Efficient Implementations of CRAFT Cipher For Internet of Things

Abstract

The rapid growth of the Internet of Things (IoT) highlights the importance of lightweight cryptography in maintaining security. However, enhancing performance while ensuring the same level of security remains a significant challenge. This paper presents two innovative architectures for the CRAFT lightweight block cipher, aiming to enhance performance without compromising security. The novel Serial and Unrolled architectures are introduced to achieve low area usage and high throughput, respectively. Specifically, the Serial architecture reduces the datapath from 64-bit to 4-bit, significantly decreasing the area. The Unrolled architecture, on the other hand, minimizes latency from 32 to 16. Additionally, Boolean satisfiability (SAT) solvers are employed to identify a lower-cost area implementation of the S-Box. The proposed designs underwent evaluation on three distinct FPGA platforms: Artix-7, Kintex-7, and Spartan-7. The results suggest that the low area design reduces area usage by 15.82% compared to the PRESENT cipher. On the other hand, the unrolled design doubles the throughput rate at 100MHz and significantly reduces energy consumption per bit by 47.89% compared to the PRESENT cipher. To the best of our knowledge, the low area design sets a new area record on the FPGA configured with a 64-bit block size and 128-bit key size. Therefore, the proposed designs could offer enhanced performance while maintaining security for IoT devices.

Keywords: Internet of Things, Lightweight block cipher, Field Programmable Gate Arrays(FPGA), Low-area, High-throughput

1. Introduction

The Internet of Things (IoT) is rapidly integrating into various aspects of everyday life. With this advancement, a growing number of security issues are emerging. These security concerns are extensively discussed in [1]. To ensure data protection, the cryptography techniques outlined in [2] are recommended.

However, the resource constraints of many IoT devices pose challenges for the implementation of robust security measures. These devices often have limited memory, processing power, and energy. Therefore, the security measures need to be lightweight to ensure they do not overburden the resources. Lightweight cryptography, a subset of cryptography, provides solutions specifically designed for these resource-limited devices, as discussed in [3].

The field of lightweight cryptography has received considerable attention in recent years. Examples of this include PRESENT [4], LED [5], Midori [6], QTL [7], GIFT [8], CRAFT [9], Shadow [10], DULBC [11], IVLBC [12], BipBip [13], and LELBC [14]. More ciphers can be found in [15]. The implementation of lightweight ciphers for various applications has also been widely studied.

Efficient implementation allows lightweight ciphers to be used in various settings. A hardware implementation can enhance the performance of these ciphers in resource-limited environments. Several researchers have proposed different architectures and optimizations for various ciphers. Lara-Nino et al. [16] introduced a 16-bit datapath architecture for the PRESENT cipher, reducing both the area and power consumption. Pandey et al. [17] suggested an optimized key schedule of PRESENT, which resulted in a smaller area. Shahbazi et al. [18] proposed an 8-bit serial architecture for AES, also reducing the area and power consumption. Li et al. [19] presented unrolled architectures and a low-cost architecture for PRINCE, separately optimizing the throughput and area. Bharathi et al. [20] enhanced the performance of the PRESENT cipher by expanding the key length. Lastly, Yang et al. [21] shared components in the cipher process for LILLIPUT, resulting in a smaller area.

This work presents the first implementation of CRAFT on FPGA platforms. Two architectures for CRAFT, Serial and Unrolled, are proposed. The Serial architecture reduces the datapath from 64-bit to 4-bit, meaning it only uses one S-Box, which significantly reduces the area usage. The Unrolled architecture reduces the latency of the encryption process, thereby improving the throughput rate. The optimal implementation of the S-Box, aimed at further area reduction, is determined using a SAT solver in conjunction with the GEC encoding scheme. The experiments are conducted on three different FPGA platforms: Artix-7, Kintex-7, and Spartan-7. The source code for the proposed designs is available online.[1] The main contributions of this article are as follows.

- Two architectures for CRAFT, Serial and Unrolled, are proposed. These are optimized for area and throughput, respectively. The Serial architecture reduces the area usage by 15.72% compared to the work of Bharathi et al. [20]. The Unrolled architecture doubles the throughput rate compared to the same work.
- The optimal implementation of the S-Box, which results in further area reduction, is identified using a SAT solver.

[1] `https://github.com/xjh2000/craft_implementation`

The proposed S-Box implementation achieves a 28.9% area reduction compared to the work of Bao et al. [22].

- The architectures are implemented across three different FPGA platforms: Artix-7, Kintex-7, and Spartan-7. This variety allows engineers to select the platform that best suits their application needs.

The remainder of this article unfolds as follows: Section 2 delves into the specifics of CRAFT. The duo of proposed architectures for CRAFT are explored in Section 3. Section 4 lays out the metrics and environment used for experimental evaluation. An in-depth performance analysis of all the architectures is presented in Section 5. Lastly, Section 6 encapsulates the work done and points towards potential avenues for future research.

2. Specification of CRAFT

CRAFT is a lightweight tweakable block cipher that operates on a 64-bit plaintext size, a 128-bit key size, and a 64-bit tweak size. It outputs a 64-bit ciphertext. The confusion and diffusion properties of CRAFT ensure that the distribution of probabilities between the plaintext and ciphertext is independent. For more details on CRAFT, refer to Figure 1 which depicts its architecture. The encryption process of CRAFT is outlined in Algorithm 1. The decryption process is similar to the encryption process, with the only difference being that the round keys are applied in reverse order. The main notations used throughout this paper are outlined in Table 1.

Table 1: Main Notations

Notation	Description
TK_i	tweakeys used in the i^{th} round
RC_i	round constant for the i^{th} round
R_i	Function for the i^{th} round
SB	Sub-Box
MC	Mix-Columns
PN	PermuteNibbles
PK	Permutation used in key schedule
$\oplus$	XOR operation
$\|\|$	Concatenation operation
$\sim$	Inverse operation
$\wedge$	And operation
$\vee$	Or operation

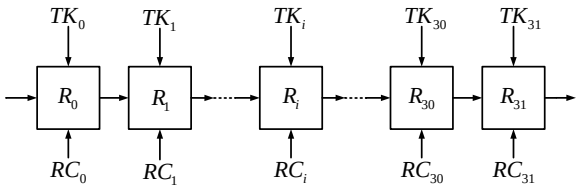

Figure 1: Architecture of CRAFT

Algorithm 1 CRAFT Encryption Process

Input: Plaintext P, Key $K_0||K_1$, Tweak T
Output: Ciphertext C

$TK_0 \leftarrow K_0 \oplus T$
$TK_1 \leftarrow K_1 \oplus T$
$TK_2 \leftarrow K_0 \oplus PK(T)$
$TK_3 \leftarrow K_1 \oplus PK(T)$
$C \leftarrow P$
for $i \leftarrow 0$ to 31 **do**
  $C \leftarrow MC(C)$
  $C_{4,5} \leftarrow C_{4,5} \oplus RC_i$
  $C \leftarrow C \oplus TK_{i \bmod 4}$
  if $i \neq 31$ **then**
    $C \leftarrow PN(C)$
    $C \leftarrow SB(C)$
  end if
end for

The round function is composed of three distinct operations: Mix-Columns, PermuteNibbles, and Sub-Box. The Mix-Columns operation is a linear transformation that multiplies the input column by a constant matrix, M, to generate the output column. Notably, M is an involutory matrix, as shown in Equation (1).

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \quad (1)$$

The PermuteNibbles operation, an involutory permutation, operates on 4-bit nibbles. This operation triggers additional S-Boxes, thereby bolstering the cipher's security. The illustration of the PermuteNibbles operation is provided in Equation (2). The permutation PK is utilized in the key schedule, as depicted in Equation (3).

$$PN = [15, 12, 13, 14, 10, 9, 8, 11, 6, 5, 4, 7, 1, 2, 3, 0] \quad (2)$$

$$PK = [12, 10, 15, 5, 14, 8, 9, 2, 11, 3, 7, 4, 6, 0, 1, 13] \quad (3)$$

The Sub-Box operation, a nonlinear transformation, introduces confusion into the cipher. This operation is performed using a 4-bit S-Box. The values are represented in hexadecimal notation, as shown in Table 2.

Two Linear Shift Feedback Registers (LSFRs), a and b, are used to concatenate the round constants. The round constants is defined as $RC = (a_3, a_2, a_1, a_0, 0, b_2, b_1, b_0)$. The initial round constant, RC_0, is set to 0x11.

3. Implementations

For the first time, the components of CRAFT have been optimized to achieve efficient area and throughput, resulting in two proposed implementation architectures: Serial and Unrolled.

Table 2: S-Box of CRAFT

Input	Output	Input	Output
0	c	8	8
1	a	9	9
2	d	a	1
3	3	b	5
4	e	c	0
5	b	d	2
6	f	e	4
7	7	f	6

3.1. Serial Architecture (SA)

Compared to round-based architectures, serial architectures can significantly reduce area usage by reusing components. For example, the quantity of S-Boxes is diminished from 16 to 1. The clock gating technique is also employed to enable each component and minimize the energy consumption of encryption.

The architecture, depicted in Figure 2, comprises a single Sub-Box, a 4-bit Mix-columns, and two register banks. The Key-Register is used to store keys. The State-Register, on the other hand, is used to store plaintext. They also hold intermediate results temporarily. The design incorporates a feedback path to store intermediate results in the State-Register bank. Additionally, the PermuteNibbles function is integrated into the State-Register bank.

3.1.1. S-Box Optimization

The S-Box is a crucial component of the encryption algorithm, adding to its complexity. However, it also demands a significant amount of area. There are several ways to implement the S-Box. One prevalent approach is to use a lookup table (LUT), a technique described by Lara-Nino et al. [16]. This approach, while effective, requires many flip-flops, which can lead to a substantial increase in area consumption. An alternative method is to use the logical equivalent expression of the S-Box. This method, suggested by Bao et al. [22] and Feng et al. [23], can help reduce area consumption.

Boolean satisfiability (SAT) solvers can be used to find S-Boxes that meet specific implementation requirements, such as meet to certain software or hardware implementation requirements. To elaborate, the S-Box implementation can be encoded as Boolean constraints. This is done by representing the S-Box as a truth table and using Boolean variables to denote the input and output bits of the S-Box. The constraints are then formulated based on the desired properties of the S-Box. Once the S-Box properties are encoded as Boolean constraints, these constraints can be satisfied by a SAT solver. This assignment corresponds to an S-Box that fulfills the desired properties.

A measure of the number of logical gates required to implement the Boolean formula that represents a SAT instance is the Gate Equivalent Complexity (GEC). To calculate the GEC, the Boolean formula is converted into a circuit of logical gates, such as AND, OR, and NOT gates. The total count of these gates in the circuit gives the GEC of the instance. In this design, the GEC encoding scheme from Feng et al. [23] is optimized and used to implement the S-Box. The encoding scheme is detailed in Equations (4):

$$
\begin{aligned}
&\forall i \in \{0, 1, \ldots, K-1\}: \\
&T_i = F_{if}(GT_i[0], \sim (X_{4i} \cdot X_{4i+1}) \cdot \sim X_{4i+2} \cdot X_{4i+3}) \\
&\quad + F_{if}(GT_i[1], X_{4i+2} \cdot (X_{4i} + X_{4i+1})) \\
&\quad + F_{if}(GT_i[2], X_{4i} \cdot X_{4i+1} \cdot X_{4i+2}) \\
&\quad + F_{if}(GT_i[3], X_{4i+2}) + F_{if}(GT_i[4], X_{4i}) \\
&\quad + F_{if}(GT_i[5], X_{4i} \cdot X_{4i+1}) \\
&\quad + F_{if}(GT_i[6], X_{4i} + X_{4i+1}) + F_{if}(GT_i[7], max)
\end{aligned}
\tag{4}
$$

where K is numbers of the logical gates, $X_{4i} - X_{4i+3}$ is the input of the i^{th} logical gate, T_i is the output of the i^{th} logical gate, and F_{if} is a function that returns the value of the second argument if the first argument is true and returns the value of zero otherwise. The value of max is all one's in the binary expression, which is represented logically as an inverse. GT_i denotes the type of the i^{th} logical gate, represented as an 8-bit binary number. The least significant bit of GT_i is indexed at seven. Table 3 enumerates the various types of logical gates employed in this encoding scheme.

Equations (5) display the optimized scheme of the S-Box, where $X_3 - X_0$ represents the input and $Y_3 - Y_0$ represents the output. The proposed S-Box scheme is implemented using four MOAI1 gates, three MAOI1 gates, and one AND3 gate. This configuration of the S-Box module results in a 28.9% reduction in area compared to the method proposed by Bao et al. [22], based on gate equivalent (GE) estimation using the UMC 180nm library.

$$
\begin{aligned}
T_0 &= \text{MAOI1}(X_0, X_1, X_0, X_1) \\
T_1 &= \text{AND3}(X_3, X_2, X_3) \\
T_2 &= \text{MAOI1}(X_1, X_2, X_0, X_3) \\
T_3 &= \text{MOAI1}(X_1, X_0, X_2, X_2) \\
T_4 &= \text{MOAI1}(X_3, T_0, T_3, T_3) \\
T_5 &= \text{MOAI1}(T_3, T_0, X_0, T_1) \\
T_6 &= \text{MAOI1}(X_0, T_0, X_3, T_0) \\
T_7 &= \text{MOAI1}(X_0, T_1, T_2, T_2) \\
Y_3 &= T_4 \quad Y_2 = T_6 \quad Y_1 = T_7 \quad Y_0 = T_5
\end{aligned}
\tag{5}
$$

3.1.2. Mix-Columns Optimization

The Mix-Columns component is a linear transformation of the input column. The output column is generated by multiplying the input column with a constant matrix M. M is a involutory matrix, which means $M^2 = E$, where E is the identity matrix. It is easy to decrypt the ciphertext by multiplying the ciphertext with M again. Equation (6) illustrates the Mix-columns component. Here, $I_{3,j}$, $I_{2,j}$, $I_{1,j}$, and $I_{0,j}$ represent the

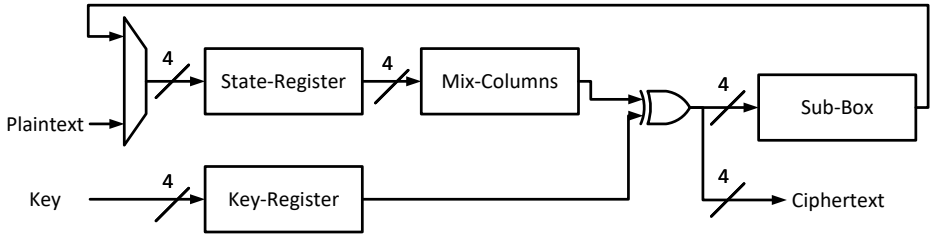

Figure 2: Serial architecture of CRAFT

Table 3: Encoding of different types of logical gates

logical expression	GT_i	gate type
$X_0 \oplus X_1$	2	XOR
$\sim (X_0 \oplus X_1)$	3	XNOR
$X_0 \wedge X_1$	4	AND
$\sim (X_0 \wedge X_1)$	5	NAND
$X_0 \vee X_1$	6	OR
$\sim (X_0 \vee X_1)$	7	NOR
$\sim X_0$	9	NOT
$\sim X_1$	11	NOT
$\sim X_2$	17	NOT
$X_0 \oplus X_1 \oplus X_2$	18	XOR3
$\sim (X_0 \oplus X_1 \oplus X_2)$	19	XNOR3
$X_0 \wedge X_1 \wedge X_2$	32	AND3
$\sim (X_0 \wedge X_1 \wedge X_2)$	33	NAND3
$X_0 \vee X_1 \vee X_2$	118	OR3
$\sim (X_0 \vee X_1 \vee X_2)$	119	NOR3
$\sim ((X_0 \wedge X_1) \vee (\sim (X_2 \vee X_3)))$	176	MAOI1
$\sim (\sim (X_0 \wedge X_1) \wedge ((X_2 \vee X_3)))$	177	MOAI1

input column, while $I'_{3,j}$, $I'_{2,j}$, $I'_{1,j}$, and $I'_{0,j}$ denote the output column. The column index is given by j, where j ranges from 0 to 3.

$$\begin{bmatrix} I'_{3,j} \\ I'_{2,j} \\ I'_{1,j} \\ I'_{0,j} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_{3,j} \\ I_{2,j} \\ I_{1,j} \\ I_{0,j} \end{bmatrix} \tag{6}$$

In order to reduce the area of this component, the serial architecture of Mix-Columns is utilized, as shown in Figure 3. The serial architecture of Mix-Columns requires four 4-bit registers, two multiplexers, and three XOR gates. The operation of Mix-Columns involves three distinct stages: freeze, shift, and add. During the freeze stage, the register values are kept unchanged by setting both CM_0 and CM_1 to 0. In the shift stage, a shift in the register values from RM_0 to RM_4 is induced by setting both CM_0 and CM_1 to 1. Finally, in the add stage, an addition operation on the column values is executed according to Equation (6). This is achieved by setting CM_0 and CM_1 to 0 and 1, respectively.

Figure 4 presents the timing diagram for the serial architecture of the Mix-Columns operation. It requires five clock cycles to compute the next columns from the previous ones, and an additional four clock cycles to transfer data from the internal register of Mix-Columns to the State-Register. Therefore, a complete state round requires a total of 36 clock cycles.

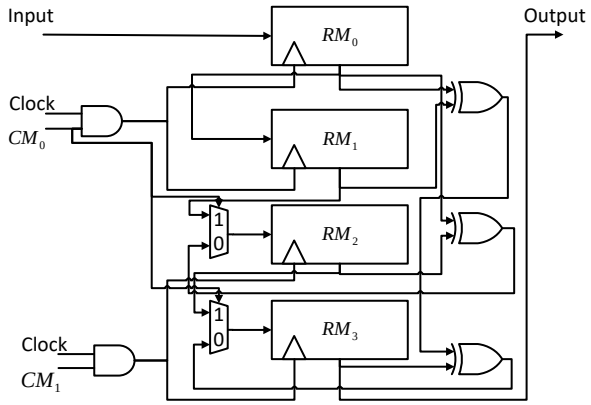

Figure 3: Serial Architecture of Mix-Columns with clock gating

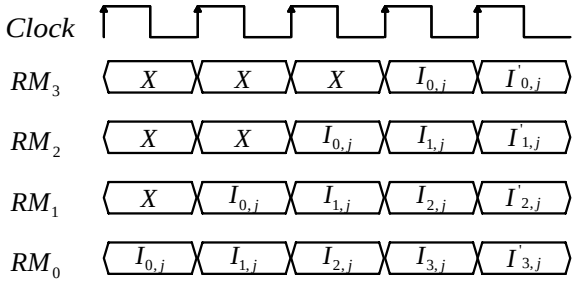

Figure 4: Timing Diagram for the Serial Architecture of Mix-Columns

3.1.3. Control Units

As depicted in Figure 5, the finite-state machine (FSM) initiates the encryption process by loading the initial key into the Key-Register and the plaintext into the State-Register. During the Key-Schedule phase, the key is expanded while the gate

clocks of the Mix-Columns and State-Register are turned off. Next, the Mix-Columns phase begins, where one column of the State-Register is stored in the Mix-Columns registers. The Mix-Columns operation on one column takes five clock cycles to execute in this phase. Once this phase is finished, the gate clocks for the State-Register and Key-Register are turned off. Following this, the Sub-Box phase commences. During this phase, the data from the Mix-Columns registers is transferred again to the State-Register and XORed with the keys. This process requires an additional four clock cycles. This cycle between the Mix-Columns and Sub-Box phases is repeated four times for the four columns of the State-Register. Subsequently, the Permute operation is carried out within the State-Register, requiring a single clock cycle. The encryption process concludes when the Round counter reaches 31, at which point the ciphertext is stored in the State-Register.

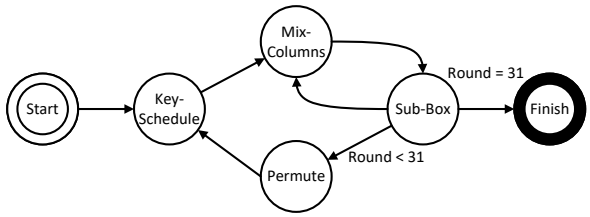

Figure 5: Finite-state machine for Serial Architecture

In the work of Shahbazi et al. [18], it is discussed that the dynamic power consumption of the encryption process can be mitigated through the use of clock gating. The clock gating technique is independently applied to the State-Register, Key-Register, and Mix-Columns. For instance, in the Permute phase, the gate clocks of the Key-Register and Mix-Columns are disabled as these components are not in use. This helps save a significant amount of power. Figure 6 shows the timing diagram of a design that uses the clock gating technique.

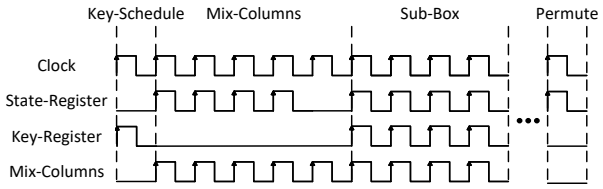

Figure 6: Timing diagram for Serial Architecture

3.2. Unrolled Architecture (UA)

The unrolled architecture shown in Figure 7 includes two Sub-Boxes, two Mix-Columns, a Key, a State-Register, two PermuteNibbles, and one feedback path. It's designed to perform a 30-round encryption process in just 15 cycles. Only the Mix-Columns and Add-Key operations are carried out in the final cycle, finishing the encryption process in a total of 16 cycles.

The unrolled architecture, which is based on the iterative architecture from the work of Beierle et al. [9], completes the encryption process in only 16 cycles, compared to the 32 cycles needed by the iterative architecture. Here, a cycle includes two round functions of CRAFT. While this approach might use more area, it provides higher throughput at the same frequency.

3.3. Iterative Architecture (IA)

The architecture proposed by Bharathi et al. [20] operates on a round-based architecture. It employs a single round function to encrypt a block, incorporating a Sub-Box, a Permute, and an Add-Key operation. This round function is executed 32 times to encrypt a single block. Simultaneously, the Key Schedule operates in parallel with the round function. This architecture is illustrated in Figure 8. For comparison, different architectures are listed in Table 4.

Table 4: Description of Different Architectures

Architecture	Cipher	Description	Reference
SA	CRAFT	Serial	This work
UA	CRAFT	Unrolled	This work
IA	PRESENT	Iterative	[20]

4. Experimental Evaluation

In ASIC implementations, the Gate Equivalent (GE) is often used to evaluate the area consumption of a design. One GE corresponds to the area of a two-input NAND gate. The area is computed in terms of GEs. This is done by dividing the total area (measured in μm^2) by the area of a two-input NAND gate (also measured in μm^2). However, the number of GEs can vary depending on the specific technology used, as Turan et al. discuss in their work [24]. For instance, the number of GEs for the same design will differ between UMC 180nm technology and TSMC 180nm technology. Therefore, GE is not suitable for comparing the area consumption of different designs on different technologies. For a fair comparison, the area consumption of the proposed designs is assessed using FPGA implementations. This is a technique similarly employed in the study by Mohajerani et al. [3].

4.1. Platform

The designs proposed in this study were implemented on a Xilinx FPGA board, utilizing the Vivado v2023.2 software for deployment. Benchmarking was performed across three distinct FPGA platforms to ensure a diverse testing environment: Artix-7(xc7a100tcsg324-1), Kintex-7(xc7k70tfbg484-1), and Spartan-7(xc7s100fgga484-1). Artix-7 offers high performance in resource-limited situations. Spartan-7 is designed for high-restriction environments. Kintex-7 is well-suited for use in applications such as 3G and 4G wireless technologies.

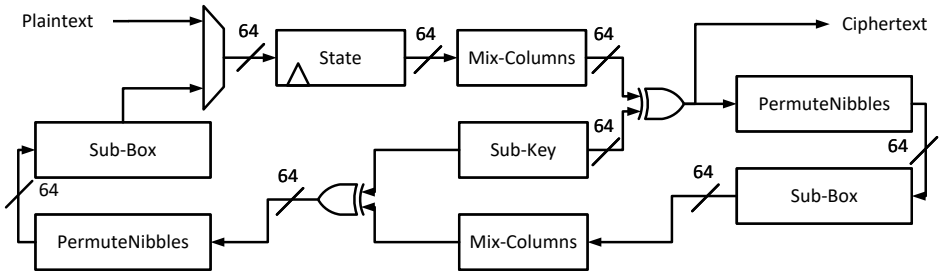

Figure 7: Unrolled architecture of CRAFT

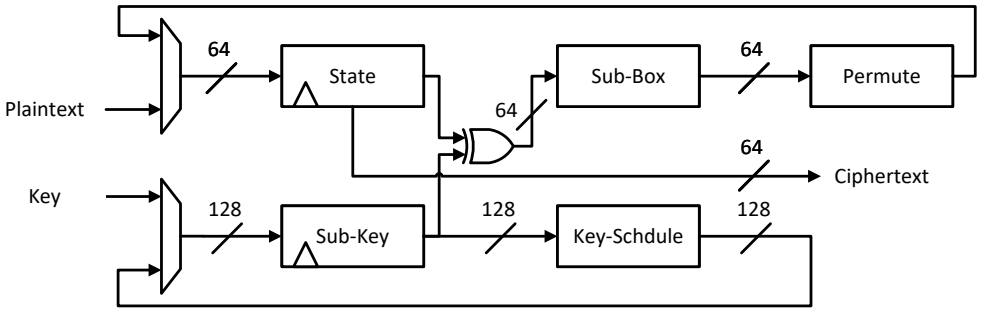

Figure 8: Iterative architecture of PRESENT

4.2. Area

The area consumed by the proposed designs is quantified using the Area metric, which encompasses components such as Flip-Flops, LUTs, and Slices. For a balanced comparison, the embedded memory blocks of the FPGA were not utilized. This was achieved by disabling the relevant settings in the VHDL, as recommended in the design guidelines provided by Xilinx [25]. Also, all designs were synthesized and implemented using the same settings, specifically, the default settings of Vivado Synthesis and Implementation.

4.3. Throughput

The efficiency of the proposed designs is assessed using the Throughput metric. This metric uses three parameters: the maximum throughput rate, the throughput rate at 100MHz, and the throughput rate per slice. The maximum throughput rate is the highest rate that our designs can achieve, calculated using Equation (7). The throughput rate at 100MHz shows the rate achievable when the clock frequency is set to 100MHz, calculated using Equation (8). The throughput rate per slice is a measure of efficiency, calculated by dividing the throughput rate by the number of Slices, as defined in Equation (9). In these computations, the Plaintext Size is set to 64-bit. Latency denotes the count of clock cycles needed to encrypt a single block, and Slices represent the quantity of Slices consumed by the design.

$$MaxThroughput(Thr) = \frac{MaxFrequency \times BlockSize}{Latency} \quad (7)$$

$$Throughput_{@100MHz}(Thr^*) = \frac{100MHz \times BlockSize}{Latency} \quad (8)$$

$$ThroughputPerSlice = \frac{Thr}{Slices} \quad (9)$$

4.4. Power and Energy

The Power metric, which includes both dynamic and static power consumption, is used to evaluate the power consumption of the proposed designs, as defined in Equation (10). On the other hand, the Energy metric measures the energy consumption of the designs. It's calculated by multiplying the power consumption by the time needed to encrypt a single block. This time is determined by dividing the latency by the frequency, as explained in Equation (11).

$$\begin{aligned} \text{Total Power}(TP) &= \text{Dynamic Power}(DP) \\ &\quad + \text{Static Power}(SP) \end{aligned} \quad (10)$$

$$Energy(E) = \frac{TP \times Latency}{Frequency} \quad (11)$$

5. Results

This section details the outcomes of the proposed designs, categorized into three aspects: area consumption, throughput performance, and power and energy metrics. The results are

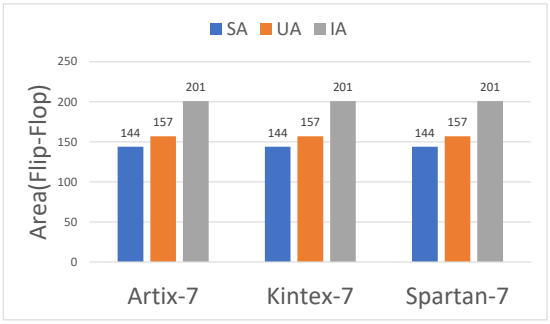

Figure 9: Comparison of Flip-Flop in Three Architectures

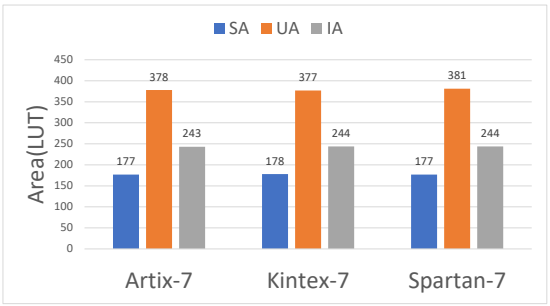

Figure 10: Comparison of Look-Up Tables in Three Architectures

demonstrated across three different FPGA platforms: Artix-7, Kintex-7, and Spartan-7. The area consumption of the designs is presented in Table 5. The throughput results can be found in Table 6, and the details of power and energy consumption are provided in Table 7.

The area consumption of the proposed designs is evaluated based on three factors: Flip-Flops (FF), Look-Up Tables (LUT), and Slices. These designs are compared with the iterative architecture of PRESENT, as detailed in the work of Bharathi et al. [20]. The results indicate that the proposed serial architecture consume 15.72% less area than the iterative architecture of PRESENT.

Regarding the FF, the key schedule of the CRAFT cipher is implemented using multiplexers. This eliminates the need for FF to store the sub-key, resulting in a lower FF count compared to other ciphers. This is a significant factor contributing to the CRAFT cipher's requirement of less than 1000 GE, which is the lowest known requirement on the IBM 130 nm ASIC library, as demonstrated in the study by Beierle et al. [9]. A comparison of FF counts is provided in Figure 9.

As illustrated in Figure 10, when it comes to LUT, the proposed serial architecture require fewer LUTs than the iterative architecture of PRESENT. This is attributed to the fact that the proposed designs utilize a single S-Box, in contrast to the 16 S-Boxes used by the iterative architecture of PRESENT. Furthermore, the proposed designs also require fewer LUTs than the unrolled architecture of CRAFT, which uses 32 S-Boxes, compared to just one in the proposed designs.

Thanks to the reduction in Flip-Flop (FF) and Look-Up Table (LUT) usage, the serial architecture has fewer slices compared to the iterative architecture of PRESENT. In terms of Slices efficiency, Spartan-7 outperforms both Artix-7 and Kintex-7 platforms. These results are illustrated in Figure 11. However, the lower Max Frequency of Spartan-7 will be considered in the Throughput comparison.

Figure 12 illustrates that the proposed serial architecture have a higher Max Frequency than the iterative architecture of PRESENT. This improvement is due to two key factors. First,

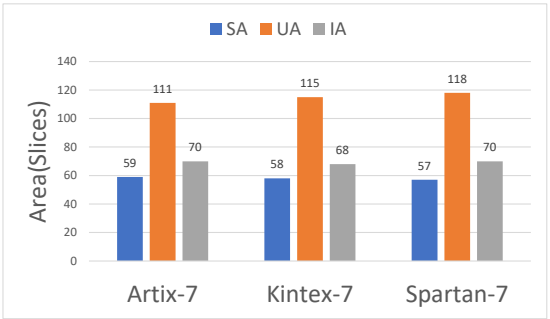

Figure 11: Comparison of Slices in Three Architectures

Table 5: Area Used in Three Architectures

Platform	Design	*State*(*bit*)	*Key*(*bit*)	*FF*	*LUT*	*Slices*
Artix-7	SA	64	128	**144**	**177**	**59**
	UA	64	128	157	378	111
	IA	64	128	201	243	70
Kintex-7	SA	64	128	**144**	**178**	**58**
	UA	64	128	157	377	115
	IA	64	128	201	244	68
Spartan-7	SA	64	128	**144**	**177**	**57**
	UA	64	128	157	381	118
	IA	64	128	201	244	70

Table 6: Throughput Results in Three Architectures

Platform	Design	*Latency*	*MaxF*(*MHz*)	*Thr*(*Mbps*)	*Thr**(*Mbps*)[a]	*Thr*/*Slices*(*Kbps*/*Slices*)
Artix-7	SA	1215	**557.41**	29.36	5.27	497.65
	UA	**16**	142.38	569.52	**400.00**	5130.81
	IA	32	274.04	548.08	200.00	7829.71
Kintex-7	SA	1215	**853.97**	44.98	5.27	775.57
	UA	**16**	175.25	701.00	**400.00**	6095.65
	IA	32	357.78	715.56	200.00	10522.94
Spartan-7	SA	1215	**525.76**	27.69	5.27	485.87
	UA	**16**	138.86	555.44	**400.00**	4707.12
	IA	32	296.29	592.58	200.00	8465.43

[a] Throughput rate at 100MHz

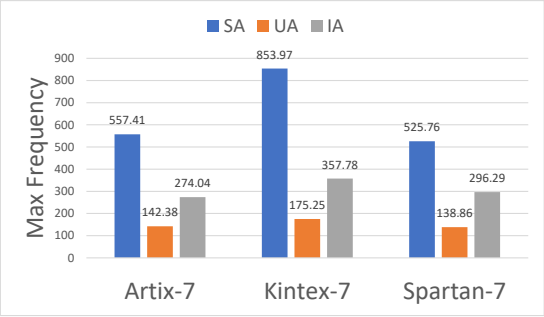

Figure 12: Comparison of Max Frequency in Three Architectures

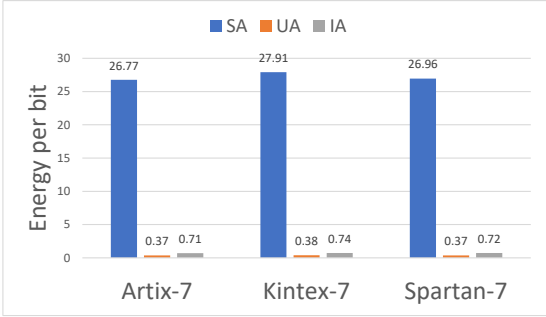

Figure 13: Comparison of Energy Per Bit in Three Architectures

the S-Box is optimized with the GEC encoding scheme, reducing its delay. Second, the serial architecture of the design further reduces the overall delay of the encryption process. However, among all platforms, Spartan-7 has the lowest Max Frequency, primarily because it has the fewest LUTs, as shown in Figure 10. The unrolled architecture reduces the latency to 16, which doubles the Throughput rate at 100MHz compared to the iterative architecture of PRESENT. This data is presented in Table 6.

The serial architecture has the highest energy per bit due to its higher latency, which results in the smallest area. Conversely, the unrolled architecture has the lowest energy per bit because it has the lowest latency. The energy per bit of the iterative architecture of PRESENT falls between that of the serial architecture and the unrolled architecture. Compared to the iterative architecture of PRESENT, the unrolled architecture reduces energy per bit by 47.89%. Figure 13 illustrates the energy per bit for the three architectures.

Table 7: Power and Energy Consumption in Three Architectures

Platform	Design	$DP(mW)$	$SP(mW)$	$TP(mW)$	$E(uJ)$	$E/bit(nJ/bit)$
Artix-7	SA	2.00	139.00	141.00	1.71	26.77
	UA	7.00	139.00	146.00	**0.02**	**0.37**
	IA	2.00	139.00	141.00	0.05	0.71
Kintex-7	SA	2.00	145.00	147.00	1.79	27.91
	UA	8.00	145.00	153.00	**0.02**	**0.38**
	IA	3.00	145.00	148.00	0.05	0.74
Spartan-7	SA	2.00	140.00	142.00	1.73	26.96
	UA	7.00	140.00	147.00	**0.02**	**0.37**
	IA	3.00	140.00	143.00	0.05	0.72

DP: Dynamic Power SP: Static Power TP: Total Power E: Energy

6. Conclusion

Given the diverse performance requirements arising from the use of IoT devices in various contexts, achieving optimal security without compromising performance presents a significant challenge. Implementing effective solutions is one way to attain this balance between security and performance.

This research presents two unique designs for the CRAFT Lightweight cipher - Serial and Unrolled - both aimed at boosting performance. The Serial architecture reduces the area consumption by 15.72% compared to the iterative architecture of PRESENT. The Unrolled architecture, on the other hand, reduces the latency to 16, effectively doubling the throughput rate at 100MHz compared to the iterative architecture of PRESENT. Additionally, the Unrolled architecture reduces energy per bit by 47.89% compared to the iterative architecture of PRESENT. To the best of our knowledge, the Serial architecture establishes a new record for area efficiency on an FPGA configured with a 64-bit block size and 128-bit key size. These proposed architectures are therefore highly suitable for environments with IoT devices.

Future work could involve investigating the application of these proposed architectures in real-world IoT devices and measuring their performance. Additionally, implementing these architectures on ASICs could further reduce area consumption.

References

[1] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, A. Zanella, Iot: Internet of threats? A survey of practical security vulnerabilities in real iot devices, IEEE Internet Things J. 6 (5) (2019) 8182–8201. doi:10.1109/JIOT.2019.2935189.

[2] D. Swessi, H. Idoudi, A survey on internet-of-things security: Threats and emerging countermeasures, Wirel. Pers. Commun. 124 (2) (2022) 1557–1592. doi:10.1007/S11277-021-09420-0.

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

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: an ultra-lightweight block cipher, in: P. Paillier, I. Verbauwhede (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings, Vol. 4727 of Lecture Notes in Computer Science, Springer, 2007, pp. 450–466. doi:10.1007/978-3-540-74735-2_31.

[5] J. Guo, T. Peyrin, A. Poschmann, M. J. B. Robshaw, The LED block cipher, in: B. Preneel, T. Takagi (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings, Vol. 6917 of Lecture Notes in Computer Science, Springer, 2011, pp. 326–341. doi:10.1007/978-3-642-23951-9_22.

[6] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, F. Regazzoni, Midori: A block cipher for low energy, in: T. Iwata, J. H. Cheon (Eds.), Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II, Vol. 9453 of Lecture Notes in Computer Science, Springer, 2015, pp. 411–436. doi:10.1007/978-3-662-48800-3_17.

[7] L. Li, B. Liu, H. Wang, Qtl: A new ultra-lightweight block cipher, Microprocessors and Microsystems 45 (2016) 45–55. doi:10.1016/j.micpro.2016.03.011.

[8] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, Y. Todo, GIFT: A small present - towards reaching the limit of lightweight encryption, in: W. Fischer, N. Homma (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529 of Lecture Notes in Computer Science, Springer, 2017, pp. 321–345. doi:10.1007/978-3-319-66787-4_16.

[9] C. Beierle, G. Leander, A. Moradi, S. Rasoolzadeh, CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks, IACR Trans. Symmetric Cryptol. 2019 (1) (2019) 5–45. doi:10.13154/TOSC.V2019.I1.5-45.

[10] Y. Guo, L. Li, B. Liu, Shadow: A lightweight block cipher for iot nodes, IEEE Internet Things J. 8 (16) (2021) 13014–13023. doi:10.1109/JIOT.2021.3064203.

[11] J. Yang, L. Li, Y. Guo, X. Huang, DULBC: A dynamic ultra-lightweight block cipher with high-throughput, Integr. 87 (2022) 221–230. doi:10.1016/J.VLSI.2022.07.011.

[12] X. Huang, L. Li, J. Yang, IVLBC: an involutive lightweight block cipher for internet of things, IEEE Syst. J. 17 (2) (2023) 3192–3203. doi:10.1109/JSYST.2022.3227951.

[13] Y. Belkheyar, J. Daemen, C. Dobraunig, S. Ghosh, S. Rasoolzadeh, Bipbip: A low-latency tweakable block cipher with small dimensions, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2023 (1) (2023) 326–368. doi:10.46586/TCHES.V2023.I1.326-368.

[14] Q. Song, L. Li, X. Huang, Lelbc: A low energy lightweight block cipher for smart agriculture, Internet of Things 25 (2024) 101022. doi:10.1016/j.iot.2023.101022.

[15] A. A. Zakaria, A. H. Azni, F. Ridzuan, N. H. Zakaria, M. Daud, Systematic literature review: Trend analysis on the design of lightweight block cipher, J. King Saud Univ. Comput. Inf. Sci. 35 (5) (2023) 101550. doi:10.1016/J.JKSUCI.2023.04.003.

[16] C. A. Lara-Nino, A. Diaz-Perez, M. Morales-Sandoval, Lightweight hardware architectures for the present cipher in FPGA, IEEE Trans. Circuits Syst. I Regul. Pap. 64-I (9) (2017) 2544–2555. doi:10.1109/TCSI.2017.2686783.

[17] J. G. Pandey, T. Goel, A. Karmakar, Hardware architectures for PRESENT block cipher and their FPGA implementations, IET Circuits

Devices Syst. 13 (7) (2019) 958–969. doi:10.1049/IET-CDS.2018.5273.
[18] K. Shahbazi, S. Ko, Area-efficient nano-aes implementation for internet-of-things devices, IEEE Trans. Very Large Scale Integr. Syst. 29 (1) (2021) 136–148. doi:10.1109/TVLSI.2020.3033928.
[19] L. Li, J. Feng, B. Liu, Y. Guo, Q. Li, Implementation of PRINCE with resource-efficient structures based on fpgas, Frontiers Inf. Technol. Electron. Eng. 22 (11) (2021) 1505–1516. doi:10.1631/FITEE.2000688.
[20] R. Bharathi, N. Parvatham, Light-weight present block cipher model for iot security on fpga., Intelligent Automation & Soft Computing 33 (1) (2022) 35–49. doi:10.32604/iasc.2022.020681.
[21] J. Yang, L. Li, X. Huang, Low area and high throughput hardware implementations for the lilliput cipher, International Journal of Circuit Theory and Applications (2023). doi:10.1002/cta.3892.
[22] Z. Bao, J. Guo, S. Ling, Y. Sasaki, Peigen–a platform for evaluation, implementation, and generation of s-boxes, IACR Transactions on Symmetric Cryptology (2019) 330–394doi:10.46586/tosc.v2019.i1.330-394.
[23] J. Feng, Y. Wei, F. Zhang, E. Pasalic, Y. Zhou, Novel optimized implementations of lightweight cryptographic s-boxes via sat solvers, IEEE Transactions on Circuits and Systems I: Regular Papers (2023) 1–14doi:10.1109/tcsi.2023.3325559.
[24] M. S. Turan, K. McKay, D. Chang, L. E. Bassham, J. Kang, N. D. Waller, J. M. Kelsey, D. Hong, Status report on the final round of the nist lightweight cryptography standardization process (2023). doi:10.6028/nist.ir.8454.
[25] A. Xilinx, Ultrafast design methodology guide for xilinx fpgas and socs: Super logic region (2022).