Proceedings of the 13th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'13), 16–19 October 2013, Riga, Latvia, p. 387–396. ISBN 978-9984-818-58-0
Transport and Telecommunication Institute, Lomonosova 1, LV-1019, Riga, Latvia

LOW-COST PROGRAMMABLE HARDWARE SOLUTIONS IMPROVING CONFIDENTIALITY IN COMPUTER SYSTEMS

Jarosław Sugier

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

This paper presents implementations of contemporary symmetric ciphers in low-cost programmable FPGA devices. The implementations were prepared in the same environment and with the same design methodology for the three selected representative algorithms: AES (Rijndael) and Serpent known from the AES contest, and Salsa20, a relatively newer proposition of a stream cipher submitted and evaluated in the eSTREAM programme. As the hardware platform two families of popular devices form Xilinx were used: standard Spartan-3 and more advanced Spartan-6. In the paper, after introductory analysis of the elementary operations utilized in the ciphers and a brief discussion of their possible realizations with available FPGA resources, purely iterative implementations of the three algorithms are proposed where only one cipher round is represented in hardware and its computation is completed in one clock cycle. Optionally, for Salsa20 cipher an organization with two rounds represented in hardware is additionally included for comparison. The presented results that were obtained after automatic implementation of the designs with Xilinx software provide source material for evaluation of the potential of the three ciphers when they are implemented in specific environment of contemporary low-cost FPGA devices and also offer particular comparison about how the situation changes with development of new, more powerful devices.

Keywords: symmetric cipher, stream cipher, AES, Serpent, Salsa20, FPGA, iterative architecture

1. Introduction

Reliable and dependable operation of numerous contemporary computer systems is based on protection of data security and confidentiality. This is most often assured with application of appropriate cryptographic methods in data transmission and storage, with symmetric ciphers being chronologically first, most matured and still most widely used method employed for this purpose. Apart from software implementations in CPU-based systems, realizations of the ciphers directly at the gate and flip-flop level as a dedicated hardware often remains the only option when very high efficiency of data encryption or decryption is required. Today, with ever-growing capabilities of programmable Field Programmable Gate Arrays (FPGA) their implementations with this category of hardware become more and more popular.

In this paper we propose iterative implementations of the three well-established ciphers – AES, Serpent and Salsa20 – in two low-cost, popular-grade FPGA devices from Xilinx: Spartan-3 and Spartan-6. To achieve low-cost i.e. size efficient implementation, from numerous organizational options it was chosen to articulate the ciphers in purely iterative architecture: for each cipher, only one round was reflected in hardware and the data was passed through it as many times as it is required to complete the whole processing, with computations of each round executed in one clock cycle.

The terms "popular-grade" or "low-cost" that we refer to in the title and in the text have determined our design methodology and are understood as follows: 1) the programmable devices used for implementation are chosen from inexpensive, popular and commonly used line of FPGA chips, widely available on the market; 2) the design is described in hardware description language (HDL) on relatively high level of abstraction (no less than at Register Transfer Level, RTL); 3) after specification in HDL the design is synthesized and implemented fully automatically by standard software provided by the chip manufacturer, without any special "handmade" optimisation, neither in layout nor routing.

This paper is organized as follows. In the next section, to sketch a historical background, we briefly outline the recent progress in contemporary cryptology that resulted in development of the three ciphers investigated in this works, as well as we introduce these algorithms briefly. In section 3 we discuss specific issues of cipher implementations in FPGA devices: feasible options of hardware organization for a round-based cipher, specific resources available in the selected device families and problems that may arise in their applications to elementary operations found in the cipher transformations.

Finally, section 4 presents specific iterative architectures designed for all the three ciphers and discusses the results obtained after their automatic implementations in the two device families.

2. Cryptography Methods and Computer Ciphers

In this chapter we will briefly present the history of development of contemporary cryptographic methods from a more general point of view and, specifically, introduce the three ciphers that are investigated in this work. In order to learn all their internal details that are mentioned later in this paper the reader, if necessary, can refer to their official specifications listed in the bibliography section.

2.1 Developments in contemporary cryptology algorithms: a historical background

Chronologically the first and still the most common way of securing confidentiality and safety in computer data transmission or storage is application of a symmetric cipher. In this kind of a cipher the information is encrypted using some secret key and the same key is later required for decryption. The strength of the cipher ensures that encrypted data, without the knowledge of the key, is meaningless to an external observer or attacker and its deciphering is computationally *very* costly: for example, testing all possible key combinations would require thousands of years of computations even using the fastest computers. The encrypted data, though, is only as secure as is the secrecy of the key: once the key disclosed, it can be used by anyone to freely decrypt the information. Keeping the key confidential during its transmission or storage is of crucial importance and presents a main challenge in practical applications when e.g. the encrypter must share the key with the decrypter.

The first symmetric cipher used commonly in public IT systems was the Data Encryption Standard (DES), developed by IBM and standardized by U.S. National Institute of Standards and Technology (NIST) in 1977. It had been used worldwide as the optimal solution until the mid-1990s when its strength was seriously questioned by successful attacks. Due to relatively short length of the DES key (56 bits) it then became possible to complete brute-force attacks (which consisted essentially in simple exhaustive searches of all the 2^{56} key combinations) in more and more acceptable times using specialized hardware platforms and / or distributed computing. A series of "DES Challenges" organized by RSA Laboratories proved that the cipher could be broken in 96 days in 1997, 41 days early in 1998, 56 hours in July 1998 and, ultimately, 22 hours and 15 minutes in January 1999 [13]. Therefore, facing an imminent demise of DES, in January 1997 NIST issued a first call for a successor algorithm, to be called an Advanced Encryption Standard or AES. In contrast to development of the DES (parts of which were classified as confidential triggering a lot of speculations and suspicions), this time NIST cooperated closely with industry and cryptographic community and the selection process was entirely open and transparent.

The request called for unclassified, publicly disclosed encryption algorithm, available royalty-free, worldwide, and it ignited significant interest in the international cryptologist community. In response, a total of 15 new ciphers were submitted from several different countries. After two conferences organized by NIST to promote public examination of the proposals (AES1, August 1998 and AES2, March 1999) the five finalists were announced in August 1999. Their AES2 votes were as follows:

- Rijndael: 86 positive, 10 negative
- Serpent: 59 positive, 7 negative
- Twofish: 31 positive, 21 negative
- RC6: 23 positive, 37 negative
- MARS: 13 positive, 83 negative

During the last AES3 Conference in April 2000 the authors had the last chance to present their proposals and then, in October 2000, NIST announced the final decision, which was consistent with AES2 voting: the winner was Rijndael cipher. Under the new name of AES it was announced the U.S. Federal Information Processing Standard 197 (FIPS 197, [12]) in November 2001. In this paper we will look at the two best competitors of this contest: the Rijndael (from that moment called just AES) and Serpent.

The AES development stirred large international interest and resulted in substantial scientific progress in the area of computer cryptology and cryptanalysis. In 2004 another initiative, now based in Europe, was started: a 4-year project ECRYPT (European Network of Excellence for Cryptology) was funded within the Information Societies Technology (IST) Programme of the European Commission's Sixth Framework Programme (FP6). A number of sub-projects – the so-called virtual labs – were run within, each of them dedicated to a different area: symmetric (STVL) and asymmetric (AZTEC) algorithms, protocols (PROVILAB), implementations (VAMPIRE), and watermarking and perceptual

hashing (WAVILA). In STVL, one of the major efforts was eSTREAM, the ECRYPT Stream Cipher Project, which promoted design of efficient and compact stream ciphers suitable for widespread adoption. As its effect, a portfolio of new stream ciphers was announced in April 2008 and revised in September 2008, with seven stream ciphers grouped in two profiles: software and a hardware one. The third cipher discussed in this paper, Salsa20, was among them.

Both AES finalists investigated here – Serpent and Rijndael – belong to the same class of round-based cipher algorithms and bear significant resemblance. Processing in both encoding and decoding is split into rounds – the block of data is repeatedly altered by a set of (invertible) transformations constituting a round, with optional slightly modified initial and final transformations. Each round needs a separate round key which is computed from the external (user) key and is supplied by a separate processing path running along the data path, being often of equal complexity. To summarize the distinction between the two ciphers shortly, Rijndael is faster (having fewer rounds, although with a bit more complex operations within) but Serpent is more secure. After the NIST final decision most of the attention concentrated on Rijndael for obvious reasons, but second-to-the-winner Serpent still deserves consideration because of its advantages that won significant appreciation during the AES contest (it is worth noting that in the AES2 ballot it received the least number of negative votes).

Serpent, according to the authors [2-3], was designed to provide users with the highest practical level of assurance, so that no shortcut attack would be found. Furthermore, only well understood mechanisms were considered so one could rely on the existing experience of block cipher cryptanalysis. To provide additional confidence for many years to come authors applied twice as many rounds as were, in their opinion, sufficient to block all shortcut attacks known at that time. Moreover, special care was paid to efficient software computation of the cipher. Its elementary operations were defined in a way that facilitated special optimisations geared towards bit-slice implementations. In fact even the official specification of the algorithm in [2] included extended discussion of effective bit-slice implementation of basic transformations using MMX instructions available in Pentium processors. On the other hand a supplementary publication of the authors [1] evaluated implementation of the cipher in simple 8-bit microprocessor systems with limited memory resources.

Salsa20, on the other hand, differs significantly from both AES finalists and represents a different idea of cipher organization. Being a hash function at its core, it also has a very uniform round-based internal structure, but it does not use round keys – the user key is only incorporated in the input data block, which is processed in its entirety by the rounds.

2.1 The AES, Serpent and Salsa20 ciphers

AES [12] and Serpent [1-3] both are symmetric block ciphers that are examples of substitution-permutation networks (SPN). Their processing consists in a set of *rounds*, with every round defining a specific set of *elementary operations* executed repeatedly over a given *block of data*, called the *state*. Independently from cipher (data) path there is a separate processing path whose task is to provide every round with its individual *key*, generated form user-supplied secret *external key*.

In this work we analyse versions of the two ciphers where encoded data block, internal state and the user key are all 128b[it] wide. Processing of the AES is divided into exactly 10 rounds plus one auxiliary executed at the beginning of the process and there is also a total of 10 round keys used (exactly one in each round). In Serpent the transformation flow is divided into 32 almost identical rounds but since the last round needs two keys, total of 33 different round keys must be generated.

Salsa20 [4-5], on the other hand, represents a different approach in cipher definition. At its core the method is based on a hash function which operates in the counter mode as a stream cipher: the 64B[yte] input consisting of 32B of the *key* (or twice repeated 16B key) together with 8B *nonce* (*number once* – a unique message identifier) plus 8B *counter* and 16 constants bytes is hashed into 64B result which is then XOR'ed with the plaintext. State of the cipher is also 64B wide and is represented as a series of 4B *state words*. The hash result is XOR'ed with plaintext to give ciphertext (during encryption) or with ciphertext to give plaintext (during decryption). The simple XOR operation as the final and the only transformation applied to the plaintext makes encryption and decryption equally efficient. It also allows to use the same hash module in both operations what significantly simplifies either software or hardware implementations. In such a scheme of processing there is no feedback of the data stream to the hash stream. The internal structure of the hash function is again round-based: a set of 20 nearly identical rounds are executed over the 64B state but, in contrast to "classic" AES designs, the rounds do not use keys and there is no separate path for key computations: the external key is used only as the input to the first round along with other data.

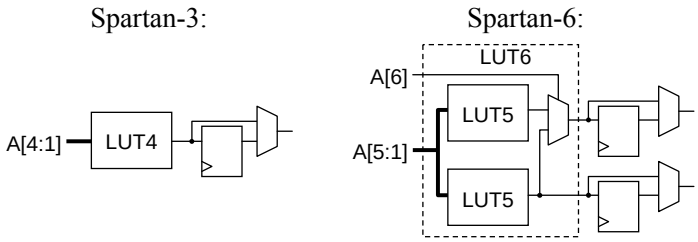

Figure 1. Configurable resources of a single logic cell in Xilinx Spartan-3 and Spartan-6 devices

All the three ciphers share a set of similar elementary operations that make up the rounds: bit-wise XOR sum of the bit vectors, their rotations by a constant number of positions, bit substitutions of either 8b/8b or 4b/4b form, an arithmetic sum of the vectors interpreted as binary numbers, and non-linear transformations which in digital representation are expressed as complex and irregular XOR networks.

Further detailed presentation of the three ciphers investigated in this work can be found in the bibliography positions [1-5,12] while their various hardware implementations, including applications on the FPGA platforms, are described in [6-11, 14-17, 20].

3. Implementing the Ciphers in FPGA Devices

Any round-based cipher – including all three ones considered in this paper – can be efficiently implemented in a CPU-based digital system in software using an iterative scheme: operations of a single round are expressed in the code once and then applied to the state variables repeatedly in an iterative loop r times (with r standing for number of rounds). Parallel execution of multiple program threads which is viable in contemporary multi-core processors can be utilized to speed-up computation of one round provided that its processing can be separated into multiple independent tasks. In Salsa20, for example, the quarterround transformations are ideally suited to such a parallelization while AES and Serpent do not offer such simple opportunity.

When transferring the algorithm directly down to digital elements in hardware (logic gates and flip-flops) the designer is facing a larger diversity of feasible implementation options. In general, there are two opposite extreme approaches: the iterative loop of the cipher can be completely unrolled with all the rounds replicated in hardware as a cascade of r modules, or the loop is not unrolled at all with just one round module implemented in hardware and its operation on the set of state signals is repeated r times. Furthermore, as a mid-range solution the loop can be unrolled in part: one fourth, for example, of the rounds can be reproduced in hardware and the state signals are passed through four times. Moreover, in the case of purely iterative organization with the loop not unrolled processing of the single round can be further divided into multiple execution of a sub-module – again, in the case of Salsa20 these would be the quarterround function – and thus computations of one round can be accomplished in multiple steps, i.e. in multiple clock cycles.

The choice of architecture type usually depends on cost criteria defined specifically for the particular application environment. The decision reflects the optimal balance between required speed vs. acceptable size and power consumption of the hardware: organizations with completely unrolled loop can finish encoding in one clock cycle but for contemporary ciphers with many rounds of complicated processing they usually demand very large (or even huge) amounts of resources. On the other hand, the iterative organizations need to implement in silicon just one round or even a part of it (e.g. one fourth in case of a single Salsa20 quarterround) so the size of the design becomes reduced to a small fraction but the speed of the processing is usually equally decreased.

3.1 Specifics of the FPGA resources

Xilinx, Inc. is the inventor of Field Programmable Gate Array (FPGA) devices and still one of the most successful suppliers of these circuits worldwide. It was decided to choose two popular-grade device families from this manufacturer for case studies included in this work: an older, now more archetypal, Spartan-3 [18] and a newer Spartan-6 [19]. In this point we will concisely discuss the basic aspects of these two architectures that affect efficiency of cipher implementation, in both size and speed characteristics of the resultant hardware.

For a simplified structure of a logic cell in the two Xilinx families, please, see Figure 1. In all FPGA devices from this manufacturer, the so-called *Look-Up Table* (LUT) is the element located in every

cell, which is provided for generation of any combinational function. A single LUT is a ROM table filled with zeroes and ones during configuration according to the function which should be presented at its output. In case of Spartan-3 devices, the LUTs are 4-input tables holding 16b each thus they can generate any function of maximum 4 variables. A function of fever variables still must occupy one LUT while any wider function will use more of them (5-input function = 32b or 2 LUTs, 6-input function = 64b or 4 LUTs, etc.). In Spartan-6 architecture, in turn, every LUT table has the total capacity of 64b being sufficient for generation of any 6-input Boolean function but, alternatively, can be configured for generation of any two 5-input functions provided that they share the same set of input variables.

A 6-input / 2×5-input LUT generator found in Spartan-6 may present a significant advantage in implementation of *wide* Boolean functions (i.e. functions of many variables) when compared to 4-input LUTs of Spartan-3. As we have shown in [14], this can be especially beneficial e.g. in a case of the AES substitution boxes which create an important and substantial part in definition of the cipher. While a Serpent s-box is a 4-input function, which fits efficiently in a single Spartan-3 LUT element, AES s-boxes have 8 inputs and their mapping in Spartan-3 is problematic. In such case switching to Spartan-6 gives noticeable improvements that are not so much evident when moving Serpent to the new architecture.

Additionally, in both architectures the signal, which goes out of the LUT, can be optionally stored in the flip-flop, so virtually every signal generated in the array can be easily synchronously registered: introducing some amount of registers into the FPGA project, like it is e.g. in pipelined designs, usually can be accomplished at very little additional cost.

In both Spartan families two logic cells represented on Figure 1 make up *a slice* – an elementary unit of FPGA organization which includes two LUTs and 2 (Spartan-3) or 4 (Spartan-6) flip-flops. Size of any design after its implementation in the array is usually expressed as a number of occupied slices with numbers of utilized LUTs and registers communicated as the two supplementary measures. Moreover, it should be noted that in real designs all resources (all LUTs and all registers) are never actually utilized inside *every* occupied slice; in some of them the registers adjacent to occupied LUTs remain idle or – conversely – only a register is utilized and the related LUT remains unused.

3.2 Hardware implementations of cipher elementary transformations

In all the three investigated ciphers we can find a limited set of similar elementary transformations which are executed over the state words. The following points will discuss specifics of their implementations with resources available in the FPGA families under consideration.

(A) Byte substitution (AES, Serpent). `SubByte` in AES or `SBox` in Serpent serves as a non-linear transformation that operates independently on state bits using a substitution table. Putting aside fundamental mathematical properties of such transformation and their cryptographic merit, in digital implementation both operations transcode each word of the State to another value that is read from a static table defined in the standard. If the operation is to be performed in parallel on all bits of the State (a principal option to choose when striving for high throughput) then the transcoding modules of such type need to be multiplicated one by another in appropriate quantity.

In AES, to transform the state, 16 s-boxes (each 8b wide) need to be applied in parallel. The modules must be implemented carefully in hardware though, not only because the fully parallel implementation of the algorithm with unrolled loop may generate large number of s-boxes, but also because a function of 8 bits is considered wide and its implementation may be problematic in case of the older Spartan-3 architecture, as it was already signalled in the preceding point. In Spartan-3, a distributed memory module storing one AES s-box table would take 256 × 8 / 16 = 128 LUT elements + additional logic for address decoding. Multiplying this number by 16 (all State bytes transcoded in parallel in one cycle) we reach total of 2048 LUTs for storage of one `SubByte` operation alone. For this family of devices this can be a significant load especially when such a module is replicated for each round (×10) as it may be in some architecture. Switching to a new Spartan-6 architecture with true 6-input LUTs brings a noticeable improvement in case of this cipher: single substitution would need 256 × 8 / 64 = 16 LUTs (6b input ones) per byte or 256 LUTs per complete `SubByte` operation (the whole state matrix). Moreover dedicated FiMUX elements would greatly simplify additional address multiplexing.

In Serpent the substitution is defined for data chunks two times smaller and parallel processing of the whole state data needs 128b / 4b = 32 modules. The observations made above for AES regarding different implementation options are still valid but the numbers are different: every Serpent s-box implemented as a 16×4 distributed ROM would need just 4 LUTs in Spartan-3 or 2 LUTs in Spartan-6, giving a total of 128 or 64 LUTs per round. Moreover, unlike AES which applies repeatedly the same 8x8

substitution, Serpent defines 8 different 4x4 s-boxes (i.e. mappings of 4 bits into 4 bits) with each round using its dedicated version. As a result each s-box is used in precisely four rounds in the whole loop, and in each of these it is used 32 times in parallel to transform the whole 128-bit state block. Since the Spartan-3 LUTs are 4b wide, they are instantly sufficient for implementation of the substitution. Moving to Spartan-6 does not brings any improvement here; as a matter of fact, the 6b LUTs would operate here in two halves as two 5b LUTs to compute 2 of the required 4 transformed bits, thus utilizing just half of its potential (two 4b LUTs would suffice).

(B) Row shifting (AES only). This operation performs cyclic rotations of rows in the 4×4 State array: the i-th row ($i = 0…3$) is rotated by i columns to the left (encryption) or to the right (decryption). In software this would require $3 \times 4 = 12$ byte transfers (the first row with $i = 0$ is actually not rotated), but in hardware, since no data is modified, this is just static signal re-ordering that is accomplished completely in routing and does not absorb any logic resources.

(C) Column mixing (AES) and linear transformation (Serpent). In both ciphers this constitutes linear transformation that complements the non-linear substitution boxes. In AES, each column of the state matrix is treated as a polynomial over GF(2^8) and is multiplied modulo $x^4 + 1$ with specific fixed polynomial $a(x)$ (encryption) or its inverse $a^{-1}(x)$ (decryption). Numerically this is implemented as a matrix multiplication where each output byte is computed through series of `xor` and shift operations that, from the hardware point of view, make up a combinatorial circuit. Additionally it should be noted that coefficients of $a^{-1}(x)$ polynomial are bigger than those of $a(x)$ and as a result the effective structure of `xor` network is remarkably more complex for `INVMixColumn` operation. Consequently, decryption unit usually takes up more resources and has lower maximum frequency of operation.

In Serpent, although this operation is based on different theoretical background, it also results in an irregular `xor` network and its implementation, since very little optimisation at the gate level can be done, presents a challenge for the place-and-route tool.

(D) Bit-wise XOR addition (AES, Serpent, Salsa20). This simple operation adds bit per bit (mod 2) the state words: 128 `xor` gates (128 LUTs) are needed to accomplish this for the whole state in AES and Serpent, while in Salsa20 such operation is repeated four times within each quarterround but for smaller – 32b – words. For hardware implementations this is a straightforward operation and does not pose such difficulties in routing as the previous one because it avoids irregular complexity of intermediate signals. Every such operation creates 128 2-input functions for AES and Serpent (32 2-input functions for Salsa20) which, without optimisation, would require 128 LUT elements in Saprtan-3 or, thanks to possible generation of two functions in a single LUT, 64 elements in Spartan-6 (32 / 16 elements for Salsa20). In any case, since the functions are of only two variables, the LUTs would not be used to their full potential and only the optimisation procedures of the implementation tool can improve this by combining the XOR function with some another transformation adjacent in the processing dataflow chain. Such merging was not explicitly defined in the VHDL code in the projects discussed later in this work, though.

(E) Rotations by the constant amount of bits (AES, Serpent, Salsa20). Such transformations appear throughout the processing in all the discussed ciphers but, like it was in the case of row shifting, although in software these would require due amount of processor cycles for word rotations and transfers, in hardware, since no data is modified, they are just static bit permutations that are accomplished completely in routing and do not occupy any logic resources nor induce any extra delay in data propagation.

(F) 32b addition (Salsa20). This transformation, which is used exclusively in the last cipher, represents a standard task for the hardware representation. Adders are typical and very well-known functional units which are used in many digital designs and FPGA chips have special dedicated resources for their implementation in the form of fast propagation paths for carry signals. As a result 32b adders can be implemented with good efficiency, although their large amount does pose some difficulty in laying out the design since they must occupy continuous fragments of the programmable array.

4. Iterative Implementations of the Three Ciphers

In this chapter we will present iterative implementations of the three ciphers: AES, Serpent and Salsa20. After discussing particular issues of iterative organizations of each cipher, some of which required specific solutions to problems of synchronization between data processing and key generation, results of practical implementations are presented.

In all cases the designs were created by porting the specifications to the VHDL language using strict RTL style: there were no instances of library elements, no sequential (procedural) descriptions were

inserted and the code was free from references to any specific hardware attributes. After definition of all internal signals as `std_logic_vector` type, particular elementary operations were defined as separate entities with exception of key mixing, which was implemented simply with built-in `xor` operator at the place of their occurrence, and mod 2^{32} addition, which was expressed simply with the '+' operator overloaded for `std_logic_vector` arguments. Substitution boxes, both 8b (AES) and 4b (Serpent), were defined according to general templates recommended for ROM specification. AES row shifting and other rotations in all ciphers were treated as simple bit reordering of the respective vectors and expressed directly within concurrent signal assignments. The column mixing (AES) and linear transformation (Serpent) were converted to pure XOR networks and represented with due number of concurrent assignments of logical expressions.

4.1 Architecture of the AES module

It was relatively simple to express architecture of this cipher using the "10 × 1" iterative scheme with just one round implemented in hardware. A module representing such a single round was supplemented with necessary multiplexing logic (loading the data in – looping back – loading the data out) and a simple controller responsible for counting the repetitions of the loop (round numbers) and switching the multiplexers. The controller, in its minimal form, comprised a single "idle/busy" flip-flop and a round counter mod 11. In execution of the last round there is no column mixing, so additional multiplexer was necessary at the input of the key mixing unit which was responsible for loading shift rows output when round number was equal 10.

Very suitably for iterative hardware implementation, in AES no special resynchronisation between data and key computation was required: since the first round (numbered 0) uses the external key, its special preparation is not required. Instead, during the first clock cycle when the S_1 state vector is computed, simultaneously the K_1 round key can be prepared form the external key K so that it is ready for round R_1 in the next cycle. The consecutive rounds work in the same way: R_i (i.e. the S_{i+1} signal) is computed in parallel with simultaneous preparation of K_{i+1}.

4.2 Architecture of the Serpent module

Also in case of this cipher only one round was implemented in hardware ("× 1" organization) but its more irregular structure made the iterative scheme more complex.

Firstly, while in the AES there is just one s-box transformation used in all rounds in both data and key processing, the Serpent defines 8 different s-boxes, each one being applied in exactly four rounds in the cipher path and in another four rounds in the key path. In iterative organization where just one "universal" round is realized in hardware this means that a "universal" s-box had to be created which included the contents of all 8 regular substitution tables and additionally provided extra 3b input for selection signal for an internal 8:1 multiplexer. Such a solution is not elegant because, effectively, the s-box becomes a 7-input function (4b data + 3b selection) in place of a 4-input one, which makes its implementation with FPGA resources notably more complicated. For this reason one-round iterative implementation is usually not optimal for Serpent; as an alternative it could be considered to implement 8 rounds in hardware (each holding one s-box) with the data block looped back 4 times during the encoding ("8 × 4" scheme instead of "32 × 1"). Nevertheless, such organization would represent significant loop unrolling and it was not selected for this study for consistency of examined solutions.

Secondly, the more involved key generation in Serpent complicated synchronization with processing of the coded data. The first problem was that computation of key K_i in iteration i depends on prekeys w from not only the directly preceding iteration $i - 1$ but also $i - 2$, so additional registers holding previous values of w were required. Furthermore, also the last cipher round needed special modification. Because it uses two keys – K_{31} and K_{32} – to follow the pace of iterative key generation it also had to be split into two iterations: the first one (no. 32) executed key mixing with bit substitution and the second one (no. 33) performed only key mixing. This increased total length of the loop to 33 iterations and complicated round implementation which had to take into account different data flows in the last two iterations creating additional multiplexers between transformation units. An alternative solution – computing two keys for the last iteration – would create even more problems that would degenerate performance to a greater degree.

Even with these modifications, the result was not optimal: in such organization computation of the round key and actual encoding of the data were performed sequentially: in every clock cycle first the key K_i was computed and only then the cipher path could start its processing: the data block was mixed with K_i, and the other two transformations were executed. These two segments of computations are independent and

could be executed in parallel provided that the cipher pipeline is delayed for one clock cycle so that the round R_i is computed one cycle *after* the key K_i. Such a solution required a very simple modification: additional 128 flip-flops were placed right at the beginning of the cipher path and they created an additional delay stage at the entrance to the loop generating the required 1-cycle delay in data path. The total number of loop iterations was further increased from 33 to 34 but reduction in minimum clock period compensated this increase more than adequately.

4.3 Architecture of the Salsa20 modules

Iterative implementation of this cipher was the simplest one of all three investigated algorithms mainly thanks to its very regular structure and the absence of the separate key generation path (the key is just loaded with other input data only at the beginning of computations and all the 512b of data is then processed in unity). The only exception in this regularity is – different input permutations of even vs. odd numbered rounds what defines the difference between column- and row- rounds ([5]). For this reason it was decided to investigate two simple variants of iterative organization of this cipher: "10 × 2" (denoted later as SalsaX2) and "20 × 1" (SalsaX1).

The "10 × 2" variant needed 10 clock cycles to complete the processing of a single block before the next one can be loaded but it had implemented in hardware the whole double round, i.e. a column round followed by the row round, each comprising four instances of quarterround entities. Since all the ten repetitions of the double round are *strictly* identical, implementation of this scheme was as simple as possible.

The "20 × 1" variant encoded one block of data in 20 clock cycles and in general it contained the same simple control logic as the "10 × 2", but here the 20 iterations were not exactly the same: the even-numbered rounds (when numbered from 0 to 19) should apply the column round transformation while the odd-numbered ones – the row round one. In both cases the hardware consisted of just four quarterround modules and they were replicated in hardware only once but additionally two blocks of extra multiplexing of inputs and outputs were necessary to differentiate permutations of the state words in the even and odd iterations, based on the least significant bit of the round counter.

4.4 Implementation results

Table 1 presents parameters of the designs after their implementation in XC3S2000-5FGG676 (Spartan-3) and XC6SLX150-3FGG484 (Spartan-6) devices. In all cases the VHDL specification was synthesized by the XST synthesis tool in Xilinx ISE design suite and then implemented for the given two devices. The implementation was fully automatic, without none hand-made optimisations neither in placement nor in routing. For every design the table specifies basic speed and size parameters: the minimum clock period (T_{clk}) and resulting maximum operating frequency (f_{max}) as they were estimated by the post-place & route static timing analysis, the latency expressed in clock cycles and in nanoseconds, the overall throughput in Gbps calculated for the given f_{max}, and size of the design expressed in the number of occupied slices, LUTs and registers. As a synthetic efficiency measure which is commonly

Table 1. Parameters of the cipher implementations in Spartan-3 and Spartan-6 devices

	Spartan-3				**Spartan-6**			
	AES	**Serpent**	**SalsaX2**	**SalsaX1**	**AES**	**Serpent**	**SalsaX2**	**SalsaX1**
min T_{clk} [ns]	13.0	10.4	51.7	24.7	6.3	5.6	20.8	12.4
f_{max}[MHz]	77.0	96.2	19.4	40.4	160	180	48.0	80.5
Latency [T_{clk}]	11	34	10	20	11	34	10	20
Latency [ns]	143	353	517	495	69	189	208	249
Throughput [Gbps]	**0.875**	**0.354**	**0.99**	**1.04**	**1.81**	**0.66**	**2.46**	**2.06**
Mbps / Slice	**0.15**	**0.17**	**0.49**	**0.56**	**3.77**	**1.26**	**3.00**	**2.60**
Occupied slices	**5 948**	**2 145**	**2 036**	**1 858**	**493**	**536**	**818**	**791**
Number of slice LUTs	7 986	3 995	3 374	3 250	1 367	1 566	2 955	2 611
Number of slice registers	781	783	1 286	1 294	817	806	1 317	1 296

used for evaluation of speed vs. size balance – Mbps of the throughput per one occupied slice (Mbps/slice) is given.

When analysing the results it should be kept in mind that Salsa20 module works on 512b of data while the two other ciphers – only on 128b (although there are another 128b of "state" in key computation), so the raw size of the designs should be interpreted with this modification. Therefore, on the Spartan-3 platform it is surprising to see the Salsa20 – in both variants – occupying the same or even lower number of slices as the Serpent and significantly smaller number than the AES. This also illustrates that AES implementation in Spartan-3 is problematic due to insufficiencies in representation of 8b-wide s-boxes found in this cipher. These problems are not present on the newer Spartan-6 platform and sizes of the designs approximately return to the expected proportions: the AES implementation is actually a little smaller than the Serpent and both of them occupy significantly less area than the Salsa20 modules.

If looking at the raw throughput figures, the Salsa20 cipher offers the fastest rate of data processing: although in Spartan-3 device the AES tries to keep up the pace, in Spartan-6 the lead is evident. Serpent, on the other hand, seems to be the slowest contender in this competition but this probably could be justified by its incompatibility with "32 × 1" iterative scheme that was selected for this comparison; apparently an "8 × 4" organization could offer some improvement.

Finally, comparing the parameters of the two Salsa20 variants it can be seen that the size difference is far from 1:2 ratios as one could expect: number of slices in SalsaX2 architecture is greater only by 5-10% on both platforms despite the fact that it includes implementation of two rounds vs. just one in SalsaX1. This shows that the overhead introduced by the extra multiplexing on the input and output of odd/even numbered rounds is quite large and it counterweights the expected savings coming from half the number of rounds implemented in silicon. The performance comparison, in turn, depends on the hardware platform. In the older architecture of Spartan-3 the reduction in SalsaX1 T_{clk} is almost to the expected 50% so the overall throughput and Mbps per slice are practically the same for the two variants. In Spartan-6, on the other hand, again we see not so large decrease and therefore the performance is in favour of the SalsaX2 implementation.

5. Conclusions

In this paper we have successfully implemented the three ciphers: AES, Serpent and Salsa20 in purely iterative architectures using as the hardware platform Spartan-3 and Spartan-6 FPGA devices from Xlinx. The results have shown that from the two classic ciphers: AES and Serpent, the official U.S. standard can offer the best overall performance in terms of Mbps/slice only if the newer and more powerful Spartan-6 devices are used: its elementary operation of byte substitution is too complex for logic resources available in the array of the older Spatan-3 family.

It is also interesting to see effectiveness of the relatively new Salsa20 cipher compared with more mature and also more established contenders form the 1997 NIST competition. Although actually a hash function at its core, the Salsa20 algorithm is also promoted by its author as the more secure and the faster alternative to the AES standard. Even though it was selected to the final portfolio of the eSTREAM project ciphers only in the software profile (and not in the hardware one), this work verifies Salsa20 potential when it is implemented in popular FPGA devices: in Spartan-6 the "10 x 2" iterative organization can reach the best raw throughput of nearly 2.5 Gbps vs. 1.8 Gbps of AES, although with regard to Mbps/slice efficiency it is inferior to the standard.

As the final remark it should be noted that in order to complete such a comparison the cryptographic quality of the three ciphers should also be evaluated – but such evaluation is often inconclusive even in expert debates and remains beyond the scope of this work.

References

1. Anderson, R., Biham, E., Knudsen, L. (1998). Serpent and Smartcards. In *Lecture Notes in Computer Science: Smart Card Research and Applications* (pp. 246-253), Vol. 1820. Berlin Heidelberg: Springer Verlag.
2. Anderson, R., Biham, E., Knudsen, L. (1998). Serpent: A Proposal for the Advanced Encryption Standard. In Proceedings of the First Advanced Encryption Standard (AES) Candidate Conference, Ventura, California, 20–22 August 1998. Retrieved March, 2013, from http://www.cl.cam.ac.uk/~rja14/serpent.html

3. Anderson, R., Biham, E., Knudsen, L. (2000). The Case for Serpent. In Proceedings of the Third AES Candidate Conference (AES3), April 13–14, 2000, New York, USA. Retrieved March, 2013, from http://csrc.nist.gov/archive/aes/index.html
4. Bernstein, D.J. (2008). The Salsa20 family of stream ciphers. In *New Stream Cipher Designs*. Berlin, Heidelberg: Springer Verlag,
5. Bernstein, D.J. (2005). The Salsa20 Stream Cipher. In Proc. SKEW - Symmetric Key Encryption Workshop, Aarhus, Denmark, 26–27 May 2005. Retrieved April, 2013, from http://cr.yp.to/snuffle.html
6. Gaj, K., Chodowiec, P. (2000). Comparison of the hardware performance of the AES candidates using reconfigurable hardware. In Proceedings of the Third AES Candidate Conference (AES3), April 13–14, 2000, New York, USA. Retrieved March, 2012, from http://csrc.nist.gov/archive/aes/index.html
7. Gaj, K., Southern, G., Bachimanchi, R. Comparison of hardware performance of selected Phase II eSTREAM candidates. In Proc State of the Art of Stream Ciphers Workshop, eSTREAM, ECRYPT Stream Cipher Project, Vol. 26, p.2007.
8. Good, T., Benaissa, M. (2007). Hardware results for selected stream cipher candidates. In Proc State of the Art of Stream Ciphers Workshop (pp. 191-204).
9. Krukowski, Ł., Sugier, J. (2010). Designing AES cryptographic unit for automatic implementation in low-cost FPGA devices. *Int. J. Critical Computer Based Systems*, 1, 104–116.
10. Lázaro, J., Astarloa, A., Arias, J., Bidarte, U., Cuadrado, C. (2004). High Throughput Serpent Encryption Implementation. In *Lecture Notes in Computer Science: Field Programmable Logic and Application* (pp. 996-1000), Vol. 3203.
11. Liberatori, M., Otero, F., Bonadero, J.C., Castineira, J. (2007). AES-128 Cipher. High Speed, Low Cost FPGA Implementation. In Proc. of the Third Southern Conference on Programmable Logic. Mar del Plata, Argentina: IEEE Comp. Soc. Press.
12. National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES). *Federal Information Processing Standards Publication 197*. Retrieved March, 2013, from http://csrc.nist.gov/publications/PubsFIPS.html
13. RSA Laboratories: DES Challenges. http:// www.rsa.com.
14. Sugier, J. (2011). Implementing AES and Serpent ciphers in new generation of low-cost FPGA devices. In: *Advances in Intelligent and Soft Computing: Complex Systems and Dependability* (pp. 273-288), Vol. 170. Berlin Heidelberg: Springer Verlag.
15. Sugier, J. (2011). Implementing Serpent cipher in field programmable gate arrays. In Proceedings of the 5th International Conference on Information Technology, ICIT 2011 Amman, Jordan, May 11-13 (pp. 91-96). Amman, Jordan.
16. Sugier, J. (2010). Low-cost hardware implementation of Serpent cipher in programmable devices. In *Monographs of System Dependability, Vol. 3: Technical Approach to Dependability* (pp. 159-172). Wrocław, Poland: Publishing House of Wrocław University of Technology.
17. Sugier, J. (2013). Low-cost hardware implementations of Salsa20 stream cipher in programmable devices. *J. Polish Safety and Reliability Association*, 4(1), 121-128.
18. Xilinx, Inc. *Spartan-3 Family Data Sheet*, www.xilinx.com, access: April 2013.
19. Xilinx, Inc. *Spartan-6 Family Overview*, www.xilinx.com, access: April 2013.
20. Yan, J., Heys, H.M. (2007). Hardware implementation of the Salsa20 and Phelix stream ciphers. In Proc. Canadian Conf. Electrical and Computer Engineering CCECE 2007 (pp. 1125-1128). IEEE.