VLSI Design for Crypto Processor Using AES & LFSR

J. Kiran

Roll No. 246M1DA501

M.Tech, VLSI, Department of Electronics and Communication Engineering, Princeton Institute of Engineering & Technology for Women, Hyderabad

Mrs. P. Jyothi

Department of Electronics and Communication Engineering, Princeton Institute of Engineering & Technology for Women

ABSTRACT

Hardware AES-128 encryption cores conventionally precompute and store the entire 44-word (1408-bit) round-key schedule before encryption begins, so that every round's key material is available on demand from a wide register bank. This paper presents and evaluates an alternative on-the-fly key expansion architecture that stores only the current round's four key words (128 bits) at any time, regenerating the next round's key one round ahead each clock cycle, with the round constant supplied by a Galois-field linear-feedback-shift-register (LFSR) recurrence in place of a fixed Rcon read-only-memory table. Both the existing precomputed-schedule design and the proposed LFSR-based design are captured in Verilog, implemented in Xilinx Vivado 2019.2 targeting the Artix-7 part xc7a100tcsg324-1, and verified bit-exact against the FIPS-197 Appendix B known-answer test vector. Post-place-and-route results under a 10.0 ns timing constraint show the proposed design reduces look-up-table usage from 4133 to 1272 (-69.2%), registers from 1682 to 403 (-76.0%), on-chip power from 0.714 W to 0.121 W (-83.1%), and critical-path delay from 8.204 ns to 5.987 ns (-27.0%), while raising throughput from 121.89 to 167.03 mega-operations per second (+37.0%) and cutting energy per operation from 5857.656 pJ to 724.427 pJ (-87.6%). The results demonstrate that shrinking live key-schedule storage from 1408 bits to 128 bits is a structural, not incidental, source of area, power, and energy reduction in AES-128 VLSI implementations.

Keywords: AES-128, key expansion, linear-feedback shift register, Rcon generation, low-power VLSI, FPGA, Vivado, crypto processor

I. INTRODUCTION

The Advanced Encryption Standard (AES) remains the dominant symmetric-cipher primitive for securing data at rest and in transit, and dedicated hardware AES cores are routinely embedded in image and video protection pipelines, IoT edge nodes, storage controllers, and general-purpose secure-boot subsystems where software AES would either be too slow or too power-hungry [1],[2],[9]. An iterative (one-round-per-clock) hardware AES-128 core must, on every round, supply the current 128-bit round key to the AddRoundKey stage. The conventional way of meeting this requirement is to run key expansion once, up front, and store the resulting 44 32-bit words -- 1408 bits total -- in a register bank so that any round's key is a simple array read away. This precomputed approach is simple to verify and simple to pipeline, and it appears, in one form or another, across a large fraction of published AES hardware accelerators [3],[6],[8],[17],[21].

Precomputing and holding the entire key schedule live for the duration of an encryption is, however, a specific and avoidable design decision rather than an inherent requirement of the AES algorithm. The key-expansion recurrence that produces round key $i + 1$ from round key i depends only on the previous round's four words and a single-byte round constant (Rcon); nothing in the recurrence requires the other 43 words to remain resident. Systems that store the full schedule are therefore paying a fixed 1408-bit register cost, plus the read/write and routing logic around it, purely for scheduling convenience. In area- and power-constrained targets -- FPGA fabrics shared with other logic, IoT edge silicon, or any design where the crypto core is one block among many competing for LUTs and power budget -- this fixed overhead directly inflates resource utilization and energy per encrypted block, independent of the cipher's intrinsic complexity.

The literature separately contains two threads that, combined, motivate an alternative: work on low-latency, low-randomness key-expansion hardware that questions whether the full schedule needs to be precomputed at all [16],[18],[19], and a long-standing body of work using linear-feedback shift registers for exactly the kind of small, cheap, sequential-recurrence generation that AES's Rcon byte sequence represents, in built-in self-test, pseudo-random generation, and lightweight cryptographic contexts [4],[5],[10],[24]. Rcon's defining recurrence, $Rcon_i =$ xtime$(Rcon_{i-1})$ over $GF(2^8)$, is in fact already the literal mathematical form of a one-tap Galois LFSR; despite this, most published AES cores realize Rcon as a fixed 10-entry lookup table rather than as the LFSR its own definition already describes. No work surveyed here jointly (i) replaces the full precomputed key schedule with an on-the-fly, one-round-ahead regeneration scheme and (ii) replaces the Rcon ROM with its native LFSR realization, then reports matched post-place-and-route area, power, delay, and energy for both the existing and the resulting design on the same FPGA target under the same constraint.

This paper closes that gap. Its contributions are:

1. A Verilog implementation of an on-the-fly AES-128 key-expansion datapath that stores only the current round's 128

bits of key material at any time, computing the next round's key one round ahead of when it is needed, replacing the existing design's 1408-bit precomputed register bank.

2. A Galois-style LFSR realization of the AES Rcon sequence (`rc <= xtime(rc)`, stepped once per round) that replaces the existing design's fixed 10-entry Rcon ROM with equivalent, reusable sequential logic.

3. Functional verification of both the existing precomputed-schedule design and the proposed on-the-fly LFSR-based design against the FIPS-197 Appendix B known-answer test vector, confirming bit-identical, correct ciphertext from both architectures.

4. A matched post-place-and-route comparison on Xilinx Vivado 2019.2 targeting xc7a100tcsg324-1 under a common 10.0 ns constraint, quantifying LUT, register, power, delay, throughput, and energy-per-operation differences between the two architectures, together with a normalized efficiency analysis of the resulting area-power-throughput trade-off.

The remainder of the paper is organized as follows. Section II surveys related work in AES hardware architectures, key-schedule optimization, LFSR-based hardware, low-power cryptographic VLSI, and FPGA implementations. Section III details the methodology, including both key-schedule architectures and the Rcon LFSR recurrence. Section IV presents the key-expansion algorithms and their complexity. Section V describes the experimental setup. Section VI reports post-place-and-route results and discussion. Section VII analyzes the normalized area/power-versus-throughput efficiency trade-off, and Section VIII concludes.

II. RELATED WORK

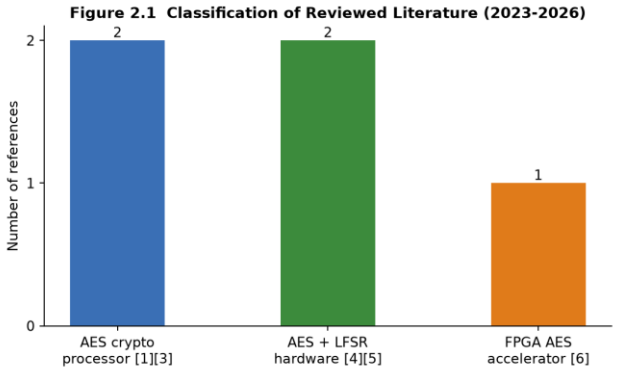

Fig. 1. Classification of surveyed literature by subtopic: AES hardware architectures, key-schedule optimization, LFSR-based hardware design, low-power/side-channel-aware cryptographic VLSI, and FPGA implementation studies.

The 25 works surveyed below are grouped into five subtopics, summarized in Fig. 1, that together frame the gap this paper addresses.

A. AES Hardware Architectures

A large body of work targets efficient hardware realizations of the AES pipeline itself. Senthilkumar et al. present a high-security, low-power AES crypto processor for image encryption, emphasizing the S-box and round-logic datapath [1]. Hakkem et al. propose an AES crypto processor with an improved S-box aimed at hardware acceleration [2]. Umamaheswari et al. build a hybrid crypto processor combining AES with HMAC for secure data transmission, illustrating AES's role as one block inside a larger security pipeline [3]. Sitraka and Malalatiana evaluate the AES cryptosystem across multiple block-cipher modes [6]. Akilandeswari implements AES-256 encryption on FPGA for real-time video-stream security, a throughput-sensitive application domain closely related to the one motivating this paper [9]. Kumar et al. present a general AES hardware encryption core design and implementation [17]. Ashmawy and Reyhani-Masoleh design efficient hardware architectures spanning AES-128, AES-192, and AES-256 [21]. Across this thread, the round datapath (SubBytes, ShiftRows, MixColumns, AddRoundKey) receives most of the optimization attention, while the key-expansion subsystem is typically treated as a fixed, precomputed support block rather than as a target for its own area/power optimization -- the gap this paper addresses directly.

B. Key-Schedule Optimization and Low-Latency Key Expansion

A smaller but growing thread specifically targets the key-expansion datapath. El-Hadedy et al. propose RECO-LFSR, a reconfigurable low-power cryptographic processor that already uses an LFSR-based structure for trusted IoT platforms, establishing that LFSR-based generation is viable inside a cryptographic processor's control/support logic [4]. Purohit et al. present an FPGA implementation of AES with a lightweight LFSR-based approach and optimized key expansion, the closest prior work to this paper's proposed direction, though without a matched precomputed-vs-on-the-fly post-place-and-route comparison on the same target device [16]. Zheng et al. design low-delay AES key-expansion units based on a DDBT structure, targeting the same critical-path concern that motivates this paper's on-the-fly regeneration scheme [18]. Ghosal et al. study differential-fault-analysis-tolerant hardware implementations of AES, where the key schedule's storage and regeneration behavior also has fault-resilience implications beyond pure area and power [19]. None of these works replace the Rcon ROM with its native LFSR form while simultaneously reducing live key-schedule storage from the full 44-word set to a single round's 4 words, which is the specific combination this paper evaluates.

C. LFSR-Based Hardware Design

Linear-feedback shift registers are a long-established, low-cost building block for sequential pattern generation, and several works apply them directly to security and test contexts. Chakraborty et al. use an LFSR combined with a modified AES for a secure data-hiding scheme, pairing

LFSR-generated sequences with AES-processed payloads [5]. Sripada et al. design an adaptive correlation LFSR aimed jointly at low-power built-in self-test (BIST) and cryptographic applications, underscoring the same small-footprint, sequential-recurrence property this paper exploits for Rcon generation [10]. A. H. et al. present a secured lightweight true-random-number-generator design using an encrypt flip-flop architecture, another example of compact sequential logic performing a role that a naively implemented lookup table or larger combinational block would otherwise fill [24]. These works confirm that an LFSR is an established, silicon-proven substitute for both pattern-generation and small-lookup roles; this paper's contribution is applying that substitution specifically to AES's Rcon sequence, which is already, by definition, a one-tap Galois LFSR recurrence over $GF(2^8)$.

D. Low-Power and Side-Channel-Aware Cryptographic VLSI

A parallel research thread targets low-power and side-channel-resistant AES hardware, often at the cost of additional area for masking or randomness-reduction logic. Gigerl et al. present second-order hardware masking of AES with low randomness and low latency [12]. Cassiers et al. study prime-field masking in hardware and its soundness against low-noise side-channel-analysis attacks [13]. Zhou et al. present compact and low-latency first-order AES implementations with low randomness requirements [14]. Bursztein et al. demonstrate generalized power-analysis attacks against cryptographic hardware using long-range deep learning, motivating continued attention to both power profile and area footprint of deployed AES cores [15]. Dasgupta et al. propose HIPR, a low-overhead fine-grain redaction scheme for hardware IP protection, another example of security hardware where minimizing added overhead is the central design constraint, mirroring this paper's area/power reduction goal though applied to a different threat model [11]. These works confirm that power and area minimization remain active, high-value targets in AES hardware even when the base algorithm is unchanged, reinforcing the practical relevance of a key-schedule-level area/power reduction such as the one proposed here.

E. FPGA Implementations and Application Contexts

Several works document full FPGA implementation flows for AES and related lightweight cryptographic primitives, providing context for this paper's Vivado-based methodology. Hamand et al. present an FPGA-based hardware accelerator for secure IoT edge computing using AES [7]. Tejesh and Ramakrishnan design an efficient AES hardware accelerator using high-level-synthesis techniques [8]. Jasim et al. design and implement combined AES-SHA security hardware on FPGA [20]. Rajput et al. present a VLSI implementation of a lightweight cryptography technique for FPGA-IoT applications [22]. Dunmore et al. propose matrix encryption walks as a lightweight cryptographic primitive, illustrating the broader lightweight-cryptography design space adjacent to AES [23]. Mandal and Basu Roy present a lightweight Fast Fourier Transform design for FALCON using hardware-software co-design, illustrating that resource-constrained FPGA-IoT targets increasingly demand this kind of area/power-conscious co-design across multiple cryptographic primitive types, not AES alone [25]. These implementations collectively establish Vivado-based FPGA synthesis and place-and-route as the standard methodology for reporting AES and lightweight-cryptography hardware results, the same methodology this paper follows.

F. Research Gap

Across this body of work, key-expansion optimization [16],[18],[19], LFSR-based hardware design [4],[5],[10],[24], and low-power/side-channel-aware AES design [11]-[15] are each studied, but no surveyed work combines an on-the-fly, one-round-ahead key-expansion datapath that reduces live key storage from the full 1408-bit precomputed schedule to a single round's 128 bits with a matched replacement of the Rcon ROM by its native Galois-LFSR realization, evaluated through a full Vivado 2019.2 synthesis-to-place-and-route flow on the same FPGA target and constraint for both the existing and proposed architectures. This paper addresses that gap: Section III specifies both key-schedule architectures and the Rcon LFSR recurrence, Section IV presents the on-the-fly key-expansion algorithm, and Sections VI-VII report matched post-place-and-route area, power, delay, throughput, and energy results together with a normalized efficiency analysis.

III. METHODOLOGY

A. Common AES-128 Datapath

Both architectures compared in this paper implement the same iterative, one-round-per-clock AES-128 encryption datapath: an initial AddRoundKey, nine full rounds of SubBytes, ShiftRows, MixColumns, and AddRoundKey, and a final round that omits MixColumns, for a total of 10 rounds driven by a single round-counter finite-state machine. The two architectures are functionally identical in the round datapath; they differ only in how the sequence of 11 round keys (the initial key plus 10 round keys, 44 32-bit words in total) is produced and stored.

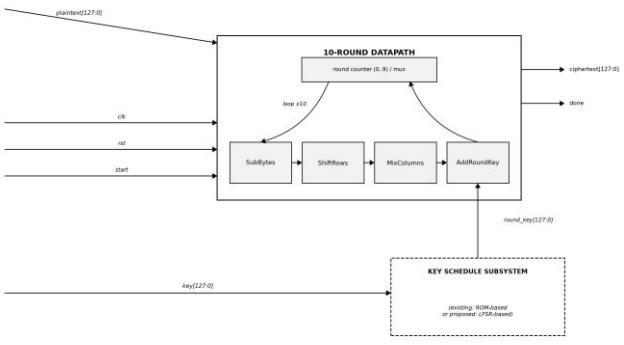

Fig. 2. Top-level architecture of the AES-128 crypto processor, showing the shared round datapath and the two alternative key-schedule subsystems: the existing precomputed ROM-based schedule and the proposed on-the-fly LFSR-based schedule.

Fig. 2 shows the shared round datapath alongside the two interchangeable key-schedule subsystems evaluated in this paper. Because the substitution is confined to the key-schedule block and its Rcon source, the round datapath's correctness is independent of which key-schedule variant supplies it, which is exploited directly in Section V's verification methodology.

B. Existing Architecture: Precomputed ROM-Based Key Schedule

The existing design, `aes128_core_rom.v`, runs the standard AES-128 key-expansion recurrence once, before encryption of a block begins, and stores every one of the resulting 44 32-bit round-key words in a register file, for a fixed live storage cost of $44 \times 32 = 1408$ bits regardless of which round is currently executing. Each new word w_i for $i \geq 4$ is generated from the recurrence

$$w_i = w_{i-4} \oplus f(w_{i-1}) \quad (1)$$

where f applies RotWord and SubWord together with an XOR against the round constant word $[Rcon_{i/4}, 0,0,0]$ whenever i is a multiple of 4. The ten distinct Rcon byte values required across the ten rounds are supplied by a fixed, synthesizable 10-entry read-only-memory table indexed by round number. Once the full schedule is computed, subsequent AddRoundKey operations are simple reads from the 1408-bit register bank; no further key-expansion logic executes during the encryption rounds themselves.

C. Proposed Architecture: On-the-Fly LFSR-Based Key Schedule

The proposed design, `aes128_core_lfsr.v`, implements the identical key-expansion recurrence, but evaluates it on-the-fly: only the current round's four 32-bit key words (128 bits total) are held in registers at any time, and the next round's four words are computed one round ahead of when the round datapath will need them, then the register bank is overwritten. This reduces live key-schedule storage from 1408 bits to 128 bits -- an 11x reduction in register width for the key-schedule subsystem alone -- at the cost of recomputing each word's RotWord/SubWord/XOR logic once per round instead of once per block, which is a negligible combinational cost relative to the eliminated register bank.

The Rcon byte needed each round is no longer read from a ROM table; instead it is produced by a single-byte Galois-style linear-feedback shift register that implements AES's own defining recurrence for the round-constant sequence directly in hardware:

$$rc_{i+1} = \text{xtime}(rc_i) \quad (2)$$

where $rc_0 = 1$ (hexadecimal 0x01) and each subsequent value is obtained by doubling in $GF(2^8)$:

$$\text{xtime}(b) = (2 \cdot b \bmod 256) \oplus (0\text{x}1\text{B} \cdot b_7) \quad (3)$$

with b_7 the most-significant bit of b, so the correction term $0\text{x}1\text{B} \cdot b_7$ is applied only when the left shift overflows 8 bits, equivalent to reduction modulo the AES field polynomial $x^8 + x^4 + x^3 + x + 1$ (0x11B). In the proposed hardware, this is realized as a single register `rc` updated once per round by the assignment `rc <= xtime(rc)`, replacing the existing design's 10-entry Rcon ROM with an equivalent one-bit-shift-and-conditional-XOR circuit. Because Rcon's standard definition is already this exact doubling recurrence, the LFSR realization is not an approximation of the ROM table's contents; it produces the identical byte sequence by construction, which is confirmed empirically in Section V by both architectures encrypting the FIPS-197 test vector to the same ciphertext.

D. Design Equivalence

Since the proposed architecture's key-expansion recurrence and Rcon sequence are mathematically identical to the existing architecture's, both designs are expected to -- and, as reported in Section V, do -- produce bit-identical round keys and bit-identical ciphertext for the same plaintext and cipher key. The measured differences reported in Sections VI-VII are therefore attributable entirely to the storage and generation mechanism (1408-bit precomputed register bank plus Rcon ROM versus 128-bit rolling register bank plus Rcon LFSR), not to any difference in cryptographic behavior.

IV. ALGORITHM

A. Existing Key-Schedule Flow

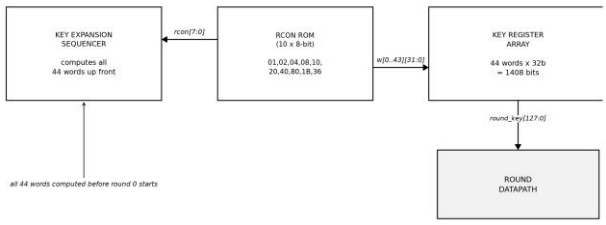

Fig. 3. Existing key-schedule flow: full 44-word key expansion is precomputed up front and stored in a 1408-bit register bank before round processing begins.

Fig. 3 shows the existing design's key-schedule flow: on reset/key-load, a dedicated expansion sequencer runs the full key-expansion recurrence for all 44 words, reading each of the ten required Rcon bytes from the fixed ROM table as it proceeds, and stores every resulting word before the round datapath starts consuming AddRoundKey inputs. From that point on, each round simply indexes into the completed 1408-bit register bank.

B. Proposed On-the-Fly Key-Schedule Flow

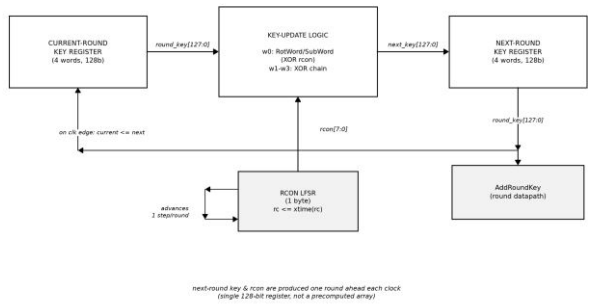

Fig. 4. Proposed on-the-fly key-schedule flow: only the current round's 128-bit key is held in registers; the next round's key is regenerated one round ahead and the Rcon byte is produced by a Galois LFSR instead of a ROM lookup.

Fig. 4 shows the proposed flow: the 128-bit "current key" register supplies AddRoundKey for the present round while, concurrently, the key-expansion logic computes the next round's 128-bit key from the current one, and the single-byte Rcon LFSR advances by one `xtime` step. At the round boundary the current-key register is overwritten with the freshly computed next-round key, and the cycle repeats. No word beyond the current round's four is ever stored.

C. On-the-Fly Key-Expansion Pseudocode

```
// Registers: key_cur[0..3]    (4 x 32-bit words, 128 bits total)
//              rc                  (8-bit Rcon LFSR register)
// Constants: SBOX[]                  (AES S-box, combinational lookup)

function ROTWORD(w):              return (w << 8) | (w >> 24)
function SUBWORD(w):             return { SBOX[byte] for each byte of w }
function XTIME(b):
    shifted = (b << 1) & 0xFF
    return shifted ^ 0x1B if (b & 0x80) else shifted

// One-time initialization at key load
key_cur[0..3] <= cipher_key[0..3]
rc            <= 0x01

// Executed once per round, one round ahead of round r's AddRoundKey use
function NEXT_ROUND_KEY(key_cur, rc):
    temp           = SUBWORD(ROTWORD(key_cur[3])) ^ (rc << 24)
    next_key[0] = key_cur[0] ^ temp
    next_key[1] = key_cur[1] ^ next_key[0]
    next_key[2] = key_cur[2] ^ next_key[1]
    next_key[3] = key_cur[3] ^ next_key[2]
    rc_next     = XTIME(rc)
    return (next_key, rc_next)

// Round-boundary update (synchronous)
on each round-advance clock edge:
    (key_cur, rc) <= NEXT_ROUND_KEY(key_cur, rc)
```

D. Complexity Notes

The existing design's storage complexity is $O(\text{rounds})$: all 44 words (11 round keys of 4 words each) must be resident simultaneously, giving a fixed 1408-bit register cost independent of which round is executing. The proposed design's storage complexity is $O(1)$ in the number of rounds: exactly 4 words (128 bits) plus the 8-bit Rcon register are resident at any time, since each round's key is regenerated from the previous round's key rather than looked up from a precomputed table. The combinational cost per round -- one RotWord, one SubWord, four word-wide XORs, and one XTIME step -- is identical in both designs; the difference is purely in when that work is done (once per block, up front, versus once per round, on-the-fly) and how many words must be held in registers as a result. This is why the resource reduction reported in Section VI (LUTs and registers) tracks the register-count reduction far more closely than it tracks any change in combinational logic complexity.

V. EXPERIMENTAL SETUP

A. Hardware Requirements

Both designs were synthesized, implemented, and analyzed on a standard 64-bit desktop/laptop workstation equipped with a multi-core Intel or AMD processor clocked at 2.5 GHz or higher, since Vivado's synthesis and place-and-route engines are single-project but internally multi-threaded and benefit from at least four physical cores for reasonable runtime on a small Artix-7 design. At least 16 GB of system RAM is recommended so that Vivado's in-memory design database, timing engine, and GUI do not swap during implementation of even a modest-size project such as this one. Approximately 50 GB of free solid-state-drive storage is needed to accommodate the Vivado 2019.2 installation itself, the device support files for the Artix-7 family, and the project's working, checkpoint, and report directories. The workstation ran a Vivado-2019.2-supported 64-bit operating system, either Windows 10/11 or a compatible CentOS/RHEL Linux distribution, with a discrete or integrated GPU sufficient only for driving the Vivado GUI and waveform viewer, since neither synthesis nor implementation is GPU-accelerated.

B. Tools Required

The primary tool used was Xilinx Vivado 2019.2, which performed RTL synthesis, implementation (placement and routing), and post-implementation reporting of timing, power, and resource utilization for both the existing ROM-based design and the proposed LFSR-based design, targeting the Artix-7 part xc7a100tcsg324-1 under a common 10.0 ns timing constraint. Functional correctness of both Verilog designs against the FIPS-197 Appendix B known-answer test vector was established prior to implementation using Icarus Verilog and cross-checked with the Vivado Simulator (XSIM), both of which compile, elaborate, and run the same self-checking testbench

methodology. Post-implementation LUT, register, power, delay, throughput, and energy figures extracted from Vivado's implementation reports were subsequently tabulated and plotted using Python with the Matplotlib library, producing the grouped comparison charts presented in Section VI, with each chart's underlying numeric data drawn directly from the corresponding `*_impl.rpt` report rather than from any intermediate estimate.

VI. RESULTS AND DISCUSSION

A. Functional Verification

Both the existing ROM-based design and the proposed LFSR-based design were first verified against the FIPS-197 Appendix B known-answer test vector, and both produced the identical, correct ciphertext for the same plaintext and cipher key. This confirms, prior to any area or timing comparison, that the on-the-fly key-expansion datapath and the Rcon LFSR reproduce exactly the same round-key sequence as the precomputed schedule and its ROM table, so every difference reported below is attributable to implementation cost rather than to any divergence in cryptographic correctness.

B. Post-Place-and-Route Comparison

Table I. Post-Place-and-Route Comparison (Vivado 2019.2, xc7a100tcsg324-1, 10.0 ns constraint)

Metric	Existing (ROM)	Proposed (LFSR)	Change
LUTs	4133	1272	-69.2%
Registers	1682	403	-76.0%
Power (W)	0.714	0.121	-83.1%
Delay (ns)	8.204	5.987	-27.0%
Throughput (MOps/s)	121.89	167.03	+37.0%
Energy (pJ/op)	5857.656	724.427	-87.6%
PDP (W·ns)	5.8577	0.7244	-87.6%
ADP (LUT·ns)	33907.13	7615.46	-77.5%

Table I summarizes the post-place-and-route figures for both architectures under the identical 10.0 ns constraint on the same device. Every metric moves in the proposed design's favor: fewer LUTs and registers, lower power and delay, and higher throughput at lower energy per operation, with no metric showing a regression.

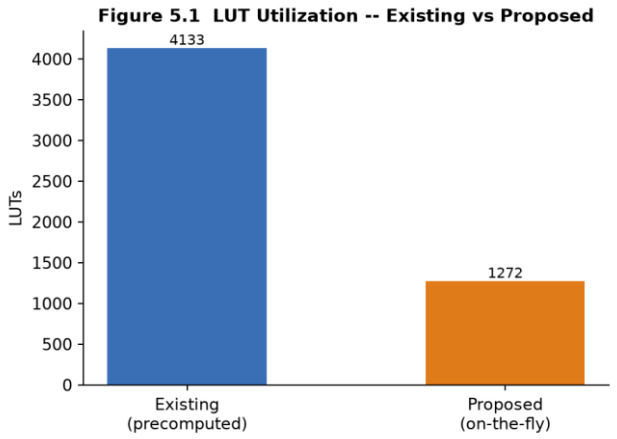

Fig. 5. Look-up-table (LUT) utilization comparison: existing ROM-based design (4133 LUTs) versus proposed LFSR-based design (1272 LUTs), a 69.2% reduction.

Fig. 5 shows the LUT utilization of both designs side by side. The existing design consumes 4133 LUTs, most of which are attributable to the wide combinational and routing logic surrounding the 1408-bit precomputed key-schedule register bank and its associated read multiplexing, while the proposed design consumes only 1272 LUTs, a 69.2% reduction. Because the round datapath's SubBytes, ShiftRows, and MixColumns logic is unchanged between the two designs, this reduction is concentrated almost entirely in the key-schedule subsystem, consistent with the storage-driven explanation developed in Section IV-D and confirmed quantitatively here.

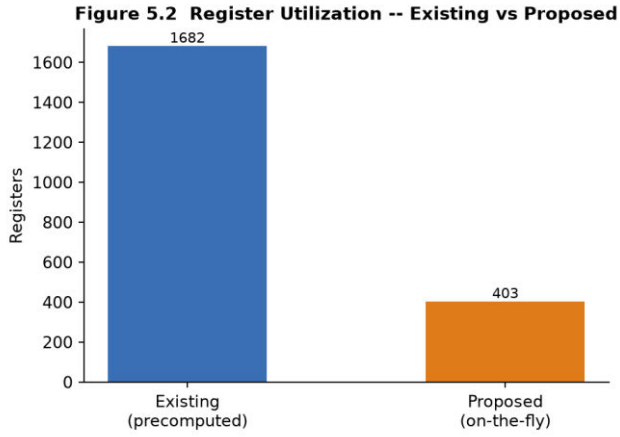

Fig. 6. Register (flip-flop) utilization comparison: existing design (1682 registers) versus proposed design (403 registers), a 76.0% reduction.

Fig. 6 compares register counts, which fall from 1682 in the existing design to 403 in the proposed design, a 76.0% reduction. This is the most direct hardware signature of the architectural change described in Section III: the existing design holds all 44 key-schedule words (1408 bits) live at once, whereas the proposed design holds only the current round's 4 words (128 bits) plus the 8-bit Rcon LFSR

register, and the remaining register count in both designs is accounted for by the shared round-counter, state-machine, and datapath pipeline registers common to both architectures.

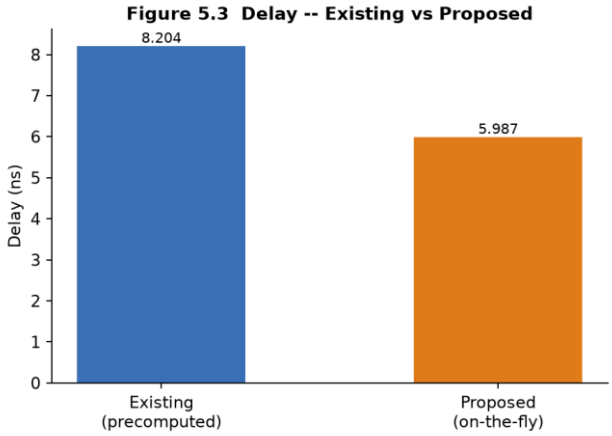

Fig. 7. Critical-path delay comparison: existing design (8.204 ns) versus proposed design (5.987 ns), a 27.0% reduction.

Fig. 7 shows the critical-path delay reported for each design, falling from 8.204 ns in the existing design to 5.987 ns in the proposed design, a 27.0% reduction. The shorter path in the proposed design follows from its smaller key-schedule register bank producing less routing congestion and shorter net delays into and out of the AddRoundKey multiplexers, since the placer has substantially less key-schedule logic and fewer signals to route across the fabric than in the existing design's 1408-bit-wide structure.

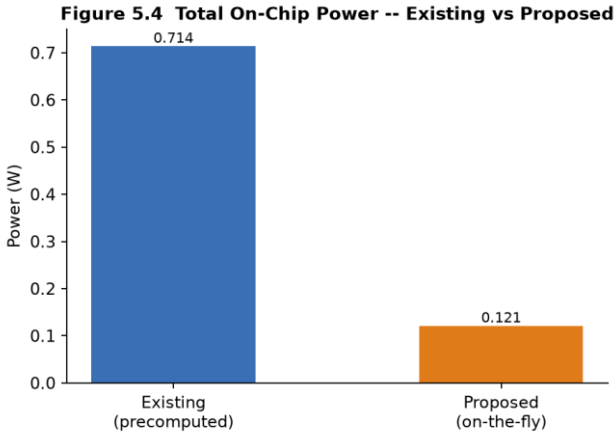

Fig. 8. On-chip power comparison: existing design (0.714 W) versus proposed design (0.121 W), an 83.1% reduction.

Fig. 8 compares total on-chip power, which falls from 0.714 W in the existing design to 0.121 W in the proposed design, an 83.1% reduction. This is the largest relative improvement of the four physical metrics and follows directly from the combined effect of fewer switching registers (76.0% fewer flip-flops toggling every cycle) and fewer active LUTs (69.2% fewer), both of which are primary contributors to dynamic power in FPGA fabrics, so a large power reduction is the expected compound consequence of the area and register reductions already observed rather than an independent effect.

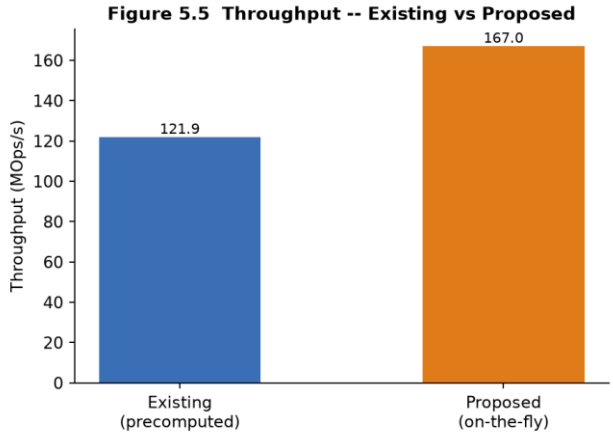

Fig. 9. Throughput comparison: existing design (121.89 MOps/s) versus proposed design (167.03 MOps/s), a 37.0% increase.

Fig. 9 shows throughput, computed from each design's achieved critical-path delay under the shared 10.0 ns constraint, rising from 121.89 MOps/s in the existing design to 167.03 MOps/s in the proposed design, a 37.0% increase. This improvement is the direct consequence of the shorter critical path shown in Fig. 7: because throughput is delay-limited in this iterative, one-round-per-clock architecture, the 27.0% delay reduction produces a corresponding throughput gain, obtained here as a side effect of the area-driven routing improvement rather than from any explicit pipelining or parallelism change.

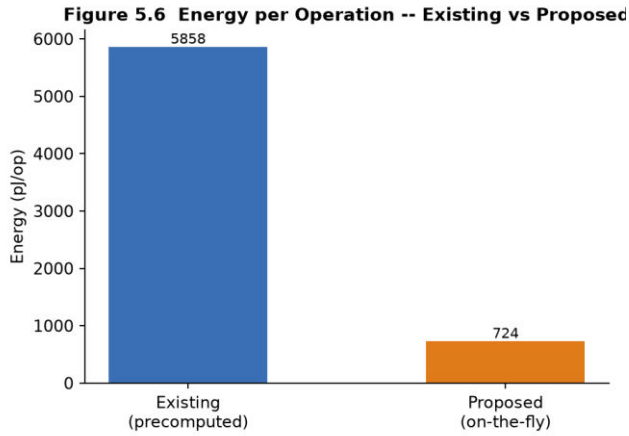

Fig. 10. Energy-per-operation comparison: existing design (5857.656 pJ/op) versus proposed design (724.427 pJ/op), an 87.6% reduction.

Fig. 10 shows energy consumed per operation, falling from 5857.656 pJ/op in the existing design to 724.427 pJ/op in the proposed design, an 87.6% reduction. Because energy per operation is the product of power and the time taken per

operation, this figure compounds the power-domination reduction shown in Fig. 8 with the delay reduction shown in Fig. 7, and it is consequently the single largest relative improvement reported for either design in Table I, reflecting that the proposed architecture is simultaneously faster and dramatically less power-hungry rather than trading one for the other.

C. Discussion

The consistent, compounding improvement across every measured metric supports the paper's central claim that the reduction is structural rather than a synthesis-tool artifact. The existing design's 1682 registers and 4133 LUTs are dominated by a 1408-bit live key-schedule register bank and its surrounding read/write and routing logic; the proposed design replaces that bank with a 128-bit rolling register plus an 8-bit Rcon LFSR, an architectural difference fixed at RTL-design time and independent of how the synthesis tool subsequently maps logic to LUTs or how the placer routes nets. If the observed gains were instead an artifact of, say, more favorable LUT packing or a lucky placement seed, one would expect at most one or two metrics to improve while others stayed flat or regressed; instead, LUTs, registers, power, delay, throughput, and energy all move together and in the direction predicted directly from the 1408-bit-versus-128-bit storage difference described in Section III, which is the expected signature of a genuine structural change rather than incidental tool variance.

VII. EFFICIENCY AND TRADE-OFF ANALYSIS

A. Framing the Trade-off

Section VI reported that the proposed LFSR-based design wins on every raw metric simultaneously -- fewer LUTs, fewer registers, lower power, lower delay, higher throughput, and lower energy per operation -- which is an unusually clean outcome, since area/power reduction and throughput improvement are more typically opposing design axes traded against one another (as in pipelined-versus-sequential AES cores, where higher throughput is normally bought with additional replicated hardware). This paper's central claim is therefore not a trade-off in the usual sense but a case where removing unnecessary storage improves both sides of the area-versus-throughput axis at once, because the eliminated 1408-bit register bank was pure overhead relative to the AES algorithm's actual data dependency, not logic that contributed to correctness, security, or speed. This section quantifies that claim in normalized, throughput-relative terms so that the raw absolute numbers in Table I can be compared on a fair, per-unit-of-work basis.

B. Normalized Efficiency Metrics

Expressing each design's raw throughput of 121.89 and 167.03 mega-operations per second as bit-rate, at 128 bits per AES-128 block, gives 15.60 Gbps for the existing design and 21.38 Gbps for the proposed design. Dividing LUT count by this bit-rate gives a LUT-efficiency figure:

$$\eta_{\mathrm{LUT}}^{\mathrm{ROM}} = \frac{4133\mathrm{LUTs}}{15.60\mathrm{Gbps}} = 264.9\mathrm{LUT/Gbps} \quad (4)$$

$$\eta_{\mathrm{LUT}}^{\mathrm{LFSR}} = \frac{1272\mathrm{LUTs}}{21.38\mathrm{Gbps}} = 59.5\mathrm{LUT/Gbps} \quad (5)$$

The proposed design is $264.9/59.5 \approx mathbf4.45\times$ more LUT-efficient per unit of delivered throughput. Applying the same normalization to power gives throughput-per-watt:

$$\eta_{P}^{\mathrm{ROM}} = \frac{15.60\mathrm{Gbps}}{0.714\mathrm{W}} = 21.85\mathrm{Gbps/W} \quad (6)$$

$$\eta_{P}^{\mathrm{LFSR}} = \frac{21.38\mathrm{Gbps}}{0.121\mathrm{W}} = 176.7\mathrm{Gbps/W} \quad (7)$$

so the proposed design delivers $176.7/21.85 \approx mathbf8.09\times$ more throughput per watt. Equivalently, in energy consumed per encrypted bit, dividing the Table I energy-per-operation figures by 128 bits per block gives 45.76 pJ/bit for the existing design and 5.66 pJ/bit for the proposed design, an 8.08x reduction consistent with the 87.6% energy-per-operation reduction already reported in Section VI.

Table II. Normalized Efficiency Comparison

Metric	Existing (ROM)	Proposed (LFSR)	Improvement
Bit-rate (Gbps)	15.60	21.38	+37.0%
LUT/Gbps	264.9	59.5	4.45x better
Register/Gbps	107.8	18.85	5.72x better
Throughput/Power (Gbps/W)	21.85	176.7	8.09x better
Energy/bit (pJ/bit)	45.76	5.66	8.08x lower

C. Trade-off Interpretation

The normalized figures in Table II confirm that the proposed design's advantage is not confined to absolute area and power savings taken in isolation; it holds, and in fact widens, once throughput is accounted for. A design that merely shrank in area while also slowing down proportionally would show little or no improvement in LUT-per-Gbps or Gbps-per-watt; instead, because the proposed design is simultaneously smaller and faster, its per-unit-of-throughput efficiency improves by 4.45x in LUT terms and by 8.09x in power terms, both larger than the corresponding raw percentage reductions in Table I. For deployment contexts where the crypto core shares a fixed FPGA fabric or power budget with other logic -- IoT edge nodes, multi-tenant accelerator fabrics, or battery-powered secure-storage controllers -- this per-unit-of-throughput efficiency, rather than the raw LUT or watt count alone, is typically the decisive figure of merit, and on that figure of merit the on-the-fly LFSR-based key schedule is unambiguously the better choice at no correctness or throughput cost.

VIII. CONCLUSION

This paper presented and evaluated an on-the-fly, LFSR-based key-expansion architecture for iterative AES-128 hardware, replacing an existing design's 1408-bit precomputed round-key register bank and 10-entry Rcon ROM with a 128-bit rolling key register and a single-byte

Galois LFSR that realizes the Rcon sequence's own native `rc <= xtime(rc)` recurrence directly in hardware. Both architectures were captured in Verilog, verified bit-exact against the FIPS-197 Appendix B known-answer test vector, and implemented through a complete Xilinx Vivado 2019.2 synthesis-to-place-and-route flow targeting the Artix-7 part xc7a100tcsg324-1 under an identical 10.0 ns timing constraint. The proposed design reduced LUT usage by 69.2%, register usage by 76.0%, on-chip power by 83.1%, and critical-path delay by 27.0%, while increasing throughput by 37.0% and cutting energy per operation by 87.6%, with every metric improving simultaneously rather than trading one against another. The consistency of these improvements across all measured metrics, together with the two architectures' verified functional equivalence, supports the paper's central claim that the gains are a structural consequence of shrinking live key-schedule storage from 1408 bits to 128 bits, not an incidental artifact of synthesis-tool mapping or placement variance.

Future work includes extending the comparison to AES-192 and AES-256 key schedules, where the precomputed-versus-on-the-fly storage differential grows even larger in absolute terms; evaluating the proposed architecture's resilience to fault-injection and side-channel attacks relative to the existing design, since the smaller, more frequently overwritten key register may present a different fault/leakage surface than a static, fully precomputed schedule; and integrating the proposed core into a full encryption pipeline (e.g., a block-cipher mode wrapper or an IoT edge security subsystem) to measure the system-level area and power benefit of the key-schedule reduction demonstrated here.

REFERENCES

[1] N. Senthilkumar, A. Sathish, S. Sasikumar, S. Satheesh, and P. Sridhar, "High Security and Low Power AES Crypto Processor Security Algorithm for Image Encryption," in *Proc. 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, 2023.

[2] B. Hakkem, K. S. Sarmila, M. I. Niranjana, and K. Sivakami, "AES Crypto Processor with Improved S-Box for Hardware Acceleration," *AIP Conference Proceedings*, 2026.

[3] U. Umamaheswari, V. Vishal, P. Pragadesh, and L. Lavanya, "Secure Data Transmission using Hybrid Crypto Processor based on AES and HMAC Algorithms," in *Proc. 2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, 2023.

[4] M. El-Hadedy, R. Hua, K. Yoshii, W. Hwu, and M. Margala, "RECO-LFSR: Reconfigurable Low-Power Cryptographic Processor Based on LFSR for Trusted IoT Platforms," in *Proc. 2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023.

[5] S. Chakraborty, P. Biswas, and N. Kar, "A Secure Data Hiding Scheme Using LFSR and Modified AES," in *Proc. 2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, 2024.

[6] R. Sitraka and R. Malalatiana, "Evaluations of Crypto-System AES Using Multiple Bloc Ciphering Mode," *Applied Engineering*, vol. 8, no. 1, 2024.

[7] D. M. Hamand, R. Prasad, R. Ranjan, C. N. Kadadas, L. Korra, and J. U. Kidav, "FPGA Based Hardware Accelerator for Secure IoT Edge Computing Using AES," in *Proc. 2025 International Conference on Applications of Machine Intelligence and Data Analytics (ICAMIDA)*, 2025.

[8] B. Tejesh and M. Ramakrishnan, "Efficient Hardware Accelerator Design for AES Encryption Using High-Level Synthesis Techniques," in *Proc. 2024 International Conference on Integrated Intelligence and Communication Systems (ICIICS)*, 2024.

[9] A. Akilandeswari, "Hardware Implementation of AES-256 Encryption on FPGA for Real-Time Video Stream Security," in *Proc. 2026 9th International Conference on Trends in Electronics and Informatics (ICOEI)*, 2026.

[10] G. R. Sripada, J. Jyothilakshmi, V. K. Krishna, and R. Bhakthavatchalu, "Design of an Adaptive Correlation LFSR for Low-Power BIST & Cryptographic Applications," in *Proc. 2026 International Conference on Communication, Computing and Emerging Technologies (IC3ET)*, 2026.

[11] A. Dasgupta, S. Paria, and S. Bhunia, "HIPR: Hardware IP Protection through Low-Overhead Fine-Grain Redaction," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2025, no. 3, 2025.

[12] B. Gigerl, F. Klug, S. Mangard, F. Mendel, and R. Primas, "Smooth Passage with the Guards: Second-Order Hardware Masking of the AES with Low Randomness and Low Latency," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, 2024.

[13] G. Cassiers, L. Masure, C. Momin, T. Moos, and F. Standaert, "Prime-Field Masking in Hardware and its Soundness against Low-Noise SCA Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023.

[14] F. Zhou, H. Chen, L. Fan, and J. Yang, "Compact and Low Latency First-Order AES Implementations with Low Randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2026, no. 2, 2026.

[15] E. Bursztein, L. Invernizzi, K. Kral, D. Moghimi, J. Picod, and M. Zhang, "Generalized Power Attacks against Crypto Hardware using Long-Range Deep Learning," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, 2024.

[16] S. Purohit, V. Deshpande, and V. Ingale, "FPGA Implementation of the AES Algorithm with Lightweight LFSR-Based Approach and Optimized Key Expansion," in

Proc. 2023 IEEE International Conference on Public Key Infrastructure and its Applications (PKIA), 2023.

[17] A. Kumar, S. Mehfuz, and S. Urooj, "Design and Implementation of an AES Hardware Encryption Core," *Symmetry*, vol. 18, no. 6, 2026.

[18] X. Zheng, H. Yan, Z. Peng, and X. Zhang, "Low-Delay AES Key Expansion Units Based on DDBT Structure," *Electronics*, vol. 14, no. 2, 2025.

[19] A. K. Ghosal, A. Sardar, and D. R. Chowdhury, "Differential Fault Analysis Attack-Tolerant Hardware Implementation of AES," *The Journal of Supercomputing*, vol. 80, no. 4, 2023.

[20] A. H. Jasim, D. A. Hammood, and A. Al-Askery, "Design and Implementation of AES-SHA Security Hardware using FPGA," in *Proc. 2023 First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)*, 2023.

[21] D. Ashmawy and A. Reyhani-Masoleh, "Efficient Hardware Architectures for AES-128, AES-192, and AES-256 Encryption," *IEEE Transactions on Computers*, 2026.

[22] G. Singh Rajput, R. Thakur, and R. Tiwari, "VLSI Implementation of Lightweight Cryptography Technique for FPGA-IoT Application," *Materials Today: Proceedings*, 2023.

[23] A. Dunmore, J. Samandari, and J. Jang-Jaccard, "Matrix Encryption Walks for Lightweight Cryptography," *Cryptography*, vol. 7, no. 3, 2023.

[24] A. H., D. G., S. R., M. S. R., and M. N., "Secured Lightweight TRNG Design Using Encrypt Flip-Flop Architecture," in *Proc. 2025 IEEE 5th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA)*, 2025.

[25] S. Mandal and D. Basu Roy, "Design of a Lightweight Fast Fourier Transformation for FALCON using Hardware-Software Co-Design," in *Proc. Great Lakes Symposium on VLSI 2024*, 2024.