Journal of Polish Safety and Reliability Association
Summer Safety and Reliability Seminars, Volume 9, Number 3, 2018

Sugier Jarosław
Wrocław University of Science and Technology, Faculty of Electronics, Poland

Efficiency of Spartan-7 FPGA devices in implementation of contemporary cryptographic algorithms

Keywords

hardware implementation, loop unrolling, pipelining, AES, BLAKE, KECCAK, SHA-3.

Abstract

Hardware implementations of cryptographic algorithms are ubiquitous in contemporary computer systems where they are used to ensure appropriate level of security e.g. in high-speed data transmission, authentication and access control, distributed cloud storage, etc.. In this paper we evaluate size and speed efficiency of FPGA implementations of selected popular cryptographic algorithms in the newest cost-sensitive Spartan-7 devices form Xilinx, Inc.. The investigated set of algorithms included four examples: the AES-128 standard symmetric block cipher, the BLAKE-256 hash function and two size variants of the KECCAK-$f[b]$ compression function, $b = 400$ and 1600, with the larger variant being used as the core of the new SHA-3 standard. The main aim of this research was to provide a uniform and comparable implementation approach for all the ciphers so that the new potentials of the Spartan-7 internal architecture would be put to the test in realization of their specific cryptographic transformations and data distribution. Each of the four algorithms was implemented in five architectures: the basic iterative one (with one instance of the cipher round instantiated in hardware) plus two loop unrolled ones (with two and four or five rounds in hardware) and their two pipelined variants (with registers at the outputs of each round enabling parallel processing of multiple streams of data). Uniform implementation methodology applied to 20 cases of cipher & architecture combinations created a consistent testbed, producing comparable results which allowed to evaluate efficiency of the new hardware platform in implementation of the different algorithms in various unrolled and pipelined organizations.

1. Introduction

Contemporary IT systems, being most often geographically distributed complex systems exposed to a hostile environment, must ensure appropriate level of data security and the only way of meeting this demand is application of suitable cryptographic algorithms. In the recent two decades a lot of research was devoted to developing new and more efficient algorithms for this purpose. When high data throughput is required their implementation in hardware rather than software is often the only solution and this aspect of cipher realization is taken into account by their authors right from the start of development. Among the hardware implementation options the programmable devices and in particular Field Programmable Gate Arrays (FPGA) are often the optimal choice thanks to their flexibility, short development cycle and low prototyping cost [6]-[7], [9].

But contemporary ciphers are one of the most difficult kind of designs for implementation in FPGA devices. Their large sizes, irregular internal processing and random data distribution resulting in chaotic yet widely spread routing create a lot of problems for implementation tools and stretch capabilities of the arrays to their limits. Therefore suitability of a specific cipher (which furthermore can be expressed in hardware in various architectures of parallel or sequential processing) for an FPGA implementation is an important issue depending additionally on particularities of the programmable device selected as the hardware platform.

In this context the aim of the paper was to evaluate size and speed efficiency of FPGA implementations of selected popular cryptographic algorithms in the newest cost-sensitive Spartan-7 devices form Xilinx, Inc.. The investigated set of algorithms included four examples: the AES-128 standard symmetric block cipher, the BLAKE-256 hash function and two size variants of the KECCAK-*f*[*b*] compression function, $b = 400$ and 1600, with the larger variant being used as the core of the new SHA-3 standard. Each of these algorithms was implemented in five architectures: the basic iterative one (with one instance of the cipher round instantiated in hardware) plus two loop unrolled ones (with two and four or five rounds in hardware) and their two pipelined variants (with registers at the outputs of each round enabling parallel processing of multiple streams of data). Uniform implementation methodology applied to 20 cases of cipher & architecture combinations created a consistent testbed, producing comparable results which allowed to evaluate implementation efficiency of the different algorithms in various unrolled and pipelined organizations on the new hardware platform which is available commercially only form 2017.

Contents of the paper is organized as follows. Construction of the algorithms is outlined briefly in the second chapter in order to identify their main characteristics which affect efficiency of FPGA implementation. The third chapter describes the five architectures every cipher is implemented in and discusses organization of main Spartan-7 resources which are fundamental in cipher realizations. The results obtained after implementation of the 20 test cases are the subject of the fourth chapter: after evaluation of the main size and performance characteristics the analysis is extended with examination of the derived – loop unrolled and pipelined – architectures in comparison to the basic iterative one which allowed to assess scalability of the loop unrolling mechanism in the different ciphers on the new platform. Conclusions completes the text in the last, fifth chapter.

2. The algorithms

2.1 AES

The Advanced Encryption Standard was developed in an open competition initiated by U.S. National Institute of Standards and Technology in order to create a new symmetric block cipher which would stand sufficiently strong against increasing strength of new attacks backed up by a growing computational power of dedicated cryptographic hardware. The contest ended in 2001 with selecting the Rijndael algorithm – a proposal which came from two Belgian cryptographers: Joan Daemen and Vincent Rijmen.

In this work we will test AES-128 – the 128-bit size version of the cipher [8]. The algorithm in general is a substitution-permutation network which processes the encoded chunk of data – *the state* – in a series of n_r = 10 almost identical *rounds*. Each round uses its own *key* which is generated from the user-supplied external key by a separate key expansion routine. Cipher encoding and key expansion share very similar set of elementary transformations and constitute two 128b-wide processing paths which needs to be executed in parallel, hence the total data width of this algorithm in hardware is 256b.

The state S is constructed as a 4×4 array of bytes and each round apply four elementary transformations upon it in the following order (see the right-hand part of *Figure 1*):

- substitution *SBox* where each byte of the state is replaced by another one according to a specific invertible static transcoding function;
- row shifting *SR* where each k-th row ($k = 0\ldots3$) of the state array is rotated by k columns to the left in encryption or to the right in decryption;
- column mixing *MC* operating on the whole state columns rather than on individual bytes and calculating its result through an involved series of shift and xor operations (which models polynomial multiplication modulo $x^4 + 1$ over GF(2^8));
- key mixing where the round key is bitwise xor'ed with the state vector.

The key schedule (the left-hand part of *Figure 1*) works on a set of four w_i words initialized with the external key and produces the round keys with identically defined *SBox* substitutions and bitwise xor operations.

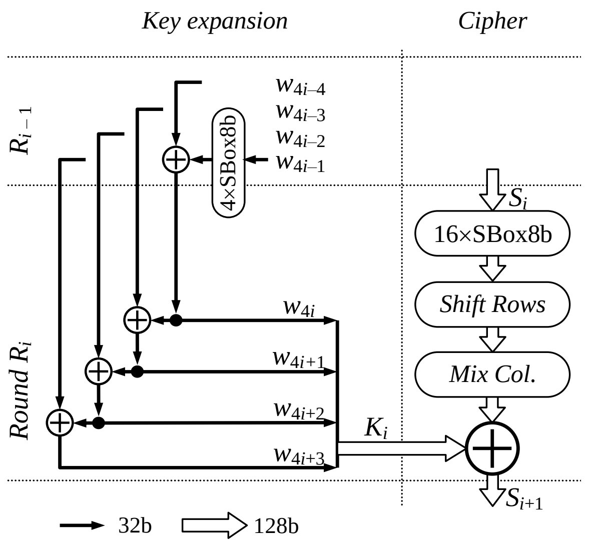

Figure 1. Internal composition of one AES round: key expansion and cipher paths

Putting aside row shifting (which in hardware is just a static signal re-ordering and can be accomplished completely in routing, without absorbing any logic resources), the main challenge in realization of this algorithm in an FPGA array is efficient expression of column mixing and byte substitution. Especially the latter operation, being essentially a wide 8b to 8b boolean function, is difficult in implementation with LUT generators which must be combined to store 256 bytes of its truth table. In the older Spartan-3 devices each *SBox* aggregated 16 LUTs plus some multiplexing logic and this not only produced enlarged design size but also remarkably complicated overall routing leading consequently to long propagation delays and low operating frequencies. It was shown in [11] that these problems were partially alleviated in the newer Spartan-6 family which offered 4 times larger LUT tables.

2.2 BLAKE

Building on positive experiences of the AES contest, in 2007 NIST started another competition aimed at developing a new hash method which would eventually replace the SHA-2 standard. The BLAKE algorithm, proposed by Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan, was selected as one of the strictest 5 finalists and, although by the final NIST decision it ultimately lost to KECCAK, the cipher was repeatedly acclaimed in all stages of competition for its good cryptographic strength and great performance, especially in software.

In this work we are considering the BLAKE-256 size variant which generates 256b hash output, internally handling 32b words and 512b state. At the heart of the method [1] there is a *compression function* which is iteratively applied to 512b blocks of an arbitrarily long input message and the FPGA implementation of this function is the subject of our investigation.

Processing of the compression is organized around a 512b state seen as a 4×4 matrix of 32b words $v_0 \dots v_{15}$. Initially the state is filled with the current chain hash value and then it goes through a series of $n_r = 14$ rounds with each round modifying twice all the words by applying *a G function*:

$$G_0(v_0, v_4, v_8, v_{12}); \quad G_1(v_1, v_5, v_9, v_{13}); \\ G_2(v_2, v_6, v_{10}, v_{14}); \quad G_3(v_3, v_7, v_{11}, v_{15}); \tag{1}$$

and then

$$G_4(v_0, v_5, v_{10}, v_{15}); \quad G_5(v_1, v_6, v_{11}, v_{12}); \\ G_6(v_2, v_7, v_8, v_{13}); \quad G_7(v_3, v_4, v_9, v_{14}). \tag{2}$$

Each application of the $G_i()$ function transforms a set of four state words given as explicit parameters; as an additional side input the message words are also loaded although they do not appear on the argument list. The ordinal number of the function i ($0 \div 7$) determines which permutation element $\sigma_r(i)$, message m_i and constant c_i words are used within the given $G_i()$ instance – but other than that each instance applies the same processing transformations which are graphically represented in *Figure 2*.

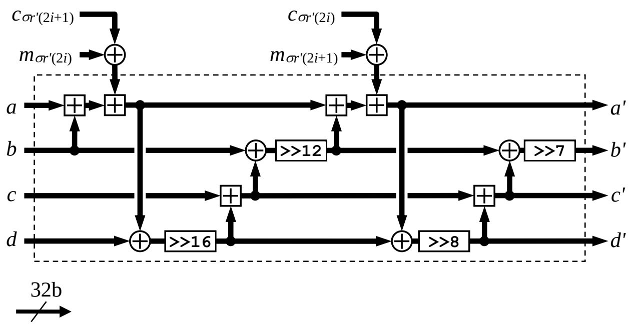

Figure 2. Operations of the *G* function – one eighth of the BLAKE round.

The operations – which are the key elements in the hardware implementation – are:
⊕ - bitwise xor of two 32b vectors,
+ - addition mod 2^{32} of two vectors (i.e. regular 32b addition with carry out ignored),
>> - right rotation of a vector by a constant number of positions.

Again, any rotation of a vector by a constant number of positions is a null-operator in hardware so the repertoire of BLAKE elementary operators is extremely simple: it includes only 32 adders and xor gates. This would be an ideal situation for optimization algorithms applied during the implementation process (a cascade of simple operators would be aggregated in large sized LUT generators) but such an optimization cannot deal efficiently with adders: in FPGA cells fast addition is accomplished with dedicated carry resources located along the LUT elements so the in-LUT aggregation must stop in the points where an addition appears on the data path.

2.3 KECCAK-*f*[400] and *f*[1600]

The KECCAK algorithm ([3]) was proposed for the SHA-3 contest by Guido Bertoni, Joan Daemen (co-author of AES), Michaël Peeters and Gilles Van Assche. The proposal included actually a set of 7 size variants of the method from which the largest one – with 1600b state – in 2015 was selected for the new SHA-3 standard.

In this work we will analyse hardware implementations of the KECCAK-*f*[*b*] permutation function which is at the core of the so called *sponge construction* [2] and calculates actual hash values of the input message. With its size parametrized by l which can take any integer value from 0 to 6, the function operate on a state of $b = 25 \times w$ bits ($w = 2^l$, b = 25, 50, 100, 200, 400 800, and 1600) where a single word w (1, 2, 4, …32b) is called *a lane*. Any specific-size variant of the function computes its result by processing the state in a series of $n_r = 12 + 2l$ rounds (12, 14, 16, 18, 20, 22, or 24). The rounds are internally identical but they apply different *w*-bit constants in their final transformation. For the tests in this paper we have selected implementations of two size variants: a 20-round KECCAK-*f*[400] with the datapath width comparable to AES and the full-size, 24-round KECCAK-*f*[1600] which is used in the SHA-3 standard.

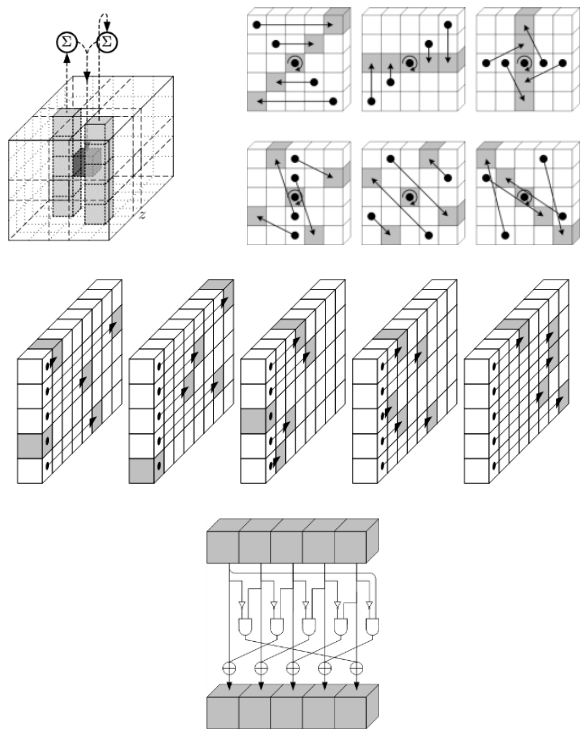

Figure 3. Ideas of bit transformations inside a KECCAK round as presented by the authors in [3].

Both variants (as well as the 5 remaining ones) share the same parametrized definition. The specification describes one KECCAK round as a sequence of 5 transformations of the state *A* which is represented as a 3-dimensional array *A*[5][5][*w*]. Each transformation is defined as a set of elementary operations on individual bits of the 25 lanes. Although the operations are as simple as xor or fixed rotations, their arguments are scattered pseudo-randomly all over the lanes so that it is virtually impossible to determine any regular datapaths inside the round in a manner similar to *Figures 1* and *2*. Of the three ciphers considered in this comparison KECCAK'S operations, although can be algebraically described in a concise way, are the most fine-grained ones. *Figure 3* presents only their basic ideas – yet not the round's complete processing which results from their superposition.

3. Implementing the ciphers in hardware

3.1 Architectures of cipher modules

Any round-based cipher can be efficiently implemented in software in an iterative manner: operations of a single round are expressed in the code once and then applied to the state variables repeatedly n_r times in a loop. Depending on the in-round data dependencies, round processing can sometimes be parallelized and split into multiple concurrent threads of execution. Such a generic iterative idea can be mapped into hardware (either ASIC or FPGA) in different ways [4]. Overall, the two extreme approaches would be as follows: the loop of the cipher can be completely unrolled with all the rounds replicated as a cascade of n_r hardware modules, or the loop is not unrolled at all with just one round module implemented in hardware and its operation on state signals is repeated n_r times (that is, in n_r clock cycles) in a manner resembling iterating in software. Between these extremes, 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 would be passed through them four times.
Another technique – called pipelining – well known in hardware organisation can be applied if there is a cascade of modules creating a long path of data propagation in the design: if registers are added at the end of each round, different sets of data can be processed simultaneously in each of the modules during the same clock cycle which introduces parallelism and increases greatly total throughput of the data stream being processed. Such an approach is usually very effective in cipher organizations because almost every unrolled implementation is naturally a very good candidate for pipelining.
Notations for these cases will be adopted after universal taxonomy presented e.g. in [5]: an architecture with *k* unrolled rounds will be denoted as x*k* (including the basic iterative – x1) and their pipelined counterparts – as PPL*k*.
In this study we followed the approach of our previous comparisons and, focusing on high speed architectures, each cipher was tested in 5 architectures:

- the basic iterative one - x1,
- the two loop unrolled ones - x2 and x5,
- their two pipelined variants - PPL2 and PPL5.

Only in the case of the largest, 24-round KECCAK-*f*[1600] algorithm as the longest unrolled architectures the x4/PPL4 organizations were taken instead of x5/PP5 – in order to keep integer divisions of the n_r parameter.
We should also note that the x1 organization of each cipher, being its smallest and thus easiest to

implement, will serve also as a point of reference in evaluation of the derived, larger constructions.

3.2 Resources of the Spartan-7 devices

Introduced commercially in 2017, Spartan-7 family is a relatively new addition to the latest, most advanced and most sophisticated 7th generation of programmable devices offered by Xilinx, Inc. – the inventor and still one of the most accomplished producers of the Field Programmable Gate Arrays. Fabricated in 28 nm HPL process the S7 chips can efficiently combine high performance with low cost and small form-factor packaging. In the internal organization special care was paid to enhancements in routing capabilities which often limit the effective capacity of the array in real applications: their organization is based on the Advanced Silicon Modular Block (ASMBL) architecture and helps in removing bottlenecks which arise as a result of geometric layout constrains or global signals routing.
Every node of the S7 array – a configurable logic block – consists of two *slices* and *Figure 4* presents essential resources found in each of them [13]. For implementations of any cipher the most important are those used for realization of boolean logic because the combinational functions usually constitute the core of the cryptographic processing. In S7 the boolean functions are generated by 6-input Look Up Tables (LUT) with each individual LUT, being essentially a 64-bit PROM memory, is capable of representing on its O6 output any function of up to 6 variables. Additionally, since the 6-input functions are relatively wide and sometimes larger number of smaller ones are needed in the design, each LUT can be configured to compute two functions of 5 variables (holding two 32-bit tables): in this case the second function is available on the optional O5 output but such a configuration is only possible if the two functions share the same 5 input signals. LUT usage statistics which is generated after implementation provides number of elements in the design with only the O6 output used (generation of one maximum sized, 6-input function), with both O5 and O6 used (generation of two 5-input functions) or only O5 (generation of a 5 or less input function).
Every LUT output can be optionally stored in a flip-flop. As usual in any FPGA array, S7 resources offer an abundance of flip-flops so creating e.g. the pipeline registers is not a problem. Therefore in the evaluations of this work we will not pay too much of an attention to the register usage because they are a secondary resource, insignificant in evaluation of cipher sizes even in the pipelined architectures.
The figure also shows availability of F7 & F8 multiplexers which are used for joining LUT outputs when creating functions of more than 6 variables. Each of the two pairs of LUTs inside the slice (A&B and C&D) can generate any 7-input function on the output of its F7 mux giving two independent functions per slice. Similarly, to implement an arbitrary 8-bit function its 256-bit truth table can be stored in the 4 LUTs so that the function value propagates through a F7 mux to the output of the F8 mux. Thus all the logic of a slice suffices for representation of one 8-bit function – and it is worth noting that in the previous generations of devices with 4-input LUT elements synthesis of such a function was not possible within a slice, required inter-CLB signal routing and thus led to much worse performance parameters.

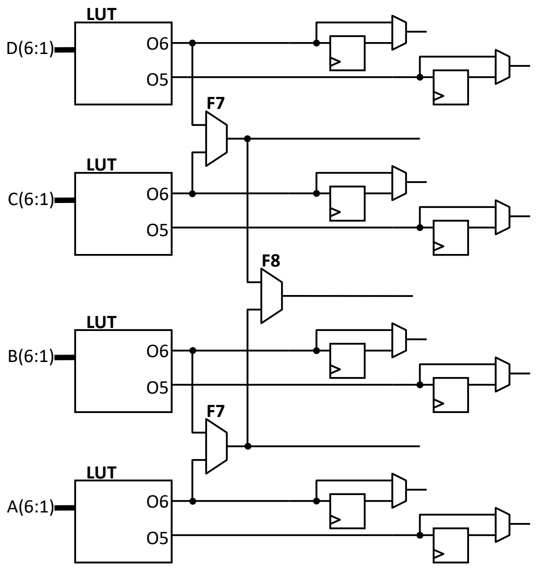

Figure 4. Resources inside one Spartan-7 slice.

4. Implementation results

4.1 Designs for implementation

To create actual hardware implementations of all the 4 algorithms a complete practical designs were prepared: in each one the main cipher unit was equipped with some basic input / output buffers providing means for iterative loading the plaintext and unloading the results. The buffers had actually a basic functionality of serial-to-parallel input shift registers and parallel-to-serial output ones and were necessary because number of input and output bits of the cipher units exceeded 250 available pins even in the selected largest FGAA-484 package. Nevertheless, the buffers consumed only flip-flops (and not any combinational resources) so they interfered very little with the actual cipher units which were, as already noted, heavily logic-oriented.
The designs were automatically synthesized and implemented by Xilinx Vivado software for the Spartan-7 XC7S50 FGGA484-2 device [14]. The chip was sufficiently large to accommodate even the largest organizations: with 32600 LUT elements

available, the most sized BLAKE designs took at most 67% of them while Keccak and AES units fit within, respectively, 36 and 16%.
For the 4 ciphers and 5 architectures the implementation covered a set of 20 particular cases – and their results will be presented in the following points.

4.2 Size parameters

Parameters describing size of the implemented cases are presented in *Table 1*. As it was already pointed in the previous chapter the fundamental size metric is the use of LUT elements which is given as the first parameter for each architecture, followed by a detailed statistics of their O5/O6 usage, a number of occupied slices and utilization of F7 and F8 multiplexers. For a better visualization LUT numbers of all designs are presented graphically in *Figure 5*. The reported sizes are – in general – in accordance with expectations based upon internal sizes of the ciphers, but some interesting variations should be noted.

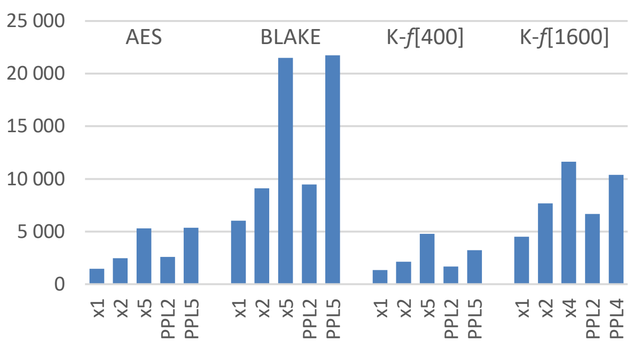

Figure 5. Size of the tested implementations expressed in number of used LUT elements.

The AES has been known for problematic implementations in the older generations of FPGA chips where their smaller, 4-input LUT tables struggled in realizations of 8-bit substitution boxes this cipher uses heavily. In Spartan-7 there are no symptoms of these problems and the AES turns out to be nearly the smallest unit in all organizations, on par with the reduced KECCAK-*f*[400]. As the table proves this is accomplished with extensive use of F7/F8 multiplexers – if there was a substitution box any larger than 8b in the algorithm the problems would probably return. Looking at the lowest across all ciphers LUT O5&O6 usage one can tell that the AES logic was relatively least varied and the offered LUT/F7/F8 in-slice configuration handled its demands quite efficiently.
BLAKE is the other cipher which uses F7/F8 multiplexers equally intensively. Looking at raw LUT and slice numbers we can see that this cipher led to the largest implementations of all the four algorithms – a rather unexpected record because the 512-bit BLAKE's state is approximately 3 times smaller than that of KECCAK-*f*[1600]. Further analyses of the combinational paths of this cipher in the next point will help in explanation why in this case resource utilization is so much above the expected numbers. The unrolled architectures of this cipher – the x5 and PPL5 cases – turned out to be the largest designs of all the cases, surpassing remarkably their KECCAK counterparts.

Table 1. Implementation results: size metrics.

	LUTs	LUT usage			Slices	MUX	
		O5	O6	O5&O6		F7	F8
AES							
x1	1479	0,00%	91,8%	8,2%	499	144	40
x2	2464	0,04%	95,7%	4,3%	799	384	160
x5	5309	0,04%	95,9%	4,1%	1489	984	412
PPL2	2594	0,04%	94,7%	5,2%	869	320	160
PPL5	5360	0,00%	97,0%	3,0%	1503	1280	640
BLAKE							
x1	6048	0,00%	94,0%	06,0%	1663	620	40
x2	9114	0,01%	86,7%	13,2%	2434	1984	992
x5	21510	0,01%	86,7%	13,3%	5575	5120	2560
PPL2	9485	0,03%	84,7%	15,3%	2522	1984	992
PPL5	21736	0,01%	85,0%	15,0%	5783	5120	2560
KECCAK-*f*[400]							
x1	1344	0,00%	99,1%	00,9%	364	0	0
x2	2130	0,28%	82,8%	16,9%	616	0	0
x5	4785	0,10%	86,3%	13,6%	1296	0	0
PPL2	1694	0,24%	77,4%	22,4%	577	0	0
PPL5	3231	0,12%	85,3%	14,6%	963	0	0
KECCAK- *f*[1600]							
x1	4526	0,13%	86,6%	13,3%	1401	0	0
x2	7691	0,01%	82,9%	17,1%	2183	0	0
x4	11649	0,00%	89,0%	11,0%	3679	0	0
PPL2	6664	0,02%	90,5%	09,5%	1891	0	0
PPL4	10392	0,01%	83,0%	17,0%	2924	0	0

Implementations of the SHA-3 core cipher, like the AES ones, produce very stable and predictable results in all architectural alternatives. The *f*[400] variant gives actually the smallest implementations of all the ciphers and the *f*[1600] is bigger but less than by a factor of 4 as the raw increase in state size would suggest. This stands in bright contrast to our previous evaluations of this algorithm in Spartan-3 and Spartan-6 devices ([10]) where implementations of the full-sized KECCAK were either difficult to complete due to routing congestion or led to oversized designs. New potentials of the Spartan-7 array offer substantial improvements here especially when compared to the Spartan-6 capabilities.

4.3 Performance parameters

Table 2 summarizes most important performance characteristics of the 20 implementations. Speed aspect is represented in the first column by the value of the minimum clock period T_{clk} as it was estimated after static timing analysis of the final, fully routed design. The following three columns provide parameters of the longest combinational path in the design (which determined the T_{clk}): number of logic levels in total and dedicated carry logic elements among them, and the percentage of the propagation delay incurred by the logic vs. routing resources.
The minimum clock period is also presented graphically in *Figure 6* and one look at the graph reveals an interesting observations: the fastest designs are the *both* KECCAK size variants, meaning that even the *f*[1600] unit was faster (or on par in case of the unrolled architectures) than the AES implementations. While the superiority of the *f*[400] cipher was seen already in the size analysis, keeping it also by the larger variant is worth noting. It means that the 4x size increase of the *f*[1600] version over the *f*[400] one affected very little the length of the propagation tracks; instead, the increase was absorbed mainly in the width of the data paths. Such regular, almost ideal scaling of logic with an enlarged size is rarely seen in FPGA implementations yet is virtually astounding if presents itself in realizations of a very dense and involved cipher algorithm – the biggest one in our test suite.

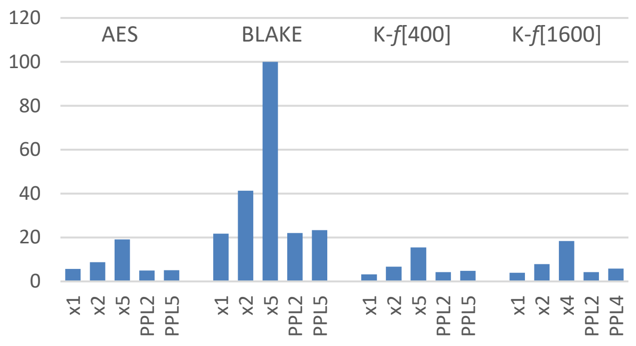

Figure 6. Speed of the tested implementations expressed as the minimum clock period T_{clk} [ns].

The performance results are not so spectacular for the BLAKE cipher. Parameters of the longest propagation paths reveal the problem: this algorithm (as the only one of the tested set) relies on utilization of the dedicated carry logic resources and this impairs efficient use of the LUT generators up to their extended abilities in the new device. Because the BLAKE transformations are a mixture of xors, rotations *and* 32-bit additions – and the adder circuits being implemented with dedicated carry logic must remain outside the LUT tables – data paths in this cipher are implemented as boolean fragments (included in LUTs) running between adders. This explains very high numbers of levels of logic in this cipher: 49 (x1) ÷ 222 (x5) compared to 2 ÷ 13 in the much bigger KECCAK-*f*[1600]. Even if another consequence of this feature is exceptionally low routing part in the delay of the longest path (approx. 50% vs 70 ÷ 80% in other ciphers) this positive effect cannot compensate multiple extra propagation costs induced by the very large number of elements along the path.

Table 2. Implementation results: performance metrics.

	min T_{clk} [ns]	Longest path: levels of logic	carry elements	logic	routing
AES					
x1	5,61	6	0	19,1%	80,9%
x2	8,79	11	0	21,6%	78,4%
x5	19,04	24	0	20,3%	79,7%
PPL2	4,97	5	0	21,4%	78,6%
PPL5	5,13	6	0	22,7%	77,3%
BLAKE					
x1	21,7	50	33	49,4%	50,6%
x2	41,3	92	59	51,0%	49,0%
x5	100,0	222	148	50,2%	49,8%
PPL2	22,0	49	32	50,5%	49,5%
PPL5	23,3	53	34	49,0%	51,0%
KECCAK-*f*[400]					
x1	3,15	3	0	23,7%	76,3%
x2	6,65	6	0	20,1%	79,9%
x5	15,52	13	0	15,0%	85,0%
PPL2	4,20	5	0	21,4%	78,6%
PPL5	4,79	5	0	23,0%	77,0%
KECCAK-*f*[1600]					
x1	3,90	2	0	17,3%	82,7%
x2	7,92	6	0	17,8%	82,2%
x4	18,32	13	0	9,9%	90,1%
PPL2	4,17	2	0	14,5%	85,5%
PPL4	5,80	3	0	14,9%	85,1%

4.4 Efficiency of loop unrolling

Scalability of the loop unrolled architectures (in our case: of x*k* and PPL*k* ones, $k > 1$) is the ability to keep the size and the minimum clock cycle in proportion to the number of rounds instantiated in hardware. As the previous studies have shown e.g. in [12] some hardware platforms may exhibit significant weaknesses in this aspect, mainly due to difficulties in reproduction of involved and irregular internal organization of a round when their long cascade must co-exist in hardware.

This issue will be verified using a simple comparison among each group of cipher implementations: taking the x1 design as a point of reference, its LUT size and T_{clk} period will be used to compute the estimated size and speed of the derived architectures and then the estimations will be compared against actual parameters. The estimations will assume ideal efficiency of round replication in loop unrolling: size in the unrolled (x*k*) and pipelined (PPL*k*) organizations should increase linearly with *k*, while the clock period should remain constant in the PPL*k* cases (the data propagate in each clock cycle still through one round) and should increase in proportion to *k* in the unrolled x*k* ones.

Figure 7 presents the results of such analysis applied to design sizes (number of LUTs), showing the ratios *actual_size* : *estimation* for all the derived architectures in each cipher group. The lower the ratio, the smaller (lower number of LUT) was the actual implementation in comparison to what could be expected from the relevant x1 case. The value of 100% is the threshold separating "better than" (ratio < 100%) from "worse than" (ratio > 100%) the estimation.

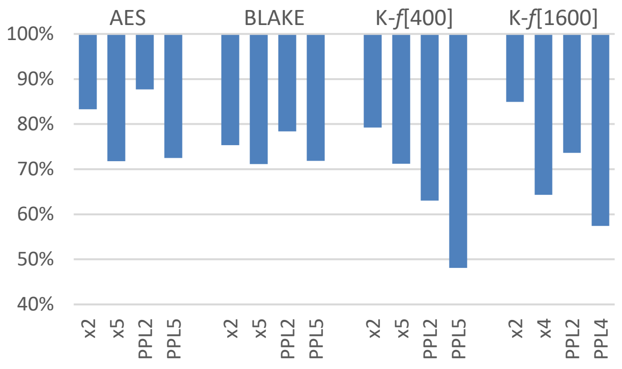

Figure 7. Deviation in % of actual sizes of the derived architectures from the predictions computed on the base of the respective x1 cases.

As the chart proves, all the unrolled architectures were implemented with amount of LUT which was smaller than simple multiplication by *k* would indicate. The extreme 50% reduction is observed in PPL5 organizations of the *f*[400] cipher while in most other cases the reduction remains around 20%. In general this is a situation observed already on the older platforms [11]-[12] but such a consistent confirmation of this trend across all the ciphers is worth noting. Even the largest KECCAK-*f*[1600] behaves here in the same manner as much smaller AES or the *f*[400] variant – without any signs of saturation or overloading of the FPGA array. It does makes a difference as compared to e.g. situation on the Spartan-6 platform where any unrolled implementations of the *f*[1600] cipher could not be even successfully completed due to routing congestion.

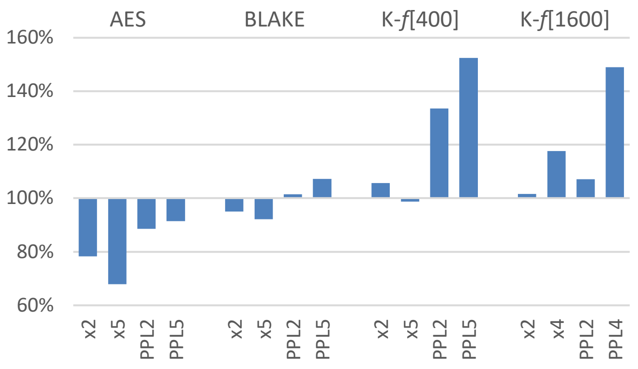

Figure 8. Actual clock periods of the derived architectures as percentages of their estimations from the x1 case.

Effects of the scaling are not so consistent in the clock period which shows *Figure 8*. Only in the smallest and simplest AES algorithm all the unrolled implementations can actually work faster than the x1 case would suggest (actual T_{clk} shorter by 10% to 30%) but in all other ciphers there are implementations which are slower than their predictions. The worst ratios are noted for the KECCAK ciphers but again – this is not a process which intensifies with increasing size of the variant because the 4 times bigger *f*[1600] cipher behavers actually closer to the estimations than its *f*[400] sibling in the x2, PPL2 and PPL5 implementations. Exceeding the estimation by 30 – 50% usually is not a problem and the observed stability of the results (also this time without any signs of saturation or overloading of the array) is more important.

5. Conclusions

The new Spartan family offer logic resources which are comparable in their capacity to the ones of the previous Spartan-6 devices but extensions in global array organization (positively affecting routing capabilities) do offer real improvements in overall efficiency of cipher implementations. These improvements are not much seen in the older AES algorithm which was quite efficiently handled already by the previous generation of Spartan-6 devices. Neither AES size nor complexity of its internal data distribution created any challenge that would call for new capabilities of the S7 architecture.

The BLAKE algorithm still remains difficult in FPGA implementation due to its extensive use of the 32-bit adders which split paths of combinational propagation and limit the use of extended capabilities of the LUT elements. This factor leads to large implemented sizes which also on the S7 platform exceed proportions of the actual dimensions of this cipher based e.g. on size of its state words.

It is the KECCAK compression function which benefits most from the new Spartan-7 potentials in this comparison. Despite the largest internal size, KECCAK-*f*[1600] implementations were consistently smaller than those of BLAKE and not as much larger than the AES ones as the difference in the raw data sizes would suggest. This high size efficiency was accompanied also by good performance characteristics: despite the size, the KECCAK implementations were on average faster not only than the BLAKE ones but also than their (much smaller) AES peers.

References

[1] Aumasson, J.-P., Henzen, L., Meier, W. & Phan, R.C.-W. (2010). SHA-3 proposal BLAKE, version 1.3. https://www.131002.net/blake/blake.pdf (accessed March 2018).

[2] Bertoni, G., Daemen, J., Peeters, M. & Van Assche, G. (2011). *Cryptographic sponge functions*. http://keccak.noekeon.org/ (accessed March 2018).

[3] Bertoni, G., Daemen, J., Peeters, M. & Van Assche, G. (2011). *The Keccak reference*. http://keccak.noekeon.org/ (accessed March 2018).

[4] Gaj, K., Homsirikamol, E., Rogawski, M., Shahid, R. & Sharif, M. U. (2012). Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using Xilinx and Altera FPGAs. *The Third SHA-3 Candidate Conference*, Washington, DC, USA.

[5] Gaj, K., Kaps J.P., Amirineni, V., Rogawski, M., Homsirikamol, E., Brewster, B.Y. (2010). ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs. *20th International Conference on Field Programmable Logic and Applications*, Milano, Italy.

[6] Junkg, B. & Apfelbeck, J. (2011). Area-efficient FPGA implementations of the SHA-3 finalists. *2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, IEEE, 235-241.

[7] Liberatori, M., Otero, F., Bonadero, J.C. & Castineira, J. (2007). AES-128 Cipher. High Speed, Low Cost FPGA Implementation. *Proc. Third Southern Conf. on Programmable Logic*. Mar del Plata, Argentina, IEEE Comp. Soc. Press.

[8] National Institute of Standards and Technology (2001). *Specification for the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publication 197*. http://csrc.nist.gov/publications/PubsFIPS.html (accessed March 2018).

[9] Strömbergson, J. (2008). *Implementation of the Keccak hash function in FPGA devices*. http://www.strombergson.com/files/Keccak_in_FPGAs.pdf (accessed March 2018).

[10] Sugier, J. (2014). Low cost FPGA devices in high speed implementations of KECCAK-*f* hash algorithm. Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) *Proc. 9th Int. Conf. Dependability and Complex Systems DepCoS-RELCOMEX*. Springer AISC, **286**, 433-441.

[11] Sugier, J. (2015). Efficiency of FPGA architectures in implementations of AES, Salsa20 and KECCAK cryptographic algorithms. *J. Polish Safety and Reliability Association*, **6**(2), 117-124.

[12] Sugier, J. (2016). Implementation efficiency of BLAKE and other contemporary hash algorithms in popular FPGA devices. Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) *Dependability Engineering and Complex Systems. Proc. 11th Int. Conf. Dependability and Complex Systems DepCoS-RELCOMEX*. Springer AISC, **470**, 457-467.

[13] Xilinx, Inc. (2016). 7 Series FPGAs Configurable Logic Block. www.xilinx.com, ug474.pdf (accessed March 2018).

[14] Xilinx, Inc. (2018). 7 Series FPGAs Data Sheet: Overview. www.xilinx.com, ds180.pdf (accessed March 2018).