Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A High-Throughput DSP-Free PRNG for FPGA based on Xoshiro128** and Integer Chaotic Mixing

Milad Mohammadian
Urmia University
Bahadin Azizinasab
University of Guilan
Abobakr Mohammadian
Urmia University
Mahrokh G. Shayesteh
m.shayesteh@urmia.ac.ir
Urmia University
Mohammad Saeed khayati
University of Tabriz
Sheida Mohammadpour
Semnan University
Mosleh yousefi
University of Guilan

Article

Keywords:

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11422754/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A High-Throughput DSP-Free PRNG for FPGA based on Xoshiro128** and Integer Chaotic Mixing

Milad Mohammadian · Bahadin Azizinasab · Abobakr Mohammadian · Mahrokh G. Shayesteh · Mohammad Saeed Khayaty · Sheida Mohammadpour · Mosleh Yousefi

Received: date / Accepted: date

Abstract This paper presents a resource-conscious hybrid pseudo-random number generator (PRNG) core implemented in an out-of-context (OOC) flow on the Xilinx Zynq-7000 SoC platform. The proposed architecture combines a Xoshiro128** state engine with a Weyl sequence and a discretized integer Tent map in a compact datapath. The design uses branchless integer arithmetic only and does not require digital signal processing (DSP) slices or Block RAM. The fully pipelined implementation achieves an initiation interval of 1 and sustains a throughput of 5.82 Gbps at 181.82 MHz. Post-implementation results on a Zynq xa7z010 device show a footprint of 934 slice LUTs, 1071 registers, and a dynamic power of 29 mW (zero-delay estimate, a lower bound). The generator passes the NIST SP 800-22, Dieharder, and TestU01 BigCrush test suites in the reported evaluation, and exhibits near-uniform output statistics and measurable sensitivity to changes in the initialization state. The six active 32-bit initialization parameters define a nominal 192-bit initialization-input space. An ablation study isolates each layer: the Weyl and Tent stages do not improve the statistical quality already provided by Xoshiro128**, but they remove its linear output structure and extend the state-sequence period, at a cost of roughly 36% additional LUTs and no DSP or BRAM. These results suggest that the proposed design may be suitable for resource-constrained embedded applications that require a compact, DSP-free statistical random-number source.

Keywords Chaos-based PRNG, FPGA, Xoshiro128**, DSP-free, High-Level Synthesis (HLS), TestU01, BigCrush.

M. Mohammadian
Faculty of Electrical Engineering and Computer Science, Urmia University, Urmia, Iran
E-mail: miladmohammadian1995@gmail.com

B. Azizinasab
Department of Electrical Engineering, University of Guilan, Rasht, Iran
E-mail: b_azizinasab@msc.guilan.ac.ir

A. Mohammadian
Faculty of Electrical Engineering and Computer Science, Urmia University, Urmia, Iran
E-mail: ab.mohammadian@urmia.ac.ir

M. G. Shayesteh
Faculty of Electrical Engineering and Computer Science, Urmia University, Urmia, Iran
E-mail: m.shayesteh@urmia.ac.ir
Corresponding author: M. G. Shayesteh

M. S. Khayaty
Faculty of Mechanical Engineering, University of Tabriz, Tabriz, Iran
E-mail: mohammad.s.khayyati96@ms.tabrizu.ac.ir

S. Mohammadpour
Department of Electrical and Computer Engineering, Semnan University, Semnan, Iran
E-mail: sh.mohammadpour@alumni.sbu.ac.ir

M. Yousefi
Department of Electrical Engineering, University of Guilan, Rasht, Iran
E-mail: mosleh_yousefi@msc.guilan.ac.ir

1 Introduction

1.1 Motivation

A pseudo-random number generator (PRNG) is a deterministic algorithm that produces a sequence of values that appears random when initialized with a seed. PRNGs are widely used in cryptography, computational science, and simulation, where long period lengths and statistical unpredictability are important [1, 2].

Conventional linear PRNGs have well-known structural limitations: their output obeys a linear recurrence that can be detected by linear-complexity tests, and if the internal state is inferred from the output, the remaining sequence becomes predictable. This motivates continued research on generators whose output does not expose such simple structure [3].

The sensitivity of nonlinear dynamical systems to initial conditions, often referred to as the butterfly effect, makes chaos-based designs an interesting alternative. Chaotic systems use deterministic but highly nonlinear processes to transform small differences in initial conditions into substantially different output sequences. Chaotic PRNGs are often based on nonlinear maps such as Logistic, Tent, and Hénon [4]. To improve the statistical properties of chaos-based PRNGs, the work in [5] proposed a method for constructing 4D Hamiltonian conservative chaotic systems based on generalized Euler equations, investigating hidden conservative chaos and coexisting orbits.

Chaos-based generators have also been proposed for resource-limited edge and internet of things (IoT) devices [6]. At the same time, software-based PRNGs may not meet the throughput and latency requirements of high-speed systems, which motivates hardware-accelerated solutions.

Implementing chaotic PRNGs on FPGAs involves a trade-off among statistical quality, throughput, and hardware footprint. Standard continuous chaotic maps often require high-precision floating-point arithmetic to avoid dynamical degradation, such as short cycle lengths and reduced entropy, which can increase the use of FPGA resources, particularly digital signal processing (DSP) slices, as well as power consumption. On the other hand, lightweight implementations do not always satisfy stringent test suites such as TestU01 BigCrush [7].

1.2 Related Studies

The integration of chaos-based PRNGs into reconfigurable hardware was studied in several prior works. Early approaches focused on increasing dynamical complexity to improve statistical behavior, including n-dimensional chaotic systems [8], reconfigurable Lorenz/Lü designs [9], coupled map lattices [10], weakly coupled discrete maps [11], and one-dimensional cascading nonlinear models [12]. However, these coupling mechanisms can increase logic utilization and path latency. Recursive structures may also make high-performance pipelining more difficult, as seen in the generalized Hénon map [13]. Higher-dimensional frameworks, such as 5D hyperchaotic memristive topologies [14] and multi-type hidden attractors [15], can provide larger state spaces but may also reduce throughput due to computational complexity.

Artificial neural networks (ANNs) and memristive systems were also used to model hyperchaotic dynamics. Frameworks based on feed-forward neural networks (FNNs) achieved throughputs around 166 Mbps [16, 17], and up to 301 Mbps on a Kintex-7 FPGA [18]. In contrast, memristive Hopfield networks using CORDIC algorithms can consume substantial hardware resources, yielded at 6.94 Mbps throughput [19]. Multistable neural attractors introduced state-allocation strategies that add implementation overhead [20]. While some discrete-time memristive variants reached 1.2 Gbps [21], their statistical validation may still be limited in some cases.

To address the throughput bottleneck, recent studies focused on hardware-friendly optimization and hybrid architectures [22, 23]. A combination of ANN oscillators and ring oscillators achieved 241 Mbps [24], while a Tinkerbell map with an adaptive perturbation mechanism achieved 2.33 Gbps, but required 16 DSP slices [2]. To maximize bit rates, discrete hyperchaotic systems with cross-coupled topologies reached 10.28 Gbps, although at the cost of higher architectural complexity [25]. Furthermore, architectures such as the one in [2] were validated only with SmallCrush, so their behavior under more exhaustive batteries remains to be further evaluated.

1.3 Contributions

The proposed architecture addresses the trade-off between statistical quality, throughput, and hardware footprint at the microarchitectural level by using integer-based operations, a fully pipelined structure, and no DSP slices or Block RAMs (BRAM) in the reported implementation. With these design choices, the implementation achieves a sustained throughput of 5.82 Gbps. The resulting design offers a practical trade-off between statistical behavior, throughput, and hardware footprint, which may be useful for resource-constrained embedded systems. Combining a linear generator with a Weyl sequence, or post-processing a generator with a chaotic map, is not new in itself [23, 26]; the contributions of this work therefore lie in the specific DSP-free hardware realization of the fused generator and in the per-layer analysis of what each added layer costs and changes, which shows that the two added layers are effective in combination rather than individually. The contributions of this work are highlighted as follows:

1) *Branchless Integer Mapping for the Chaotic Layer:* We propose a constant-time hardware mapping for the piecewise chaotic layer. The most significant bit (MSB) is used as the multiplexer select signal, which avoids control-flow branching. The design uses shift-and-add logic and array partitioning, and in this implementation does not require dedicated DSP slices or embedded memory blocks.
2) *HLS Optimization for Pipelined Execution:* Using C-to-RTL optimization directives, the data dependencies in the multi-domain state updates are reduced. The HLS compiler scheduled the main loop with an initiation interval of 1. This allows the concurrent execution of GF(2) recurrences and integer ring additions, producing one 32-bit output word per clock cycle under the reported timing constraints.
3) *Post-Implementation Metrics:* In an out-of-context (OOC) implementation on a Xilinx Zynq xa7z010 device, the core achieves a clock frequency of 181.82 MHz. The implementation reports a throughput of 5.82 Gbps, 934 Slice LUTs, and 29 mW of dynamic power (zero-delay estimate, a lower bound; 42 mW including glitch activity).
4) *Statistical and Sensitivity Evaluation:* The generator is evaluated using TestU01 BigCrush, NIST SP 800-22, and Dieharder. Additional measurements of entropy, autocorrelation, and differential sensitivity are also performed. The NPCR and UACI metrics quantify the output sensitivity to the initialization state; as we show, this sensitivity is inherited from the Xoshiro128** engine.
5) *Ablation and Structural Analysis:* We isolate the contribution of each layer (plain Xoshiro128**, +Weyl, +Tent, and the full design) under the same FPGA flow, reporting resources, timing, and power. We do not claim that the added layers improve statistical quality, which plain Xoshiro128** already provides, but show that they remove the linear output structure of plain Xoshiro128** (direct state recovery and a key-independent linear-complexity distinguisher) and extend the state-sequence period. Each layer alone raises the cost of the best state-recovery attack we found only to about 2^{32} guesses, whereas the two layers together require a joint guess of about 2^{63}. We also provide an exhaustive analysis of the integer Tent map. All results are reproducible from a deposited package.

The remainder of this paper is organized as follows: The mathematical formulation and structural properties of the proposed hybrid chaotic algorithm are explained in Section 2. Section 3 describes the custom FPGA microarchitecture, high-level synthesis integration, and hardware verification methodology. Section 4 presents the experimental results, including empirical statistical testing and additional analyses of the generated sequences. Finally, Section 5 concludes the paper and discusses possible research directions.

2 Proposed Hybrid Architecture and Mathematical Formulation

This section presents the mathematical formulation and internal state evolution of the proposed hybrid PRNG. The generator combines three complementary deterministic mechanisms: i) a Xoshiro128**-based state engine, ii) a Weyl sequence mixer, and iii) a discretized integer Tent map. These components operate according to distinct mathematical structures and provide complementary mixing mechanisms. The Xoshiro128** core serves as the primary long-period state engine, the Weyl sequence introduces additive mixing in the integer domain, and the integer Tent map provides a piecewise stretch-and-fold state transformation. The resulting component outputs are subsequently fused to generate the final pseudorandom sequence.

2.1 The Deterministic Backbone: Xoshiro128**

The xoshiro128** algorithm is employed as the primary state engine owing to its long theoretical period and efficient state-transition mechanism [27]. At time t, the internal state is represented by the vector $S_t = [s_0, s_1, s_2, s_3]$, where s_0, s_1, s_2, s_3 denote four independent 32-bit state registers that collectively form the 128-bit internal state and are updated at each iteration according to the Xoshiro transition rules. The all-zero state is excluded from the valid state space during initialization [27]. The output of the Xoshiro128** core is then generated using a nonlinear scrambling function [27] as

$$R_t = \mathrm{rotl}(s_1 \times 5, 7) \times 9 \tag{1}$$

where $\mathrm{rotl}(x, k)$ denotes a circular left rotation of the 32-bit word x by k bit positions. In contrast to a logical shift, a circular rotation reinserts the bits shifted out of one side of the word into the opposite side. Since the multipliers 5 and 9 are odd, they are invertible modulo 2^{32}, preserving the bijective nature of the corresponding modular multiplication operations. The combination of modular multiplication and circular rotation provides nonlinear bit diffusion in the output transformation [27].

Following output generation, the internal state is updated according to [27]:

$$T = s_1 \ll 9 \tag{2}$$
$$s_2 = s_2 \oplus s_0 \tag{3}$$
$$s_3 = s_3 \oplus s_1 \tag{4}$$
$$s_1 = s_1 \oplus s_2 \tag{5}$$
$$s_0 = s_0 \oplus s_3 \tag{6}$$
$$s_2 = s_2 \oplus T \tag{7}$$
$$s_3 = \mathrm{rotl}(s_3, 11) \tag{8}$$

where $\oplus$ denotes the bitwise exclusive-OR (XOR) operation and $\ll$ denotes a logical left shift. The state transition mixes the four 32-bit words through a sequence of XOR operations, a shifted intermediate term, and a circular rotation. This recurrence provides the deterministic long-period state evolution of the backbone, whereas the nonlinear output transformation enhances the diffusion of the generated values.

2.2 Multi-Domain Mixing Layers

To complement the xoshiro128** state evolution, two auxiliary mixing layers are incorporated into the proposed PRNG: a Weyl sequence mixer and a discretized integer Tent map. These layers introduce additive and piecewise stretch-and-fold transformations that differ from the XOR-based state transition of the Xoshiro128** core. In the following, each layer is explained.

Layer 1: The Weyl Sequence Mixer

The first auxiliary layer is based on a Weyl sequence, whose state evolves through an additive recurrence over the integer ring modulo 2^{32}:

$$w_{t+1} = w_t + \gamma \tag{9}$$

where the addition is performed modulo 2^{32}. The constant γ is an odd integer derived from the golden ratio [26]; any odd increment yields a Weyl sequence of full period 2^{32}. For a 32-bit state, γ is determined as

$$\begin{aligned}\gamma &= \mathrm{round}\left(2^{31}(3-\sqrt{5})\right)\\ &= 1640531527\\ &= \texttt{0x61C88647},\end{aligned} \tag{10}$$

which is the two's-complement companion $2^{32} - \gamma'$ of the commonly used golden-ratio constant $\gamma' =$ `0x9E3779B9`; both are odd and therefore generate full-period sequences.

The resulting additive progression provides a deterministic traversal of the 32-bit state space. To diffuse the state bits and avoid directly exposing the underlying linear progression, the output contribution of the Weyl layer is defined as

$$M_{\mathrm{weyl}} = \mathrm{rotl}(w_{t+1}, 7) + \mathrm{rotl}(w_{t+1}, 13), \tag{11}$$

where the addition is performed modulo 2^{32}. The two rotations combine bit positions from different significance regions before the modular addition. In contrast to the carry-less XOR operations of the Xoshiro128** core, the integer addition in this layer introduces carry propagation, thereby providing a complementary mixing mechanism.

Layer 2: The Integer Tent Map

The second auxiliary layer employs a discretized integer Tent map to introduce a stretch-and-fold state evolution [28]. For a 32-bit unsigned state $x_t \in [0, 2^{32} - 1]$, the map is defined as

$$x_{\mathrm{map}} = \begin{cases} 2x_t & \text{if } x_t < 2^{31} \\ 2(2^{32} - 1 - x_t) & \text{if } x_t \geq 2^{31} \end{cases} \tag{12}$$

where the first branch expands the lower half of the state domain, whereas the second branch folds the upper half back into the valid numerical range. This stretching-and-folding operation transforms the input state and contributes sensitivity to the evolution of the auxiliary state.

A bitwise perturbation is subsequently applied to the transformed state:

$$s_{\mathrm{next}} = x_{\mathrm{map}} \oplus (x_{\mathrm{map}} \gg 11) \tag{13}$$

The logical right shift $\gg 11$ shifts the bits of x_{map} by 11 positions, and the XOR with the original state introduces additional dependencies between bit positions and further mixes the state representation.

The output contribution of the Tent map layer is then obtained as

$$M_{\mathrm{tent}} = \mathrm{rotl}(s_{\mathrm{next}}, 5) \oplus \mathrm{rotl}(s_{\mathrm{next}}, 19) \tag{14}$$

where the two rotation constants combine bits originating from different positions of the perturbed state, thereby further diffusing the transformed state before

the final fusion stage. We note that, considered as a bitwise map, this integer Tent step is algebraically affine (degree one); its role is therefore to add an independent, keyed stream to the fusion and to extend the state-sequence period, rather than to provide algebraic nonlinearity. A detailed state-transition and cycle analysis is given in Section 4.7.

2.3 Fusion and Output Generation

The final output is generated by combining the contributions of the three complementary components. Specifically, the scrambled xoshiro128** output is fused with the Weyl and Tent map outputs using bitwise XOR:

$$y_t = R_t \oplus (M_{\text{weyl}} \oplus M_{\text{tent}}) . \tag{15}$$

The proposed PRNG therefore combines three distinct state-generation and mixing mechanisms. The xoshiro128** core provides long-period deterministic state evolution based on XOR, logical shifts, rotations, and modular multiplication. The Weyl layer introduces additive progression and carry-propagating integer mixing, while the integer Tent map contributes stretching, folding, bitwise perturbation, and rotational diffusion. Their outputs are subsequently integrated through the XOR-based fusion operation in (15).

This heterogeneous construction avoids relying exclusively on a single state-transition mechanism. By combining the deterministic evolution of the xoshiro128** core with the additive mixing of the Weyl sequence and the stretch-and-fold transformation of the integer Tent map, the proposed architecture increases the complexity of the overall state-to-output mapping; as shown in Section 4.7, this removes the GF(2)-linear output structure of Xoshiro128**, whereas the statistical quality of the output is already provided by Xoshiro128** itself.

3 Hardware Implementation and Verification Methodology

This section describes the hardware architecture, implementation setup, and verification procedure used to evaluate the proposed composite chaotic PRNG. To estimate the standalone physical characteristics of the core, the RTL design was evaluated using an OOC synthesis and implementation flow, which reduces the impact of system-level SoC wrapper logic and interconnect overhead. The target platform is the Xilinx Zynq xa7z010clg225-1I device, and the design was compiled using Vivado HLS and the Vivado Design Suite. Under this setup, the reported resource usage, timing margins, and power figures reflect the behavior of the core implementation rather than the full SoC integration. An FPGA was chosen rather than an ASIC because the target is a reconfigurable SoC platform, on which the core is used as a soft, DSP-free AXI4 component that can be updated and integrated without fabrication. An ASIC implementation would offer lower cost per unit, higher clock frequency, and lower power, as low-area and low-power VLSI architectures reported for other compute-intensive workloads illustrate [29, 30], but requires high non-recurring engineering cost and long time to deployment and cannot be modified after fabrication; a comparison of both realizations for a chaotic generator is given in [22]. Since we have no ASIC implementation data, we report FPGA results only.

As shown in Fig. 1, the core is configured through a dedicated AXI4-Lite slave interface, which exposes the target stream length, eight 32-bit key words, and a warmup-cycle counter as memory-mapped registers. Four of the eight key words seed the individual Xoshiro128** state words $s_0, \ldots, s_3$, one seeds the Weyl accumulator w_0, and one seeds the Tent-map register x_0, while the remaining two words are currently reserved, allowing future seed-mixing extensions without modifying the register map. The warmup-cycle register is present in the register map but is not activated in the present version; all results reported in this work are therefore obtained with a warmup of zero cycles, and the released testbench and the statistical datasets use the same setting.

Before physical implementation, the synthesized RTL was checked using Vivado HLS C/RTL co-simulation. In this step, the C reference model of the composite recurrence described in (1)–(15) was used as the reference implementation. An automatically generated testbench applied identical seed and stream-length inputs to both the C model and the RTL description, and the resulting 32-bit output sequences were compared on a cycle-by-cycle basis. The co-simulation completed with a `Pass` status, indicating bit-exact agreement between the reference model and the synthesized datapath before place-and-route. This confirms that the pipelining and array-partitioning steps described in Section 3.1 preserve the intended behavior of (1)–(15).

3.1 C-to-RTL Microarchitectural Optimization

The multi-domain chaotic and linear recurrence algorithms were translated from high-level C++ descriptions to a register-transfer-level (RTL) netlist using

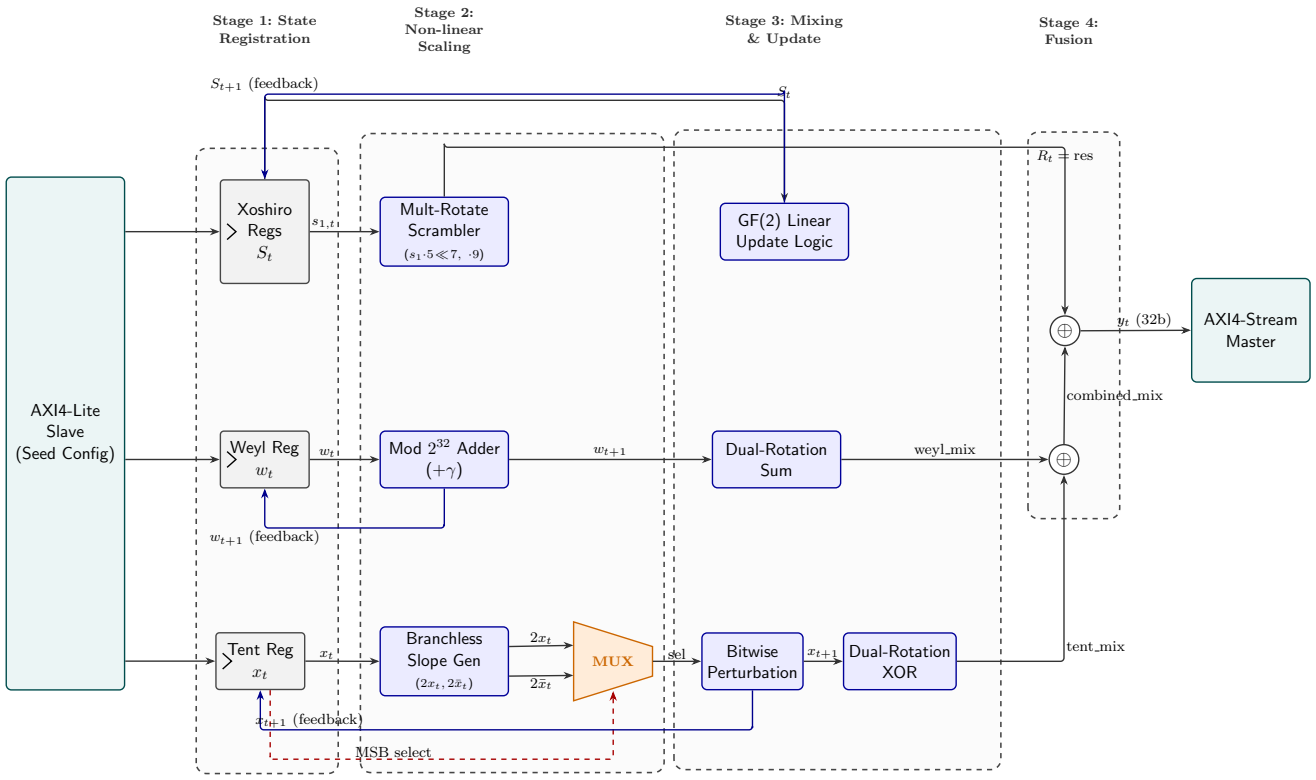


Fig. 1. Hardware architecture of the proposed DSP-free hybrid PRNG core: the AXI4 interfaces, the Xoshiro128** backbone, the Weyl and Tent mixing layers with their state-feedback paths, and the XOR fusion. The four stages are the logical stages of the datapath.

Vivado HLS within an OOC design flow. To support streaming operation, optimization directives were applied to increase instruction-level parallelism, resulting in an initiation interval of 1 ($II = 1$) for the main processing loop. During high-level synthesis, the scheduling engine estimated a logic propagation delay of 4.171 ns under a target clock period of 5.00 ns (200 MHz). Because pre-layout HLS estimates are based on simplified wireload models, they do not fully capture interconnect overhead, net fan-out, and parasitic RC delays introduced during physical implementation. For this reason, the clock period used during placement and routing was set to 5.500 ns (181.82 MHz). Under this operating point, the implemented core met the timing constraints, reporting a worst negative slack (WNS) of 0.068 ns and a worst hold slack (WHS) of 0.132 ns.

3.1.1 Multiplierless Shifter-and-Adder Realization

To reduce the area and dynamic power overhead associated with dedicated hardware arithmetic macros, the proposed PRNG avoids variable multiplications that would normally require DSP slices. The mixing and scrambling networks are formulated using compile-time integer constants and bit-level permutations. In the output generation stage, the non-linear scrambling function $R_t = \mathrm{rotl}(s_1 \times 5, 7) \times 9$ uses odd constants in modulo-2^{32} arithmetic. In the synthesized design, these constant operations are represented as low-latency shifter-and-adder logic rather than generic multiplier blocks. For example, the product by 5 can be written as

$$s_1 \times 5 = (s_1 \ll 2) + s_1 \tag{16}$$

and the subsequent scaling by 9 can be realized in a similar way using shift-and-add logic. Since these operations use fixed coefficients rather than runtime variables, the resulting RTL netlist is implemented using standard look-up tables (LUTs) and routing resources. In the reported implementation of this work, the core runs at 181.82 MHz and uses no DSP48E1 blocks or block RAM (BRAM) tiles.

3.1.2 Constant-Time Branchless Chaotic Mapping

The hardware realization of the discrete chaotic integer Tent map involves a piecewise conditional boundary in its mathematical definition. In conventional hardware mappings, such conditional transitions are typically

synthesized into control-flow branching logic governed by state flags. In software, such a transition is a data-dependent branch. In the proposed datapath, a branchless bit-masking and multiplexing structure is used instead, so that every Tent update takes exactly one clock cycle regardless of the state value.

As illustrated in Fig. 1, the high-level synthesis flow maps the conditional piecewise behavior into parallel combinational datapaths in the RTL design. At each active clock cycle, the core evaluates both candidate expressions: the rising slope ($2x_t$) is computed using a hardwired left shift, while the falling slope ($2(2^{32}-1-x_t)$) is computed using bitwise inversion and shifting. Instead of using control-flow branching, the final path is selected by a 32-bit 2-to-1 multiplexer. The multiplexer select line is connected to the most significant bit (bit 31) of the chaotic register. Both candidates are therefore computed in every cycle, and the update completes within the same single cycle for every state value.

3.1.3 Memory Bottleneck Mitigation via Array Partitioning

To support a sustained throughput of one 32-bit pseudorandom word per clock cycle ($II = 1$), the design requires concurrent access to several state elements. In many implementations, compound state arrays are mapped to embedded BRAM macros. However, sequential memory structures can limit parallel access because dual-port BRAMs can service only a limited number of operations per cycle. Since the synchronized state updates of the Xoshiro128** core (S_0–S_3), the additive Weyl sequence accumulator (w_t), and the perturbation stages of the Tent map involve multiple read and write operations within the same cycle, a purely sequential memory layout would introduce stalls.

To address this limitation, array partitioning was applied during microarchitectural optimization. This directive separates the multi-word internal state vectors into independent 32-bit register elements implemented with standard D-type flip-flops (FDRE) in the logic slices. By avoiding shared memory access and address arbitration, the state elements can be accessed in parallel within the same clock cycle. In the reported implementation, this approach resulted in zero BRAM tiles and 1071 flip-flops. The single-cycle initiation interval is confirmed after implementation by the functional simulation of the routed netlist used for the power analysis (Section 3.2), in which 20,000 output words were transferred in 20,000 consecutive clock cycles with `TREADY` continuously asserted, every word being bit-exact with the RTL simulation. Fig. 2 illustrates the AXI4-Stream handshake for a 20-word stream with the zero key: Fig. 2a shows the beginning of the stream, where a word is transferred in every cycle in which `TVALID` and `TREADY` are both high and is held while `TREADY` is low, and Fig. 2b shows the end of the stream, where `TLAST` is asserted on the final word.

Table 1. Post-implementation resource, timing, and power metrics under out-of-context (OOC) constraints

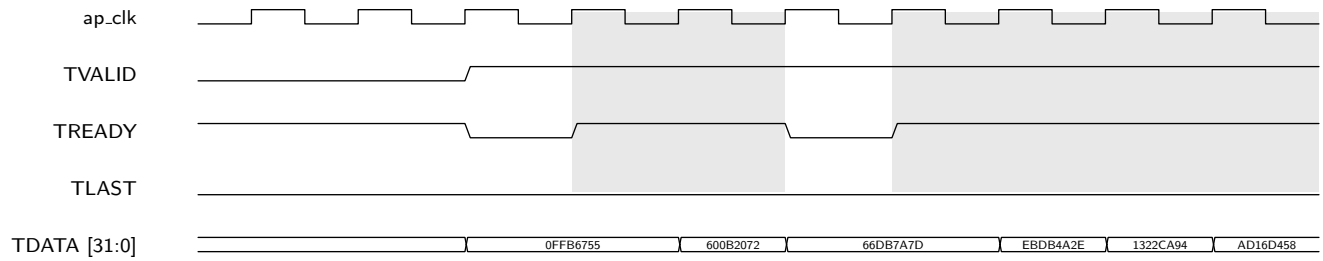
Parameter	**Value**	**Ratio**
Logic Resources (Routed)		
Device Fabric	Zynq xa7z010clg225-1I	
Slice LUTs	934	5.31%
Slice Registers	1071	3.04%
DSP48E1 Primitives	0	0.00%
BRAM_18K Tiles	0	0.00%
Timing (ap_clk)		
Clock Period Constraint	5.500 ns	—
Nominal Frequency	181.82 MHz	—
WNS	0.068 ns	—
WHS	0.132 ns	—
Power (SAIF-Driven)†		
Dynamic Power	0.029 W	24.37%
Static Power	0.090 W	75.63%
Total On-Chip Power	**0.119 W**	**100.0%**

†Zero-delay functional simulation at full streaming rate; this excludes glitch activity and is a lower bound. A post-route timing simulation, used as a sensitivity check, gives a dynamic power of 0.042 W.

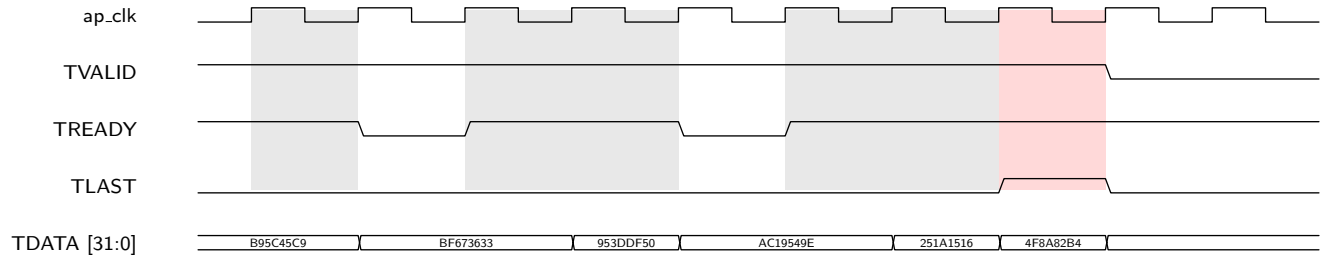
3.2 Physical Post-Implementation Metrics

The core was implemented out of context with a scripted Vivado 2019.1 flow: HLS C-synthesis with a 5.0 ns target, followed by synthesis, placement, physical optimization, and routing with a 5.5 ns clock constraint on the Xilinx Zynq xa7z010clg225-1I. The same flow is used for every variant of the ablation study in Section 4.7 and is included in the released package, so all reported hardware numbers can be regenerated. The resource, timing, and power metrics of the routed netlist are summarized in Table 1. The entire multi-domain mixing logic and its streaming wrapper require 934 Slice LUTs and 1071 Slice Registers; these counts include the AXI4-Lite configuration interface, the AXI4-Stream output interface, and the control logic generated by HLS, not only the generator datapath. The routed core uses no DSP48E1 slices and no BRAM tiles: all computation is realized in standard logic slices.

Timing closure is achieved at a nominal operating frequency of 181.82 MHz under a 5.500 ns hardware clock constraint. The routed netlist meets timing with a positive worst negative slack (WNS) of 0.068 ns and



(a) Beginning of the stream: the first word is held while `TREADY` is low and one word is transferred in every cycle in which `TVALID` and `TREADY` are both high.



(b) End of the stream: `TLAST` is asserted only on the final word (highlighted), which terminates the AXI4-Stream frame.

Fig. 2. AXI4-Stream output handshake of the proposed PRNG core for a 20-word stream with the zero key. The data words are those of the C reference model (words 1–6 in (a) and 15–20 in (b)); the `TREADY` pattern is illustrative. Shaded cycles mark transfers, in which `TVALID` and `TREADY` are both high.

a positive worst hold slack (WHS) of 0.132 ns. Based on the achieved single-cycle loop execution ($II = 1$), the sustained physical throughput of the core engine is evaluated as a direct function of the operating frequency and the output bit-width, with TREADY continuously asserted; the main loop has an iteration latency of 3 clock cycles, so the first word appears 3 cycles after the stream starts and one word follows in every subsequent cycle. The throughput is calculated as:

$$\text{Throughput} = \frac{181.82\ \text{MHz}}{1\ \text{cycle/output}} \times 32\ \text{bits} = 5.818\ \text{Gbps} \tag{17}$$

Dynamic power was estimated with Vivado `report_power` from the switching activity (SAIF) of a zero-delay functional simulation of the routed netlist. The simulation streams 20,000 output words at the full rate (TREADY continuously asserted) inside a SAIF window that excludes reset and configuration; every output word was verified bit-exact against the RTL simulation, 100% of the design nets were matched, and three initialization keys gave identical results at the 1 mW resolution of the tool. Because a zero-delay simulation does not capture glitch activity, the resulting dynamic power of 0.029 W is a lower bound. As a sensitivity check, a post-route timing simulation with SDF back-annotation gives 0.042 W; it is not used as the reported value because the gate-level simulation model produced incorrect values in about 6% of the output words. Both flows are part of the released package. The total on-chip power is 0.119 W, of which 0.090 W is the static power of the device.

4 Experimental Results and Discussion

This section presents the experimental assessment of the proposed PRNG. The test sequences were generated with the C reference model, which the HLS C/RTL co-simulation verified to be bit-exact with the RTL; the results therefore apply to the output of the hardware core.

4.1 Statistical Analysis

To empirically validate the randomness of the generated sequences, we employed three widely used statistical test suites: NIST SP 800-22 [31], Dieharder [32], and TestU01 [33].

4.1.1 NIST SP 800-22 Test Results

The statistical test suite was used to assess the statistical behavior of the generated sequences. The evaluation was performed on 500 independent binary

sequences, each comprising 1×10^6 bits, at a significance level of $\alpha = 0.01$. Under this configuration, the exact minimum passing proportion is 489 out of 500 sequences, and the uniformity P-value must exceed 0.0001. For the random excursions and random excursions variant tests, the evaluated sample size was reduced to 323 sequences based on valid random walk cycles, requiring a minimum of 315 passing sequences. The lowest passing counts actually observed were 490/500 and 318/323, so every sub-test passes.

As summarized in Table 2, the proposed hybrid PRNG passed all 15 statistical categories in the reported evaluation. For tests comprising multiple sub-evaluations (cumulative sums, non-overlapping template, random excursions, random excursions variant, and serial), the table reports the average P-value and average passing proportion across the corresponding sub-tests, denoted by an asterisk. Individual sub-test results are provided in the supplementary log files. The linear complexity and binary matrix rank tests also showed consistently high passing rates. Plain Xoshiro128** is already known to pass the TestU01 BigCrush battery [27]; the added layers are therefore not claimed to improve statistical quality. Their effect on the linear structure of the output is analyzed separately in Section 4.7.

Table 2. NIST SP 800-22 statistical test suite results for 500 Independent sequences (Length: 1×10^6 bits, $\alpha = 0.01$)

Statistical Test	P-Value	Proportion	Result
Frequency	0.9781	0.9900	PASS
Block Frequency	0.5102	0.9860	PASS
Cumulative Sums*	0.2897	0.9910	PASS
Runs	0.4083	0.9880	PASS
Longest Run	0.1167	0.9840	PASS
Rank	0.3041	0.9800	PASS
FFT	0.8237	0.9880	PASS
Non-Overlapping Template*	0.4998	0.9903	PASS
Overlapping Template	0.1326	0.9920	PASS
Universal	0.4083	0.9860	PASS
Approximate Entropy	0.5708	0.9900	PASS
Random Excursions*	0.4453	0.9899	PASS
Random Excursions Variant*	0.5155	0.9904	PASS
Serial*	0.8051	0.9910	PASS
Linear Complexity	0.7753	0.9900	PASS

Note: * The average values of multiple sub-tests.

4.1.2 Dieharder Test Battery Analysis

The Dieharder battery (version 3.31.1) [32], which includes 114 statistical tests, was applied to the generated sequences. The test procedure was repeated over three independent runs with different seeds to reduce the effect of seed-dependent artifacts. In a few cases, p-values were close to the rejection threshold, but the corresponding follow-up runs returned passing results. Overall, the generator passed all 114 tests. A summary of the results, grouped by test family, is given in Table 3.

Table 3. Summary of final Dieharder statistical test battery results

Dieharder Test Category	**Stats.**	**Result**
Diehard Subset (birthdays, craps, squeeze, ...)	19	PASS
STS Subset (monobit, runs, serial)	32	PASS
RGB Subset (bitdist, min. distance, lagged sum)	54	PASS
DAB & Marsaglia Subsets (filltree, gcd)	9	PASS
Overall Evaluation Suite	**114**	**PASS**

4.1.3 TestU01 Statistical Validation

The TestU01 framework (version 1.2.3) [33] was used to assess the statistical behavior of the generated sequences beyond the standard test suites. The proposed architecture was evaluated using the SmallCrush, Crush, and BigCrush batteries. The generator was connected through the `unif01_CreateExternGenBits` interface, which consumes the raw 32-bit output words without any post-processing; the seeds used for every battery are listed in the released package.

The data requirements of each battery depend on the number and dimensionality of its internal tests. Summing the pseudorandom draws consumed across all sub-tests of each battery, SmallCrush processed at least $2^{28.8}$ bits ($\approx$ 0.06 GB) across 15 statistics, and Crush processed at least $2^{40.6}$ bits ($\approx$ 214 GB) across 144 statistics, in 5.6 s and 24 min 36 s of CPU time, respectively. BigCrush evaluated 160 statistics and was used to examine the longer-range behavior of the generator.

In the reported experiments, the generator passed all 160 statistics in the BigCrush battery [33], consuming at least $2^{41.9}$ bits ($\approx$ 502 GB of streaming data) in 2 hours, 53 minutes, and 12 seconds of CPU time. The observed p-values remained within the acceptable range of $[0.001, 0.999]$ for the reported runs. As summarized in Table 4, the results indicate that the proposed design maintains consistent statistical behavior under the tested conditions.

Table 4. Summary of TestU01 empirical battery results

Test Battery	Stats.	Data Volume	Result
SmallCrush	15	$\geq 2^{28.8}$ (0.06 GB)	PASS
Crush	144	$\geq 2^{40.6}$ (214 GB)	PASS
BigCrush	160	$\geq 2^{41.9}$ (502 GB)	PASS

Note: Data volume (bits, with GB in parentheses) is a lower-bound estimate obtained by summing the number of pseudorandom draws ($N \times n$) reported for each sub-test in the TestU01 output logs. All batteries reported zero failures.

4.2 Key Space Analysis

The initialization-input space is one characteristic of a pseudorandom generator. In the proposed implementation, six active 32-bit initialization parameters provide a nominal input space of 2^{192} possible bit patterns before excluding invalid or equivalent initial states. The number of distinguishable initializations is smaller, as discussed below.

The proposed hardware interface exposes an 8-word AXI-Lite key input (256 bits total) to allow future extension; however, the initialization key used by the core consists of six independent 32-bit parameters: four state registers (s_0, s_1, s_2, s_3) that initialize the xoshiro128** backbone, one seed (w_0) for the Weyl sequence accumulator, and one initial state (t_0) for the discretized Tent map, i.e., the initial value x_0 of the Tent register. The remaining two key words are reserved for future extension of the mixing architecture and do not currently affect the internal state transition.

Since each of the six active parameters is an independent 32-bit unsigned integer, the nominal key space is given by

$$K_{space} = 2^{32\times 6} = 2^{192} \tag{18}$$

The seeding logic remaps the forbidden all-zero Xoshiro128** state and the special values $w_0 = 0$ and $t_0 = 0$ to fixed non-zero defaults, and, because $\mathrm{Tent}(x) = \mathrm{Tent}(\sim x)$, the states t_0 and $\sim t_0$ produce identical sequences; the number of distinguishable initializations is therefore reduced to at most about 2^{191}. The key space is obtained entirely from integer-domain state variables without floating-point arithmetic, which differs from continuous chaotic PRNGs that often require higher-precision computation to achieve comparable parameter space.

4.3 Initialization-State Sensitivity Analysis

To assess the sensitivity of the generator to small perturbations of the initialization state, the first 40 output words were compared for two initialization states that differ by one bit in the least significant register of the xoshiro128** state. As shown in Fig. 3, the first output word is identical, because it depends only on s_1 before the first state update, and the next few words differ in only part of their bits while the flipped bit spreads through the Xoshiro128** state; from the sixth word on, the two sequences are unrelated.

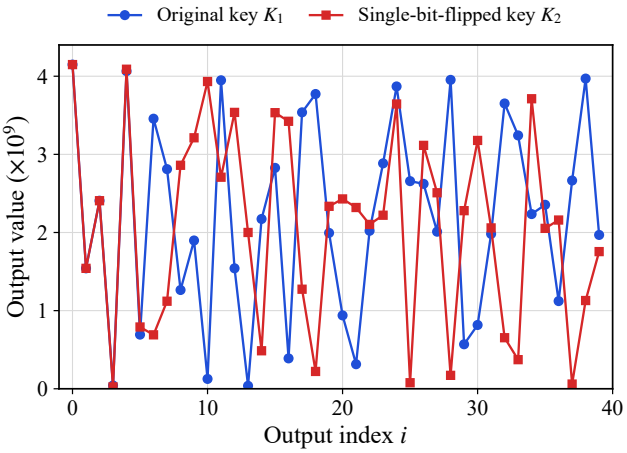


Fig. 3. Initialization-state sensitivity: the first 40 output words generated from an initialization state K_1 and a single-bit-flipped state K_2.

4.4 Output Sensitivity Analysis Using NPCR and UACI

To quantitatively assess initialization-state sensitivity beyond visual inspection, the number of changing pixel rate (NPCR) and the unified average changing intensity (UACI) metrics are computed on the 32-bit output streams generated from two initialization states differing by exactly one bit in the least significant bit of the s_0 register of the xoshiro128** core.

Given two output sequences S_1 and S_2 of length $N = 65536$ words, NPCR and UACI are defined as [7]:

$$NPCR = \frac{1}{N}\sum_{n=0}^{N-1} W(n) \times 100\% \tag{19}$$

where $W(n) = 1$ if $S_1(n) \neq S_2(n)$, and $W(n) = 0$ otherwise.

$$UACI = \frac{1}{N}\sum_{n=0}^{N-1} \frac{|S_1(n) - S_2(n)|}{2^{32} - 1} \times 100\% \tag{20}$$

where $S_1(n)$ and $S_2(n)$ denote the n-th 32-bit output words of the two sequences. The measured NPCR of 99.998474% and UACI of 33.242817% indicate that a single-bit perturbation of the initialization state produces substantial differences between the two output sequences. For independent uniform 32-bit words the expected NPCR is $100(1 - 2^{-32})\%$ and the expected UACI is 33.333% with a sampling standard error of 0.092% at $N = 65536$; the measured UACI is within about one standard error ($z = -0.98$). The single pair of equal words responsible for the NPCR not reaching 100% occurs at the first output word, which depends only on s_1 before the state update and is therefore identical under a flip of the s_0 bit; this is a property of the Xoshiro128** output function, not of the added layers. Over 1000 random initialization states, each with a random single-bit flip in the Xoshiro128** state, the mean UACI was 33.328% (standard deviation 0.094%), compared with 33.333% (0.095%) for pairs of independent sequences from an operating-system random-number generator used as a baseline. Moreover, because the Weyl and Tent states do not depend on the Xoshiro128** state, the output difference of the full design under such a flip equals that of plain Xoshiro128** exactly (0 mismatching words out of 13,107,200 over 200 pairs); the sensitivity measured here therefore originates entirely from the Xoshiro128** engine. These results are summarized in Table 5, which reports the measured sensitivity metrics for the tested output sequences.

Table 5. NPCR and UACI results for the proposed PRNG ($N = 65536$ words)

Metric	Measured Value	Ideal Value
NPCR	99.998474%	$100(1 - 2^{-32})\%$
UACI	33.242817%	≈33.3%

4.5 Information Entropy

Information entropy provides one measure of uncertainty in the generated output sequence. For an ideal random source producing 8-bit symbols (byte values in the range $[0, 255]$), the theoretical maximum entropy is 8 bits per symbol, computed using Shannon's entropy formula as:

$$H(m) = -\sum_{i=0}^{255} p(m_i) \log_2 p(m_i) \tag{21}$$

where $p(m_i)$ denotes the empirical probability of occurrence of byte value m_i within the generated sequence, obtained by decomposing the 32-bit output words into individual bytes. For a finite sample of 262,144 bytes (65,536 words), the expected Shannon entropy of a truly uniform source is approximately 7.999298 bits/byte rather than the asymptotic 8; the measured values are therefore consistent with an ideal uniform distribution within the finite-sample bound.

The information entropy of the byte-decomposed output stream was computed for both the original and single-bit-flipped key sequences used in the NPCR/UACI evaluation. The obtained entropy values of $H_{normal} = 7.999357$ bits/byte and $H_{flipped} = 7.999348$ bits/byte are close to the finite-sample expectation of 7.999298 bits/byte noted above. These values suggest a nearly uniform byte distribution in the reported data, as also reflected by the histogram in Fig. 4.

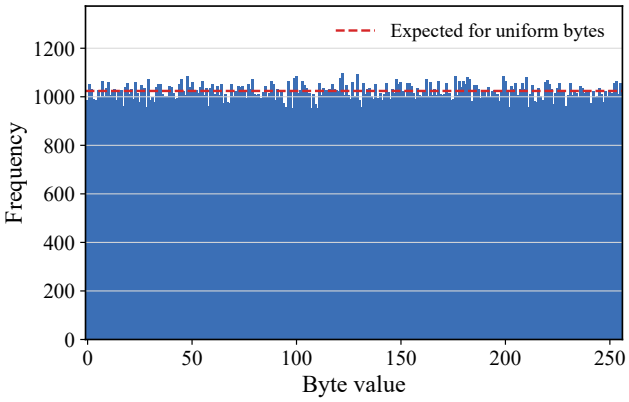


Fig. 4. Byte-value histogram of the output sequence generated using the original key K_1, illustrating a near-uniform distribution across all 256 possible byte values.

4.6 Autocorrelation Analysis

Autocorrelation analysis measures the dependence of the output sequence on delayed versions of itself. For a random sequence, the autocorrelation function is expected to be close to a delta function, with near-zero correlation at non-zero lags, which indicates the absence of strong periodic structure [7].

The normalized autocorrelation function (ACF) of the generated output sequence was computed over $N = 65536$ words using the fast Fourier transform (FFT) method, defined as:

$$ACF(\tau) = \frac{\sum_n (x(n) - \bar{x})(x(n+\tau) - \bar{x})}{\sum_n (x(n) - \bar{x})^2} \tag{22}$$

where $x(n)$ denotes the n-th output sample, $\bar{x}$ its mean, and τ the time lag.

By definition, the ACF equals unity at zero lag; Fig. 5 therefore reports only the non-zero lags $\tau \in [1, 200]$, over which the autocorrelation remains close to zero. The maximum absolute autocorrelation value observed was 0.010682, with a mean absolute value of 0.003335 across all tested lags. Thirteen of the 200 tested lags exceeded the 95% white-noise confidence bound ($\pm 1.96/\sqrt{N}$), which is consistent with the false-positive rate expected for a finite random sample. These results indicate limited long-range correlation in the reported sequence.

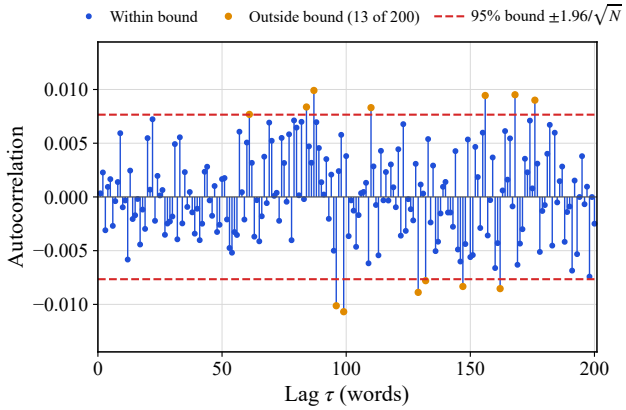


Fig. 5. Autocorrelation function of the generated PRNG output sequence for non-zero lags $\tau \in [1, 200]$; the zero-lag value (unity by definition) is omitted. All values remain close to zero, consistent with an ideal delta-like autocorrelation.

4.7 Ablation Study and Structural Analysis

To clarify what each mixing layer contributes, we evaluated four variants with the same scripted HLS and Vivado out-of-context flow as in Table 1 (xa7z010clg225-1I, 5.5 ns constraint): V0, plain Xoshiro128** only; V1, Xoshiro128** with the Weyl layer; V2, Xoshiro128** with the integer Tent layer; and V3, the full proposed design. All four variants sustain $II = 1$ and therefore the same 5.818 Gbps throughput; they differ only in logic cost, timing margin, dynamic power, and structural behavior. Their hardware metrics are summarized in Table 6.

The power columns of Table 6 are obtained in the same way as for Table 1. P_{dyn} is the zero-delay SAIF estimate at full streaming rate (20,000 output words, TREADY continuously asserted); for every variant, three initialization keys gave identical results at the 1 mW resolution of the tool, every output word matched the RTL simulation, and 100% of the design nets were matched. Its clock and logic-and-signal components are rounded separately, so their sum may differ from P_{dyn} by 1 mW. The timing-simulation column is the glitch sensitivity check described in Section 3.2, in which the gate-level simulation model produced incorrect values in up to about 6% of the output words. In this out-of-context flow the clock net is ideal: it is neither routed nor buffered in any variant, so the clock component is a placement-dependent model estimate rather than the power of a physical clock tree, and it does not follow the flip-flop count (V3 has the most flip-flops but the lowest clock estimate). The data-dependent cost of each layer is therefore the logic-and-signal component, which grows with every added layer.

We emphasize that plain Xoshiro128** (V0) is already known to pass the TestU01 BigCrush battery [27], as does the full design (V3) in Section 4; we therefore make no claim that the layers improve statistical test results, and the long statistical batteries were run on the full design only. The value of the layers is structural, as analyzed next, and it comes at a modest hardware cost: roughly 249 additional LUTs (36%) and 10 mW of additional logic-and-signal power over V0 in the zero-delay estimate. Including glitch activity, the dynamic power of the full design rises by 45% (29 to 42 mW), compared with 11% for V0, so the added carry-propagating and multiplexing logic also adds glitch power. The total dynamic power of V3 is lower than that of V1 and V2 only because of the clock estimate explained above and should not be read as a power saving of the added layers.

4.7.1 Linear State Recovery and a Keyless Distinguisher

The output scrambler $R_t = \mathrm{rotl}(s_1 \times 5, 7) \times 9$ is a bijection, and the Xoshiro128** state transition is linear over GF(2). Consequently, four consecutive outputs of plain Xoshiro128** yield 128 linear equations that pin the internal state: in our experiments the full 128-bit state was recovered from four outputs in 2000 of 2000 trials, at about 28 μs per recovery, with no knowledge of any key. For the full design (V3), the same linear solve succeeds only once the 64-bit auxiliary key (w_0, t_0) has been stripped, which makes the attack a guess-and-determine attack [34]; each candidate (w_0, t_0) costs about 27 μs, so an exhaustive 2^{64} search on a single core would take on the order of 1.6×10^7 years. The best

Table 6. Layer ablation under the same scripted HLS/Vivado out-of-context flow (xa7z010clg225-1I, 5.5 ns constraint). All variants achieve $II = 1$ and 5.818 Gbps; DSP and BRAM usage is zero. P_{dyn} is the zero-delay estimate, given with its clock and logic-and-signal components; the last power column includes glitch activity (timing simulation). See the text for the method.

Variant	**LUT**	**FF**	**WNS (ns)**	**Clock (mW)**	**Logic+sig. (mW)**	P_{dyn} **(mW)**	P_{dyn}**, timing (mW)**	**Mbps/ LUT**
V0 (Xoshiro128** only)	685	847	0.190	18	10	28	31	8.49
V1 (+ Weyl)	853	975	0.292	24	17	40	48	6.82
V2 (+ Tent)	794	943	0.173	22	15	37	45	7.33
V3 (full, proposed)	934	1071	0.068	10	20	29	42	6.23

shortcut we found lowers this to about 2^{63} (because t_0 and $\sim t_0$ produce identical streams, see below) and to about $2^{61.3}$ for the $\approx 91\%$ of keys whose t_0 lies on the dominant Tent cycle. We found no cheaper route, because the low output bits carry no usable linear information about w_0 (maximum observed bias 0.0027 against a statistical noise level of 0.0011 over 2×10^5 random keys). These are the measured costs of the best attack we identified.

Applying the same attack to the intermediate variants shows what each layer adds on its own. With the Weyl layer alone (V1), the linear solve succeeds after guessing the 32-bit w_0, that is, after at most 2^{32} guesses; with the Tent layer alone (V2), it succeeds after guessing t_0, that is, after at most 2^{31} guesses (about $2^{29.3}$ for keys on the dominant Tent cycle). At the measured cost per guess, either search takes less than two days on a single core. In the full design the two keys must be guessed jointly, so the two costs multiply to about 2^{63}: the two layers are effective in combination rather than individually.

As a complementary structural test, we applied a keyless linear-complexity (LC) distinguisher, computed with the Berlekamp–Massey algorithm [35], after undoing the public scrambler. Plain Xoshiro128** then exhibits linear complexity of exactly 128 (all 80 of 80 streams flagged with LC $\leq$ 130), directly revealing its order-128 linear recurrence. The full design shows LC $\approx 300 \approx N/2$ (none of the 80 streams flagged), so the distinguisher that trivially flags plain Xoshiro128** does not flag V3. The same holds for V1 and V2 (mean LC ≈ 300, none of 80 streams flagged), so either layer alone removes this linear signature.

4.7.2 Algebraic Degree of the Integer Tent Map

An exhaustive algebraic-normal-form analysis over reduced word widths (8 to 16 bits), and on a 12-bit subcube of the full 32-bit state, shows that the integer Tent step is affine (algebraic degree 1) at every width and for every iteration count tested. In the complete V3 output the mapping is affine in the Tent key t_0 only; it remains non-affine in w_0 (carry-propagating addition) and in the Xoshiro128** state (modular multiplication). The integer Tent layer therefore does not add algebraic nonlinearity. Its role is to contribute an independent, keyed stream to the XOR fusion and to extend the state-sequence period, rather than to harden the generator algebraically.

4.7.3 Cycle Structure and State-Sequence Period

Functional-graph analyses of digitized Tent maps in fixed-precision integer arithmetic have been reported [36]; here we apply such an analysis exhaustively to the specific map used in this work, including its xorshift perturbation. Iterated over all 2^{32} states, the integer Tent map has image size exactly 2^{31} and satisfies $\mathrm{Tent}(x) = \mathrm{Tent}(\sim x)$ for every x, so t_0 and its bitwise complement generate identical sequences. The state space decomposes into 12 cycles, and any state reaches a cycle in at most one step. The dominant cycles have length $2^{29.276}$ (three of them, covering $\approx 90.8\%$ of seeds), followed by $2^{26.954}$ ($\approx 6.1\%$) and progressively shorter cycles down to length 15, plus one fixed point. The only fixed point is state 0; since the seeding logic remaps $t_0 = 0$, it is reached only from $t_0 =$ `0xFFFFFFFF`, for which the Tent stream is constantly zero, a single weak value out of 2^{32}. Combining this with the Xoshiro128** and Weyl periods, the state-sequence period, computed exactly as the least common multiple of the component periods, is $2^{128} - 1$ for V0, $(2^{128} - 1) \cdot 2^{32} \approx 2^{160}$ for V1, from 2^{128} up to about $2^{153.4}$ for V2, and for the full design ranges from about 2^{160} up to about $2^{185.4}$, reaching $2^{185.4}$ for roughly 97% of seeds. Finally, a single-bit seed-flip changes on average 0.497 of the output bits for the 128 Xoshiro128** seed bits (ideal 0.5) and 0.484 for the 32 Tent bits, while the 32 Weyl bits change 0.125 of the output bits, reflecting that w_0 enters the fusion additively. All ablation variants, power measurements,

attack experiments, and cycle censuses reported here are reproducible from the deposited package.

4.8 Comparison with State-of-the-Art Architectures

Table 7 places the proposed design in the context of recent FPGA-based chaotic PRNGs. Rather than ranking the designs by a single figure, the table is organized to make the comparison conditions explicit, and the proposed generator is reported both as the plain xoshiro128** backbone and as the full hybrid so that the cost of the added layers is visible.

On the limits of cross-design comparison. A direct numerical comparison between these works is not strictly meaningful, and we do not claim one. The designs use different FPGA families (Zynq-7000, Artix-7, Kintex-7, and an Intel/Altera Cyclone V), different arithmetic formats (from 8-bit fixed point to 64-bit floating point), different operating frequencies, and different power-reporting conventions. Logic utilization in particular depends strongly on these choices: a 64-bit floating-point datapath [10] necessarily uses far more look-up tables than an integer datapath, and logic reported for an Altera device [19] is counted in adaptive logic modules that do not map one-to-one to Xilinx LUTs, so its LUT and Mbps/LUT entries are left blank. This limitation is not specific to our study; for example, [7] notes in its own comparison that contrasting resource consumption or frequency across different FPGA families and vendors is not appropriate, since the underlying logic elements differ. The entries in Table 7 should therefore be read together with the arithmetic format, device family, and the conditions discussed here, not as a single ranking.

Resource and throughput trade-offs. With that caveat, two broad groups are visible. High-dimensional maps and structure-varying networks [7, 10, 19] emphasize dynamical complexity; they report comparatively low throughput (0.007–1.045 Gbps) and use dedicated DSP slices (20–52). At the other end, designs that target multi-gigabit rates rely on parallel hardware multipliers: the Tinkerbell generator in [2] reaches 2.33 Gbps with 16 DSP slices, and the discrete hyper-chaotic system in [25] reaches 10.28 Gbps but uses 224 DSP slices. In an FPGA, DSP slices are a limited hard resource that is often shared with other system functions, so heavy DSP use can reduce portability to smaller devices.

Where the proposed design sits. The proposed generator does not achieve the highest throughput or the smallest area in the table. Its distinguishing property is that it uses *no* DSP slices and *no* Block RAM, whereas every other entry uses between 16 and 224 DSP slices; the entire datapath is realized in standard logic slices using branchless integer arithmetic. At 5.818 Gbps it is the second fastest entry after the DSP-heavy design of [25], and among the integer/fixed-point designs it offers a competitive normalized throughput of 6.23 Mbps per LUT (second to the 227-LUT Tinkerbell design of [2], which however is DSP-based and validated only with SmallCrush). Of the other works, only [2] reports dynamic power, and the proposed design has a lower energy per bit (5.0 pJ/bit as a zero-delay lower bound and 7.2 pJ/bit including glitch activity, versus 19.3 pJ/bit for [2]); we restrict this comparison to dynamic power because the other works report total on-chip power (0.438 W for [10]; 0.668 W at 10 Gbps for [25]) or no power figure; total power includes device-wide static leakage and is not a per-design metric. A power comparison with the remaining works is therefore not possible, and we do not report one.

Statistical-validation depth. The works that reach multi-gigabit throughput [2, 25] report TestU01 only at the SmallCrush level, so the behavior of those ultra-fast generators under the longer Crush and BigCrush batteries is not established in the respective papers. The proposed design is validated with NIST SP 800-22, the full Dieharder battery, and the complete TestU01 BigCrush battery (160 statistics; full subtest logs are provided as supplementary material). We note that plain xoshiro128** is already known to pass BigCrush [27], and the xoshiro-only row is included in Table 7 for the resource and power comparison; it has the same throughput as the full design because both produce one 32-bit word per cycle at the same frequency, and its energy per bit is 4.8 pJ/bit as a zero-delay lower bound and 5.3 pJ/bit including glitch activity; the value of the added layers is analyzed separately in Section 4.7 and is structural rather than statistical.

Summary. Taken together, the proposed design is best described not as the fastest or smallest generator, but as a DSP-free, BRAM-free integer core that reaches multi-gigabit throughput in standard logic while being validated to the depth of the full BigCrush battery. This combination of zero hard-macro usage, competitive normalized throughput and energy, and deep statistical validation is the practical trade-off that the design targets for resource-constrained embedded systems, rather than a record in any single metric.

Table 7. Condition-aware comparison with recent FPGA-based chaotic PRNGs. Numbers are reproduced from the cited works; normalized metrics are derived from them. The designs use different FPGA families, arithmetic formats (in parentheses), and power-reporting conventions, so the entries are *not* directly comparable and should be read together with the stated conditions (see the text). The last numeric column gives the normalized throughput with the dynamic energy per bit in parentheses; – marks values that cannot be derived from the cited work. The proposed design additionally uses no Block RAM.

Design	Algorithm (arith.)	FPGA	Freq. (MHz)	LUT	DSP	Thr. (Gbps)	Mbps/ LUT (pJ/bit)	Stat. batteries
[10]	2D-SPCCM (fp-64)	Zynq-7000	108.3	16,233	52	0.017	0.001 (–)	NIST, TU01
[19]	5D-MHNN (fx-32)	Cyclone V	33.8	–	40	0.0069	–	NIST, TU01
[7]	SVCNC (fx-8)	Artix-7	100.0	3,113	20	1.045	0.34 (–)	NIST, TU01
[2]	Tinkerbell (fx-24)	Artix-7	97.0	227	16	2.330	10.26 (19.3)	NIST, DH, SmallCrush
[25]	TL-COTDCM (fx-128)	Kintex-7	120.5	4,694	224	10.284	2.19 (–)	NIST, SmallCrush
This work (xoshiro only)	xoshiro128** (int-32)	Zynq-7000	181.8	685	0	5.818	8.49 (4.8)	BigCrush [27]
This work (full)	**Hybrid (int-32)**	**Zynq-7000**	**181.8**	**934**	**0**	**5.818**	**6.23 (5.0)**	**NIST, DH, BigCrush**

5 Conclusion

This paper presented a hybrid pseudo-random number generator for FPGA platforms that targets high-throughput operation in resource-constrained embedded applications. To reduce the resource overhead and throughput limitations associated with classical chaotic implementations, we proposed a DSP-free hybrid architecture combining the Xoshiro128** algorithm with a Weyl sequence and a branchless integer-based Tent map. Designed using high-level synthesis for the Xilinx Zynq-7000 SoC, the post-place-and-route implementation uses 934 Slice LUTs. The optimized datapath achieved a sustained throughput of 5.82 Gbps while consuming 29 mW of dynamic power (zero-delay estimate; 42 mW including glitch activity). The reported design exposes a nominal initialization space of 2^{192} and passes the NIST SP 800-22, Dieharder, and TestU01 BigCrush test suites. These results suggest that the proposed architecture may be suitable for resource-constrained embedded applications that need a high-throughput, DSP-free source of statistically strong pseudorandom numbers. Compared with recent FPGA-based chaotic PRNG architectures, the proposed design achieves a favorable balance between resource efficiency, throughput, and statistical validation depth, delivering 5.82 Gbps without requiring any DSP slices while passing the full TestU01 BigCrush battery.

The work has the following limitations. The integer Tent step is algebraically affine and therefore adds no algebraic non-linearity; the Weyl seed w_0 has weak one-bit sensitivity (about 1/8 of the output bits change); the number of distinguishable initializations is below the nominal 2^{192}; the reported power is an out-of-context estimate with an ideal clock net, and the zero-delay value is a lower bound; and the TestU01 batteries were run with a single seed.

Future work may include extending the proposed architecture to other FPGA families, evaluating the core inside a complete SoC design, where the clock network is routed and its power can be measured directly, and replacing the affine integer Tent step with a non-linear layer.

Declarations

Funding

No funds, grants, or other support was received for conducting this study or preparing this manuscript.

Conflicts of interest/Competing interests

The authors have no conflicts of interest to declare relevant to this article's content.

Data and Code Availability

The datasets generated and analyzed in this study, together with the complete source code, synthesis and implementation scripts, and the automated driver used to reproduce the reported results, are openly available in a Zenodo repository at `https://doi.org/10.5281/zenodo.23895795` [37]. The package includes the C/RTL reference model of the generator, the Vivado HLS and Vivado out-of-context flow scripts, the statistical-test configurations, seeds, and raw output logs (NIST SP 800-22, Dieharder, and TestU01), the ablation and structural-analysis scripts, and the figure-generation scripts. A single top-level driver regenerates the resource, timing, power, sensitivity, and structural results from the source code and summarizes the statistical-test results from the deposited logs.

References

1. Sarra Senouci, Sid Ali Madoune, Mohammed Raouf Senouci, Abdelkader Senouci, and Zhangchun Tang. A

novel prng for fiber optic transmission. *Chaos, Solitons & Fractals*, 192:116038, 2025.
2. Seghier Abdelkrim, Merah Hocine, Merah Lahcene, Özen Özer, Talbi Larbi, and Ali-Pacha Adda. Secure chaos-based prng for low-power and resource-constrained devices: S. abdelkrim et al. *Nonlinear Dynamics*, pages 1–31, 2025.
3. C Gilbert and MA Gilbert. Evaluation of the efficiency of advanced number generators in cryptographic systems using a comparative approach. *International Journal of Scientific Research and Modern Technology*, 3(11), 2024.
4. Rasika B Naik and Udayprakash Singh. A review on applications of chaotic maps in pseudo-random number generators and encryption. *Annals of Data Science*, 11(1):25–50, 2024.
5. Musha Ji'e, Dengwei Yan, Shuqi Sun, Fengqing Zhang, Shukai Duan, and Lidan Wang. A simple method for constructing a family of hamiltonian conservative chaotic systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(8):3328–3338, 2022.
6. Wenhao Liu, Kehui Sun, Huihai Wang, Binglun Li, and Yongjiu Chen. Inverse proportional chaotification model for image encryption in iot scenarios. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 72(1):254–264, 2024.
7. Chenyu Wang, Jun Zheng, and Yining Qian. Structure-varying complex network chaotic model and its hardware implementation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 72(9):4673–4685, 2024.
8. Yuyao Luo, Chunlei Fan, Chengbin Xu, and Xinyu Li. Design and fpga implementation of a high-speed prng based on an nd non-degenerate chaotic system. *Chaos, Solitons & Fractals*, 183:114951, 2024.
9. Ahmed A Rezk, Ahmed H Madian, Ahmed G Radwan, and Ahmed M Soliman. Reconfigurable chaotic pseudo random number generator based on fpga. *AEU-international Journal of Electronics and Communications*, 98:174–180, 2019.
10. Jianeng Tang, Jianan Hou, and Yinghang Zhao. 2-d chaotic model with fpga implementation and its application in prng. *IEEE Transactions on Industrial Electronics*, 72(12):14106–14115, 2025.
11. Fethi Dridi, Safwan El Assad, Wajih El Hadj Youssef, Mohsen Machhout, and Abed Ellatif Samhat. Design, fpga-based implementation and performance of a pseudo-chaotic number generator. *Advances in Electrical and Computer Engineering*, 21(2):41–48, 2021.
12. Zhongyun Hua and Yicong Zhou. One-dimensional nonlinear model for producing chaos. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(1):235–246, 2017.
13. Radu Hobincu and Octaviana Datcu. Fpga implementation of a chaos based prng targetting secret communication. In *2018 International symposium on electronics and telecommunications (ISETC)*, pages 1–4. IEEE, 2018.
14. Fei Yu, Lixiang Li, Binyong He, Li Liu, Shuai Qian, Zinan Zhang, Hui Shen, Shuo Cai, and Yi Li. Pseudorandom number generator based on a 5d hyperchaotic four-wing memristive system and its fpga implementation: Prng based on a 5d hyperchaotic four-wing memristive system and its fpga implementation. *The European Physical Journal Special Topics*, 230(7):1763–1772, 2021.
15. Sen Zhang, Chunbiao Li, Jiahao Zheng, Xiaoping Wang, Zhigang Zeng, and Guanrong Chen. Generating any number of diversified hidden attractors via memristor coupling. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(12):4945–4956, 2021.
16. Youcef Alloun, Mohamed Salah Azzaz, Abdenour Kifouche, Mahdi Madani, and El-Bay Bourennane. An efficient design and fpga implementation of a hyper-chaotic and deep learning based prng for image security application. In *2024 1st International Conference on Electrical, Computer, Telecommunication and Energy Technologies (ECTE-Tech)*, pages 1–6. IEEE, 2024.
17. Ismail Koyuncu, Karthikeyan Rajagopal, Murat Alcin, Anitha Karthikeyan, Murat Tuna, and Metin Varan. Control, synchronization with linear quadratic regulator method and ffann-based prng application on fpga of a novel chaotic system. *The European Physical Journal Special Topics*, 230(7):1915–1931, 2021.
18. Bishoy K Sharobim, Mohammed H Yacoub, Wafaa S Sayed, Ahmed G Radwan, and Lobna A Said. Artificial neural network chaotic prng and simple encryption on fpga. *Engineering Applications of Artificial Intelligence*, 126:106888, 2023.
19. Jeferson Andres Garzon Gonzalez, Jose de Jesus Rangel-Magdaleno, and Jesus Manuel Munoz-Pacheco. Fpga implementation of a multi-prng based on a multiscroll chaotic hopfield neural network. *IEEE Transactions on Industrial Informatics*, 2025.
20. Xiangxin Leng, Xiaoping Wang, and Zhigang Zeng. Multi-type pseudo-random numbers generation control and transmission based on multistable attractors with fpga implementation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.
21. Serhii Haliuk, Oleh Krulikovskyi, Dmytro Vovchuk, and Fernando Corinto. Memristive structure-based chaotic system for prng. *Symmetry*, 14(1):68, 2022.
22. P Vijayaraghavan and R Amutha. Asic vs fpga realization of a cryptographically secure random number generator using chaotic map. In *2025 IEEE 5th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA)*, pages 1–6. IEEE, 2025.
23. Miguel Garcia-Bosque, Adrián Pérez-Resa, Carlos Sánchez-Azqueta, Concepción Aldea, and Santiago Celma. Chaos-based bitwise dynamical pseudorandom number generator on fpga. *IEEE Transactions on Instrumentation and Measurement*, 68(1):291–293, 2018.
24. Murat Tuna. A novel secure chaos-based pseudo random number generator based on ann-based chaotic and ring oscillator: design and its fpga implementation. *Analog Integrated Circuits and Signal Processing*, 105(2):167–181, 2020.
25. Shouliang Li, Yinzhe Liu, Fengyuan Ren, and Zhen Yang. Design of a high throughput pseudorandom number generator based on discrete hyper-chaotic system. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(2):806–810, 2022.
26. Richard Peirce Brent. Some long-period random number generators using shifts and xors. *The Proceedings of ANZIAM*, 48:C188–C202, 2006.
27. David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *ACM Transactions on Mathematical Software (TOMS)*, 47(4):1–32, 2021.
28. Muhammad Irfan and Muhammad Asif Khan. Cryptographically secure pseudo-random number generation using a robust chaotic tent map: a novel approach. *Cogent Engineering*, 12(1):2558751, 2025.
29. Mohammed A. Alhartomi, Mohd. Tasleem Khan, Saeed Alzahrani, Ahmed Alzahmi, Rafi Ahamed Shaik, Jinti Hazarika, Ruwaybih Alsulami, Abdulaziz Alotaibi, and Meshal Al-Harthi. Low-area and low-power VLSI architectures for long short-term memory networks. *IEEE*

Journal on Emerging and Selected Topics in Circuits and Systems, 13(4):1000–1014, 2023.
30. Boyang Chen, Mohd Tasleem Khan, George Goussetis, Mathini Sellathurai, Yuan Ding, Joao F. C. Mota, and Jongeun Lee. COMET: Co-optimization of CNN models using efficient-hardware OBC techniques. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 73(8):5597–5609, 2026.
31. Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical Report Special Publication 800-22 Revision 1a, National Institute of Standards and Technology (NIST), 2010.
32. Robert G. Brown, Dirk Eddelbuettel, and David Bauer. Dieharder: A random number test suite (version 3.31.1). Duke University Physics Department, 2018.
33. Pierre L'ecuyer and Richard Simard. Testu01: Ac library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):1–40, 2007.
34. Philip Hawkes and Gregory G. Rose. Guess-and-determine attacks on SNOW. In *Selected Areas in Cryptography (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46. Springer, 2003.
35. James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.
36. Kai Tan and Chengqing Li. Graph structure of the generalized tent map over ring $\mathbb{Z}_{2^e}$. arXiv preprint arXiv:2501.12645, 2025.
37. Milad Mohammadian, Bahadin Azizinasab, Abobakr Mohammadian, Mahrokh G. Shayesteh, Mohammad Saeed Khayaty, Sheida Mohammadpour, and Mosleh Yousefi. Reproduction package for "A High-Throughput DSP-Free PRNG for FPGA based on Xoshiro128** and Integer Chaotic Mixing", version 1.1.0. Zenodo, `https://doi.org/10.5281/zenodo.23895795`, 2026.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- dspfreeprngfpgav1.1.0.zip