Design and Implementation of a flexible Multi-purpose Cryptographic System on low cost FPGA

Original Scientific Paper

Ahmed Maache
Laboratory of Signals and Systems
Institute of Electrical Engineering and Electronics
University M'Hamed Bougara of Boumerdes, Algeria
a.maache@univ-boumerdes.dz

Abdesattar Kalache
Laboratory of Signals and Systems
Institute of Electrical Engineering and Electronics
University M'Hamed Bougara of Boumerdes, Algeria
kalacheabdessattar@gmail.com

Abstract *– The design of cryptographic hardware that supports multiple cryptographic primitives is common in literature. In this work, a new design is presented consisting of a multi-purpose cryptographic system featuring both 128-bit pipelined AES-CORE (Advanced Encryption Standard) for high-speed symmetric encryption and a Keccak hash core on a low-cost FPGA. The KECCAK-CORE's security and performance parameters are tunable in the sense that capacity, bitrate, and the number of rounds can be user-defined. Such flexibility enables the core to suit a large range of security requirements. The structure of Keccak's sponge construction is exploited to enable different modes of operation. An example application outlined in this work is Pseudo Random Number Generation (PRNG). With few adjustments, the KECCAK-CORE was also operated as a post-processing unit for True Random Number Generation (TRNG) that uses the analog Lorenz chaotic circuit as a physical entropy source. The multi-purpose design was implemented in VHDL targeting an IntelFPGA Cyclone-V FPGA. For AES symmetric encryption, a maximum throughput of 31.1Gbps was achieved and a logic usage of 25146LEs (23% of the FPGA) in the case of the pipelined variant of AES-CORE. For the KECCAK-CORE, maximum throughput figures of 5.81, 8.4, and 11Gbps were achieved for the three SHA-3 variants 512, 384, and 256-bit respectively, with an area usage of 8947LEs (8%). The system as a whole occupies an area of 26909LEs (26%). The random sequences generated by the system operating in PRNG and TRNG post-processing modes successfully passed the National Institute of Standards and Technology (NIST) statistical test suite (NIST SP 800-22). The information entropy analysis performed on the post-processed TRNG sequences indicates an average of 0.935.*

Keywords*: Advanced encryption standard (AES), Pipelining, FPGA, Keccak, RNG, secure hash algorithm-3 (SHA -3)*

1. INTRODUCTION

Security is crucial in today's embedded systems. To ensure data integrity and confidentiality, various cryptographic algorithms have been developed. The structure of these algorithms is highly parallelizable, which makes them favorable for hardware implementations. Also, the emphasis on high communication bandwidths requires the use of hardware-accelerated cryptographic algorithms most commonly ASIC (Application Specific Integrated Circuit) accelerators which offer the highest performance and power efficiency. However, these are not cost-efficient for individual use. FPGAs (Field Programmable Gate Arrays) are flexible alternatives that provide reconfiguration and lower costs for small production. In addition, they can facilitate the design/prototype of cryptographic hardware that combines multiple cryptographic primitives [1], [2], [3]. This flexibility enables the system to support various cryptographic operations and protocols required by applications such as IEEE 802.11, Bluetooth, Transport Layer Security (TLS), Global System for Mobile (GSM), ISO/IEC 29192, etc.

The advantages of using flexible multi-purpose cryptosystems that use ASIC and FPGA lie in their ability to provide a common implementation that supports various requirements for different new technologies such as Wireless Sensor Networks (WSN) and the Internet of Things (IoT). These technologies often need

to overcome the challenge of deploying efficient cryptographic algorithms on their resource-constrained nodes [4]. The use of resource-shared multi-purpose cryptographic designs can help in overcoming such challenges and offer area/power reductions rather than using multiple distinct cores. This flexibility can also assist in protecting against different attacks by switching to a more suitable security level of the same algorithm if needed. In terms of performance, throughput requirements may also dictate that the system should switch to a more appropriate configuration [1].

The multi-purpose design proposed in this paper includes two main cryptographic classes: block cipher for symmetric-key encryption/decryption and hash functions. The symmetric encryption is performed using AES128. A new Keccak design and implementation (SHA-3) is presented, which supports multiple modes of operations with tunable security/performance parameters and extendable output length. The security/performance parameters include the capacity, bitrate, and number of rounds. A practical mode of operation supported by the proposed construction is examined, that is a reseedable cryptographically secure PRNG, which is beneficial when high throughput bursts of random bits are needed. Furthermore, this work studies the use of the KECCAK-CORE as a post-processing unit for physical entropy bits (sampled from the analog Lorenz chaotic system) in TRNG to eliminate statistical defects inherited in the *digitized analog signals* (*das*). The post-processed bits can be used to seed the PRNG, the two generators can be used to generate in-system signals required for encryption such as keys and nonces. For consistency and simplicity, the general nature NIST's statistical test suite is used to evaluate both RNGs [5].

Contributions of this work are summarized as follows:

- The implementation of a pipelined AES operated in CTR (Counter) mode with no RTL (Register Transfer Level) register balancing. The architecture provides high throughput while maintaining a good throughput-to-area ratio.
- The hardware implementation/evaluation of the most recent provably secure PRNG in cryptography, the sponge-based PRNG.
- Investigation of the cryptographic hash function based post-processing by taking advantage of the implemented Keccak algorithm to debias the chaotic Lorenz system. Even though hashing algorithms' implementations are relatively demanding in hardware, this is not an issue in the case of this work since the hash function was already implemented for other purposes.

This paper is organized as follows. In Section 2, related literature is outlined. Section 3 covers some needed cryptographic background concepts related to AES, KECCAK, and RNG. Section 4 explains the design and implementation of the proposed system and its supported modes of operation. Section 5 presents the obtained results in terms of performance, area usage, and randomness/entropy-related tests. Section 6 compares these results to other related works. Conclusions are laid out in Section 7.

2. RELATED WORK

Several implementations of resource-shared AES with AES-like hash functions have been presented in the literature. Authors in [1] proposed a resource-shared crypto-coprocessor of AES with SHA-3, in which they fit four non-pipelined AES-128 units with SHA3-256 by integrating lookup tables and sharing the unified XOR sections. However, this design is limited in terms of AES throughput because it is non-pipelined. Also, the Six Input Equation optimization used cannot take advantage of other devices with different LUT input sizes. In [6], a resource-sharing design of AES and Fugue was outlined. However, Fugue was a second-round candidate that got eliminated due to security flaws and its average throughput-to-area ratio [7]. Other studies [8-11] have implemented resource-shared AES with Grøstl-256 on different FPGA platforms. Compared to Keccak, Grøstl-256 has relatively small security margins, lower throughput, throughput/area, and energy consumption-per-bit on FPGAs and ASICs [12]. Also, many of these designs are non-pipelined, which results in low area usage with relatively low throughput. Furthermore, the hashing units are of fixed output lengths and security parameters, which in turn limits their suitability for different applications. HLS (High-Level Synthesis) was used in [13] to implement a dynamically configurable SHA-3 accelerator in terms of digest length and capacity. However, the use of HLS is relatively less efficient than pure HDL (Hardware Description Language) design flow. High-speed pipelined SHA-3 designs were presented in [14-18]. However, they support no flexibility, and hence a narrower application scope.

Authors in [19] proposed an ultra-high throughput fully pipelined AES operated in CTR mode. A 60Gbps inner and outer pipelined AES architecture was proposed in [20]. More sophisticated timing optimization techniques were described in [21] where an 82Gbps pipelined AES was designed using 2-slow balancing techniques. While these works provide ultra-high throughputs, the area usage is not proportional to the increase in throughput, hence, a low throughput-to-area ratio. In [3], a highly customized VLSI (Very Large Scale Integration) design of an advanced AES Cryptoprocessor is presented. The design supports multiple modes of operations targeting the European Processor Initiative (EPI) that features multiple hardware cryptographic accelerators, including SHA, which are controlled by a secure RISC-V processor.

PRNGs are well studied in the literature. FPGA stream cipher/LFSR-based PRNGs were proposed in [22], [23]. Chaos-based PRNGs were presented in [24], [25] where the three-dimensional chaotic systems were imple-

mented digitally, which implies a deterministic fully predictable system with no sensitivity to initial conditions. Bits from the three variables were post-processed to generate pseudo-random numbers. The above-stated works provide very compact hardware implementation and good performance. On the other hand, hardware and software designs of cryptographically secure PRNGs were proposed. Authors in [26] demonstrated high throughput and low power FPGA implementations of two PRNGs, one of which is the computationally secure Blum Blum Shub generator. In contrast to the algorithmic nature of PRNGs, TRNGs are physical. However, an algorithmic debiasing step performed on the digitized bits is required; also referred to as TRNG post-processing. Authors in [27] relied on SHA-256 to debias bits generated by ring oscillators which is fundamentally different from chaotic systems.

3. BACKGROUND

3.1. AES ALGORITHM

AES can provide up to the TOP SECRET level of security with high performance on both software and hardware. It features multiple key lengths of 128, 192, and 256 bits each with the number of rounds 10, 12, and 14 respectively. Initially, the encryption starts by XORing the key and the input data before feeding the result through the rounds. Each round consists of four steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey, except for the last round which requires no column mixing. SubBytes step is a nonlinear byte-to-byte correspondence described by the S-BOX which is obtained by calculating the multiplicative inverse of the input byte followed by an affine transform. ShiftRows is a transformation of the state via circular shifts with different values at each row. The previous two steps provide the required confusion to avoid any differentiability between the input and output. MixColumns is a transformation that operates on the State column-by-column, treating each column as a four-term polynomial [28] to ensure diffusion. The AddRoundKey step consists of XORing the resultant state from Mixcolumns with the RoundKey provided by the key expansion.

CTR is a feedback-free block cipher mode where the ciphertext is a result of XORing the plaintext with the output of encrypted successive counter values concatenated with a nonce. The use of a single XOR operation on the plaintext results in a symmetry between the encryption and decryption. Unlike feedback modes, CTR mode is highly parallelizable, random read accessible, and suffers no error propagation problems making its implementation straightforward.

3.2. SHA-3 AND KECCAK FAMILY

AES SHA-3 is a subset of the broader cryptographic primitive family Keccak of hash functions. It is based on the novel Hermetic Sponge Construction approach. Although it features various state widths b = {25, 50, 100, 200, 400, 800, 1600}-bit, only permutation of b = 1600 bits was submitted [29]. The Sponge Construction is divided into two phases: *absorption* and *squeezing* phase as seen in Fig. 1. It uses $b = r + c$ bits of state, where r is the rate at which states are updated with message bits between each application of the permutation function, c is the capacity and defines the security level. Increasing the capacity results in a higher security level with a performance penalty and vice-versa [29]. Its value is set by the user depending on the application. The input state of Keccak-f[b] is arranged in a three-dimensional matrix of $5\times5\times w$, where w defines the length of the lane and equals $b/25$.

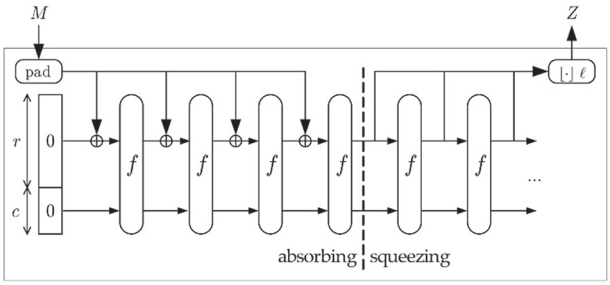

Fig. 1. The Sponge Construction [30]

In the absorption phase, the input block of length r is XORed with the state lane-wise. If the length of the current input block is less than r, padding is required. After the input data is completely absorbed, the output is obtained by truncating the state. The output length is a user choice in certain applications. For hash functions, the lengths are 224, 256, 384, and 512-bit. If the required output length is greater than the bitrate, it is obtained by truncating the outputs after feeding through Keccak-f until the required length is satisfied, which is known as the squeezing phase [31]. Each round R of Keccak-$f[b]$ consists of five-step mappings $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ where:

$$\begin{aligned}
&[\theta] : C[x][z] \leftarrow a[x][0][z] \oplus a[x][1][z] \oplus ... \oplus a[x][4][z] \\
&\quad D[x][z] \leftarrow C[x-1][z] \oplus C[x+1][z-1] \\
&\quad a[x][y][z] \leftarrow a[x][y][z] \oplus D[x][z] \\
&[\rho][\pi] : B[y][(2.x+3.y)][z] \leftarrow A[x][y][rp(x,y)] \\
&[\chi] : a[x][y][z] \leftarrow B[x][y][z] \oplus (B[x+1][y][z] \oplus 1) \\
&\quad .B[x+2][y][z] \\
&[\iota] : a[0][0][z] \leftarrow a[0][0][z] \oplus RC_i[z]
\end{aligned} \tag{1}$$

where *a[x,y,z]* represents a particular lane of a state; *B*, *C*, and *D* are the intermediate results, and RC_i is the round constant. Note that π and ρ steps are combined because they are merely two consecutive permutations of the state. The values of this permutation are hardwired in the *rp* array. The number of rounds to be executed n_r is calculated as $12+ 2(\log_2(b/25))$.

3.3. RANDOM NUMBER GENERATION

3.3.1. Sponge-based PRNG: PRNG is a deterministic algorithm that, given a truly random binary sequence

of length *n* referred to as a seed, outputs a binary sequence of length *N* > *n* that is completely determined by the seed and 'looks' random. Thus, if the seed is compromised, the output sequences of the PRNG are known. Therefore, a requirement for a seed is randomness, hence a common technique is to seed a PRNG with a TRNG and then use the PRNG afterward. This is because PRNG has superior performance relative to TRNG. Another requirement for the seed is a sufficient length for it to be insusceptible to brute-force recovery. Moreover, the seed should contain enough entropy for the application since the seed entropy defines an upper limit for the entropy that the PRNG can deliver. Ideally, a PRNG can be constructed with a random oracle that responds deterministically to every unique query with a truly random response chosen uniformly from its output domain. Non-empty seeds are fed to the PRNG to update the state and random output bits are fetched afterward. It should be known that previous seeds should be stored, hence the name history-keeping mode [32]. The history is encoded by seed-complete encoding function *e(h)*. The encoded history is provided to the random oracle during the fetch call producing a random sequence that is truncated at every feed-fetch cycle and so on (Fig. 2). The memory needed for history-keeping mode grows linearly with the number of past queries; rendering it impractical.

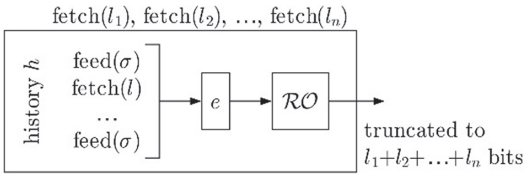

Fig. 2. History-keeping mode PRNG [32]

Instead of a random oracle, sponge construction can be used as in [32], which yields a history-keeping mode similar to reseedable PRNG. The sponge construction relies on a fixed length state which implies no memory growth. Furthermore, sponge construction features similarities with the mode; where every input absorption of length *r* into the state is a feed request and the same goes for fetch requests (Fig. 1). In addition, the last *c* bits of the state are never directly affected by the input blocks. Capacity *c* determines the attainable security level of the construction.

3.3.2. TRNG: TRNG extracts bits from a non-deterministic physical process. Naturally, the physical process produces a continuous-time analog signal, which gets digitized uniformly to yield *das*. Due to their physical nature, TRNGs are typically a separate piece of hardware connected to the application via an interface (USB or PCI bus...). Examples of physical processes used are Josephson's Junction [33], Johnson's noise [34], and Zener diode's shot noise [35]. Unlike PRNGs, TRNGs suffer from uneven probabilities of zeros and ones. The difference of probabilities of 0s and 1s is termed bias *b* = *(p(1)-p(0))/2*. For that, a post-processing step is needed where the digitized data is transformed into uniformly distributed random numbers i.e *b = 0*. Post-processing also helps to eliminate other statistical defects introduced by the physical source. While the output's entropy of the PRNG is bounded up by the seed's entropy, TRNGs' output entropy increases after each random number is generated.

3.4. CHAOTIC SYSTEMS AS ENTROPY SOURCES

Chaotic systems are by no means random. They exhibit a deterministic behavior since this class of systems can be described neatly as a system of nonlinear ordinary differential equations. Since the system cannot be solved analytically, mathematicians resort to phase space to have a qualitative rather than quantitative understanding of the system. Solutions to the system's differential equation can be represented as a trajectory in phase space. The system being chaotic implies that two trajectories will be $||E(t)|| \sim ||E(t_0)||.e^{\lambda t}$ apart from each other, where $t > t_0$ and λ is Lyapunov exponent. This encapsulates the sensitive dependence on initial conditions resulting from the exponential divergence of nearby trajectories [36]. This limits the horizon of prediction up to a time t_h, known as the Lyapunov time; which corresponds also to the loss of one bit of information [37]. In addition to the sensitivity to initial conditions, chaotic systems exhibit nonperiodic behavior. In this work, the Lorenz system will be used as an entropy source for the RNG, which is a simplified mathematical model for atmospheric convection (details are in Section 4.3.2.).

A successive sampling of the Lorenz system results in a seeding material for the PRNG or a das for the TRNG. The entropy of the material depends on the sampling frequency, sampling resolution, and quantization loss of the quantization function. The source entropy should be at least equal to the security strength of the instantiation [38]. The instantiation shall provide reseeding in case the internal state of the PRNG is compromised. Periodic reseeding shall reduce the likelihood of a security threat.

4. SYSTEM DESIGN AND IMPLEMENTATION

The multi-purpose cryptographic system consists of two cryptographic cores. The first is *AES-CORE* dedicated mainly to 128-bit symmetric-key encryption operated in CTR mode. The second is a tunable *KECCAK-CORE*, based on KECCAK[b=1600], that supports different modes of operation. The normal mode of operation of the KECCAK-CORE is a certified hash function (SHA-3) that outputs digests of a fixed predefined length. However, its utility can be extended to include an RNG, which can be used to generate high-quality keys/seeds required by the AES-CORE CTR mode. For that reason, the output register of the KECCAK-CORE is not only connected to the device output ports but also to the input port of the AES-CORE. An overall system diagram is seen in Fig. 3.

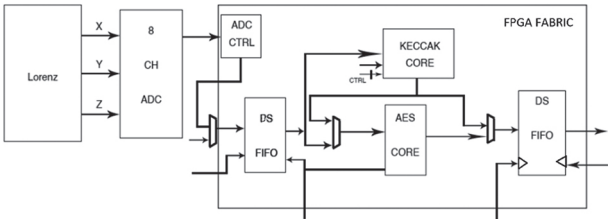

Fig. 3. Multipurpose Cryptographic System Overview

The system is synchronized to an external interfacing circuitry by two Double Sided FIFOs (First-In-First-Out), one for the input ports and one for the output ports. The input DSFIFO is a 16-bit write-side and 128-bit read-side (which is equal to the system's data bus width). The DSFIFO-based synchronization is dedicated only to data. Control signals synchronization, on the other hand, is performed with register synchronization chains. The output DSFIFO's write-side is 128-bit muxed with the output registers of both cores, meanwhile, the read-side is 16-bit.

The on-board ADC is used to provide the system with accessibility to physical entropy sources (Lorenz chaotic circuit). It is an 8-channel 12-bit SAR clocked at 1Mhz. The system can be adjusted to support any number of ADC channels with FIFO synchronization, however, the implemented system supports only one channel for the sake of simplicity and thus only one of the three system's variables can be sampled at once. Interfacing the ADC is done using IntelFPGA ADC Controller IP (Intellectual Property), which is very straightforward. The IP provides parallel access to all channels simultaneously. The channel is muxed into the input FIFO of the system. The detailed RTL view of the system is shown in Fig. 4. The following subsections will explain the implementation details of each part of the system.

4.1. AES-CORE IMPLEMENTATION

4.1.1. Non-pipelined CTR AES-128: AES-128 requires successive ten rounds that only differ in the round key derived from the KeyExpansion. Each round key is calculated based on the previous key and a unique round constant. The KeyExpansion procedure in the ten rounds is also identical. The four round steps alongside the key scheduler were implemented as combinational logic. SubBytes step was implemented as 16 parallel LUTs representing the S-box table. ShiftRows is a rewiring of the state, hence, its implementation is straightforward and requires no hardware. MixColumns step requires multiplication by "two" and "three" in $GF(2^8)$ which is achieved easily since multiplication by "two" is a shift to the left while multiplication by "three" is a shift and XOR. The AddRoundKey step is an XOR operation between the state and the round key. Non-pipelined AES128 core is implemented with a single block of round encryption and KeyExpansion. Round constants are provided by an FSM along with a control signal to exclude the MixColumns step from the 10th round. The ciphertext is a result of the plaintext XORed with the latched output of round 10; the 32-bit counter of CTR mode is then incremented, concatenated with the 96-bit nonce, and fed into the round block. The 128-bit encryption requires 11 clock cycles and the next plaintext should be provided during that window. A similar core was used in [39] to enc/dec real-time video data as an example application.

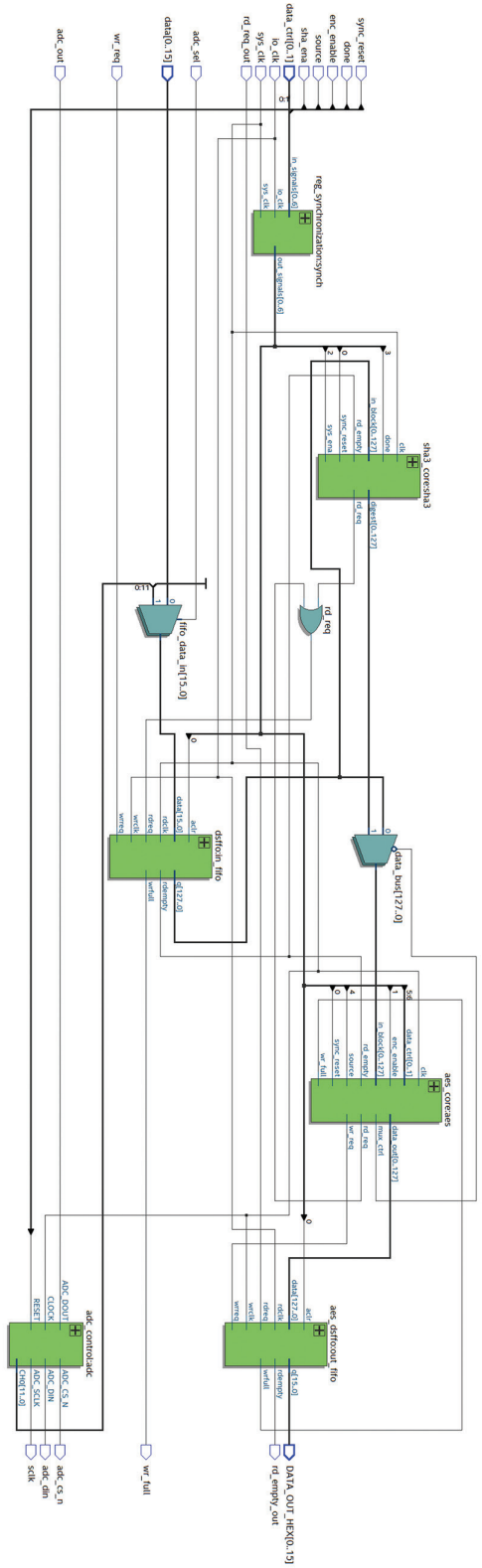

Fig. 4. RTL view of the on-FPGA system

4.1.2. Pipelined CTR AES-128: AES's implementation has a wide range of requirements and constraints on throughput and area. For the implementation to be compatible with high-speed applications, high throughput architectures are required. One method to improve the throughput is pipelining. This design is obtained by placing registers between the unrolled encryption and Key Scheduler rounds. Encryption and KeyExpansion rounds are identical to those of the non-pipelined variant. However round-key and control signals are hard-wired in each round and no FSM control unit is needed, hence, minimizing the critical path delay. Encryption rounds run parallel to the KeyExpansion. Therefore, timing has to be exact for each round key from the RoundExpansion to arrive at its corresponding encryption round. The key, seed, and plaintext registers are multiplexed into the input port of the core along with the corresponding control signals for compactness. The input port itself is demultiplexed to the output of the Sponge PRNG and the device input pins. Therefore, both seed and key can be obtained from the RNG or provided by an external interfacing circuitry. Double Sided FIFOs are used to separate the clock domains. This architecture encrypts a 128-bit block every clock cycle with a latency of 11 clock cycles.

4.2. KECCAK-CORE IMPLEMENTATION

For the implementation of Keccak-f, the straightforward unfolded structure was chosen as seen in Fig. 5.

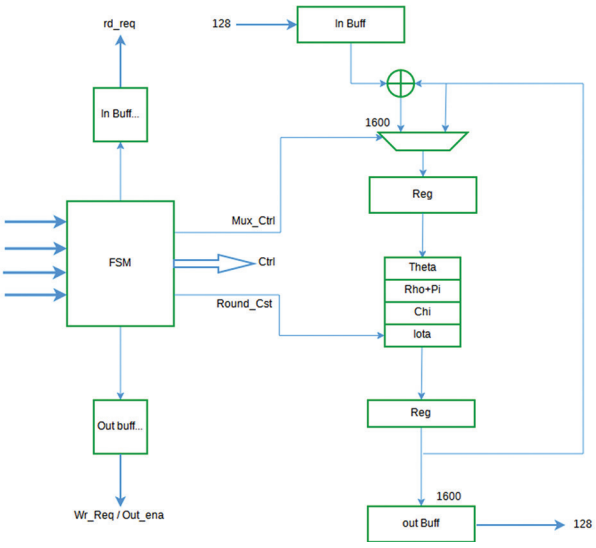

Fig. 5. KECCAK-CORE Implementation

A multiplexer controlled by a round counter determines whether the state is updated with the result of the round function or with the new input block during the absorption phase. The input block is stored in an input buffer in a form of a SIPO (Serial-In-Parallel-Out) shift register. At every clock, 128-bit input enters the register in parallel with Keccak-f computation to avoid overhead. The state is stored in a 1600-bit register that is updated after each round. The five steps forming the round function are implemented using combinational logic with no inner round pipelining.

The round constants are provided by an FSM. After the last round, the state is truncated to 256-bit to output the hash digest.

The number of rounds n_r is user-defined with two constraints: it must be at least 24, and the number of rounds can only be multiples of 12. The rationale for that is not a security concern. It is related to flexibility and usability of the implementation in applications where slowing down the hash operation is beneficial. The user-defined n_r mode is obtained by reusing the states in the FSM. The control unit will loop indefinitely between the 24 rounds' states as long as a control signal is pulled high. The control signal is checked at the 12^{th} and 24^{th} states, which explains the multiples of 12. The output digest buffer is connected to the output DSFIFO of the device and the input port of the AES-CORE.

Four modes of operation for the KECCAK-CORE are defined. First of which is a '*randomized hash function*' where data is fed as 16-bit chunks to the input DSFIFO of the device and should be padded beforehand. The input bits are first registered in the input buffer of KECCAK-CORE before being forwarded to the absorption phase of Keccakf [$b = 1600, n_r = 24$]. A digest is then obtained using this configuration. Note that the values of *c* and *r* have not been stated, even though they define the level of security/performance of the hash function and the SHA-3 hash variant (output length), because they are user-defined. The second mode of operation is a '*slow one-way function/key derivation function*', which is obtained by simply increasing n_r, hence, slowing down the function. The remaining two modes are a '*reseedable PRNG*' and a '*post-processing unit for TRNG*', which are explained in the next subsection.

4.3. RNG IMPLEMENTATION

4.3.1. Sponge-based PRNG: The PRNG design should be flexible enough in terms of performance and security to suit most cryptographic applications. For instance, the requirements of generating random bits for nonces are different from those to be used as cryptographic keys and so on. Furthermore, since RNGs require sources of entropy as mentioned earlier, the implemented RNG's parameters have to be tunable according to the available entropy source. Lastly, the RNG has to have access to multiple physical entropy sources simultaneously while keeping the implementation as compact as possible. All of this can be achieved by tweaking the Keccak hash function.

First, the security/performance tradeoffs of the PRNG should be adjustable according to the application. As mentioned earlier, the capacity value determines a ceiling to the security level that the sponge function provides [29] and defines the resistance of the construction to state recovery attacks; an increase in capacity *c* implies a decrease in the bitrate *r* and a drop in performance occurs as a result. One practical way to adjust the capacity/bitrate is to alter the input buf-

fer shift register in KECCAK-CORE by masking the last c bits. This method is quite simple and requires minor resources. However, a huge entropy loss is conceived due to the masked bits being no longer stored in the input DSFIFO. This entropy loss is not tolerated in the case of large capacities or when a continuous flow of entropy from the sources is required.

A better approach is to control the read signal from the KECCAK-CORE to the input DSFIFO. By tweaking the control unit at a few rounds' states, the read signal is pulled high at a given interval of rounds defined by an external 3-bit signal. The featured capacity values are {256,1024} with a stride of 128. Secondly, since the number of rounds dictates the security margin, it is adjustable as mentioned earlier. This may also be helpful in the case of entropy sources with high correlations even though Keccak's security margin is already thick and should fulfill its security claims even in the case of a decrease in the number of rounds [40].

Lastly, the implemented RNG is reseedable, meaning that an additional entropy source can be added after random bits have been generated. Instead of throwing away the current state of the PRNG, reseeding combines the current state of the generator with the new seed material [41]. The implementation can be operated as a reseedable sponge PRNG, using the sponge function of KECCAK-CORE in a cascaded way without the state being reinitialized. This implementation features two requests: feed and fetch. At first, the seed material is fed to the input DSFIFO as chunks of 16 bits before them being absorbed by the sponge's input buffer shift register as chunks of 128 bits. When the input seed length *l* is equal to *r*, where *r* is user-defined as discussed earlier, the input buffer is XORed with the current state and forwarded to Keccak-f. This sequence is repeated until the DSFIFO issues a read-empty signal indicating that the seed material is completely absorbed. Next, the sponge is switched to the squeezing phase where the desired output length is obtained [32]. After applying the iterated Keccak-f, the state is stored in a shift register that is controlled externally to iterate through the desired length of the output $l \leq r$. The output shift register is connected to a 128-bit digest buffer that feeds to an output DSFIFO with a 128-bit write-side and 16-bit read-side controlled by the user to fetch the random bits.

One can fetch one random bit directly after feeding each seed with length $l < r$ through Keccak-f[$b = 1600$]. In that case, the implementation is operating in the duplex construction mode. Unlike a sponge function that is stateless in between calls, the duplex construction accepts calls that take an input string and return an output string depending on all inputs received so far. The instance of the duplex construction (Fig. 6) is known as a duplex object [41]. This can go further by letting duplex objects non-identical to one another. This asymmetry is supported by the implementation and can be obtained by altering the capacity and number of rounds control signals.

The output of the PRNG is connected to the multiplexed input port of AES-CORE to deliver the generated 128-bit keys and 96-bit nonces required for the current encryption session. The Sponge performance/security parameters must be chosen accordingly.

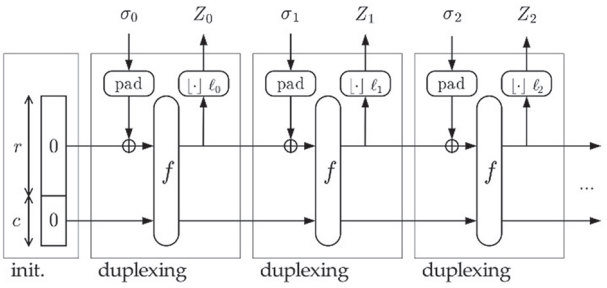

Fig. 6. The Duplex Construction [30]

4.3.2. Post-processing the Lorenz System: Chaotic dynamical systems are chosen as an entropy source for their unpredictability over sufficiently long periods. The Lorenz system was selected owing to its popularity and simple analog implementation. Fig. 7 shows an analog Lorenz circuit with non-linear feedback loops from the outputs of integrator circuits to two analog multipliers AD633. The output of the integrators (implemented using LM358P op-amps) represent the values *x, y,* and *z* of the Lorenz system given by Equation 2. It can be noted that this system does not exhibit chaotic behavior until a specific range of values for the system parameters *ρ, σ,* and *β*. These physical parameters are set by the values of capacitors and resistors, hence, they can be tuned. The range of voltages can be adjusted to suit the ADC range by changing the resistance connected to the op-amp inputs [42].

$$\begin{aligned} \frac{dx}{dt} &= \frac{1}{R_1 C}.(y-x) \\ \frac{dy}{dt} &= \frac{1}{R_3 C}.x - \frac{1}{R_5 C}.y - \frac{1}{100 R_4 C}.xz \\ \frac{dz}{dt} &= \frac{1}{100 R_6 C}.xy - \frac{1}{R_7 C}.z \end{aligned} \tag{2}$$

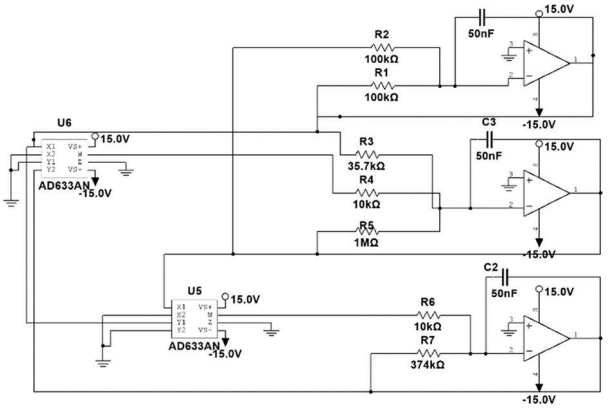

Fig. 7. Analog Lorenz circuit (*ρ=28, σ=10, β=8/3)*

The outputs *x,y,* and *z* can be connected to three ADC channels of the FPGA board. In the conducted experiments, only the *x* output is connected and sampled with a frequency of *1Mhz* even though the power spectrum of

the circuit is concentrated on the lower frequencies. This will eventually lead to correlations in the values of *x* due to continuity, which will be more apparent due to SAR's quantization loss. Also, relying on two signals simultaneously does not increase the source's entropy due to the high mutual entropy between any pair of variables of Lorenz equations. If multiple physical processes are desired, one way is to take advantage of causal chains with a high degree of independence [43].

The *das* suffers from statistical defects, which is a manifest of the system's continuities and autocorrelations due to the system's memory. Another plausible problem is the upper bound limit on entropy which was shown in the works of [44], [45] on chaotic univariate maps. Fortunately, the two works are not concerned with continuous chaotic multivariate systems, and no work disproved the conjecture for this category. The entropy source may eventually produce zero information asymptotically [46]. In addition, a good evaluation of the TRNG's entropy takes into account the sampling frequency, the ADC resolution, the aging of components that may cause the noise source to completely break down, and the non-ideal behavior of the different components (for instance the low-pass behavior of the op-amps used) [43]. Investigating this point is out of the scope of the present work. This work, however, provides an assessment of the performance of the KECCAK-CORE post-processing step on eliminating statistical defects and examines the results of the RNG in non-ideal conditions.

A sponge-based TRNG design is straightforward and similar to that of a sponge-based PRNG. The major difference is that in the process of producing true random numbers, the KECCAK-CORE's sponge function resembles a post-processing step. The number of rounds is fixed to a minimum of 24 rounds. In contrast to PRNG mode, no output is squeezed before the end of absorbing the entire input block of the desired entropy, and the state is reinitialized afterward. This implies that there is no need for the capacity parameter *c* and the bitrate *r* to be set to meet the performance requirements of the application. However, if the user does not discard the state, or if the mode is switched to a PRNG seeded with the physically generated das, the user should readjust the capacity *c* to provide the resistance required against attacks.

5. RESULTS AND DISCUSSION

All cores were manually implemented in VHDL (VHSIC Hardware Description Language) and synthesized using Intel Quartus Prime 20.1 on Cyclone-V FPGA (5CSXFC6D6F31C6) using the DE-10 Standard board.

5.1. AES-CORE

The non-pipelined AES-CORE hit an f_{max} of 215.3Mhz which implies a critical path delay of 4.64ns from the control unit state register to the datapath register before the encryption unit. Each round of encryption is completed in one clock cycle, which means that the entire encryption operation takes 11 clock cycles per block. That yields a maximum throughput of 2.33Gbps. This core is lightweight with 3% area consumption. The synthesis optimization mode was set to 'balanced' and the advanced physical optimization was turned on. In general, this core is low-area with good throughput.

The pipelined AES-CORE hit an f_{max} of 260.92MHz, a 21% improvement over the non-pipelined one. This is because the signals in the pipelined architecture are hardwired in each round and no control unit is needed; thus minimizing the maximum delay on the critical path. The maximum throughput is roughly 31Gbps, which is 13 times faster than its non-pipelined counterpart. As seen in Table 1, the core utilizes around 25kLEs -Logic Elements- (equivalent to 9973ALMs -Adaptive Logic Modules-) which accounts for 23% of the device's total logic elements.

5.2. KECCAK-CORE

The KECCAK-CORE used 8947LEs (3415ALMs) which account for 8% of the FPGA, making it lightweight and suitable for embedded applications. The usage of dedicated logic registers accounts for over 70% of the total logic usage. This is due to the inherited reliance on permutation in the Keccak family. In addition to low-area usage and high flexibility, the core clocks at 271.69MHz.

The maximum throughput of the core is of the form:

$$Throughput = \frac{TP_{max}}{n_r + 1}.(1 - \frac{c}{b}) \quad (3)$$

where $b = 1600$, TP_{max} is a constant that depends on the maximum frequency f_{max}. It can be viewed as the throughput of the core with $c = 0$ and $n_r = 1$, resulting in TP_{max} =404.73Gbps. Of course, operating the core in these conditions is dangerous and never recommended given that the security margin against distinguishers is coupled with the number of rounds [12], [47], [48]. While lowering the capacity decreases the level of attainable security in the sponge construction. Moreover, eliminating the capacity introduces a vulnerability to length extension attacks in Hash-based Message Authentication Codes (HMACs) as well as violating the assumptions made by the original authors in [30]. The values of n_r and b are substituted as defined by NIST's standard [31], and the capacity for each SHA-3 variant is equal to twice the digest length l yielding:

$$Throughput = \frac{TP_{max}}{25}.(1 - \frac{2.l}{1600}) \quad (4)$$

Results obtained for the three SHA-3 variants are shown in Table 1.

Table 1. Results of the AES/SHA-3 implementations

Core	Area (LEs)	fmax (MHz)	Thr. (Mbps)	Mbps/ LE
AES Non-pipelined	3548	215.30	2385.9	0.67
AES Pipelined	25146	260.92	31846.4	1.26
SHA-3 256	8947	271.69	11.00	2.60
SHA-3 384	8947	271.69	8.40	1.98
SHA-3 512	8947	271.69	5.81	1.38

5.3. SPONGE-BASED PRNG

The performance of the PRNG depends on the capacity. However, operating the KECCAK-CORE in a reseedable PRNG mode is fundamentally different from that of a hashing standard mode. Capacity and bitrate can be user-defined, but one should set the parameters along with a proper limit on the number of output blocks squeezed before each seed refresh. For instance, The *KECCAK[r,c]* PRNG provides a resistance of 2^c against state recovery if the number of output bits between times where the state has gained at least *c* bits of fresh seeding material has an upper bound of $r.2^{r/2}$ [32].

Pseudo-random numbers are generated using the previously implemented KECCAK-f[b=1600], more specifically KECCAK[*r=1088, c=512*]. The random sequence is obtained by squeezing the state after providing the empty string as input. The system is clocked at a 100MHz clock frequency. Each squeezed block is 256-bit, taken every 24 clock cycles. If the state is reseeded with *n* high entropy bits, it yields a resistance of 2^n to any state recovery attack. Random bits are extracted using Quartus Prime's Signal Tap Logic Analyzer as shown in Fig. 8. The Figure shows the random number generated by the sponge-based PRNG which is the output state of sha-3 256-bit core (*sha3|in_state[0..255]* at the top) every time sample (24 clock cycles) and its individual output bits (*sha3|in_state[i]*). These generated bits were collected for randomness and entropy analysis.

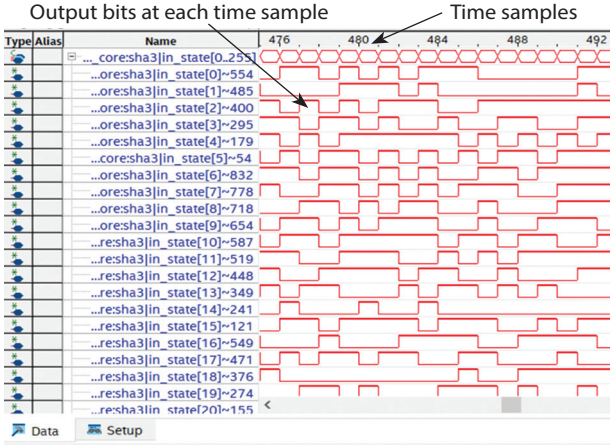

Fig. 8. Random Bits Extraction via Signal Tap

To assess the quality of the pseudo-random numbers generated, the NIST SP 800-22 verification is used [5]. Results in Table 2 indicate that the sequence passed all tests with all P-values > 0.01, which implies that the sequence is random.

On the analog side, the Lorenz chaotic system is implemented using discrete components. Its attractor is observed after plotting variables *x* and *y* in XY mode on a digital oscilloscope as seen in Fig. 9. The *x* variable is connected to an ADC channel. Due to the high sampling frequency which is much higher than the highest frequency component in the signal (< 10Khz which is the cutoff frequency of the LM358P at unity gain), das taken at short periods and fed into the post-processing unit can suffer from serious statistical defects. The same verification tests were also used to assess the quality of the generated random numbers. The verification was performed on 450000 bits without an internal state reset. Results in Table 2 indicate that the random sequence passed all tests.

Table 2. NIST's Statistical tests results

Test	PRNG P-value	TRNG P-value
Frequency Test (Monobit)	0.01397	0.30962
Frequency Test within a Block	0.12058	0.49488
Run Test	0.56053	0.35807
Longest Run of Ones	0.13052	0.12262
Binary Matrix Rank	0.29228	0.12260
Discrete Fourier Transform	0.69992	0.38835
Non-Overlapping Template Matching	0.36968	0.54103
Overlapping Template Matching	0.84336	0.13222
Maurer's Universal	0.31458	0.16966
Linear Complexity	0.22099	0.41366
Serial test	0.70625	0.51154
Approximate Entropy	0.17980	0.04060
Cumulative Sums (Forward)	0.01225	0.43620
Cumulative Sums (Reverse)	0.01225	0.54158
Random Excursions	0.14555	0.28605
Random Excursions Variant	0.26355	0.83167

Fig. 9. The Lorenz Attractor XY on the oscilloscope

The throughput of the post-processing unit depends mainly on the physical process itself and the resolution of the ADC. If the process's frequency content is assumed to be at frequencies higher than the sampling frequency with high entropy, then the TRNG is bottlenecked by the board's SAR ADC maximum sampling frequency 20Mhz, in this case, the throughput will be around 12.71Mbps. If all 8 channels are used (by taking advantage of causal chains), the throughput is around 101.72Mbps. Moreover, if better precision is required, a higher-resolution ADC can be utilized. However, the sampling frequency is limited by the system's maximum clock frequency which is in the order of 100Mhz. If the interfacing circuit's frequency goes higher than

that, no timing violation will occur because of the high maximum frequency of the FIFO's BRAM, however, the interfacing circuit will be severely bottlenecked.

5.5. INFORMATION ENTROPY ANALYSIS

The entropy of the TRNG sequence is estimated using Maurer's universal statistical test [49]. The test is performed on a sequence of length n (in this case n = 450000 bits). Practically, the compression estimator of NIST 800-90B [50] was used, which is an extension of Maurer's test to approximate the lower bound on the min-entropy. The algorithm is described in detail in [50], [51]. The min-entropy ensures a conservative measurement of the per-bit entropy of the source since it corresponds to the difficulty of predicting the most likely outcome. Finally, the min-entropy value is found to be H=5.61 per 6-bit, which corresponds to a per-bit min-entropy of 0.935.

The min-entropy calculated is lower than the ideal value of one. However, considering the high-frequency sampling of the low-frequency implementation of the analog Lorenz system, in addition to the low-resolution ADCs coupled with the low pass behavior of the op-amps used, the achieved entropy result can be considered respectable. Also, the sponge-based post-processing demonstrated its effectiveness in removing statistical defects in non-ideal conditions. However, better results can be achieved by feeding the system with high-frequency physical sources.

5.6. FPGA RESOURCE USAGE

The proposed design is composed of a tunable Keccak core and a pipelined AES-128 core. Individually, the cores utilize 3415 + 9973 = 13388ALMs (6024 Slices). However, the synthesized complete design consumed only 10310ALMs (4639 Slices), a 23% area reduction. This indicates an efficient logic packing of the two cores in the target device. The post-fitting netlist overall logic usage is 26 % for the pipelined AES variant and 11% for the non-pipelined variant, less than 1% of Block RAM, and 10% of the device's total pins. Table 3 summarises the FPGA post-place-and-route resource usage of the whole system.

Table 3. FPGA resource usage summary

Resource	Usage	%
ALMs used in final placement	10,310 / 41,910	25
Dedicated logic registers	7,167 / 83,820	9
Combinational ALUT usage for logic	12,606	
-- 7 input functions	0	
-- 6 input functions	6,907	
-- 5 input functions	1,540	
-- 4 input functions	1,966	
-- <=3 input functions	2,193	
M10k blocks	8 / 553	1
Total block memory bits	12,288/ 5,662,720	<1
I/O pins	50 / 499	10

6. COMPARISON TO RELATED WORK

To date, the work presented in [1] is the only study that combined AES with SHA3-256 with the design being resource-shared. Most reported works on AES/Hash function designs tend to resource-share AES with Grøstl due to their common structure similar to the case of Fuge and Whirlpool hash functions. The resource-shared implementations are compact and generally more hardware efficient. For a fair comparison, the Throughput-Per-Slice (TPS) metric is calculated for the aforementioned resource-shared cryptosystem designs, and then compared with the proposed design herein in both hashing and symmetric encryption.

Most of the resource-shared works were synthesized on Xilinx Virtex-5/6 families. Meanwhile, the present implementation was performed on the IntelFPGA Cyclone-V device. Thus, an equivalent conversion between Slice and Logic Element is necessary. A single ALM is equivalent to 0.45 Virtex-5/6 slice and Cyclone-III/IV's LE is equivalent to 0.12 Virtex-5/6 slice [52], while 1 BRAM is equivalent to 128 slices [53]. Another note regarding comparing device performance, the Cyclone-V FPGA is a speed grade 6 device, meaning that it supports a maximum global clock frequency of 550Mhz which exactly matches Virtex-5's max frequency, whereas, 601Mhz is the maximum global frequency of the Virtex-6 devices. This would hint that the obtained throughput figures might be higher should Virtex-6 FPGA be used.

The comparison of the proposed design with other AES/Hash-function works is demonstrated in Table 4. The proposed design recorded the highest area utilization, a few times higher than the compact AES/Grøstl designs and 1.89× higher area utilization than the highest throughput AES/Grøstl reported in [11], and 1.35× higher than that of AES/SHA3-256 design presented in [1]. However, the recorded results are anticipated due to the relatively complex nature of both cores (pipelining of AES and flexibility of Keccak). Moreover, the works presented in the table are resource-shared. Consequently, they are supposed to yield more compact implementations. However, despite the flexibility of the presented KECCAK-CORE, it yields several times higher throughput than all Grøstl, Whirlpool, and Fuge hash functions when operating as SHA3-256. The throughput of the tunable KECCAK-CORE is close to that of the untunable core presented in [1], which is, to date, the highest hashing throughput recorded in AES/hash-function designs on FPGAs.

The proposed system achieved the second-highest TPS of 2.43, which is higher than all reported Grøstl implementations. Whereas for AES enc/dec, the highest throughput and TPS figures were recorded with 3.79× and 2.8× higher encryption throughput and TPS respectively than those reported in [1].

Table 4. Results comparison with other unified designs

Study	Device	Cores	Area (LE/Slice+BRAM)	Equiv. Slices	TP (Mbps)	TPS (Mbps/Slice)
Järvinen [6]	Cyclone-III	Fuge-256	4520LEs	552	972	1.76
		AES-128 enc			778	1.41
Kochar [54]	Virtex-5	Whirlpool	6742LUTs	2247	410	0.18
		AES-128 enc			205	0.09
At [8]	Virtex-7	Grøstl-256	185+1	313	98	0.31
		AES-128 enc/dec			229	0.73
Pelnar [9]	Virtex-6	Grøstl-256	302+0	302	13.24	0.04
		AES-128 enc			13.8	0.05
		AES-128 dec			9.99	0.03
Guo [10]	Cyclone-IV	Grøstl-256 + 4×AES-128 enc	15135LEs	1847	3877	2.10
Järvinen [6]	Cyclone-III	Grøstl-256	13723LEs	1675	1434	0.86
		4×AES-128 enc			2869	1.71
Rogawski [55]	Cyclone-III	Grøstl-256 + 4×AES-128 enc/dec	23758LEs	2851	2378	0.83
Rogawski [11]	Virtex-6	Grøstl-256 + 4×AES-128 enc/dec	2447+0	2447	4212	1.72
Kundi [1]	Virtex-6	SHA3-256	1380+16	3428	14876.13	4.34
		4×AES-128 enc/dec			8400.64	2.45
This work	Cyclone-V	Keccak (tunable)	10310ALMs	4639	11264	2.43
		Pipelined AES-128			31846	6.86

7. CONCLUSION

A multi-purpose cryptographic design has been presented consisting of a 128-bit AES-CORE symmetric encryption and a tunable KECCAK-CORE that can be user-configured by modifying the capacity, bitrate, and the number of rounds. The design achieved the second-highest TPS of 2.43 compared to other works, with an area usage of roughly 26% of the low-cost Cyclone-V FPGA. In addition to the basic functions of symmetric encryption and hash function, the system was also operated as a reseedable PRNG and a post-processing unit for a chaotic-based TRNG. Both PRNG and TRNG random sequences successfully passed NIST's statistical tests. The information entropy analysis performed on the post-processed TRNG sequences indicates a respectable average of H=5.61 per 6-bit (0.935), suggesting that the sponge-based post-processing showed its effectiveness in removing statistical defects in non-ideal conditions. In future work, a more efficient design could be investigated by utilizing the sponge as a symmetric key algorithm.

Acknowledgement

The authors acknowledge the support of Wameedth Scientific Club for providing some of the components and IntelFPGA University program for donating the DE-10 board.

8. REFERENCES

[1] D.-S. Kundi, A. Khalid, A. Aziz, C. Wang, M. O'Neill, W. Liu, "Resource-Shared Crypto-Coprocessor of AES Enc/Dec with SHA-3", IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 67, No. 12, 2020, pp. 4869-4882.

[2] K. Shahzad, A. Khalid, Z. E. Rákossy, G. Paul, A. Chattopadhyay, "CoARX: A Coprocessor for ARX-based Cryptographic Algorithms", Proceedings of the 50th IEEE Design Automation Conference, Austin, TX, USA, 29 May 2013, pp. 1-10.

[3] P. Nannipieri, S. D. Matteo, L. Baldanzi, L. Crocetti, L. Zulberti, S. Saponara, L. Fanucci, "VLSI Design of Advanced-Features AES Cryptoprocessor in the Framework of the European Processor Initiative", IEEE Transactions on VLSI Systems, Vol. 30, No. 2, 2022, pp. 177-186.

[4] B. Ilyas, S. M. Raouf, S. Abdelkader, T. Camel, S. Said, H. Lei, "An Efficient and Reliable Chaos-Based IoT Security Core for UDP/IP Wireless Communication", IEEE Access, Vol. 10, 2022, pp. 49625-49656.

[5] L. Bassham et al. "A statistical test suite for random and pseudorandom number generators for cryptographic applications", https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762 (accessed: 2022)

[6] K. Järvinen, "Sharing Resources Between AES and the SHA-3 Second Round Candidates Fugue and Grøstl", Proceedings of the 2nd SHA-3 Candidate Conference, 23 August 2010, p. 2.

[7] M. Sonmez, R. Perlner, L. Bassham, W. Burr, D. Chang, S. jen Chang, M. Dworkin, J. Kelsey, S. Paul, R. Peralta, "Status report on the second round of

the SHA-3 cryptographic hash algorithm competition", NIST, Technical Report 7764, 2011.

[8] N. At, J. L. Beuchat, E. Okamoto, I. San, T. Yamazaki, "A low area unified hardware architecture for the AES and the cryptographic hash function Grøstl", Parallel and Distributed Computing, Vol. 106, 2017, pp. 106-120.

[9] M. Pelnar, M. Muehlberghuber, M. Hutter, "Putting together What Fits together - GrÆStl", Smart Card Research and Advanced Applications, Springer Berlin Heidelberg, 2013, pp. 173-187.

[10] K. Guo, H. M. Heys, "A Pipelined Implementation of the Grøstl Hash Algorithm and the Advanced Encryption Standard", Proceedings of the 26th IEEE Canadian Conference on Electrical and Computer Engineering, Regina, Canada, 5-8 May 2013, pp. 1-4.

[11] M. Rogawski, K. Gaj, E. Homsirikamol, "A High-Speed Unified Hardware Architecture for 128 and 256-bit Security Levels of AES and the SHA-3 Candidate Grøstl", Microprocessors and Microsystems, Vol. 37, No. 6, 2013, pp. 572-582.

[12] S. jen Chang, R. Perlner, W. Burr, M. Sonmez, J. Kelsey, S. Paul, L. Bassham, "Third-round report of the SHA-3 cryptographic hash algorithm competition", NIST, Technical Report 7896, 2012.

[13] K. E. Ahmed, M. M. Farag, "Hardware/software co-design of a dynamically configurable SHA-3 System-on-Chip (SoC)", Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems, Cairo, Egypt, 6-9 Dec 2015, pp. 617-620.

[14] H. Mestiri, I. Barraj, M. Machhout, "A High-Speed KECCAK Architecture Resistant to Fault Attacks", Proceedings of the 32nd International Conference on Microelectronics, Aqaba, Jordan, 14-17 December 2020, pp. 1-4.

[15] T. Newe, M. Rao, D. Toal, G. Dooly, E. Omerdic, A. Mathur, "Efficient and High Speed FPGA Bump in the Wire Implementation for Data Integrity and Confidentiality Services in the IoT", Sensors for Everyday Life: Healthcare Settings, Springer International Publishing, 2017, pp. 259-285.

[16] R. Shahid, M. Sharif, M. Rogawski, K. Gaj, "Use of Embedded FPGA Resources in Implementations of 14 Round 2 SHA-3 Candidates", Proceedings of the International Conference on Field-Programmable Technology, New Delhi, India, 12-14 December 2011, pp. 1-9.

[17] K. Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, M. U. Sharif, "Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using Xilinx and Altera FPGAs", Cryptology ePrint Archive, Paper 2012/368, 2012.

[18] B. Jungk, M. Stottinger, M. Harter, "Shrinking KECCAK Hardware Implementations", Proceedings of the SHA-3 2014 Workshop, Univeristy of California, Sanata Barbara, CA, USA, 22 Aug 2014.

[19] A. Soltani, S. Sharifian, "An Ultra-high Throughput and Fully Pipelined Implementation of AES Algorithm on FPGA", Microprocessors and Microsystems, Vol. 39, No. 7, 2015, pp. 480-493.

[20] X. Zhang, M. Li, J. Hu, "Optimization and Implementation of AES Algorithm based on FPGA", Proceedings of the IEEE 4th International Conference on Computer and Communications, Chengdu, China, 7-10 December 2018, pp. 2704-2709.

[21] R. Farashahi, B. Rashidi, S. Sayedi, "FPGA based Fast and High Throughput 2-Slow Retiming 128-bit AES encryption algorithm", Microelectronics Journal, Vol. 45, No.8, 2014, pp. 1014-1025.

[22] T. Tuncer, E. Avaroğlu, "Random Number Generation with LFSR based Stream Cipher Algorithms", Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 22-26 May 2017, pp. 171-175.

[23] H. Tang, T. Qin, Z. Hui, P. Cheng, W. Bai, "Design and implementation of a configurable and aperiodic pseudo random number generator in FPGA", Proceedings of the IEEE 2nd International Conference on Circuits, System and Simulation, Guangzhou, China, 14-16 July 2018, pp. 47-51.

[24] R. Hobincu, O. Datcu, "FPGA Implementation of a Chaos based PRNG Targeting Secret Communication", Proceedings of the International Symposium on Electronics and Telecoms, Timisoara, Romania, 8-9 November 2018, pp. 1-4.

[25] S. Gomar, M. Ahmadi, "A digital pseudo random number generator based on a chaotic dynamic

system", Proceedings of the 26th International Conference on Electronics, Circuits and Systems, Genoa, Italy, 27-29 November 2019, pp. 610-613.

[26] B. Paul, G. Trivedi, P. Jan, Z. Němec, "Efficient PRNG design and implementation for various high throughput cryptographic and low power security applications", Proceedings of the 29th International Conference Radioelektronika, Pardubice, Czech Republic, 16-18 April 2019, pp. 1-6.

[27] S. Łoza, Ł. Matuszewski, "A true random number generator using ring oscillators and SHA-256 as post-processing", Proceedings of the International Conference on Signals and Electronic Systems, Poznan, Poland, 11-13 September 2014, pp. 1-4.

[28] NIST, Security requirements for cryptographic modules, https://doi.org/10.6028/NIST.FIPS.140-2 (accessed: 2022)

[29] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, "The KECCAK SHA-3 Submission", https://keccak.team/files/Keccak-submission-3.pdf (accessed: 2022)

[30] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, "Cryptographic sponge functions", https://keccak.team/files/CSF-0.1.pdf (accessed: 2022)

[31] M. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions", https://doi.org/10.6028/NIST.FIPS.202 (accessed: 2022)

[32] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, "Sponge-based Pseudo-Random Number Generators", Cryptographic Hardware and Embedded Systems, Springer Berlin Heidelberg, 2010. , pp. 33-47

[33] H. Shimakage, Y. Tamura, "Chaotic oscillations in Josephson Junctions for Random Number Generation", IEEE Transactions on Applied Superconductivity, Vol. 25, No. 3, 2015, pp. 1-4.

[34] J. B. Johnson, "Thermal Agitation of Electricity in Conductors", Physical Review, Vol. 32, No. 1, 1928, pp. 97-109.

[35] M. Stipčević, "Fast Nondeterministic Random Bit Generator based on Weakly Correlated Physical Events", Review of Scientific Instruments, Vol. 75, No. 11, 2004, pp. 4442-4449.

[36] S. H. Strogatz, "Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering", 2nd Edition, CRC Press, 2015.

[37] P. Gaspard, "Chaos, Scattering and Statistical Mechanics", Cambridge University Press, 2005.

[38] E. Barker, J. Kelsey, Recommendation for random number generation using deterministic random bit generators, https://doi.org/10.6028/NIST.SP.800-90Ar1 (accessed : 2022)

[39] A. Maache, A. Touati, A. Ouali, "Implementation of an AES-based Real-time Video Encryption/Decryption using FPGA/HPS", Proceedings of the 19th International Multi-Conference on Systems, Signals & Devices, Setif, Algeria, 6-10 May 2022, pp. 702-706.

[40] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, R. Van Keer,B. Viguier, "Kangarootwelve: Fast hashing based on keccak-p", Applied Cryptography and Network Security, Springer International Publishing, 2018, pp. 400-418.

[41] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications", Selected Areas in Cryptography, Springer International Publishing, 2012, pp. 320-337.

[42] Horowitz Group, Build a Lorenz attractor, http://seti.harvard.edu/unusual_stuff/misc/lorenz.htm (accessed: 2022)

[43] W. Schindler, W. Killmann, "Evaluation Criteria for True (Physical) Random Number Generators used in cryptographic applications", Cryptographic Hardware and Embedded Systems - CHES 2002, Springer Berlin Heidelberg, 2003, pp. 431-449.

[44] G. J. Wallinger, Information theory and chaotic systems, https://www.spsc.tugraz.at/sites/default/files/InformationTheory_ChaoticSystems_final.pdf (accessed: 2022)

[45] C. Liu, L. Ding, Q. Ding, "Research About the Characteristics of Chaotic Systems based on Multiscale Entropy", Entropy, Vol. 21, No. 7, 2019, p. 663.

[46] M. Stipčević, Ç. K. Koç, "True Random Number Generators", Open Problems in Mathematics and Computational Science, pp. 275-315, Springer, 2014.

[47] C. Boura, A. Canteaut, C. D. Canniere, "Higher-order differential properties of Keccak and Luffa", Fast Software Encryption, Springer Berlin Heidelberg, 2011, pp. 252-269.

[48] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, "Note on zero-sum distinguishers of Keccak-f", https://keccak.team/files/NoteOnKeccakParametersAndUsage.pdf (accessed: 2022)

[49] U. M. Maurer, "A Universal Statistical Test for Random Bit Generators", Journal of Cryptology, Vol. 5, No. 2, 1992, pp. 89-105.

[50] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, M. Boyle, "Recommendation for the entropy sources used for random bit generation", NIST Special Publication, Vol. 800, No. 90B, 2018, p. 102.

[51] Y. Kim, C. Guyot, Y.-S. Kim, "On the Efficient Estimation of Min-Entropy", IEEE Transactions on Information Forensics and Security, Vol. 16, 2021, pp. 3013-3025.

[52] Intel Corporation, "Stratix III FPGAs vs. Xilinx Virtex-5 devices: Architecture and performance comparison", White paper WP01007, 2007.

[53] P. Bulens, F. X. Standaert, J. J. Quisquater, P. Pellegrin, G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs", Progress in Cryptology, Springer, 2008, pp. 16-26.

[54] T. Kochar, S. Nandi, S. Biswas, "A single chip implementation of AES cipher and Whirlpool hash function", Proceedings of the Annual IEEE India Conference, India, 18-20 December 2009, pp. 1-4.

[55] M. Rogawski, K. Gaj, "A high-speed unified hardware architecture for AES and the SHA-3 candidate Grøstl", Proceedings of the 15th Euromicro Conference on Digital System Design, Cesme, Turkey, 5-8 September 2012, pp. 568-575.