Advances in Nonlinear Variational Inequalities
ISSN: 1092-910X
Vol 28 No. 6s (2025)

# Analysis of FPGA-Based Secure Architecture of Hash Functions for Blockchain Algorithms

**Ms.Preeti R Lawhale,[1] Dr.Sujata N.kale,[2]**

PhD Scholar,PG Department of Applied Electronics,SGBAU,Amravati

Professor, PG Department of Applied Electronics,SGBAU,Amravati

***Article History:***

**Received:** 5-01-2025

**Revised:** 19-02-2025

**Accepted:** 27-02-2025

**Abstract:**

In recent years, blockchain technology has arisen as a safe and decentralized framework for various applications, including cryptocurrencies, digital contracts, and supply chain management. At the heart of blockchain's security lies cryptographic hash functions, such as SHA-256, which preserve immutability and data integrity. However, implementing hash functions in software can be inefficient for real-time applications due to high computational requirements. This paper proposes and analyzes an FPGA-based architecture for lightweight secure hash functions, specifically focusing on performance, scalability, and energy efficiency. By leveraging the parallelism and reconfigurability of FPGAs, the proposed architecture significantly enhances the throughput of hash computations while ensuring robustness and security, making it an ideal solution for blockchain algorithms. Bit permutation, linear transformation, and S-Box functionality are used in the suggested design. Performance evaluation is conducted through the implementation of the SHA-256 algorithm, with comparisons to traditional software-based methods. For an appropriate FPGA-based application deployment, a comparison of suggested designs is detailed on various FPGA families. The results demonstrate substantial improvements in terms of power efficiency and security in blockchain environments.

**Keywords:** FPGA, Lightweight Cryptography, Blockchain, Hash Function, Power Consumption, SHA-256, Hardware Acceleration, Secure Architecture.

## 1. Introduction

Blockchain technology has transformed the way secure and decentralized systems operate by introducing distributed ledger technology, enabling trustless transactions without the need for intermediaries. One of the foundational elements of blockchain technology is cryptographic hashing, which ensures data integrity, immutability, and security across the distributed network. The reliability and success of blockchain protocols like Bitcoin and Ethereum heavily depend on cryptographic hash functions, such as the widely used SHA-256 [1]. However, as blockchain networks scale, the computational requirements for cryptographic hashing significantly increase, demanding more efficient, scalable, and power-optimized solutions. This is particularly critical in environments where large volumes of transactions are processed in real-time, such as in cryptocurrency mining and smart contract verification [2-3]. Traditional

cryptographic hashing solutions implemented on CPUs and GPUs face challenges related to computational speed, power consumption, and scalability. The computational complexity of hash functions, especially in consensus mechanisms like Proof-of-Work (PoW), necessitates the repetitive computation of hash values at high frequencies. As blockchain networks expand, the energy consumption of such systems grows exponentially, making them inefficient and unsustainable for long-term use, particularly in energy-sensitive applications like blockchain mining [4][6-8].

To address these challenges, Field Programmable Gate Arrays (FPGAs) have emerged as a promising hardware acceleration platform for cryptographic hashing in blockchain systems. FPGAs offer parallel processing capabilities, reconfigurability, and energy efficiency, making them well-suited for high-performance cryptographic operations [10]. By implementing cryptographic hash functions, such as SHA-256, on FPGA-based architectures, blockchain systems can achieve improved performance, reduced latency, and significant energy savings compared to traditional software-based methods. In the context of blockchain security protocols, FPGAs are a promising alternative that can enhance the speed and efficiency of hash computations, reduce energy consumption, and offer scalable, secure solutions that can adapt to the evolving needs of blockchain technology. This makes FPGA-based cryptographic hashing an essential component of future blockchain infrastructure [12][15-18]. This paper presents an FPGA-based architecture for secure hash functions, focusing on performance, power consumption, and security. We aim to provide an in-depth analysis of the proposed architecture's effectiveness in enhancing blockchain algorithms, particularly in scenarios that demand high throughput and low power consumption. This paper's primary contributions are as follows:

- Design of FPGA-based compact parallel SHA-256 processor that utilizes minimal area resources.
- Optimized SHA-256 processor architecture relative to existing methodologies.
- Development of a lightweight cryptographic hash function that ensures a secure and efficient hashing process.
- Develop a lightweight framework that accommodates IoT applications, and blockchain applications.

The remainder of the paper is organized as follows. In section 2, the related work on other conventional and lightweight hash-based cryptographic algorithms is discussed. Section 3 presents the background and motivation related to the research topic, and the proposed processor's hardware design is outlined in Section 4. In section 5, experimental result analysis is discussed. This work is brought to a conclusion in section 6.

## 2. Objectives

The advancement and execution of reliable and secure hashing algorithms in hardware, especially in FPGA, have garnered considerable focus in recent years due to their essential function in improving performance and security. Numerous studies have underscored the progress and obstacles in this domain, offering a thorough assessment of the present research

landscape and prospective trajectories. Recent survey studies from prestigious publications have thoroughly examined the implementation methodologies, and performance metrics of cryptographic techniques and optimization methods on reconfigurable hardware systems.

FPGA implementation of the SHA-256 algorithm, the predominant blockchain hash function to enhance processing capacity and energy efficiency in IoT devices, addressing security and privacy concerns. The various consensus mechanisms in blockchain, including PoS (Proof of Stake), PoA (Proof of Authority), and PoW (Proof of Work) have been examined and contrasted utilizing VHDL. A lightweight cryptographic core developed on a System-on-Chip (SoC) that incorporates four authentication protocols, two encryption protocols, and a key generation/exchange mechanism for ultra-low-cost devices. An iterative design is employed to enhance the area-performance trade-offs of the PHOTON hash function. Advanced lightweight cryptographic hash methods examined [5] for severely restricted systems, classify design trends, evaluate cryptographic characteristics, and scrutinize cryptanalytic vulnerabilities. More than 20 Round 2 candidates in the NIST Lightweight Cryptography initiative have been realized in hardware by teams across [6]. Lightweight cryptographic designs and optimized hardware including 32-bit datapaths, including the SIMON 64/128, and LED 64/128 algorithms, are introduced for restricted devices [7]. Investigation of hardware and software for the implementation of the SHA256 algorithm as proposed in reference [8]. A novel SHA-256 hardware design has been presented that employs binary tree-structured adder trees to enhance hash computation performance. The suggested AES-SHA-3 implementation on the Xilinx Virtex FPGA series achieves optimal hardware efficiency, measured as Throughput per Slice (TPS) [10].

A multimode SHA-2 accelerator hardware design is suggested, exhibiting flexibility and exceptional performance at the SoC level [11]. A unique methodology is presented, grounded on a hybrid hardware/software architecture, specifically engineered for the PoW consensus, the predominant consensus mechanism in blockchain technology [12]. A new compact PHOTON-Beetle implementation is presented, in which the fundamental matrix multiplication is performed in a serialized manner to minimize hardware footprint [13]. The unfolding approach is utilized in the hash function by generating the hash value output through alterations to the SHA-256 architecture [14]. A serialized design is described that balances sequential and combinational logic while utilizing fewer flip-flops [15]. A method has been presented to utilize FPGA for the implementation of variational irreducible polynomials by a hashing algorithm [16]. The high-accuracy short read alignment system is created using a read mapping technique based on seed-and-extend hash tables [17]. The prevalent lightweight hash algorithms on FPGA platforms to ascertain the most appropriate for blockchain-IoT devices [18]. The significance of hash functions and the comparative analysis of various cryptographic methods utilizing blockchain technology [19]. A low-power, high-speed hardware design for the Scrypt function is suggested to produce blocks for the Scrypt-based blockchain network [20].

Specialized focus is directed into FPGA optimization methodologies for the three hash standards [21]. ECIES (Elliptic Curve Integrated Encryption Scheme) protocol based on

hardware implementations is delineated utilizing a secure SoC [22]. The FPGA implementation of five innovative non-cryptographic hash functions has been presented, which are appropriate for networking and security applications necessitating rapid search and/or counting structures [23]. The authenticated encryption algorithms that advanced to the second stage of the NIST lightweight cryptography standardization process are discussed [24]. The implementation of lightweight blockchain technology primarily as a means of protecting IoT systems has been examined [25]. A suggested pipeline approach for the implementation of hardware and temporal redundancies, assessing hardware resources and performance to optimize the critical route [26]. Hamming Bird 2 approach for generating a hash utilizing a chaotic key through logistic maps and lightweight cryptography is proposed [27]. During the partial product accumulation phase, the construction of the 4-2 adder compressor in SHA-256 enhances computational efficiency by reducing the number of adders [28]. A novel design for SHA-3 has been presented, intending to enhance the function's performance using pipelining. Four configurations of pipelining are suggested: two, three, four, and six phases [29].

A private blockchain system for Industrial IoTs utilizing FPGA technology is presented, wherein transaction creation occurs within the FPGA in a secure and isolated environment [30]. The finetuning pipeline and parallel processing utilizing the Split technique have been suggested [31]. A novel design is suggested that aims to enhance the throughput rate of the Keccak hash algorithm by demonstrating efficient outputs [32]. A novel methodology for SHA-512 in a localized blockchain application has been devised [33]. A comparative investigation of the hardware implementation of security algorithms on several FPGAs has also been conducted [34]. The two unique hybrid hash functions, SHAES-2 and SHAES-3, are constructed using the SHA-2 and SHA-3 hash functions, respectively, in conjunction with the masked Advanced Encryption Standard (AES) algorithm [35]. A lightweight cryptographic hash function designed by utilizing linear transformation, S-Box, and bit permutation techniques [36]. The amalgamation of RSA and SHA enables the construction of a block on an FPGA, which, when combined with further blocks, constitutes an encrypted Blockchain [37].

Various hardware acceleration techniques for cryptographic hash functions have been proposed in the literature. Most efforts have focused on conventional and general optimization of SHA-256 for secure communication protocols. However, little research has explored the application of FPGA-based architectures specifically tailored for blockchain, where hash rates are exponentially higher than in conventional encryption systems. Several works have explored FPGA implementations of SHA-256, focusing on improving processing speed and optimizing resource utilization. However, few have thoroughly examined the trade-offs between performance and energy efficiency in blockchain environments, where power consumption is equally as important as computational speed.

## BACKGROUND AND MOTIVATION

### Blockchain Overview

It is a distributed ledger technology (DLT). Although used in finance, it has other purposes. Blockchain is known for its data storage and validation through a decentralized chain of blocks.

Verification chains ensure that the blockchain network recognizes data changes. This method uses asymmetric, symmetric, hash functions, and Merkle trees.

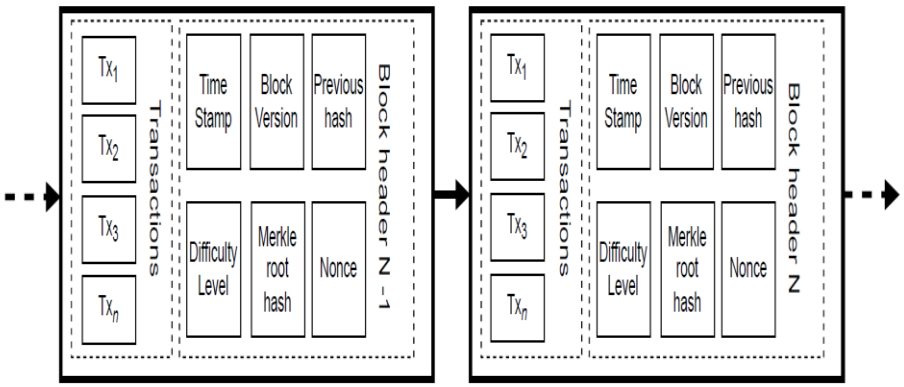


**Figure 1:** Diagrammatic representation of a blockchain system [1]

Generic blockchain blocks are shown in Figure 1. One area is for transactions and the other is for the header. Based on the blockchain technology, each block can have dozens to hundreds of transactions, marked Tx1 to Txn in Figure 1. It describes the block and its chain history in the header. Following are its domains: The block hash preserves the chain sequence, while the block version specifies a validation framework, the Merkle tree hash includes the root of the tree for transactions, the difficulty level indicates mining effort, and the nonce is a random value determined by miners to validate the consensus algorithm. Blockchain blocks are mined to validate transactions and header structure. Mined blocks must include transactions or data. Blockchain peer-to-peer nodes execute these transactions. To reduce blockchain technology's storage costs, the block's header contains a Merkle tree that summarizes all transactions.

Secure Hash Function SHA– 256

It is a widely used cryptographic hash function, essential in securing data and ensuring integrity in many applications, including blockchain, digital signatures, and data authentication. It generates a fixed-length 256-bit hash from variable-length input data. SHA-256 is computationally intensive, which poses challenges in software-based implementations, especially for real-time applications like blockchain mining. FPGA-based hardware acceleration offers a solution to address the high performance and efficiency demands of SHA-256.

SHA-256 operates by processing data in blocks of 512 bits. The main steps in the algorithm include:

Message Padding: The input message is padded so that its length is a multiple of 512 bits.

Message Parsing: The padded message is divided into 512-bit blocks.

Hash Computation: Each block is processed through the compression function for 64 rounds, using a set of constants and bitwise operations, producing a 256-bit hash digest.

Output: The final digest serves as the cryptographic hash of the input data.

**Motivation**

Blockchain relies on cryptographic hash functions to maintain the integrity and immutability of data blocks. Hash functions such as SHA-3, SHA-256, and MD5 compress input data into fixed-size digests. For blockchain security, hash functions need to be fast, secure, and energy-efficient. However, software-based implementations can suffer from significant limitations, including:

High Computational Overhead: Performing millions of hash operations per second in blockchain environments can strain CPU resources, leading to delays and increased power consumption.

Latency and Bottlenecks: The demand for high throughput in mining applications results in latency when hash functions are computed sequentially by general-purpose processors.

Energy Efficiency: In distributed networks, especially for energy-conscious environments like blockchain mining, power-efficient hashing is critical. FPGAs offer a compelling alternative by significantly reducing energy consumption.

The potential of FPGAs for hardware acceleration provides an opportunity to mitigate these issues. Their ability to handle massive parallel processing, coupled with reconfigurability, makes them ideal candidates for implementing secure and efficient hash functions in blockchain algorithms.

## 3. MATERIAL AND METHODS

**SHA-256 with Parallel Architecture**:

The SHA-256 hash architecture has been selected for implementation in the standard hash function framework. This architecture, comprising both the preprocessing block and the hash computation block, is executed utilizing a parallel design, as seen in Figure 1. This enables the construction of a rapid and efficient SHA-256. These layouts were developed using VHDL to enhance throughput performance.

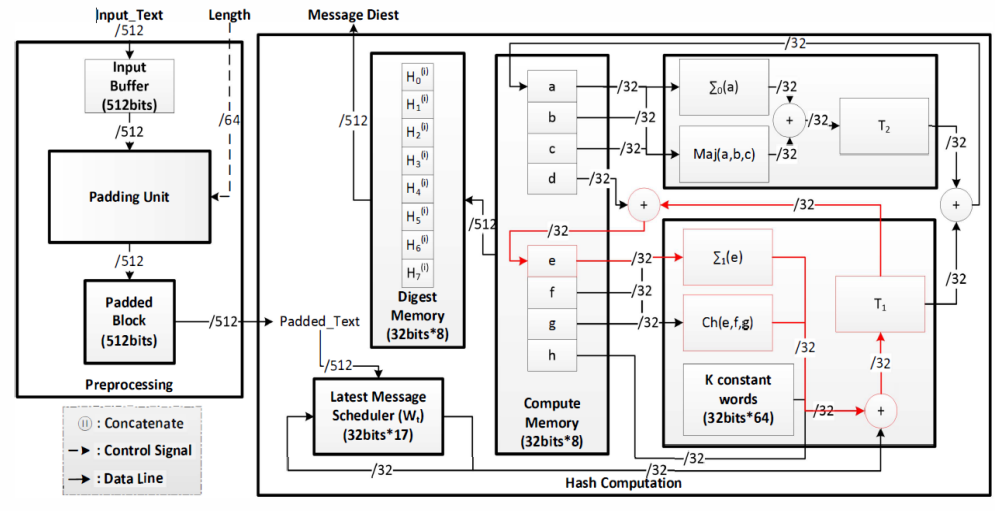


**Figure 2**: Conventional parallel SHA-256 architecture

The architecture of SHA-256, when executed with both preprocessing and hash calculation blocks in a parallel configuration, seeks to enhance performance through the utilization of parallelism. SHA-256 is a cryptographic hash algorithm that generates a 256-bit output (hash) from an input message. The function encompasses many activities, including padding, message scheduling, and a series of logical processes executed across 64 iterations. A parallel design increases throughput by executing many processes concurrently.

The proposed architecture attains increased throughput by using pipelining in the new message preparation block and the hash computation block. During the preliminary processing phase, the 512-bit padded blocks of input data are deconstructed and examined. The last 64-bit block signifies the termination of the message and comprises one '1' bit, a sequence of '0' bits, and the length of the message. Subsequently, Wt message scheduling blocks are generated by the block scheduler. Eight working variables and two temporary variables (T1 and T2) in the Compute Memory block are derived from the Wt. Wt from 0 to 15 correspond to the padded blocks stored in separate 16-bit registers, but Wt from 16 to 63 necessitate further calculations using the preceding Wt within a single 32-bit register. The current message scheduling module can substantially decrease the number of necessary registers to compute Wt between 16 and 63. During hash computation, it retains the current padded message and readies the appropriate transmission at the designated moment. Ultimately, in the 63rd round, eight operational variables are included into the hash values from preceding rounds to generate the message digest of the input message. The red line in Figure 1 denotes the essential pathway for the hash computation block. The computation of the variable is contingent upon the data associated with the route. Multiple 32-bit addition operations may impede the route's velocity. We employ adder and subtractor IP logics to reduce latency on this critical channel and enhance overall performance.

**Lightweight Hash Function**

A lightweight hash processor utilizing the SHA-256 algorithm is engineered to attain cryptographic security while reducing resource use, power consumption, and physical space in hardware implementations. Lightweight designs are essential for resource-limited situations. Figure 2 illustrates a comprehensive overview of the design methodology and architecture for a lightweight SHA-256 hash processor.

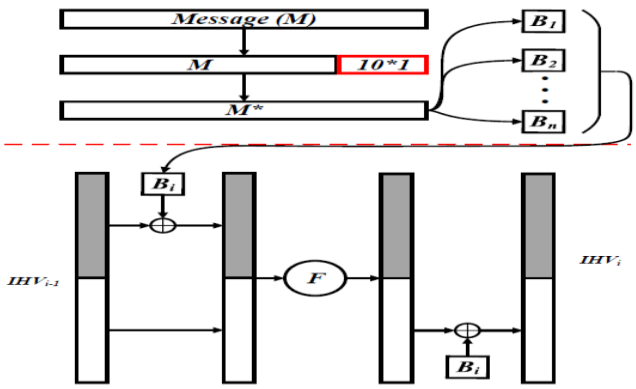


**Figure 3**: The architecture of a lightweight hash processor

A lightweight hashing technique designed for IoT-sensitive devices. The primary configuration of the proposed design is seen in Figure 2. The padding technique is employed to augment the size of an input message (M) with a variable length, ensuring it exceeds 512. Subsequently, 512-bit blocks of uniform size, B1, B2,..., Bn, are generated from the padded message (M). The compression function F operates on the divided blocks in a bijective way, applying XOR between a block and a segment of the IHV both before and after the processing of the compression function. The IHV has a capacity of 1024 bytes and is initialized according to the specified output hash size. If a 256-bit output hash size is necessary, the initial two bytes of the IHV will be 0x0100, with the subsequent bytes designated as zeros. The five phases of the proposed architecture consist of padding, parsing, IHV setup, compression function calculations, and final hash formation. We will discuss each phase in the subsequent text.

**Message Bit Padding**

The padding method was employed in the proposed design. Algorithm 1 delineates the fundamental framework of this technique. The objective of padding a variable-length input message (M) is to augment its size beyond the block size (B). The residual bits are determined by dividing the message length by the block size, as demonstrated in the process, and this value is then required for padding. Subsequently, two binary 1s are appended to the input message to signify the padding boundaries, together with the requisite number of bits (P). Following padding, the input message is denoted as M, which is generated.

**Parsing of the Padded Message**

The padded message is divided into equal-sized blocks of 512 bits (B1, B2,..., Bn). A 512-bit block consists of eight 64-bit words; for instance, the message block Bi is denoted by the eight words B0i, B1i, B2i,..., B7i. The compression function processes each message block.

**Initialization of Hash Values**

During this step, the identical technique was employed in the JH-hash submission. The IHV is contingent upon the specified output hash length, with the initial two bytes of the IHV containing the size of the output hash, while the remaining portion of the IHV is initialized to zero. The suggested system accommodates multiple hash lengths: 160, 224, 256, 384, and 512.

**Compression Function Calculation**

The equation demonstrates the sequential processing of message blocks using the compression function. The five stages of the compression function are S-box application, bit grouping, permutation, linear transformation, and degrouping. During the grouping phase, an input message block Bi and the Initial Hash Value (IHV) are combined, allowing each input (message block and IHV) to contribute two bits to each S-Box. The S-Box step identifies the S-Box to be employed using the round constant (Rc). The bits are rearranged before undergoing permutation through the linear transformation layer. This method continues for each block until the conclusion of the eighteen rounds. Upon processing the current block, the output is transmitted as IHV for the calculation of the subsequent block. Each block in the proposed design undergoes eighteen cycles, the quantity of which is specified.

After the division of the input into two words during the grouping phase, it is transmitted to the S-box selection layer. The choice of an S-Box from the 4 × 4-bit S-Boxes is dictated by the round constant (Rc). An equal quantity of bits from the message block and the IHV is included in the input received by the S-Box. The linear transformation layer employs an XOR operation to sequentially convert two words into the received words. The words are subsequently permuted and positioned adjacent to other words in the ensuing round of block calculation when the Pd permutation is employed. During the degrouping phase, which occurs after the completion of the 18 rounds, all bits are returned to their original locations to produce the final hash. Upon processing the final block in our design, the outcome is 1024 bits. The minimum required bits are utilized in the hash generation procedure to produce the final hash. It illustrates the whole framework of the inner round function. It exhibits the permutations and traverses the linear transformation and S-Box layers for the input data (x0, x1,..., x15).

**Generation of the Final Hash**

To create the final 256-bit hash, the least significant bits from the output are considered. Below is an overview of such an architecture and its working flow block diagram as shown in Figure 3:

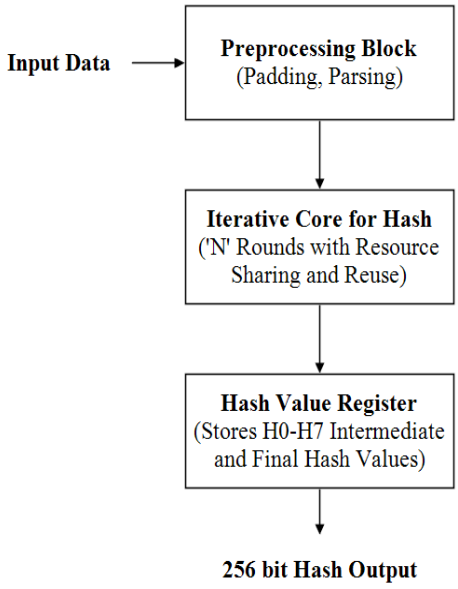


**Figure 4**: Workflow of Lightweight Hash Processor

The description of the main components in the workflow of lightweight hash processor design:

**1.Preprocessing Block**:

This block handles message padding and parsing. It prepares the input message into 512-bit blocks as per the SHA-256 specification.

**2.Iterative Hash Core**:

**Message Schedule Unit**: Generates the message schedule (W[0-63]) for each 512-bit block.

**Round Computation Unit**: Performs the 64 rounds of SHA-256 using shared resources like adders, XOR gates, and shift registers. The core handles functions such as Ch, Maj, Σ, and σ through a minimal set of arithmetic and logical operations.

The iterative nature allows a single computation unit to be reused across multiple rounds, significantly saving hardware resources.

**3.Hash Value Register**:

Stores the intermediate and final hash values. The initial hash values (H[0] to H[7]) are loaded, and after each block is processed, the register is updated with the computed hash.

**4.Control Logic**:

Manages the sequence of operations, coordinating the message schedule, round computation, and hash value updates.

Here's a simple flowchart shown in Figure 4 outlining the working flow of an FPGA-based lightweight hash function:

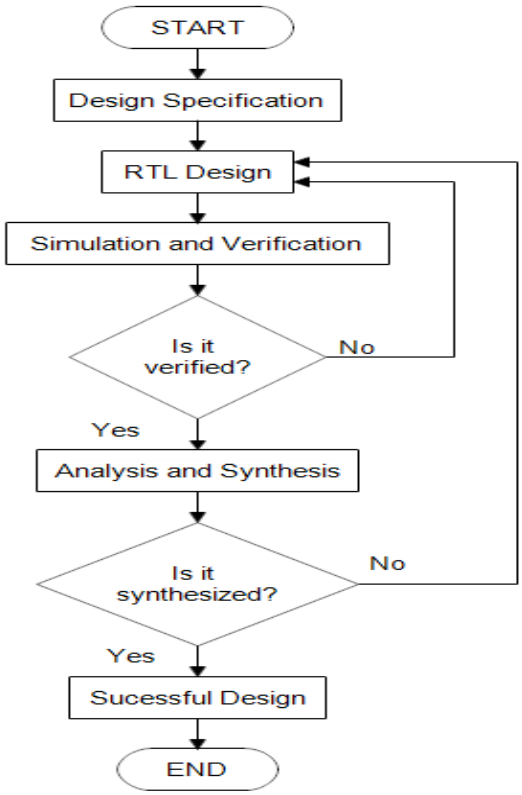


**Figure 5**: Working flowchart of FPGA based Implementation

This flowchart provides a fundamental outline for implementing a lightweight hash function on an FPGA, emphasizing great efficiency and minimal resource use. First, determine and articulate the design functionality requirements. The user must supply a Hardware Description Language (HDL) or a schematic design. The HDL format is more suited for handling massive structures, as it allows for numerical specification instead of requiring the drawing of each component. Subsequently, following the definition of the HDL, the logic is simulated and

validated with a simulation tool. After satisfactory verification, analysis and synthesis are conducted. Subsequently, an electrical design automation tool generates a technology-mapped netlist following a successful synthesis procedure. The netlist can then be adapted to the real FPGA design using a procedure known as place-and-route, often executed by the FPGA manufacturer's proprietary place-and-route software. The user will authenticate the map, location, and route outcomes using time analysis, simulation, and further verification techniques. Upon completion of the design and validation process, the resulting binary file is utilized to reconfigure the FPGA. The predominant hardware description languages are VHDL and Verilog. Despite the similarities between these two languages, VHDL is chosen for programming due to its widespread usage.

| Algorithms 1: Lightweight Hash Processor |
|---|
| **Input:**<br>Padded Message M = {M0, M1 . . . , Mn } Blocks |
| **Output:**<br>Output Hash (256 bits) |
| **Procedure:**<br>Step1: Initialize hash values: h0, h1, h2, h3, h4, h5, h6, h7<br>Step2: Initialize array of round constants: k0…. k63<br>Step3: Pre-processing: Data Padding<br>Step4: Process the message in successive 512-bit chunks:<br>break message into 512-bit chunks<br>for each chunk<br>Generates the message schedule (W[0-63]) for each 512-bit block.<br>Performs the 64 rounds of SHA-256 using shared resources like adders, XOR gates, and shift registers. The core handles functions such as Ch, Maj, Σ, and σ through a minimal set of arithmetic and logical operations.<br>The iterative nature allows a single computation unit to be reused across multiple rounds, significantly saving hardware resources.<br>Step5: Stores the intermediate and final hash values. The initial hash values (H[0] to H[7]) are loaded, and after each block is processed, the register is updated with the computed hash:<br>digest:= hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7 |

**Lightweight Hash Functions for Blockchain Applications**

The design and analysis of compact parallel and lightweight hash functions for blockchain algorithms are crucial for enabling secure, efficient, and scalable blockchain implementations,

especially in resource-constrained environments. By focusing on reduced complexity, compact structures, and optimized performance, lightweight hash functions provide a promising solution for next-generation blockchain applications. The continuous evolution of cryptographic research and advances in hardware design will pave the way for more robust and energy-efficient blockchain systems that can be deployed across a wide range of industries and devices.

The creation of block headers in a blockchain using a lightweight hash function is a crucial step to ensure both security and efficiency, especially in resource-constrained environments like IoT networks. The block header contains essential metadata about a block, which helps in linking it with previous blocks and ensuring data integrity. Here's an algorithm 2 that outlines the process:

Algorithm 2: Creation of Block Headers in Blockchain Using Lightweight Hash

**Inputs:**

- Block Data (Transactions, Timestamp, etc.)
- Previous Block Hash
- Lightweight Hash Function (LHF)

**Outputs:**

- Block Header
- Current Block Hash

**Procedure:**

Step1: Initialize Block Components:

- Collect the block data (e.g., transactions, timestamp, nonce).
- Retrieve the hash of the previous block, Prev_Block_Hash

Step2: Concatenate Header Elements:

- Create a header string that includes:

Header = Prev_Block_Hash || Timestamp || Nonce || Merkle_Root
Where:

- Prev_Block_Hash: Hash of the previous block.
- Timestamp: The time when the block is created.
- Nonce: A number that is adjusted to achieve the target hash.
- Merkle_Root: The root of the Merkle tree, created from the transactions within the block.

Step3: Apply Lightweight Hash Function:

- Compute the block hash using the lightweight hash function LHF

Block_Hash = LHF(Header)

Step4: Validate the Hash (Proof of Work - Optional):

- In case of Proof-of-Work (PoW), ensure that the computed hash meets the difficulty criteria (e.g., a certain number of leading zeros):

Step5: Store Block Header:

- Once a valid hash is found, store the block header components: Block_Header = (Prev_Block_Hash, Timestamp, Nonce, Merkle_Root)

Step6: Link the Block in the Chain:

- The current block's hash Block_Hash will serve as the Prev_Block_Hash for the next block.

Step7: Output the Block Header and Block Hash:

- The output contains the block header metadata and the hash for linking to the next block.

## 4. EXPERIMENTAL RESULTS AND DISCUSSION

### Experimental Setup

The proposed parallel design-based SHA-256 and lightweight hash processor design was created in VHDL and simulated in ModelSim Software. Altera Quartus II was used for all of the design analysis, synthesis, placement, and routing. Quartus II is used to port the suggested VHDL design to an Altera FPGA board. The suggested architecture's data flow is analyzed via a simulation of its VHDL design using IDE tools. At the end of each loop, we compare all of the variables to an emulated standard SHA-256 hash algorithm in MATLAB. After synthesizing, tables display a summary of the proposed architecture's device utilization and a comparison of the suggested method's performance to that of existing designs are presented.

### Evaluation Parameters

The FPGA implementation of SHA standards necessitates memory and speed due to the large number of computations kept during hash synthesis. In order to get the most out of an FPGA, it's essential to optimize the architecture in terms of speed and memory. The following metrics must be considered while evaluating FPGA performance:

*1.* Logic Resource Utilization: Indicative of the overall amount of lookup tables or configurable logic blocks employed in a given design. However, because to variations in placements methods between FPGA manufacturers, there may be situations in which a direct comparison of area is misleading. Therefore, using an FPGA from the same vendor is advised for this purpose.

*2.* Minimum Propagation Delay: A signal's latency is the time it takes to go from its source to its intended destination signal.

*3.* Maximum Frequency: The maximum possible operating clock rate for a certain FPGA.

*4.* Power Consumption: It is indicative of the overall hardware power consumption when a certain design is implemented. This characteristic is typically characterized by its frequency.

*5.* Throughput: Hardware implementation speed is measured in terms of the number of bits processed in a particular amount of time. The formula for throughput:

$$\text{Throughput} = (\text{Block\_Size}) / (T * \text{Nclk})$$

*6.* Efficiency: Ration of throughput to several slices. It is defined by the below equation.

$$\text{Efficiency} = \text{Throughput} / \text{Number of Slices}$$

**Result Simulation**

The simulation and analysis of a conventional hash function processor involve evaluating its performance in terms of above key parameters. The implementation of SHA256 is coded by using finite-state machine logic as shown in figure 5. A mathematical model of computing called the finite state machine can only be in one of a finite number of states at once. The implementation is performed in three main states, idle, busy, and final. In an idle state, input data and all other variables with hash values are initialized. In a busy state, all main computation of variables is performed. In the final state, the intermediate calculated hash is appended to give the final 256 bits hash value.

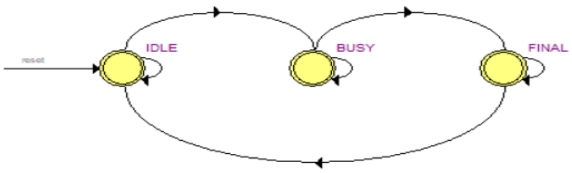


**Figure 6:** Implementation of SHA256 using FSM

The operation of compact parallel architecture-based SHA 256 is compiled and verified using Modelsim software. In Figure 6, the result of the timing simulation showing input data and output hashed message. Initially, at every positive edge of the clock with active low reset, when enable is high, input data is applied, and output hash value is generated after successive cycles. The various test data are tested to generate the output hash values. The output hash values generated are again verified with the hash value generated from standard SHA 256 in MATLAB software.

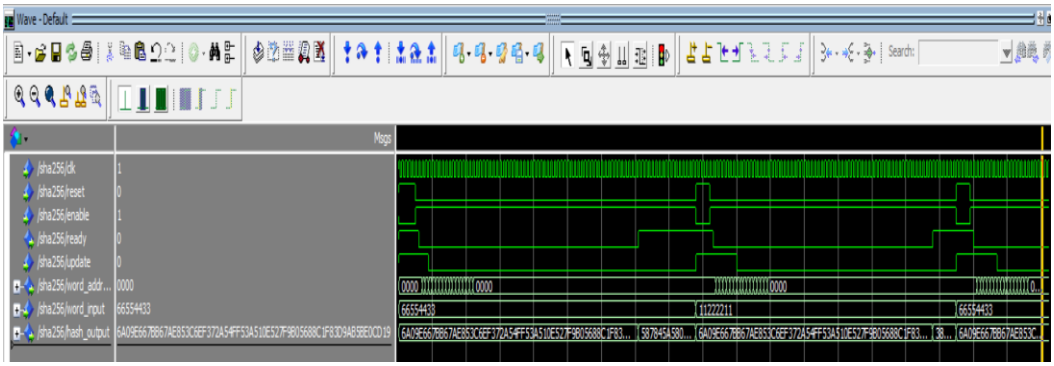

**Figure 7**: simulation result of SHA 256

The simulation and analysis of a lightweight hash function processor involves evaluating its performance with an emphasis on efficiency, particularly in resource-constrained environments such as IoT devices or embedded systems. The lightweight hash functions processor based on SHA-256 are implemented on FPGA (Field-Programmable Gate Array) platforms for efficiency and performance analysis.

Using Modelsim simulator tool, the lightweight hash's functionality is constructed and confirmed. Figures 7 present the input data and the hashed output message as a consequence of the time simulation.

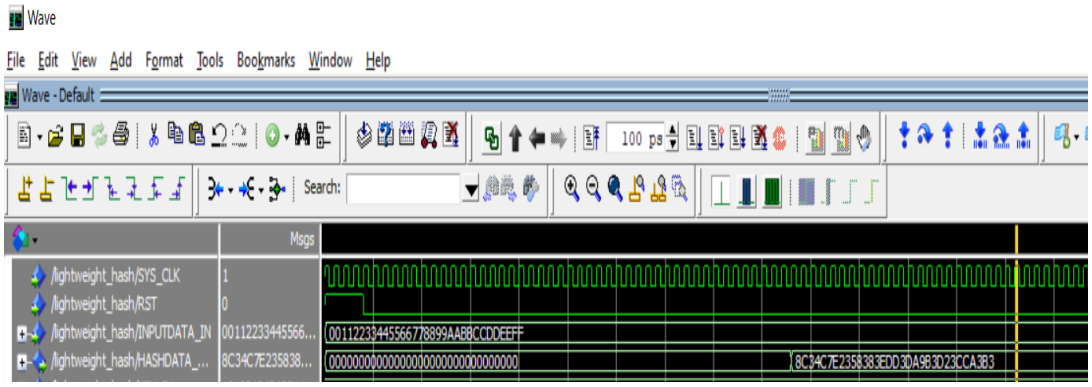


**Figure 8**: simulation result of lightweight hash

Initially, input data is applied at every positive edge of the clock with an active low reset while enable is high, and an output hash value is created after a series of cycles. To produce the output hash values, the various test data are tested. The produced hash values are used to confirm the output hash values once again.

Result Synthesis

The complete SHA256 implementation on various ALTERA FPGA devices is given in Table 1. For all the considered FPGA families, the throughput increases linearly with the number of high-performance FPGAs. This is justified by the fact that the number of clock cycles is the dominant factor of the throughput. The design reaches a maximum throughput of 1090.512 Mbps on StratixIII with an efficiency of 0.327 Mbps/lr.

**Table 1: Performance Evaluation of Parallel Architecture based SHA256 on various FPGA device platforms**

| Evaluation Parameters | **FPGA Devices** | | | |
|---|---|---|---|---|
| | **CycloneII-EP2C35F672C6** | **CycloneIV-EP4CE75F29C7** | **StratixII-EP2S15F672I4** | **StratixIII-EP3SE50F780I4L** |
| **Logic Resources (LR)** | 4899 | 4893 | 3321 | 3334 |

| | | | | |
|---|---|---|---|---|
| **Min Propagation Delay (ns)** | 7.573 | 3.714 | 6.761 | 3.668 |
| **Max Frequency (MHz)** | 132.048 | 269.251 | 147.907 | 272.628 |
| **Power Consumption (mW)** | 65.13 | 42.10 | 31.64 | 44.92 |
| **Throughput (Mbps)** | 528.192 | 1077.004 | 591.628 | 1090.512 |
| **Efficiency (Mbps/LR)** | 0.107 | 0.220 | 0.178 | 0.327 |

Tables 2 show the full implementation of lightweight hash processor on a range of ALTERA FPGA devices. The throughput increases approximately linearly with the quantity of high-performing FPGAs in each of the assessed FPGA families. Since the throughput is mostly determined by the number of clock cycles, this makes sense. On Stratix III, the design achieves a maximum throughput of 1090.512 Mbps with an efficiency of 0.327 Mbps/lr.

**Table 2: Performance Evaluation of Lightweight Hash on various FPGA device platforms**

| Evaluation Parameters | FPGA Devices | | | |
|---|---|---|---|---|
| | **CycloneII-EP2C35F672C6** | **CycloneIV-EP4CE75F29C7** | **StratixII-EP2S15F672I4** | **StratixIII-EP3SE50F780I4L** |
| **Logic Resources (LR)** | 2835 | 2880 | 2236 | 2378 |
| **Min Propagation Delay (ns)** | 5.42 | 2.12 | 4.78 | 2.06 |
| **Max Frequency (MHz)** | 98.55 | 150.45 | 100.80 | 143.85 |
| **Power Consumption (mW)** | 50.74 | 38.64 | 25.44 | 30.91 |

| | | | | |
|---|---|---|---|---|
| **Throughput (Mbps)** | 300.54 | 840.425 | 280.478 | 700.546 |
| **Efficiency (Mbps/LR)** | 0.1060 | 0.2918 | 0.1254 | 0.2945 |

The performance of several FPGA devices is visualized in Figure 8 to Figure 13 concerning logic resources, minimal propagation delay, maximum frequency, power consumption, throughput, and efficiency. The Cyclone and Stratix FPGA families, which are often low-power and high-performance FPGA, are the two primary FPGA kinds on which the evaluation performance is concentrated. A Stratix II FPGA device with a minimal logic resource usage of 3321 results in a minimal power demand of 31.64 mW for lightweight hash. I/O thermal power consumption accounts for the power usage. The StratixIII device, which has strong FPGA performance, has a minimum propagation delay of 3.668 ns, also referred to as clock-to-output latency. Due to its increased throughput, the StratixIII device has an efficient efficiency performance (0.327) in terms of the ratio of throughput to logic resources used for lightweight hash.

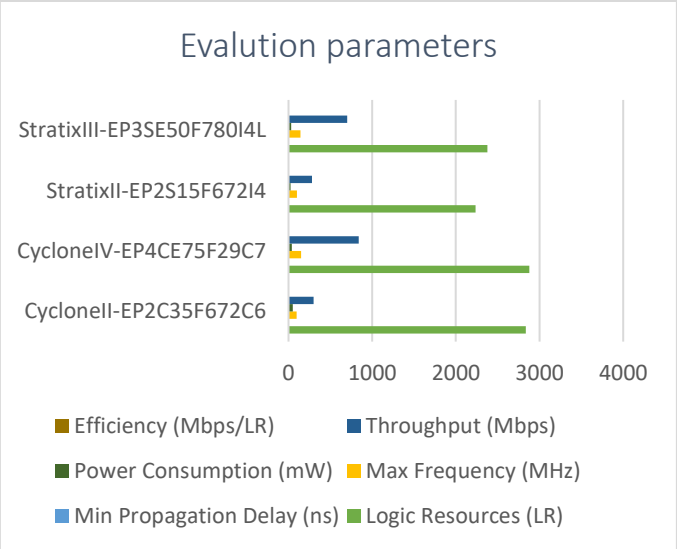


**Figure 9:** Evaluation parameters of Lightweight Hash

**Comparative Analysis**

Table 3, comparison against conventional hash and lightweight hash processors based on ALTERA Stratix-III FPGA platform targeted for low power design architecture. The logic resources used in the conventional hash are much larger with 3337 elements than those used in the lightweight hash with 2378 elements. Relatively, power consumption in lightweight hash is found to be much smaller with 30.91mW which is much better for blockchain header creation.

**Table 3**: Comparative Analysis of Conventional Hash and Lightweight Hash

| Evaluation Parameters | Conventional Hash Processor | Lightweight Hash Processor |
|---|---|---|
| Logic Resources (LR) | 3334 | 2378 |
| Min Propagation Delay (ns) | 3.668 | 2.06 |
| Power Consumption (mW) | 44.92 | 30.91 |

In Table 4, a comparison against various architectures based on the ALTERA FPGA platform [38][39] is just a reference to show the saving in logic resources that can be achieved. Compared to existing state of art work, our approach requires minimal area resources. However, considering both metrics, maximum frequency, and throughput performance, our design is more efficient with 272.628MHz and 1090.512Mbps respectively.

**Table 4**: Comparative Analysis

| Research Work | Platform | Logic Resources | Max Freq. (MHz) | Throughput (Mbps) |
|---|---|---|---|---|
| **[38]** | Stratix II | 2150 | 143.164 | 909.816 |
| **[39]** | Stratix EP1S10F484C5 | 104760 | 74 | 595 |
| **Conventional Hash** | StratixIII-EP3SE50F780I4L | 3334 | 272.628 | 1090.512 |

Table 5 illustrates the savings in logic resources by contrasting many designs using the ALTERA FPGA platform [69][98]. Compared to the state of the art, our solution uses a lot less hardware logic area resources. Our approach, which clocks in at 150.45MHz and 840.425Mbps, respectively, is superior when maximum frequency and throughput performance are also considered.

**Table 5**: Comparative Analysis

| Ref Work | Platform | Logic Resources | Max Freq. (MHz) | Throughput (Mbps) |
|---|---|---|---|---|
| **[40]** | Cyclone-IV | 104760 | 74 | 595 |
| **[41]** | Spartan 6 | 2150 | 143.164 | 709.816 |

| | | | | |
|---|---|---|---|---|
| **Lightweight Hash** | Cyclone-IV | 2880 | 150.45 | 840.425 |

## CONCLUSION

With the potential to support multiple application contexts addresses a wide range of issues. Among the most ground-breaking innovations recently, blockchain has taken center stage. For blockchain to be integrated into technologies like the Internet of Things, consensus algorithms and blockchain architecture must be improved. This paper proposed an efficient FPGA-based architecture for implementing secure hash functions with lightweight capability that can fulfill the requirements of blockchain algorithms. Initially, the parallel architecture-based conventional SHA-256 hash function is developed using VHDL language and its functionality is verified using the ModelSim Simulator tool. Later, a lightweight hash function is developed. Bit permutation, linear transformation, and S-Box functionality are used in the proposed design considering the motivation of SHA-256 and AES algorithm. The results demonstrate that the proposed architecture performs better in terms of speed, memory, and power consumption than other lightweight protocols and parallel architecture-based hash functions. The proposed FPGA-based solutions can significantly improve the performance and energy efficiency of blockchain hashing processes, making them ideal for deployment in blockchain networks that require high-speed, low-power cryptographic operations. Future work will focus on integrating more complex hash functions and extending the design to support multiple blockchain consensus algorithms.

## Refrences

[1] Santos CEB Jr, Silva LMDD, Torquato MF, Silva SN, Fernandes MAC. SHA-256 Hardware Proposal for IoT Devices in the Blockchain Context. Sensors (Basel). **2024 Jun 17**;24(12):3908. doi: 10.3390/s24123908. PMID: 38931692; PMCID: PMC11207617.

[2] J. Ktari, T. Frikha, M. Hamdi and H. Hamam, "Enhancing Blockchain Consensus With FPGA: Accelerating Implementation for Efficiency," in IEEE Access, vol. 12, pp. 44773-44785, **2024**, doi: 10.1109/ACCESS.2024.3379374.

[3] Gookyi, D. A., & Ryoo, K. A Lightweight System-On-Chip Based Cryptographic Core for Low-Cost Devices. Sensors, 22(8), 3004. (2021) https://doi.org/10.3390/s22083004

[4] M. O. A. Al-Shatari, F. A. Hussin, A. A. Aziz, G. Witjaksono and X. -T. Tran, "FPGA-Based Lightweight Hardware Architecture of the PHOTON Hash Function for IoT Edge Devices," in IEEE Access, vol. 8, pp. 207610-207618, **2020**, doi: 10.1109/ACCESS.2020.3038219.

[5] S. Windarta, S. Suryadi, K. Ramli, B. Pranggono and T. S. Gunawan, "Lightweight Cryptographic Hash Functions: Design Trends, Comparative Study, and Future Directions," in IEEE Access, vol. 10, pp. 82272-82294**,** **2022**, doi: 10.1109/ACCESS.2022.3195572.

[6] Mohajerani, Kamyar, Richard Haeussler, Rishub Nagpal, Farnoud Farahmand, Abubakr Abdulgadir, Jens-Peter Kaps and Kris Gaj. “FPGA Benchmarking of Round 2 Candidates

in the NIST Lightweight Cryptography Standardization Process: Methodology, Metrics, Tools, and Results." IACR Cryptol. ePrint Arch. 2020 (2020): 1207.

[7] Hadj Youssef, W. E., Abdelli, A., Dridi, F., & Machhout, M. (2019). Hardware Implementation of Secure Lightweight Cryptographic Designs for IoT Applications. Security and Communication Networks, 2020(1), 8860598. https://doi.org/10.1155/2020/8860598

[8] Manel, Kammoun & Elleuchi, Manel & Mohamed, Abid & Obeid, Abdulfattah (2021). HW/SW Architecture Exploration for an Efficient Implementation of the Secure Hash Algorithm SHA-256. Journal of Communications Software and Systems. 17. 87-96. 10.24138/jcomss-2021-0006.

[9] Michael DiNardi, Damu Radhakrishnan, "SHA-256 Hash Function on Intel DE10 Lite FPGA", International Journal For Research in Applied Science and Engineering Technology, 2023, https://doi.org/10.22214/ijraset.2023.57521

[10] D. -e. -S. Kundi, A. Khalid, A. Aziz, C. Wang, M. O'Neill and W. Liu, "Resource-Shared Crypto-Coprocessor of AES Enc/Dec With SHA-3," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 12, pp. 4869-4882, Dec. **2020,** doi: 10.1109/TCSI.2020.2997916.

[11] H. L. Pham, T. H. Tran, V. T. Duong Le and Y. Nakashima, "A High-Efficiency FPGA-Based Multimode SHA-2 Accelerator," in IEEE Access, vol. 10, pp. 11830-11845, **2022,** doi: 10.1109/ACCESS.2022.3146148.

[12] Frikha, T., Chaabane, F., Aouinti, N., Cheikhrouhou, O., Amor, N. B., & Kerrouche, A. (2020). Implementation of Blockchain Consensus Algorithm on Embedded Architecture. Security and Communication Networks, **2021(1**), 9918697. https://doi.org/10.1155/2021/9918697

[13] S. Khan, W. -K. Lee, A. Karmakar, J. M. B. Mera, A. Majeed and S. O. Hwang, "Area–Time Efficient Implementation of NIST Lightweight Hash Functions Targeting IoT Applications," in IEEE Internet of Things Journal, vol. 10, no. 9, pp. 8083-8095, **1 May, 2023,** doi: 10.1109/JIOT.2022.3229516.

[14] Suhaili, Shamsiah & Norhuzaimin, Julai.. FPGA-based Implementation of SHA-256 with Improvement of Throughput using Unfolding Transformation. Pertanika Journal of Science and Technology. **2022** 30. 581-603. 10.47836/pjst.30.1.32.

[15] Wali, Heera & Iyer, Nalini.. Power, Performance and Area Analysis of Sponge Based Lightweight HASH Function. International Journal of Electrical and Electronic Engineering & Telecommunications. **(2023**), 245-256. 10.18178/ijeetc.12.4.245-256.

[16] Huang SC, Huang S, Yin HL, Ma QL, Yin ZJ. High-Speed Variable Polynomial Toeplitz Hash Algorithm Based on FPGA. Entropy (Basel). **2023 Apr 11**;25(4):642. doi: 10.3390/e25040642. PMID: 37190430; PMCID: PMC10137740.

[17] X. Cui, H. Shi, J. Zhao, Y. Ge, Y. Yin and K. Zhao, "High Accuracy Short Reads Alignment Using Multiple Hash Index Tables on FPGA Platform," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, **2020**, pp. 567-573, doi: 10.1109/ITOEC49072.2020.9141738.

[18] Abed, S., Jaffal, R., Mohd, B.J. et al. An analysis and evaluation of lightweight hash functions for blockchain-based IoT devices. Cluster Comput 24, 3065–3084 **(2021).** https://doi.org/10.1007/s10586-021-03324-1.

[19] Monika Parmar and Harsimran Jit Kaur, "Comparative Analysis of Secured Hash Algorithms for Blockchain Technology and Internet of Things" International Journal of Advanced Computer Science and Applications(IJACSA), 12(3), **2021**. http://dx.doi.org/10.14569/IJACSA.2021.0120335.

[20] Lam, D. K., Le, V. T., & Tran, T. H. Efficient Architectures for Full Hardware Scrypt-Based Block Hashing System. Electronics, 11(7), 1068. **(2021).** https://doi.org/10.3390/electronics11071068

[21] Al-Odat, Zeyad & Ali, Mazhar & Abbas, Assad & Khan, Samee. Secure Hash Algorithms and the Corresponding FPGA Optimization Techniques. ACM Computing Surveys. 53. 1-36. 10.1145/3311724, **(2020).**

[22] J. -B. Choi, D. -S. Kim, J. -Y. Choe and K. -W. Shin, "Hardware Implementation of ECIES Protocol on Security SoC," 2020 International Conference on Electronics, Information, and Communication (ICEIC), Barcelona, Spain, **2020**, pp. 1-4, doi: 10.1109/ICEIC49074.2020.9051263.

[23] T. Claesen, A. Sateesan, J. Vliegen and N. Mentens, "Novel Non-cryptographic Hash Functions for Networking and Security Applications on FPGA," 2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, Italy, **2021**, pp. 347-354, doi: 10.1109/DSD53832.2021.00059.

[24] Gookyi, Dennis & Ryoo, Kwangki. NIST Lightweight Cryptography Standardization Process: Classification of Second Round Candidates, Open Challenges, and Recommendations. Journal of Information Processing Systems. 17. 253 - 270**. (2021).**

[25] Alfrhan, Aishah & Moulahi, Tarek & Alabdulatif, Abdulatif.. Comparative study on hash functions for lightweight blockchain in Internet of Things (IoT). Blockchain: Research and Applications. 2. 100036. 10.1016/j.bcra.2021.100036. **(2021)**

[26] Arturo, C., & Alberto, L. A SHA-256 Hybrid-Redundancy Hardware Architecture for Detecting and Correcting Errors. Sensors, 22(13), 5028**. (2021).** https://doi.org/10.3390/s22135028

[27] Abid Muslam, Alaa & Ali, Abid & Joundy, Manar & Mabrouk, Mohamed & Zrigui, Mounir. (2023). Proposal of a Modified Hash Algorithm to Increase Blockchain Security. 10.13140/RG.2.2.36352.40968.

[28] P.Pavithara, R.Renuka, P.Sabena Yasmin, K.Naresh, "Design and FPGA implementation of folded SHA-256 using 4-2 adder compressor", Natural Volatiles & Essential Oils Journal, **(2021),** Issue: Volume: 8 Issue: 5.

[29] Assad, Fatima & Fettach, Mohamed & el Otmani, Fadwa & Tragha, Abderrahim. High-performance FPGA implementation of the secure hash algorithm 3 for single and multi-message processing. International Journal of Electrical and Computer Engineering (IJECE). **(2022)** 12. 1324. 10.11591/ijece.v12i2.pp1324-1333.

[30] H. -Y. Kim, L. Xu, W. Shi and T. Suh, "A Secure and Flexible FPGA-Based Blockchain System for the IIoT," in Computer, vol. 54, no. 2, pp. 50-59, Feb. **2021,** doi: 10.1109/MC.2020.3022066.

[31] Rawal, Bharat & Satheesh, Naidu & Aleti, Satheesh & Reddy, Saikethan & Chandu, Triven. **(2022**). Optimization of SHA 256 with Finetune Pipeline and Parallel Processing with Split Techniques.

[32] Sideris, A., Sanida, T., & Dasygenis, M. A Novel Hardware Architecture for Enhancing the Keccak Hash Function in FPGA Devices. Information, 14(9),475. **(2023).** https://doi.org/10.3390/info14090475

[33] Davda, Yatri. "Design of Hash Algorithm for Blockchain Security." Blockchain Applications in Cryptocurrency for Technological Evolution, edited by Atour Taghipour, IGI Global, **2023**, pp. 118-135. https://doi.org/10.4018/978-1-6684-6247-8.ch007

[34] K. Kumar, K. R. Ramkumar, A. Kaur and S. Choudhary, "A Survey on Hardware Implementation of Cryptographic Algorithms Using Field Programmable Gate Array," 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, **2020**, pp. 189-194, doi: 10.1109/CSNT48778.2020.9115742.

[35] Manne Muddu Sravani, Ananiah Durai Sundarajan, Nabihah Ahmad; FPGA implementation of novel hybrid hash function SHAES for digital signatures. AIP Conf. Proc. 26 October **2023**; 2564 (1): 030004. https://doi.org/10.1063/5.0123048.

[36] Z. A. Al-Odat, E. M. Al-Qtiemat and S. U. Khan, "An Efficient Lightweight Cryptography Hash Function for Big Data and IoT Applications," 2020 IEEE Cloud Summit, Harrisburg, PA, USA, **2020,** pp. 66-71, doi: 10.1109/IEEECloudSummit48914.2020.00016Vijayakumar Peroumal, Rajashree R, Anusha Kulkarni and Prachi Thakur, “FPGA Implementation of Secure Block CreationAlgorithm for Blockchain Technology”, ECS Transactions, Volume 107, Number 1, DOI: 10.1149/10701.5519ecst.

[37] S. binti Suhaili and T. Watanabe, "Design of high-throughput SHA-256 hash function based on FPGA," 2017 6th International Conference on Electrical Engineering and Informatics (ICEEI), Langkawi**, 2017**, pp. 1-6, doi:10.1109/ICEEI.2017.8312449.

[38] R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, “A compact FPGA-based processor for the Secure Hash Algorithm SHA-256,” Computers & Electrical Engineering, vol. 40, no. 1, pp. 194–202, Jan. **2014,** doi: https://doi.org/10.1016/j.compeleceng.2013.11.014.

[39] F.Assad, F. Elotmani, M. Fettach and A. Tragha, "An optimal hardware implementation of the KECCAK hash function on virtex-5 FPGA," 2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBIoTS), Casablanca, Morocco, **2019**, pp. 1-5, doi: 10.1109/SysCoBIoTS48768.2019.9028028.

[40] M. Sundal and R. Chaves, "Efficient FPGA Implementation of the SHA-3 Hash Function," 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, **2017**, pp. 86-91, doi: 10.1109/ISVLSI.2017.24.

**[41]** Preeti Lawhale, Sujata.N. Kale. "A Survey On Secure Architectures Using Hash Function Based On FPGA for Block Chain Enabled IoT Devices" **,**11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP), **2023**

.