On the Hardware Implementation of the MUGI Pseudorandom Number Generator

Paris Kitsos and Athanassios N. Skodras
Digital Systems and Media Computing Laboratory
School of Science & Technology
Hellenic Open University
Patras, Greece
e-mail: pkitsos@ieee.org

Abstract—**A high-speed hardware implementation of the MUGI pseudorandom number generator is presented in this paper. The MUGI generator is part of the ISO/IEC 18033-4: 2005 standard and it is expected to be used in many applications. The design has been coded in VHDL and FPGA devices have been used for its hardware implementation. A maximum throughput equal to 7 Gbps is achieved for a clock frequency of 110 MHz. As no other MUGI implementations do exist, the comparison with previous keystream generator implementations such as RC4, E0, A5/1, are given. These comparisons prove that the MUGI implementation is much faster compared to these implementations.**

I. INTRODUCTION

Wireless communications are expected to play a crucial role in the realization of the grand vision of 4G communications: the always best-connected scenario for anybody, to anything from anywhere at anytime. Wireless communications offer to the end-user the means for easy access to a global information and communications infrastructure. At the same time, wireless communications facilitate seamless connectivity amongst a host of computing, communications and sensing devices that collaborate to form a supporting ambient and pervasive computing environment.

With the arrival of the information age, cryptography has grown to an essential tool for a wide segment of industry and commerce. It can be used to protect all forms of electronic communications such as fax, e-mail, cellular phones and home banking systems. Cryptography can replace the sealed envelope of a paper-oriented system and ensure privacy on electronic media. More important is its ability to prevent forgery of electronic documents, and to supply mutual authentication of senders and recipients of these documents. Without such tools, wide-scale electronic commerce using the Internet would have been impossible.

Private-key cryptographic algorithms can be classified into block and stream ciphers. Block ciphers permute N-bit blocks of plaintext data under the influence of the secret key and generate N-bit blocks of encrypted data. Stream ciphers typically operate serially by generating a stream of pseudorandom key bits, the keystream (stream ciphers are also called *pseudorandom number generators*).

Fig. 1 shows the general diagram of the cipher process with stream cipher. The stream cipher takes two parameters, the secret key, K, and the initialization vector, IV, and produces the keystream bits, z_t. In stream encryption each plaintext symbol, P_t, is encrypted by applying a group operation with a keystream symbol, z_t, resulting in a ciphertext symbol c_t. In modern cipher the operation is the simple bitwise XOR.

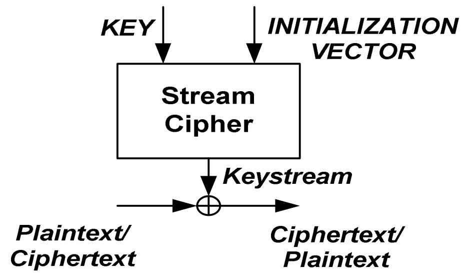

Fig. 1. The stream cipher process

Decryption takes the substraction of the keystream symbol from the ciphertext symbol. With the bitwise XOR this is the same operation.

The most known stream cipher used in many applications is the RC4 [1] and it was designed by R. Rivest in 1987. Moreover, there are many attacks and weaknesses mostly in key scheduling of RC4 [2]-[3]. Many attempts have taken place in order to propose new stream ciphers to match the current security levels.

Following this direction the hardware implementation of the new pseudorandom number generator called Multi Giga (MUGI) [4] is investigated. MUGI generator has been adopted by the International Organization for Standardization (ISO/IEC) 18033-4: 2005 standard [5].

This paper is structured as follows: In Section II the MUGI cipher is presented. In Section III the proposed architecture is analysed. In Section IV synthesis results and comparisons with previous published stream ciphers are given. Finally, Section V concludes the paper.

II. MUGI PSEUDORANDOM NUMBER GENERATOR

MUGI is a pseudorandom number generator (PRNG) used as a stream cipher. The design aims to be suitable for both software and hardware implementations. MUGI has two independent parameters as input. The first one is a 128-bit secret key while the second one is a 128-bit initial, public, vector. MUGI generates a 64-bit length random bit string in each round.

Since the MUGI is a PANAMA-like [6] stream cipher it consists of four main operational modules. As the Fig. 2 shows, similar to PANAMA, the *Internal State* is divided into two parts, *State a* and *Buffer b*.

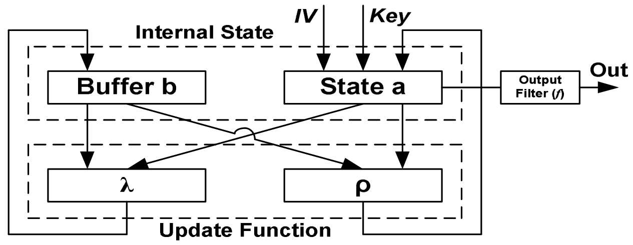

Fig. 2. A PANAMA-like stream cipher

The *Update Function* is divided in proportion to the internal state. Note that each update function uses another internal state as a parameter. We denote the update function of State *a* and Buffer *b* as ρ and λ function respectively. The output filter *f* abstracts some bits of State *a* for each round.

III. Proposed Architecture

The objective of the research described here is to ascertain how fast the MUGI pseudorandom number generator can operate on a synchronous hardware device. As such, the architecture which has been developed is implemented using the XILINX Virtex-E and Virtex-II [7]. The architecture that performs the MUGI pseudorandom number generator is shown in Fig. 3. As this figure shows the main parts of the proposed architecture are the State ***a***, the Buffer ***b*** and the functions ρ and λ. In addition, one 128-bit register is used for latching of the secret key and initialization vector. Also, the ***K/I init*** component is used for key and initialization vector transformations [4] before the algorithm initialization phase starts. The *Auxiliary Buffer1* holds the Buffer ***b*** data while the *Auxiliary Buffer2* holds the State ***a*** data during the initialization phase. Finally, there are a 3x192 multiplexer (MUX) and two XOR gates, 128-bit and 64-bit respectively, accomplishing the generator architecture.

The initialization phase of MUGI is divided into 3 steps. Firstly Buffer ***b*** with a secret key, ***K***, is initialized. Secondly the initialization of the State ***a*** with the initial vector, ***I*** takes place. Finally, the whole internal state is mixed. So, when the key is transformed is fed by the State ***a*** through the *IN2* multiplexer input. Then, the *a*, iterates only the function ρ and puts a part of each $a^{(t)}$ into Buffer ***b*** as follows, $b_{15-i} = (r^{i+1}(a,0))_0$. In the previous equation r^i means the ***i***-th iteration of r and $r\ (a,0)$ means the input from Buffer ***b*** is zero. In other words, the data stored into Buffer ***b*** are not used in this step. The *Auxiliary Buffer1* is responsible for this. In addition the output data of the State ***a*** are never used in the first step of the initialization phase.

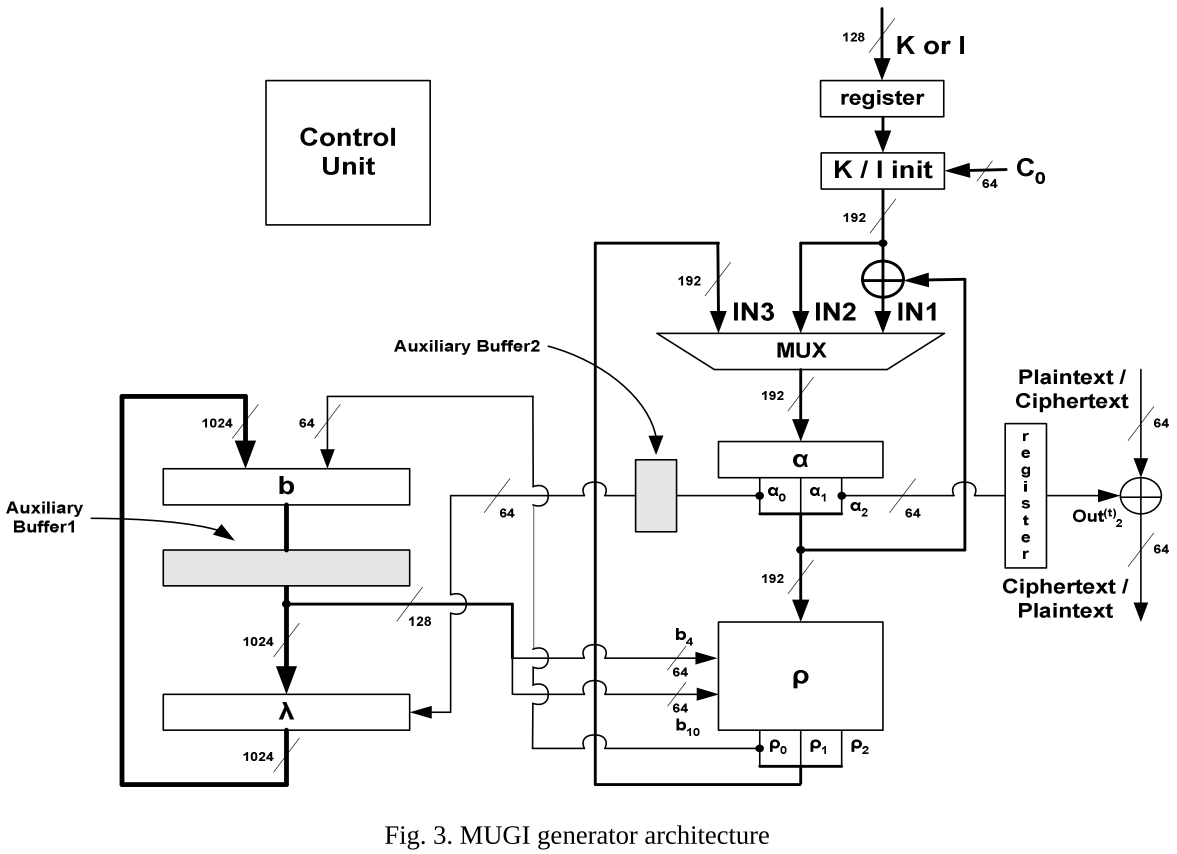

Fig. 3. MUGI generator architecture

The *Auxiliary Buffer2* is responsible for this. In the second step the mixed State *a* with value $a(K) = r^{16}(a_0, 0)$ and the initial vector, *I*, are required. If the *I* is added to State *a* through the *IN1* multiplexer input, State *a* is mixed again by 16 rounds iteration of function ρ. So, the mixed State *a* is represented as $r^{16}(a(K,I),0)$.

Finally, the last initialization phase step is a 16 rounds iteration of the whole update function *Update*,

$$a = Update^{16}(r^{16}(a(K,I),0), b(K)) \qquad (1)$$

where the notation $b(K)$ in the above equation denotes that Buffer *b* is initialized by the secret key *K*.

For security purposes, the algorithm output bits should not be available to the users during the initialization process. So, a 64-bit register is located at the generator output that does not latch its input bits during the initialization process.

After the initialization, the 64-bit register latches the generated bits and MUGI generates a 64-bit keystream. If we denote the output at round *t* as *Out*(*t*), then the output is given as,

$$Out(t) = a_2^{(t)} \qquad (2)$$

In other words MUGI outputs the lower 64-bit of State *a* at the beginning of the round process.

The State *a* and Buffer *b* are 192-bit and 1024-bit registers, respectively. The update function of State *a* is the function ρ. It is a kind of target heavy Feistel structure with two F-functions and uses Buffer *b* as a parameter. The VLSI implementation of the function ρ is depicted in Fig. 4.

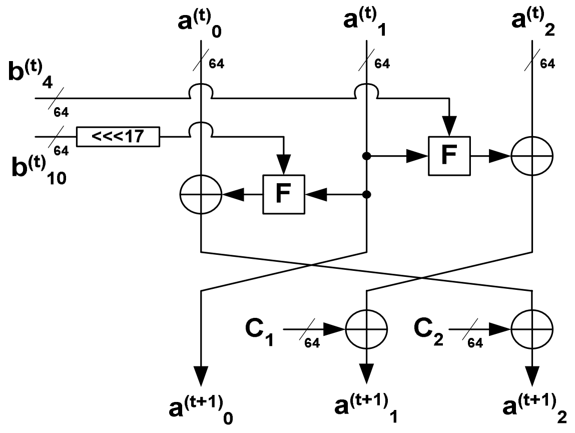

Fig. 4. The ρ function VLSI implementation

In addition, the hardware architecture of the F-function is depicted in Fig. 5. The bytewise substitution S-box is the same as the one in AES [8], while the linear transformation is the combination of a 4 x 4 matrix and a bytewise shuffling. MUGI uses MDS matrix which is the component of AES [8].

In addition, the function λ is the update function of Buffer *b* and uses a part of State *a* as a parameter. The mathematical background of function λ can be found in [4]. The hardware implementation consists of simple XOR operations and bit-shifting. Also, the values of the C0, C1 and C2 constants are defined in [4].

Finally, the Control Unit is responsible for the correct operation of the whole algorithm.

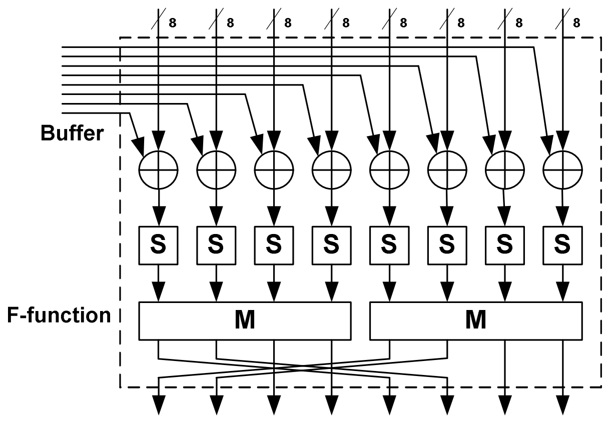

Fig. 5. The F-function hardware architecture

IV. Experimental Results and Performance Evaluation

The proposed implementation was captured in VHDL using structural description logic. The implementation was simulated for correct operation using the test vectors provided by the specifications [4]. The VHDL code was synthesized for Xilinx (Virtex-E and Virtex-II) FPGA devices [7]. The implementation then was simulated again for the verification of the correct functionality in real time operating conditions.

The required cipher S-boxes have been implemented by means of LUTs, increasing the algorithm time performance. The synthesis results for both Virtex-E and Virtex-II FPGAs are shown in Table 1.

TABLE 1
FPGA Synthesis Results

FPGA Devices	V200EBG352		2V500FG456	
Resources	***Used***	***Utiliz.***	***Used***	***Utiliz.***
I/Os	196	76 %	196	75 %
FGs	4183	89 %	3927	64 %
CLB Slices	2092	89 %	1964	64 %
DFFs	2437	44 %	2437	35 %
F (MHz)	96		110	
Throughput (Gbps)	6.14		7	

The throughput is estimated after the initialization phase. Performance comparisons between the proposed system and previous published architectures are shown in Table 2. According to our knowledge, no other implementation of the MUGI pseudorandom number generator has been previously published. So, comparisons with others similar generators [9]-[14] are given in order to have a fair and detailed comparison of the proposed system.

TABLE 2
PERFORMANCE COMPARISONS

Stream Cipher	FPGA Device	F (MHz)	Throughput (Mbps)
A5/1 [9]	2V250FG25	188.3	188.3
A5/1 [10]	nn	3000	3000
HELIX [9]	2V250FG25	32.0	1024.0
W7 [9]	2V250FG25	96.0	768.0
RC4 [11]	2V250FG256	64	22
E0 [12]	2V250FG25	189	189
WG [13]	ASIC	1000	125
Achterbahn [14]	ASIC	1000	8000
MUGI#1	V200EBG352	96	6140
MUGI#2	2V500FG456	110	7000

MUGI#1 symbolizes the implementation on VIRTEX V200EBG352 FPGA, while *MUGI#2* symbolizes the implementation on 2V500FG456 FPGA. The A5/1 is a synchronous single-bit stream cipher. As the above table shows the proposed implementations outperform the A5/1 implementations in [9], [10]. The HELIX implementation [9] uses a 256-bit key and a 128-bit initialization vector and outputs an 8-bit keystream. In addition, the W7 [9] is a single-bit cipher that supports a key of 128-bit length. The well known RC4 [11] is a variable key-size stream cipher which produces an 8-bit keystream. Also, in [12] the implementation of the stream cipher, E0, which is used in Bluetooth is presented. In [13] and [14] the straightforward implementations of two new stream ciphers, WG and Achterbahn respectively, are presented. These ciphers have been submitted and are under consideration from the ECRYPT (European Network of Excellence for Cryptology) project [15]. The WG is a single-bit synchronous cipher while the Achterbahn is a synchronous stream cipher which has parallel implementation ability and produces a keystream with lengths equal to 1-, 2-, 4- and 8-bit per clock cycle. With this technique the time performance is increased. In our comparisons (Table 2) the faster case was selected. As Table 2 shows, the proposed MUGI implementations outperform all of the previous implementations in terms of time performance. Only the Achterbahn cipher has better performance compared to ours. However some Achterbahn security weaknesses do exist [16] that do not allow an effective substitution of MUGI by Achterbahn.

V. CONCLUSIONS

The MUGI pseudorandom number generator is the most recent generator that has been standardized. MUGI has undergone much cryptanalysis not only for all the standard attacks on stream ciphers but also using differential and linear cryptanalysis. At this time, none of these attacks have been proved successful.

A MUGI high-speed hardware architecture is described in this paper and implemented by means of FPGA devices. Experimental results prove that the MUGI implementation is a flexible solution in applications with very high-speed specification demands. The implementation on FPGAs achieves a throughput of 6.14 Gbps or 7 Gbps, depending on the FPGA devices used..

REFERENCES

[1] B. Schneier, “Applied Cryptography - Protocols, Algorithms and Source Code in C”, Second Edition, John Wiley and Sons, New York, 1996.

[2] S. Fluhrer, I. Mantin, and A. Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4”, In Proc. of 8th Annual Workshop on Selected Areas in Cryptography (SAC 2001), Toronto, Canada, 2001.

[3] P. D. Kundarewich, S. J. E. Wilton, A. J Hu, “`A CPLD-based RC4 Cracking System'”, In Proc. of 1999 Canadian Conference on Electrical and Computer Engineering , May 1999.

[4] D. Watanabe, S. Furuya, H. Yoshida, and K. Takaragi, “MUGI Pseudorandom Number Generator”, Specification, 2001, on line available at http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html

[5] International Organization for Standardization, “ISO/IEC 18033-4:2005: Information Technology – Security Techniques – Encryption Algorithms – Part 4: Stream ciphers”, 2005.

[6] J. Daemen, and C. Clapp, “Fast Hashing and Stream Encryption with PANAMA”, In Proc. of Fast Software Encryption: 5th International Workshop, FSE'98, Paris, France, March 1998.

[7] Xilinx Inc., San Jose, California, 2005, www.xilinx.com

[8] J. Daemen and V. Rijmen, “AES proposal: rijndael,”, AES algorithm submission, September 3, 1999, on line available at http://www.nist.gov/aes/.

[9] M. D. Galanis, P. Kitsos, G. Kostopoulos, N. Sklavos, and C. E. Goutis, “Comparison of the Hardware Implementation of Stream Ciphers”, *The International Arab Journal of Information Technology (IAJIT)*, Colleges of Computer and Information Society, Vol. 2, No. 4, October 2005, pp. 267-274.

[10] L. Batina, J. Lano, N. Mentens, S.B. Ors, B. Preneel, I. Verbauwhede, “Energy, Performance, Area versus Security Trade-offs for Stream Ciphers”, Workshop Records of SASCS – The State of the Art of Stream Ciphers (Brugge, Belgium, 2004), pp. 302–310. Available at http://www.isg.rhul.ac.uk/research/projects/excrypt/stvl /sasc-record.zip

[11] P. Kitsos, G. Kostopoulos, N. Sklavos and O. Koufopavlou, “Hardware Implementation of the RC4 stream Cipher”, In Proc. of 46th IEEE Midwest Symposium on Circuits & Systems '03, December 27-30, Cairo, Egypt, 2003.

[12] P. Kitsos, N. Sklavos, K. Papadomanolakis and O. Koufopavlou, “Hardware Implementation of Bluetooth Security”, IEEE Pervasive Computing, vol. 2, no.1, pp. 21-29, January-March 2003.

[13] D. Gligoroski, S. Markovski, L. Kocarev and M. Gusev, “Edon80 - Hardware Synchronous Stream Cipher”, Symmetric Key Encryption Workshop (SKEW), Scandinavian Congress Center, Aarhus, Denmark, 26-27 May 2005.

[14] B. Gammel, R. GÄottfert, and O. Knifer, The Achterbahn Stream Cipher”, eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002, 2005. http://www.ecrypt.eu.org/stream.

[15] ENCRYPT - European Network of Excellence in Cryptology, “Call for Stream Cipher Primitives”, Scandinavian Congress Center, Aarhus, Denmark, 26-27 May 2005, http://www.ecrypt.eu.org /stream/.

[16] T. Johansson, W. Meier and F. Muller, “Cryptanalysis of Achterbahn”, eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002, 2005. http://www.ecrypt.eu.org/stream/papersdir/064.pdf.