A High-Speed FPGA Implementation of A Bit-slice Ultra-Lightweight Block Cipher, RECTANGLE

Soheil Feizi*, Ali Nemati*, Arash Ahmadi*†, Vahab Al-din Makki*
Email: {s.f31391, ali_nemati_1989}@yahoo.com , {aahmadi, v.makki}@razi.ac.ir
*Department of Electronic Engineering, Razi University, Kermanshah, Iran
†Department of Electrical Engineering, University of Windsor, Ontario, Canada

***Abstract*— Field programmable gate arrays (FPGAs) are fundamental flexible integrated circuits that enable engineers to program any circuits that they want in the field. If we want to mention a practical building block of a system, FPGA shines as a really good option and when this equipment is more advanced, all digital circuits and systems can be placed on this so FPGAs can be configured as system on a chip (SoC). Today in security protocols, block ciphers are one of the important basis in ensuring security. In 2014 Wentao Zhang et al proposed a new Bit-slice Ultra-Lightweight block cipher named as RECTANGLE, which are suitable for extremely constrained environments and multiple platforms. In this paper, we simulate and synthesize the RECTANGLE lightweight algorithm. Our methodology is used high level synthesis for give more flexibility to achieve suitable throughput in our architecture. Implementation results of the RECTANGLE cryptography algorithm on a FPGA are presented. The design target is efficiency of throughput and speed.**

Keywords— Lightweight Cryptography; RECTANGLE Algorithm; High Level Synthesis; FPGA.

I. INTRODUCTION

The basic idea to design and manufacturing FPGAs take from PLDs in 1985 and at this time have received to close to ten million gates. FPGAs have a flexible and regular structure that consisting of logic blocks and connections between them, which allows the user to access and control his design. In fact, the designer can implement his ideas into the circuit and make any changes whenever he wants. In the past decades, logical circuit design by FPGAs is one of many hardware design and execute ways but nowadays is a complex and specialized skills for hardware engineers that can take years of training and experience to be a master in this domain [7, 8].

FPGAs covering a wide range of applications, including data communications, military, aerospace, nuclear energy and industrial controls. Production time and effectiveness cost are the two parameters that the benefits of FPGAs compared to fixed logic devices such as Application Specific Integrated Circuits (ASICs). Almost all lightweight cryptographic algorithms have been implemented in ASIC technology but many articles about their implementation in FPGA is not provided. The best option for providing security in low-resource deeply-embedded systems is using block ciphers [7,8].

Different applications have different security requirements, and in any case to achieve the desired level of security, protocols should be developed. Each block cipher is adaptively to initial cryptographic principles and it seems that any protocol for constrained environments and embedded systems can be implemented using a block cipher with the suitable data block and key size. Although some standard cryptographic algorithms such as Advance Encryption Standard (AES) have been proposed for lightweight applications, choosing AES is not a good option for highly constrained environments [2, 14].

Some of the most popular cryptography algorithms such as PRESENT [20], PICCOLO [4], KATAN [3], and HIGHT [19] are implemented on software platforms for example AVR Atmel microcontroller and hardware platforms such as FPGA to appraise their area requirements, memory efficiency and energy consumption in performance analysis. Moreover, degree of confusion and diffusion are evaluated in security analysis. Obviously, any effort to optimize the performance of a parameter on one side will have a negative impact on the performance of other parameters on the other side meaning that the tradeoff law is in place [2, 3, 4].

In 2014, Wentao Zhang et al proposed a new family of highly optimized block cipher RECTANGLE that has flexibility and superior performance in lightweight applications. Bit-slice technique is used in this block cipher by the designers in order to achieve high throughput and very low area in hardware. RECTANGLE uses the substitution-permutation network as structure. The S and P (substitution and permutation) layers consist of 16 4×4 S-boxes in parallel and 3 rotations, respectively [1]. In this paper, we designed high-speed architectures of RECTANGLE cryptography algorithm for Xilinx Spartan-3 FPGA. Our designs of RECTANGLE are described in VHSIC Hardware Description Language (VHDL) and used Xilinx ISE tools to synthesizing for Xilinx Spartan-3 FPGAs. All Obtained results are after place and route stage.

The contents of this paper is organized as follows: In Section II we explain the specification of RECTANGLE briefly. Section III presents the synthesis of the RECTANGLE block cipher family. In Sections IV we present the implementation results of RECTANGLE algorithm on FPGA, respectively. Section V concludes the paper.

978-1-4673-9280-8/15/$31.00 ©2015 IEEE

II. RECTANGLE ALGORITHM DESCRIPTION

RECTANGLE block cipher algorithm covers 64 bit data block length with 80 or 128 bit key length to start the encryption process [1].

A. The cipher and sub key states

The cipher state contains a set of data that include a 64-bit initial input data or output data of the encryption process, which can involve any round even if the data reach the final stage of the encryption. The main reason for naming algorithm to RECTANGLE is that we can display cipher state in the form of 4×16 array of bits [1]. Let $W = w_{63}\| \ldots \|w_1\|w_0$ define a 64-bit cipher state, the first 16 bits $w_{15}\| \ldots \|w_1\|w_0$ are placed in row 0 and the next 16 bits $w_{31}\| \ldots \|w_{17}\|w_{16}$ are placed in row 1, and will continue to do so, as quite visible in (1). Also, a 64-bit sub key can be used as a 4×16 array of bits Similar to the input data block. To understand explanations better, the authors of the algorithm have displayed a cipher state in two ways as illustrated in (1), [1].

$$\begin{bmatrix} w_{15} & w_{14} & \ldots & w_2 & w_1 & w_0 \\ w_{31} & w_{30} & \ldots & w_{18} & w_{17} & w_{16} \\ w_{47} & w_{46} & \ldots & w_{34} & w_{33} & w_{32} \\ w_{63} & w_{62} & \ldots & w_{50} & w_{49} & w_{48} \end{bmatrix} = \begin{bmatrix} a_{0,15} & a_{0,14} & \ldots & a_{0,2} & a_{0,1} & a_{0,0} \\ a_{1,15} & a_{1,14} & \ldots & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{2,15} & a_{2,14} & \ldots & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{3,15} & a_{3,14} & \ldots & a_{3,2} & a_{3,1} & a_{3,0} \end{bmatrix} \quad (1)$$

B. The Round Transformation

This algorithm has not feistel structure and follows the substitution-permutation network. RECTANGLE algorithm has 25 rounds and each of them includes the following three phases: Add Round key, Sub Column and Shift Row. After the last round, there is a final Add Round Key [1].

1) Add Round key: XOR logical operation between the current state and the round sub key [1].

2) Sub Column: The operation of Sub Column is depicted in Fig. 1. The input of a S-box is $Col(j) = a_{3,j}\|a_{2,j}\|a_{1,j}\|a_{0,j}$ for $0 \le j \le 15$, and the output is $S(Col(j)) = b_{3,j}\|b_{2,j}\|b_{1,j}\|b_{0,j}$. The S-box belongs to the RECTANGLE algorithm acts as a 4-bit to 4-bit S-box S: $F^4_2 \rightarrow F^4_2$. The operation of this S-box are fully specified in the Table I. The values are given in hexadecimal. [1].

3) Shift Row: Each row is shifted and rotated to the left to the necessary extent. Row 0 is not rotated, row 1, 2 and 3 are left rotated over 1, 12 and 13 bits consecutive. The <<< (x) symbol is used for left rotation over x bits, the operation Shift Row is depicted below [1]:

$$\begin{aligned} (a_{0,15}\ a_{0,14} \ldots a_{0,1}\ a_{0,0}) &\xrightarrow{<<<(0)} (a_{0,15}\ a_{0,14} \ldots a_{0,1}\ a_{0,0}) \\ (a_{1,15}\ a_{1,14} \ldots a_{1,1}\ a_{1,0}) &\xrightarrow{<<<(1)} (a_{1,14}\ a_{1,13} \ldots a_{1,0}\ a_{1,15}) \\ (a_{2,15}\ a_{2,14} \ldots a_{2,1}\ a_{2,0}) &\xrightarrow{<<<(12)} (a_{2,3}\ a_{2,2} \ldots a_{2,5}\ a_{2,4}) \\ (a_{3,15}\ a_{3,14} \ldots a_{3,1}\ a_{3,0}) &\xrightarrow{<<<(13)} (a_{3,2}\ a_{3,1} \ldots a_{3,4}\ a_{3,3}) \end{aligned}$$

C. Key expansion

When the 80-bit key to be considered as $V = v_{79}\| \ldots \|v_1\|v_0$, these bits are firstly stored in an 80-bit register. Equation (2) shows the details [1].

$$\begin{bmatrix} v_{19} & v_{18} & \ldots & v_2 & v_1 & v_0 \\ v_{39} & v_{38} & \ldots & v_{22} & v_{21} & v_{20} \\ v_{59} & v_{58} & \ldots & v_{42} & v_{41} & v_{40} \\ v_{79} & v_{78} & \ldots & v_{62} & v_{61} & v_{60} \end{bmatrix} = \begin{bmatrix} k_{0,19} & k_{0,18} & \ldots & k_{0,2} & k_{0,1} & k_{0,0} \\ k_{1,19} & k_{1,18} & \ldots & k_{1,2} & k_{1,1} & k_{1,0} \\ k_{2,19} & k_{2,18} & \ldots & k_{2,2} & k_{2,1} & k_{2,0} \\ k_{3,19} & k_{3,18} & \ldots & k_{3,2} & k_{3,1} & k_{3,0} \end{bmatrix} \quad (2)$$

The 16 rightmost columns of the key register are placed side by side and are attached together to obtain the 64-bit of the i-th sub key K_i at round i, i.e. $K_i = (k_{3,15}\|\cdots\|k_{3,1}\|k_{3,0}) \| (k_{2,15}\|\cdots\|k_{2,1}\|k_{2,0})\|(k_{1,15}\|\cdots \|k_{1,1}\|k_{1,0})\|(k_{0,15}\|\cdots\|k_{0,1}\|k_{0,0})$ [1].

TABLE I. S-BOX (HEXADECIMAL) [1]

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(X)	9	4	F	E	A	1	0	6	C	7	3	8	2	B	5	D

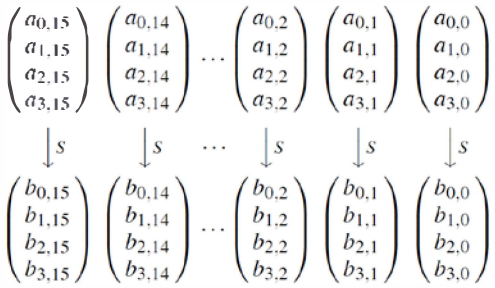

Fig. 1. Procedure of SubColumn Operation [1]

After extracting K_i, the register key values are changed and updated as follows [1]:

1) 0-th column will be affected by the S-box, i.e., $k_{3,0}\|k_{2,0}\|k_{1,0}\|k_{0,0} := S(k_{3,0}\|k_{2,0}\|k_{1,0}\|k_{0,0})$ [1].

2) Each row is shifted and rotated to the left to the necessary extent. Row 0 and 1 are left rotated over 7 and 9 bits consecutive. Row 2 and 3 are left rotated over 11 and 13 bits successive.

3) $R_C[i]$ is a round constant with 5 bits. The 5-bit key state is XORed with $R_C[i]$. $(k_{0,4} \| k_{0,3} \| k_{0,2} \| k_{0,1} \| k_{0,0})$, i.e., $(k_{0,4} \| k_{0,3} \| k_{0,2} \| k_{0,1} \| k_{0,0}) := (k_{0,4} \| k_{0,3} \| k_{0,2} \| k_{0,1} \| k_{0,0}) \oplus R_C[i]$, [1]. Finally, K_{25} is extracted from the updated key state.

D. The Cipher

The process of turning an arbitrary plaintext into ciphertext in this algorithm takes 25 rounds with a final sub key XOR. The pseudo C code for RECTANGLE algorithm comes after this [1]:

```
Generate Round Keys ( )
for i = 0 to 24 do {
    Add Round Key (STATE, Ki)
    Sub Column (STATE)
    Shift Row (STATE) }
Add Round Key (STATE, K25)
```

III. HIGH-LEVEL SYNTHESIS

The synthesis role is to catch a specification of the needed behavior of a system and a collection of restrictions and aims to be contented, and to gain a structure that taking into account the aims and constraints to implement the desired behavior [6]. The input characteristic gives the needed mappings from consecution of inputs to sequences of outputs, where those inputs and outputs may communicate with the outside environment or with another system- level part. The characteristic should inflict the interior architecture of the desired system to be designed as small as conceivable. The existence of related series of registers, functional units, multiplexers and other necessary elements is a good feature for an appropriate synthesis system that generates

a delineation of a data path. If the control is not integrated into the data path, and it usually is not, the synthesis system must as well as generate the characteristic of the control unit [6]. Resource sharing allows us to get the maximum possible efficiency of a hardware i.e. we can perform several orders with same hardware. In scheduling phase according to the goal of the design and implementation, logical operations are divided into the specific control steps. In allocation phase depending on the type of operation that needs to be done, appropriate operators are selected from library of resources. For example, registers used for storing data values. To start the simulation and synthesis steps, we have chosen RECTANGLE-80 algorithm. According to section II, it is explicit that data block length = 64 bit, key length = 80 bit and rounds = 25. Also, an example of this type of algorithm is illustrated in Table II. At first, we've written VHDL code of RECTANGLE-80 algorithm and Simulation results are shown in Fig. 3. Then due to the structure of the RECTANGLE encryption algorithm, synthesis can be divided into two sub-sections: Synthesis of encryption part and Synthesis of key expansion part. In view of the Fig. 1 presented in Section II, Let $A = A_3 \| A_2 \| A_1 \| A_0$ denote the 4 16-bit inputs to the S-box S and let $S(A) = B_3 \| B_2 \| B_1 \| B_0$ as 4 16-bit outputs that $A_3, \ldots, A_0$ and $B_3, \ldots, B_0$ selected as a row of state. By using the logical relations, the RECTANGLE S-box can be represented by the following Boolean functions:

$$\begin{cases} B_2 = A_1 \text{ xor } [A_2 \text{ xor } (A_0 \text{ or } A_3)] \\ B_1 = (A_3 \text{ xor } B_2) \text{ xor } [A_0 \text{ and } [A_2 \text{ xor } (A_0 \text{ or } A_3)]] \\ B_3 = (A_0 \text{ xor } A_1) \text{ xor } [[A_2 \text{ xor } (A_0 \text{ or } A_3)] \text{ or } (\text{not } B_1)] \\ B_0 = [A_2 \text{ xor } (A_0 \text{ or } A_3)] \text{ xor } [(\text{not } B_1) \text{ or } B_3] \end{cases} \quad (3)$$

Regarding the relations obtained, S-box table, encryption and key expansion parts presented in Section 2, it is clear that substitution operations require to AND, OR, XOR and NOT operators. Therefore for synthesis of the algorithm, these logical operations have been located such a way that in each control step, we have needed at most 2 ALUs, 2 Shift Registers and 2 standard Registers. Data Flow Graph (DFG) for the algorithm is drawn and changes have been applied such a way that each operation round of the algorithm is done in 18 Control Step (CS). Also, Sub-operations have been located in such a way that for doing each round of the algorithm just 2 ALUs, 2 Shift-Registers and 7 Registers are required. Efficient state for scheduling, allocation and binding is shown in Fig. 2. Whole encryption operation is performed in $(18 \times 25) + 1 = 451$ CS.

In next step, designed hardware to run the RECTANGLE algorithm is discussed as depicted in Fig. 4. This hardware includes fourteen Muxes, two ALUs, seven Registers and two Shift-Registers. Designed sequencer for this hardware is shown in Fig. 5. This sequencer includes two AND gate, one OR gate, one counter and a very simple controller. According to the number of control steps, counter determines which outputs of the controller are '0' and which are '1'. Sequencer gives necessary instruction execution time of the hardware as presented in Table III. A noteworthy point in Fig. 2 is that all operations are performed by ALU2 are single-bit operations, while operations are done by ALU1 are at most 64-bit for XOR and 16-bit for other operations. Therefore, we consider this point for choosing ALUs to reduce the area and cost.

TABLE II. TEST VECTOR [1]

Rec-80	Plaintext	Key	Ciphertext
Binary	0000000000000000 0000000000000000 0000000000000000 0000000000000000	00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000	0111100100011110 1100111100101100 1110111100001001 1010011101100011
Hex	0000000000000000	00000000000000000000	A763EF09CF2C791E

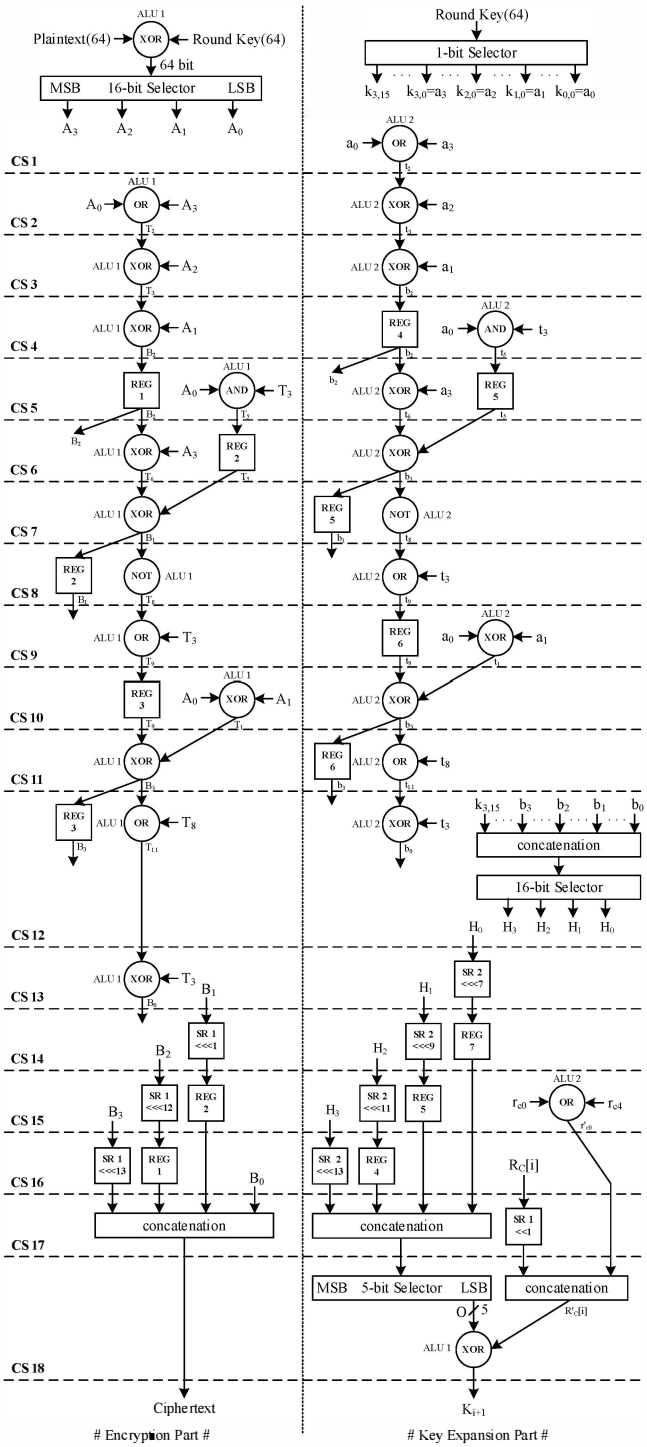

Fig. 2. Scheduling, Allocation and Binding

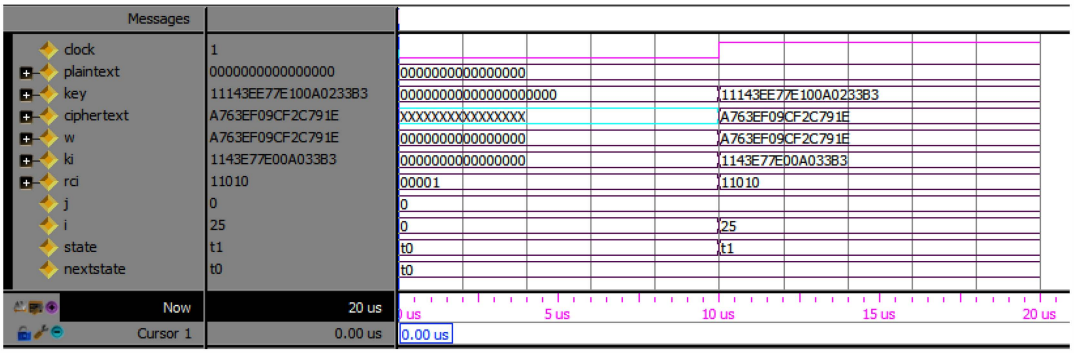

Fig. 3. Simulation Result

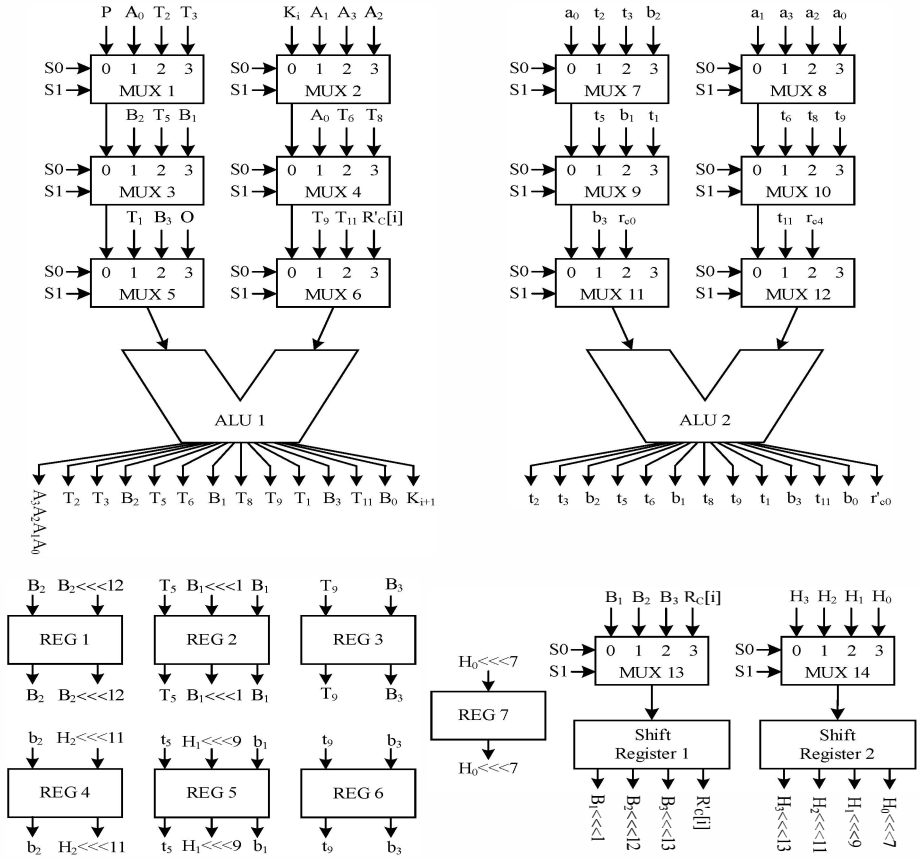

Fig. 4. Hardware Design

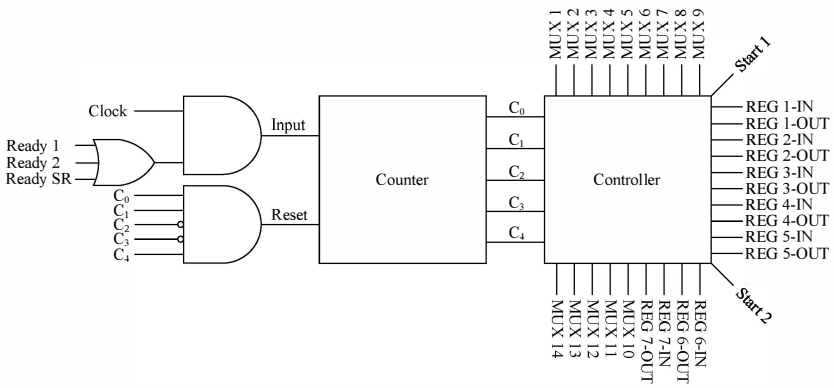

Fig. 5. Sequencer Unit

TABLE III. SEQUENCER PARAMETERS

Control Steps ($C_4C_3C_2C_1C_0$)	Start 1	Start 2	REG 1-IN	REG 1-OUT	REG 2-IN	REG 2-OUT	REG 3-IN	REG 3-OUT	REG-4 IN	REG 4-OUT	REG 5-IN	REG 5-OUT	REG 6-IN	REG 6-OUT	REG 7-IN	REG 7-OUT	MUX 1 (S_1S_0)	MUX 2 (S_1S_0)	MUX 3 (S_1S_0)	MUX 4 (S_1S_0)	MUX 5 (S_1S_0)	MUX 6 (S_1S_0)	MUX 7 (S_1S_0)	MUX 8 (S_1S_0)	MUX 9 (S_1S_0)	MUX 10 (S_1S_0)	MUX 11 (S_1S_0)	MUX 12 (S_1S_0)	MUX 13 (S_1S_0)	MUX 14 (S_1S_0)
00001	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	00	00	00	00	00	00	01	00	00	00	00	××	××
00010	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	01	10	00	00	00	00	01	10	00	00	00	00	××	××
00011	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	11	00	00	00	00	10	00	00	00	00	00	××	××
00100	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	11	01	00	00	00	00	10	11	00	00	00	00	××	××
00101	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	11	××	00	01	00	00	11	01	00	00	00	00	××	××
00110	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	××	10	01	00	00	00	××	××	01	01	00	00	××	××
00111	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	××	××	10	10	00	00	××	××	10	××	00	××	××	××
01000	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	××	××	11	××	00	××	10	××	00	10	00	00	××	××
01001	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	11	××	00	11	00	00	00	00	00	00	00	00	××	××
01010	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	01	01	00	00	00	00	××	××	11	11	00	00	××	××
01011	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	××	××	××	××	01	01	××	××	××	10	01	00	××	××
01100	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	××	××	××	11	10	00	10	××	00	××	00	01	××	××
01101	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	××	00	××	00	10	××	××	××	××	××	××	××	11
01110	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	××	××	××	××	××	××	××	××	××	××	××	××	00	10
01111	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	××	××	××	××	××	××	××	××	××	××	10	10	01	01
10000	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	××	××	××	××	××	××	××	××	××	××	××	××	10	00
10001	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	××	××	××	××	××	××	××	××	××	××	××	××	11	××
10010	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	××	××	××	××	11	11	××	××	××	××	××	××	××	××
10011	RESET																													

IV. HARDWARE IMPLEMENTATION

In general, architectures can be divided into 2 categories: architectures are appropriate for high-cost and high-throughput applications and architectures are proper for low-cost and low-area applications. It is tried to design an efficient architecture of RECTANGLE cryptography algorithm for Xilinx Spartan-3 FPGA. We have implemented RECTANGLE, with a very high-speed hardware architecture that has throughput 773 Mb/s. The Xilinx synthesis technology synthesizes VHDL code to make Xilinx files with .ngr and .ngc extensions after the functional simulation. We implemented the RECTANGLE-80 algorithm using Xilinx ISE development tools on FPGA model Spartan-3 S50. The results and resource utilization of the design is summarized in Table IV. Comparisons on utilization between RECTANGLE and previous works is depicted in Table IV to show its efficiency for throughput with considering resource limited hardware.

TABLE IV. IMPLEMENTATION RESULT

Resource / Algorithm	Area (Slice)	Throughput (Mb/s)	Max. Freq. (MHz)	Device
This Work	483	773	149	XC3S50
SIMON [24]	399	216.8	135	XC3S50
SEA [25]	438	260	241	XC4VLX25
CLEFIA [26]	323	690	194	XC3S1200-4
X-TEA [27]	254	36	-	XC3S50-5
MICKEY [28]	115	233	233	XC3S50

V. CONCLUSION

In this paper, we have described, synthesized and implemented an ultra-lightweight block cipher RECTANGLE, which has 64 bit and 80 or 128 bit for block and key, respectively. RECTANGLE algorithm is a lightweight flexible algorithm for using in RFID systems and wireless sensor networks (WSNs) and deployment capabilities in many other applications as well. Due to the flexibility of the algorithm used in various applications and environments, key length can be selected with different sizes. In RECTANGLE-80, due to having few resources and high throughput as shown in our implementation results, it is very suitable for lightweight application and embedded systems. As is presented in prior section, physical resources of FPGA used to implement RECTANGLE encryption system are very small, compared with the FPGA capacity, allowing further development of algorithms with higher encryption capacity. Maximum frequency of implementation is 149 MHz. This hardware implementation illustrates that the algorithm can be implemented with high throughput and speed.

REFERENCES

[1] W. Zhang, Z. Bao, D. Lin, V. Rijmen and B. Yang, "RECTANGLE: A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms", Cryptology ePrint Archive, 2014. http://eprint.iacr.org/.

[2] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks and L. Wingers, "The simon and speck of lightweight block ciphers," National Security Agency, USA, 2013.

[3] C. D. Cannière, O. Dunkelman, and M. Kneževi´c, "KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers," In CHES 2009, Lecture Notes in Computer Science No. 5747, pages 272–88. Springer-Verlag, 2009.

[4] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, “Piccolo: An Ultra-Lightweight Blockcipher,” Sony Corporation, Japan, 2011.

[5] M. Zwolinski, “Digital System Degin with VHDL,” 2nd Edition, 2004.

[6] G. D. Micheli, “Synthesis and optimization of digital circuits,” 1994.

[7] V. Sklyarov, I. Skliarova, A. Barkalov and L. Titarenko, “Synthesis and Optimization of FPGA-Based Systems,” Springer, 2014.

[8] F. Rodriguez-Henriquez, N.A. Saqib, A. Díaz Pérez and C.K. Koc, “Cryptographic Algorithms on Reconfigurable Hardware,” Springer, 2007.

[9] E. Vahedi,V. W.S. Wong, I. F. Blake, “An Overview of Cryptography,” Chapter 5, University of British Columbia, Canada, 2014.

[10] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, pp. 202-203, 2001.

[11] Xilinx, Inc., Spartan-3 User Guide, available at www.xilinx.com.

[12] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, “A Survey of Lightweight Cryptography Implementations,” IEEE Design & Test of Computers, vol. 24, no. 6, pp. 522-533, 2007.

[13] P. Kitsos, N. Sklavos, M. Parousi and A. N. Skodras, “A comparative study of hardware architectures for lightweight block ciphers”, Elsevier Journal, Computers and Electrical Engineering, pp. 148-160, 2012.

[14] FIPS, “Advanced Encryption Standard (AES)”, Federal Information Processing Standards Publication 197.

[15] Mi Lu, “Arithmetic and Logic in Computer Systems”, Texas A&M University, Published by John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[16] M. Hell, T. Johansson, A. Maximov and W. Meier, “The Grain family of stream ciphers”, New Stream Cipher Designs - The eSTREAM Finalists, LNCS 4986, pp. 179–190, Springer, 2008.

[17] C. D. Canniere and B. Preneel, “Trivium”, Lecture Notes in Computer Science , LNCS 4986, pp. 244– 266, Springer, 2008.

[18] T. Isobe, “A single-key attack on the full GOST block cipher”, FSE’11 , LNCS 6733, pp. 290–305, 2011.

[19] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, “HIGHT: A new block cipher suitable for low-resource device”, CHES’06 , LNCS 4249, pp. 46–59, Springer-Verlag, 2006.

[20] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin and C. Vikkelsoe, “PRESENT: An ultra-lightweight block cipher”, CHES’07 , LNCS 4727, pp. 450–466, Springer-Verlag, 2007.

[21] N. Arora and Y. Gigras, “FPGA Implementation of Low Power and High Speed Hummingbird Cryptographic Algorithm”, International Journal of Computer Applications (0975 – 8887) Vol. 92 – No. 16, 2014.

[22] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, “TWINE: A Lightweight, Versatile Block Cipher,” NEC Corporation, 1753 Shimonumabe, Nakahara-Ku, Kawasaki, Japan, 2012.

[23] P. Coussy and A. Morawiec, “High Level Synthesis From Algorithm to Digital Circuit,” Springer, 2008.

[24] A. Aysu, E. Gulcan and P. Schaumont, “SIMON says: Break area records of block ciphers on FPGAs,” Embedded Systems Letters, IEEE 6(2), 37-40, 2014.

[25] F. Mace, F.X. Standaert, and J.J. Quisquater, “Fpga implementation(s) of a scalable encryption algorithm,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16(2), pp. 212-216, 2008.

[26] R. Chaves, “Compact clefia implementation on fpgas,” Embedded Systems Design with FPGAs, pp. 225-243. Springer, 2013.

[27] J.P. Kaps, “Chai-tea, cryptographic hardware implementations of xtea,” INDOCRYPT 2008, Lecture Notes in Computer Science, vol. 5365, pp. 363-375. Springer, 2008.

[28] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, “Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates,” State of the Art of Stream Ciphers Workshop, SASC 2008, Lausanne, Switzerland. pp. 151-162, 2008.