High-Performance FPGA Implementations of Lightweight ASCON-128 and ASCON-128a with Enhanced Throughput-to-Area Efficiency

Ahmet MALAL
ASELSAN Inc., Ankara, Turkey
Middle East Technical University, Ankara, Turkey
ahmet.malal@metu.edu.tr

***Abstract*—The ASCON algorithm was chosen for its efficiency and suitability for resource-constrained environments such as IoT devices. In this paper, we present a high-performance FPGA implementation of ASCON-128 and ASCON-128a, optimized for the throughput-to-area ratio. By utilizing a 6-round permutation in one cycle for ASCON-128 and a 4-round permutation in one cycle for ASCON-128a, we have effectively maximized throughput while ensuring efficient resource utilization. Our implementation shows significant improvements over existing designs, achieving 34.16% better throughput-to-area efficiency on Artix-7 and 137.58% better throughput-to-area efficiency on Kintex-7 FPGAs. When comparing our results on the Spartan-7 FPGA with Spartan-6, we observed a 98.63% improvement in throughput-to-area efficiency. However, it is important to note that this improvement may also be influenced by the advanced capabilities of the Spartan-7 platform compared to the older Spartan-6, in addition to the design optimizations implemented in this work.**

***Index Terms*—ASCON, Lightweight Cryptography, Hardware Cryptography, FPGA Implementation**

I. Introduction

The National Institute of Standards and Technology (NIST) announced the Lightweight Cryptography competition in 20018, to develop cryptographic standards that meet the needs of emerging technologies, such as IoT devices, RFID tags, and embedded systems [1]. These devices often have limited computational power, memory, and battery life, making traditional cryptographic algorithms too resource-intensive. The competition aimed to find algorithms that provide robust security while being efficient enough to run on these constrained platforms, ensuring that security is not compromised even in low-resource environments.

The standardization process for the Lightweight Cryptography competition consisted of three rounds. It began with NIST issuing a call for submissions, inviting researchers to propose algorithms that met the outlined criteria. In the first round, NIST received 57 submissions, which were then evaluated for their security, efficiency, and implementation features. This evaluation involved feedback from both NIST experts and the broader cryptographic community. Following the first round, 32 of the 57 submissions passed to the second round in March 2019. Out of these, 10 candidates were selected to proceed to the final round, announced in March 2021. After further evaluations, the `ASCON` family was declared the winner in February 2023, and the official NIST standard was published on June 16, 2023 [1].

Efficient implementation of cryptographic algorithms is essential, especially for resource-constrained devices like IoT and embedded systems [2], [3], [4]. In critical applications, such as healthcare or automotive systems, efficient algorithms help maintain robust security without compromising performance, making them vital for reliable and secure operations [5].

In this paper, we present a high-performance FPGA implementation of `ASCON-128` and `ASCON-128a`. We compared our results with those of other researchers focusing specifically on FPGA implementations. Our primary goal was to design an architecture that maximizes the throughput-to-area ratio, ensuring high efficiency. Given that IoT devices are often resource-constrained, achieving a high throughput while minimizing the use of hardware resources is crucial. This efficiency allows for better performance and lower costs, making the cryptographic solutions more suitable for practical applications.

The paper is structured as follows: Section 2 covers the `ASCON` algorithm. Section 3 details the proposed architecture and design. Section 4 discusses the results and comparisons. The paper concludes in Section V.

II. `ASCON` Specifications

In this section, we explained the key components of `ASCON` algorithm, providing a comprehensive explanation of each stage. Two different versions of `ASCON` are examined in the paper: `ASCON-128` and `ASCON-128a`. Algorithm related parameters are given in the Table I [6].

`ASCON` has a sponge construction with a 320-bit state, structured into four distinct stages: initialization, associated data processing, plaintext-ciphertext transformation, and finalization [6]. The general structure of the algorithm is illustrated in Figure 1.

TABLE I
PARAMETER TABLE OF ASCON

	ASCON-128	ASCON-128a
IV Bit Size	64	64
Key Bit Size	128	128
Nonce Bit Size	128	128
State Bit Size	320	320
Rate Bit Size	**64**	**128**
Capacity Bit Size	**256**	**192**
Tag Bit Size	128	128
Perm. Round (p^a)	12	12
Perm. Round (p^b)	**6**	**8**

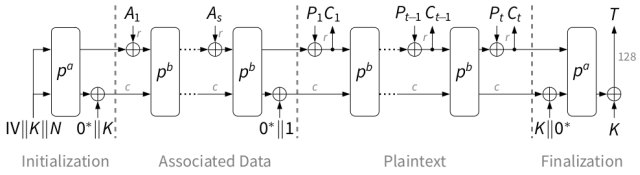

Fig. 1. The Structure of ASCON.

A. Initialization

The initial state is initialized with a 64-bit initialization vector (IV), a 128-bit key, and a 128-bit nonce. The permutation function p^a is then applied a fixed number of times, specifically 12 iterations, for both the ASCON-128 and ASCON-128a variants. Following the completion of these permutations, the 128-bit key is XORed with the final 128-bit of the state.

B. Associated Data

The bit rate, 64-bit for ASCON-128 and 128-bit for ASCON-128, is XORed with the initial rate bits of the state. Following this XOR operation, the permutation function p^b is applied, where p^b consists of 6 iterations for ASCON-128 and 8 iterations for ASCON-128a. Subsequently, the next block of associated data is XORed with the corresponding rate bits of the state, and this process is repeated iteratively until all associated data has been processed. After processing the final block of associated data, an additional p^b permutation is performed. Lastly, the domain parameter $0x1$ is XORed with the state.

C. Plaintext - Ciphertext

This stage is similar to the previous one, with the key difference being that ciphertext is produced during the XOR operation. The rate is 64-bit for ASCON-128 and 128-bit for ASCON-128a. Despite the increased number of rounds (p^b) in ASCON-128a, its performance is not negatively impacted compared to ASCON-128 due to the doubling of the rate. This process is repeated iteratively until all plaintext blocks have been processed. Finally, the initial 128-bit key is XORed with the first 128-bit of the capacity.

The decryption process is the same with encryption process, with the roles of plaintext and ciphertext reversed. This works effectively due to the properties of the XOR operation.

D. Finalization

The final p^a permutations are applied to the state. Following this, the 128-bit initial key is XORed with the last 128-bit of the state to generate the tag. The 128-bit tag is produced after the completion of all stages.

E. Permutation

This is the core function of the algorithm, aimed for permuting the bits of the state. As previously mentioned, the state consists of 320-bit. Both ASCON-128 and ASCON-128a use the same permutation function. A single round of the permutation consists three layers, which are as follows:

1) Adding Constant: The third block of the state is XORed with the corresponding constant value according to the round number using Table II.

TABLE II
ROUND CONSTANTS AND NUMBER OF ROUNDS FOR ASCON

ρ^{12}	ρ^8	ρ^6	Constant	ρ^{12}	ρ^8	ρ^6	Constant
0	0	0	00000000000000f0	6	2	0	0000000000000096
1	1	0	00000000000000e1	7	3	1	0000000000000087
2	2	0	00000000000000d2	8	4	2	0000000000000078
3	3	0	00000000000000c3	9	5	3	0000000000000069
4	0	0	00000000000000b4	10	6	4	000000000000005a
5	1	0	00000000000000a5	11	7	5	000000000000004b

2) S-Box Layer: This is the only non-linear layer of the algorithm, designed to be resistant against differential and linear attacks. ASCON-128 uses a 5×5 S-Box, meaning that each 5-bit input is mapped to a corresponding 5-bit output using an S-Box table. The 5×5 S-Box table is given in Table III.

Alternatively, it is possible to generate output bits directly without using the S-Box table, as each bit of the output can be formed as a nonlinear combination of the input bits. The specific formulas for this process are provided below. Assuming x is the 5-bit input and y is the 5-bit output, where $x = x_4x_3x_2x_1x_0$ and $y = y_4y_3y_2y_1y_0$.

$$
\begin{aligned}
y_0 &= x_4x_1 \oplus x_3x_2 \oplus x_2x_1 \oplus x_2x_0 \oplus x_1x_0 \oplus x_1 \oplus x_0 \\
y_1 &= x_4 \oplus x_2x_3 \oplus x_3 \oplus x_3x_1 \oplus x_2 \oplus x_1x_2 \oplus x_1 \oplus x_0 \\
y_2 &= x_4x_3 \oplus x_4 \oplus x_2 \oplus 1 \\
y_3 &= x_4x_0 \oplus x_3x_0 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \\
y_4 &= x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1
\end{aligned}
$$

The state of ASCON is represented by five 64-bit blocks. A single bit from each of these blocks is processed through the S-box layer across all 64 indices. The final state is obtained after all 64 indices have been processed.

We have implemented both methods on FPGA, and found no difference in resource utilization between them.

3) Linear Layer: This is the diffusion layer, where the 64-bit blocks of the state are shifted and XORed in a specific order. The precise formulas are provided below. Let s represent the 320-bit state, with the blocks of the state denoted as $s = s_0s_1s_2s_3s_4$. ($\ggg$) is the symbol for circular right shift.

TABLE III
ASCON 5×5 S-BOX TABLE

Input	0	1	2	3	4	5	6	7
Output	4	11	31	20	26	21	9	2
Input	8	9	10	11	12	13	14	15
Output	27	5	8	18	29	3	6	28
Input	16	17	18	19	20	21	22	23
Output	30	19	7	14	0	13	17	24
Input	24	25	26	27	28	29	30	31
Output	16	12	1	25	22	10	15	23

$$s_0 = s_0 \oplus (s_0 \ggg 19) \oplus (s_0 \ggg 28)$$
$$s_1 = s_1 \oplus (s_1 \ggg 61) \oplus (s_1 \ggg 39)$$
$$s_2 = s_2 \oplus (s_2 \ggg 1) \oplus (s_2 \ggg 6)$$
$$s_3 = s_3 \oplus (s_3 \ggg 10) \oplus (s_3 \ggg 17)$$
$$s_4 = s_4 \oplus (s_4 \ggg 7) \oplus (s_4 \ggg 41)$$

III. PROPOSED ARCHITECTURE

In this section, we present a detailed explanation of the proposed architecture. Figure 2 provides an overview of the top-level model of our design. As shown in the figure, the design have two distinct AXI4-Stream Slave interfaces. The first interface is responsible for handling the transactions of the Initialization Vector (IV), Key, and Nonce. This interface operates with a fixed bus width of 64-bit for both the `ASCON-128` and `ASCON-128a` configurations. Given the algorithm's 320-bit state, the complete set of IV, Key, and Nonce values is received across five successful transactions.

For `ASCON-128`, the permutation stages are defined as $p^a = 12$ and $p^b = 6$, with a greatest common divisor of 6. Accordingly, we have configured the architecture to process 6 rounds per cycle. The initialization and finalization stages each require two-cycle, while the associated data and plaintext/ciphertext processing stages each require one-cycle. This configuration allows the architecture to achieve full performance during the encryption phase, with the algorithm processing 64-bit blocks per cycle. While the ability to encrypt 64-bit in each cycle is advantageous, it should be noted that the critical path in `ASCON-128` is extended due to the 6 consecutive combinational round functions.

For `ASCON-128a`, the permutation stages are defined as $p^a = 12$ and $p^b = 8$, with a greatest common divisor of 4. Consequently, we have selected 4 rounds to be processed per cycle. The initialization and finalization stages each require three-cycle, while the associated data and plaintext/ciphertext processing stages each require two-cycle. In this configuration, the algorithm encrypts/decrypts 128-bit blocks over two-cycle (bit rate). The critical path in `ASCON-128a` is shorter compared to `ASCON-128`, as it involves only 4 consecutive combinational round functions.

A. FSM of `ASCON-128`

The finite state machine of the proposed design for `ASCON-128` is given in Figure 3. The process begins when a specific condition (`temp_ctr = 4`) is met, at which point the IV, key, and nonce are loaded, transitioning the machine into the 'ST_INIT_FIRST' state. This initialization phase consists of two-cycle, progressing from 'ST_INIT_FIRST' to 'ST_INIT_SECOND'. The transition to the 'ST_ASSOC_DATA' state occurs when the `s01_axis_tvalid` signal is active, indicating that the system is ready to receive associated data until the `s01_axis_tlast` signal is asserted.

Next, the machine enters the 'ST_ENCRYPT' state, where the main encryption/decryption process takes place. In each cycle, the algorithm produces a 64-bit output to encrypt the plaintext. In this state, whenever a valid plaintext is received, the corresponding ciphertext is immediately generated. Once the encryption is complete, as indicated by the `s01_axis_tlast` signal, the state machine proceeds to the finalization stages ('ST_FINAL_FIRST' and 'ST_FINAL_SECOND'). This finalization phase also takes of two-cycle to produce 128-bit tag.

B. FSM of `ASCON-128a`

The finite state machine presented in Figure 4 outlines the control flow of the `ASCON-128a` configuration. The state machine begins in the 'ST_IDLE' state, where the process is initialized when a specific condition (`temp_ctr = 4`) is met, leading to a transition into the 'ST_INIT_FIRST' state. The initialization phase consists of three-cycle, progressing from 'ST_INIT_FIRST' to 'ST_INIT_SECOND' and from 'ST_INIT_SECOND' to 'ST_ASSOC_DATA_FIRST'.

Following initialization, the machine enters the 'ST_ASSOC_DATA_FIRST' state when the `s01_axis_tvalid` signal is active, in order to receive associated data. If the `s01_axis_tlast` signal is asserted, the machine transitions to the 'ST_ASSOC_DATA_SECOND' state; otherwise, it remains in the current state. The transition between 'ST_ASSOC_DATA_FIRST' and 'ST_ASSOC_DATA_SECOND' continues until all associated data is received, at which point the `flag` is set to '1', enabling the transition to the encryption phase.

The encryption process begins in the 'ST_ENCRYPT_FIRST' state, where the algorithm processes data to produce encrypted output. When the `s01_axis_tvalid` is active, the machine transitions to the 'ST_ENCRYPT_SECOND' state, which finalizes the encryption of the current block. In total, it takes two-cycle to encrypt one block. The machine continues to process data in these two states until the encryption is complete.

Once encryption is finalized, the machine enters the 'ST_FINAL_FIRST' state, followed by the 'ST_FINAL_SECOND' and 'ST_FINAL_THIRD' states. In total, it takes three-cycle for finalization stage to produce 128-bit tag. Finally, the machine returns to the 'ST_IDLE' state, ready for the next operation.

IV. RESULTS AND COMPARISON

The implementations were conducted using the xc7a200tfbg 676-2 variant of the Artix-7 FPGA, the xc7k160tfbg 676-3

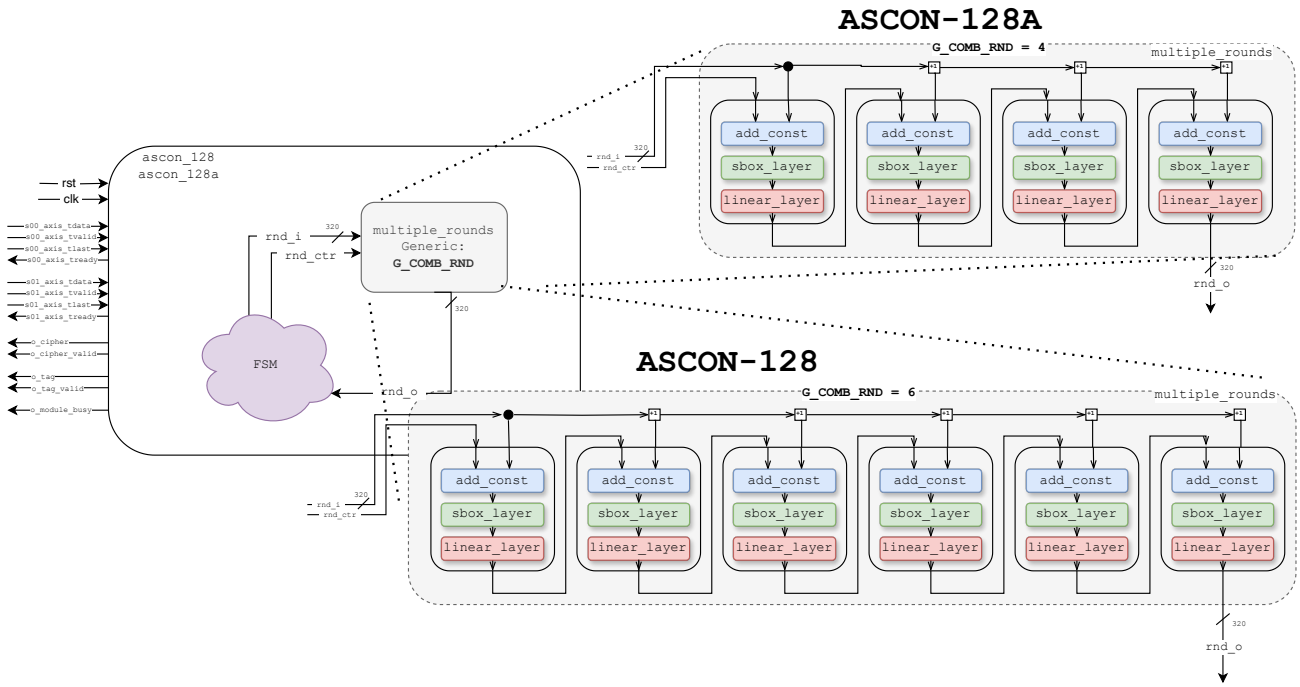

Fig. 2. The Proposed Architecture of ASCON-128 and ASCON-128a.

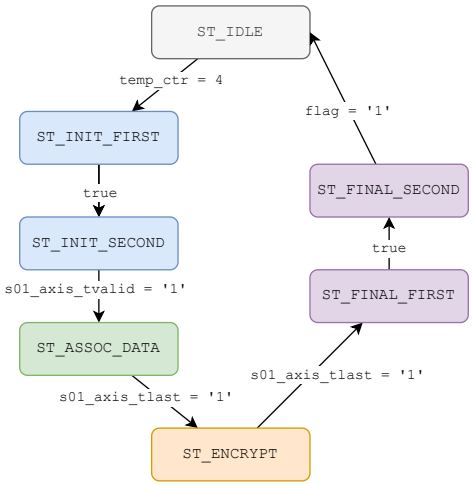

Fig. 3. Finite State Machine of ASCON-128.

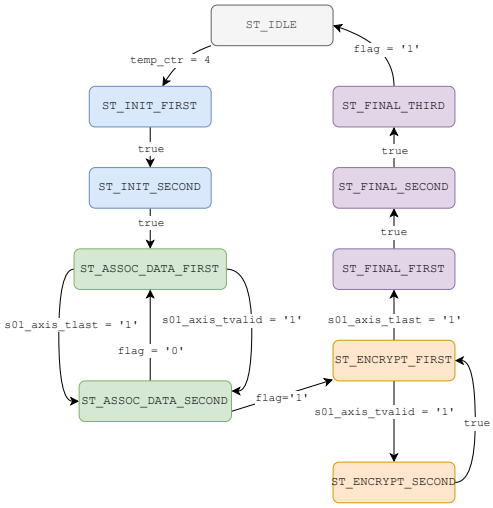

Fig. 4. Finite State Machine of ASCON-128a.

variant of the Kintex-7 FPGA, and the xc7s100fgga 676-2 variant of the Spartan-7 FPGA. The synthesis and implementation processes were carried out using Vivado 2023.2 ML edition. The design was written in VHDL, and the functionality was verified using Python scripts to ensure accuracy and performance.

Throughput is defined as the number of encrypted bits per second, and is calculated using the formula:

$$\text{Throughput} = \frac{\text{Rate}}{\text{Period} \times \text{Clock Cycle}}.$$

In our ASCON-128 design, encryption is performed at a rate of 64 bits per cycle, resulting in a throughput of:

$$\text{Throughput}_{\text{ASCON-128}} = \frac{64}{\text{Period}}.$$

For the ASCON−128a design, encryption occurs at a rate of 128 bits over two cycles, yielding a throughput of:

$$\text{Throughput}_{\text{ASCON-128a}} = \frac{128}{2 \times \text{Period}}.$$

These calculations are essential for accurately computing the performance of the proposed architectures. The cycles used in the initialization and associated data stages increase latency but do not affect throughput. Consequently, the time elapsed during these stages was not considered in our throughput calculations. For ASCON−128, the total latency is expressed as $(2 + x/64)$ cycles, where 2-cycle come from the initialization phase and x represents the number of bits of associated data processed. Similarly, for ASCON−128a, the latency is calculated as $(3 + x/128)$ cycles, with 3 cycles dedicated to initialization and x denoting the associated data bits.

The article [7] provides a comprehensive survey on the NIST Lightweight Cryptography Standard, offering an in-depth analysis of its principles, attacks, and implementations.

A. Artix-7

Table IV provides a detailed comparison of various ASCON implementations on the Artix-7 FPGA, focusing on different architectures and their performance metrics. The comparison covers the area in terms of Look-Up Tables (LUTs), operating frequency, period, throughput, and efficiency, measured as throughput per LUT (Mbps/LUT).

Compared to other designs, the ASCON−128 implementation in this work achieves a throughput of 3535.91 Mbps with an efficiency of 1.19 Mbps/LUT. This is significantly higher than the 6-round implementation by [4], which only achieves 792.56 Mbps with an efficiency of 0.208 Mbps/LUT, and the single-round implementation by [4], which, despite achieving a relatively high throughput of 668.4 Mbps, has a lower efficiency of 0.868 Mbps/LUT.

The ASCON−128a implementation from this work further improves upon these results, achieving the highest throughput of 5333.3 Mbps with an better efficiency of 2.44 Mbps/LUT. This is much better even the most efficient previous design, which achieved 1683.2 Mbps with an efficiency of 0.887 Mbps/LUT [8].

Overall, the implementations presented in this work significantly outperform existing designs, both in terms of raw throughput and efficiency, demonstrating the effectiveness of the proposed optimizations and architectural choices.

B. Kintex-7

Table V presents a comparison of various ASCON implementations on the Kintex-7 FPGA, highlighting key performance metrics such as area (measured in LUTs), operating frequency, period, throughput, and efficiency (measured as Mbps/LUT).

The ASCON−128 implementation from this work, shows a significant improvement in throughput and efficiency compared to previous designs. Specifically, it achieves a throughput of 5925.92 Mbps with an efficiency of 2.01 Mbps/LUT. This performance is markedly better than the 6-round implementation by [4], which reaches a throughput of 1354.49 Mbps with an efficiency of 0.356 Mbps/LUT, and the single-round implementation by [4], which, while achieving 646.46 Mbps in throughput, has a lower efficiency of 0.846 Mbps/LUT.

The ASCON−128a implementation in this work further enhances these results, demonstrating a throughput of 10158 Mbps and an efficiency of 3.32 Mbps/LUT, which is the highest among the implementations compared.

Overall, the table illustrates the important improvements achieved by the ASCON−128 and ASCON−128a implementations in this work, particularly in terms of throughput and efficiency on the Kintex-7 FPGA.

C. Spartan FPGAs

Table VI compares various ASCON implementations on Spartan FPGAs. It's important to note that our implementations are for the more recent Spartan-7 FPGAs, while the other implementations listed are for the older Spartan-6 series. This difference is significant because Spartan-6 FPGAs require an older synthesis environment, Xilinx ISE, which typically results in lower performance compared to designs synthesized for Spartan-7 using modern tools like Xilinx Vivado.

Our ASCON−128 implementation on the Spartan-7 FPGA achieves a throughput of 3440.86 Mbps with an efficiency of 1.16 Mbps/LUT, and the ASCON−128a implementation reaches an impressive 5333.33 Mbps with an efficiency of 2.45 Mbps/LUT. These results represent a improvement over the Spartan-6 implementations. For instance, the best-performing Spartan-6 design listed, using 1913 LUTs, achieves a throughput of 1116.4 Mbps with an efficiency of 0.584 Mbps/LUT, which is significantly lower than the performance metrics achieved on Spartan-7.

The better results obtained in our implementations are expected due to the advancements in FPGA technology and the use of more sophisticated synthesis tools. Spartan-7 FPGAs are designed to deliver higher performance, and the use of Vivado further optimizes the design, resulting in better efficiency and throughput. Therefore, the comparison highlights not just the optimizations in our ASCON implementations but also the natural improvements that come with using a more modern FPGA platform and synthesis environment.

V. Conclusion and Future Work

In this research, we have proposed a design focused on achieving high performance with an optimized throughput-to-area ratio. We implemented a 6-round permutation for ASCON−128 and a 4-round permutation for ASCON−128a to maximize throughput. While our results indicate significant improvements, with 34.16% better efficiency on the Artix-7 FPGA and 137.58% better efficiency on the Kintex-7, it's important to note that the 98.63% better efficiency observed on the Spartan-7 FPGA compared to Spartan-6 results may be partially attributed to the advancements in FPGA technology rather than our design optimizations. The newer Spartan-7 platform offers enhanced performance capabilities, which likely contribute to these improved results.

TABLE IV
HARDWARE IMPLEMENTATIONS OF ASCON ON ARTIX-7 FPGA

ASCON Architecture	FPGA Platform	Area (LUT)	Area (FF)	Frequency (MHz)	Period (ns)	Throughput (Mbps)	Efficiency (Mbps/LUT)
[9]	Artix-7	1808 LUT	N/A	232 MHz	4.31 ns	39 Mbps	0.021
[4],(6-round)	Artix-7	3728 LUT	N/A	64.07 MHz	15.61 ns	792.56 Mbps	0.208
[5]	Artix-7	1330 LUT	N/A	107 MHz	9.34 ns	457 Mbps	0.343
[10]	Artix-7	1723 LUT	N/A	219 MHz	4.57 ns	987 Mbps	0.572
[4],(1-round)	Artix-7	770 LUT	N/A	260.75 MHz	3.84 ns	668.4 Mbps	0.868
[8]	Artix-7	1898 LUT	N/A	263 MHz	3.80 ns	1683.2 Mbps	0.887
This Work- ASCON-128	Artix-7	2957 LUT	595 FF	55.25 MHz	18.1 ns	3535.91 Mbps	1.19
This Work- ASCON-128a	Artix-7	2183 LUT	653 FF	83.33 MHz	12.0 ns	5333.3 Mbps	2.44

TABLE V
HARDWARE IMPLEMENTATIONS OF ASCON ON KINTEX-7 FPGA

ASCON Architecture	FPGA Platform	Area (LUT)	Area (FF)	Frequency (MHz)	Period (ns)	Throughput (Mbps)	Efficiency (Mbps/LUT)
[11], Encryption	Kintex-7	944 LUT	734	181 MHz	5.52 ns	N/A Mbps	N/A
[11], Decryption	Kintex-7	1058 LUT	735	181 MHz	5.52 ns	N/A Mbps	N/A
[12], ASCON-128	Kintex-7	1055 LUT	328	303 MHz	3.30 ns	N/A Mbps	N/A
[12], ASCON-128a	Kintex-7	1107 LUT	328	212 MHz	3.20 ns	N/A Mbps	N/A
[4], (6-round)	Kintex-7	3728 LUT	N/A	100.36 MHz	9.96 ns	1354.49 Mbps	0.356
[4], (1-round)	Kintex-7	702 LUT	N/A	268.24 MHz	3.73 ns	646.46 Mbps	0.846
This Work- ASCON-128	Kintex-7	2958 LUT	595 FF	92.59 MHz	10.8 ns	5925.92 Mbps	2.01
This Work- ASCON-128a	Kintex-7	3051 LUT	653 FF	158.73 MHz	6.3 ns	10158 Mbps	3.32

TABLE VI
HARDWARE IMPLEMENTATIONS OF ASCON ON SPARTAN FPGAS

ASCON Architecture	FPGA Platform	Area (LUT)	Area (FF)	Frequency (MHz)	Period (ns)	Throughput (Mbps)	Efficiency (Mbps/LUT)
[13]	Spartan-6	1640 LUT	N/A	146.1 MHz	6.84 ns	114 Mbps	0.070
[14]	Spartan-6	680 LUT	N/A	216 MHz	4.63 ns	60.1 Mbps	0.088
[15]	Spartan-6	2048 LUT	N/A	N/A MHz	N/A ns	255.4 Mbps	0.1247
[2]	Spartan-6(3-p)	3630 LUT	N/A	126.1 MHz	7.93 ns	448.25 Mbps	0.133
[2]	Spartan-6(2-p)	2720 LUT	N/A	147.2 MHz	6.79 ns	392.61 Mbps	0.144
[2]	Spartan-6(1-p)	2060 LUT	N/A	206.2 MHz	4.85 ns	315.2 Mbps	0.153
[16]	Spartan-6	1985 LUT	N/A	96.89 MHz	10.32 ns	406.66 Mbps	0.235
[8]	Spartan-6	1913 LUT	N/A	174.4 MHz	5.73 ns	1116.4 Mbps	0.584
This Work- ASCON-128	Spartan-7	2957 LUT	595 FF	53.76 MHz	18.6 ns	3440.86 Mbps	1.16
This Work- ASCON-128a	Spartan-7	2178 LUT	653 FF	83.33 MHz	12.0 ns	5333.33 Mbps	2.45

For future work, a more compact implementation could be explored by adopting a RAM-based approach, which would allow for a reduction in the number of LUTs required and thus improve the design in terms of area efficiency by utilizing memory blocks. While we employed 4-round and 6-round permutation methods in our design, investigating the potential of a one-round method could yield even better results in terms of area.

The code used in this research is available at our GitHub repository: https://github.com/AhmetMALAL/ascon-vhdl.

REFERENCES

[1] National Institute of Standards and Technology, "Lightweight cryptography project," 2023. Accessed: 2024-08-28.

[2] S. Khan, W. Lee, and S. O. Hwang, "Scalable and efficient hardware architectures for authenticated encryption in iot applications," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11260–11275, 2021.

[3] S. Khan, W.-K. Lee, and S. O. Hwang, "Evaluating the performance of ascon lightweight authenticated encryption for ai-enabled iot devices," in *2022 TRON Symposium (TRONSHOW)*, pp. 1–6, 2022.

[4] S. Khan, W. 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," *IEEE Internet Things J.*, vol. 10, no. 9, May 1, pp. 8083–8095, 2023.

[5] K. Raj and S. Bodapati, "FPGA based light weight encryption of medical data for iomt devices using ASCON cipher," in *IEEE International Symposium on Smart Electronic Systems, iSES 2022, Warangal, India, December 18-22, 2022*, pp. 196–201, IEEE, 2022.

[6] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021.

[7] J. Kaur, A. C. Canto, M. M. Kermani, and R. Azarderakhsh, "A comprehensive survey on the implementations, attacks, and countermeasures of the current NIST lightweight cryptography standard," *CoRR*, vol. abs/2304.06222, 2023.

[8] B. Rezvani and W. Diehl, "Hardware implementations of NIST lightweight cryptographic candidates: A first look," *IACR Cryptol. ePrint Arch.*, p. 824, 2019.

[9] A. Abdulgadir, W. Diehl, and J. Kaps, "An open-source platform for evaluation of hardware implementations of lightweight authenticated ciphers," in *2019 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2019, Cancun, Mexico, December 9-11, 2019* (D. Andrews, R. Cumplido, C. Feregrino, and M. Platzner, eds.), pp. 1–5, IEEE, 2019.

[10] K. Mohajerani, R. Haeussler, R. Nagpal, F. Farahmand, A. Abdulgadir, J. Kaps, and K. Gaj, "FPGA benchmarking of round 2 candidates in the NIST lightweight cryptography standardization process: Methodology, metrics, tools, and results," *IACR Cryptol. ePrint Arch.*, p. 1207, 2020.

[11] A. Kandi, A. Baksi, T. Gerlich, S. Guilley, P. Gan, J. Breier, A. Chattopadhyay, R. R. Shrivastwa, Z. Martinásek, and S. Bhasin, "Hardware implementation of ascon," in *NIST LWC Workshop*, 2023.

[12] K. Mandal, D. Saha, S. Sarkar, and Y. Todo, "Sycon: a new milestone in designing ascon-like permutations," *J. Cryptogr. Eng.*, vol. 12, no. 3, pp. 305–327, 2022.

[13] W. Diehl, F. Farahmand, A. Abdulgadir, J. Kaps, and K. Gaj, "Face-off between the CAESAR lightweight finalists: ACORN vs. ascon," *IACR Cryptol. ePrint Arch.*, p. 184, 2019.

[14] P. Yalla and J. Kaps, "Evaluation of the CAESAR hardware API for lightweight implementations," in *International Conference on ReCon-*

Figurable Computing and FPGAs, ReConFig 2017, Cancun, Mexico, December 4-6, 2017, pp. 1–6, IEEE, 2017.

[15] W. Diehl, A. Abdulgadir, F. Farahmand, J. Kaps, and K. Gaj, "Comparison of cost of protection against differential power analysis of selected authenticated ciphers," *Cryptogr.*, vol. 2, no. 3, p. 26, 2018.

[16] A. Koppuravuri, H. Pasupuleti, S. Gvk, and J. Bapat, "A high throughput ASCON architecture for secure edge iot devices," in *37th International Conference on VLSI Design and 23rd International Conference on Embedded Systems, VLSID 2024, Kolkata, India, January 6-10, 2024*, pp. 486–491, IEEE, 2024.