Implementing Serpent Cipher in Field Programmable Gate Arrays

Jaroslaw Sugier

Wroclaw University of Technology
Institute of Computer Engineering, Control and Robotics
ul. Janiszewskiego 11/17, 50-372 Wroclaw, Poland
jaroslaw.sugier@pwr.wroc.pl

Abstract— **The Serpent cipher finished second in the AES contest and although it was not chosen as the winner it was widely praised for excellent cryptographic strength and efficiency. Nowadays main attention is turned towards the Rijndael algorithm (the AES standard) but the Serpent method still deserves due consideration as a viable alternative. This paper investigates possible hardware implementations of the cipher in Field Programmable Gate Array (FPGA) devices. The paper stresses solutions that follow pure data-path approach: structure of the hardware is organized along the way of propagation of data being processed from the plain text to cipher text, without modifications that would re-use hardware for iterative repetitions of some transformations in order to reduce total design size and resource occupancy. Although the iterative organization is natural for round-based ciphers and is used in vast majority of projects presented in literature, here we aim specifically at combinatorial and pipelined architectures in order to illustrate interesting new potential that can be exploited especially in case of this cipher in programmable FPGA environment. The target implementation platform is Spartan-3E family from Xilinx, probably the most popular devices in low-cost applications at present time.**

Keywords— **Serpent cipher, cryptography processor, FPGA, hardware implementation**

I. INTRODUCTION

Ciphers are used in many applications of contemporary IT systems and they play the crucial role in preserving their security, safety and reliability. In this work we discuss low-cost FPGA implementations of the Serpent ([2]) cipher. Based on the original results that are presented here one can investigate the potential of the method and evaluate expenses at which the superior efficiency can be achieved.

Today, after many years of official use of the ciphers that were developed for the AES contest, there are numerous hardware implementations of the algorithms that use both mask- and field-programmable gate arrays ([5], [6], [9], [11], [12], [14]). Most of the solutions described in the literature are highly customized for specific device architectures and / or operating environments. Their excellent performance parameters were possible thanks to elaborate optimizations, often including manual fine-tuning during mapping, layout and routing stages of FPGA implementation. Hardware platforms for these projects demanded the fastest, largest and also most expensive chip families in the FPGA world.

Such “top-notch specialization” approach was natural in the early years of AES conception, but today the situation has changed thanks to ever-growing capacity of programmable devices. First of all, the cipher units became just one of the modules in the system-on-the chip implemented in a popular, often low-cost, FPGA. In designs of this kind the AES optimization cannot dominate the whole project. Secondly, in common equipment designed for personal use the encoding / decoding throughput parameters does not need to reach multi-Gbps values; numbers in the range of single Gbps are sufficient for popular transmission channels like High-Speed USB or mass-storage devices. In this situation not the performance of the unit (generally understood almost always as high data throughput) but its flexibility and fast, fully automatic implementation become a highly valued features that smoothes the progress of the design and reduces time-to-market. In this paper problem of *effective* Serpent implementation is considered taking these new aspects into account.

II. THE SERPENT CIPHER

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

A. Organization of the Algorithm

Serpent ([2]) is a symmetric block cipher that belongs to a class of substitution-permutation networks (SPN). It was developed by Ross Anderson (University of Cambridge Computer Laboratory), Eli Biham (Technion Israeli Institute of Technology), and Lars Knudsen (University of Bergen, Norway). In the version that was submitted for AES contest the method operates on 128 bit blocks of data using in the processes a 256 bit external key. The transformation flow is divided into 32 uniform rounds repeated over the data block with each round consisting of (nearly identical) sequence of elementary operations. Each round requires its special 128-bit round key; since the last round needs two keys, total of 33 different round keys are required and these are generated from the external key in a separate key schedule.

Fig. 1 represents data transformations that constitute the encryption process. Let P be a 128b plaintext, B_i – a data block that enters the i-th round R_i, K_i – the round key, C – encoded ciphertext. Before the plaintext block enters the procedure a special bit reordering – so called Initial

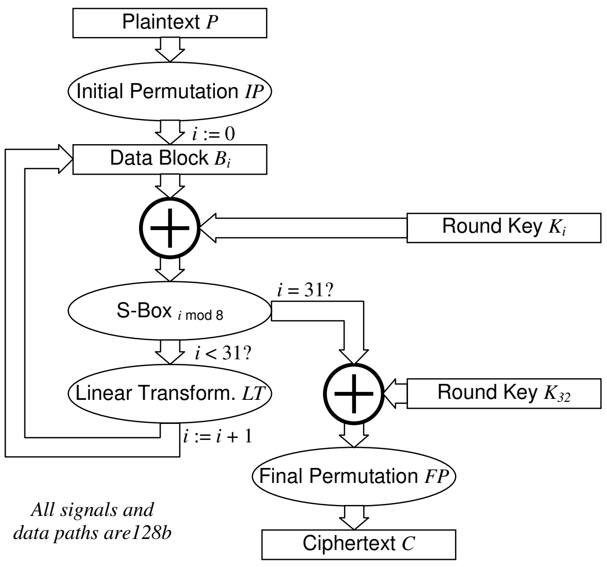

Fig. 1 Data processing during Serpent encryption in iterative representation.

Permutation IP – is performed. The plaintext P after permutation gives block B_0, which is the input to the first round. The output of the first round (R_0) is B_1, the output of the second round (R_1) is B_2, the output of round i is B_{i+1}, and so on, until the output of the last round is received as B_{32}. Then the Final Permutation FP (which is an inverse of IP) is applied to give the ciphertext C.

Inside the 32 rounds the actual encoding is carried out. The first 31 ones (0…30) are identical and the last one (31) is slightly modified. As the first transformation in each round, the block B_i is XOR-ed with the round key K_i that is supplied by the key schedule. The resulting vector is then passed through *Substitution Boxes*. The algorithm defines 8 different S-Boxes numbered 0 … 7 with each round R_i using S-Box number i mod 8. The vector created by S-Boxes undergoes *Linear Transformation LT* giving block B_{i+1} that is the input to the next round. In the last round R_{31} the linear transformation is replaced with XOR operation with the last key K_{32} (therefore two keys are required in this round).

The whole data path from the plaintext P to the ciphertext C can be formally described by a sequence of the following equations:

$$B_0 := IP(P) \tag{1}$$

$$B_{i+1} := LT(SBox_{i \bmod 8}(B_i \oplus K_i)), \qquad i = 0 \dots 30 \tag{2}$$

$$B_{32} := SBox_7(B_{31} \oplus K_{31}) \oplus K_{32} \tag{3}$$

$$C := FP(B_{32}) \tag{4}$$

B. Elementary Transformations

The three elementary operations that make up the rounds are: key mixing, bit substitution and linear transformation. Of these three, the first one is just an 128-bit 2-input XOR operation, but the latter are more evolved and require more complex implementations that must be designed according to specification.

Like in Rijndael, static substitution boxes perform the non-linear transformation of the data block in every round. Unlike the AES winner which applies repeatedly the same one 8x8 substitution, the Serpent defines 8 different 4x4 S-boxes (i.e. mappings of 4 bits into 4 bits) with each round using just one S-box. As a result each S-box is used in precisely four rounds, and in each of these it is used 32 times in parallel to transform the whole 128-bit block. In the initial version of the algorithm the authors adopted the S-boxes from DES in order to ensure a high level of public confidence that no secret trapdoor was inserted in them. Later, after public investigation of properties of DES S-boxes that was inspired by new advances in differential and linear cryptanalysis, a new (and better) ones were proposed with even stronger immunity to attacks. Again, to keep high level of public confidence their contents was generated by a special numerical routine which was explicitly clarified and justified.

As far as the linear transformation that concludes the rounds is concerned, initially simple rotations of the 32-bit subwords were proposed. In order to ensure maximal avalanche effect, the idea was to choose these rotations in a way that guaranteed maximal effect in the fewest number of rounds. However, as the avalanche was still slow, the authors had to move to more complex transformations and found the XOR operation sufficiently effective: each output bit of the LT is the exclusive-or of specific (from 3 to 7) input bits. More complex operations like words addition were also investigated but their cost was too high in both hardware and software implementations and therefore they were dropped ([2]).

C. The Key Schedule

The task of the key schedule is to generate 33 round keys K_i from the external key K that is supplied by the user. The key K can be of almost any length but when the proposal for AES standard was formulated it was fixed at 128, 192 or 256 bits with special expansion procedure that is to be applied to keys

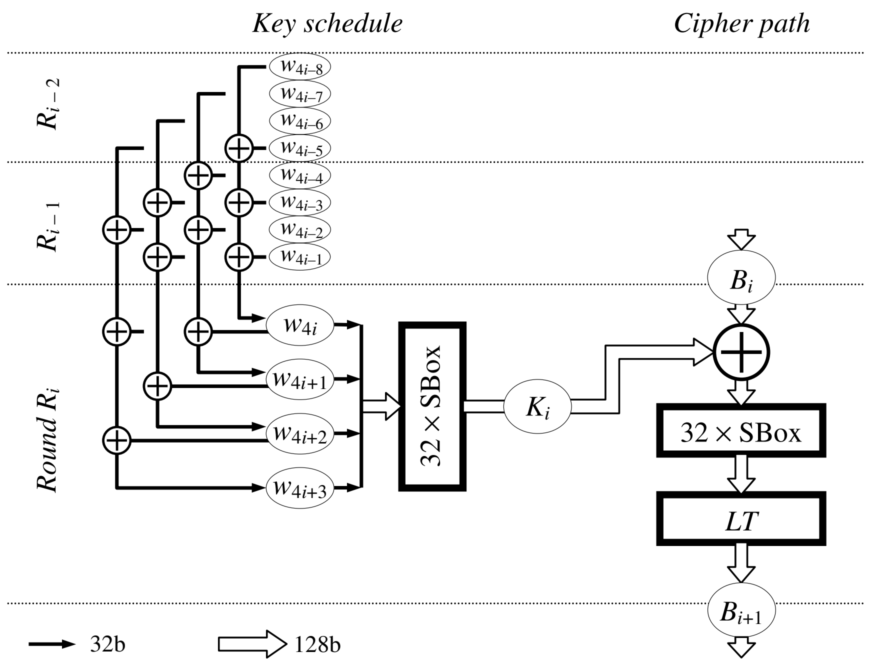

Fig 2. Data paths in single round of the Serpent cipher. The contents of the first prekey words $w_{-1} \ldots w_{-8}$ needed in rounds R_0 and R_1 are taken from the external key K. In the last round R_{31} additional key mixing with K_{32} is performed instead of the linear transformation LT.

with less than 256 bits. The procedure maps short keys to the full length by appending one "1" bit to the MSB end, followed by as many "0" bits as required to make up 256 bits. This routine maps every short key to a full-length key with no two short keys being equivalent and ensures that the key schedule receives as an input external key that is exactly 256 bits long.

The schedule first creates a set of 32-bit *prekeys* w_i. The starting 8 prekeys numbered from –1 to –8 are simply filled with bits of the external (user) key K:

$$\{w_{-1}, w_{-2}, \ldots w_{-8}\} := K \tag{5}$$

and then another 132 prekeys $w_0 \ldots w_{131}$ are generated by the following affine recurrence:

$$w_i := (w_{i-1} \oplus w_{i-3} \oplus w_{i-5} \oplus w_{i-8} \oplus \phi \oplus i) <<< 11 \tag{6}$$

where <<< denotes rotation and ϕ is the fractional part of the golden ratio $\left(\sqrt{5}+1\right)/2$ (represented as 32-bit vector 0x9E3779B9 in hexadecimal notation). The underlying polynomial $x^8 + x^7 + x^5 + x^3 + 1$ is primitive, which together with the addition of the round index guarantees an even distribution of key bits throughout the rounds and eliminates weak and related keys.

The round keys are now calculated from the prekeys using the same set of 8 substitution boxes that are defined for the cipher path. The general rule is that the key K_i is computed from a group of four prekeys w_{4i}, w_{4i+1}, w_{4i+2} and w_{4i+3} that undergoes bit substitution and reordering:

$$\begin{aligned} K_0 &:= IP(\ SBox_3(\ w_0, w_1, w_2, w_3\ )\ ) \\ K_1 &:= IP(\ SBox_2(\ w_4, w_5, w_6, w_7\ )\ ) \\ &\ldots \\ K_{31} &:= IP(\ SBox_4(\ w_{124}, w_{125}, w_{126}, w_{127}\ ) \\ K_{32} &:= IP(\ SBox_3(\ w_{128}, w_{129}, w_{130}, w_{131}\ ) \end{aligned} \tag{7}$$

To avoid repetitive use of the same substitution as later in the round, during computation of K_i the schedule uses S-boxes number $(3 - i) \bmod 8$.

III. Hardware Implementation

Apart from simplicity of elementary operations, ease of hardware implementation of the Serpent algorithm comes from the fact that its processing flow is composed of (almost) identical rounds that are repeatedly executed over a given block of data. This leads to many potential processing schemes that blend different flavors of combinational, pipelined and iterative architectures ([5], [10], [13], [14]).

Fig. 2 graphically represents cipher transformations equivalent to equations (2) (cipher path) and (6)—(7) (key schedule). As it can be seen, data dependency is more complex in case of key computations since key for round R_i is generated form prekeys of the two preceding rounds, R_{i-1} and R_{i-2}.

A. Combinational Architecture

In this organization hardware structure closely follows flow of the data that is being encoded. All 32 rounds of the cipher are implemented as separate hardware modules and create a continuous combinational path from the inputs (user key K and plain text P) to the output (cipher text C). The complete unit operates as a combinational module that transforms 128 + 256 = 384 input bits (data + key) into 128 output bits (cipher) with some propagation delay.

The hardware implementing one round is organized along identical paths as those in Fig. 2, that summarized data transformations within the round: computation of the keys relies on 32-bit prekey words w_i while the data path transforms 128-bit vectors B_i. The RTL code that describes this organization identifies the required number of internal bit vectors and defines combinational modules that are represented by solid ovals on the diagram: the substitution boxes (8 versions) and the linear transformation component (one version for all rounds). There are no flip-flops used in this solution.

B. Cipher-Only Pipelined Architecture

General idea of pipelining is to introduce evenly spaced extra registers in the middle of the combinational circuit in such a way that several blocks of data can be processed at the same time during one clock cycle. In the architecture presented above the natural points of placing the pipeline registers are signals B_i that cross boundaries of cipher rounds; this transforms each round into one pipeline stage. In technical terms such organization can be interpreted as 32-stage outer loop pipelining ([5]). Valid output appears 32 clock cycles after input and the unit requires $32 \times 128 = 4096$ flip-flops for pipeline registers.

The problem with this approach is that the key schedule path remains combinational and this fact slows down changes of the external key during operation: loading a new key invalidates the pipeline contents for 32 clock ticks until new data fill all the stages. This may exclude this architecture from environments with frequent key changes but if the key remains constant most of the time it is the optimal organization in terms of both speed and size.

C. Fully Pipelined Architecture

To synchronize computation of the round keys with operation of the cipher path the key schedule must be pipelined in an equivalent way. This task is additionally complicated by more complex data dependencies in the path; detailed presentation of the proposed solution and discussion of its specific features can be found in [15].

Describing briefly, simple estimation based on Fig. 2 would suggest introducing just $33 \times 128 = 4224$ flip-flops for storage of the key bits in all rounds, but in order to guarantee proper synchronization of the both paths (cipher + key) as well as to achieve optimal timings final solution needs to be more complicated. First of all, since the keys are computed from w_i words these also must be stored in another 4224 flip-flops and this number must be doubled to keep w_i contents for two clock ticks (the key K_i depends on prekeys form the two preceding rounds, R_{i-1} and R_{i-2}). Such a high flip-flop utilization usually is not a problem in case of FPGA hardware, since there is sufficient amount of registers in programmable blocks along data paths that remains unused in combinational circuits. As the implementation has shown, this is true also in the case of this project and the automatic implementation tool has had no special problems dealing with design layout and routing even after introducing over 16000 flip-flops.

Furthermore, the last cipher round needed additional modification. Because it uses two keys – K_{31} and K_{32} – to follow the structure of key schedule it also must be split into two stages: the first one contains key mixing with bit substitution and the second one performs only key mixing. This increases total length of the pipeline to 33 stages. Finally, to achieve optimal timing key computation and actual data encoding can be done in parallel provided that the cipher pipeline is delayed for one clock cycle so that the round R_i is computed one cycle later after the key K_i. As a result total number of pipeline stages increased from 33 to 34 but, at the same time, minimum clock period could be reduced by nearly 1/3.

D. Deciphering Units

In Serpent the only way to decipher an encoded block is to execute the same set of 32 rounds in reversed order with elementary operations replaced by their inversed versions. The specification contains definition of inverse substitution boxes as well as an inverse linear transformation. From hardware point of view the nature of both operations remains unchanged: the inverse S-boxes are just 4-input Boolean functions and the inverse *LT* is computed by a XOR network, although a more complex one (the underlying polynomials have higher degree thus require more gates in implementation).

As the same set of 33 round keys must be generated from the external key, the key schedule remains identical but now the keys are used in reversed order (K_{32} first, K_0 last) so they must be pre-computed and available before the actual deciphering can be started. This requirement makes any key schedule pipelining unreasonable in decoding; of the architectures considered above only the first two ones (combinational and cipher-only pipelined) were used for construction. A diagrams analogous to Fig. 2 representing decoding paths is not included to save space but its structure can easily be deduced.

IV. Implementations

All the considered architectures were implemented in Xilinx XC3S1600E-5 device using the Xilinx Synthesis Tool (XST) and ISE Design Studio ([15]) and the results are summarized in Table 1. The implementation was fully automatic and the optimization was set to “speed” with appropriate constrains specified in UCF file being the only optimization goals. In all cases the design consisted of the full cipher path and the key schedule unit, i.e. it constituted the complete encoding or decoding unit. All the architectures fit the Spartan-3E chip, with combinatorial one taking only 68% of the array and fully pipelined organizations occupying all logical blocks, yet using not more than 81% of available LUTs and 58% of flip-flops. This fact indicates that more dense packing is still possible if extra space for another components co-existing in the same chip would be required.

The simplest combinational architecture is not only the smallest one but also offers the minimal latency of 150 ns, although this leads to throughput below 1Gbps. With regard to throughput criterion alone, the cipher-only pipelined

TABLE I
DIFFERENT ARCHITECTURES OF SERPENT MODULE IMPLEMENTED IN XC3S1600E DEVICE ([15])

Architecture:	**Combinational**	**Combinatorial (deciphering)**	**Cipher-only pipelined**	**Cipher-only pipelined (dec.)**	**Fully pipelined**
Slices (of 14752)	9 999	11 748	11 897	11 926	14 750
LUTs (of 29504)	18 756	22 184	22 680	22 960	24 331
Flip-flops (of 29504)	0	0	4 224	4 096	16 768
$t_{\min}$ [ns]	150	162	6.5	6.7	7.3
$f_{\max}$ [MHz]	6.7	6.2	154	149	137
Throughput [Mbps]	854	791	19 692	19 116	17 534
Pipeline stages	1	1	32	32	34
Latency [ns]	150	162	208	214	248
Mbps per slice	0.085	0.067	1.66	1.60	1.19
Critical path delay: logic	44.7%	49.3%	47.4%	49.5%	43.5%
route	55.3%	50.7%	52.6%	50.5%	56.5%

organization is the winner as it offers the highest frequency of operation. It must be remembered, though, that this design performs poorly in applications were frequent changes of the external key must be supported. If this condition must be met the fully pipelined architectures should be selected.

As expected, deciphering unit has slightly worse timing parameters that its encoding counterpart due to somewhat more complex implementation of the inverse linear transformation.

V. CONCLUSIONS

A. *Serpent vs. AES (Rijndael) in FPGA Implementations*

Compared to 8x8 S-boxes required in Rijndael, the fact that Serpent uses 4×4 substitutions, i.e. combinational functions of 4 input variables, greatly simplifies their implementation in FPGA device. As the elementary parts available in the logic cells of the Spartan-3E array that are used for synthesis of any combinational function – the Look-Up Tables (LUT) – have 4 inputs, each output bit of the S-box can be computed by exactly one LUT, i.e. by the minimal amount of resources. Transformation of the whole block requires 128 LUTs in every round and the entire path with 32 rounds needs 4096 LUTs.

An equivalent realization of Rijndael S-boxes would need much more resources. Being an 8-input function, every output bit would now require 2^8 / 16 = 16 LUTs for storing the substitution function alone plus some additional LUTs for multiplexing their outputs (in terms of ROM organization: for address decoding). Even without the extra multiplexing logic this would require 128 x 16 = 2048 LUTs in each round and 20480 LUTs in the entire 10-round cipher. The difference is so big that it practically prohibits implementation of Rijndael substitution function in LUT elements; instead, larger synchronous Block RAM modules must be used but they, consequently, make purely combinational (asynchronous) operation of the module impossible ([5], [13], [14]). Feasibility of combinational implementation seems to be one of the most interesting differences between FPGA implementations of Serpent and Rijndael and therefore it was explored in this paper.

B. *Final Remarks*

The Serpent cipher remains overshadowed by the winner of the AES selection procedure – the Rijndael algorithm – but nevertheless it should not be forgotten since it offers better security. At the time of AES selection its extra cost coming from larger number of rounds (33 vs. 10 in Rijndael) was important but today this cost difference has been reduced by the progress in capacity and capabilities of programmable devices, as the results presented in this text show. Additionally, relative simplicity of its substitution boxes creates new options for hardware implementations that are not available in case of the Rijndael.

In general Serpent is well suited to fully automatic FPGA synthesis and implementation: the software tools can deal with its regular, round-based structure quite efficiently despite the size of the whole design, producing acceptable results without user intervention involving hand optimizations of layout or routing. Finally, although this paper is intentionally concentrated on combinational and pipelined organizations, if hardware size is critical the algorithm can be equally well implemented in iterative architectures.

REFERENCES

[1] RSA Laboratories, DES Challenges, http:// www.rsa.com, pp. 1997–99.

[2] R. Anderson, E. Biham, L. Knudsen, “Serpent: A Proposal for the Advanced Encryption Standard”, The First Advanced Encryption Standard (AES) Candidate Conference, Ventura, California, August 20–22, 1998 (http://www.cl.cam.ac.uk/~rja14/serpent.html), 1998.

[3] R. Anderson, E. Biham, L. Knudsen, “The Case for Serpent”, The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, USA (proceedings available from http: //csrc.nist.gov/encryption/aes/round2/conf3/aes3conf.htm), 2000.

[4] R. Anderson, E. Biham, L. Knudsen, “Serpent and Smartcards”, Smart Card Research and Applications, Proc. 3rd International Conference CARDIS '98, Louvain-la-Neuve, Belgium, September 14–16, 1998. *Lecture Notes in Computer Science*, Volume 1820, Springer, 2000.

[5] K. Gaj, P. Chodowiec, “Comparison of the hardware performance of the AES candidates using reconfigurable hardware”, The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, USA (proceedings available from http://csrc.nist.gov/encryption/aes/ round2/conf3/aes3conf.htm), 2000.

[6] P. Mroczkowski, “Implementation of the block cipher Rijndael using Altera FPGA”, Military University of Technology, Warsaw, 2000.

[7] D. A. Osvik, “Speeding up Serpent”, The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, USA (proceedings available from http://csrc.nist.gov/ encryption/aes/round2/conf3/aes3conf.htm), 2000.

[8] National Institute of Standards and Technology, “Specification for the Advanced Encryption Standard (AES)”, Federal Information Processing Standards Publication 197, November 26, 2001 (http://www.csrc.nist.gov), 2001.

[9] J. Lazaro, A. Astarloa, J. Arias, U. Bidarte, C. Cuadrado, “High Throughput Serpent Encryption Implementation”, Field Programmable Logic and Application, *Lecture Notes in Computer Science*, Volume 3203, Springer, 2004.

[10] P. P. Chu, *RTL Hardware Design Using VHDL*, John Wiley & Sons, New Jersey, 2006.

[11] M. Liberatori, F. Otero, J. C. Bonadero, J. Castineira, “AES-128 Cipher. High Speed, Low Cost FPGA Implementation”, Proc. Third Southern Conference on Programmable Logic, Mar del Plata, Argentina, IEEE Comp. Soc. Press, 2007.

[12] M. Wójcik, “Effective implementation of Serpent algorithm”, dissertation for M.Sc. degree, Faculty of Electronics and Information Technology, Warsaw University of Technology, 2007.

[13] Ł. Krukowski, J. Sugier, “Designing AES cryptographic unit for automatic implementation in low-cost FPGA devices”, *Int. J. Critical Computer Based Systems*, Vol. 1, Nos. 1/2/3, 2010, pp. 104–116.

[14] K. Piwko, “Hardware implementation of cryptographic algorithms in programmable logic devices”, dissertation for M.Sc. degree, Wrocław University of Technology, Faculty of Electronics, 2010.

[15] J. Sugier, “Low-cost hardware implementation of Serpent cipher in programmable devices”, *Monographs on System Dependability Vol. 3: Technical Approach to Dependability*, Oficyna Wydawnicza Politechiki Wrocławskiej, Wrocław, 2010.