International Journal of Early Childhood Special Education (INT-JECSE)
DOI:10.48047/intjecse/v18i3.7 ISSN: 1308-5581 Vol 18, Issue 03 2026

A study of the use of hash functions in logic gate arrays in IoT applications.

Abbass Hussien Saleh

abbass@ik.edu.iq

Ibn Khaldun Private University College, Baghdad

Abstract:

This study presents the design and implementation of the Ascon lightweight hash function, compliant with the NIST standard, on FPGA for Internet of Things (IoT) applications. The hash function is specifically designed to reduce energy consumption and enhance performance in resource-constrained environments, utilizing optimization algorithms to minimize hardware complexity. Simulation and implementation results demonstrate that the proposed design outperforms existing methods in terms of speed, power consumption, and hardware resource utilization, making it an ideal solution for security applications in IoT.

Research Aim: The aim of this study is to design and implement a lightweight hash function based on the NIST standard on FPGA to meet the security requirements of the Internet of Things (IoT). The Ascon algorithm is identified as an optimal candidate for this implementation. The hash function is designed to minimize power consumption and hardware footprint while ensuring sufficient performance and security for resource-constrained IoT environments. This design enables the provision of optimized and scalable solutions for securing data in small, widely used IoT devices.

Research method: This research involves several main steps. First, the security requirements and hardware/software constraints associated with the Internet of Things (IoT) are analyzed. Subsequently, a lightweight

hashing algorithm based on the NIST standard is selected and optimized for FPGA implementation, with the Ascon algorithm chosen for this thesis. The hardware design of the algorithm is carried out using hardware description languages such as VHDL, and simulation and synthesis tools are employed to evaluate its performance, power consumption, and resource utilization. Finally, the design is implemented on a real FPGA platform, and its efficiency in terms of speed, power consumption, and hardware resource usage is assessed and compared with existing approaches.

Findings: The VHDL implementation demonstrates that the designed hash function can process data at high speeds, making it ideal for real-time IoT applications. The design is optimized to minimize the use of FPGA hardware resources, including gates, lookup tables (LUTs), and memory blocks, enabling deployment on small and cost-effective FPGAs. Furthermore, the VHDL implementation shows that the design is highly adaptable to various FPGA architectures and can be customized for a wide range of IoT applications.

Keywords: Hash Functions, **Lightweight, FPGA, NIST, IoT Applications**

Introduction:

The Internet of Things (IoT) as an emerging technology has created the impetus for many innovative applications[1]. The Internet of Things (IoT) can significantly improve the quality of life through intelligent applications when combined with other important technologies such as artificial intelligence and cloud computing. Popular examples such as smart cities and smart homes are only possible when the Internet of Things is integrated with other appropriate technologies [2]. In an IoT environment, sensitive and important data is transmitted at high speed. The integrity of the transmitted data must remain intact to ensure secure communications. Data integrity is highly desirable for IoT applications, which can be achieved using cryptographic hash functions. A hash function converts an input hash of arbitrary length into an output hash value of fixed length. It can be used to verify the authenticity of messages (e.g., IoT sensor data) that are communicated over IoT networks. In IoT systems that require lightweight

solutions due to resource constraints such as power consumption, memory, and processing power, lightweight synchronization functions act as one of the most important tools in ensuring security. One of the ways to design lightweight synchronization functions is to use FPGAs[3]. .

General Steps of Encryption in Ascon:

The ASCON algorithm consists of a number of inputs and outputs. The inputs include the key, nonce, associated data, and plaintext.

Key: A 128-bit secret key shared by the sender and receiver. 2. Unique number: A unique 128-bit number that must be different for each message. 3. Associated data: Data that is not encrypted but whose integrity must be checked (can also be a zero bit). 4. Plaintext: Data that is to be encrypted. Outputs include 5. Ciphertext: Encrypted data generated from the plaintext. 6. Label: A 128-bit label that It is used to verify the integrity and integrity of data[4].

Ascon is designed to implement authenticated encryption with associated data (AEAD), which includes encryption, authentication, and data integrity. Here, the implementation steps of Ascon AEAD (such as Ascon-128) are explained in detail. The Ascon cipher suite includes a family of authenticated encryption schemes. Ascon and the Ascon-Hash hash function are based on the Ascon-Xof extensible output function.

Confirmed Encryption :

For Ascon confirmed encryption schemes, the family members are parameterized by key length $160 \leq k$ bits, rate (data block size) r, and inner round numbers a and b. Each scheme specifies a confirmed encryption algorithm Ek,r,a,b and a decryption algorithm Dk,r,a,b[5]. The encryption method is authenticated by Ek,r,a,b, a secret key K of k bits, a

The nonce takes as input a 128-bit public message number N, associated data A of arbitrary length, and a plaintext P of arbitrary length. This produces an output consisting of the verified ciphertext C of the plaintext length P plus a 128-bit authentication tag T that authenticates both the associated data and the encrypted message:

$$\varepsilon_{k,r,a,b}(K, N, A, P) = (C, T) \qquad 1$$

The decryption and authentication method Dk,r,a,b takes as input the key K, N, nonce, the associated data A, the ciphertext C, and the tag T, and if the tag authentication is correct[6]. If the tag verification fails, the output shows:

$$D_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}. \qquad 2$$

Confirmed Encryption Table 1-3 lists recommended examples for authentication encryption and specifies their parameters, including the key size k, the nonce and tag constants, the rate r, and the number of rounds a for initial and finalization pa and b for intermediate substitution pb for processing linked data and plaintext[7]. The list is ordered by priority: the primary recommendation is Ascon-128 and the secondary recommendation is Ascon-128a. Both schemes are identical to the CAESAR candidates that were selected as the primary choices for the lightweight use cases in the CAESAR final working example.

Table 1. Parameters of the different ASCON versions [9]

Name	Algorithms	BIT SIZE OF				Rounds	
		key	nonce	tag	data block	P^a	P^b
Ascon-128	ε, $D_{128,64,12,6}$	128	128	128	64	12	6
Ascon-128a	ε, $D_{128,128,12,8}$	128	128	128	128	12	6

Table 2 lists a suggested hash pattern and specifies its parameters, including the hash output size h, the rate r, and the number of rounds a to replace pa. The list is ordered by priority: the first recommendation is Ascon-Hash only.

Table 2. ASCON-Hash Parameters [9]

Name	Algorithms	bit size of		Rounds
		hash	data block	p^a

ASCON-HAsH	**$X_{256,64,12}$ WITH e = 256**	**256**	**64**	**12**

If you want to pair confirmed encryption with hashing, it is recommended to pair the primary proposals, Ascon-128 and Ascon-Hash, as they both have the same rate.

Encryption and Confirmed Encryption Algorithms

The Ascon mode of operation for authenticated encryption is based on two-way modes like Monkey Duplex but uses a stronger key initialization and key finalization function[8].

Encryption operations and the decoding is shown in Figure 1 and Figure-2, and its pseudocode is specified in Figure 3.

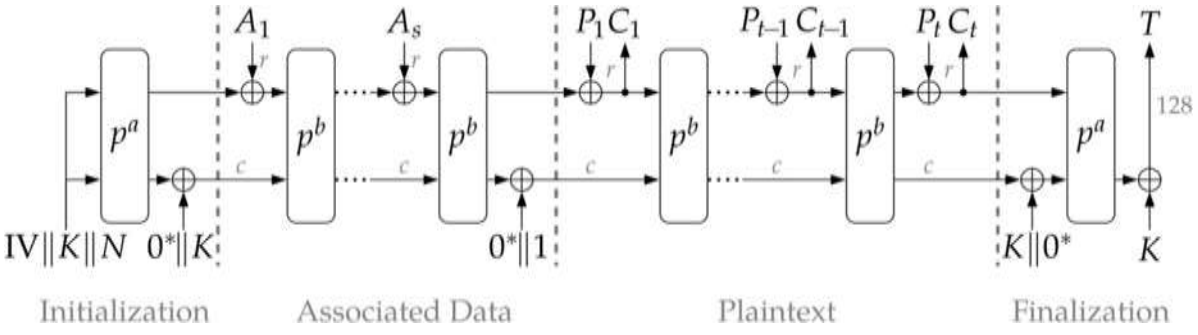

Figure 1. Encryption in ASCON algorithm [9]

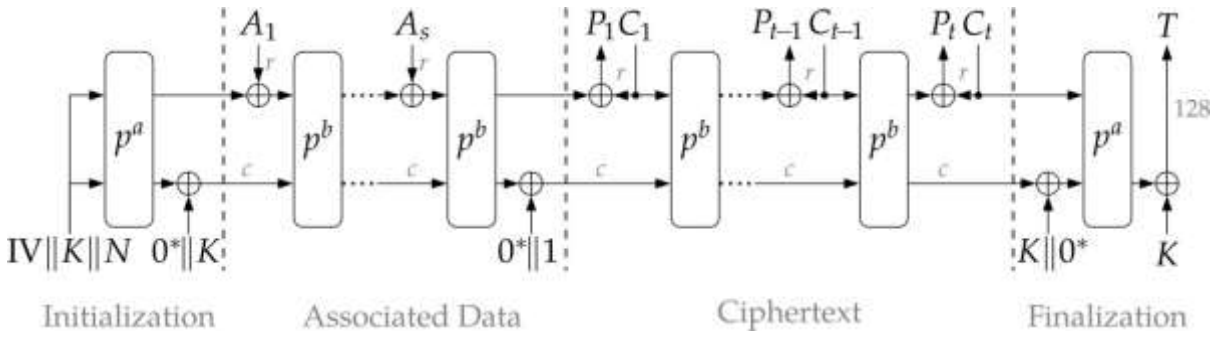

Figure 2. Decoding in ASCON algorithm

The Ascon algorithm consists of three main stages:

Initialization: In this stage, the key, nonce, and initial constant values 1- are placed into the internal state of the algorithm[10].

The 320-bit Ascon initial state is formed by the secret key K of k bits and the nonce N of 128 bits, as well as an IV that specifies the algorithm (including the key size k, the rate r, the number of initial and final rounds). a and the number of intermediate rounds b, each expressed as an integer 8 The verses are written:

$$S \leftarrow Pa(s) \oplus (0^{320-k} \parallel K) \qquad 3$$

Processing of related data 1:

In this stage, the related data is processed to ensure its integrity[11].

Ascon processes the related data A in blocks of r bits. It adds a 1 and the smallest 0 to A to obtain a multiple of r bits and divides it into s blocks of r bits, ||As A2||.. A1||. If A is empty, no padding is applied and S=0

Design of NIST lightweight synchronization function on FPGA for use in the Internet of Things

Figure 3: Pseudocode of encryption and decryption steps in ASCON

Authenticated Encryption $\mathcal{E}_{k,r,a,b}(K,N,A,P)$	Verified Decryption $\mathcal{D}_{k,r,a,b}(K,N,A,C,T)$
Input: key $K \in \{0,1\}^k, k \le 160$, nonce $N \in \{0,1\}^{128}$, associated data $A \in \{0,1\}^*$, plaintext $P \in \{0,1\}^*$	**Input:** key $K \in \{0,1\}^k, k \le 160$, nonce $N \in \{0,1\}^{128}$, associated data $A \in \{0,1\}^*$, ciphertext $C \in \{0,1\}^*$, tag $T \in \{0,1\}^{128}$
Output: ciphertext $C \in \{0,1\}^{\lvert P\rvert}$, tag $T \in \{0,1\}^{128}$	**Output:** plaintext $P \in \{0,1\}^{\lvert C\rvert}$ or $\perp$
Initialization $S \leftarrow \mathrm{IV}_{k,r,a,b} \parallel K \parallel N$ $S \leftarrow p^a(S) \oplus (0^{320-k} \parallel K)$	**Initialization** $S \leftarrow \mathrm{IV}_{k,r,a,b} \parallel K \parallel N$ $S \leftarrow p^a(S) \oplus (0^{320-k} \parallel K)$

Processing Associated Data
if $\lvert A\rvert > 0$ **then**
$A_1 \dots A_s \leftarrow r$-bit blocks of $A \parallel 1 \parallel 0^*$
for $i = 1, \dots, s$ **do**
$S \leftarrow p^b((S_r \oplus A_i) \parallel S_c)$
$S \leftarrow S \oplus (0^{319} \parallel 1)$
	Processing Associated Data
	if $\lvert A\rvert > 0$ **then**
	$A_1 \dots A_s \leftarrow r$-bit blocks of $A \parallel 1 \parallel 0^*$
	for $i = 1, \dots, s$ **do**
	$S \leftarrow p^b((S_r \oplus A_i) \parallel S_c)$
	$S \leftarrow S \oplus (0^{319} \parallel 1)$

Processing Plaintext $P_1 \dots P_t \leftarrow r$-bit blocks of $P \parallel 1 \parallel 0^*$ **for** $i = 1, \dots, t-1$ **do**	**Processing Ciphertext** $C_1 \dots C_{t-1} \tilde{C}_t \leftarrow r$-bit blocks of C, $0 \le \lvert\tilde{C}_t\rvert < r$ **for** $i = 1, \dots, t-1$ **do**

4

For each block Ai with i=1,2,… the first bits of Sr r are xored from state S,

Summary and comparison of three ASCON implementation methods and simulation results

Methods for implementing the ASCON algorithm on FPGA include three main approaches: Iterative, Serial, and Unrolling[12]. Each of these methods has its own advantages and limitations, which are described and compared below. In the iterative implementation, each stage of the ASCON algorithm is executed as a loop, and each round is processed in one clock cycle. A finite state machine (FSM) is used to control the flow of the algorithm.

The execution speed of this method is the most efficient in terms of hardware resources and is suitable for systems with limited resources. However, this method is slower than other methods, because each round requires several clock cycles [13]. The maximum operational capacity is up to

LUT 1237 in serial implementation in this registers and 921 about Gb/s. 2.148 and moderate resource consumption including number method,

operations are performed completely serially, meaning that only one simple operation such as XOR or NIST Lightweight Synchronizer

Function Design on FPGA for Application in the Internet of Things 44

is processed on limited bits at a time. Minimal hardware resource consumption and suitable for low-resource devices such as IoT systems. However, the lower speed and increased overall latency are due to the sequential execution of operations. The maximum throughput is about 0.347 Gb/s and the resource consumption is lower, but the number of LUTs is 1532 and 921 registers are used, which leads to lower efficiency. In the Unrolling implementation, the processing loops are opened and several stages of the algorithm are executed simultaneously[14]. This increases the speed and reduces the overhead of the loops. High speed and increased throughput, suitable for systems that require high performance. But consume more hardware resources and increase design complexity. In particular, in the Unrolling2 and Unrolling4 methods, the code size increases. In the Unrolling2 method, the maximum throughput is up to 0.582 Gb/s with the number of LUTs. 1802, and in the Unrolling4 method, the throughput is increased and reaches about 0.727 Gb/s, but the clock frequency is reduced. As a result, the iterative method, with its optimal resource consumption and moderate speed, is suitable for general applications. The serial method, with its lower resource consumption, is suitable for low-resource systems but at the cost of reduced speed, and the unrolling method, especially unrolling4, offers the best performance in terms of speed and processing power, but requires more hardware resources and is more complex to design[15]. The choice of implementation method depends on the application requirements: for low resource consumption, the serial method is suitable; for a balance between resources and efficiency, the iterative method is recommended; and for maximum operational capacity, the unrolling method is optimal[16].

Discussion and Conclusion:

The Ascon method was chosen because it was one of the ten algorithms selected in the NIST standard for encryption and decryption methods and was selected as the best algorithm among them. The main goal in implementing it on various platforms is to improve performance, reduce resource and power consumption, and ensure data security in IoT devices.

The results obtained showed that using FPGA as a hardware platform has provided high flexibility and precise optimization, so that the designed function can meet the processing needs of low-resource systems. In addition, the use of optimization techniques such as reducing the occupied area, improving operational efficiency, and maintaining security against common attacks such as side-channel attacks makes this design a suitable option for various applications in the IoT field. Comparison with previous methods also showed that this design has significant improvements in terms of efficiency and security and can be used as a practical solution in embedded systems and smart devices. The implementation of the ASCON algorithm is investigated and performed in the iterative, serial, and unrolling methods. The iterative method is faster than the serial method, but uses more resources [17]. The use of the serial architecture reduces the need for logical resources such as LUTs and registers, but at the same time it requires more time to complete the processing. The unrolling method uses more resources than the iterative method due to the opening of the loop. The unrolling method can be faster than the iterative method because the number of clocks is less than the iterative method when multiple loops are opened. However, in the Ascon algorithm, due to the dependence of each loop stage on the previous stage, high performance has not been observed in simulation. Based on the work done in this thesis, the following suggestions can be made for improvement and development of the work in the future. Implementing hybrid architectures, such as combining serial and parallel designs, can be used to improve performance while reducing resource consumption. For example, implementing critical parts of an algorithm such as S-Box in parallel and other parts in serial can create a trade-off between speed and resource consumption. Testing and evaluating the implementation on various FPGAs with more or less limited resources, such as low-cost series from Xilinx or Intel, which can verify the adaptability of the design to different hardware [18].

To continue the work, other Xillinx boards can be used. On the other hand, if the algorithm can combine the three operations that are performed on the data and repeated, this can greatly improve the speed of the algorithm. On the other hand, better results can be achieved by combining the techniques used and being precise in implementation.

References:

[1] Ivan Damg˚ard. A Design Principle for Hash Functions. In Crypto ’89, volume 435 of LNCS, pages 416–427. Springer-Verlag, 1989.

[2] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge Functions. In ECRYPT Hash Workshop, 2007.

[3] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Asiacrypt ’05, volume 3788 of LNCS, pages 474–494. Springer-Verlag, 2005

[4] Orr Dunkelman and Eli Biham. A Framework for Iterative Hash Functions – HAIFA. In 2nd NIST Cryptographich Hash Workshop, 2006.

[5] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function, 2008

[6] Alfred Menezes, Paul Oorschot, and Scott Vanstone. Handbook of Applied Cryptography, chapter Hash Functions and Data Integrity, pages 321–384. CRC Press, 1996.

[7] W. Liang and Long Jing, "A cryptographic algorithm based on Linear Feedback Shift Register," 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Taiyuan, China, 2010, pp. V15-526-V15-529, doi: 10.1109/ICCASM.2010.5622523.

[8] Nouri, Mahdi & Safarinia, Mahyar & Pourmahdi, Payam & Garshasebi, Mohammad. (2014). The Parallel One-way Hash Function Based on Chebyshev-Halley Methods with Variable Parameter. International Journal of Computers, Communications and Control. 9. 24-36. 10.15837/ijccc.2014.1.867.

[9] Dobraunig, C., Eichlseder, M., Mendel, F. et al. ASCON v1.2: Lightweight Authenticated Encryption and Hashing. J Cryptol **34**, 33 (2021). https://doi.org/10.1007/s00145-021-09398-9

[10] N. Baurai, B. Patnaik, B. Acharya and Z. Mishra, "Area optimised implementation of block cipher ELEPHANT for resource constrained devices in cyber physical system," 2024 15th International Conference on Computing Communication and Networking

Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10725970.

[11] N. Yasmin and R. Gupta, "Performance Analysis of Lightweight Algorithm GIFT-COFB for 3-D Printer Security," 2022 8th International Conference on Signal Processing and Communication (ICSC), Noida, India, 2022, pp. 475-478, doi: 10.1109/ICSC56524.2022.10009201.

[12] Fernández-García, Carlos & Mora, Jose & Jiménez-Fernández, Carlos. (2024). TinyJAMBU Hardware Implementation for Low Power. IEEE Access. PP. 1-1. 10.1109/ACCESS.2024.3438378.

[13] S. N. Allu, A. N. Tentu and G. R. Kancharla, "An Efficient Quantum cryptanalysis of lightweight hash function - PHOTON-Beetle," 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2023, pp. 1604-1610, doi: 10.1109/ICACRS58579.2023.10404523.

Daemen, Joan & Hoffert, Seth & Peeters, Michael & Assche, Gilles & Keer, Ronny. (2020). Xoodyak, a lightweight cryptographic scheme. IACR Transactions on Symmetric Cryptology. 60-87. 10.46586/tosc.v2020.iS1.60-87.

[14] Khairallah, Mustafa. (2021). Romulus: Lighweight AEAD from Tweakable Block Ciphers. 10.1007/978-981-16-6344-4_6.

[15] Beierle, Christof & Biryukov, Alex & Santos, Luan & Großschädl, Johann & Perrin, Léo & Udovenko, Aleksei & Velichkov, Vesselin & Wang, Qingju. (2020). Lightweight AEAD and Hashing using the Sparkle Permutation Family. IACR Transactions on Symmetric Cryptology. 208-261. 10.46586/tosc.v2020.iS1.208-261.

[16] Canteaut, Anne & Duval, Sébastien & Leurent, Gaëtan & Naya-Plasencia, María & Perrin, Léo & Pornin, Thomas & Schrottenloher, André. (2020). Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. IACR Transactions on Symmetric Cryptology. 160-207. 10.46586/tosc.v2020.iS1.160-207.

[17] W. J. Buchanan and L. Maglaras, "Review of the NIST Light-Weight Cryptography Finalists," 2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of

Things (DCOSS-IoT), Pafos, Cyprus, 2023, pp. 469-474, doi: 10.1109/DCOSS-IoT58021.2023.00079

[18] Mohamed El-Hadedy, Xinfei Guo, Kazutomo Yoshii, Yichen Cai, Robert Herndon, Bryan Banta, Wen-Mei Hwu, RECO-ASCON: Reconfigurable ASCON hash functions for IoT applications, Integration, Volume 93,2023,102061, ISSN 0167-9260, https://doi.org/10.1016/j.vlsi.2023.102061.