulation; random number; FPGA implementation; online simulation; high-energy calorimetry

Academic Editor: Alexander Barkalov

Received: 18 December 2025
Revised: 2 March 2026
Accepted: 4 March 2026
Published: 6 March 2026

1. Introduction

The generation of high-quality random numbers plays a crucial role in a wide range of computational and scientific applications, including simulations, digital signal processing, statistical analysis, genetic algorithms, machine learning, financial modelling, cryptography, and high-energy physics [1–5]. Beyond these traditional uses, hardware-based random number generators can also be employed to emulate detector signals with nanosecond-scale pulses, supporting the evaluation of particle-detector readout systems [6].

Random numbers can be produced either by True Random Number Generators (TRNGs), which yield fully unpredictable sequences, or by Pseudo-Random Number Generators (PRNGs), which are deterministic but designed to mimic randomness based on an initial seed [7]. Due to their efficiency and suitability for parallel architectures,

PRNGs are widely adopted in FPGA implementations, with Linear Feedback Shift Registers (LFSRs) standing out for their simplicity, speed, and low hardware cost [8,9]. Several techniques, such as the use of primitive polynomials, polynomial modulators, and alternative entropy sources, have been explored to enhance the statistical properties of LFSR-based generators [10–13].

An uncorrelated PRNG was presented in [14]. The technique uses parallel LFSR circuits to generate uniformly distributed random numbers with no correlation between them. It is a lightweight design implemented in FPGA technology, requiring few hardware resources while supporting high clock frequencies.

However, many practical applications require random numbers following non-uniform probability distributions, particularly the Gaussian distribution, which is essential in Monte Carlo simulations, noise modeling, and numerous stochastic processes. Multiple hardware-oriented approaches have been proposed to generate non-uniform random variables, including transformation-based methods (e.g., Box–Muller), rejection-based techniques (e.g., Ziggurat), and inversion-based implementations using the inverse cumulative distribution function (ICDF) [15,16]. Although effective, these approaches often face a trade-off between accuracy, computational complexity, and FPGA resource consumption. Recent works have addressed these challenges by employing floating-point arithmetic, hierarchical segmentation, or optimized polynomial approximations to reduce hardware overhead while maintaining statistical fidelity [17–19]. Additionally, statistical validation frameworks such as the Dieharder test suite have been employed to ensure the reliability of hardware-generated sequences [20].

In high-energy particle experiments, Monte Carlo simulation plays a fundamental role at multiple stages of detector design and data processing. Tasks such as selecting the electronics for read-out channels, evaluating system performance, and developing new energy-reconstruction techniques all benefit from realistic simulations of physical processes. Since particle colliders operate at extremely high collision rates (40 MHz in the case of the Large Hadron Collider) the associated detector systems must satisfy stringent timing constraints [21–26]. A real-time simulator capable of reproducing the energy deposition of particle interactions can therefore be highly valuable, enabling the study of new filtering strategies, improvements in energy-reconstruction algorithms, and the exploration of other related processes.

In this context, a challenge arises: generating values that accurately follow a target distribution while respecting strict memory constraints and running at FPGA. Addressing these issue, this work combines efficient uniform PRNG generation with a memory-optimized ICDF-based (Inverse Cumulative Distribution Function) mapping technique. First, a decorrelated uniform PRNG is implemented using seven parallel LFSRs and a selector that cyclically shifts their outputs, producing sequences with reduced correlation by relying solely on XOR logic and a multiplexer structure [14]. Second, a non-uniform RNG architecture is proposed in which multiple memory blocks store different regions of the cumulative distribution function (CDF). This segmentation preserves resolution in critical regions of the distribution without requiring excessively large look-up tables (LUTs). The proposed approach was evaluated in a particle physics environment using Monte Carlo simulations.

Table 1 presents a qualitative comparison between representative state-of-the-art techniques and the proposed method. In contrast to transformation- and rejection-based approaches, which often involve complex control logic and iterative processing steps, the method proposed in this work is designed around a simplified architecture based on deterministic memory access and lightweight combinational logic. This architectural choice targets reduced implementation complexity on FPGA platforms and avoids reliance on

dedicated DSP resources, which are commonly constrained in high-throughput systems. Furthermore, by partitioning the ICDF into multiple memory regions, the proposed approach seeks to balance memory requirements with distribution resolution, addressing limitations typically associated with single large LUT implementations.

Table 1. Qualitative comparison between hardware-oriented non-uniform RNG methods and the proposed approach.

Method	FPGA Complexity	DSP Usage	Memory Demand	High-Frequency Suitability
Box–Muller (HW implementations)	High	High	Low	Limited
Ziggurat-based methods	Medium to high	Medium	Medium	Moderate
Polynomial/approximation-based ICDF	Medium	Low to medium	Low	High
Single-LUT ICDF mapping	Low	None	Very high	High
Proposed segmented ICDF (this work)	**Low**	**None**	**Moderate**	**Very high**

The remainder of this paper is organized as follows. Section 2 presents the theoretical background of Linear Feedback Shift Registers (LFSRs), while Section 3 discusses Monte Carlo-based distribution generation, with emphasis on the challenges imposed by limited LUT resources. Section 4 describes the proposed FPGA architecture, detailing both the decorrelated pseudo-random number generator and the multi-memory inverse CDF mapping scheme. Section 5 introduces particle colliders as the reference application used throughout this work. Section 6 presents two implementation examples of the proposed approach, and finally, Section 7 summarizes the conclusions and outlines future research directions.

2. LFSR Pseudo-Random Number Generator

Linear Feedback Shift Registers (LFSRs) are compact hardware structures composed of a shift register and XOR-based feedback logic. At each clock cycle, the stored bits are shifted and a new bit is generated from a linear combination of selected register positions, known as taps, defined by the feedback polynomial. For an n-bit LFSR, the feedback bit is computed according to

$$b_{\text{new}} = b_{k_1} \oplus b_{k_2} \oplus \cdots \oplus b_{k_m} \tag{1}$$

where b_k are the tapped bits. Depending on the design, the new bit may enter the register through either the least significant bit (LSB) or the most significant bit (MSB). When the feedback polynomial is primitive, the LFSR cycles through all $2^n - 1$ non-zero states, producing a maximum-length sequence with good statistical properties, such as balanced bit distributions and low autocorrelation [27].

The Figure 1 shows an example of a 5-bit LFSR circuit implemented with shift registers, where the feedback bit is defined by a primitive polynomial applied at the XOR gate. The initial register values, known as the seed, determine the sequence of generated pseudo-random numbers. When the same seed is used, the sequence of samples remains identical across executions.

Uncorrelated Pseudo-Random Number Generator

Studies indicate that although LFSR-based pseudo-random number generators exhibit generally low correlation, they may still present noticeable correlation between temporally adjacent samples [14]. This behavior may be undesirable depending on the application,

such as high-energy physics simulations [21–23,26]. The objective, therefore, is to maintain the advantages of LFSR circuits while mitigating the initial correlation among samples.

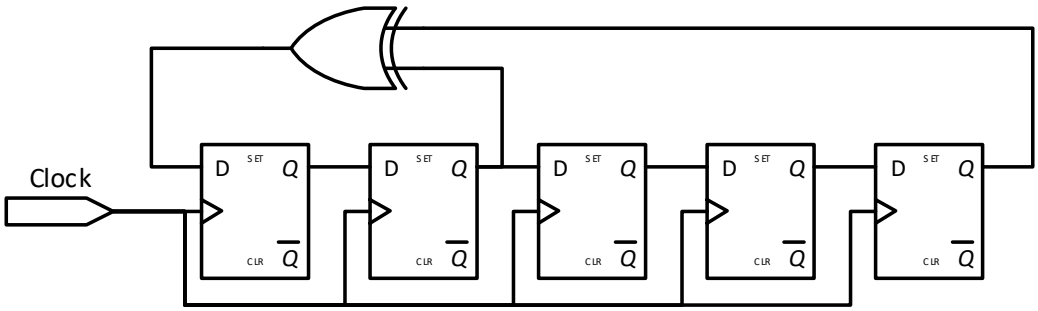


Figure 1. 5-bit LFSR circuit.

A common strategy to reduce the undesired correlation is to employ multiple LFSR circuits operating in parallel, with different seeds, combined with a switching mechanism that selects which circuit provides the generator output at each clock cycle [14]. The required number of parallel LFSRs depends on the extent of the initial correlated region; for example, if the first six samples exhibit unacceptable correlation, seven generators may be used to overcome this limitation.

The Figure 2 illustrates this architecture. The counter controlling the multiplexer has its modulo defined by the number of parallel LFSR circuits. An essential requirement is that each generator must be initialized with a distinct seed. This architecture is simple to implement, requires few hardware resources, supports high clock frequencies, and produces sequences of statistically uncorrelated random numbers.

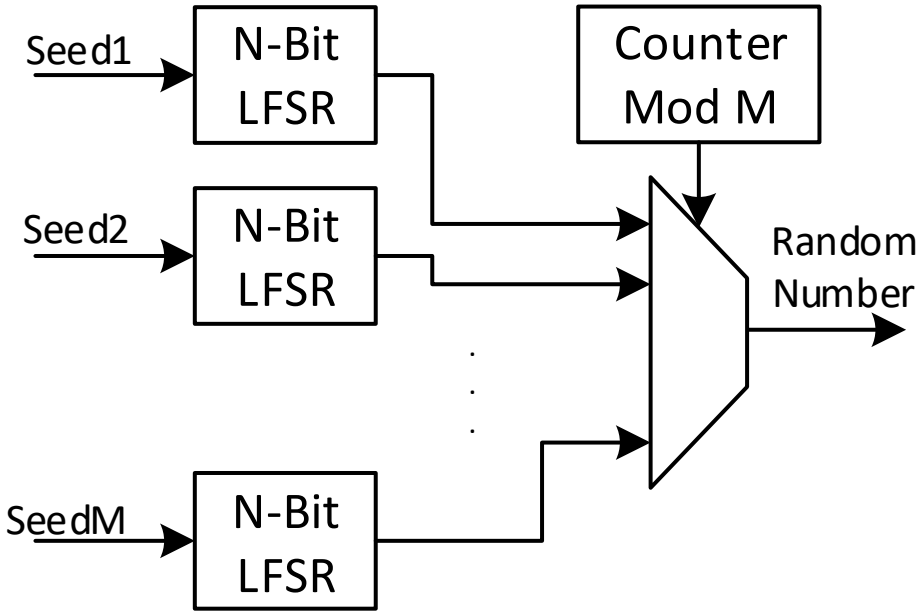
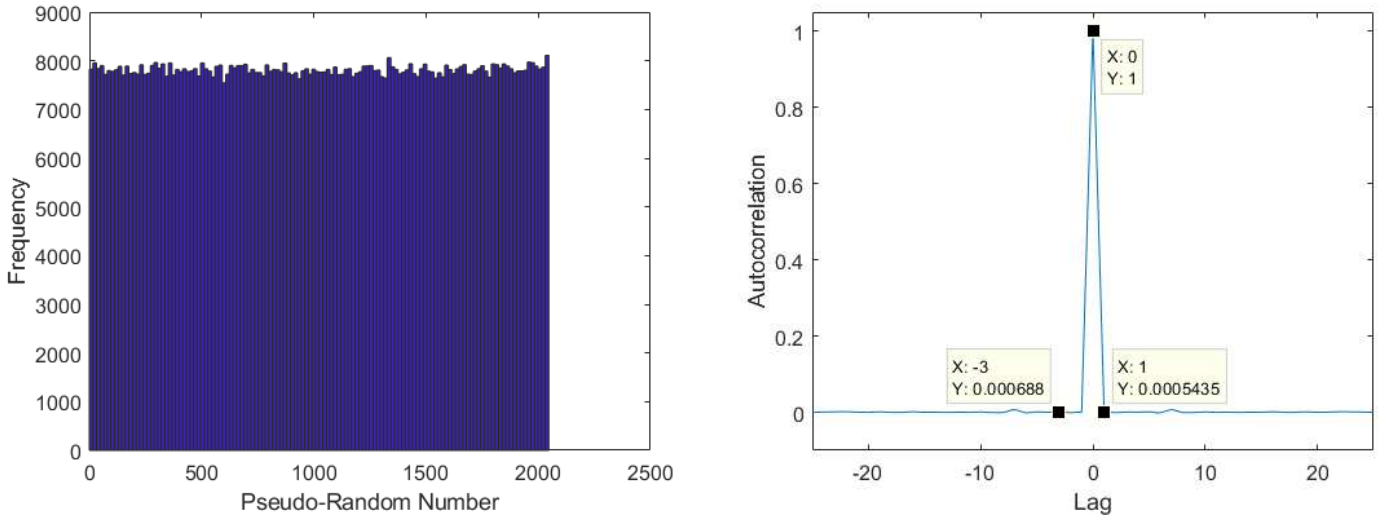


Figure 2. Block diagram for the uniformly distributed random number generator.

To highlight the statistical characteristics of the generated dataset, Figure 3a and Figure 3b present, respectively, the histogram and the autocorrelation of the generated sequence. The histogram confirms that the distribution closely follows a uniform profile, while the autocorrelation plot (particularly in the zoomed region) shows no significant correlation among the first samples.



(**a**) (**b**)

Figure 3. (**a**) Histogram of the PRNG. (**b**) Autocorrelation of the sequence produced by PRNG.

3. Monte Carlo Simulation

The study of physical processes often relies on statistical analysis. Researchers typically collect real-world data and investigate its underlying behavior. In this context, the statistical properties of a process are crucial for predicting environmental conditions and, when necessary, for developing tools that mitigate undesirable effects that could degrade system performance.

A widely adopted approach for characterizing such processes is the Probability Density Function (PDF). The PDF describes how probability is distributed over the possible values of a random variable. The PDF assigns a probability density to each value of the random variable, thereby providing a complete statistical description of its behavior.

Monte Carlo simulation is a technique that relies on randomness to generate samples that mimic the statistical behavior of a target variable, and several approaches can be used to accomplish this. One particularly convenient and easily implementable method is based on the inverse of the CDF [28]. In this approach, a uniformly distributed random variable is used as the input for the simulation process. Equation (2) defines such a random variable following a uniform distribution.

$$U \sim \mathcal{U}(0,1) \tag{2}$$

Let X be the random variable to be simulated and let $F_X(x)$ denote its CDF. Once the distribution of X is defined, the next step is to compute the ICDF and use the uniformly distributed variable U as its input. By generating a sequence of values for U and applying the ICDF, one obtains samples of a random variable that follow the same distribution as X, as expressed in Equation (3).

$$X = F_X^{-1}(U) \tag{3}$$

3.1. ICDF Reference Table

Knowing the analytical form of a PDF or CDF often enables direct computation of the corresponding ICDF. In such cases, the resulting function is continuous, allowing a uniformly distributed random number to be directly mapped to the desired output value. However, in many practical scenarios, the target random variable does not fit well to a single known distribution, thus requiring alternative methods.

Given a real dataset of the target variable, one may construct its histogram, which (depending on the number of bins and the normalization) approximates the underlying PDF. A histogram estimates the PDF by partitioning the data range into discrete intervals and counting the number of samples that fall within each bin. Let h_i denote the number of samples in the i-th bin and N the total number of samples. The normalized histogram, obtained by dividing each h_i by N, provides an empirical approximation of the probability mass associated with each bin.

The empirical CDF is then obtained by accumulating these normalized frequencies [29]. Specifically, for a given bin k, the value of the empirical CDF is computed as shown in Equation (4), where x_k represents the upper boundary of the k-th bin. This cumulative sum corresponds to the proportion of observations that do not exceed x_k. Repeating this accumulation for all bins yields the complete empirical CDF.

$$F_{\text{emp}}(x_k) = \sum_{i=0}^{k} \frac{h_i}{N} \tag{4}$$

The ICDF for such cases can be derived by discretizing the empirical CDF. The y-axis is divided into equally spaced points, and the corresponding x-axis values are stored. A practical implementation of the ICDF consists of assigning integer labels to these points, from zero to the total number of points minus one, and associating each label with its respective stored x-value [30]. This structure can be interpreted as a reference table, or, from a hardware perspective, a memory block containing the desired output values.

Although this technique is particularly useful for distributions that lack a closed-form expression, known distributions may also be used to generate such reference tables when hardware implementation is desired. Figure 4 presents an example of an exponential CDF with the selected discretization points. In this illustration, the y-axis is divided into ten equally spaced intervals.

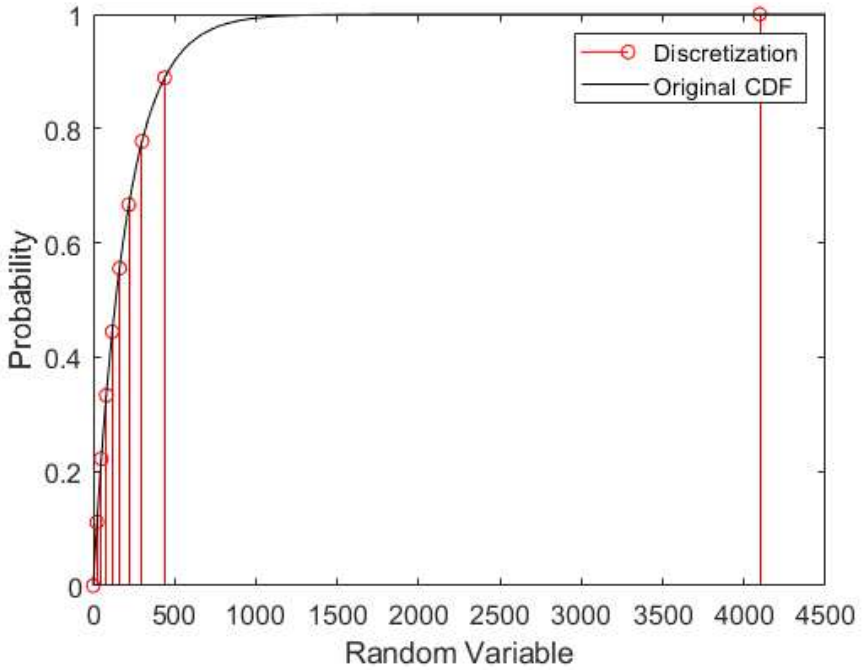


Figure 4. CDF and selected points for the reference table.

Monte Carlo simulation using the ICDF can be adapted with a small modification to the uniformly distributed input variable. Instead of producing a random number between zero and one, the generator outputs an integer between zero and the number of discretization

points minus one. This integer directly indexes the reference table—i.e., the corresponding memory position—to retrieve the desired random sample.

3.2. Approximation Problem

The described technique introduces an approximation due to the discretization of the CDF. If the CDF contains one or more extended flat regions, these intervals will not be accurately represented. As illustrated in Figure 4, any random value between 480 and 4096 will never occur, which may be unsuitable depending on the target application. Figure 5 shows a histogram of a generated exponential random variable, plotted with a logarithmic scale on the y-axis, where this issue becomes more evident, even if a 1024 table positions were used.

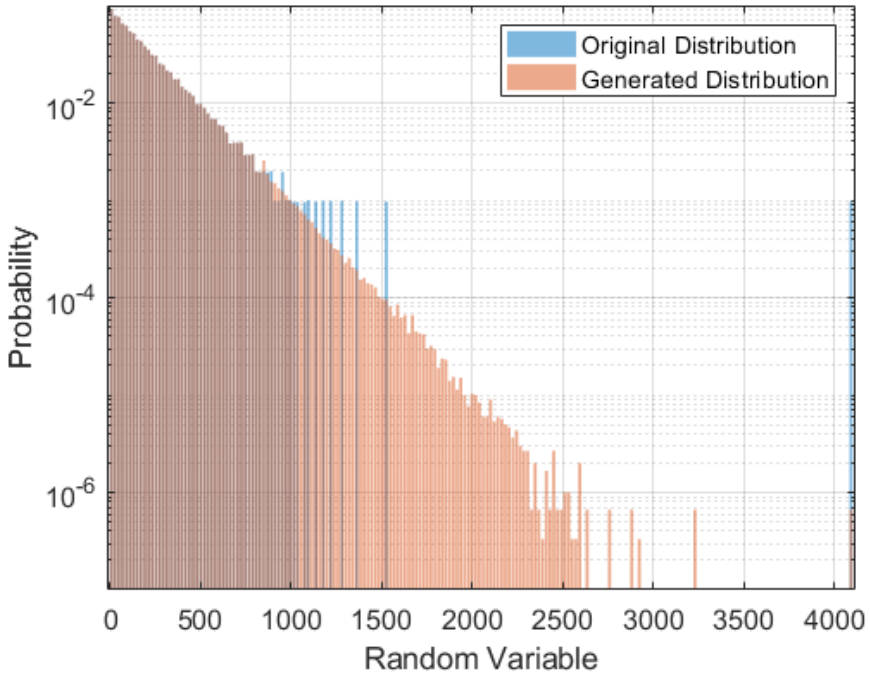


Figure 5. Histogram of generated Random Number with 1024 positions.

One straightforward way to mitigate this problem is to increase the number of sampled points in the discretized CDF, thereby improving the resolution along the x-axis. However, depending on the distribution, the number of required points can become very large. In simulation environments where hardware resources are abundant, this is not a limiting factor. In contrast, for embedded systems implemented on resource-constrained hardware such as FPGAs, memory usage becomes a critical concern. In some cases, achieving sufficient resolution may require on the order of 10 MB of memory—an amount typically unavailable in many FPGA devices.

Increasing the number of samples to capture flat regions necessarily causes oversampling in the remaining portions of the CDF, where such fine resolution is unnecessary. This oversampling leads directly to excessive memory usage. To improve efficiency, the objective is to allocate additional memory only to represent the flat regions of the distribution, while avoiding redundancy in the well-behaved segments. This optimization strategy is discussed in the next section.

4. Multi-Memory Approach

Analyzing Figure 5, it becomes clear that the gap observed in the histogram must be filled to properly match the original distribution. The most effective strategy is to increase

the resolution only within the flat region of the CDF, thereby conserving memory resources. For instance, in the sampled CDF of Figure 4, if the generated uniformly distributed random value lies between 0.8912 and 1, then the corresponding output random variable may assume a wide range of values along the x-axis. In such a case, an algorithm must determine whether the sampled point falls within this interval, and, if so, consult an additional reference table (stored in a different memory segment) to refine the output value.

In some situations, the flat region of the CDF may contain additional subregions that are also flat but were not visible at the original scale. In these cases, more than one additional memory layer may be required to accurately represent the random variable. The algorithm must therefore include an additional decision step to check whether the refined memory index still lies within a secondary flat region, and, if necessary, perform another lookup in a deeper memory layer. Figure 6 presents a flowchart summarizing the decision-making process performed by the algorithm.

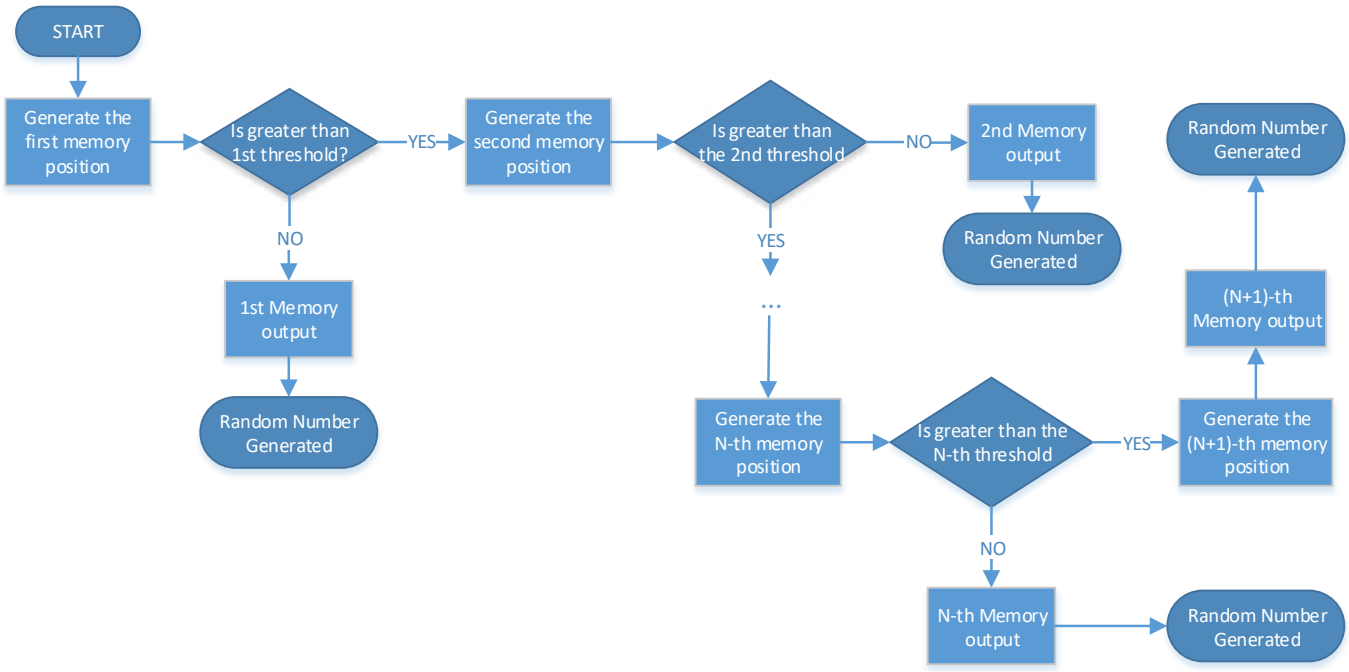


Figure 6. Flowchart for the algorithm decision.

To further clarify the procedure, consider again the example of the exponential distribution. As discussed previously, the flat region of the CDF is poorly sampled. Suppose the inverse CDF is stored in a memory with 1024 positions. In this configuration, the region between 854 and 1023 may be improved by allocating a second memory, also with 1024 positions, dedicated exclusively to this segment. If the histogram of the generated values is plotted again, another region may still appear undersampled. In that case, a third memory can be added, covering the interval between 1709 and 1024 of the second memory. Sampling this final portion of the CDF with 1024 points significantly improves the histogram quality compared to the single-memory implementation. Figure 7 illustrates the coverage range of each memory layer.

In this example, a total of 3072 memory positions are used across the three layers. Without memory optimization, achieving equivalent accuracy would require sampling the CDF with approximately 770,000 points, which would exceed the memory capacity of many FPGA devices. It is important to note that the three memory blocks do not necessarily need to have the same size; each can be dimensioned according to the resolution required for the specific region it covers.

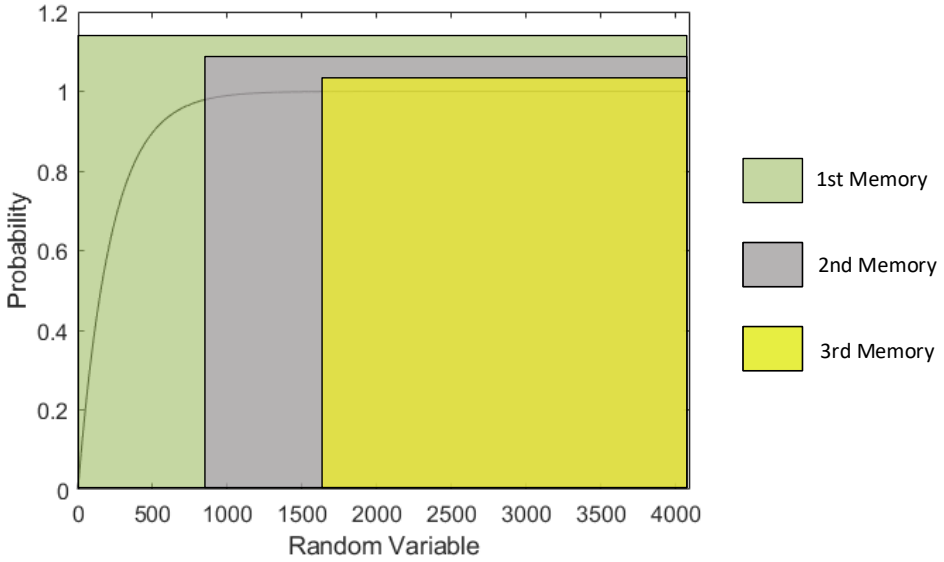


Figure 7. Memory regions for the CDF.

4.1. Defining the Range for Each Memory

The performance of the proposed algorithm depends strongly on how the boundaries of each memory layer are determined. Although this selection can be performed manually by inspecting the generated histograms and adjusting the ranges accordingly, such an approach becomes impractical when multiple random variables must be generated or when several memory layers are required.

The core difficulty in choosing these ranges lies in evaluating the distance between consecutive x-axis values of the discretized CDF. Large differences between adjacent values indicate the presence of a flat region within that interval. Figure 8 presents the plot of the differences between consecutive values stored in the first reference table of the previous example. A noticeable knee appears between 700 and 1000, after which the differences tend to grow, highlighting the beginning of the flat region.

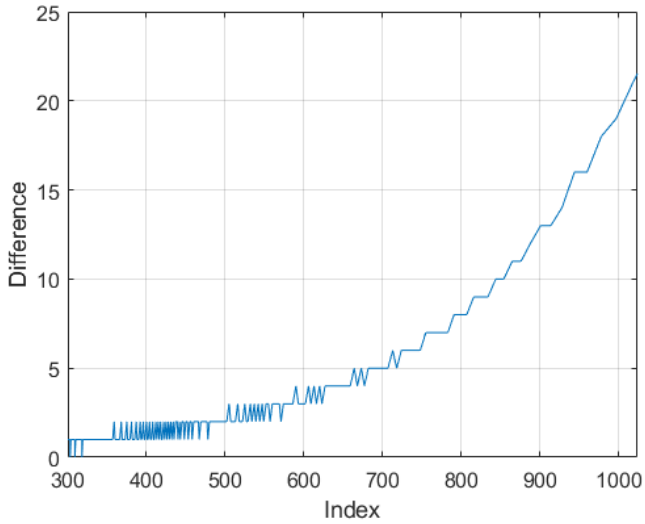


Figure 8. Plot of difference between consecutive values of the memory positions.

To automate the selection of these ranges, an acceptable threshold can be defined for the relative increase between consecutive samples (e.g., 10). Whenever the computed increase exceeds this predefined threshold, the algorithm assigns the corresponding index to a new memory layer, and the associated memory position is stored as the boundary of

the range. A more objective criterion can be obtained by computing the relative difference with respect to the maximum stored difference. In practice, relative difference thresholds of 1% or 2% were found to provide good results.

4.2. FPGA Implementation

The flowchart presented earlier describes the sequence of operations required by the algorithm. While this representation is suitable for software implementations, it is not practical for FPGA-based architectures. In FPGA technology, circuits operate in parallel and typically under synchronous clock control, which requires a fundamentally different approach. In particular, the circuit cannot wait for the output of the first memory to decide whether it should access the next memory layer.

To guarantee a fixed-latency random number generation, all memory addresses for every memory layer must be produced simultaneously. Given these input positions, all memories will compute their corresponding candidate output values in parallel. However, the final selected output must respect the priority rules imposed by the predefined boundaries that determine when each memory layer should be activated.

In the FPGA implementation, the decision chain begins from the last two memories to preserve this priority. First, the position associated with the second-to-last memory is evaluated: if it falls within the specified flat region, the temporary output is taken from the final memory; otherwise, the value from the second-to-last memory is selected. Next, the same process is applied to the third-to-last memory, where the algorithm chooses between the temporary output from the previous step and the value obtained from the current memory layer, depending on the established range. This decision chain continues until the first memory is reached.

In summary, the architecture consists of a cascade of multiplexers whose selection signals are defined by the range limits of each memory layer. The complete circuit can be visualized in Figure 9, which presents the block diagram of the described implementation. This approach results in a simple and efficient circuit, requiring no DSP blocks or complex structures.

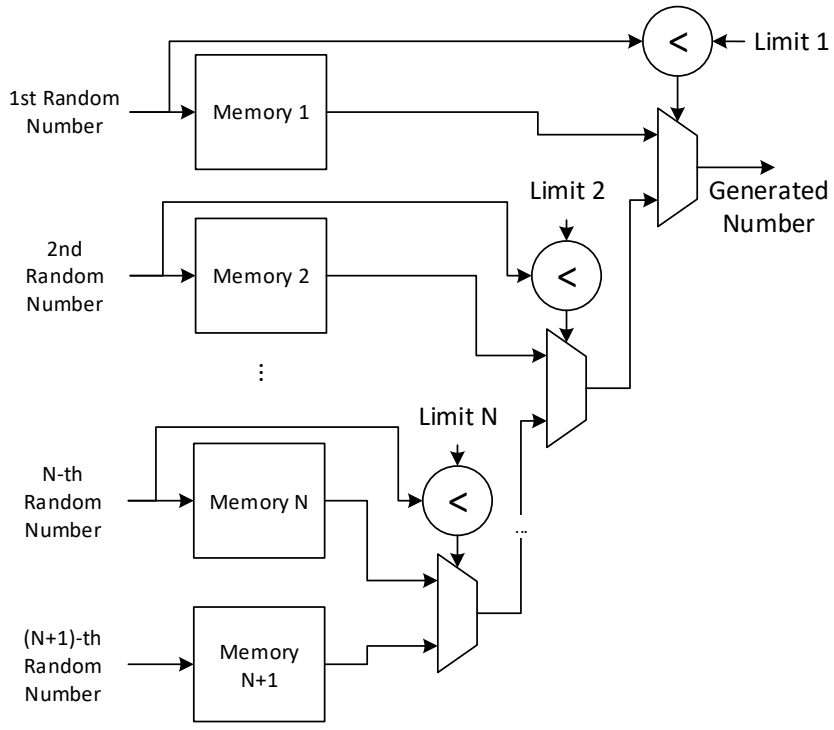


Figure 9. FPGA implementation for random number generation.

5. Case Study

This section introduces a high-energy physics scenario in which the proposed method is evaluated. Since the goal is to generate random numbers following a desired distribution for real-world applications, this domain provides a suitable test environment for analyzing the performance of the proposed implementations.

In particle physics, scientists investigate the fundamental constituents of matter, aiming to understand the formation of the universe, its evolution, and the interactions between elementary particles. High-energy physics experiments provide the means to test and validate theoretical predictions, such as those described by the Standard Model.

To achieve this, particle colliders are widely employed. These machines accelerate particles to velocities close to the speed of light and force them to collide at predefined interaction points. The energy released in these collisions produces a variety of secondary particles, which propagate outward from the interaction region. Detectors are strategically positioned around this point to measure the energy deposits and reconstruct the resulting particles and their trajectories [31].

Each detector subsystem uses dedicated electronics tailored to the type of particle being measured, its expected energy, and the distance from the collision point. Calorimeters, for instance, detect particles through different interaction mechanisms, and in some technologies, this process involves the production of scintillation light whose intensity is proportional to the deposited energy [32]. Modern calorimeters frequently rely on Silicon Photomultipliers (SiPMs) to convert this light into electrical pulses suitable for processing [6].

The generated signal is extremely short in duration and is commonly modeled as an impulse whose amplitude reflects the particle's deposited energy. A shaping stage is typically applied to extend the signal in time, improving the resolution and enabling accurate sampling using an analog-to-digital converter. Once digitized, reconstruction algorithms can be applied to identify events of interest. Figure 10 provides an illustration of this process.

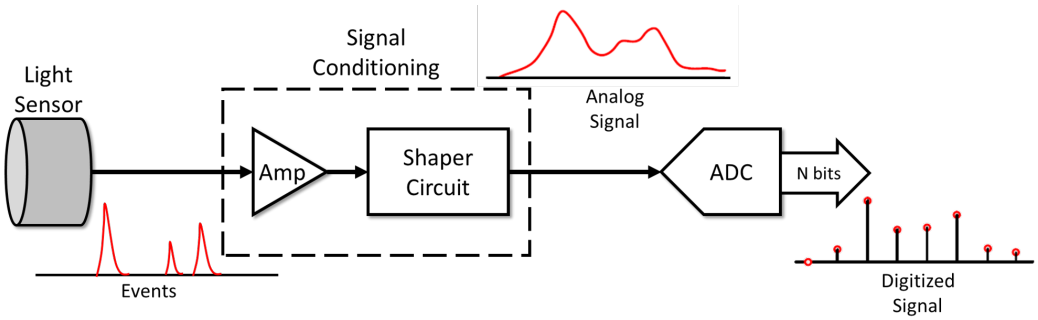


Figure 10. Block diagram for signal generation.

Target Distributions for Evaluation

The energy deposited by particle interactions is the fundamental information that detectors aim to measure. This deposited energy can be simulated using Monte Carlo techniques based on theoretical models. During the design and commissioning of particle accelerators, such simulations were essential for ensuring proper system operation.

Real data collected by detectors can also be employed to generate more realistic simulation scenarios. The observed samples may be fitted to a known probability distribution, or alternatively, the empirical histogram itself can be used as the basis for constructing a simulation model.

In this context, several studies [21–26] have investigated the statistical characteristics of particle detector responses. The geometric arrangement of detector cells significantly influences the resulting energy distribution profiles observed in the measurements. Since particle detectors typically consist of a very large number of cells, mapping the distribution of energy deposition for each cell provides valuable information for the development of new particle identification techniques.

According to various analyses [21–26], calorimeters such as the Tile Calorimeter (TileCal) exhibit energy distributions that approximate an exponential form. The parameters of the distribution—such as the mean and standard deviation—vary depending on the experimental conditions (e.g., luminosity) and the spatial location of the cell within the detector.

Once the energy distribution profile of a given cell has been characterized, a simulator can be used to generate signals that follow the same statistical behavior. A real-time simulator operating at the collision frequency can support the development of advanced energy reconstruction algorithms and trigger systems, which are essential components of high-energy physics experiments.

The next section presents two approaches for generating random numbers suitable for modeling the desired energy distributions in FPGA hardware. Implementations based on exponential distributions are described, together with analyses of maximum operating frequency, statistical properties of the generated sequences, and FPGA resource utilization.

6. Results

The two proposed implementations (the uncorrelated LFSR-based RNG and the multi-memory RNG) were previously developed and evaluated and can be combined into a single architecture. The uncorrelated pseudo-random number generator supplies the memory addresses that drive the sampling process in the LUT-based generator. All implementations target FPGA devices, and the DE10-Nano-SoC board was selected as the evaluation platform for benchmarking [33]. Figure 11 shows a schematic representation of the complete FPGA architecture used for random number generation.

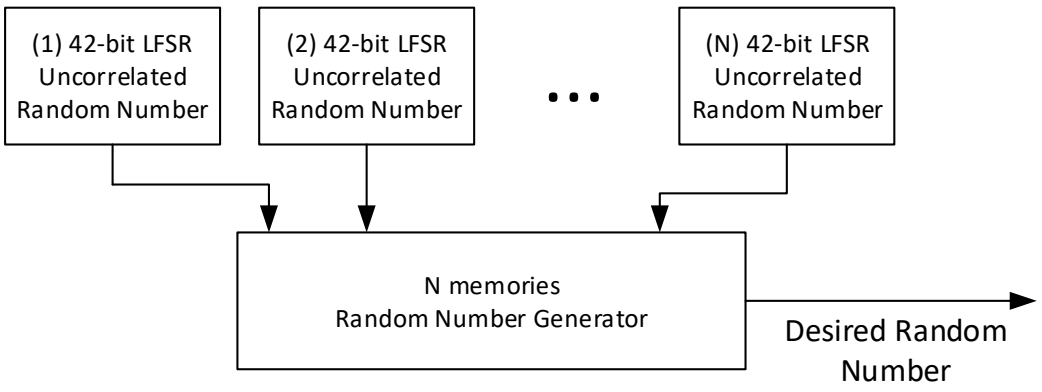


Figure 11. Block diagram of the complete random number generation architecture.

The evaluation uses representative probability distributions, since the number of memories, their depth, and bit width vary according to the specific distribution targeted by each application. Histograms are provided as a qualitative visualization of the generated samples, while quantitative statistical metrics are reported to assess distributional accuracy and randomness properties. While FPGA resource usage is reported to assess feasibility and scalability.

To apply the proposed method, the target CDF is first generated and the number of memory locations is defined. The memory is populated by sampling the ICDF, while uniformly distributed random numbers are used to generate the output sequence. The

resulting histogram is then compared with the target distribution. If the fit is not satisfactory, an additional memory is introduced.

A threshold based on the relative difference is defined, and the new memory is filled using the subdivided region of the distribution. By employing two independent uniformly distributed random numbers together with the threshold, the two memories generate a new histogram, which is again compared with the target distribution. If the fit remains inadequate, another memory is incorporated and the process is repeated. This iterative procedure continues until the accuracy requirements defined by the expert are met. Figure 12 presents the workflow to implement the proposed method.

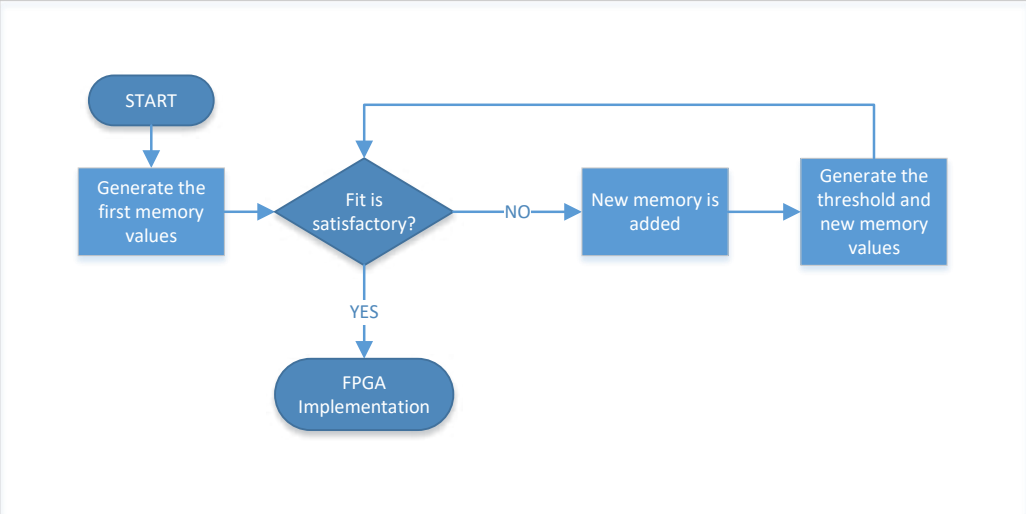


Figure 12. Workflow for apply the method.

Once the number of memories and their corresponding entries are defined, the architecture can be implemented on FPGA by instantiating a number of uncorrelated LFSRs equal to the number of memories and storing the corresponding ICDF values. At this point, the hardware is ready to generate the desired random sequences.

6.1. Uniform PRNG Validation for Memory Addressing

The proposed multi-memory random number generation architecture relies on uniformly distributed pseudo-random numbers to address the LUT that implement the ICDF. As a result, the statistical quality of the uniform PRNG directly impacts all non-uniform distributions generated by the system, independently of the target PDF.

For this reason, a dedicated statistical validation of the uniform PRNG is first presented, prior to the analysis of the non-uniform random number generators. The validation focuses on three fundamental properties required for hardware-oriented PRNGs: uniformity, independence, and absence of structural or periodic patterns.

Since all address streams are generated by the same uncorrelated LFSR-based architecture, statistical tests were applied to a representative output stream. Independence between parallel streams, which are used simultaneously to address different memories, was evaluated separately through cross-correlation analysis.

Figure 13 shows the histogram of the generated uniform memory address values, confirming the expected uniform coverage of the available address space.

Uniformity of the generated address values was evaluated using the Kolmogorov–Smirnov (KS) test, which compares the empirical cumulative distribution function with the theoretical uniform distribution over the interval $[0, 1)$. For the 11-bit address space adopted in the FPGA implementation, the KS statistic indicates good agreement with the expected uniform behavior, with no statistically significant deviation observed.

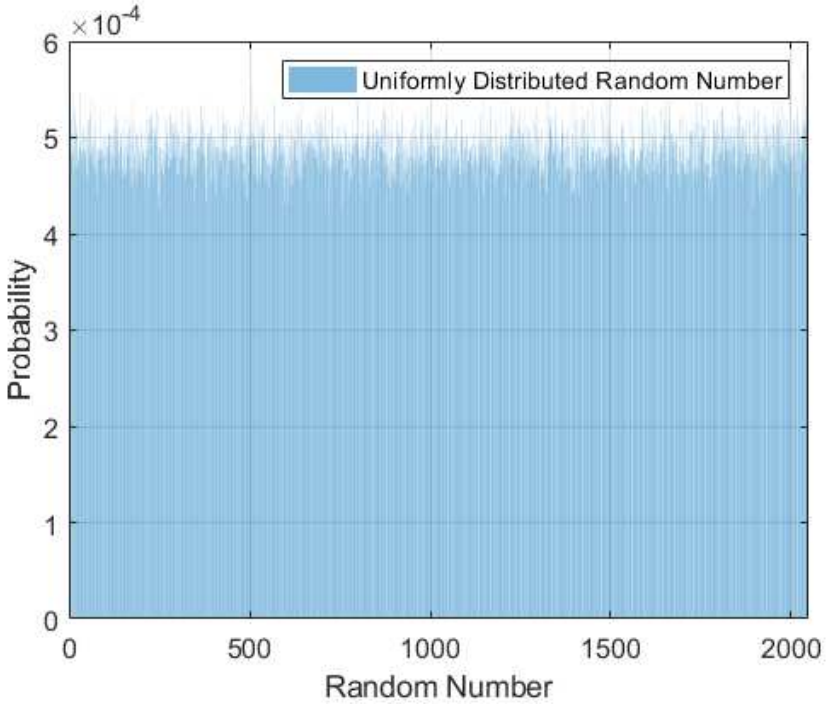


Figure 13. Histogram of uniform PRNG outputs used for memory addressing.

Independence between consecutive samples was assessed using a runs test, adopting the statistical formulation described in the NIST SP 800–22 recommendations. The measured runs statistics are consistent with those expected from an independent random sequence, and the normalized autocorrelation (AC) coefficients remain close to zero for all evaluated lags, indicating the absence of linear temporal dependence.

Spectral randomness tests based on bit-level frequency-domain analysis were not considered, as the PRNG outputs correspond to fixed-width memory addresses rather than balanced binary sequences. In this context, autocorrelation and cross-correlation metrics provide a more direct and interpretable assessment of independence for the intended hardware application.

The results of the statistical validation of the uniform PRNG are summarized in Table 2. All uniform address streams are generated by the same PRNG and therefore exhibit identical statistical properties prior to any thresholding or conditional selection.

Table 2. Statistical validation of the uniform PRNG used for memory addressing.

KS Statistic	KS p-Value	Runs p-Value	Max. |AC| (Lags 1–100)
0.00114	0.147	0.763	$< 1.5 \times 10^{-3}$

The uniform PRNG operates with an 11-bit address space, matching the LUT depth used in the FPGA implementation. When evaluated prior to any thresholding or conditional selection, the generated address stream exhibits good uniformity and independence, as confirmed by the Kolmogorov–Smirnov and runs tests, as well as low autocorrelation values.

The Kolmogorov–Smirnov statistic indicates good agreement with the theoretical uniform distribution, with no statistically significant deviation observed. The runs test yields p-values consistent with those expected from an independent sequence, indicating the absence of temporal clustering or systematic transitions. In addition, the normalized autocorrelation coefficients remain close to zero for all evaluated lags, further confirming the lack of linear dependence between consecutive samples.

When multiple LUTs are used, several uniform streams are generated in parallel to address different memory segments. In the proposed architecture, the maximum level of parallelism is four address streams (corresponding to the Gaussian configuration), therefore assessing the mutual dependence among four simultaneous streams is sufficient to cover the worst-case operating condition.

To ensure that this parallel usage does not introduce systematic correlation, pairwise cross-correlation coefficients were computed between all address streams used simultaneously, forming a 4×4 correlation matrix, as shown in Figure 14. Low off-diagonal correlation values confirm that the streams are effectively uncorrelated and can be safely employed together in the multi-memory architecture without degrading the statistical quality of the generated non-uniform random variables.

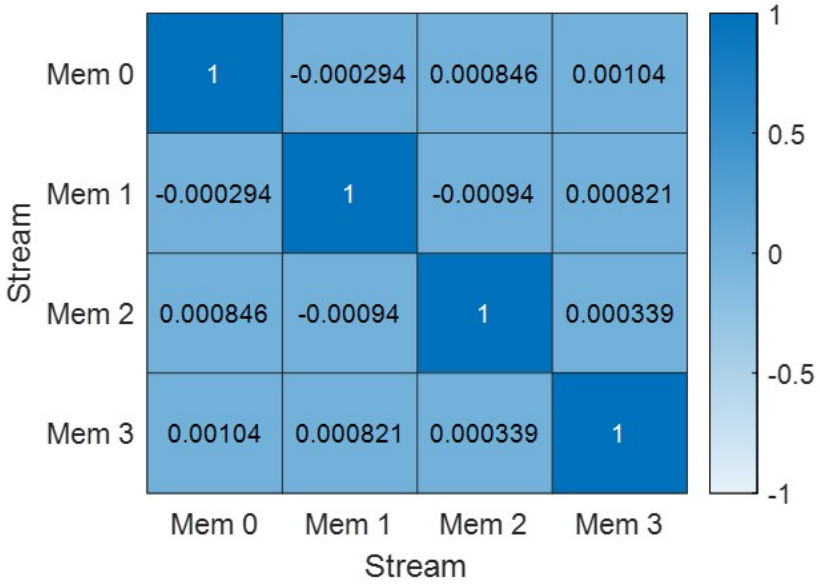


Figure 14. Heatmap of the 4×4 cross-correlation matrix between the maximum number of parallel uniform address streams used in the proposed architecture. Off-diagonal values close to zero indicate negligible cross-correlation between streams.

The maximum off-diagonal coefficient was $\rho_{\max} = 1.03 \times 10^{-3}$, confirming that the streams are effectively uncorrelated and can be safely employed together in the multi-memory architecture without degrading the statistical quality of the generated non-uniform random variables.

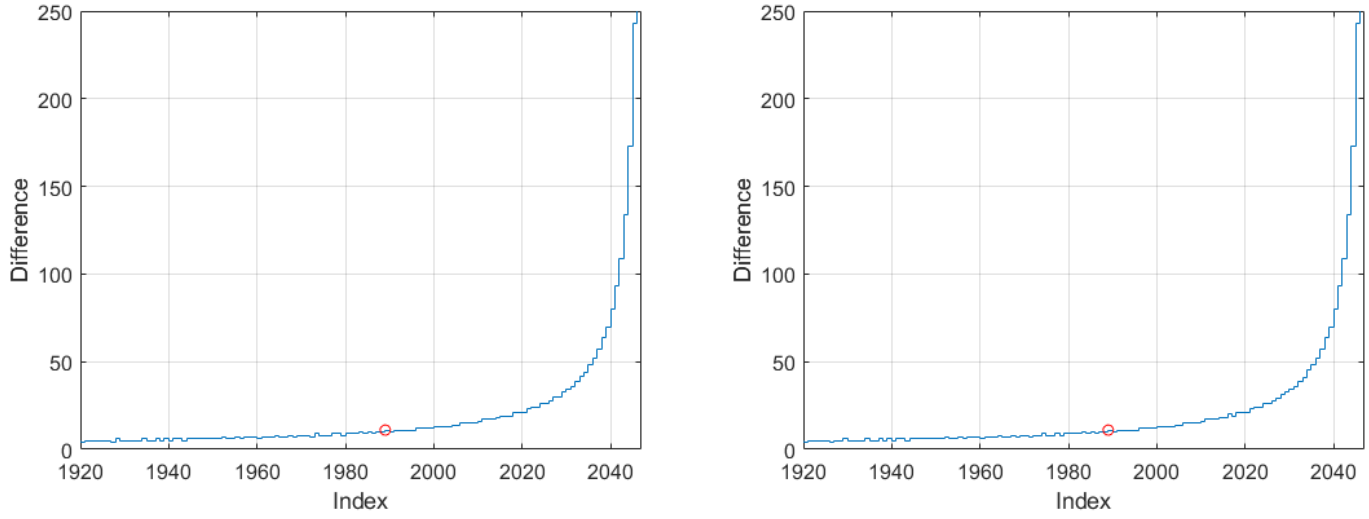
6.2. Exponential Distribution

Simulations in nuclear and high-energy physics commonly use exponential distributions to model deposited energy or particle yields. A typical parameterization found in recent works [21,22] models particle energies using an exponential distribution with scale parameter equal to 600 MeV. The probability density function used as reference is shown in (5):

$$f(x) = \begin{cases} \frac{1}{600}\, e^{-\frac{1}{600}x}, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{5}$$

In this evaluation, LUTs containing 2048 entries were adopted. The uniform pseudo-random generator provides 11-bit addresses, ensuring uniform coverage of all memory positions. To define the boundaries between consecutive LUTs, the differences between sequential memory entries were examined. Figure 15a,b show the difference patterns for two memories. In both cases, the transition threshold corresponds to the point where

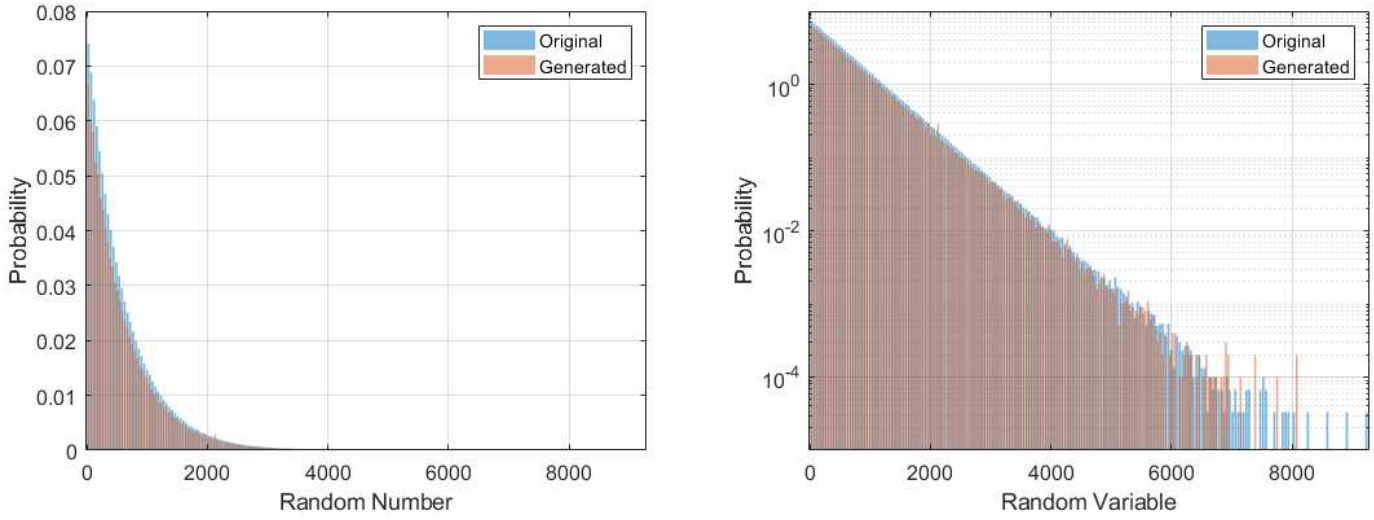
the relative difference equals 1%, occurring at address 1989. Although the same value appeared in both memories in this example, this is not a requirement and typically varies from memory to memory.



(**a**) (**b**)

Figure 15. (**a**) Difference between consecutive entries of the first memory. (**b**) Equivalent analysis for the second memory.

The final exponential-distributed samples are shown in Figure 16a. A logarithmic scale (Figure 16b) highlights the tail behavior, demonstrating that the generated distribution remains consistent with the analytical reference even in low-probability regions.



(**a**) (**b**)

Figure 16. (**a**) Histogram of generated exponential-distributed samples. (**b**) Histogram with logarithmic scale applied to the vertical axis.

A quantitative assessment of the generated exponential random variables is presented in Tables 3 and 4. Both the proposed multi-memory implementation and a reference single-LUT approach were evaluated using one million samples. No negative samples were observed, confirming the correct support of the exponential distribution with $\lambda = 1/600$.

Table 3 summarizes cumulative-distribution-based accuracy metrics. The maximum KS deviation D_{KS} is reported as a compact measure of global agreement, while the mean squared error (MSE) and the maximum absolute deviation of the empirical CDF quantify

local discrepancies. To explicitly assess the low-probability region, the relative error of the empirical tail probability at the 99.9% quantile is also reported.

Table 3. CDF-based accuracy metrics for the exponential RNG outputs.

Method	D_{KS}	CDF_{MSE}	CDF_{max}	ε_{tail}
Multi-Memory	2.4×10^{-3}	9.4×10^{-7}	2.4×10^{-3}	−1.8%
Single-LUT	2.0×10^{-3}	7.4×10^{-7}	2.0×10^{-3}	−1.2%

Table 4. Statistical moments of the generated exponential distributions compared with theoretical expectations.

Method	μ	σ^2	γ_1	γ_2
Multi-memory	601.11	3.63×10^5	1.99	5.86
Single-LUT	601.10	3.63×10^5	1.99	5.93
Theoretical	600	3.60×10^5	2	6

The results indicate that both implementations achieve very similar levels of statistical accuracy. Although the single-LUT exhibits slightly smaller CDF deviations due to its finer discretization, the differences remain small and within the same order of magnitude. In particular, the relative tail error remains below 2% for both methods, demonstrating that the proposed segmentation strategy preserves accuracy even in low-probability regions relevant to detector simulations.

The statistical moments of the generated distributions are reported in Table 4. The estimated mean μ and variance σ^2 closely match the analytical values, while the skewness γ_1 and excess kurtosis γ_2 remain near their theoretical expectations.

These results confirm that the proposed multi-memory architecture reproduces the overall shape of the exponential distribution with high fidelity.

Special attention was given to the tail region, which is critical in detector simulations. At the 99.9% quantile, the empirical tail probabilities closely match the theoretical values, with relative errors below 2% for both implementations. This demonstrates that the proposed segmentation strategy preserves accuracy even in low-probability regions.

The estimated statistical moments are also consistent with the analytical reference. The mean and variance match the theoretical values within negligible relative error, while the skewness and excess kurtosis remain close to the expected values for an exponential distribution.

The FPGA resource utilization of the exponential RNG implementations is summarized in Table 5, including adaptive logic module (ALM) usage, register count, on-chip memory consumption, and maximum operating frequency.

The proposed multi-memory architecture requires only a small fraction of the available logic resources and does not rely on DSP blocks, which confirms that the generation of non-uniform random variables can be achieved using simple combinational logic and registers. The moderate logic footprint reflects the fixed-latency and fully parallel addressing strategy adopted in the design.

Table 5. DE10-Nano-SoC resource utilization for exponential RNG implementations.

Implementation	ALMs	Registers	Memory Bits	Max. f_{clk}
Multi-memory	307	1011	79,872 (1%)	225 MHz
Single-LUT	254	–	3,407,872 (60%)	132 MHz

A key aspect of the proposed approach is its efficient use of on-chip memory. By distributing the ICDF representation across multiple smaller LUTs, the required memory footprint is drastically reduced while preserving statistical accuracy, as demonstrated. This characteristic is particularly relevant for FPGA-based detector simulations, where memory resources often represent a limiting factor.

For reference, a large single-LUT implementation storing the complete ICDF was also evaluated under identical conditions. As expected, this approach achieves comparable statistical performance but at the cost of a substantially larger memory allocation. While the logic utilization remains of the same order, the single-LUT design consumes a dominant fraction of the available on-chip memory and exhibits a lower maximum operating frequency due to increased memory access and routing complexity.

Taken together, the results demonstrate that the proposed multi-memory architecture achieves statistical accuracy comparable to that of a much larger single-LUT implementation, while requiring nearly two orders of magnitude less on-chip memory. The improvement offered by the proposed method therefore lies not in surpassing the absolute accuracy of a large LUT, but in reproducing its statistical behavior with a dramatically reduced hardware cost, making it particularly attractive for large-scale FPGA-based detector simulations.

6.3. White Gaussian Noise Implementation

Gaussian noise is widely used in detector simulations, especially for modeling electronic noise and fluctuations in energy deposition. Following the example in [21], a zero-mean Gaussian distribution with a standard deviation of 20 MeV was targeted. The corresponding PDF is given by

$$F(x) = \frac{1}{20\sqrt{2\pi}} e^{-x^2/800}. \tag{6}$$

Figure 17 shows the cumulative distribution function, where the asymptotically flat tails imply that covering the entire range uniformly would require large memories. Because the Gaussian distribution is symmetric, only its positive half was stored in LUTs. A separate 1-bit random generator selects the sign: a value of one assigns a negative sign, while zero corresponds to a positive output.

The positive region was partitioned into four memories of 512 entries each (16-bit values). The threshold values used to define the partitions were based on a relative difference of 1% for the first three memories and 2% for the last memory. Figure 18a shows the histogram of the positive samples. After applying the sign generator, the complete Gaussian distribution is obtained (Figure 18b). Because the FPGA operates in fixed-point arithmetic, a scaling factor of 512 was applied to the numeric values to avoid floating-point representation.

The statistical validation of the generated Gaussian random variables focuses on both cumulative distribution accuracy and distribution shape. In addition to global goodness-of-fit metrics, special attention is given to symmetry and tail behavior, which are critical for noise modeling in detector simulations.

Table 6 summarizes cumulative-distribution-based metrics for the Gaussian RNG outputs. The Kolmogorov–Smirnov statistic and the maximum CDF deviation indicate good agreement with the target distribution for both implementations. The proposed multi-memory architecture exhibits slightly smaller CDF deviations in this configuration.

To explicitly evaluate rare-event accuracy, the empirical probability beyond 3σ was compared with the theoretical Gaussian tail probability. Both implementations reproduce the tail behavior within a relative error below 10%, with the multi-memory approach yielding a smaller deviation in this region.

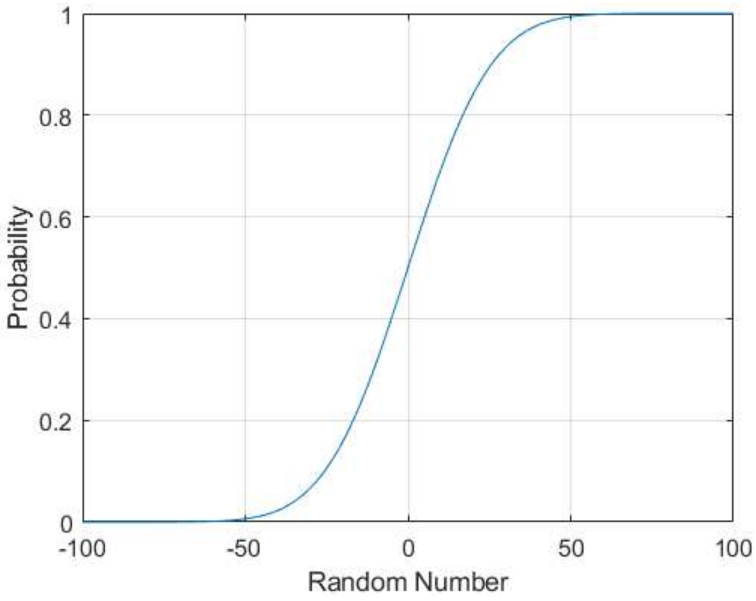
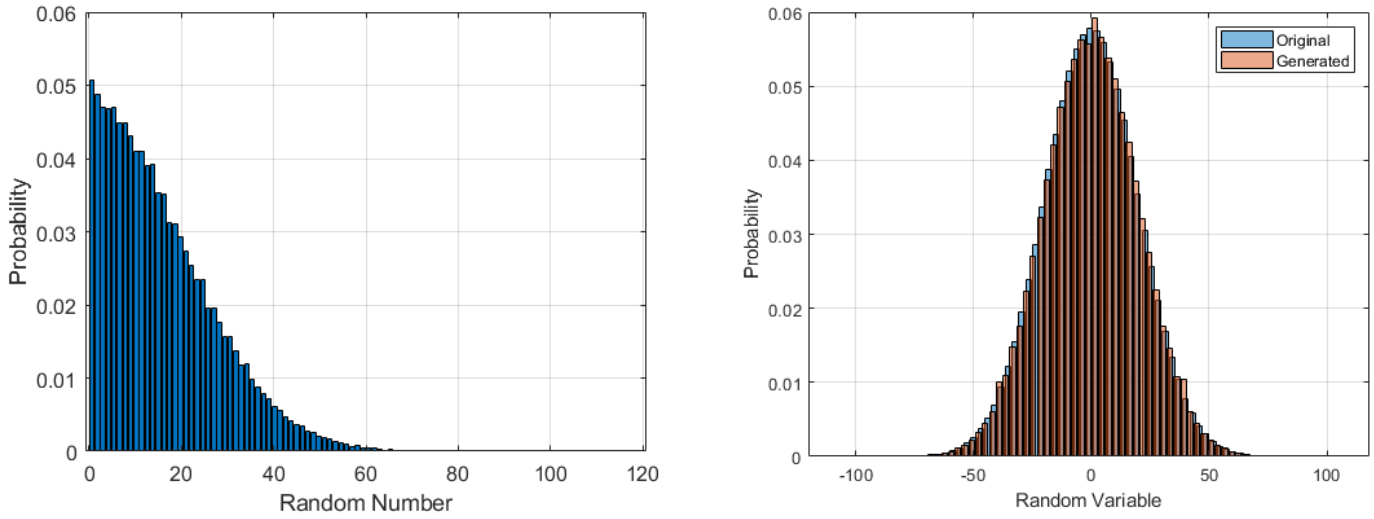


Figure 17. CDF of the Gaussian distribution.



(**a**) (**b**)

Figure 18. (**a**) Histogram of the positive region of the Gaussian distribution. (**b**) Histogram of the complete generated Gaussian noise.

Table 6. CDF-based accuracy metrics for the Gaussian RNG outputs.

Method	D_{KS}	CDF_{MSE}	$\mathrm{CDF}_{\max}$	$P(\lvert x\rvert > 3\sigma)$	$\varepsilon_{\mathrm{tail}}$
Multi-memory	2.05×10^{-3}	4.07×10^{-7}	2.05×10^{-3}	2.57×10^{-3}	-4.8%
Single-LUT	3.49×10^{-3}	1.64×10^{-6}	3.49×10^{-3}	2.93×10^{-3}	$+8.7\%$

The KS statistic indicates good agreement with the target Gaussian distribution for both methods. The observed maximum deviations of the empirical CDF remain below 0.4%, with the multi-memory implementation exhibiting slightly smaller deviations in this configuration. These differences are primarily associated with discretization and fixed-point effects in the tails, rather than systematic bias.

In addition to cumulative-distribution-based metrics, the shape and symmetry of the generated Gaussian noise were evaluated through their statistical moments. Table 7 reports

the estimated mean μ, variance σ^2, skewness γ_1, and excess kurtosis γ_2, allowing direct comparison with the theoretical Gaussian distribution.

Table 7. Statistical moments of the generated Gaussian distributions.

Method	μ	σ^2	γ_1	γ_2
Multi-memory	13.24	1.05×10^8	−0.002	−0.024
Single-LUT	16.21	1.04×10^8	−0.0003	0.058
Theoretical	0	400	0	0

Both implementations closely reproduce the theoretical Gaussian tail probability, with relative errors below 10%. The multi-memory architecture yields a slightly smaller relative tail error, indicating that the segmentation strategy preserves accuracy even in low-probability regions.

The statistical moments reported in Table 7 confirm the quality of the generated Gaussian noise. The estimated mean remains close to zero and the variance matches the target value, while the skewness and excess kurtosis remain near zero, indicating good symmetry and consistency with the Gaussian distribution shape. These results confirm that the sign-bit reconstruction does not introduce measurable bias, including in the vicinity of zero crossings.

The quantization factor depends on the accuracy required to approximate the original distribution. The proposed method stores only the generated ICDF values, and increasing or decreasing the quantization factor affects solely the bit width of each memory entry. Furthermore, if floating-point representation is required, it can be adopted at the cost of using 32-bit memory words.

The FPGA resource utilization for the Gaussian RNG implementations is summarized in Table 8. The proposed multi-memory architecture requires a modest increase in logic utilization due to the additional control logic and sign handling, but achieves a substantial reduction in on-chip memory usage. This reduction is enabled by exploiting the symmetry of the Gaussian distribution and by concentrating resolution where it is most needed.

Table 8. DE10-Nano-SoC resource utilization for Gaussian RNG implementations.

Implementation	ALMs	Registers	Memory Bits	Max. f_{clk}
Multi-memory	578	1623	34,816 (<1%)	143 MHz
Single-LUT	180	–	138,264 (2%)	194 MHz

The reference single-LUT implementation achieves comparable statistical accuracy with a smaller logic footprint, but at the cost of a significantly larger memory allocation. Moreover, its higher maximum operating frequency does not offset the increased memory usage in scenarios where multiple parallel channels must be instantiated.

Overall, the results confirm that the proposed multi-memory architecture provides an efficient and scalable solution for Gaussian noise generation in FPGA-based detector simulations, offering a favorable trade-off between statistical accuracy, memory usage, and architectural complexity.

6.4. High Energy Physics Application

Two examples of random number distributions commonly encountered in high-energy physics studies were presented. These studies often rely on software-based simulations, where testing algorithms in real time is generally not feasible. Therefore, implementing a real-time simulator in hardware, such as an FPGA, can significantly assist researchers in evaluating and validating signal processing techniques.

Considering the LHC, specifically the Tile Calorimeter of the ATLAS experiment, the number of readout channels reaches approximately 10,000, with calorimeter towers comprising more than 100 channels each. In this context, a real-time simulator for a single tower must include both exponential and Gaussian random number generators for every channel. Consequently, resource utilization becomes a critical factor to ensure that all channels can be accommodated within a single FPGA device.

This multi-channel requirement is not exclusive to TileCal but is also present in other high-energy physics experiments, which can similarly benefit from the proposed implementation.

Considering one exponential distribution and one noise distribution per channel, the total block memory usage is 114,688 bits. On the DE10-Nano-SoC board, this memory requirement allows the implementation of up to 49 channels when considering block memory constraints. In terms of ALMs, the design supports up to 47 channels, with a total usage of 885 ALMs. As memory usage increases, the maximum number of implementable channels decreases accordingly.

Another important performance metric is the maximum operating frequency. The proposed simulator achieves a maximum frequency of 143 MHz, which is higher than the operating frequencies typically required in high-energy collider experiments (e.g., 40 MHz). Therefore, the proposed technique is well suited for real-time simulators intended to generate signals that emulate detector readout channels.

7. Conclusions

This work presented a new approach for synthesizing random variables with arbitrary probability distributions using a purely LUT-based strategy suitable for FPGA implementation. By partitioning the cumulative distribution function into multiple memory blocks, the method improves the representation of poorly sampled regions without increasing circuit complexity. The technique depends solely on a uniformly distributed random number generator, previously demonstrated in earlier studies [14], making it a lightweight and modular solution.

The two case studies—exponential and Gaussian distributions—demonstrated that the architecture requires only about 3% of the available ALMs on the DE10-Nano-SoC device, uses no DSP blocks, and achieves high operating frequencies. These characteristics make the method well suited for real-time hardware-based simulators intended to support research in high-energy physics and related fields.

The proposed multi-memory architecture reproduces the statistical behavior of large single-LUT implementations with substantially reduced memory requirements and fixed latency, making it well suited for FPGA-based detector simulations. Future work may include the application of formal randomness test suites, such as DIEHARDER or NIST SP 800-22, to further characterize the generated sequences under alternative validation frameworks.

Despite its advantages, the method also presents certain limitations. The implementation is highly distribution-specific: even small modifications to the target distribution parameters (e.g., mean or standard deviation) may require recalculating memory boundaries or increasing the number of LUTs. Furthermore, generating optimized memory contents typically requires offline processing by domain experts, as the complete workflow cannot be reduced to a simple change of parameters in an analytical expression. While some steps can be automated through dedicated software tools, they must still be validated prior to hardware deployment.

Several avenues for future work remain open. Extending the proposed method to model additional probability distributions would broaden its applicability. Integrating the generator into a real-time simulation platform would enable the benchmarking of

online signal-processing algorithms under realistic conditions. Developing an automated procedure to determine the required number of memory layers is also essential to eliminate the need for expert-driven fine tuning.

Another relevant improvement involves storing floating-point values directly in LUTs, thereby avoiding fixed-point quantization constraints and further enhancing accuracy and flexibility. Additionally, a comprehensive evaluation of fan-out, latency, and power consumption constitutes an important direction for future investigation.

Author Contributions: Conceptualization, T.C.A.P., T.M.Q. and L.M.d.A.F.; methodology, T.C.A.P. and L.M.d.A.F.; software, T.C.A.P.; validation, T.C.A.P., T.M.Q. and L.M.d.A.F.; formal analysis, T.C.A.P., T.M.Q. and L.M.d.A.F.; data curation, L.M.d.A.F.; writing—original draft preparation, T.C.A.P.; writing—review and editing, T.C.A.P., T.M.Q. and L.M.d.A.F.; visualization, T.C.A.P., T.M.Q. and L.M.d.A.F.; supervision, T.M.Q. and L.M.d.A.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available at https://cernbox.cern.ch/s/HAzH4fyeHfF0s4e (accessed on 1 March 2026).

Acknowledgments: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES)—Finance Code 001. The authors are thankful to CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) and RENAFAE (Rede Nacional de Física de Altas Energias) for their support.

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

Abbreviations

The following abbreviations are used in this manuscript:

TRNG	True Random Number Generator
PRNG	Pseudo-Random Number Generator
FPGA	Field-Programmable Gate Array
LFSR	Linear Feedback Shift Register
ICDF	Inverse Cumulative Distribution Function
CDF	Cumulative Distribution Function
XOR	Exclusive OR
LSB	Least Significant Bit
MSB	Most Significant Bit
PDF	Probability Density Function
SiPM	Silicon Photomultiplier
LUT	Look-Up Table
ALM	Adaptive Logic Module
DSP	Digital Signal Processor

References

1. Asmussen, S.; Glynn, P.W. *Stochastic Simulation: Algorithms and Analysis*; Springer: New York, NY, USA, 2007.
2. Coates, R.F.W.; Janacek, G.J.; Lever, K.V. Monte Carlo simulation and random number generation. *IEEE J. Sel. Areas Commun.* **1988**, *6*, 58–66. [CrossRef]
3. Wen, Y.; Yu, W. Machine learning-resistant pseudo-random number generator. *Electron. Lett.* **2019**, *55*, 515–517. [CrossRef]
4. Ezilarasan, M.R.; Britto, P.J.; Leung, M. High performance FPGA implementation of single MAC adaptive filter for independent component analysis. *J. Circuits Syst. Comput.* **2023**, *32*, 2350294. [CrossRef]

5. Costa, V.L.R.d.; Lopez, J.; Ribeiro, M.V. A system-on-a-chip implementation of a post-quantum cryptography scheme for smart meter data communications. *Sensors* **2022**, *22*, 7214. [CrossRef] [PubMed]
6. Quirino, T.M.; Paschoalin, T.C.A.; Gonçalves, G.I.; Lisboa, P.H.B.; Andrade Filho, L.M.d.; Peralva, B.S.-M. Online pulse compensation for energy spectrum determination: A pole-zero cancellation and unfolding approach. *Electronics* **2025**, *14*, 493. [CrossRef]
7. Knuth, D.E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed.; Addison-Wesley: Reading, MA, USA, 1998.
8. Li, Y.; Chow, P.; Jiang, J.; Zhang, M.; Wei, S. Software/hardware parallel long-period random number generation framework based on the WELL method. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2014**, *22*, 1054–1059. [CrossRef]
9. Madhupavani, G.; Suneetha, K. Design of random number generation using 256-bit LFSR in FPGA. *Int. J. Adv. Technol. Innov. Res.* **2017**, *9*, 1105–1110.
10. Durga, R.S.; Rashmika, C.; Madhumitha, O.N.; Suvetha, D.; Tanmai, B.; Mohankumar, N. Design and synthesis of LFSR-based random number generator. In Proceedings of the 2020 3rd International Conference on Smart Systems and Inventive Technology (IC-SSIT), Tirunelveli, India, 20–22 August 2020; pp. 438–442.
11. Bailey, K. FPGA implementation of reversible LFSR with primitive polynomial using Verilog HDL. In Proceedings of the 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballari, India, 23–24 April 2022; pp. 1–5.
12. Gudla, V.V.; Jyothi, V.S.S.S.S. Design and implementation of digital clock manager based pseudo-true random number generator. In Proceedings of the 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 7–9 October 2022; pp. 1–5.
13. Zode, P.; Zode, P.; Deshmukh, R. FPGA-based novel true random number generator using LFSR with dynamic seed. In Proceedings of the 2019 IEEE 16th India Council International Conference (INDICON), Rajkot, India, 13–15 December 2019; pp. 1–3.
14. Paschoalin, T.C.A.; Dias, U.R.F.; Aguiar, M.S.; Santos, D.F.d.; Quirino, T.M.; Andrade Filho, L.M.d. Uncorrelated pseudo-random generator for FPGA. In Proceedings of the 2025 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI), Manaus, Brazil , 25 August–1 September 2025; pp. 1–5. [CrossRef]
15. Thomas, D.B.; Luk, W.; Leong, P.H.W.; Villasenor, J.D. Gaussian random number generators. *ACM Comput. Surv.* **2007**, *39*, 1–38. [CrossRef]
16. Aniruddhan, S. Quadrature generation techniques in CMOS relaxation oscillators. In Proceedings of the 2012 IEEE International Symposium on Circuits and Systems (ISCAS), Seoul, Republic of Korea, 20–23 May 2012; pp. 1375–1378. [CrossRef]
17. de Schryver, C.; Schmidt, D.; Wehn, N.; Korn, E.; Marxen, H.; Korn, R. A new hardware-efficient inversion-based random number generator for non-uniform distributions. In Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 13–15 December 2010; pp. 190–195. [CrossRef]
18. Lee, D.-U.; Cheung, R.C.C.; Villasenor, J.D.; Luk, W. Inversion-based hardware Gaussian random number generator. In Proceedings of the 2006 IEEE International Conference on Field Programmable Technology, Bangkok, Thailand, 13–15 December 2006; pp. 33–40. [CrossRef]
19. Banks, S.; Beadling, P.; Ferencz, A. FPGA implementation of pseudo-random number generators for Monte Carlo methods in quantitative finance. In Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 3–5 December 2008; pp. 271–276. [CrossRef]
20. Brown, R.G. Dieharder Test Suite. 2014. Available online: https://webhome.phy.duke.edu/~rgb/General/dieharder.php (accessed on 1 March 2026).
21. Duarte, J.; Filho, L.M.A.; Filho, E.F.S.; Farias, P.; de Seixas, J.M. Online energy reconstruction for calorimeters under high pile-up conditions using deconvolutional techniques. *J. Instrum.* **2019**, *14*, P12017. [CrossRef]
22. Barbosa, D.P.; Filho, L.M.d.; Peralva, B.S.; Cerqueira, A.S.; Seixas, J.M.d. Sparse representation for signal reconstruction in calorimeters operating in high luminosity. *IEEE Trans. Nucl. Sci.* **2017**, *64*, 1942–1949. [CrossRef]
23. Teixeira, T.; Andrade Filho, L.M.d.; Seixas, J.M.d. Sparse deconvolution methods for online energy estimation in calorimeters operating in high luminosity conditions. *J. Instrum.* **2021**, *16*, P09008. [CrossRef]
24. Quirino, T.M.; Andrade Filho, L.M.d. Non-negative sparse deconvolution method for PMT signals in radiation detectors. *Nucl. Instrum. Methods Phys. Res. Sect. A* **2024**, *1061*, 169142. [CrossRef]
25. Quirino, T.; Cit, G.; Cesário, G.; Inácio, S.; Pereira, A.; Massafferri, A.; Manhães, L.; Rocha, D. Application of Wiener filter and sparsity theory for pulse unfolding from scintillator detector. In Proceedings of the 2024 International Symposium on Instrumentation Systems, Circuits and Transducers (INSCIT), Bento Goncalves, Brazil, 2–6 September 2024; pp. 1–6. [CrossRef]
26. Rimes, S.d.; Peralva, B.S.M.; Libotte, G.B.; Paschoalin, T.C.A.; Filho, L.M.d.A. A maximum likelihood approach for signal estimation and data quality assessment for high-energy calorimeter systems. *IEEE Trans. Nucl. Sci.* **2025**, *72*, 3801–3808. [CrossRef]
27. St. Denis, T.; Johnson, S. *Cryptography for Developers*; Syngress: Burlington, MA, USA, 2007; pp. 91–137; ISBN 978-1-59749-104-4.

28. Shonkwiler, R.W.; Mendivil, F. Some probability distributions and their uses. In *Explorations in Monte Carlo Methods*; Undergraduate Texts in Mathematics; Springer: Cham, Switzerland, 2024. [CrossRef]
29. Azriel, D.; Schwartzman, A. The empirical distribution of a large number of correlated normal variables. *J. Am. Stat. Assoc.* **2015**, *110*, 1217–1228. [CrossRef] [PubMed]
30. Glasserman, P. *Monte Carlo Methods in Financial Engineering*; Springer: New York, NY, USA, 2004.
31. Evans, L.; Bryant, P. LHC machine. *J. Instrum.* **2008**, *3*, S08001. [CrossRef]
32. Wigmans, R. *Calorimetry: Energy Measurement in Particle Physics*; Oxford University Press: Oxford, UK, 2000.
33. Intel Corporation. DE10-Nano User Manual. Available online: https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Nano/DE10_Nano_User_Manual.pdf (accessed on 18 December 2025).