

Fast Implementation of Stream Cipher K2 on FPGA

Yuto Nakano, Kazuhide Fukushima, Shinsaku Kiyomoto, and Yutaka Miyake

Abstract—K2 is a software-oriented stream cipher that is being standardized in ISO/IEC 18033-4 and has been evaluated in the CRYPTREC project. Existing research has shown that K2 is secure against various kinds of attacks and has demonstrated high performance in software implementations. On the other hand, few studies have dealt with the hardware implementation of K2. One of the studies undertaken by the designers of K2, in which the optimization of the hardware implementation was considered, showed high throughput for K2. However, the implementation on hardware has not yet been optimized. In this paper, we improve the implementation of K2 on Xilinx Virtex-5 by considering four different implementations based on the reference implementation. The result shows that the highest performance is 15168 Mbps; about three times faster than the reference, and the smallest circuit size is 646 slices; about half that of the reference.

Keywords—Stream cipher, K2, Hardware implementation, FPGA

I. INTRODUCTION

STREAM ciphers are widely used to encrypt/decrypt private information in a variety platforms, from resource constrained devices to high-end PCs. Before encryption/decryption of the data, stream ciphers must be initialized with a secret key and a publicly known initial vector (IV). After the initialization, it produces a stream of key values called *keystream*. The keystream is exclusive-ored (XORed) with plaintext to produce a ciphertext. To decrypt, the cipher is initialized with the same key and IV, and the ciphertext is XORed with the generated keystream.

K2 [1] is a word-oriented stream cipher using dynamic feedback control as irregular clocking. K2 has a dynamic feedback control mechanism for the byte-level feedback function of FSRs and realizes fast encryption/decryption for software implementations. Existing research has shown that K2 is a secure and high-performance stream cipher in software implementations and can be used in many applications. Hardware implementation of K2 is also considered [2]. In [2], Kiyomoto *et al.* proposed two different implementations; a compact and a fast implementations.

In this paper, we consider a further optimized implementation of K2, and show the evaluation results for FPGA implementations. We propose four different designs in order to optimize both the performance and circuit size. First, we try to improve the clock frequency by dividing the critical paths and then consider paralleling the non linear function. The evaluation results suggested that K2 is faster than existing ciphers and attains a reasonable level of efficiency. The fastest implementation achieves 15168 Mbps and the most compact implementation uses only 646 slices.

The rest of the paper is organized as follows. A description of K2 is provided in Section II. In Section III, we consider four

kinds of implementation in order to improve the performance of K2 and evaluate the implementations in Section IV. Finally Section V concludes this paper.

II. STREAM CIPHER K2

K2 was proposed by Kiyomoto *et al.* in 2007 and has a similar structure to SNOW-family stream ciphers [3], [4]. The K2 stream cipher consists of two feedback shift registers (FSRs), *FSR-A* and *FSR-B*, and a nonlinear function with four internal registers *R1*, *R2*, *L1*, and *L2*, as shown in Figure 1.

A. Linear Feedback Shift Registers

The size of each register is 32 bits. *FSR-A* has five registers, and *FSR-B* has eleven registers. Two bits are extracted from *FSR-A* to determine a feedback polynomial of *FSR-B*.

Let β be the roots of the primitive polynomial;

$$x^8 + x^7 + x^6 + x + 1 \in GF(2)[x].$$

A byte string y denotes $(y_7, y_6, \dots, y_1, y_0)$, where y_7 is the most significant bit and y_0 is the least significant bit. y is represented by

$$y = y_7\beta^7 + y_6\beta^6 + \dots + y_1\beta + y_0.$$

In the same way, let γ , δ , ζ be the roots of the primitive polynomials,

$$x^8 + x^5 + x^3 + x^2 + 1 \in GF(2)[x],$$

$$x^8 + x^6 + x^3 + x^2 + 1 \in GF(2)[x],$$

$$x^8 + x^6 + x^5 + x^2 + 1 \in GF(2)[x],$$

respectively.

Let α_0 be the root of the irreducible polynomial of degree four

$$x^4 + \beta^{24}x^3 + \beta^3x^2 + \beta^{12}x + \beta^{71} \in GF(2^8)[x].$$

A 32-bit string Y denotes (Y_3, Y_2, Y_1, Y_0) , where Y_i is a byte string and Y_3 is the most significant byte. Y is represented by

$$Y = Y_3\alpha_0^3 + Y_2\alpha_0^2 + Y_1\alpha_0 + Y_0.$$

Let α_1 , α_2 , α_3 be the roots of the irreducible polynomials of degree four;

$$x^4 + \gamma^{230}x^3 + \gamma^{156}x^2 + \gamma^{93}x + \gamma^{29} \in GF(2^8)[x],$$

$$x^4 + \delta^{34}x^3 + \delta^{16}x^2 + \delta^{199}x + \delta^{248} \in GF(2^8)[x],$$

$$x^4 + \zeta^{157}x^3 + \zeta^{253}x^2 + \zeta^{56}x + \zeta^{16} \in GF(2^8)[x],$$

Y. Nakano, K. Fukushima, S. Kiyomoto, and Y. Miyake are with KDDI R&D Laboratories Inc., 2-1-15 Ohara Fujimino Saitama Japan
e-mail: {yuto, ka-fukushima, kiyomoto, miyake}@kddilabs.jp

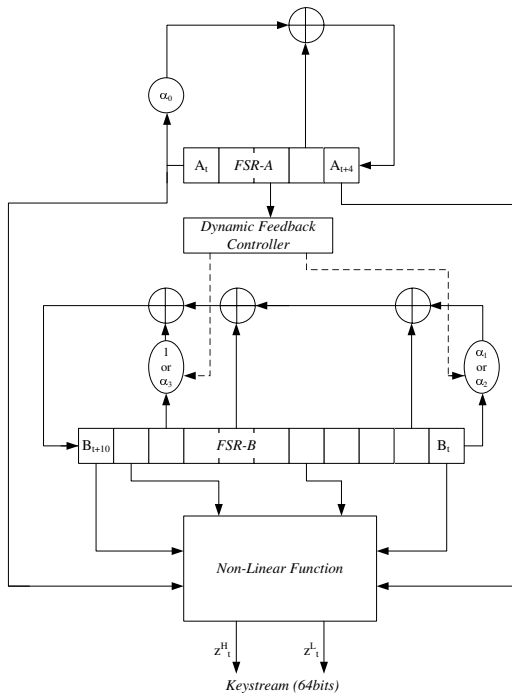


Fig. 1. K2 Stream Cipher

respectively.

The feedback polynomials $f_A(x)$, and $f_B(x)$ of *FSR-A* and *FSR-B*, respectively, are as follows;

$$\begin{aligned} f_A(x) &= \alpha_0 x^5 + x^2 + 1, \\ f_B(x) &= (\alpha_1^{cl1_t} + \alpha_2^{1-cl1_t} - 1)x^{11} \\ &\quad + x^{10} + x^5 + \alpha_3^{cl2_t} x^3 + 1. \end{aligned}$$

Let $cl1$ and $cl2$ be the sequences describing the output of the dynamic feedback controller. The outputs at time t are defined in terms of some bits of *FSR-A*. Let A_x denote the output of *FSR-A* at time x , and $A_x[y] = \{0, 1\}$ denote the y -th bit of A_x , where $A_x[31]$ is the most significant bit. Then $cl1$ and $cl2$ (called clock control bits) are described as follows,

$$cl1_t = A_{t+2}[30], \quad cl2_t = A_{t+2}[31].$$

Both $cl1_t$ and $cl2_t$ are binary variables; more precisely, $cl1_t = \{0, 1\}$, and $cl2_t = \{0, 1\}$. Stop-and-go clocking is effective in terms of computational cost, because no computation is required in the case of 0. However, the feedback function has no transformation for feedback registers with a probability $1/4$ where all clockings are stop-and-go clockings. Thus, we use two types of clocking for the feedback function. *FSR-B* is defined by a primitive polynomial, where $cl2_t = 0$.

B. Nonlinear Function

The nonlinear function of K2 is fed the values of two registers of *FSR-A* and four registers of *FSR-B* and that of internal registers $R1$, $R2$, $L1$, $L2$, and outputs 64 bits of the keystream every cycle. Fig. 2 shows the non-linear function

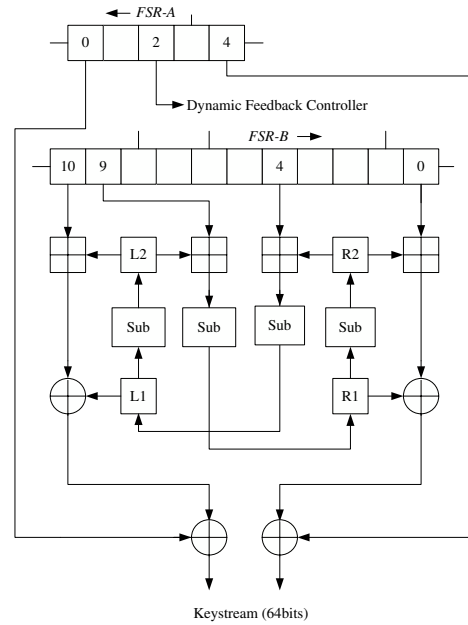


Fig. 2. Nonlinear Function of K2

of K2. The nonlinear function includes four substitution steps that are indicated by *Sub*.

The *Sub* step divides the 32-bit input string into four 1-byte strings and applies a non-linear permutation to each byte using an 8-to-8 bit substitution, and then applies a 32-to-32 bit linear permutation. The 8-to-8 bit substitution is the same as s-boxes of AES [5], and the permutation is the same as AES *Mix Column* operation. The 8-to-8 bit substitution consists of two functions: g and f . The function g calculates the multiplicative inverse modular of the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ without 0×00 , and 0×00 is transformed to itself (0×00). The function f is an affine transformation defined by;

$$\begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \\ 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

where $a = (a_7, \dots, a_0)$ is the input and $b = (b_7, \dots, b_0)$ is the output, and a_0 and b_0 are the least significant bit (LSB).

Let C be (c_3, c_2, c_1, c_0) and output D be (d_3, d_2, d_1, d_0) , where c_i, d_i are 8-bit values. The linear permutation $D = p(C)$ is described as follows;

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

in $GF(2^8)$ of the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$.

C. Keystream Output

Let keystream at time t be $Z_t = (z_t^H, z_t^L)$ (each z_t^x is a 32-bit value, and z_t^H is a higher string). The keystream z_t^H , z_t^L is calculated as follows:

$$\begin{aligned} z_t^L &= B_t \boxplus R2_t \oplus R1_t \oplus A_{t+4} \\ z_t^H &= B_{t+10} \boxplus L2_t \oplus L1_t \oplus A_t \end{aligned}$$

where A_x and B_x denote outputs of *FSR-A* and *FSR-B* at time x , and $R1_x$, $R2_x$, $L1_x$, and $L2_x$ denote the internal registers at time x . The symbol \oplus denotes bit-wise exclusive-or operation and the symbol \boxplus denotes a 32-bit addition. Finally, the internal registers are updated as follows;

$$\begin{aligned} R1_{t+1} &= \text{Sub}(L2_t \boxplus B_{t+9}), \quad R2_{t+1} = \text{Sub}(R1_t) \\ L1_{t+1} &= \text{Sub}(R2_t \boxplus B_{t+4}), \quad L2_{t+1} = \text{Sub}(L1_t) \end{aligned}$$

where $\text{Sub}(X)$ is an output of the *Sub* step for X .

D. Initialization Process

The initialization process of K2 consists of two steps, a key loading step and an internal state initialization step. First, an initial internal state is generated from a 128-bit initial key and a 128-bit initial vector (IV) using the key scheduling algorithm. The key scheduling algorithm is similar to the round key generation function of AES and the algorithm extends the 128-bit initial key to 384 bits. The key scheduling algorithm is described as;

$$K_i = \begin{cases} IK_i & (0 \leq i \leq 3) \\ K_{i-4} \oplus \text{Sub}((K_{i-1} \ll 8) \oplus (K_{i-1} \gg 24)) \\ \oplus \text{Rcon}[i/4 - 1] & (i = 4n) \\ K_{i-4} \oplus K_{i-1} & (i \neq 4n) \end{cases}, \quad (1)$$

where $IK = (IK_0, IK_1, IK_2, IK_3)$ is the initial key, i is a positive integer $0 \leq i \leq 11$, and n is a positive integer. The function $\text{Sub}(X)$ in the key scheduling algorithm is the same as that in the non-linear function. This function is different from the round key generation function of AES, while the other part of the key scheduling algorithm is the same as the AES round key generation. $\text{Rcon}[i]$ denotes $(x^i \bmod x^8 + x^4 + x^3 + x + 1, 0x00, 0x00, 0x00)$ and x is $0x02$. The internal state is initialized with K_i and $IV = (IV_0, IV_1, IV_2, IV_3)$ as follows:

$$\begin{aligned} A_m &= K_{4-m} \quad (m = 0, \dots, 4), \quad B_0 = K_{10}, \quad B_1 = K_{11}, \\ B_2 &= IV_0, \quad B_3 = IV_1, \quad B_4 = K_8, \quad B_5 = K_9, \quad B_6 = IV_2, \\ B_7 &= IV_3, \quad B_8 = K_7, \quad B_9 = K_5, \quad B_{10} = K_6. \end{aligned}$$

The internal registers, $R1$, $R2$, $L1$, and $L2$ are set to $0x00$. After the above processes, the cipher clocks 24 times

($j = 1, \dots, 24$), updating the internal states. The internal states A_{j+4} B_{j+10} are also updated as follows:

$$\begin{aligned} A_{j+4} &= \alpha_0 A_{j-1} \oplus A_{j+2} \oplus z_{j-1}^L \\ B_{j+10} &= (\alpha_1^{cl_{1j-1}} + \alpha_2^{1-cl_{1j-1}} - 1) B_{j-1} \oplus B_j \oplus B_{j+5} \\ &\quad \oplus \alpha_3^{cl_{2j-1}} B_{j+7} \oplus z_{j-1}^H \end{aligned}$$

The recommended maximum number of cycles for K2 without re-initializing and re-keying is 2^{58} cycles (2^{64} keystream bits).

III. HARDWARE IMPLEMENTATION OF K2

In this section, we consider four different kinds of implementation to obtain an efficient and fast implementation. We use Xilinx ISE12.1 for the logic synthesis targeting Virtex 5.

A. Reference Implementation

The reference implementation of K2 was available at CRYPTREC¹[6]. The reference implementation is optimized without compromising the readability.

The reference implementation achieves 5440 Mbps and the circuit size is 1210 slices. Figure 3 shows the block diagram of the reference implementation. The initialization phase and the keystream generation phase share FSRs and the nonlinear part. First, the extended key is generated from the initial key, then the extended key and the IV are set to two FSRs by setting a signal "SCHEM_STAT" to 1. Four registers of the nonlinear part are also set to 0s at the same timing. If a signal "doing" equals 1, the cipher is clocked to update its internal state.

B. Implementation 1

We make three modifications from the reference in this section; namely inserting a flip-flop, modifying a modular addition, and constructing a table for 8-bit permutation.

The worst path of the reference implementation is during initialization. We had a timing error when we assigned 86 MHz for the clock frequency in the reference implementation. Therefore, we need to divide this critical path in order to achieve higher performance the cipher. In the initialization process, the 128-bit initial key is expanded into the 384-bit expanded key, which involves the nonlinear functions when $i = 4n$ in Equation (1). This nonlinear part is the worst path. We insert a flip-flop to divide the worst path in order to improve the clock frequency. By inserting the flip-flop, the throughput is improved to 8000 Mbps, and used area is reduced to 1108 slices.

Then, we obtain the new worst path which involves a modular addition. Hence we can improve the performance by modifying the modular addition. In the reference implementation, the modular addition was implemented as a combination of a full adder and a half adder. Higher performance can be accomplished by simply denoting '+' instead of the combination of adders. As a result, the throughput is now 9088 Mbps, and used area is reduced to 1025 slices.

¹CRYPTography Research and Evaluation Committees, <http://www.cryptrec.go.jp/english/index.html>

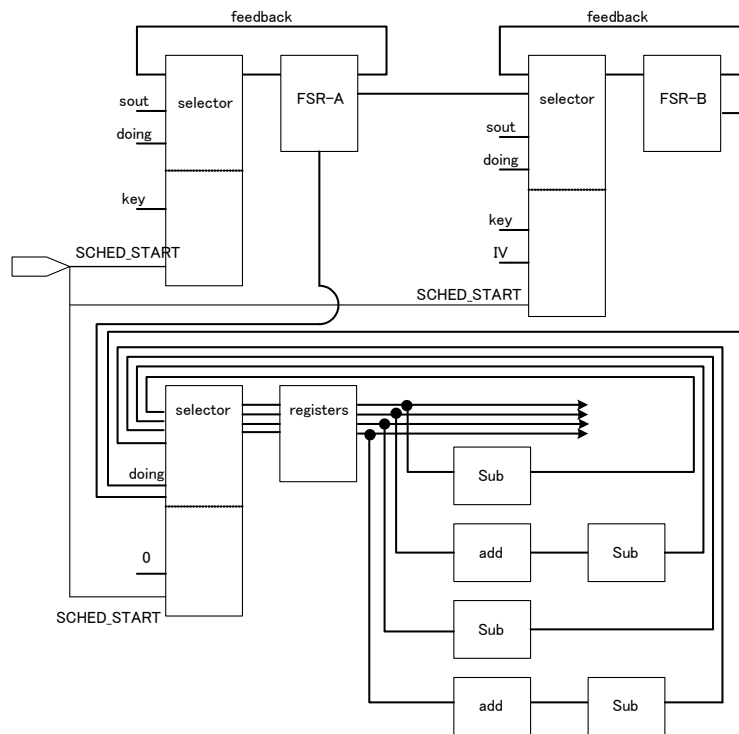


Fig. 3. Block Diagram of Reference Implementation

The critical path now involves the *Sub* step. The *Sub* step consists of an 8-to-8 bit substitution and a 32-to-32 bit permutation. First, the 32-bit input string is divided into four 1-byte strings and applies a nonlinear permutation to each byte using an 8-to-8 bit substitution, and then applies the 32-to-32 bit permutation. Since the nonlinear permutation is the 8-bit operation, we can construct a table which returns the result of the nonlinear permutation. Then, the throughput is improved to 14848 Mbps and used area is compressed to 646 slices.

C. Implementation 2

From the result of Section III-B, the worst paths are now given by (see Figure 2),

- $B_4 \rightarrow$ modular addition \rightarrow *Sub* \rightarrow $L1$,
- $B_9 \rightarrow$ modular addition \rightarrow *Sub* \rightarrow $R1$.

In this section, we consider dividing these critical paths by adding two registers, $L0$ and $R0$, between the addition and *Sub*.

Two registers B_4 and B_9 are input to the modular addition, and the values of B_4 and B_9 at time t are the same as those of B_5 and B_{10} at time $t+1$, respectively. Hence, we can expect to achieve higher performance by allocating two selectors which select the input to the modular addition from B_4 or B_5 and B_9 or B_{10} . We also need another two selectors between the $L2$, $R2$ and *Sub* because of the selectors we have allocated between the FSR-B and the addition.

The diagram of this implementation is given in Figure 4.

This method requires four selectors and two registers in total, this evolves an increase in the number of critical paths,

and neither throughput nor clock are improved. The throughput is 14592 Mbps and used area is 740 slices.

D. Implementation 3

The basic idea of this method is the same as that in Section III-C. The addition and the registers are allocated in parallel, and the result of the operation is selected by the selectors to input to the *Sub* step. We can reduce the number of selectors by applying this implementation, and the throughput is improved to 15168 Mbps. This implementation uses 777 slices.

The diagram of this implementation is given in Figure 5.

E. Implementation 4

The structure of this implementation is the same as that in Section III-D, we use Coregenerator vsn12.1 to generate the core of the modular addition and the register. The core itself is quite fast, however, the core is fixed and cannot be optimized in combination with the other part. Therefore, it results in performance similar to that in Section III-D. The throughput is 15168 Mbps and used area is 770 slices.

The diagram of this implementation is given in Figure 6.

F. Pipeline

We also consider pipelining by implementing the nonlinear part in parallel. We considered implementing the non linear function in double and quad, however, waiting occurs until all the data are ready. The data required to generate keystream are dependent on the internal state at previous clocks, hence,

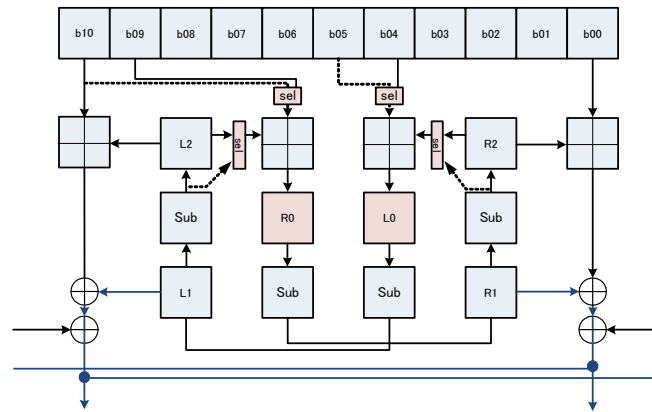


Fig. 4. Implementation 2

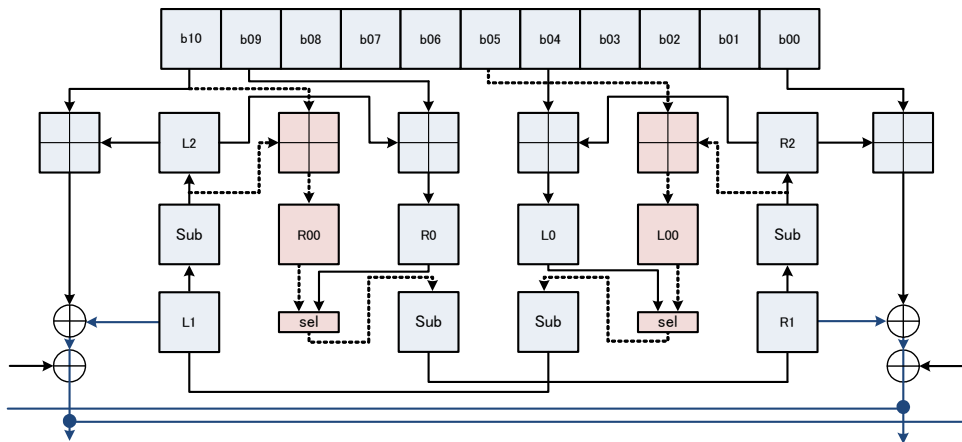


Fig. 5. Implementation 3

we can not always obtain the required data at the right timing. Therefore, it is difficult to improve the throughput by pipelining.

IV. EVALUATION RESULT

In this section, we present the evaluation results of hardware implementations using an FPGA simulator. We implemented the K2 stream cipher in the Xilinx Virtex 5 FPGA. We use Xilinx ISE 12.1 for post-place and route simulation and static timing analysis.

Table I shows the evaluation result of our implementations. Data rate, clock frequency, throughput, and area denote the number of bits that the cipher generates for each cycle, maximum clock frequency of the circuit, maximum throughput of encryption/decryption estimated by the maximum clock frequency, and the number of slices of the circuit, respectively. We also evaluated the efficiency of the implementations by dividing the throughput by the area, which is the same index used in previous studies.

Comparisons with previously published evaluation results of FPGA implementations for other ciphers are shown in Table II. The key length column denotes the key length of

each algorithm. There is a trade-off between the performance of the algorithm and its security level (effective key length). Thus, we compare an index "Normalized Efficiency" as well as in [2]. The index is calculated as $(T \times L)/(S \times 256)$, where T , S , and L denote the throughput, the area size, and the key length respectively. We conclude that K2 is the most efficient algorithm among these algorithms.

The throughput of our implementation is much faster than other block/stream ciphers and the previous implementation. In terms of efficiency, the implementations of K2 are far more efficient compared with other stream ciphers, and is about double that of AES implementations. The FPGA implementation of K2 is suitable for applications that require high speed encryption/decryption and also for resource constrained devices.

V. CONCLUSION

This paper presented the evaluation results of high speed implementations of K2 on Virtex 5. We achieved approximately 15 Gbps in the Virtex 5 FPGA implementation, which is about 3 times faster than the reference implementation and that achieved in the previous work of [2]. Remarkably, this

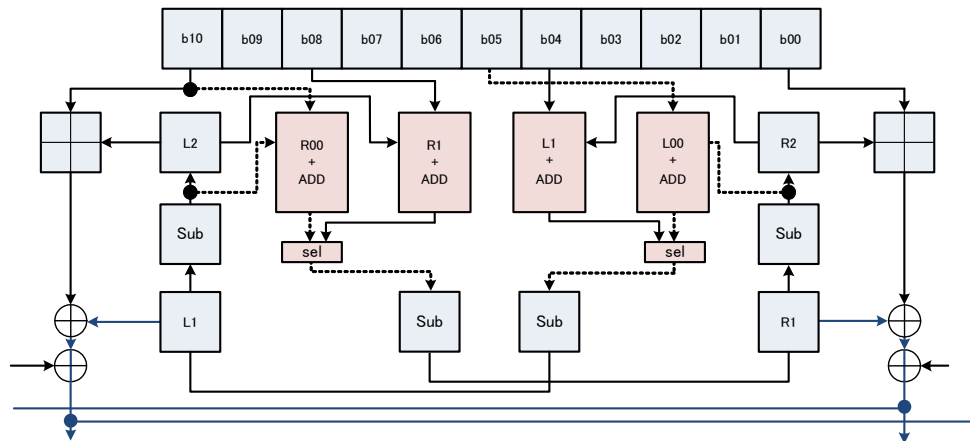


Fig. 6. Implementation 4

TABLE I
SUMMARY OF OUR IMPLEMENTATIONS ON VIRTEX-5 FPGA

Design	Clock Freq. (MHz)	Throughput (Mbps)	Area (slice)	Throughput/Area (Mbps/slice)
Reference	85	5440	1210	4.5
Implementation 1	232	14848	646	22.98
Implementation 2	228	14592	740	19.72
Implementation 3	237	15168	777	19.52
Implementation 4	237	15168	770	19.70

high speed implementation is realized with only 770 slices, which is about half that of the reference implementation and about one-third of [2]. However, while we used a different FPGA from [2], the comparison is nevertheless informative.

Furthermore, we evaluated the efficiency of the implementations using throughput per area. The ratio in the most efficient implementation is 22.98, which is quite efficient compared to other hardware oriented stream ciphers such as Trivium and Grain. We also evaluated the efficiency in terms of “normalized efficiency”, which enables us to make a more objective comparison since key length often affects the performance of the cipher. The evaluation results suggest that the FPGA implementation of K2 is suitable for applications using high speed encryption/decryption.

REFERENCES

- [1] S. Kiyomoto, T. Tanaka, and K. Sakurai, “K2: A stream cipher algorithm using dynamic feedback control,” in *SECURITY*, 2007, pp. 204–213.
- [2] S. Kiyomoto, T. Tanaka, and K. Sakurai, “FPGA-Targeted Hardware Implementations of K2,” in *SECURITY*, E. Fernández-Medina, M. Malek, and J. Hernando, Eds. INSTICC Press, 2008, pp. 270–277.
- [3] P. Ekdahl and T. Johansson, “SNOW -a new stream cipher,” *The NESSIE submission paper*, 2000.
- [4] P. Ekdahl and T. Johansson, “A New Version of the Stream Cipher SNOW,” in *SAC*, ser. LNCS, K. Nyberg and H. M. Heys, Eds., vol. 2595. Springer, 2002, pp. 47–61.
- [5] J. Daemen and V. Rijmen, *The Design of Rijndael, Information Security and Cryptography, Texts and Monographs*. Springer Verlag, 1998.
- [6] CRYPTREC, “Call for attack to evaluate the cryptographic techniques submitted in FY 2009,” http://www.cryptrec.go.jp/english/topics/cryptrec_20101001_callforattack.html.
- [7] M. D. Galanis, P. Kitsos, G. Kostopoulos, and O. Koufopavlou, “Comparison of the performance of stream ciphers for wireless communications,” in *Proc. of CCCT'04*, 2004, pp. 113–118.
- [8] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, “Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates,” in *SASC 2008 Workshop Record*. eSTREAM Project, 2008, pp. 151–162.
- [9] M. Kasper, S. Kumar, K. Lemke-Rust, and C. Paar, “A compact implementation of edon80,” *eSTREAM Report 2006/057*, 2006.
- [10] K. Gaj, G. Southern, and R. Bachimanchi, “Comparison of hardware performance of selected Phase II eSTREAM candidates,” in *SASC, eSTREAM, ECRYPT Stream Cipher Project, Report*, 2007.
- [11] P. Bulens, K. Kalach, F. X. Standaert, and J. J. Quisquater, “FPGA implementations of estream phase-2 focus candidates with hardware profile,” in *SASC 2007 Workshop Record*. eSTREAM Project, 2007, pp. 205–214.
- [12] D. Stefan and C. Mitchell, “On the Parallelization of the MICKEY-128 2.0 Stream Cipher,” in *SASC, eSTREAM, ECRYPT Stream Cipher Project, Report*, 2007.
- [13] T. Good, W. Chelton, and M. Benaissa, “Review of stream cipher candidates from a low resource hardware perspective,” in *SASC 2006 Workshop Record*. eSTREAM Project, 2006.
- [14] D. Stefan, “Hardware Framework for the Rabbit Stream Cipher,” in *Inscrypt*, ser. Lecture Notes in Computer Science, F. Bao, M. Yung, D. Lin, and J. Jing, Eds., vol. 6151. Springer, 2009, pp. 230–247.
- [15] P. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou, “Hardware implementation of the rc4 stream cipher,” in *Micro-NanoMechatronics and Human Science, 2003 IEEE International Symposium on*, vol. 3, dec. 2003, pp. 1363 – 1366 Vol. 3.
- [16] P. Léglise, F.-X. Standaert, G. Rouvroy, and J.-J. Quisquater, “Efficient Implementation of Recent Stream Ciphers on Reconfigurable Hardware Devices,” in *26th Symposium on Information Theory in the Benelux*, 5 2005, pp. 261–268.
- [17] Z. Liu, lingchen Zhang, J. Jing, and W. Pan, “Efficient Pipelined Stream Cipher ZUC Algorithm in FPGA,” in *ZUC Workshop*, 2010.
- [18] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, “Implementation of the AES-128 on Virtex-5 FPGAs,” in *AFRICACRYPT*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 5023. Springer, 2008, pp. 16–26.
- [19] F.-X. Standaert, “Secure and efficient use of reconfigurable hardware devices in symmetric cryptography,” Ph.D. dissertation, UCL, 2004.

TABLE II
COMPARISONS WITH FPGA IMPLEMENTATIONS OF OTHER CIPHERS

Algorithm	Key Length (bit)	Throughput (Mbps)	Area (slice)	Through./Area (Mbps/slice)	Normalized Efficiency	Device
A5/1 [7]	64	188.3	32	5.88	1.47	Virtex-II
DECIM v2 [8]	80	46.25	80	0.58	0.18	Spartan-3
E0 [7]	128	189	895	0.21	0.11	Virtex-II
Edon80 [9]	80	5.58	45	0.08	0.02	Virtex 4
F-FCSR-H v2 [8]	80	1104	342	3.23	1.01	Spartan-3
Grain [10]	80	2480	356	6.97	2.18	Spartan-3
Grain-128 [11]	128	180	48	3.77	1.88	Virtex-II
MICKEY-128 [12]	128	560	392	1.43	0.71	Virtex-II Pro
Phelix [13]	256	750	1077	0.70	0.70	Spartan-II
Pomaranich [8]	80	49	648	0.08	0.03	Spartan-3
Rabbit [14]	128	25620	1163	22.03	11.01	Virtex-5
RC4 [15]	256	176	138	1.28	1.28	Virtex-II
SNOW 2.0 [16]	256	5659	1015	5.57	5.57	Virtex-II
Trivium [10]	80	12160	388	31.34	9.79	Spartan-3
ZUC [17]	128	7111	575	12.3	6.18	Virtex-5
AES-128 [18]	128	4100	400	10.2	5.13	Virtex-5
IDEA [19]	128	6800	9793	0.69	0.35	Virtex-II
3DES [19]	168	917	604	1.51	1.00	Virtex-II
K2 <i>Compact</i> [2]	256	1920	2133	0.90	0.09	Spartan-II
K2 <i>Normal</i> [2]	256	4090	3067	1.33	1.33	Spartan-3
K2 <i>Double-keystream</i> [2]	256	4864	5295	0.92	0.92	Spartan-3
K2 <i>Quad-keystream</i> [2]	256	5223	9161	0.57	0.57	Spartan-3
K2 <i>Reference</i> [6]	128	5440	1210	4.5	2.28	Virtex 5
K2 <i>Implementation 1</i> (this paper)	128	14848	646	22.98	11.49	Virtex 5
K2 <i>Implementation 2</i> (this paper)	128	14592	740	19.72	9.86	Virtex 5
K2 <i>Implementation 3</i> (this paper)	128	15168	777	19.52	9.76	Virtex 5
K2 <i>Implementation 4</i> (this paper)	128	15168	770	19.70	9.85	Virtex 5