FPGA-TARGETED HARDWARE IMPLEMENTATIONS OF K2

Shinsaku Kiyomoto, Toshiaki Tanaka
KDDI R & D Laboratories Inc., 2-1-15 Ohara Fujimino-shi Saitama 356-8502, Japan

Kouichi Sakurai
Kyushu University, 744 Motooka Nishi-ku Fukuoka 819-0395, Japan

Keywords: Stream Cipher, K2, Pseudorandom Generator, Hardware Implementation, FPGA.

Abstract: K2 is a new type of word oriented stream cipher that has dynamic feedback control. Existing research has shown that K2 v2.0 is a high performance stream cipher in software implementations and can be used in several applications. However, no evaluation results for its performance in hardware implementations have been published. In this paper, we presented two hardware implementations of K2 v2.0: a high speed implementation and a compact implementation. We then show the evaluation results on FPGA implementation simulations. The implementations of K2 demonstrated high efficiency compared with other stream ciphers, with K2 being 4-10 times higher than AES implementations. We think that the FPGA implementation of K2 is suitable for applications using high speed encryption/decryption.

1 INTRODUCTION

Stream ciphers are used extensively to provide a reliable, efficient method for secure communications. A basic stream cipher uses several independent linear feedback shift registers (LFSRs) together with nonlinear functions in order to produce a keystream. The keystream is then XORed with plaintext to produce a ciphertext. Recently, word-oriented stream ciphers have been developed in order to improve the performance of software implementations. In the NESSIE project [1], many word-oriented stream ciphers were proposed, such as SNOW(Ekdahl and Johansson, 2000) and SOBER(Rose and Hawkes, 1999), and demonstrated good performance in software.

The need for secure data exchange becomes important not only for high-end PCs but also low-end customer products. Most of low-end devices require an additional hardware implementations for encryption/decryption of transaction data. Efficient hardware implementations of stream ciphers are important in both high-performance and low-power applications. On the eSTREAM project [2], many evaluation results of hardware performances of stream ciphers have been reported.

K2 (Kiyomoto et al., 2007b) (Kiyomoto et al., 2007a) is a new word-oriented stream cipher using dynamic feedback control as irregular clocking. The stream cipher has a dynamic feedback control mechanism for the byte-level feedback function of FSRs and realizes fast encryption/decryption for software implementations. Several cryptographic algorithms have been considered hardware implementations (Rodríguez-Henríquez et al., 2007). Existing research has shown that K2 v2.0 is a secure and high-performance stream cipher in software implementations and can be used in several applications. However, no evaluation results for the performance of hardware implementations have been published. In this paper, we presented two hardware implementations of K2 v2.0: a high speed implementation and a compact implementation. We then show their evaluation results on FPGA implementations. The evaluation results suggested that the cipher is faster than existing ciphers and attains a reasonable level of ef-

[1] New European Schemes for Signatures, Integrity, and Encryption (NESSIE), https://www.cosic.esat.kuleuven.be/nessie/

[2] ECRYPT eSTREAM http://www.ecrypt.eu.org/stream/

Kiyomoto S., Tanaka T. and Sakurai K. (2008).
FPGA-TARGETED HARDWARE IMPLEMENTATIONS OF K2.
In *Proceedings of the International Conference on Security and Cryptography*, pages 270-277
DOI: 10.5220/0001917002700277

ficiency. We think that the FPGA implementation of K2 is suitable for applications using high speed encryption/decryption.

2 STREAM CIPHER K2 V2.0

In this section, we describe the stream cipher algorithm K2 v2.0 (Kiyomoto et al., 2007a), which has a dynamic feedback control mechanism.

2.1 Linear Feedback Shift Registers

The K2 v2.0 stream cipher consists of two feedback shift registers (FSRs), *FSR-A* and *FSR-B*, a non-linear function with four internal registers $R1$, $R2$, $L1$, and $L2$, and a dynamic feedback controller as shown in Fig. 1. *FSR-B* is a dynamic feedback shift register. The size of each register is 32 bits. *FSR-A* has five registers, and *FSR-B* has eleven registers. 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 $(y_7, y_6, \ldots, y_1, y_0)$, where y_7 is the most significant bit and y_0 is the least significant bit. y is represented by

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

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

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

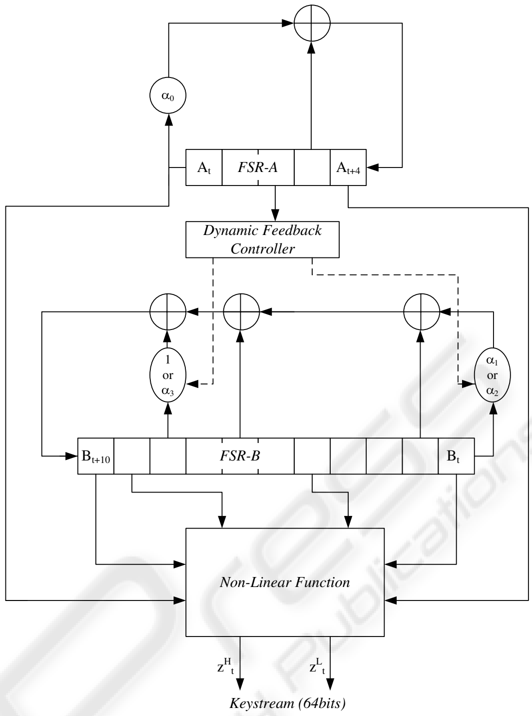

Figure 1: K2 v2.0 Stream Cipher.

$$x^4 + \delta^{34}x^3 + \delta^{16}x^2 + \delta^{199}x + \delta^{248} \in GF(2^8)[x]$$
$$x^4 + \zeta^{157}x^3 + \zeta^{253}x^2 + \zeta^{56}x + \zeta^{16} \in GF(2^8)[x]$$

respectively.

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

$$f_A(x) = \alpha_0x^5 + x^2 + 1$$
$$f_B(x) = (\alpha_1^{cl1_t} + \alpha_2^{1-cl1_t} - 1)x^{11} + x^{10} + x^5 + \alpha_3^{cl2_t}x^3 + 1$$

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 yth bit of A_x, where $A_x[31]$ is the most significant bit of A_x. 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 $cl2_t = 0$.

2.2 Nonlinear Function

The non-linear function of K2 v2.0 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 of K2 v2.0. The non-linear 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 (Daemen and Rijmen, 1998), 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 g calculates the multiplicative inverse modulo of the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ without 0x00, and 0x00 is transformed to itself (0x00). 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 $a = (a_7, \ldots, a_0)$ is the input and $b = (b_7, \ldots, b_0)$ 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 $m(x) = x^8 + x^4 + x^3 + x + 1$.

2.3 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:

$$z_t^L = B_t \boxplus R2_t \oplus R1_t \oplus A_{t+4}$$

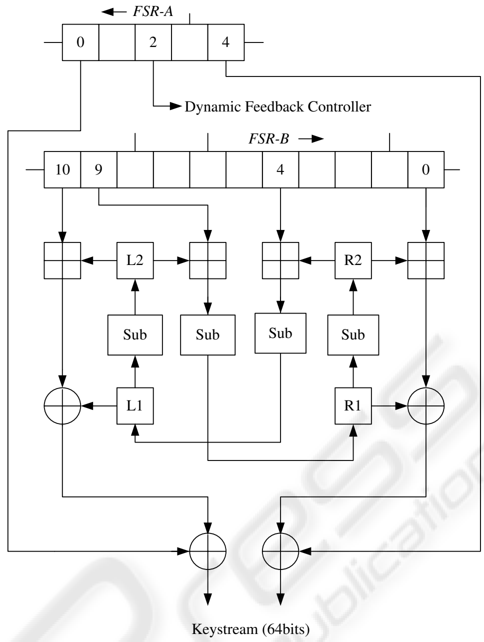

Figure 2: Non-Linear Function of K2 v2.0.

$$z_t^H = B_{t+10} \boxplus L2_t \oplus L1_t \oplus A_t$$

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 bitwise exclusive-or operation and the symbol $\boxplus$ denotes a 32-bit addition. Finally, the internal registers are updated as follows;

$$R1_{t+1} = Sub(L2_t \boxplus B_{t+9}), \quad R2_{t+1} = Sub(R1_t)$$

$$L1_{t+1} = Sub(R2_t \boxplus B_{t+4}), \quad L2_{t+1} = Sub(L1_t)$$

where $Sub(X)$ is an output of the *Sub* step for X.

2.4 Initialization Process

The initialization process of K2 v2.0 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, a 192-bit initial key, or a 256-bit initial key and a 128-bit initial vector (IV) by 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, the 192-bit initial key or the 256-bit initial key to 384 bits. The key scheduling algorithm for a 128-bit key is described as

$$K_i = \begin{cases} IK_i & \\ (0 \le i \le 3) & \\ K_{i-4} \oplus Sub((K_{i-1} \lll 8) \oplus (K_{i-1} \ggg 24)) & \\ \oplus Rcon[i/4-1] & (i = 4n) \\ K_{i-4} \oplus K_{i-1} & (i \ne 4n) \end{cases}$$

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

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

The internal registers, $R1$, $R2$, $L1$, and $L2$ are set to 0x00. After the above processes, the cipher clocks 24 times (j = 1, ..., 24), updating the internal states. The internal states A_{j+4} B_{j+10} are also updated as follows:

$$\begin{aligned} A_{j+4} =& \alpha_0 A_{j-1} \oplus A_{j+2} \oplus z^L_{j-1} \\ B_{j+10} =& (\alpha_1^{cl1_{j-1}} + \alpha_2^{1-cl1_{j-1}} - 1) B_{j-1} \oplus B_j \oplus B_{j+5} \\ & \oplus \alpha_3^{cl2_{j-1}} B_{j+7} \oplus z^H_{j-1} \end{aligned}$$

The recommended maximum number of cycles for K2 v2.0 without re-initializing and re-keying is 2^{58} cycles (2^{64} keystream bits).

3 HARDWARE IMPLEMENTATION OF K2

We considered two types of hardware implementations of K2 v2.0: a high speed implementations and a compact size implementation.

3.1 High Speed Implementation

K2 produces a 64-bit keystream for each cycle. To improve the performance of hardware implementations, we considered both a double-keystream and quad-keystream implementation of hardware; the double-keystream implementation produces a 128-bit keystream for each clock and the quad-keystream implementation produces a 256-bit keystream for each clock. The generated keystream bits are exclusive-ored with a plaintext/ciphertext. To implement the double-keystream circuit, one additional register for LFSR-A, one additional register for LFSR-B, and four additional internal memories are required to cache data for a double-length keystream in each cycle. The quad-keystream implementation also requires three registers for each LFSR, and twelve additional internal memories performed in the same manner.

The whole circuit is parallelized using additional circuit area, to generate double-length, quad-length keystream per one clock cycle. Figures 3, 4, and 5 show the quad-implementation of LFSR-A, LFSR-B,

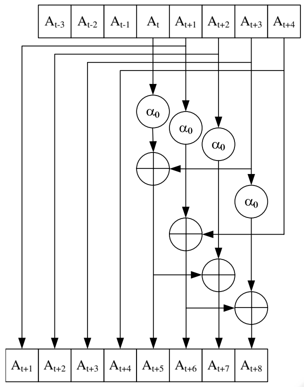

Figure 3: Quad-Keystream Implementation of LFSR-A.

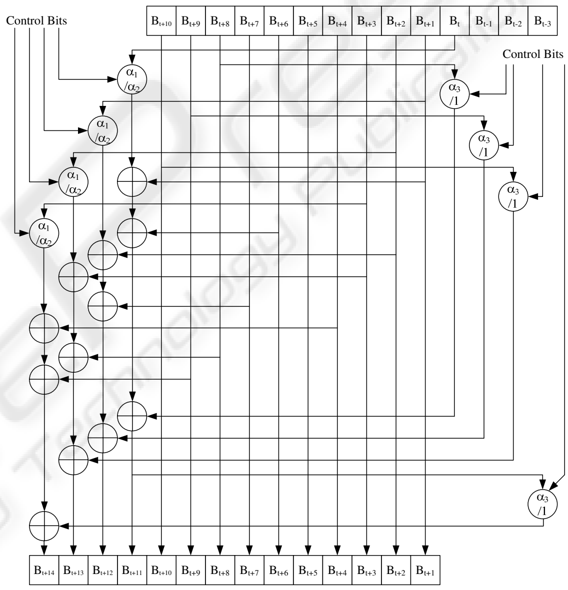

Figure 4: Quad-Keystream Implementation of LFSR-B.

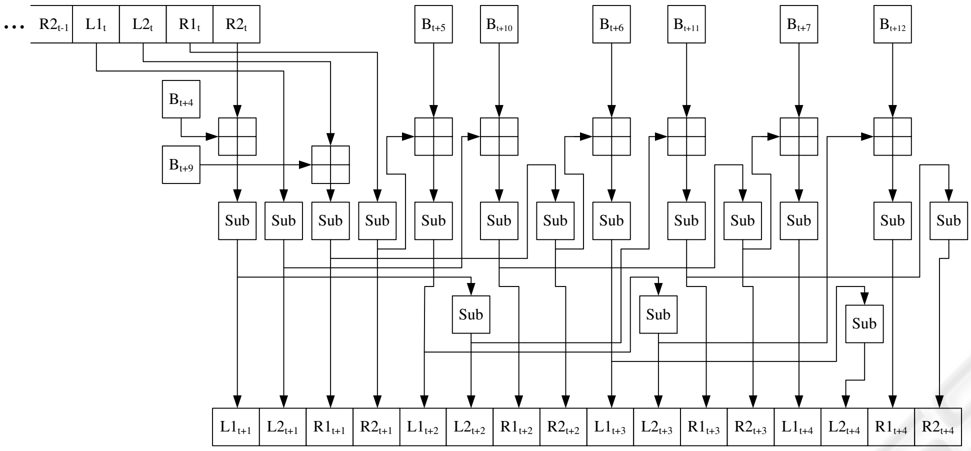

Figure 5: Quad-Keystream Implementation of Non-Linear Function.

and non-linear function, respectively.

The internal state values of LFSR-A, LFSR-B, and internal memories ($L1$ $L2$ $R1$ $R2$) are updated to the state values after four cycles by one clock cycle of the implementation.

A 128-bit keystream Z_t^{128} of the double-keystream implementation for encryption/decryption is computed as follows;

$$Z_t^{128} = (z_{t-1}^H, z_{t-1}^L, z_t^H, z_t^L)$$

Similarly, a 256-bit keystream Z_t^{256} of the quad-keystream implementation for encryption/decryption is computed as follows;

$$Z_t^{256} = (z_{t-3}^H, z_{t-3}^L, z_{t-2}^H, z_{t-2}^L, z_{t-1}^H, z_{t-1}^L, z_t^H, z_t^L)$$

3.2 Compact Implementation

Rudra *et. al.* proposed a compact implementation method for AES (Rudra et al., 2001). Their technique involves mapping field elements to a composite field representation. By using their technique, arithmetic operations on $GF(2^8)$ elements are transformed into operations in a composite field $GF((2^4)^2)$. AES operations translate to the composite field representation as $\mathbf{H}(x) = \mathbf{T}x$, where $\mathbf{H}$ denotes the mapping from $GF(2^8)$ to $GF((2^4)^2)$, and $\mathbf{T}$ denotes the corresponding transformation matrix. That is, an 8-bit operation in AES such as S-box is performed as 4-bit operations. Thus, the implementation size of the algorithm can be reduced by using their techniques.

We apply their technique to a part of the non-linear function of K2 as shown Figure 6. We add matrixes on data paths of the substitution of K2. Input values of the inverse operations are transformed into the elements of the composite field by the matrix $\mathbf{T}$ and

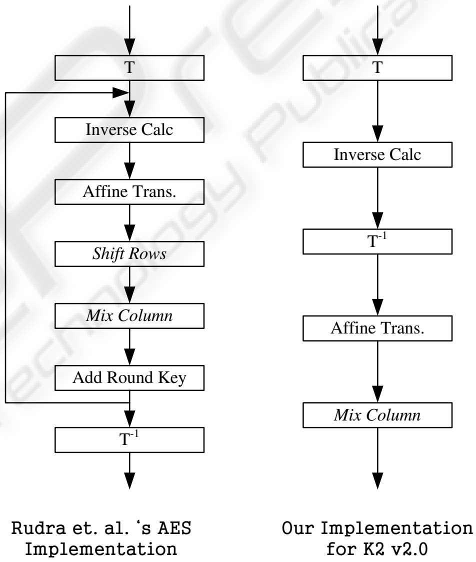

Figure 6: Non-Linear Function Architectures Transformed in the Composite Field.

its inverse matrix $\mathbf{T^{-1}}$ re-transform into the elements of $GF(2^8)$. The matrixes are an 8×8 transformation. Data in the inverse operation is defined as an element of the composite field $GF((2^4)^2)$, and data out of the inverse operation is computed as an element of $GF(2^8)$. The matrixes $\mathbf{T}$ and $\mathbf{T^{-1}}$ are defined by;

$$\mathbf{T} = \begin{bmatrix} 10100000 \\ 10101100 \\ 11010010 \\ 01110000 \\ 11000110 \\ 01010010 \\ 00001010 \\ 11011101 \end{bmatrix} \quad \mathbf{T}^{-1} = \begin{bmatrix} 00100100 \\ 11101110 \\ 10100100 \\ 01011010 \\ 10110010 \\ 01110010 \\ 10110000 \\ 01010001 \end{bmatrix}$$

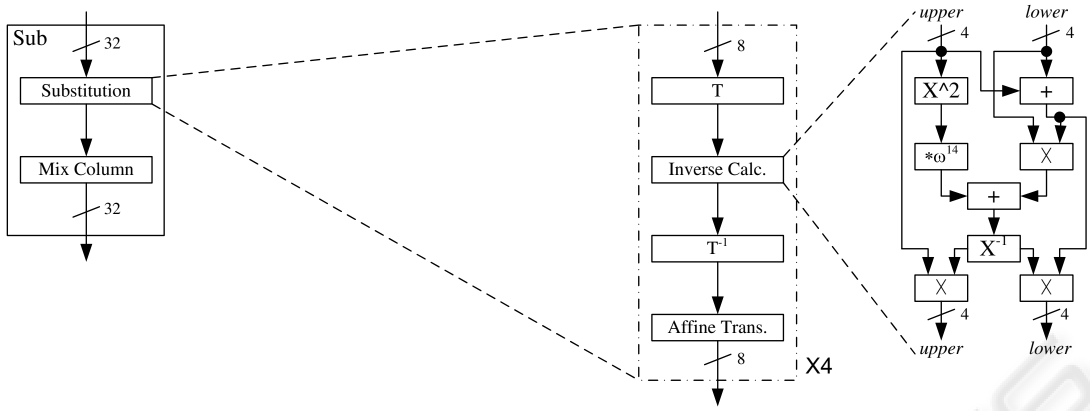

Figure 7: K2 *Sub* Operation Architecture Transformed in the Composite Field.

The *Sub* consists of substitutions and the *Mix Column*. The substitutions involve an inverse calculation and affine transformation. We transform all inverse operations that are located inside the substitution of the non-linear function into inverse operations in the composite field. The method used for transformation is similar to the method proposed by Rudra *et. al.*. They apply the method to all operations of the round function of AES. On the other hand, we apply the method only to the inverse operations. In AES implementations, the round function is used many times for producing encrypted/decrypted data. However, each *Sub* operation in the non-linear function are used once for one keystream generation in K2 implementations. Thus, the method should be applied only to inverse operation for efficient and conpact implementations of K2. We transform the *Sub* that is an 8-bit oriented operation into a 4-bit oriented operation in the composition field $GF((2^4)^2)$ as follows.

An inverse X^{-1} of an input value X is defined as;

$$X = r_0\lambda + r_1, \quad (r_0, r_1 \in GF(2^4))$$
$$X^{-1} = s_0\lambda + s_1 \quad (s_0, s_1 \in GF(2^4))$$

where λ is a generator of $GF((2^4)^2)$. In this condition, we can calculate X^{-1} as follows;

$$s_0 = (r_0 + r_1)\Delta^{-1}, \quad s_1 = r_1\Delta^{-1}$$

where $\Delta = r_0(r_0 + r_1) + \omega^{14} r_1^2$ and ω is a generator of $GF(2^4)$. An inverse operation of elements x in $GF(2^4)$ is implemented as a 4-bit-input 4-bit-output table. The affine transformation and *Mix Column* operations are the same as the original operations of K2.

Figure 7 shows the *Sub* operation architecture transformed in the composite field. The architecture consists of calculation units: squaring, multiplication, addition, multiplication of ω, 4-bit inverse operation, and exclusive-or of 4 bit values. The size of the 4-bit inverse operation is much smaller than the size of the 8-bit inverse operation. Thus, we can construct a compact implementation if these calculation units are shared by each operation and used to switch a pair of input and output values.

4 EVALUATION RESULT

In this section, we present the evaluation results of hardware implementations using an FPGA simulator. We implemented the K2 stream cipher targeted toward the Xilinx Spartan-II, Spartan-3, and Virtex-II FPGAs. The circuit sizes of implementations of normal, double-keystream, and quad-keystream implementations are larger than the capacity of Spartan-2, and thus, we evaluated normal, double, and quad-keystream implementations on Spartan-3. We also evaluated normal implementations on Virtex-II. The compact implementation is evaluated on Spartan-II, Spartan-3, and Virtex-II. We use Xilinx ISE 9.1 for post-place and route simulation and static timing analysis.

Table 1 and Table 2 show the evaluation results of high speed and compact 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. The reduction rates shown in Table 2 are calculated as $100 \times (N - M)/N$, where N is the number of slices of the normal implementation of devices and M is the number of slices of the compact implementation for the same devices.

Table 1: Evaluation Results of High Speed Implementations on FPGA.

Design	Data rate (bits/cycle)	Clock Freq. (MHz)	Throughput (Mbps)	Area (slice)	Throughput/Area (Mbps/slice)	Device
Normal	64	63.9	4090	3067	1.33	Spartan-3
Double-keystream	128	38.0	4864	5295	0.92	Spartan-3
Quad-keystream	256	20.4	5223	9161	0.57	Spartan-3
Normal	64	74.8	4787	2898	1.65	Virtex-II

Table 2: Evaluation Results of Compact Implementations on FPGA.

Target	Data rate (bits/cycle)	Clock Freq. (MHz)	Throughput (Mbps)	Area (slice)	Throughput/Area (Mbps/slice)	Reduction Rate (%)
Spartan-II	64	30.0	1920	2133	0.90	-
Spartan-3	64	39.1	2503	2140	1.17	30.2
Virtex-II	64	48.7	3117	2145	1.45	26.0

Table 3: Comparisons with FPGA Implementations of Other Ciphers.

Algorithm	Key Length (bit)	Throughput (Mbps)	Area (slice)	Through./Area (Mbps/slice)	Normalized Efficiency	Device
Trivium (Good et al., 2006)	80	102	40	2.55	0.80	Spartan-II
Grain (Good et al., 2006)	80	105	48	2.19	0.69	Spartan-II
Phelix (Good et al., 2006)	256	750	1077	0.70	0.70	Spartan-II
Edon80 (Kasper et al., 2006)	80	1.87	50	0.04	0.01	Spartan-3
DECIM v2 (Hwang et al., 2008)	80	46.25	80	0.58	0.18	Spartan-3
F-FCSR-H v2 (Hwang et al., 2008)	80	1104	342	3.23	1.01	Spartan-3
Pomaranch (Hwang et al., 2008)	80	49	648	0.08	0.03	Spartan-3
MICKEY-128 (Bulens et al., 2007)	128	200	190	1.05	0.53	Virtex-II
A5/1 (Galanis et al., 2004)	64	188.3	32	5.88	1.47	Virtex-II
RC4 (Galanis et al., 2004)	256	120.8	140	0.86	0.86	Virtex-II
E0 (Galanis et al., 2004)	128	189	895	0.21	0.11	Virtex-II
AES-128 (Good and Benaissa, 2005)	128	2.2	264	0.01	0.005	Spartan-II
AES-128 (Chodowiec and Gaj, 2003)	128	69	522	0.13	0.07	Spartan-II
AES-128 (Rouvroy et al., 2004)	128	87	1231	0.07	0.04	Spartan-3
AES-128 (Standaert et al., 2003)	128	1563	2257	0.69	0.35	Virtex1000
K2 *Compact* (this paper)	256	1920	2133	0.90	0.90	Spartan-II
K2 *Normal* (this paper)	256	4090	3067	1.33	1.33	Spartan-3
K2 $Double-keystream$ (this paper)	256	4864	5295	0.92	0.92	Spartan-3
K2 $Quad-keystream$ (this paper)	256	5223	9161	0.57	0.57	Spartan-3

We also evaluated the efficiency of the implementations by dividing the throughput by the area, which is the same index used in previous studies.

The quad-keystream implementation is expected to achieved 5 Gpbs on the Spartan-3 FPGA implementation. From the comparison of normal, double-keystream, and quad-keystream implementations, we showed that the rate of increase of area size is higher than the rate of increase of throughput by using the parallelization approach described in 3.1. The normal implementation was shown to be the most efficient among our implementation. The compact implementation technique reduces 30.2% of slices on Spartan-3, and 26.0% of slices on Virtex-II.

Comparisons with previously published evaluation results of FPGA implementations for other ciphers are shown in Table 3. The column of key length denotes a key length of each algorithm. There is a tradeoff between the performance of the algorithm and its security level (effective key length). Thus, we compare an index ”Normalized Efficiency”. 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.

The throughput of K2 is much faster than other block/stream ciphers, even though the circuit size is large. In terms of efficiency, the implementations of K2 has high efficiency when compared with other stream ciphers, and is 4-10 times higher than AES implementations. The FPGA implementation of K2 is suitable for applications that require high speed encryption/decryption and accepts medium size circuits for hardware implementations.

5 CONCLUSIONS

This paper presented the evaluation results of several FPGA implementations of K2 v2.0: high speed implementations and compact implementations. The quad-keystream implementation is expected to achieved 5 Gbps on a Spartan-3 FPGA implementation, and the circuit size of the compact implementation of K2 is 2133 on Spartan-II. Furthermore, we evaluated the efficiency of the implementations using two benchmarks: throughput per area and the normalized efficiency. The implementations of K2 has high efficiency compared with other stream ciphers, and its efficiency is 4-10 times higher than AES implementations. The evaluation results suggested that the FPGA implementation of K2 is suitable for applications using high speed encryption/decryption.

REFERENCES

Bulens, P., Kalach, K., Standaetes, F. X., and Quisquater, J. J. (2007). FPGA implementations of estream phase-2 focus candidates with hardware profile. In *SASC 2007 Workshop Record*, pages 205–214. eSTREAM Project.

Chodowiec, P. and Gaj, K. (2003). Very compact FPGA implementation of the AES algorithm. In *Proc. of CHES'03, LNCS*, volume 2779, pages 319–333. Springer Verlag.

Daemen, J. and Rijmen, V. (1998). *The Design of Rijndael, Information Security and Cryptography, Texts and Monographs*. Springer Verlag.

Ekdahl, P. and Johansson, T. (2000). SNOW -a new stream cipher. *The NESSIE submission paper*.

Galanis, M. D., Kitsos, P., Kostopoulos, G., and Koufopavlou, O. (2004). Comparison of the performance of stream ciphers for wireless communications. In *Proc. of CCCT'04*, pages 113–118.

Good, M. and Benaissa, M. (2005). AES on FPGA from the fastest to the smallest. In *Proc. of CHES'05, LNCS*, volume 3659, pages 427–440. Springer Verlag.

Good, T., Chelton, W., and Benaissa, M. (2006). Review of stream cipher candidates from a low resource hardware perspective. In *SASC 2006 Workshop Record*. eSTREAM Project.

Hwang, D., Chaney, M., Karanam, S., Ton, N., and Gaj, K. (2008). Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. In *SASC 2008 Workshop Record*, pages 151–162. eSTREAM Project.

Kasper, M., Kumar, S., Lemke-Rust, K., and Paar, C. (2006). A compact implementation of edon80. *eSTREAM Report 2006/057*.

Kiyomoto, S., Tanaka, T., and Sakurai, K. (2007a). K2: A stream cipher algorithm using dynamic feedback control. In *Proc. of SECRYPT 2007*, pages 204–213. INSTICC PRESS.

Kiyomoto, S., Tanaka, T., and Sakurai, K. (2007b). A word-oriented stream cipher using clock control. In *SASC 2007 Workshop Record*, pages 260–274. eSTREAM Project.

Rodríguez-Henríquez, F., Saqib, N. A., Díaz-Perez, A., and Koc, C. K. (2007). Cryptographic algorithms on reconfigurable hardware. *Signals and Communication Technology, Springer*.

Rose, G. and Hawkes, P. (1999). The t-class of SOBER stream cipher. *Publication Document, QUALCOMM Australia*.

Rouvroy, G., Standaert, F. X., Quisquater, J. J., and Legat, J. D. (2004). Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications. In *Proc. of ITCC 2004*, volume 2, pages 583–587.

Rudra, A., Dubey, P. K., Julta, C. S., Kumar, V., Rao, J. R., and Rohatgi, P. (2001). Efficient rijndael encryption implementation with composite field arithmetic. In *Proc. of CHES'01, LNCS*, volume 2162, pages 171–184. Springer Verlag.

Standaert, F. X., Rouvroy, G., Quisquater, J.-J., and Legat, J.-D. (2003). Efficient implementation of rijndael encryption in reconfiqurable hardware: Improvements and design tradeoffs. In *Proc. of CHES'03, LNCS*, volume 2779, pages 334–350. Springer Verlag.