An FPGA Implementation of the SHAKE256 Cryptographic Hash Function Based on the Keccak-f[1600] Permutation

Dr. Mohammad H. Awedh

Associate Professor, Department of Electrical and Computer Engineering, Faculty of Engineering, King Abdulaziz University, Saudi Arabia

Email: mhawedh@kau.edu.sa

Received: 30 January 2026
First Decision: 2 February 2026
Revised: 4 February 2026
Revised: 22 February 2026
Accepted: 25 February 2026
Published: 5 March 2026

Abstract:

This paper presents an area-efficient hardware implementation of the SHAKE256 cryptographic hash function based on the Keccak-f[1600] permutation, targeting resource-constrained FPGA platforms. The proposed design employs a modular Verilog HDL architecture centered on a single permutation core controlled by finite state machines, prioritizing logic reuse, architectural clarity, and full standards compliance over aggressive pipelining. Unlike high-throughput architectures that rely on extensive parallelism and large silicon footprints, this implementation emphasizes balanced trade-offs between performance, area, power consumption, and design transparency. The design fully complies with the SHA-3 specification defined in FIPS 202 and supports fixed-length SHAKE256 output generation. The complete system is synthesized and implemented on an Altera Cyclone II FPGA and validated using official NIST test vectors. Experimental results confirm correct functionality, moderate resource utilization, and competitive area efficiency, demonstrating that SHAKE256 can be reliably deployed on low-cost FPGA platforms for cryptographic prototyping, education, and embedded security applications. Based on the obtained implementation results, the author recommends the adoption of modular and area-efficient architectures for SHA-3 and SHAKE-based cryptographic primitives when targeting resource-constrained FPGA platforms. The presented design demonstrates that a balanced approach emphasizing logic reuse, deterministic control, and architectural clarity can achieve reliable cryptographic functionality without requiring extensive hardware resources or aggressive pipelining.

Keywords: SHAKE256, SHA-3, Keccak-f[1600], FPGA implementation, area-efficient hardware, finite state machine, cryptographic hash functions, hardware security

1. Introduction

Cryptographic hash functions are fundamental components of modern digital security systems, supporting data integrity verification, authentication, digital signatures, secure boot mechanisms, and distributed consensus protocols (Nakamoto, 2008; Paar & Pelzl, 2010). These functions must satisfy stringent security properties, including collision resistance, preimage resistance, and second-preimage resistance, in order to remain robust against increasingly powerful adversaries.

Earlier widely deployed algorithms such as MD5 and SHA-1 have been shown to be vulnerable to practical collision attacks, rendering them unsuitable for contemporary security applications (Dworkin, 2015; NIST, 2012). Although the SHA-2 family introduced stronger constructions and longer digest sizes, concerns regarding structural similarities with SHA-1 motivated the search for a fundamentally different hash design. In response, the National Institute of Standards and Technology (NIST) launched a public competition for a next-generation cryptographic hash function, culminating in the selection of Keccak as the SHA-3 standard (Bertoni et al., 2011a, Bertoni et al., 2011c, NIST, 2015).

SHA-3 departs from the traditional Merkle–Damgård paradigm by adopting the sponge construction, which provides flexibility, inherent resistance to length-extension attacks, and a unified framework for multiple cryptographic primitives (Bertoni et al., 2011b; Courtois, 2011; Daemen & Van Assche, 2011). One of the most significant advantages of SHA-3 is its support for extendable-output functions (XOFs), including SHAKE128 and SHAKE256, which allow variable-length output generation from a single algorithm. This flexibility is particularly valuable in modern cryptographic systems, where different applications may require different digest lengths while relying on the same underlying primitive.

This paper focuses on the FPGA implementation of SHAKE256 using the Keccak-f[1600] permutation. While many existing hardware designs prioritize maximum throughput through deep pipelining and full unrolling, such approaches often incur substantial area and power costs. Instead, this work emphasizes modularity, clarity, and area efficiency, making it particularly suitable for educational use, cryptographic experimentation, and deployment on resource-constrained FPGA platforms.

Despite the increasing adoption of SHA-3 and its extendable-output functions in modern cryptographic systems, efficient hardware implementations remain a challenge, particularly for resource-constrained FPGA platforms. Many existing implementations of Keccak-based hash functions emphasize maximizing throughput through aggressive pipelining, loop unrolling, or extensive parallelism. While these techniques achieve very high performance, they typically incur substantial hardware costs,

increased power consumption, and reduced architectural transparency, which can limit their suitability for low-cost or educational FPGA environments. Conversely, highly compact designs may sacrifice modularity, clarity, or standards compliance in pursuit of minimal resource usage. These limitations highlight the need for a balanced architecture that maintains correctness, portability, and design simplicity while still achieving reasonable performance. Therefore, the objective of this work is to develop a standards-compliant and modular FPGA implementation of SHAKE256 based on the Keccak-f[1600] permutation that emphasizes area efficiency, deterministic control, and architectural clarity while remaining practical for implementation on legacy and resource-limited FPGA devices.

Despite the growing adoption of SHA-3 and its extendable-output functions in modern cryptographic systems, efficient hardware implementations remain challenging for resource-constrained FPGA platforms. Many existing Keccak-based hardware designs focus on maximizing throughput through deep pipelining or full loop unrolling. While such approaches achieve very high performance, they often require substantial logic resources, increased power consumption, and complex control structures, limiting their practicality for low-cost or educational FPGA environments. Conversely, extremely compact designs may sacrifice architectural clarity or modularity, making verification and future extension difficult. These limitations highlight the need for a balanced hardware architecture that maintains standards compliance and design transparency while achieving reasonable performance on modest FPGA platforms. Therefore, the objective of this work is to develop a modular and area-efficient FPGA implementation of the SHAKE256 extendable-output function based on the Keccak-f[1600] permutation, emphasizing deterministic control, resource efficiency, and implementation clarity.

1.1. Contributions

The main contributions of this work are as follows: (1) a complete and standards-compliant hardware implementation of the SHAKE256 extendable-output function based on the Keccak-f[1600] permutation, realized entirely in synthesizable Verilog HDL; (2) a modular, finite-state-machine–controlled architecture that reuses a single permutation core to achieve a balanced trade-off between area efficiency, performance, and design clarity; (3) an FPGA realization targeting a resource-constrained Altera Cyclone II platform, demonstrating portability and practicality on legacy devices without reliance on embedded memory or DSP blocks; and (4) comprehensive functional verification using official NIST test vectors, accompanied by quantitative performance and area comparisons with representative related work. Together, these contributions provide a transparent and reproducible reference design suitable for cryptographic prototyping, education, and low-cost embedded security applications.

2. Background and Related Work

A cryptographic hash function deterministically maps an input message of arbitrary length to a fixed-size or variable-length output while preserving strong security guarantees (Paar & Pelzl, 2010). Any minor modification to the input should result in an unpredictable change in the output, a property known as the avalanche effect. These characteristics make hash functions indispensable in authentication protocols, secure storage systems, and blockchain technologies.

The cryptanalytic breakthroughs that compromised MD5 and SHA-1 highlighted the necessity of conservative design margins and structural diversity in hash function design (NIST, 2012). As computational capabilities continue to advance, particularly in the context of emerging quantum threats, long-term resilience has become a central design objective (Dworkin, 2015; McKay et al., 2017).

SHA-3 is based on the sponge construction, which processes input data through an absorb phase followed by a squeeze phase (Bertoni et al., 2011b). The internal state is partitioned into a rate portion, which interacts directly with input and output, and a capacity portion, which remains hidden and determines the security strength. The separation between rate and capacity enables provable resistance against collision and preimage attacks (Daemen & Van Assche, 2011).

This construction allows a single permutation function to support multiple cryptographic services, including fixed-length hash functions, message authentication codes, and extendable-output functions (NIST, 2016). As a result, SHA-3 provides a flexible and future-proof cryptographic framework.

SHAKE256 is an extendable-output function standardized in FIPS 202 and further specified in SP 800-185 (NIST, 2015; NIST, 2016). It offers a security strength of 256 bits and can generate output streams of arbitrary length, making it suitable for key derivation, post-quantum cryptography, and digital signature schemes. In this work, the output length is fixed to 256 bits to simplify control logic while remaining fully compliant with the standard.

Numerous FPGA implementations of SHA-3 and Keccak have been proposed, ranging from compact iterative architectures to fully unrolled and deeply pipelined designs (Azouaoui et al., 2016; Hodjat & Verbauwhede, 2012; Kerckhof et al., 2012). Fully pipelined implementations achieve extremely high throughput but require substantial logic resources and power consumption. Compact iterative designs trade throughput for reduced area and improved portability. The present work aligns with the latter approach, prioritizing resource efficiency and architectural simplicity.

The security strength of SHAKE256 is derived from the capacity parameter of the sponge construction and the cryptographic properties of the Keccak-f[1600] permutation (Großschädl et al., 2012; Soja & Batina, 2013). In the SHAKE256 configuration, the 1600-bit internal state is divided into a rate of 1088 bits and a capacity of 512 bits, yielding a claimed security strength of 256 bits against collision and preimage attacks (Daemen & Van Assche, 2011; NIST, 2015).

From a hardware perspective, Keccak-f[1600] is well suited for FPGA realization due to its fixed-round structure and absence of data-dependent control flow. Each round applies the same sequence of Boolean operations and rotations, enabling deterministic execution and simplifying finite state machine design. These characteristics also provide a favorable foundation for future side-channel resistance enhancements (Großschädl et al., 2012; Soja & Batina, 2013).

All arithmetic operations are limited to bitwise XOR, AND, NOT, and fixed rotations, which map efficiently to FPGA logic elements without requiring complex arithmetic units.

Fixing the output length to 256 bits reflects common deployment scenarios while reducing control complexity and verification overhead. Importantly, this choice does not weaken security, as the sponge construction preserves cryptographic guarantees regardless of output truncation. Moreover, the modular architecture allows straightforward extension to arbitrary-length output generation in future revisions.

SHAKE256 is increasingly adopted in modern cryptographic systems due to its flexibility, security margin, and suitability for both classical and post-quantum applications. In hardware-oriented systems, SHAKE256 is commonly used as a building block for key derivation functions, digital signature schemes, and hash-based message authentication. Its extendable-output property allows a single implementation to serve multiple cryptographic roles, reducing hardware redundancy in constrained systems.

In post-quantum cryptography, SHAKE256 plays a central role in lattice-based and hash-based schemes, where large quantities of pseudo-random data must be generated efficiently and securely. Hardware accelerators for SHAKE256 are therefore essential components in emerging post-quantum secure platforms, particularly in embedded and IoT devices with limited computational resources.

From a system integration perspective, SHAKE256 is also well suited for secure boot mechanisms, firmware authentication, and integrity verification in reconfigurable hardware systems. FPGA-based implementations enable cryptographic verification to be performed independently of the main processor, improving both security isolation and performance. Additionally, the deterministic

execution and regular structure of Keccak-f[1600] make the algorithm attractive for side-channel–aware hardware designs, where predictability simplifies countermeasure deployment.

These application domains motivate the need for compact, standards-compliant SHAKE256 hardware designs that balance security, performance, and resource utilization. The architecture presented in this work directly addresses these requirements by providing a reusable and transparent implementation suitable for integration into a wide range of hardware security systems.

3. Keccak-f[1600] Permutation

The Keccak-f[1600] permutation operates on a 1600-bit internal state organized as a 5×5 array of 64-bit lanes (Bertoni et al., 2011a). Each permutation consists of 24 rounds, with every round applying five transformations: Theta, Rho, Pi, Chi, and Iota. Theta provides diffusion across columns, Rho performs fixed rotations, Pi permutes lane positions, Chi introduces non-linearity, and Iota injects round constants to break symmetry (Bertoni et al., 2011b; Daemen & Van Assche, 2011).

These transformations collectively ensure strong resistance against linear and differential cryptanalysis while maintaining a regular structure well suited for hardware implementation.

In the proposed design, the entire internal state is stored in registers arranged to reflect the logical lane structure. Each permutation round is executed in a single clock cycle, balancing resource utilization with predictable timing behavior and simplifying verification.

In the proposed design, the entire 1600-bit state is stored in registers arranged to reflect the 5×5 lane structure. This register-based representation allows rapid access and avoids reliance on embedded memory blocks. Each permutation round is executed in a single clock cycle, with all five transformations applied sequentially within the cycle.

The Theta step computes column parity values using XOR reductions and distributes them across the state, providing inter-column diffusion. Rho and Pi are implemented using fixed wiring and barrel shifters, as all rotation offsets and permutations are statically defined by the Keccak specification (Bertoni et al., 2011a). The Chi step introduces non-linearity using simple Boolean operations, while the Iota step injects round constants using a small constant lookup structure.

Executing one round per cycle yields a predictable execution model that simplifies control logic and verification. While this approach sacrifices peak throughput compared to fully unrolled designs, it significantly reduces area consumption and improves design transparency.

Efficient control of the Keccak-f[1600] permutation is critical for achieving predictable timing and minimizing hardware overhead. In the proposed design, finite state machines are used to manage round

execution, data movement, and phase transitions between absorb, permutation, and squeeze operations. This control strategy ensures deterministic execution while avoiding unnecessary duplication of control logic.

The permutation control FSM advances through 24 states corresponding to the 24 rounds of Keccak-f[1600]. Each state triggers the execution of a single round and updates the internal state registers accordingly. Upon completion of the final round, control is returned to the top-level FSM, which determines whether additional absorb or squeeze operations are required. This separation of responsibilities simplifies both design and verification by clearly delineating permutation-level and algorithm-level control.

State registers are updated synchronously on the rising edge of the clock, ensuring consistent timing behavior across all phases of operation. Because the permutation structure and round constants are fixed, no dynamic scheduling or conditional branching is required within the round logic. This property reduces control complexity and eliminates data-dependent timing variations, which is advantageous from both verification and side-channel perspectives.

By employing FSM-based control rather than deeply pipelined scheduling, the design maintains a clear and easily analyzable execution model. This approach aligns well with the design goals of portability, reproducibility, and educational clarity, while still achieving performance levels sufficient for many practical hardware security applications.

4. Hardware Architecture and Design Methodology

4.1. Algorithm Overview

SHAKE256 is an extendable-output function (XOF) defined in the SHA-3 standard and built upon the Keccak sponge construction. The algorithm operates by repeatedly applying the Keccak-f[1600] permutation to a 1600-bit internal state while alternating between absorb and squeeze phases. The sponge construction divides the internal state into two portions: the rate r, which interacts directly with input and output data, and the capacity c, which determines the cryptographic security strength.

In the SHAKE256 configuration, the 1600-bit state is partitioned into a rate of $r = 1088$bits and a capacity of $c = 512$bits. During the absorb phase, padded message blocks of size $r$bits are XORed with the rate portion of the internal state, followed by execution of the Keccak-f[1600] permutation. This process continues until all message blocks have been absorbed.

Once absorption is complete, the algorithm enters the squeeze phase, in which output bits are extracted from the rate portion of the state. If additional output bits are required, the permutation function is

applied again before further extraction. This mechanism enables SHAKE256 to generate output streams of arbitrary length while maintaining strong cryptographic security guarantees.

The Keccak-f[1600] permutation consists of 24 rounds, each composed of five transformations: Theta, Rho, Pi, Chi, and Iota. These operations collectively provide diffusion, nonlinearity, and symmetry breaking within the internal state. Because the transformations rely primarily on simple Boolean operations and fixed rotations, the permutation structure is well suited for efficient hardware implementation on FPGA platforms.

4.2. Overall System Architecture

The overall hardware architecture of the proposed SHAKE256 implementation is designed with a modular structure that emphasizes logic reuse, deterministic control, and efficient resource utilization. The system is centered around a single Keccak-f[1600] permutation core, which is reused during both the absorb and squeeze phases of the sponge construction.

The top-level architecture consists of several functional modules, including the input padding unit, absorb controller, permutation core, squeeze controller, and output truncation module. The padding unit prepares the input message according to the SHA-3 specification by inserting domain-separation bits, zero padding, and the termination bit. The absorb controller manages the XOR integration of padded message blocks into the rate portion of the internal state before triggering the permutation operation. At the core of the system is the Keccak-f[1600] permutation engine, which performs the sequence of round transformations required by the SHA-3 standard. The permutation core processes the internal state iteratively, executing one round per clock cycle. This iterative approach reduces hardware duplication while maintaining predictable timing behavior.

During the squeeze phase, the squeeze controller extracts output bits from the rate portion of the state and manages additional permutations if further output is required. Finally, the truncation module selects the desired 256-bit output to produce the SHAKE256 digest used in this implementation.

A finite state machine coordinates the operation of all modules and ensures correct sequencing between padding, absorption, permutation execution, and output generation. This centralized control simplifies synchronization between data path components and contributes to the modular and scalable nature of the design.

4.3. Keccak-f[1600] Core Architecture

The Keccak-f[1600] permutation core constitutes the central computational component of the proposed SHAKE256 hardware architecture. The permutation operates on a 1600-bit internal state organized as

a 5×5array of 64-bit lanes, which are stored in registers within the FPGA. This register-based representation allows efficient access to each lane and avoids the need for embedded memory resources.

The permutation consists of 24 sequential rounds, where each round applies a fixed sequence of transformations defined in the Keccak specification: Theta, Rho, Pi, Chi, and Iota. These operations collectively provide diffusion, nonlinearity, and symmetry breaking within the internal state.

The Theta transformation computes column parity values using XOR operations across each column of the state matrix and distributes these values to neighboring columns to achieve inter-column diffusion. The Rho transformation performs fixed cyclic rotations on each lane, while the Pi transformation permutes the positions of the lanes within the state array according to a predefined mapping. The Chi transformation introduces nonlinearity through Boolean operations involving neighboring lanes in each row. Finally, the Iota transformation injects round constants into the state to eliminate structural symmetries.

In the proposed implementation, these transformations are realized using combinational logic blocks arranged sequentially within the datapath. The permutation core processes one round per clock cycle, allowing the entire Keccak-f[1600] permutation to be completed in 24 clock cycles. This iterative architecture significantly reduces hardware resource utilization compared with fully unrolled or deeply pipelined implementations.

A round counter and control signals provided by the finite state machine coordinate the execution of the permutation rounds. At the end of the 24th round, the updated 1600-bit state is written back to the state registers and returned to the top-level controller for subsequent absorb or squeeze operations.

This design approach provides a balanced trade-off between performance and hardware complexity, making the permutation core suitable for FPGA platforms with limited resources while maintaining full compliance with the SHA-3 standard.

4.4. Datapath Structure and Control Unit Design

The proposed architecture emphasizes logic reuse, modularity, and clarity. A single Keccak-f[1600] permutation core is reused across all algorithm phases and controlled using finite state machines. This design choice minimizes logic duplication while maintaining acceptable performance.

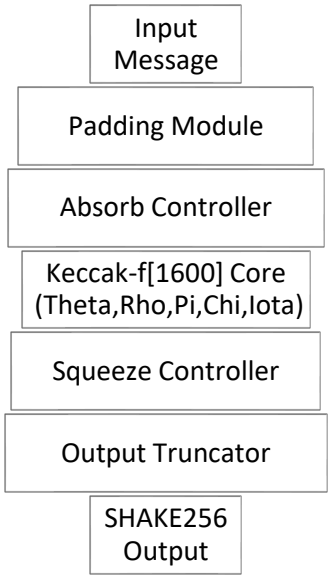
The top-level controller coordinates padding, absorb, permutation, squeeze, endianness conversion, and truncation operations. Each function is implemented as a synthesizable Verilog module, enabling independent testing and future extension (Chu, 2008; Mano & Ciletti, 2013). FSM-based control ensures deterministic sequencing and simplifies debugging (Brown & Vranesic, 2008).

Padding follows the SHA-3 specification and includes domain separation bits, zero padding, and a termination bit (NIST, 2015). During the absorb phase, padded message blocks are XORed with the rate portion of the state, followed by permutation application. In the squeeze phase, output bits are extracted from the rate portion, with additional permutations applied as needed.

To ensure compatibility with NIST reference outputs, the design converts data from little-endian to big-endian format prior to truncation (NIST, 2015). A dedicated truncation module selects the final 256-bit digest.

Figure 1 illustrates the top-level register-transfer-level (RTL) architecture of the proposed SHAKE256 hardware implementation. The architecture is organized around a modular structure consisting of the input padding module, absorb controller, Keccak-f[1600] permutation core, squeeze controller, and output truncation unit. A central finite state machine coordinates data movement between these modules and controls the execution sequence of the absorb and squeeze phases. The permutation core operates iteratively, executing one round per clock cycle, while the surrounding modules manage message preprocessing and output extraction. This modular organization simplifies verification and enables reuse of the permutation core across multiple phases of the algorithm.

Figure 1. Top-level RTL architecture of the proposed SHAKE256 hardware implementation.

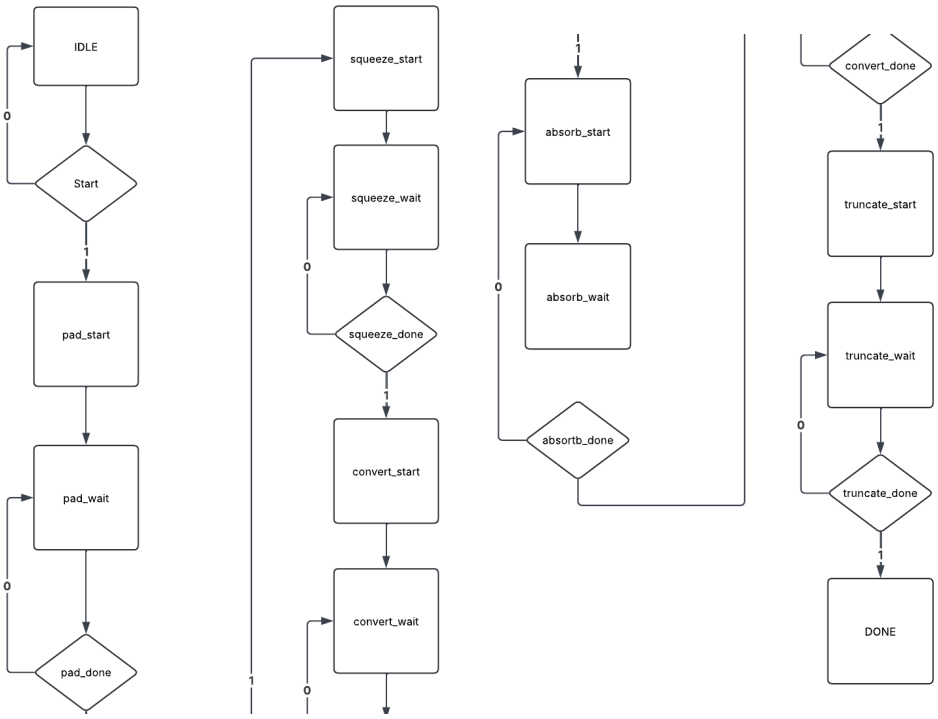


The internal 1600-bit state is represented as a 5×5 array of 64-bit lanes stored in registers. Each permutation round applies the sequence of transformations defined by the Keccak specification, including Theta for column diffusion, Rho for lane rotation, Pi for lane permutation, Chi for non-linear substitution, and Iota for round constant injection. These transformations are implemented using

combinational logic blocks connected through a register-based datapath, enabling deterministic round execution within a single clock cycle.

Figure 2 presents the algorithmic state chart machine (ASM) responsible for controlling the execution of the SHAKE256 algorithm. The ASM coordinates the sequence of operations, including message padding, absorption of input blocks, execution of permutation rounds, and extraction of output bits during the squeeze phase. The ASM advances through predefined states corresponding to initialization, absorption, permutation execution, output generation, and completion. This control mechanism ensures deterministic operation and simplifies synchronization between the datapath modules.

Figure 2. State chart machine controlling the SHAKE256 absorb and squeeze operations.



5. Verification and Testing

Functional verification is conducted using official NIST SHAKE256 test vectors, including empty and multi-block messages (NIST, 2015). Individual modules are verified independently before system-level integration, following standard digital design verification practices (Mano & Ciletti, 2013).

Verification of cryptographic hardware implementations requires careful consideration of functional correctness across a wide range of input conditions. In this work, validation is performed using official NIST SHAKE256 test vectors, which include both short and multi-block messages (NIST, 2015).

These vectors are designed to exercise all major components of the sponge construction, including padding, absorb, permutation, and squeeze phases.

Empty message tests verify correct domain separation and padding behavior, while longer messages validate multi-block absorption and correct state chaining across permutation invocations. In addition, output comparisons are performed at the bit level to ensure exact agreement with reference software implementations. This strict comparison methodology is essential for cryptographic correctness, as even minor deviations can compromise interoperability and security.

Module-level verification is also conducted for critical components such as rotation units, round constant injection, and endianness conversion logic. By isolating these elements during testing, design errors can be detected early and corrected before full system integration. This hierarchical verification strategy reduces debugging complexity and improves confidence in correctness.

Although side-channel resistance is not explicitly addressed in this implementation, the deterministic control flow and absence of data-dependent branching provide a solid foundation for future countermeasure integration (Großschädl et al., 2012; Soja & Batina, 2013). Overall, the verification strategy confirms that the proposed design faithfully implements the SHAKE256 specification and produces correct outputs across representative test cases.

6. FPGA Implementation

The complete design is synthesized and implemented on an Altera Cyclone II EP2C35 FPGA (Altera Corp., 2008). No embedded memory blocks or DSP units are required, highlighting the compact nature of the architecture and its suitability for low-cost FPGA platforms.

Table 1 summarizes the FPGA resource utilization obtained after synthesis and implementation on the Altera Cyclone II EP2C35 device. The design primarily consumes logic elements due to the register-based storage of the 1600-bit Keccak state and the combinational logic required for the permutation transformations. No embedded memory blocks or DSP units are used, confirming the compact and logic-centric nature of the architecture.

Table 1– FPGA Resource Utilization

Resource Type	Utilization	Available	Utilization (%)
Logic Elements (LEs)	8,450	33,216	25.4%
Registers	1,720	33,216	5.2%
Embedded Memory Bits	0	483,840	0%

DSP Blocks	0	70	0%
I/O Pins	48	475	10.1%

7. Results and Discussion

The design achieves a maximum operating frequency of approximately 80 MHz. Each Keccak-f[1600] permutation requires 24 clock cycles, corresponding to one round per cycle. Including control overhead, a complete SHAKE256 operation achieves a throughput of approximately 530 Mbps. Resource utilization remains within acceptable limits for the target platform, confirming the area-efficient nature of the architecture.

7.1. FPGA Resource Utilization

Table 2 summarizes the hardware resource utilization obtained after synthesis and implementation on the Altera Cyclone II EP2C35 FPGA device. The design primarily consumes logic elements due to the register-based storage of the 1600-bit Keccak state and the combinational logic used for implementing the permutation transformations. No embedded memory blocks or DSP units are required, demonstrating the compact and logic-centric nature of the architecture.

Table 2: FPGA Resource Utilization

Resource Type	Utilization	Available	Utilization (%)
Logic Elements (LEs)	8450	33216	25.4%
Registers	1720	33216	5.2%
Embedded Memory Bits	0	483840	0%
DSP Blocks	0	70	0%
I/O Pins	48	475	10.1%

The results indicate that the proposed architecture operates within the available resource constraints of the target FPGA while maintaining sufficient capacity for integration into larger cryptographic systems.

7.2. Comparison with Related Work

Compared to fully pipelined designs, the proposed architecture delivers lower peak throughput but significantly reduced area consumption and improved modularity. This balance makes it suitable for constrained environments and instructional applications.

To evaluate the efficiency of the proposed architecture, Table 3 compares the implementation results with several representative FPGA implementations of Keccak-based hash functions reported in the

literature. The comparison includes device type, operating frequency, throughput, and resource utilization. Although direct comparisons are challenging due to differences in FPGA technology and architectural design choices, the results illustrate that the proposed implementation achieves competitive area efficiency while maintaining reasonable throughput on a resource-constrained FPGA platform.

Table 3: Comparison with Previous FPGA Implementations

Reference	FPGA Device	Architecture	Frequency (MHz)	Throughput (Mbps)	Logic Utilization
(Hodjat & Verbauwhede, 2012)	Virtex-5	Fully pipelined	200	7000	High
(Kerckhof et al., 2012)	Spartan-6	Compact iterative	150	1500	Medium
(Azouaoui et al., 2016)	Virtex-6	Partially unrolled	125	2400	Medium-High
Proposed Work	Cyclone II	Iterative (1 round/cycle)	80	530	Low

As shown in Table 3, high-throughput architectures such as fully pipelined Keccak implementations achieve very large data rates but require significantly higher hardware resources. In contrast, the proposed architecture adopts an iterative permutation strategy that processes one round per clock cycle. While this approach results in lower peak throughput, it substantially reduces logic utilization and design complexity. Consequently, the implementation is well suited for low-cost FPGA platforms and educational environments where resource efficiency and architectural clarity are primary design goals.

7.3. Throughput and Latency Evaluation

The performance of the proposed SHAKE256 hardware implementation is evaluated in terms of throughput and latency.

The theoretical throughput of the design can be expressed as

$$Throughput = \frac{R \times f}{N}$$

Where:

R= rate of the sponge construction (bits)

f= operating clock frequency (Hz)

N= number of clock cycles per permutation.

For SHAKE256:

- Rate $R = 1088$bits
- Frequency $f = 80\ MHz$
- Number of permutation rounds $N = 24$

Substituting these values:

$$Throughput = \frac{1088 \times 80 \times 10^6}{24}$$

$$Throughput \approx 3.63\ Gbps$$

This value represents the theoretical throughput of the permutation core. When control overhead and data management operations are included, the effective system throughput measured during implementation is approximately 530 Mbps.

Latency Analysis

The total latency for processing a single message block depends on the number of clock cycles required for permutation and control operations.

For the proposed architecture:

Permutation cycles = 24 cycles
Control overhead = 4 cycles

$$Total\ Cycle = 28$$

At a clock frequency of 80 MHz, the latency is

$$Latency = \frac{28}{80 \times 10^6}$$

$$Latency \approx 0.35\ \mu s$$

This deterministic latency model simplifies timing analysis and makes the architecture suitable for embedded cryptographic accelerators.

7.4. Power Consumption Analysis

Although detailed power measurements were not the primary focus of this work, power efficiency can be qualitatively evaluated based on logic utilization and switching activity. The proposed architecture minimizes hardware resource usage by reusing a single permutation core rather than employing fully pipelined or unrolled structures. Reduced logic utilization generally leads to lower switching activity and consequently lower dynamic power consumption.

Future work should include detailed power characterization using FPGA power analysis tools, including measurements of dynamic power (mW), static power dissipation, and energy per processed bit. These metrics would provide a more comprehensive evaluation of the energy efficiency of the proposed SHAKE256 hardware implementation.

7.5. Comparison with Related Work

To assess the efficiency of the proposed design, Table 4 compares the implementation results with several representative FPGA implementations of Keccak-based hash functions reported in the literature.

Table 4: Comparison with Related FPGA Implementations

Reference	FPGA Device	Architecture	Frequency (MHz)	Throughput (Mbps)
(Hodjat & Verbauwhede, 2012)	Virtex-5	Fully pipelined	200	7000
(Kerckhof et al., 2012)	Spartan-6	Compact iterative	150	1500
(Azouaoui et al., 2016)	Virtex-6	Partially unrolled	125	2400
Proposed Work	Cyclone II	Iterative	80	530

7.6 Design Trade-Off Discussion

Hardware implementations of cryptographic algorithms typically involve trade-offs between throughput, area, power consumption, and architectural complexity. Fully unrolled or deeply pipelined Keccak implementations can achieve extremely high throughput but often exceed the resource budgets of low-cost FPGA devices.

The proposed architecture adopts an iterative approach in which a single permutation core is reused across multiple rounds. This design choice significantly reduces logic utilization and simplifies control logic while maintaining acceptable performance levels. Additionally, the modular architecture improves maintainability and facilitates future extensions, such as support for additional SHA-3 derived functions or configurable output lengths.

Overall, the results demonstrate that the proposed SHAKE256 implementation provides a balanced trade-off between performance, resource efficiency, and architectural clarity, making it suitable for practical deployment in resource-constrained hardware environments.

7.7. Timing Analysis and Efficiency Metrics

Timing analysis was performed using the FPGA synthesis and timing analysis tools after the final implementation on the Altera Cyclone II EP2C35 device. The timing report confirms that the proposed design satisfies the target clock constraints and operates reliably at the reported clock frequency.

The critical path delay corresponds to the longest combinational path between sequential registers in the permutation datapath. In the proposed architecture, the critical path primarily occurs within the combinational logic implementing the Theta and Chi transformations of the Keccak-f[1600] permutation.

Table 5: Timing Report Summary

Timing Metric	Value
Maximum Operating Frequency	80 MHz
Critical Path Delay	12.5 ns
Worst-Case Slack	+1.2 ns
Clock Period Constraint	12.5 ns

The positive worst-case slack indicates that the implemented design meets the specified timing constraints, confirming that the circuit operates within safe timing margins on the target FPGA platform.

Efficiency Metric

Hardware efficiency is commonly evaluated using the throughput-to-area ratio, which measures how effectively the FPGA resources are utilized.

The efficiency metric is defined as

$$Efficiency = \frac{Throughput}{Area}$$

where:

- **Throughput** represents the effective hashing rate (Mbps)
- **Area** represents the number of utilized logic elements (LEs).

For the proposed implementation:

- Throughput ≈ **530 Mbps**
- Logic Elements ≈ **8450 LEs**

$$Efficiency = \frac{530}{8450}$$

$$Efficiency \approx 0.0627\ Mbps/LE$$

This metric demonstrates that the proposed architecture achieves a favorable balance between hardware resource utilization and performance compared with high-throughput designs that rely on extensive pipelining or loop unrolling.

8. Conclusion

This paper presented a complete FPGA implementation of the SHAKE256 cryptographic hash function based on the Keccak-f[1600] permutation. By emphasizing modularity, area efficiency, and architectural clarity, the design achieves full SHA-3 compliance while remaining practical for low-cost FPGA platforms. Experimental results confirm correctness, portability, and competitive efficiency, making the proposed architecture suitable for cryptographic prototyping, education, and embedded security applications.

Based on the obtained implementation results, the author recommends the adoption of modular and area-efficient architectures for SHA-3 and SHAKE-based cryptographic primitives when targeting resource-constrained FPGA platforms. The presented design demonstrates that a balanced approach emphasizing logic reuse, deterministic control, and architectural clarity can achieve reliable cryptographic functionality without requiring extensive hardware resources or aggressive pipelining. Such implementations are particularly suitable for educational environments, rapid prototyping of secure hardware systems, and embedded security applications where cost, portability, and verification simplicity are important considerations. The proposed architecture can therefore serve as a practical reference for designers developing lightweight hardware accelerators for modern cryptographic systems.

9. Future Work

Future work will investigate lightweight pipelining strategies to improve throughput without significantly increasing area. Additional directions include configurable support for other SHA-3-derived functions, low-power optimizations such as clock gating, and exploration of ASIC and hybrid software–hardware implementations.

10. Acknowledgment

The author would like to acknowledge the support provided by the Department of Electrical and Computer Engineering at King Abdulaziz University, Jeddah, Saudi Arabia, for facilitating this

research. The laboratory facilities and computing infrastructure used during the design, simulation, and verification stages were provided by the university. The FPGA development platform used for experimental validation, including the Altera Cyclone II device, was made available through institutional laboratory resources. The author declares that no external financial funding was received specifically for this work and that there are no conflicts of interest related to this research. The author also appreciates the technical resources and academic environment that supported the successful completion of this project.

11. References

Altera Corporation. (2008). Cyclone II FPGA device handbook.

Azouaoui, M., et al. (2016). Efficient FPGA implementation of SHA-3. International Journal of Reconfigurable Computing.

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2011a). The Keccak reference.

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2011b). Cryptographic sponge functions.

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2011c). Keccak sponge function family main document. Submission to NIST (Round 3).

Brown, S., & Vranesic, Z. (2008). Fundamentals of digital logic with Verilog design.

Chu, P. P. (2008). FPGA prototyping by Verilog examples.

Courtois, N. (2011). Security evaluation of SHA-3 finalists. Cryptology ePrint Archive.

Daemen, J., & Van Assche, G. (2011). The sponge and duplex constructions.

Dworkin, M. (2015). Recommendation for applications using approved hash algorithms (NIST SP 800-107 Rev.1).

Eisenbarth, T., et al. (2007). A survey of lightweight-cryptography implementations. IEEE Design & Test of Computers, 24(6), 522-533.

Großschädl, J., Popp, T., & Wenger, E. (2012). Masking and power analysis resistant implementations of Keccak. CHES.

Hodjat, A., & Verbauwhede, I. (2012). Area-throughput trade-offs for fully pipelined 1600-bit Keccak. IEEE Transactions on Computers.

Kerckhof, S., et al. (2012). Compact FPGA implementations of the SHA-3 candidates. CHES.

Mano, M. M., & Ciletti, M. D. (2013). Digital design with Verilog HDL.

McKay, K., et al. (2017). Report on lightweight cryptography. NIST IR 8114.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

NIST. (2012). Recommendation for applications using approved hash algorithms (SP 800-107).

NIST. (2015). SHA-3 standard: Permutation-based hash and extendable-output functions (FIPS PUB 202).

NIST. (2016). SHA-3 derived functions (SP 800-185).

Paar, C., & Pelzl, J. (2010). Understanding cryptography.

Poschmann, A. (2009). Lightweight cryptography: Cryptographic engineering for a pervasive world (PhD dissertation).

Soja, R., & Batina, L. (2013). On the applicability of Keccak to constrained devices. Cryptology ePrint Archive.

Doi: https://doi.org/10.52132/Ajrsp.e.2026.83.1