LFSR: Beyond the Sequential – Unlocking the Potential of Immediate N^{th} Cycle Output

Kshiteej Kosambia[1], Prakash Darji[2]

[1]Cadence Design Systems, India

[2]Cadence Design Systems, India
[1]kshiteej@cadence.com; [2]darji@cadence.com

Abstract— Linear Feedback Shift Registers (LFSRs) are widely used in digital systems for pseudo-random sequence generation, key expansion, error detection, and data scrambling. They are popular because they offer simple hardware implementation and good statistical properties. A key limitation, however, is their strictly sequential behavior: each new state depends on the previous one. As a result, reaching the N^{th} state requires N clock cycles, which adds delay in systems that need fast synchronization or random access to intermediate states. This constraint affects high-speed encryption engines and multi-clock-domain communication circuits. Existing approaches—such as storing precomputed states or running multiple LFSRs in parallel—help reduce latency but increase area and routing overhead. Mathematically, the challenge comes from the state-transition matrix L, since computing L^N directly becomes impractical for longer LFSR sizes. This motivates the need for architectures that can compute N^{th} LFSR state transition efficiently while keeping hardware cost low.

Keywords— LFSR, Galois LFSR, pseudo-random sequence generation, state transition matrix, synchronization latency.

I. INTRODUCTION

Linear Feedback Shift Registers (LFSRs) are among the most fundamental and versatile building blocks in modern digital design. An LFSR is a type of sequential logic circuit constructed from a series of flip-flops connected in a shift register configuration, with selected outputs fed back through exclusive-OR (XOR) gates. On each clock cycle, the register contents shift by one position, and the new input bit is formed as a linear function of specific prior bits, as determined by a primitive/feedback polynomial.

The output sequence generated by an LFSR exhibits pseudo-random characteristics, making it a compact and efficient mechanism for producing deterministic yet statistically random-like binary streams. When configured with a *primitive polynomial* over Galois Field, an *n*-bit LFSR generates a maximal-length sequence (m-sequence) of period 2^n-1, cycling through all possible nonzero states before repeating. This periodic nature is crucial in applications requiring uniform statistical coverage and predictable repeatability.

Because of their simplicity, speed, and regular structure, LFSRs are pervasive in cryptography, data scrambling, error detection and correction (CRC generation), test pattern generation in built-in self-test (BIST), spread-spectrum communication, and many other designs. Unlike complex random-number generators requiring heavy arithmetic or large memories, LFSRs achieve high throughput with minimal hardware—only flip-flops and XOR gates—making them especially attractive for low-power and high-frequency integrated circuits.

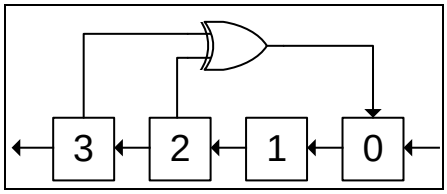

Fig. 1 Block diagram of a 4-bit Linear Feedback Shift Register (LFSR)

The representative example in Fig. 1 shows a 4-bit LFSR using taps at bit positions 3 and 2 (corresponding to the primitive polynomial x^4+x^3+1). As the Table. 1 illustrates, the circuit cycles through a sequence of 15 distinct nonzero states before returning to its initial configuration, demonstrating the periodic and deterministic behavior of the LFSR. Each subsequent state is linearly derived from its predecessor.

TABLE I
ILLUSTRATION OF THE 15-STATE PERIODIC SEQUENCE GENERATED BY A 4-BIT LFSR

State	Bit 3	Bit 2	Bit 1	Bit 0
S0	1	1	1	1
S1	1	1	1	0

S2	1	1	0	0
S3	1	0	0	0
S4	0	0	0	1
S5	0	0	1	0
S6	0	1	0	0
S7	1	0	0	1
S8	0	0	1	1
S9	0	1	1	0
S10	1	1	0	1
S11	1	0	1	0
S12	0	1	0	1
S13	1	0	1	1
S14	0	1	1	1
S0	1	1	1	1

Despite their inherent simplicity, LFSRs embody deep mathematical properties based on linear recurrence relations and finite-field algebra. These properties enable deterministic reproducibility while approximating statistical randomness—an essential balance in secure communication and testing systems. Their architectural regularity also facilitates scalable hardware implementations, pipeline-friendly designs, and minimal routing complexity, ensuring their continued relevance in both ASIC and FPGA domains.

II. Background: LFSR Implementation Architectures

The Linear Feedback Shift Register (LFSR) can be realized in multiple architectural forms depending on the desired balance among latency, hardware complexity, and randomness fidelity. Each implementation embodies the same underlying linear recurrence relation but differs in the way the state transition operation is executed: either temporally (sequentially), or spatially (through precomputed lookup), or analytically (through logic-based state computation).

This section delineates the predominant realizations of LFSR-based pseudo-random sequence generators, identifies their respective limitations, and establishes the comparative baseline for evaluating the proposed architecture.

A. Function-Based Implementation

The function-based implementation represents a more analytically efficient reformulation of LFSR progression. Instead of explicit storage or iterative shifting, the next-state logic is encapsulated as a combinational function that derives the output directly from the current state according to the recurrence relation encoded by primitive polynomial P(x).

Advantages

- *Deterministic periodicity:* Generates maximal-length sequence (2^n-1) with guaranteed pseudo-random statistics.
- *High operating frequency:* Short combinational depth between flip-flops ensures robust timing closure even at gigahertz clock rates.

Disadvantages

- *Sequential latency:* Accessing the N^{th} state mandates exactly N cycles.
- *Limited controllability:* No direct means to "jump" ahead or index into arbitrary sequence positions.
- *Synchronization overhead:* When used across multiple clock domains, resynchronization demands multi-cycle progression, limiting responsiveness.

B. Lookup Table (LUT)-Based Implementation

The lookup table–based (LUT-LFSR) design attempts to eliminate sequential latency by precomputing and storing a finite subset of future states in memory. Each address entry corresponds to a specific sequence state, containing the pre-evaluated LFSR value. At runtime, the target state S_N can be retrieved directly from the table without iterative shifting.

Advantages

- *Constant access latency:* N^{th} state is available in a single memory access.
- *Breaks sequential nature:* Eliminates sequential traversal, enabling asynchronous or event-triggered state selection.

Disadvantages

- *Exponential memory cost:* For a n bit LFSR, the required space is 2^n-1; storing even a subset of states scales exponentially.
- *Static structure:* The LUT must be regenerated if the primitive polynomial P(x) or seed changes.

- *Power and area inefficiency:* Memory access consumes more dynamic energy than sequential XOR operations, especially for *n*-bit LFSRs, where *n* is a big number.
- *Scalability constraints:* Impractical for large *n* due to storage explosion.

In essence, while LUT-based implementations theoretically achieve zero-latency access to arbitrary LFSR states, their memory and scalability overheads preclude practical use in modern VLSI design. Conversely, function-based implementations offer a computationally efficient and synthesizable balance. Retaining deterministic control flow while allowing analytical acceleration of the state-transition process. Hence, the subsequent sections focus on comparing the proposed novel architecture primarily against the function-based RTL realizations.

III. Problem Statement

Although Linear Feedback Shift Registers (LFSRs) provide a highly efficient means of generating pseudo-random sequences, their state evolution remains inherently sequential. As discussed in the previous section, traditional function-based RTL realizations operate on a one-step-per-cycle basis, governed by the recurrence relation: $S(n+1) = S(n)$.

In such architecture, accessing the N^{th} state of the sequence $S(n)$ from an initial seed S_0, N number of cycles required to generate N^{th} state of LFSR. This implies that even if a function-based implementation is used, where the next state is computed through combinational logic, the system must still perform N sequential transitions to reach the desired state. Each transition consumes one clock cycle, resulting in a temporal latency directly proportional to N.

The limitation of existing function-based LFSR realizations lies in their strict temporal dependency between consecutive states. To overcome this, an architecture capable of computing or accessing S_N directly or in sublinear time is required, without compromising sequence integrity, randomness, or hardware efficiency. This problem definition forms the analytical and architectural foundation for the proposed novel LFSR design, described in the subsequent section.

IV. Novel Approach

The proposed solution addresses this limitation by shifting from sequential to direct-access operation. Instead of traversing each intermediate state, mathematical relationships are derived that allow the computation of the N^{th} state directly from the current state. By expressing the LFSR state transition as a linear recurrence, these relationships can be reformulated into equations that bypass sequential steps. When implemented in hardware, the derived equations produce the N^{th} state combinatorially, enabling the desired output to be available on the very next clock edge. This removes the fundamental dependency on sequential shifts and allows constant-time computation of arbitrary states.

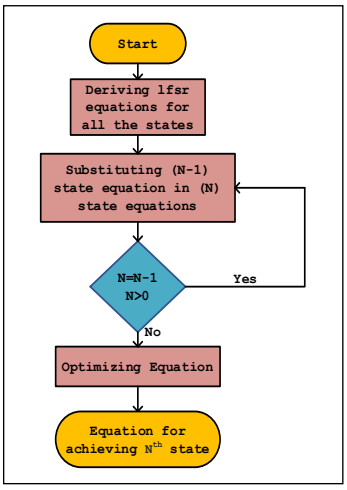

Fig. 2 Flowchart illustrating the working of the proposed novel LFSR approach

To illustrate the process of generating a direct jump-ahead mapping for an LFSR, we consider a 5-bit LFSR defined by the feedback polynomial 10100, which corresponds to the primitive polynomial: $P(x) = x5 + x3 + 1$. In its standard sequential implementation, the next state at time $k+1$ depends only on the state at time k. Computing the state after N steps therefore requires N clock cycles. Instead, we mathematically unroll the recurrence to obtain a purely combinational expression for the state at step $k+4$, expressed directly in terms of the initial state.

lfsr_1[4] = lfsr_0[0];	**lfsr_2[4]** = lfsr_1[0];	**lfsr_3[4]** = lfsr_2[0];	**lfsr_4[4]** = lfsr_3[0];
lfsr_1[3] = lfsr_0[4];	**lfsr_2[3]** = lfsr_1[4];	**lfsr_3[3]** = lfsr_2[4];	**lfsr_4[3]** = lfsr_3[4];
lfsr_1[2] = lfsr_0[3] ^ lfsr_0[0];	**lfsr_2[2]** = lfsr_1[3] ^ lfsr_1[0];	**lfsr_3[2]** = lfsr_2[3] ^ lfsr_2[0];	**lfsr_4[2]** = lfsr_3[3] ^ lfsr_3[0];
lfsr_1[1] = lfsr_0[2];	**lfsr_2[1]** = lfsr_1[2];	**lfsr_3[1]** = lfsr_2[2];	**lfsr_4[1]** = lfsr_3[2];
lfsr_1[0] = lfsr_0[1];	**lfsr_2[0]** = lfsr_1[1];	**lfsr_3[0]** = lfsr_2[1];	**lfsr_4[0]** = lfsr_3[1];

Fig. 3 Sequential State Expansion of the LFSR state across four sequential steps

By repeatedly substituting each intermediate state with its symbolic form from the previous step, we eliminate all dependencies on *lfsr_1*, *lfsr_2*, and *lfsr_3*. The result is a direct mapping from the initial state *lfsr_0* to the state four steps later.

lfsr_4[4] = lfsr_0[3] ^ lfsr_0[0];
lfsr_4[3] = lfsr_0[2];
lfsr_4[2] = lfsr_0[1] ^ lfsr_0[3] ^ lfsr_0[0];
lfsr_4[1] = lfsr_0[0] ^ lfsr_0[2];
lfsr_4[0] = lfsr_0[4] ^ lfsr_0[1];

Fig. 4 Final optimized 4-step jump-ahead equations

A real-world example where advanced LFSR blocks become the only viable solution is in high-speed data scrambling and descrambling across multiple clock domains. To better understand this, let's consider an example from modern communication interfaces, where we have two clock domains with a frequency ratio of 4:1. The low-speed (LS) domain operates at 100 MHz, while the high-speed (HS) domain runs at 400 MHz. This means that one clock cycle of the LS domain is equivalent to four clock cycles of the HS domain. When transferring data from the LS domain to the HS domain, the system must provide four cycle's worth of data for the HS domain during a single LS clock cycle. For instance, during the first LS clock, four data words (A, B, C, and D) are transferred so that the HS domain has valid data on each of its four clock cycles.

Alongside this data, the corresponding LFSR states are also required. For the first LS clock, the system needs the LFSR states +1, +2, +3, and +4, aligned with data words A, B, C, and D, respectively. In the second LS clock, the next set of data (E, F, G, and H) must be accompanied by LFSR states +5, +6, +7, and +8. Similarly, in the third LS clock, data (I, J, K, L) requires LFSR states +9, +10, +11, and +12.

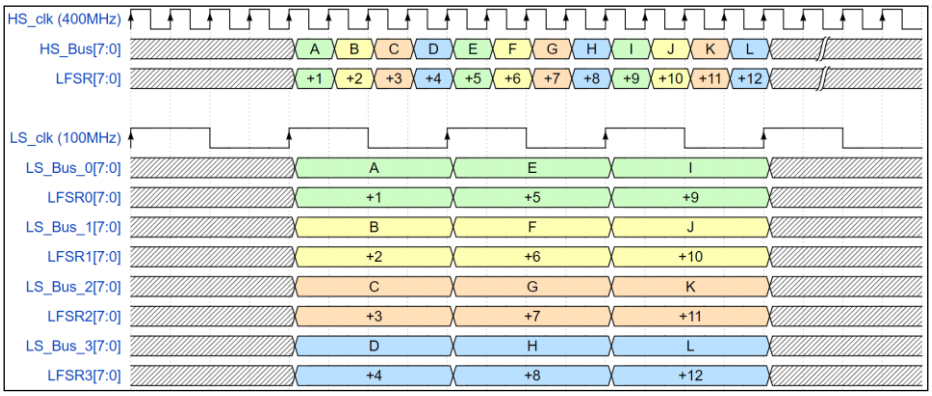

Fig. 5 Waveform illustrating data transfer between 100 MHz and 400 MHz domains at a 4:1 frequency ratio

With a conventional LFSR, reaching these higher-order states is inefficient because the circuit works sequentially. Given an N^{th} state as input, it produces only the $(n+1)^{th}$ state per clock cycle. For example, starting with +4, a conventional LFSR would require four additional LS clock cycles to compute up to +8. The same delay occurs when trying to advance from +8 to +12. This sequential dependency makes real-time alignment of multiple data streams impractical.

The proposed advanced LFSR overcomes this limitation by allowing direct multi-step advancement. Here, an N^{th} state can be used to generate the $(n+4)^{th}$ state in just one clock cycle. During the first LS cycle, we already have +1, +2, +3, and +4. These are fed into four instances of the advanced LFSR (L1, L2, L3, and L4). In the second LS cycle, each instance computes the next set of required states in parallel: L1 produces +5, L2 produces +6, L3 produces +7, and L4 produces +8. A high-level architectural diagram illustrating the proposed design methodology is presented below section. Similarly, during the third LS cycle, the same blocks take +5, +6, +7, and +8 as inputs and directly output +9, +10, +11, and +12.

This process continues for subsequent cycles. In effect, the advanced LFSR design enables immediate availability of the required LFSR states in every LS clock cycle without the sequential delays of conventional designs. The result is efficient data transfer between LS and HS domains with perfectly aligned scrambling states, and significantly reducing latency.

The implications of this design are substantial. In high-speed serial interface hardware, such as DDR memory controller, where scrambling and descrambling require deterministic pseudo-random sequences aligned with data streams. Latency in accessing required states can disrupt timing closure and reduce effective bandwidth by adding additional clock cycles, in the case of conventional LFSR design. Similarly, in encryption systems, key-stream generators must deliver specific states without additional delays. Advanced LFSRs eliminate these bottlenecks by ensuring immediate state availability. Furthermore, unlike precomputation-based methods, they do not demand additional memory.

By deriving and implementing direct state equations, advanced LFSR architectures represent a paradigm shift in pseudo-random sequence generation. They transform LFSRs from purely sequential circuits into structures capable of constant-time state access. This enables new classes of high-speed, low-latency designs in communication systems, memory controllers, and cryptographic engines. Beyond efficiency gains, the approach also enhances design scalability: as systems demand ever-higher data rates, the ability to compute arbitrary states in real time becomes indispensable rather than optional.

V. High-Level Architecture of the Proposed Novel LFSR

This section presents the high-level architecture of the proposed novel LFSR approach, highlighting its operational principle and structural organization. The design employs a conventional LFSR core that receives the system clock (*clk*), reset (*rst*), and an initialization seed (*initial_seed*). The core sequentially generates four consecutive LFSR states, denoted as $seed_1$–$seed_4$. These seeds are connected to four identical Advanced LFSR (+4) blocks, which produce outputs $lfsr_5$–$lfsr_8$. Each Advanced LFSR implements a precomputed multi-step "jump" transform, enabling it to advance its input state by four cycles within a single clock event. In the subsequent clock cycle, the same Advanced LFSR blocks receive the previously generated states (+5, +6, +7, and +8) as inputs and compute the next four states (+9, +10, +11, and +12) in parallel.

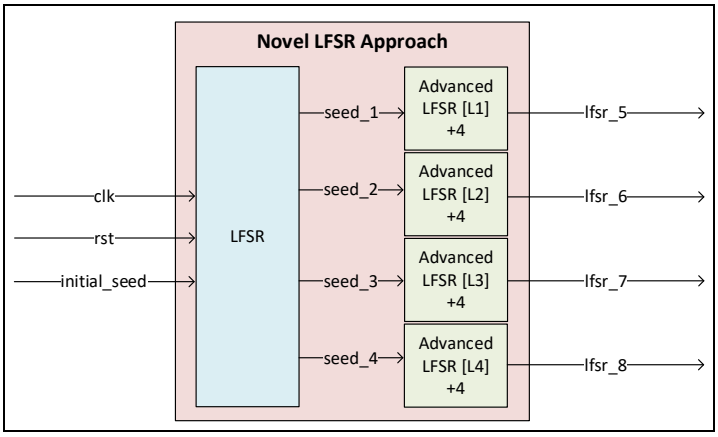

Fig. 6 Architectural overview of the proposed lookup-assisted LFSR enabling parallel generation of advanced sequence states

Consequently, state advancements by arbitrary offsets cannot be realized using the conventional LFSR architecture, since each state transition is inherently sequential and dependent on the preceding state. In contrast, the proposed novel LFSR architecture enables direct computation and retrieval of these advanced states, making such multi-step advancements feasible and practically implementable within real-time digital systems. This parallel-state mechanism transforms temporal state progression into spatial computation, achieving zero sequential latency and high throughput. The architecture is particularly beneficial for high-speed data scrambling, multi-domain synchronization, and BIST pattern generation, where concurrent pseudo-random sequences are required across multiple clock domains.

VI. Experimental Evaluation

To quantitatively evaluate the performance of the proposed LFSR architecture, both the conventional function-based implementation and the novel approach-based implementation were synthesized, analysed, and benchmarked using the Cadence Genus™ Synthesis Solution and subsequently placed and routed using the Cadence Innovus™ Implementation System under identical technology and constraint environments. This integrated flow ensures a fair, architecture-neutral comparison focused on intrinsic design efficiency rather than tool or constraint bias.

The synthesis was carried out using a TSMC 7nm standard-cell library under a target clock constraint of 5 GHz with 50% duty cycle. Both designs were implemented at module level, maintaining functional equivalence in terms of output sequence, seed initialization, and polynomial feedback logic.

A. Experimental Setup

- *Technology libraries:* tcbn07_bwph300l8p64pd_base_svtssgnp_0p675v_m40c_cworst_CCworst_T 130, tcbn07_bwph300l8p64pd_base_lvtssgnp_0p675v_m40c_cworst_CCworst_T 130.
- *Cell Type*: LVT.
- *Operating conditions:* ssgnp_0p675v_m40c_cworst_CCworst_T.
- *Interconnect mode:* Placement.
- *Area mode:* Physical library.

B. Histogram Level

Number of Logic Levels	Number(%)	Histogram
0 -> 0	0(0.0)	
1 -> 1	13(40.6)	**************************************
2 -> 2	2(6.2)	******
3 -> 3	2(6.2)	******
4 -> 4	6(18.8)	******************
5 -> 5	5(15.6)	***************
6 -> 6	4(12.5)	************
7 -> 7	0(0.0)	
8 -> 8	0(0.0)	
9 -> 9	0(0.0)	
	32	total end points

Fig. 7 Logic-level histogram for function-based implementation

Number of Logic Levels	Number(%)	Histogram
0 -> 0	0(0.0)	
1 -> 1	16(50.0)	**
2 -> 2	0(0.0)	
3 -> 3	2(6.2)	******
4 -> 4	7(21.9)	*********************
5 -> 5	5(15.6)	***************
6 -> 6	2(6.2)	******
7 -> 7	0(0.0)	
8 -> 8	0(0.0)	
9 -> 9	0(0.0)	
	32	total end points

Fig. 8 Logic-level histogram for novel approach-based implementation

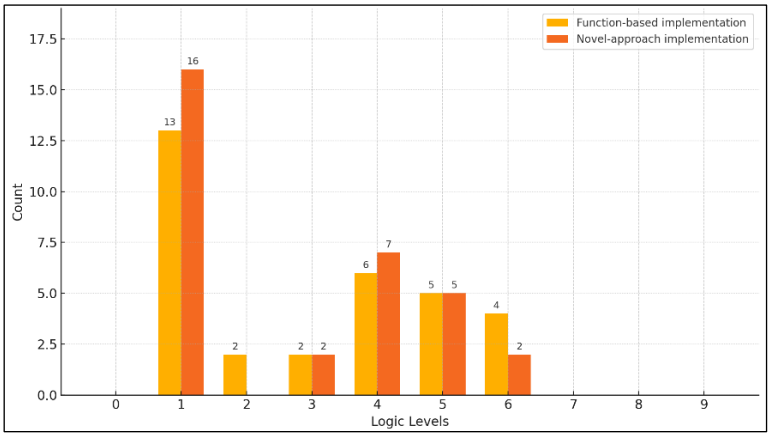

Fig. 9 Comparison: Function-based vs Novel-approach implementations

The logic-level histogram serves as an intrinsic indicator of the combinational complexity and pipeline uniformity across sequential boundaries within synthesized designs. In the function-based implementation, the distribution of logic depth spans from one to six levels, with approximately 12.5% of endpoints exhibiting the maximum logic depth. This broader spread signifies a non-uniform pipeline structure wherein certain stages are shallow while others accumulate greater combinational depth, potentially introducing localized timing bottlenecks and degrading path balance. In contrast, the novel approach-based implementation demonstrates a markedly tighter clustering between one and four logic levels, with half of the endpoints exhibiting only a single logic level between registers. Such a distribution reflects a well-partitioned and efficiently pipelined architecture, characterized by reduced combinational cone depth and enhanced timing uniformity. The reduction of deep logic paths from 12.5% in function-based design to 6.2% in novel approach–based design directly correlates with a lower critical-path propensity and improved synthesis Quality of Results (QoR). From an engineering standpoint, the structural regularity of novel approach–based design yields a more predictable and balanced timing profile, easing timing closure effort and enabling higher achievable clock frequencies in both logic synthesis and physical implementation stages. Consequently, based on the histogram-driven combinational depth analysis, novel approach–based design exhibits superior architectural balance and timing efficiency, indicating a synthesis-optimized topology that promotes faster convergence in Cadence Genus™ and subsequent Innovus™ optimization phases.

C. Synthesis Timing Results

Cost Group	Critical Path Slack	TNS	No of gates on Critical Path	Violating Paths
clk	0.2	0.0	5	0
default	No paths	0.0		
Total		0.0		0

Cost Group	Critical Path Slack	TNS	No of gates on Critical Path	Violating Paths
clk	2.4	0.0	4	0
default	No paths	0.0		
Total		0.0		0

Fig. 10 Comparative Synthesis Timing results: function-based (left) *vs* novel approach–based (right)

Novel approach–based design achieves a significantly higher positive slack of +2.4 ns, with only four gates on the critical path. This result demonstrates a more compact and efficiently synthesized data path, consistent with the earlier histogram evidence of tighter clustering around one to four logic levels. The improvement of +2.2 ns in critical path slack indicates a substantially faster timing profile, translating directly into a higher maximum operating frequency. Since both designs exhibit zero TNS and no violating paths, the distinguishing factor becomes timing headroom and path efficiency — both of which are clearly superior in novel approach–based design.

D. Area and Power Evaluation

Instance Count	
Leaf Instance Count	91
Physical Instance count	0
Sequential Instance Count	16
Combinational Instance Count	75
Hierarchical Instance Count	0
Area	
Cell Area	22.810
Physical Cell Area	0.000
Total Cell Area (Cell+Physical)	22.810
Net Area	9.160
Total Area (Cell+Physical+Net)	31.970
Power	
Leakage Power	7.587 nW
Dynamic Power	243451.825 nW
Total Power	243459.412 nW
Floorplan Utilization	74.26%
Max Fanout	18 (load_seed)
Min Fanout	1 (n_24)
Average Fanout	2.5
Terms to net ratio	3.1982
Terms to instance ratio	3.9011
Runtime	258.129073 seconds
Elapsed Runtime	553 seconds
Genus peak memory usage	3085.75
Innovus peak memory usage	4198.804688
Hostname	sjpipg116.cadence.com
Physical Data	
Total Net Length	294.98 um
Average Net Length	2.68 um
Routing Congestion	H: 0.00% V: 0.00%

Instance Count	
Leaf Instance Count	90
Physical Instance count	0
Sequential Instance Count	16
Combinational Instance Count	74
Hierarchical Instance Count	0
Area	
Cell Area	22.925
Physical Cell Area	0.000
Total Cell Area (Cell+Physical)	22.925
Net Area	9.114
Total Area (Cell+Physical+Net)	32.039
Power	
Leakage Power	7.676 nW
Dynamic Power	237539.883 nW
Total Power	237547.559 nW
Floorplan Utilization	72.05%
Max Fanout	18 (load_seed)
Min Fanout	1 (n_44)
Average Fanout	2.5
Terms to net ratio	3.1818
Terms to instance ratio	3.8889
Runtime	261.127622 seconds
Elapsed Runtime	560 seconds
Genus peak memory usage	3092.25
Innovus peak memory usage	4153.867188
Hostname	sjpipg116.cadence.com
Physical Data	
Total Net Length	293.72 um
Average Net Length	2.69 um
Routing Congestion	H: 0.00% V: 0.00%

Fig. 11 Comparative Area and Power Evaluation: function-based (left) *vs* novel approach–based (right)

The post-implementation analysis further validates the performance and efficiency trade-offs between the two synthesized designs. Both designs exhibit similar instance composition, with function–based design containing 91 total cells and novel approach–based design containing 90, including 16 sequential elements in each case. Although this difference of a single cell may appear marginal at the module level, it becomes significant when scaled to system-level integration, where hundreds or thousands of such instances are instantiated in a real-world SoC. In such cases, even a one-cell reduction per instance accumulates to a noticeable reduction in overall area, power, and routing resource utilization, thereby influencing chip-level efficiency and manufacturability.

The overall silicon footprint remains nearly identical, with total cell area values of 22.810 µm² and 22.925 µm² for function–based design and novel approach–based design, respectively, and total area (including interconnect) of 31.970 µm² and 32.039 µm². However, novel approach–based design demonstrates a marginally lower floorplan utilization of 72.05% compared to 74.26% in function-based design, implying slightly improved placement flexibility and potential routing ease during physical design. In terms of power, novel approach–based design achieves a reduction in total power from 243.459 µW to 237.547 µW, primarily due to a decrease in dynamic power consumption of approximately 2.4%, while leakage power remains constant within measurement precision (~7.6 nW). These improvements correlate with the reduced logic depth and shorter critical path length observed earlier, leading to more efficient switching activity and minimized internal capacitance. Physical data also reveal comparable total net lengths (294.98 µm in function-based design vs. 293.72 µm in novel approach-based design) and identical routing congestion metrics (0% horizontal/vertical), confirming that both designs maintain excellent layout regularity. Collectively, the results indicate that novel approach–based design offers superior power–performance efficiency, achieving lower power dissipation and better utilization uniformity without incurring area penalties, thereby reaffirming its architectural and synthesis advantages in both Genus™ and Innovus™ environments.

VII. Related Work

Research on Linear Feedback Shift Registers (LFSRs) spans algebraic sequence analysis, minimal polynomial reconstruction, and methods for accelerating state traversal. The classical Berlekamp–Massey (BM) algorithm is one of the foundational techniques in this space. BM reconstructs the minimal characteristic polynomial of an unknown binary sequence and is widely used in test-pattern generation and diagnostic workflows [4]. By identifying the shortest LFSR capable of generating a given output sequence, BM provides strong algebraic insight into sequence structure. However, BM and its practical extensions focus primarily on polynomial inference and sequence modelling, not on hardware mechanisms for directly computing the N^{th} LFSR state without stepping through intermediate states. Thus, while essential for determining LFSR structure, BM does not address the real-time latency bottlenecks targeted in this work.

A separate line of research investigates jump-ahead or skip-ahead algorithms for F^2-linear recurrences, motivated largely by parallel random-number generation and large-scale Monte-Carlo simulation. Haramoto et al. [5] and Bauke et al. [6] demonstrate that LFSR-like generators can be advanced by large step sizes using the companion-matrix formulation, enabling block-splitting and sub stream construction for distributed computation. These works use matrix exponentiation, binary decomposition of the skip distance, or precomputed tables of L^{2^i} to realize fast software jump-ahead $O(logN)$ complexity. Although highly effective in software and simulation environments, these methods rely on matrix arithmetic or substantial table storage and do not directly translate into compact combinational logic suitable for high-speed digital hardware.

Foundational sequence-theoretic work by Lin and Costello [8], Shen [9], Golomb [10], and Rueppel [11] analyze cycle structure, period decomposition, and subperiod construction of LFSR sequences, providing the algebraic foundation for multi-stream, segmented, and decimated sequence generation. Parallel LFSR and multi-bit CRC architectures have also been explored in communication and error-detection literature, where multi-step transition matrices or parallelized update equations are used to compute several output bits per cycle [12], [13]. These works derive fixed parallelization factors or polynomial-specific transforms.

More recent practical implementations, such as GPU-accelerated random-number libraries [7], utilize lookup tables storing powers of the transition matrix and combine them dynamically based on the binary representation of the desired skip distance. These approaches achieve constant latency but incur significant memory overhead, especially as the LFSR width increases. In contrast, the methodology proposed in *LFSR: Beyond the Sequential* derives symbolically optimized multi-step state equations and implements them as lightweight combinational *advanced LFSR* blocks. This enables single-cycle computation of the *N*-step-ahead state without resorting to large LUTs or repeated matrix multiplications, offering a hardware-efficient alternative that directly addresses sequential latency in real-time systems.

VIII. Conclusion

In conclusion, conventional LFSRs, though efficient for sequential bit generation, fall short in scenarios requiring immediate access to arbitrary states. The proposed advanced LFSR design addresses this by mathematically reformulating state transitions into direct-access equations, enabling constant-time computation

of the N^{th} state. Demonstrated in a multi-clock-domain communication system, this approach allows immediate generation of aligned scrambling states, ensuring seamless data transfer without latency. The method reduces hardware overhead, allows continuous throughput, and opens new possibilities for leveraging pseudo-random sequences in high-speed applications. Advanced LFSRs are thus not merely an optimization but a necessity for modern digital systems operating under stringent performance constraints.

The comparative evaluation between the two synthesized implementations establishes that novel approach–based design consistently achieves superior performance, power, and structural efficiency relative to function-based. The logic-level histogram and post-synthesis timing analyses at 5 GHz revealed that novel approach–based design possesses a shallower and more balanced combinational structure, yielding a critical-path slack improvement from +0.2 ns to +2.4 ns and one fewer gate in the longest timing path, thereby ensuring enhanced timing closure and higher attainable clock frequency. Post-implementation characterization using Cadence Innovus™ further corroborates these findings, showing reduced dynamic power ($\approx$2.4 %), comparable total area, and improved floorplan uniformity with lower utilization. Although the absolute difference in cell count between the two modules is only a single instance, this minute architectural refinement becomes highly consequential when the design is scaled to system-level integration, where hundreds or thousands of such blocks are instantiated in a full SoC. Under such scaling, even a one-cell reduction per instance compounds into measurable savings in total silicon area, power dissipation, and routing resources, directly influencing cost and manufacturability. Collectively, the results confirm that novel approach–based design delivers a superior Power–Performance–Area (PPA) trade-off, and its synthesis-optimized architecture provides a scalable and implementation-friendly solution for high-performance digital systems.

ACKNOWLEDGEMENT

The authors thank their colleagues Dipak Modi and Pankaj Kumar at Cadence Design Systems, Inc. for their guidance and support during this work.

REFERENCES

[1] M. Goresky and A. Klapper, Fibonacci and Galois Representations of Feedback-Shift Registers, Springer Science & Business Media, 2012.
[2] Cadence Design Systems, Inc., *Cadence Genus™ Synthesis Solution User Guide*, Version 23.1, San Jose, CA, USA, 2024.
[3] Cadence Design Systems, Inc., *Cadence Innovus™ Implementation System User Guide*, Version 23.1, San Jose, CA, USA, 2024.
[4] J. Acevedo, K. Kagaris, et al., "Using the Berlekamp–Massey Algorithm to Obtain LFSR Characteristic Polynomials for TPG," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2012.
[5] H. Haramoto, M. Matsumoto, T. Nishimura, F. Panneton and P. L'Ecuyer, "Efficient Jump Ahead for F_2-Linear Random Number Generators," *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 385–390, 2008.
[6] H. Bauke and S. Mertens, "Random numbers for large-scale distributed Monte Carlo simulations," *Phys. Rev. E*, vol. 75, no. 6, 066701, 2007.
[7] L. H. Martínez, et al., "LFSR generation for high test coverage and low hardware overhead," *IET Computers & Digital Techniques*, 2020.
[8] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, 2nd ed. Prentice-Hall, 2004.
[9] B. Shen, "On the structure of m-sequences and their decimations," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 611–614, 1970.
[10] S. W. Golomb, *Shift Register Sequences*. Laguna Hills, CA: Aegean Park Press, 1982.
[11] R. A. Rueppel, *Analysis and Design of Stream Ciphers*. Springer-Verlag, 1986.
[12] W. Song and T. Hsiao, "Parallel CRC computation using pipelined linear transformations," *IEEE Trans. Computers*, vol. 47, no. 7, pp. 848–857, 1998.
[13] M. G. Karpovsky and A. Taubin, "Efficient multi-bit parallel LFSR architectures," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 305–308, 1997.