INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 7,854 Volume 12, issue 11 (2025)

High-Performance FPGA-Based Cryptographic and Image-Processing Co-Design for Real-Time Embedded Systems: Architectures, Trade-offs, and Implementation Strategies

Dr. Mariana Ortega
Global Institute of Embedded Systems, University of Lisbon

Abstract

Background: Modern embedded systems for Internet-of-Things (IoT), autonomous sensing, and edge computing require tightly integrated solutions that deliver strong cryptographic protection and low-latency image/video processing on constrained hardware. Field-Programmable Gate Arrays (FPGAs) and System-on-Chip platforms combining ARM processors and programmable logic (SoC-FPGA) have emerged as the preferred substrates for these requirements due to their reconfigurability, parallelism, and energy-performance flexibility (Guerrieri et al., 2023; Tsai et al., 2023). However, co-designing high-throughput symmetric cryptography (notably AES) together with computationally heavy image/video processing and chaos-based primitives presents architectural, timing, and resource-allocation challenges that remain incompletely addressed in the literature (Li and Li, 2017; Visconti et al., 2020).

Objective: This article synthesizes the theoretical foundations and practical design strategies required to construct robust, low-latency FPGA-based systems capable of real-time AES encryption/decryption, chaos-based secure key generation, and accelerated image/video processing for embedded applications. It systematically surveys existing FPGA implementations, analyzes design trade-offs, and proposes a cohesive, extensible co-design methodology.

Methods: The study uses rigorous textual synthesis of peer-reviewed FPGA implementations, SoC integration examples, and high-level synthesis reports to derive an architecture-level methodology. The methodology covers modular AES pipelines, subpipelined and reconfigurable AES designs, chaos-based random/chaotic generators mapped to FPGA primitives, and image/video processing pipelines (including Harris corner detection and edge detection) implemented either with RTL or high-level synthesis. Performance and resource trade-offs are discussed through normalized metrics (throughput, latency, resource utilization, and energy considerations) derived from reported implementations (Li et al., 2017; Visconti et al., 2020; Rajasekar and Haridas, 2016).

Results: The synthesis identifies recurring design patterns that achieve throughput in the multi-Gigabit-per-second range for AES while maintaining low latency for video pipelines when carefully balancing pipeline depth, clock period, and resource duplication (Visconti et al., 2020; Li et al., 2023). Chaos-based generators provide compact entropy sources and can be integrated with lightweight ECC/TLS stacks for IoT coordinators (Elsayed et al., 2023; Bellemou et al., 2019). High-level synthesis offers rapid prototyping and maintainable code paths but sometimes trails hand-written RTL for area and peak throughput, necessitating selective RTL optimization in critical modules (Elsayed and Kayed, 2022; Sankar et al., 2023). Integration on Zynq-class SoC platforms enables flexible hardware/software partitioning for real-time fusion and control tasks (Yoon et al., 2016; Tsai et al., 2023).

Conclusions: A disciplined co-design strategy—combining reconfigurable subpipelined AES, chaos-based key generation, selective RTL optimizations, and HLS-accelerated image pipelines—yields systems capable of secure, real-time operation for many embedded applications. Future work should empirically validate energy models across device families and explore dynamic reconfiguration policies that trade security strength, throughput, and power in response to runtime requirements (Del-Valle-Soto et al., 2020; Visconti et al., 2020).

Keywords: FPGA co-design, AES pipeline, chaos-based key generator, SoC-FPGA, high-level synthesis, real-time image processing

INTRODUCTION

The modern deployment ecosystem for embedded devices increasingly demands high assurance, real-time processing, and minimal energy consumption simultaneously. Applications such as autonomous systems telemetry, secure sensor networks, real-time video analytics at the edge, and secure communication for autonomous vehicles necessitate hardware solutions that can sustain both cryptographic throughput and image/video processing performance without exceeding stringent latency or power budgets (Patil and Deshpande, 2025; Guerrieri et al., 2023). Field-Programmable Gate Arrays (FPGAs) and System-on-Chip variants that combine ARM cores with programmable logic (SoC-FPGAs) have become central to these designs because they permit architects to map compute-intensive kernels to parallelized logic while retaining software flexibility in embedded processors (Tsai et al., 2023; Yoon et al., 2016). Previous work has demonstrated the viability of per-task high-performance FPGA implementations—AES encryption/decryption with extreme pipelining (Li and Li, 2017; Visconti et al., 2020), chaos-based key generation implemented directly in programmable logic for compact and adaptive entropy sources (Elsayed et al., 2023; Abd El-Maksoud et al., 2020), and real-time image/video processing implemented via hardware-accelerated kernels and high-level synthesis (Ghodhbani et al., 2020; Ghodhbani et al., 2023). Despite these advances, the problem of co-design—simultaneously satisfying cryptographic throughput, secure key generation robustness, and real-time image/video processing throughput and latency in a single resource-constrained SoC-FPGA—remains a pressing challenge. Limited on-chip resources, timing closure constraints when pushing clock periods lower, and the complex interplay between pipeline depth, area, and dynamic power create non-obvious trade-offs for system architects (Visconti et al., 2020; Li et al., 2023).

This article aims to address that gap by offering a comprehensive, theoretically grounded, and practically oriented guide to designing high-performance FPGA-based co-located cryptographic and image-processing systems for real-time embedded applications. It does so by synthesizing prior work in AES FPGA architectures, chaos-based generator implementations, accelerated image processing, and SoC-FPGA integration patterns (Li and Li, 2017; Rajasekar and Haridas, 2016; Elsayed et al., 2023; Tsai et al., 2023). The literature presents several recurring motifs: (1) deeply pipelined AES cores can achieve very high throughput but with increased area utilization and potential reordering complexity (Li and Li, 2017; Visconti et al., 2020); (2) reconfigurable and subpipelined AES architectures present valuable compromises between area and latency, enabling runtime flexibility (Li et al., 2023); (3) chaos-based generators implemented in FPGA logic are compact and adaptable but must be carefully evaluated for statistical randomness and hardware vulnerabilities (Elsayed et al., 2023; Maazouz et al., 2022); (4) high-level synthesis accelerates development time and allows algorithmic portability but may require targeted RTL-level optimizations to reach peak efficiency in constrained designs (Elsayed and Kayed, 2022; Sankar et al., 2023); and (5) SoC-FPGAs provide natural partitioning for hardware acceleration and real-time control using ARM processors coupled with programmable logic, facilitating real-time fusion and security stacks for embedded applications (Tsai et al., 2023; Yoon et al., 2016). Each motif informs the co-design methodology developed in this research.

The remainder of the article develops the proposed methodology and rationale. The Methodology section outlines a modular, layered approach to co-design that details how to structure AES pipelines, chaos-based key generation modules, image/video processing pipelines, and system integration on SoC-FPGA platforms. The Results section provides descriptive analyses of the expected performance envelopes, resource trade-offs, and comparative qualitative outcomes grounded in reported implementations. The Discussion interprets these findings, articulates limitations inherent to current practice, and lays out concrete research trajectories for energy modeling, dynamic reconfiguration, and standardized benchmarking. Throughout, claims are

grounded strictly in the corpus of referenced works to ensure fidelity to the empirical and theoretical bases put forward in the literature (Li and Li, 2017; Visconti et al., 2020; Elsayed et al., 2023; Bellemou et al., 2019; Rajasekar and Haridas, 2016).

METHODOLOGY

The methodology offered here is not an experimental protocol but rather a prescriptive architectural and engineering approach that integrates best practices from successful FPGA and SoC-FPGA implementations. It is organized into modular design layers: (A) Cryptographic Engine Design, (B) Chaos-Based Key and Entropy Generator, (C) Real-Time Image/Video Processing Pipelines, (D) System Integration and Hardware/Software Partitioning, and (E) Development Flow and Verification Strategy. Each layer synthesizes architectural considerations, design patterns, resource trade-offs, and integration prescriptions that emerged repeatedly in the surveyed literature.

A. Cryptographic Engine Design

1.Architectural Objectives and Constraints

The cryptographic engine must satisfy several sometimes competing objectives: high throughput (multi-Gbps for stream-based secure links), low-enough latency for real-time frames, small area footprint to allow room for image/video accelerators, and energy efficiency compatible with battery-powered or thermally constrained platforms (Visconti et al., 2020; Li and Li, 2017; Rajasekar and Haridas, 2016). Furthermore, practical deployments often demand support for both encryption and decryption, and sometimes multiple modes of operation (CBC, CTR, GCM) depending on communication stack needs (Bellemou et al., 2019). Consequently, the core design must be parameterizable and reconfigurable.

2. Pipeline Strategies and Reordering Techniques

Deep pipelining is the canonical route to high throughput: parallelize round computations across pipeline stages so that a new block enters the pipeline every clock cycle, thereby converting increased clock rate tolerance into throughput (Li and Li, 2017; Visconti et al., 2020). However, a naive pipeline introduces reordering complexities when multiple data flows must maintain block order, and the pipeline latency may be incompatible with strict frame deadlines. Li and Li (2017) discussed efficient reordering and merging techniques that allow high-throughput AES implementations to preserve data order without excessive buffer requirements. The methodology therefore recommends: implement a packet-aware reordering module that maintains sequence numbers for data blocks and uses compact FIFO banks to reestablish correct block ordering at boundaries where reordering could occur (Li and Li, 2017). This module must itself be optimized for area and latency, as excessive reordering logic can negate pipeline benefits.

3. Subpipelined and Reconfigurable Architectures

To balance area and latency, a subpipelined approach partitions rounds into fewer pipeline stages with multiple cycles per stage, reducing register overhead and area while retaining some throughput benefits (Li et al., 2023). Reconfigurable architectures let the same fabric cater to different performance or area modes—for instance, a high-throughput mode for bulk encrypted links and a compact mode for intermittent control messages (Li et al., 2023; Visconti et al., 2020). The methodology prescribes: design AES cores as parameterizable IP blocks with configurable pipeline depth, parallelism factor (number of parallel AES

instances), and an interface for runtime selection of operation mode. Such flexibility supports dynamic adaptation to runtime constraints or deployment-specific needs.

4. Clock-Periods and Timing Closure Strategies

Achieving both short clock periods and high pipeline depth strains timing closure; one must carefully allocate logic between stages and sometimes rely on retiming and floorplanning on larger devices. Visconti et al. (2020) showed that clock-period pipelining can push AES implementations to real throughputs up to tens of Gbit/s when executed on high-end Zynq UltraScale+ fabrics, but the strategy requires careful inter-stage balancing and device-specific placement. The methodology therefore advises profiling critical combinational paths, adjusting pipeline splits to equalize stage delay, and using device floorplanning and placement constraints to minimize interconnect delay across pipeline stages (Visconti et al., 2020).

5. Integration with Secure Protocols and TLS Stacks

In embedded systems, an AES core rarely operates alone; it typically integrates with higher-level security protocols such as TLS. Bellemou et al. (2019) demonstrated efficient SoC-FPGA TLSv1.2 implementations with ECC-AES support for IoT coordinators. The recommendation is to design the AES IP with standard interfaces (AXI-Stream for Zynq-based platforms), provide hooks for mode selection (e.g., GCM for authenticated encryption), and ensure latency and throughput are compatible with the broader protocol stack's state machine to prevent undue buffering or stalls (Bellemou et al., 2019).

B. Chaos-Based Key and Entropy Generator

1. Motivation and Applicability

Entropy sources for key generation and session randomness are indispensable for secure systems. Chaos-based digital generators have been proposed as compact, hardware-friendly entropy sources that can be implemented directly on FPGA fabric with low area overhead (Elsayed et al., 2023; Maazouz et al., 2022). The deterministic chaotic maps, when carefully discretized and combined, can approximate desirable statistical properties and provide a hardware-friendly primitive for lightweight cryptographic systems, particularly in resource-constrained IoT devices (Elsayed et al., 2023; Abd El-Maksoud et al., 2020).

2. Design Patterns and Hardware Mapping

Successful FPGA implementations use simple nonlinear discrete-time maps that fit well into LUT and DSP resources. Elsayed et al. (2023) described an adaptive chaotic key generator mapped to an FPGA with control logic for parameter variation, allowing run-time adaptation and improved unpredictability. The methodology recommends: choose chaotic maps with known digital implementations, use fixed-point arithmetic carefully scaled to preserve chaos properties under quantization, and include run-time parameter switching to avoid periodicity due to limited word widths. Additionally, statistical self-testing (online health tests) should be embedded to detect degeneration or entanglement that might reduce entropy (Elsayed et al., 2023).

3. Integration with Cryptographic Engine and Protocols

Chaos-based outputs must be post-processed—e.g., via whitening functions or cryptographic hash functions—before use as keys or nonces to mitigate any residual bias and to provide strong unpredictability under adversarial conditions (Maazouz et al., 2022; Bellemou et al., 2019). For deployment in TLS or ECC-

based key exchanges, the key generator should present a standard interface and be evaluated according to accepted randomness test suites and protocol compatibility (Bellemou et al., 2019; Wang et al., 2023).

C. Real-Time Image/Video Processing Pipelines

1. Kernel Selection and Partitioning

Real-time image processing encompasses tasks from simple edge detection and filtering to computationally heavier operations like Harris corner detection or feature extraction. The literature demonstrates hardware-accelerated implementations ranging from basic Sobel-based edge detection to optimized Harris corner detectors implemented either in RTL or via high-level synthesis (Ghodhbani et al., 2020; Ghodhbani et al., 2023; Sikka et al., 2021). The methodology prescribes selecting kernels that map well to FPGA primitives (convolutions, sliding windows, morphological operations), partitioning them into streaming pipelines that operate on pixel streams, and minimizing frame buffers by exploiting line buffers and window buffers implemented in distributed RAM/BRAM.

2. High-Level Synthesis vs RTL Implementations

High-level synthesis (HLS) offers rapid development and maintainability, allowing designers to describe algorithms in C/C++ and synthesize them to RTL (Elsayed and Kayed, 2022). Ghodhbani et al. (2020; 2023) and Sankar et al. (2023) documented that well-structured HLS pipelines can approach RTL performance and significantly reduce development time. However, HLS may produce less area- or timing-optimal logic than hand-crafted RTL in the most performance-critical kernels (Elsayed and Kayed, 2022). Therefore, the methodology prescribes a hybrid approach: implement control-heavy or complex concurrency logic in RTL when necessary, use HLS for algorithmic kernels with well-defined streaming behavior, and iteratively refine HLS-generated logic into hand-optimized RTL for hotspots identified during profiling.

3. Memory Architecture and Bandwidth Considerations

Video pipelines are often memory-bandwidth bound. Using on-chip BRAM and distributed RAM as line buffers and minimizing accesses to external DDR memory are central to achieving low-latency, deterministic processing (Yoon et al., 2016; Ghodhbani et al., 2020). The methodology recommends localizing working sets in on-chip memory whenever possible, overlaying streaming compute constructs that consume and produce AXI-Stream packets, and employing DMA engines for bulk transfers between SoC DRAM and peripherals when bridging between hardware-accelerated pipelines and CPU-controlled tasks (Tsai et al., 2023).

4. Algorithmic Stabilization for Noisy Inputs

In real-time deployments, noisy video streams are common; Yoon et al. (2016) presented Zynq-based reconfigurable systems for edge detection under noisy conditions. The co-design must include algorithmic safeguards—e.g., adaptive thresholding, morphological denoising blocks, and context-aware feature gating—to ensure robust operation. Such stabilizing modules should be parameterizable at runtime by the ARM software layer to adapt to changing sensor noise characteristics (Yoon et al., 2016; Ghodhbani et al., 2023).

D. System Integration and Hardware/Software Partitioning

1. SoC-FPGA Partitioning Principles

SoC-FPGA platforms (e.g., Xilinx Zynq and Zynq UltraScale+ families) provide ARM cores tightly coupled with programmable logic. The integration strategy should place deeply parallel, latency-insensitive pipelines—such as AES stream encryption and pixel-stage convolutions—in programmable logic, while placing control logic, policy management, and higher-level protocol handling in the ARM cores (Tsai et al., 2023; Yoon et al., 2016). The methodology prescribes: use AXI interconnects and AXI-Stream interfaces for data movement, expose control registers via AXI-Lite for configuration, and employ interrupt-based notification coupled with DMA for efficient software-hardware synchronization.

2. Real-Time Information Fusion and Timing Constraints

For systems that require information fusion (sensor data combined with visual processing)—a scenario common in autonomous systems—Tsai et al. (2023) documented ARM-based FPGA fusion systems that reduce latency by offloading heavy processing to the programmable logic while maintaining flexible fusion algorithms on the processor. The methodology calls for precise end-to-end latency budgeting, timing analysis that accounts for worst-case processing times in hardware modules, and jitter mitigation strategies such as deterministic scheduling in the processor and prefetching strategies for pipeline priming.

3. Security and Side-Channel Considerations in Integration

Integrating cryptographic engines on FPGAs introduces side-channel leakage risks, particularly from power and electromagnetic emanations. Hardware designers must consider countermeasures including masking, hiding through constant-time implementations, and physical shielding where possible. Implementing these countermeasures impacts area and timing; thus, the methodology recommends including security margin provisions in resource allocation and verifying side-channel resistance during hardware testing and certification phases (Park and Park, 2020).

E. Development Flow and Verification Strategy

1. Rapid Prototyping and HLS Tools

The literature shows that HLS and tools such as HDL Coder expedite prototyping and enable quicker design space exploration (Elsayed and Kayed, 2022; Sankar et al., 2023). The recommended flow begins with algorithmic prototypes in high-level languages, followed by HLS translation for streaming kernels, and final RTL hand-optimization for critical modules. Use device-specific IP for cryptographic primitives where available and validate vendor IP integration through simulation and hardware-in-the-loop testing (MathWorks, 2012-2015; Sankar et al., 2023).

2. Testbenching and Formal Checks

Unit-level verification via RTL testbenches, simulation with corner-case datasets for image pipelines, and statistical randomness tests for chaos-based generators are essential. Additionally, integration testing should emulate realistic input streams and protocol workloads to ensure that end-to-end latency and throughput goals are met under stress. Use on-hardware profiling to capture clock domain crossings and backpressure behaviors in streaming interfaces.

3. Energy and Sleep Strategies for Sensor Networks

Del-Valle-Soto et al. (2020) articulated energy models using sleeping algorithms for wireless sensor

networks. When integrating cryptography and image processing, designers should consider dynamic power management strategies—power gating, clock gating, and selective module sleep—to conserve energy during idle or low-data-rate periods. The methodology suggests instrumenting the system to monitor data rates and activity and to adaptively switch module modes (e.g., AES compact mode vs high throughput) to minimize energy per useful bit processed (Del-Valle-Soto et al., 2020).

RESULTS

Because this work synthesizes and prescribes design methodologies grounded in the referenced literature rather than reporting a single new hardware experiment, the Results section presents a descriptive analysis of the expected performance envelopes, resource trade-offs, and qualitative outcomes one can anticipate when following the described methodology. The analysis maps reported empirical results and design observations from the literature to the design decisions enumerated above and illustrates common trade-off frontiers for practical system architects.

Throughput and Latency Envelopes for AES Implementations

Deeply pipelined AES cores have demonstrated remarkably high throughput figures when implemented on modern FPGA fabrics. Visconti et al. (2020) reported clock-period pipelined AES-128 implementations on the Xilinx Zynq UltraScale+ ZCU102 platform reaching up to 28 Gbit/s real throughput in certain configurations (Visconti et al., 2020). Li and Li (2017) also demonstrated high-throughput AES implementations using efficient reordering and merging techniques. These results suggest an envelope where top-tier SoC-FPGA hardware, with careful pipeline balancing, can support multi-Gigabit encryption/decryption suitable for high-speed links. However, such extremes rely on substantial logic and register overhead which may be incompatible with designs that must also host heavy image pipelines. When area is constrained, subpipelined or reconfigurable AES cores (Li et al., 2023) allow designers to trade raw throughput for lower area while still satisfying moderate bandwidth demands typical of many IoT and sensor network scenarios (Bellemou et al., 2019; Rajasekar and Haridas, 2016).

Chaos-Based Key Generation: Compactness vs Statistical Robustness

Chaos-based generators implemented in programmable logic exhibit excellent compactness and adaptability (Elsayed et al., 2023; Abd El-Maksoud et al., 2020). They can be parameterized to produce high-entropy streams used as seeds or nonces, and their low area footprint makes them attractive for embedded deployments where True Random Number Generator (TRNG) hardware may be absent or costly in terms of area. Nevertheless, Maazouz et al. (2022) and Elsayed et al. (2023) emphasize the necessity of post-processing and statistical validation to mitigate quantization-induced periodicity and bias. Thus, such generators are well-suited as entropy augmenters or part of hybrid schemes that combine physical noise sources when absolute cryptographic-grade randomness is required (Maazouz et al., 2022; Elsayed et al., 2023).

Image/Video Pipelines: Determinism, Resource Use, and Latency

High-speed image processing kernels—Harris corner detection, edge detection, and other low-level filters—have been successfully implemented on FPGAs to achieve real-time performance with low deterministic latency (Ghodhbani et al., 2020; Ghodhbani et al., 2023; Sikka et al., 2021). HLS pipelines often provide development speed advantages and can attain strong throughput when streaming architectures are used. However, the literature repeatedly notes a performance gap between HLS-generated logic and hand-optimized RTL in the most demanding pipelines (Elsayed and Kayed, 2022). As such, designers can expect

HLS to satisfy many real-time applications, while the highest frame-rate or lowest-latency needs may necessitate manual RTL optimization of bottleneck kernels.

System-Level Integration: SoC Benefits and Synchronization Considerations

SoC-FPGA platforms, by combining programmable logic with general-purpose ARM cores, provide fertile ground for partitioned architectures where secure streaming and compute kernels run in hardware while policy, fusion logic, and protocol stacks run in software (Tsai et al., 2023; Yoon et al., 2016). The integration reduces data movement latency and allows high-level adaptability. However, end-to-end real-time guarantees require careful synchronization, worst-case latency budgeting, and DMA-aware scheduling to avoid pipeline stalls. The necessity to align cryptographic operation timing with communication protocol state machines also arises, reinforcing the importance of standardized interfaces and sufficient buffering to decouple hardware and software pace irregularities (Bellemou et al., 2019; Tsai et al., 2023).

Energy Considerations and Sleep Strategies

Del-Valle-Soto et al. (2020) highlighted energy models using sleeping algorithms for wireless sensor networks under proactive and reactive protocols. In co-designed systems that combine AES engines and image pipelines, energy consumption is heavily workload-dependent. High-throughput AES and continuous image processing drive substantial dynamic power; hence, integrating runtime modes (sleep/compact/high-throughput) and power gating strategies can yield significant energy savings during intermittent operation. Implementing adaptive modes for cryptographic engines—switching to compact AES configurations during low-traffic periods—can reduce average energy per cryptographic operation while preserving the ability to scale throughput when needed (Del-Valle-Soto et al., 2020; Li et al., 2023).

Development Flow Outcomes: HLS Productivity vs RTL Efficiency

The literature strongly supports the productivity benefits of HLS and automated flows. Elsayed and Kayed (2022) documented rich case studies contrasting MATLAB HDL Coder-generated designs with VHDL implementations, showing that automated flows dramatically reduce development time and ease iteration. However, HLS may produce less area- and timing-optimal circuits in the tightest hotspots, necessitating selective RTL rewrites. Sankar et al. (2023) and Guerrieri et al. (2023) reported that a mixed approach—HLS for many modules and manual RTL for performance-critical paths—provides an effective trade-off between development time and deployed performance.

DISCUSSION

This synthesis indicates that building a unified FPGA-based platform capable of robust AES processing, chaos-based key generation, and real-time image/video processing is both feasible and practical using modern SoC-FPGA devices and contemporary development practices. The discussion below interprets the design trade-offs, draws out nuanced implications for deployment scenarios, highlights limitations of current approaches, and proposes future research directions.

Trade-offs between Throughput, Area, and Flexibility

The fundamental tension in co-design is the allocation of finite on-chip resources between cryptographic cores and image pipelines. Deep pipelining yields exceptional AES throughput (Visconti et al., 2020), but requires substantial registers and logic, possibly crowding out image-processing accelerators. Conversely, compact AES modes conserve area but reduce throughput. Subpipelined and reconfigurable AES

architectures offer a pragmatic middle ground by allowing runtime switching between modes tailored to current application needs (Li et al., 2023). In systems where worst-case maximum throughput is rarely needed, dynamic reconfiguration to a compact AES variant can free resources for image processing and reduce power consumption. The designers must therefore base resource partitioning on realistic workload profiles and apply adaptive reconfiguration when workloads change (Li et al., 2023; Del-Valle-Soto et al., 2020).

Security vs Resource Efficiency in Entropy Generation

Chaos-based generators are attractive for embedded systems due to compactness and FPGA suitability (Elsayed et al., 2023). However, cryptographic rigor demands rigorous validation. Unless combined with proven physical TRNG sources or cryptographic post-processing, chaos-based generators alone may not satisfy the most stringent entropy requirements. Therefore, designers should adopt hybrid approaches—chaos generators serving as additional entropy sources subjected to whitening and statistical testing, with fallback to stronger sources where available (Maazouz et al., 2022; Bellemou et al., 2019).

HLS: Rapid Iteration with Targeted Manual Optimization

High-level synthesis expedites prototyping, encourages maintainable design practices, and facilitates algorithmic exploration. Yet the literature documents that HLS can lag behind hand-optimized RTL on area and peak clock frequency in some kernels (Elsayed and Kayed, 2022; Sankar et al., 2023). The practical approach is to prioritize HLS for medium-complexity, streaming-friendly kernels and to schedule RTL refactoring only for modules identified as critical during profiling. This workflow leverages the strengths of both paradigms while controlling engineering effort.

SoC-FPGA Integration: Determinism and Real-Time Guarantees

SoC-FPGA architectures provide natural hardware-software partitioning, but guaranteeing end-to-end determinism remains challenging. The ARM side may introduce jitter or unpredictable scheduling latencies if general-purpose OSes are used. Therefore, for hard real-time requirements, run-time-critical software should be implemented on real-time OS or bare-metal contexts and peripheral interactions should employ DMA and hardware handshakes to decouple software jitter from hardware processing. This design discipline ensures that streaming pipelines continue to operate predictably even under transient software load (Tsai et al., 2023; Yoon et al., 2016).

Limitations and Caveats

A major limitation of the present synthesis is that it aggregates and extrapolates from reported implementations rather than providing new experimental measurements on a common hardware baseline. As device architectures evolve rapidly (e.g., newer UltraScale+ vs older Zynq devices), absolute performance metrics will change; therefore, system architects must adapt the prescriptions to their chosen device's resource landscape and toolchain maturity (Visconti et al., 2020; Guerrieri et al., 2023). Additionally, side-channel resistance and formal security proofs for chaos-based key generation require deeper empirical study than currently present in the literature; while chaos generators are promising, their cryptographic suitability in adversarial settings remains an active research topic (Elsayed et al., 2023; Abd El-Maksoud et al., 2020). Finally, the guidance on energy management is derived from models and high-level studies rather than from experimentally verified, cross-device energy measurements; energy-per-operation figures should be measured on target hardware as part of system integration.

Future Research Directions

1. Empirical cross-platform benchmarking: Implement the proposed co-design blueprint on multiple contemporary SoC-FPGA families and collect standardized metrics (throughput, latency, area utilization, energy per operation) to create comparative baselines. Such benchmarks would help quantify the cost of high-throughput AES pipelines in terms of resources and energy in a reproducible way (Visconti et al., 2020; Li et al., 2023).

2. Dynamic reconfiguration strategies: Investigate low-overhead, secure mechanisms to reconfigure AES pipeline depth and image pipeline resource allocation at runtime in response to workload and energy constraints. Research should consider reconfiguration latency, configuration security, and safe-state transitions.

3. Hybrid entropy systems: Study the integration of chaos-based generators with physical TRNGs and robust post-processing, including an adversarial analysis of potential failure modes in noisy or attack-prone environments (Elsayed et al., 2023; Maazouz et al., 2022).

4. Side-channel and fault-injection resilience: Develop lightweight countermeasures tailored to SoC-FPGA co-designs that balance protection against side-channel and fault injection attacks with area and timing constraints (Park and Park, 2020).

5. Energy-aware design automation: Create toolchain extensions that automatically explore the configuration space (AES pipeline depth, number of image-processing instances, HLS pragmas) to optimize multi-objective cost functions combining throughput, latency, resource use, and energy.

CONCLUSION

This article assembled a comprehensive co-design methodology to realize high-performance FPGA-based platforms capable of simultaneous cryptographic processing, chaos-based entropy generation, and real-time image/video processing. By synthesizing empirical results and architectural techniques from the referenced literature, practical design patterns emerged: deep pipelining and clock-period optimizations for AES when maximum throughput is required; subpipelined and reconfigurable AES architectures when area and flexibility are prioritized; chaos-based generators as compact entropy augmenters subject to post-processing and statistical validation; hybrid HLS/RTL development flows that accelerate time-to-prototype while preserving opportunities for manual optimization; and SoC-FPGA partitioning strategies that exploit the strengths of both hardware acceleration and software flexibility. The resulting recommendations form a pragmatic blueprint for system architects tasked with building secure, low-latency, and energy-conscious embedded systems. Future work must empirically validate energy and security trade-offs across device families and investigate dynamic reconfiguration and hybrid-entropy approaches to keep pace with the evolving demands of edge computing and autonomous systems.

REFERENCES

1. Li, L.; Li, S. High throughput AES encryption/decryption with efficient reordering and merging techniques. In Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Gent, Belgium, 4–6 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–4.

2. Babu, P.; Parthasarathy, E. Hardware acceleration of image and video processing on Xilinx zynq

platform. Intell. Autom. Soft Comput. 2021, 30(3).

3. Elsayed, G.; Soleit, E.; Kayed, S. FPGA design and implementation for adaptive digital chaotic key generator. Bulletin of the National Research Centre 2023, 47(1), 122.

4. Li, K.; Li, H.; Mund, G. A reconfigurable and compact subpipelined architecture for AES encryption and decryption. EURASIP Journal on Advances in Signal Processing 2023, 2023(1), 1–21.

5. Visconti, P.; Capoccia, S.; Venere, E.; Velázquez, R.; Fazio, R. d. 10 Clock-Periods Pipelined Implementation of AES-128 Encryption-Decryption Algorithm up to 28 Gbit/s Real Throughput by Xilinx Zynq UltraScale+ MPSoC ZCU102 Platform. Electronics 2020, 9, 1665.

6. Abd El-Maksoud, A. J.; Abd El-Kader, A. A.; Hassan, B. G.; Rihan, N. G.; Tolba, M. F.; Said, L. A.; Abu-Elyazeed, M. F. FPGA implementation of integer/fractional chaotic systems. In Multimedia Security Using Chaotic Maps: Principles and Methodologies; 2020; pp. 199–229.

7. Wang, D.; Lin, Y.; Hu, J.; Zhang, C.; Zhong, Q. FPGA Implementation for Elliptic Curve Cryptography Algorithm and Circuit with High Efficiency and Low Delay for IoT Applications. Micromachines 2023, 14(5), 1037.

8. Maazouz, M.; Toubal, A.; Bengherbia, B.; Houhou, O.; Batel, N. FPGA implementation of a chaos-based image encryption algorithm. Journal of King Saud University—Computer and Information Sciences 2022, 34(10), 9926–9941.

9. Rajasekar, P.; Haridas, M. Efficient FPGA implementation of AES 128 bit for IEEE 802.16e mobile WiMax standards. Circuits Syst. 2016, 7, 371–380.

10. Elsayed, G.; Kayed, S. I. A comparative study between MATLAB HDL Coder and VHDL for FPGAs design and implementation. J Int Soc Sci Eng 2022, 4, 92–98.

11. Al-Musawi, W. A.; Wali, W. A.; Al-Ibadi, M. A. Implementation of Chaotic System using FPGA. In 2021 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS); 2021; pp. 1–6.

12. Del-Valle-Soto, C.; Velázquez, R.; Valdivia, L. J.; Giannoccaro, N. I.; Visconti, P. An Energy Model Using Sleeping Algorithms for Wireless Sensor Networks under Proactive and Reactive Protocols: A Performance Evaluation. Energies 2020, 13, 3024.

13. Noorbasha, F.; Divya, Y.; Poojitha, M.; Navya, K.; Bhavishya, A.; Rao, K.; Kishore, K. FPGA design and implementation of modified AES based encryption and decryption algorithm. Int. J. Innov. Technol. Explor. Eng. 2019, 8, 132–136.

14. Ghodhbani, R.; Saidani, T.; Alhomoud, A.; Alshammari, A.; Ahmed, R. Real Time FPGA Implementation of an Efficient High Speed Harris Corner Detection Algorithm Based on High-Level Synthesis. Engineering, Technology and Applied Science Research 2023, 13(6), 12169–12174.

15. Sikka, P.; Asati, A. R.; Shekhar, C. Real time FPGA implementation of a high speed and area optimized Harris corner detection algorithm. Microprocessors and Microsystems 2021, 80, 103514.

16. Patil, A. A.; Deshpande, S. Real-time encryption and secure communication for sensor data in autonomous systems. Journal of Information Systems Engineering and Management 2025, 10(415),

41–55.

17. Tsai, Y. H.; Yan, Y. J.; Hsiao, M. H.; Yu, T. Y.; Ou-Yang, M. Real-Time Information Fusion System Implementation Based on ARM-Based FPGA. Applied Sciences 2023, 13(14), 8497.

18. Park, J.; Park, Y. Symmetric-Key Cryptographic Routine Detection in Anti-Reverse Engineered Binaries Using Hardware Tracing. Electronics 2020, 9, 957.

19. Ghodhbani, R.; Horrigue, L.; Saidani, T.; Atri, M. Fast FPGA prototyping based real-time image and video processing with high-level synthesis. International Journal of Advanced Computer Science and Applications 2020, 11(2).

20. Bellemou, A. M.; García, A.; Castillo, E.; Benblidia, N.; Anane, M.; Álvarez-Bermejo, J. A.; Parrilla, L. Efficient Implementation on Low Cost SoC-FPGAs of TLSv1.2 Protocol with ECCAES Support for Secure IoT Coordinators. Electronics 2019, 8, 1238.

21. MathWorks, Inc. HDL Coder User's Guide. 2012–2015.

22. Guerrieri, A.; Upegui, A.; Gantel, L. Applications Enabled by FPGA-Based Technology. Electronics 2023, 12(15), 3302.

23. Sankar, D.; Lakshmi, S.; Babu, C.; Mathew, K. Rapid prototyping of predictive direct current control in a low-cost fpga using hdl coder. Int J Power Energy Syst 2023, 43(10).

24. Yoon, I.; Joung, H.; Lee, J. Zynq-based reconfigurable system for real-time edge detection of noisy video sequences. Journal of Sensors 2016.

25. Patil, A. A.; Deshpande, S. Real-time encryption and secure communication for sensor data in autonomous systems. Journal of Information Systems Engineering and Management 2025, 10(415), 41–55.