Area-Throughput Efficient Implementations of CRAFT Cipher For Internet of Vehicles

Jiahao Xiang[a,b], Lang Li[a,b,*]

[a]College of Computer Science and Technology, Hengyang Normal University, Hengyang, 421002, , China
[b]Hunan Provincial Key Laboratory of Intelligent Information Processing and Application, Hengyang Normal University, Hengyang, 421002, , China

Abstract

Purpose: With extraordinary growth in the Internet of Vehicles (IoV), the amount of data exchanged between IoV devices is growing at an unprecedented scale. Most of the IoV devices need instant response and real-time security to ensure the safety of users. The CRAFT cipher that is a lightweight block cipher for low-area can be used in IoV devices. In order to better adapt to these environment, the objective of this paper is to explore opportunities to optimize area and throughput for CRAFT cipher targeted for low-resource IoV devices. **Methods:** A novel compact CRAFT implementation is proposed in serialized fashion to achieve a small hardware footprint. We propose novel unrolled structure of CRAFT cipher for the high throughput feature. **Results:**The results on Artix-7 show that ... **Conclusion:** Hence, our works let CRAFT cipher more suitable for IoV devices.

Keywords: Lightweight block cipher, Internet of Vehicles, Field-programmable gate array(FPGA), Low-area, High-throughput

1. Introduction

Internet of Vehicles (IoV) is an emerging concept in intelligent transportation systems (ITS) to enhance the existing capabilities of VANETs by integrating with the Internet of Things (IoT) [1]. As IoT technology continues to advance, IoV technology is also making great progress. But the same security issues that exist in IoT are also were introduced into IoV. At the some time, IoV involves a huge amount of dynamic real-time critical data so its security is a major concern.

Lightweight cryptography is a subfield of cryptography that aims to provide solutions tailored for resource-constrained devices [2]. It can provides security with low resource consumption and low delay in IoV environment.

In this work, we propose the three architectures of FPGA implementations for the CRAFT [3], respectively Round based, Serial, and Loop unrolled. This allows IoV practitioners to select the architectures that best suit their needs. The contributions of this article can be summarized as follows.

The rest of this article is organized as follows. Section 2 presents specification of CRAFT; the proposed the three architectures of FPGA implementations for the CRAFT are present in Section 3; Section 4 presents the implementation results, analysis, and comparison with other similar works; finally, the work is concluded in Section 5.

2. Specification of CRAFT

All notations used in this paper are shown in Table 1. CRAFT is a lightweight tweakable block cipher made out of involutory building blocks. It consists of a 64-bit block, a 128-bit key, and a 64-bit tweak. In this cipher, a 64-bit input plaintext P is transformed into a 64-bit output ciphertext C using a 128-bit key K and a 64-bit tweak T. Figure 1 shows the structure of CRAFT.

3. Implementations

To achieve efficient area and throughput, we have, for the first time, optimized the components of CRAFT and proposed two implementation architectures: Serial and Unrolled.

3.1. Serial Architecture (A1)

Compared to round-based architecture, serial architecture are able to reuse components and significantly reduce area usage, e.g., the number of Sub-Box is reduced from 16 to 1. The clock gating technique is also used to enable each component and reduce the energy consumption of encryption. Our proposed architecture is presented in Figure 2. The design includes one Sub-Box, one 4-bit Mix-columns, two register banks for storing keys (called Key-Register) and plaintext (called State-Register), which also act as temporary registers for storing the intermediate results. In order to store intermediate results into State-Register bank, the design has one feedback paths. PermuteNibbles is included in State-Register bank. It is noticeable that since the execution of permute requires 64-bit, in order to reuse the State-Register block, we change the order of execution of Sub-Box and Permute. And the first round of encryption process through the control signal to avoid Permute operation, to ensure the correctness of the encryption algorithm.

3.1.1. Sub-Box Optimization

Sub-Box provides a confusing characteristic for the entire encryption algorithm, however it requires a large amount of

*Corresponding author
Email address: lilang911@126.com (Lang Li)

Table 1: Notations used in this paper

Notation	Description
TK	64-bit tweakeys
RC_i	64-bit round constant in the i^{th} round

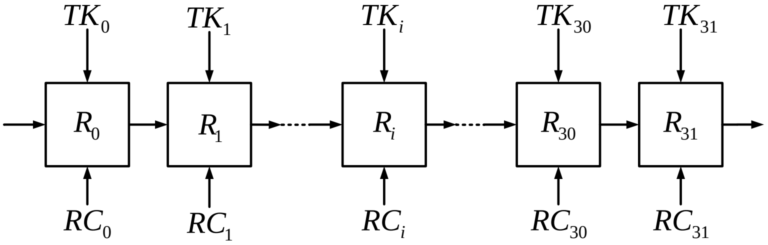

Figure 1: Structure of CRAFT

area. There are different methods of implementation of Sub-Box. The most popular implementation is using a lookup table (LUT), such as [4]. However it uses a lot of flip-flop, which will bring a lot of area consumption. Using Sub-Box's equivalent logical expression for this will reduce area consumption, such as [5], [6].

Boolean satisfiability (SAT) solvers can be used to find Sub-Box that satisfy certain implement, such as being resistant to software or hardware implement. In more detail, the Sub-Box implement can be encoded as Boolean constraints by representing the Sub-Box as a truth table and then using Boolean variables to represent the input and output bits of the Sub-Box. The constraints can then be formulated based on the desired implement of the Sub-Box. Once the Sub-Box implement are encoded as Boolean constraints, a SAT solver can be used to find a satisfying assignment to these constraints, which corresponds to an Sub-Box that satisfies the desired implement.

The gate equivalent complexity(GEC) of a SAT instance is the number of logical gates required to implement the Boolean formula that represents the instance. GEC can be calculated by converting the Boolean formula into a circuit of logical gates, such as AND, OR, and NOT gates. The number of gates in the circuit corresponds to the GEC of the instance.

In our design, we optimize and use GEC encoding scheme of [6] to implement the Sub-Box. Our encoding scheme as follows in Equations 1:

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

where K is numbers of the logical gates, $Q_{4i} - Q_{4i+3}$ 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. BB_i represents the type of the i^{th} logical gate, which is a 8-bit binary number. The different types of logical gate used in this encoding scheme are listed in Table 2.

The optimized architecture of Sub-Box is shown in Equations 2, where $X_0 - X_3$ is the input of the Sub-Box and $Y_0 - Y_3$ is the output of the Sub-Box. The proposed architecture of Sub-Box is implemented by four MOAI1 gates, three MAOI1 gates, and one AND3 gate. This module of the proposed Sub-Box reduced the area by 28.9% with [5] (based on gate equivalent estimation on UMC 180nm library).

$$\begin{aligned}
T_0 &= \text{MAOI1}(X_0, X_1, X_0, X_1) \\
T_1 &= \text{AND3}(X_3, X_2, X_3) \\
T_2 &= \text{MAOI1}(X_1, X_2, X_0, X_3) \\
T_3 &= \text{MOAI1}(X_1, X_0, X_2, X_2) \\
T_4 &= \text{MOAI1}(X_3, T_0, T_3, T_3) \\
T_5 &= \text{MOAI1}(T_3, T_0, X_0, T_1) \\
T_6 &= \text{MAOI1}(X_0, T_0, X_3, T_0) \\
T_7 &= \text{MOAI1}(X_0, T_1, T_2, T_2) \\
Y_0 &= T_5 \quad Y_1 = T_7 \quad Y_2 = T_6 \quad Y_3 = T_4
\end{aligned} \tag{2}$$

3.1.2. Mix-Columns Optimization

The Mix-Columns component is a linear transformation of the input column. The input column is multiplied by a constant matrix M to produce the output column. 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. The Mix-columns component is shown in Equation 3. where $I_{0,j}$, $I_{1,j}$, $I_{2,j}$, and $I_{3,j}$ are the input column, $I'_{0,j}$, $I'_{1,j}$, $I'_{2,j}$, and $I'_{3,j}$ are the output column, and j is the column index, $j \in \{0,\ldots,3\}$.

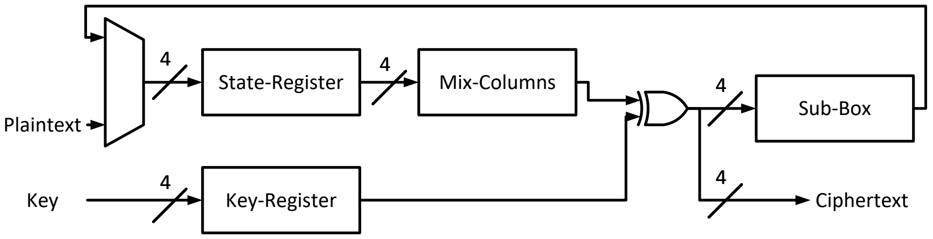

Figure 2: Serial architecture of CRAFT

Table 2: Encoding of different types of logical gate

logical expression	$BB_i[0{:}7]$	gate type
$Q_0 \oplus Q_1$	0 0 0 0 0 0 1 0	XOR
$\sim (Q_0 \oplus Q_1)$	0 0 0 0 0 0 1 1	XNOR
$Q_0 \land Q_1$	0 0 0 0 0 1 0 0	AND
$\sim (Q_0 \land Q_1)$	0 0 0 0 0 1 0 1	NAND
$Q_0 \lor Q_1$	0 0 0 0 0 1 1 0	OR
$\sim (Q_0 \lor Q_1)$	0 0 0 0 0 1 1 1	NOR
$\sim Q_0$	0 0 0 0 1 0 0 1	NOT
$\sim Q_1$	0 0 0 0 1 0 1 1	NOT
$\sim Q_2$	0 0 0 1 0 0 0 1	NOT
$Q_0 \oplus Q_1 \oplus Q_2$	0 0 0 1 0 0 1 0	XOR3
$\sim (Q_0 \oplus Q_1 \oplus Q_2)$	0 0 0 1 0 0 1 1	XNOR3
$Q_0 \land Q_1 \land Q_2$	0 0 1 0 0 0 0 0	AND3
$\sim (Q_0 \land Q_1 \land Q_2)$	0 0 1 0 0 0 0 1	NAND3
$Q_0 \lor Q_1 \lor Q_2$	0 1 1 1 0 1 1 0	OR3
$\sim (Q_0 \lor Q_1 \lor Q_2)$	0 1 1 1 0 1 1 1	NOR3
$\sim ((Q_0 \land Q_1) \lor (\sim (Q_2 \lor Q_3)))$	1 0 1 1 0 0 0 0	MAOI1
$\sim (\sim (Q_0 \land Q_1) \land ((Q_2 \lor Q_3)))$	1 0 1 1 0 0 0 1	MOAI1

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

In order to reduce the area of this component, we use the serial architecture of Mix-Columns, as shown in Figure 3. The serial architecture of Mix-Columns requires four 4-bit register, two multiplexes and three XOR gates. The operation of Mix-Columns involves three distinct stages: freeze, shift, and add. During the freeze stage, we ensure the register values remain unchanged by setting both CM_0 and CM_1 to 0. In the shift stage, we induce a shift in the register values from RM_0 to RM_4 by setting both CM_0 and CM_1 to 1. Finally, in the add stage, we execute an addition operation on the column values according to Equation 3. This is achieved by setting CM_0 and CM_1 to 0 and 1, respectively.

The timing diagram for the serial architecture of Mix-Columns is depicted in Figure 4. 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.

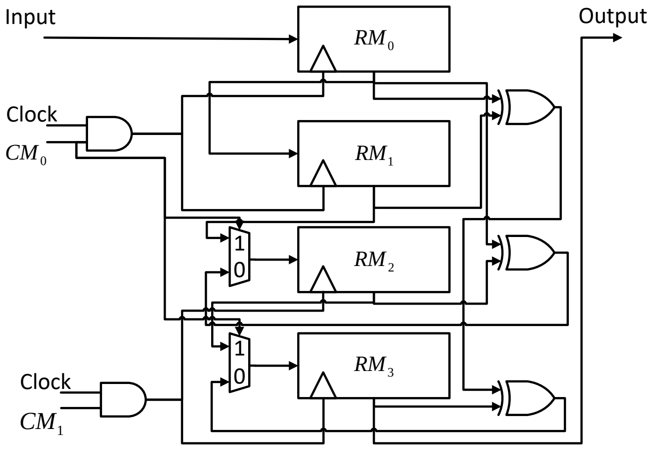

Figure 3: Serial Architecture of Mix-Columns with clock gating

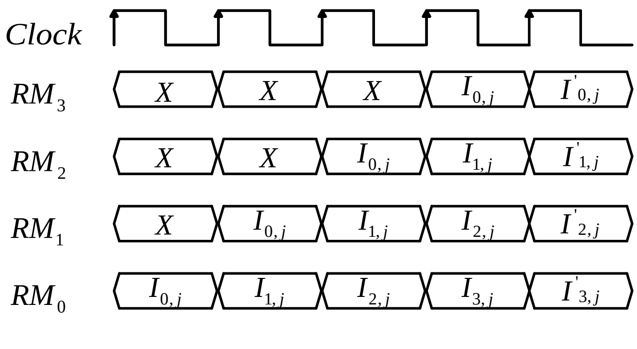

Figure 4: Timing Diagram for the Serial Architecture of Mix-Columns

3.1.3. Control Units

The finite-state machine (FSM) of the serial architecture, as shown in Figure 5, begins its encryption process by storing the initial key and plaintext in the Key-Register and State-Register, respectively. The key undergoes expansion in the Key-Schedule phase, during which the clocks of the Mix-Columns and State-Register are disabled. The Mix-Columns phase follows, storing one column of the State-Register in the Mix-Columns registers and taking five clock cycles to execute Mix-Columns on one column. Upon completion of this phase, the clocks of the State-Register and Key-Register are disabled. The Sub-Box phase then takes over, sending the data stored in the Mix-Columns registers back to the State-Register and XORing it with the keys, a process that takes another 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. The Permute operation is then executed in one clock cycle inside the State-Register. The encryption process concludes when the Round counter reaches 31, at which point the ciphertext is stored in the State-Register.

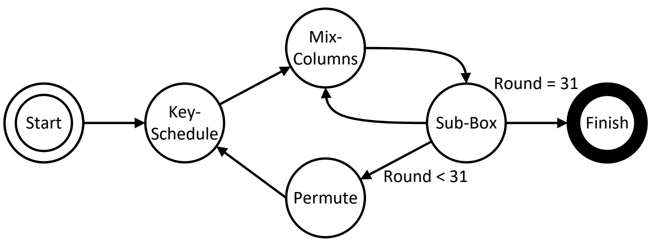

Figure 5: Finite-state machine for Serial Architecture

Clock gating, a technique discussed in [7], can help reduce the dynamic power consumption of the encryption. This technique is applied separately to the State-Register, Key-Register, and Mix-Columns. For example, during the Key-Schedule phase, the clock of the State-Register and Mix-Columns is turned off because these two blocks are not needed. This helps save a significant amount of power. Figure 6 shows the timing diagram of a design that uses the clock gating technique.

3.2. Unrolled Architecture (A2)

The 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, based on the iterative architecture from [3], 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 (A3)

The architecture from [4] 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. Concurrently, the Key Schedule runs alongside the round function. This architecture is depicted in Figure 8. For comparison, different architectures are listed in Table 3.

4. Experimental Evaluation

In ASIC implementations, the gate equivalent (GE) is often used to evaluate the area consumption of a design. A single GE is equivalent to a two-input NAND gate. To calculate the area in GE, we divide the total area (in μm^2) by the area of a two-input NAND gate (also in μm^2). However, the number of GEs can vary based on the specific technology used, as discussed in [2]. 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. In order to ensure a fair comparison, we evaluate the area consumption of the proposed designs using FPGA implementations, a method also utilized in [8].

4.1. Platform

The proposed architectures were implemented on a Xilinx FPGA board using Vivado v2023.2. To test in various environments, three different FPGA platforms were used for benchmarking: Artix-7(xc7a100tcsg324-1), Spartan-7(xc7s100fgga484-1), and Kintex-7(xc7k70tfbg484-1). Artix-7 offers high performance in resource-limited situations. Spartan-7 is designed for high-restriction environments. Kintex-7 is ideal for applications like 3G and 4G wireless, flat panel displays, and video over IP solutions.

4.2. Area

The Area metric, which includes components like Flip-Flops, LUTs, and Slices, is used to measure the area used by the proposed designs. To make a fair comparison, the FPGA's embedded memory blocks were not used. This was done by turning off the related settings in the VHDL, as suggested in [9]. Also, all designs were synthesized and implemented using the same settings, specifically, the default settings of Vivado Synthesis and Implementation.

4.3. Throughput

The performance of the proposed designs is evaluated 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 4. The throughput rate at 100MHz shows the rate achievable when the clock frequency is set to 100MHz, calculated using Equation 5. The throughput rate per slice is a measure of efficiency, calculated by dividing the throughput rate by the number of Slices (Equation 6). In these calculations, the Plaintext Size is 64-bit, Latency refers to the number of clock cycles required to encrypt a single block, and Slices refers to the number of Slices used by the design.

$$MaxThroughput(Thr) = \frac{MaxFrequency \times PlaintextSize}{Latency} \tag{4}$$

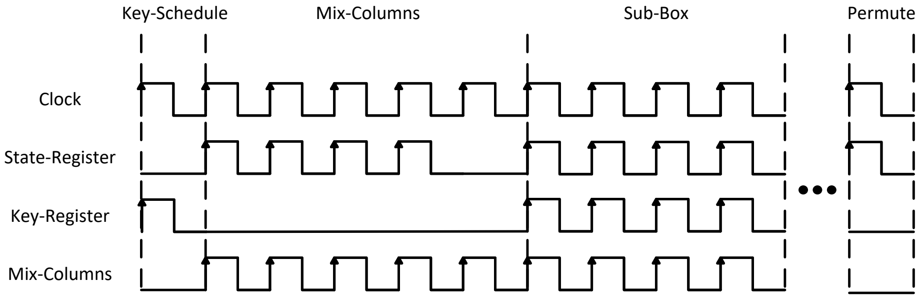

Figure 6: Timing diagram for Serial Architecture

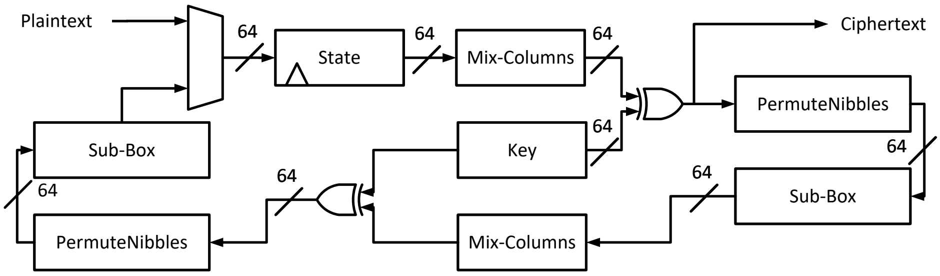

Figure 7: Unrolled architecture of CRAFT

$$Throughput_{@100MHz}(Thr^{*}) = \frac{100MHz \times PlaintextSize}{Latency} \quad (5)$$

$$ThroughputPerSlice = \frac{Thr}{Slices} \quad (6)$$

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 7. 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 8.

$$\begin{aligned}\text{Total Power}(TP) &= \text{Dynamic Power}(DP)\\ &+ \text{Static Power}(SP) \quad (7)\end{aligned}$$

$$Energy(E) = \frac{TP \times Latency}{Frequency} \quad (8)$$

5. Results

6. Conclusion

References

[1] S. Sharma, B. Kaushik, A survey on internet of vehicles: Applications, security issues & solutions, Vehicular Communications 20 (2019) 100182.

[2] K. McKay, L. Bassham, M. Sönmez Turan, N. Mouha, Report on lightweight cryptography, Tech. rep., National Institute of Standards and Technology (2016).

[3] C. Beierle, G. Leander, A. Moradi, S. Rasoolzadeh, Craft: lightweight tweakable block cipher with efficient protection against dfa attacks, IACR Transactions on Symmetric Cryptology 2019 (1) (2019) 5–45.

[4] 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.
URL https://doi.org/10.1109/TCSI.2017.2686783

[5] 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) 330–394.

[6] 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).

[7] K. Shahbazi, S.-B. Ko, Area-efficient nano-aes implementation for internet-of-things devices, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 29 (1) (2020) 136–148.

[8] K. Mohajerani, R. Haeussler, R. Nagpal, F. Farahmand, A. Abdulgadir, J.-P. Kaps, K. Gaj, Fpga benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: Methodology, metrics, tools, and results, Cryptology ePrint Archive (2020).

[9] A. Xilinx, Ultrafast design methodology guide for xilinx fpgas and socs: Super logic region (2022).

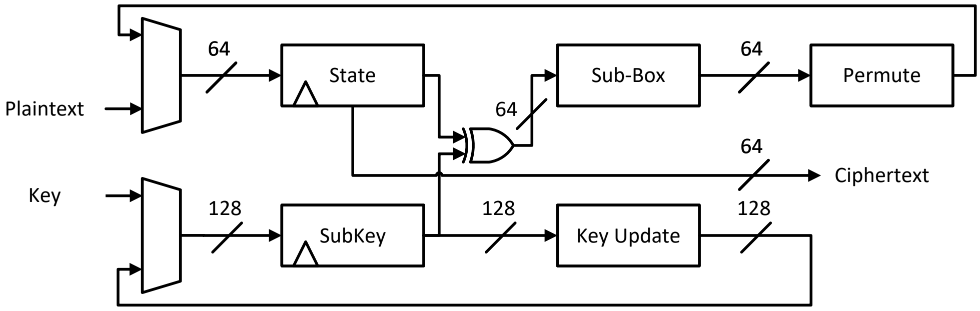

Figure 8: Iterative architecture of PRESENT

Table 3: Different architectures description

Architecture	Cipher	Block Size	Key Size	Reference	Description
A1	CRAFT	64	128	This work	Serial
A2	CRAFT	64	128	This work	Unrolled
A3	PRESENT	64	128	[4]	Iterative

Table 4: Area used for the three Architectures

Platform	Design	$State(bit)$	$Key(bit)$	FF	LUT	$Slices$
Artix-7	A1	64	128	**144**	**177**	**59**
	A2	64	128	157	378	111
	A3	64	128	201	243	70
Spartan-7	A1	64	128	**144**	**177**	**57**
	A2	64	128	157	381	118
	A3	64	128	201	244	70
Kintex-7	A1	64	128	**144**	**178**	**58**
	A2	64	128	157	377	115
	A3	64	128	201	244	68

Table 5: Throughput results for the three Architectures

Platform	Design	$Latency$	$FMAX(MHz)$	$Thr(Mbps)$	$Thr^*(Mbps)$[1]	$\frac{Thr}{Slices}(\frac{Kbps}{Slices})$
Artix-7	A1	1215	**557.41**	29.36	5.27	497.65
	A2	**16**	142.38	569.52	**400.00**	5130.81
	A3	32	274.04	548.08	200.00	7829.71
Spartan-7	A1	1215	**525.76**	27.69	5.27	485.87
	A2	**16**	138.86	555.44	**400.00**	4707.12
	A3	32	296.29	592.58	200.00	8465.43
Kintex-7	A1	1215	**853.97**	44.98	5.27	775.57
	A2	**16**	175.25	701.00	**400.00**	6095.65
	A3	32	357.78	715.56	200.00	10522.94

Table 6: Power and Energy consumption for the three Architectures

Platform	Design	$DP(mW)$	$SP(mW)$	$TP(mW)$	$E(uJ)$	$\frac{E}{bit}(\frac{nJ}{bit})$
Artix-7	A1	2.00	139.00	141.00	1.71	26.77
	A2	7.00	139.00	146.00	**0.02**	**0.37**
	A3	2.00	139.00	141.00	0.05	0.71
Spartan-7	A1	2.00	140.00	142.00	1.73	26.96
	A2	7.00	140.00	147.00	**0.02**	**0.37**
	A3	3.00	140.00	143.00	0.05	0.72
Kintex-7	A1	2.00	145.00	147.00	1.79	27.91
	A2	8.00	145.00	153.00	**0.02**	**0.38**
	A3	3.00	145.00	148.00	0.05	0.74