Hardware evaluation of eSTREAM Candidates:

Grain, Lex, Mickey128, Salsa20 and Trivium

Marcin Rogawski
PROKOM Software S.A.
Warsaw, Grójecka 127
rogawskim@prokom.pl
mrogawski@poczta.onet.pl

Abstract

The results of five eSTREAM candidates using ALTERA Field Programmable Gate Arrays are presented and analyzed. Implementation costs and performance of chosen architectures are compared and disscussed. The chosen eSTREAM proposals are divided into four classes regarding the results of their implementation features.

Keywords: Stream Ciphers, FPGA, Hardware evaluation, eSTREAM

1. Introduction

The European Network of Excellence for Cryptography (ECRYPT) has started a multi-year effort called eSTREAM to identify new stream ciphers that might become suitable for widespread adoption. A total of 34 algorithms have been submitted to eSTREAM. All of them were designed primarily for software, hardware or both profiles.

In July 2006 the second phase of evaluation started. Algorithms which have been accepted for the second phase of eSTREAM project have been divided into two classes (algorithms focused on in the second phase and other algorithms). There are ten submissions which are focused on in the second phase: Dragon, HC-256, Lex, Phelix, Py, Salsa20, Sosemanuk, Grain, Mickey-128 and Trivium.

Very few results regarding hardware implementations of the eSTREAM candidates have been published so far. Original documentation provided by designers of the submitted algorithms contains typically only rough estimates of the hardware performance. Additionally, these estimates are very difficult to compare among each other because of large differences in assumptions regarding the technology, and because of different architecture choices. The results of actual implementations of individual algorithms, published recently by independent researchers, provide only a very fragmentary knowledge, not suitable for reliable comparison.

Our main aim is to provide some details about hardware features of Grain, Lex, Mickey128, Salsa20 and Trivium stream ciphers and compare their properties to AES reference implementations.

Stream ciphers

A stream cipher is a symmetric cipher which generates a sequence of cryptographically secure bits called the key stream which is then combined with either the plaintext or ciphertext. The basic topology (Fig. 1) of a stream cipher consists of an internal state (register to store the key and an initialisation vector (IV)), function for internal state update (typically some sort of feedback shift register) and filter function (key stream generation).

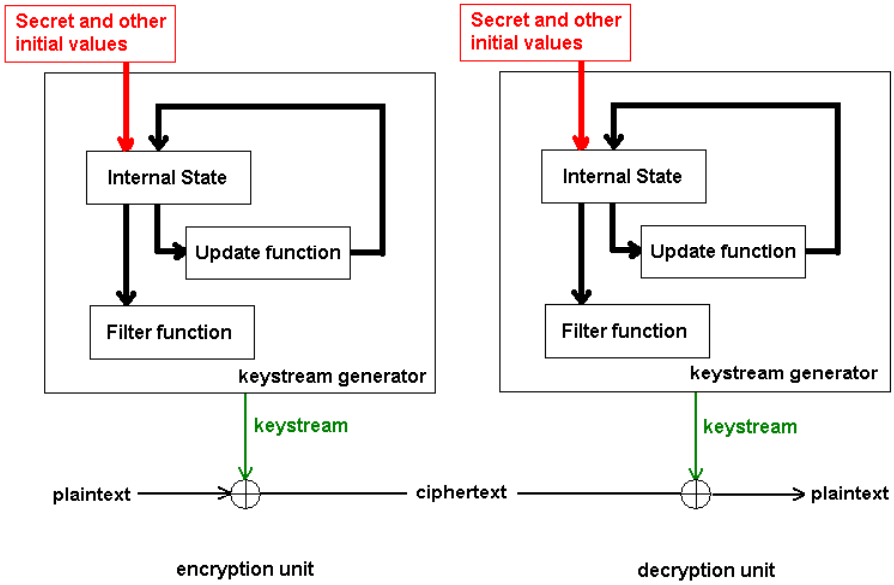

Fig.1: Block scheme of stream cipher

2. Alternative architectures

2.1. Basic organization of a stream cipher implementation

The basic organization of hardware implementation of a stream cipher is shown in Fig.2 All chosen algorithms are implemented using this organization. It includes:

a) keystream generation unit – used to compute key stream. This unit consist of memory for internal state, combinational logic for update and filter functions,

b) input interface – connection to external modules (arguments loading),

c) output interface – connection to external modules (results storing),

d) control unit – responsible for control signal generation. It provides control over other units,

e) encryption/decryption unit – usually xor operation between keystream and plain text and cipher text for encryption or decryption unit respectively.

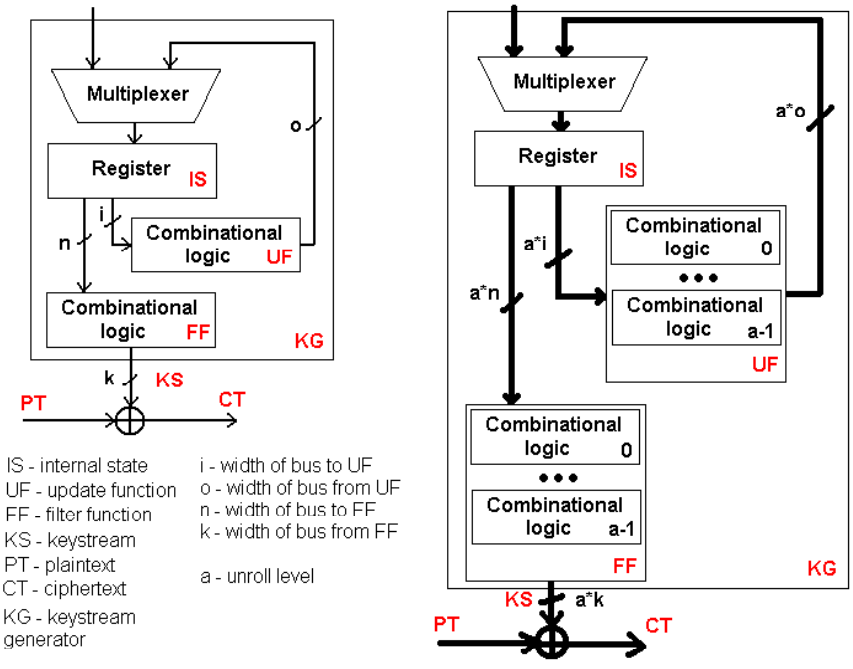

Fig. 2:Two architectures for the implementation of an encryption/decryption unit of a stream cipher: left – basic iterative (basic loop), right – a-round unrolled iterative (a-round loop unrolling)

basic-iterative

Crypto algorithms are usually iterative by nature. For example AES (with 128-bits key) uses 10 rounds for encryption every 128-bits block of plain data. We call an update function and filter function round in stream ciphers. Typically every execution of a stream cipher round allows to extract at least one bit of key stream.

Hardware engineers usually define basic-iterative or basic loop architecture solution with features as follows: one round of a stream cipher is implemented as a combinational logic, and supplemented with a single register and a multiplexer. Input block of initial values is fed to the FPGA through multiplexer and stored in internal memory (registers) in the first clock cycle. In each subsequent clock, one round of cipher is executed and the result is fed back through multiplexer and stored in register. This architectures is shown on the left part in Fig 2.

unrolled-iterative

Unrolled-iterative or loop unrolling architecture is shown in Fig 2 on the right part. Instead of a single rounds in basic iterative architecture, combinational part consist of *a* rounds executed in parallel. In each subsequent clock cycle, *a* rounds of stream cipher (*a* update functions, *a* filter functions) is executed the result (internal state) is fed to the circuit through combinational logic, and stored in register. Internal states (register contents) change in basic-iterative and unrolled-iterative architecture are compared and shown in the Fig. 3.

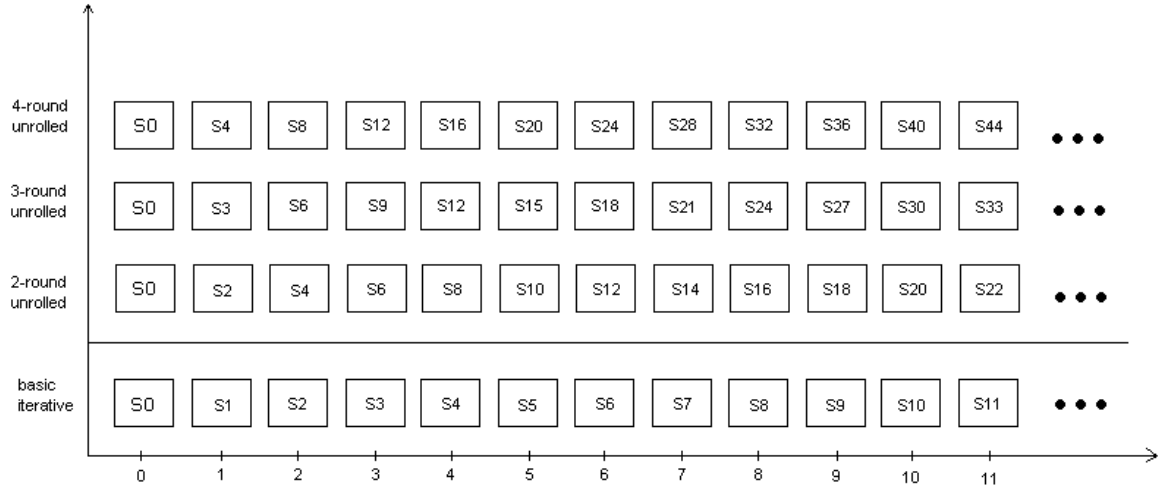

Fig.3: Differences between basic-iterative and unrolled-iterative architectures.

resource-shared

For some ciphers, it is possible to decrease circuit area by time sharing of certain resources (e.g. quarter-round in Salsa20). It is accomplished by using the same functional unit to process two (or more) parts of the date in different clock cycles. Obviously, we agree with the assumption made in [11], that use of the resource sharing in real life implementation is rather untypical, because gain in the circuit area always lead to inappropriate loss in speed of circuit. Moreover the amount of resources used by basic implementation of stream ciphers is usually small.

2.2. Asumptions

The following assumptions have been made to provide an equal basis for comparison:

a) only standard logic cells used (no built-in memories, dsp)
b) standardized interface
c) some submissions did not provide an assiociated authentication method. All algorithms were implemented without any authentication method add-on.
d) code written in VHDL (additionaly in AHDL) language
e) Quartus II 6.0 and Cyclone family (EP1C20F324C6)
f) Cyclone_power_est_2-12 for power consumption's estimation

3. Algorithms

In this section we would like to make a few comments about our impressions apeared during algoritms implementation process, few thoughts and assumptions during collecting and analysis of the results. We would also like to explain also the names of chosen architectures.

Reference implementation – AES – 128bit

The Advanced Encryption Standard [12] was standardized by National Institute of Standards and Technologies in 2001. For low-cost FPGA benchmarks two references were selected in [7]. Both Chodowiec/Gaj and Good/Benaissa implementations uses a Xilinx devices and our implementations were prepared in Altera circuits. We assume that the fair comparison with referenced implementations in area will be done, if we multiply by two their area results (Xilinx CLB Slice is equal to two Altera LE).

Our referenced AES implementation, but according to [11] it is not area-optimized and can be called basic-iterative AES architecture.

Our benchmarks are summarized in Table

	Good/Benaissa	Chodowiec/Gaj	ours
Architecture	8-bit	32-bit	128-bit
FPGA	Xilinx Spartan-II XC2S15-6	Xilinx Spartan-II XC2S30-6	Altera Cyclone EP1C20F400C6
Slices	264 CLB Slices	522 CLB Slices	5058 LE
Clock frequency (Mhz)	67	60	105
Throuhput (Mb/s)	2.2	174	611
Altera LE equivalent	~ 500	~ 1000	5058

Tab.1: Reference hardware AES implementations

Regarding future industry expectations in performance of crypto-algorithms and our area benchmarks we believe that Fig. below divide all our implementation in very fair way.

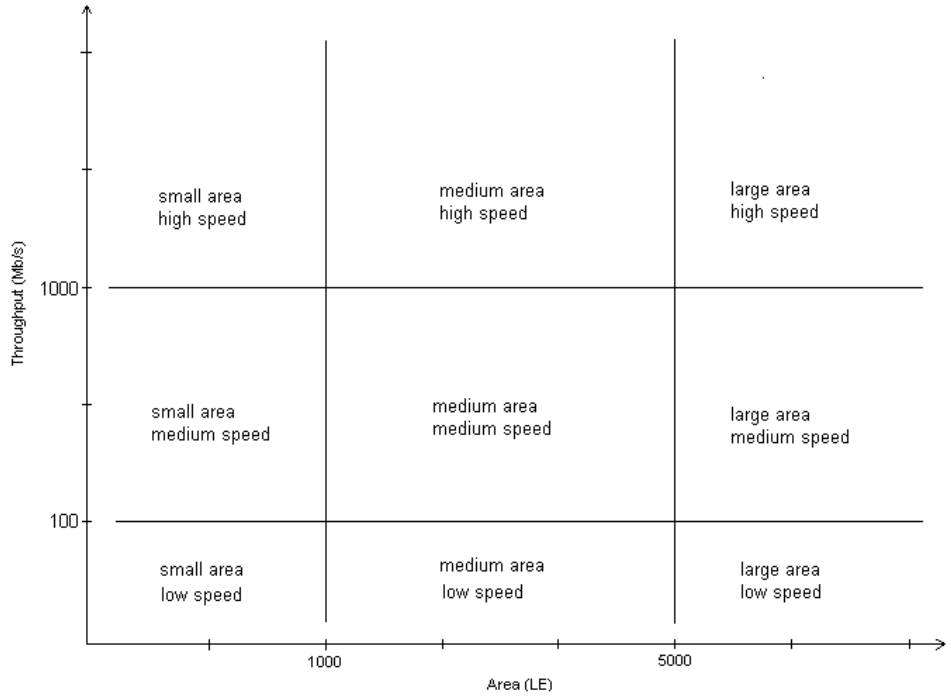

Fig.4: : Area/Throughput implementation division

Grain

Very simple and straightforwared to implement up to 16 basic rounds in unrolled-iterative architecture. Submission package of Grain included documentation and its reference C-code was written in "hardware designer very friendly" way. We were inspired by C-code to prepare to an alternative solution of g(x) function.

Firstly, we implemented table-version of the above mentioned function. Grain-1 (tab) is the name of a basic iterative architecture our hardware Grain implementation. Grain-n (tab), where n from {2, 4, 8, 16}, are the n-unrolled rounds iterative architectures. Grain-n (comb) architectures are similar, except from realization of g(x) function (it is in a gates-network form).

Lex

It is a proposal for a simple AES-based stream cipher which is at least 2.5 times faster than AES both in software and hardware implementation. We agree with author that the strongest points of this design are:

- actual soft./hard. AES implementation can be reused with simple modifications and all favourite AES implementations "tricks" may be used to LEX realization too (speed/ratio is obviously better than AES),
- potential cryptanalysis of existing standard FIPS197 will have influence on crypanalysis of LEX. Concentration of cryptoanalitical effort on AES has a indirect concentration on LEX security.

Our LEX implementation is basic-iterative and is based on our AES reference code.

Mickey-128

Compact algorithm that is very simple to implement. Reference C-code and documentation very easy to follow. We think that the weakest point of this design is difficulty with parallel realization and there is only one our implementation.

Salsa20

Very good documentation with a test vectors for every simple operation in the algorithm. From our point of view an excellent package for engineers consist of: clear documentation, optimized and simple multi-platform C-code, and most important smart test vectors.

Our architectures of Salsa20 we named: Salsa20-dr (unrolled double round iterative architecture), Salsa20-sr (single round iterative architecture – we suggest to call it basic iterative), Salsa20-qr (quarter round resource shared iterative architecture).

Trivium

Trivium has a very simple structure and it is very easy to implement it in optimized version for 4, 8, 16, 32, 64-bits environment without noticeable area penalty. All of our control modules, except one, were prepared in the same "one-hot" routine and 4*288 initial clockings needs 1*18, 2*18, … 64*18 for 64, 32, … 1 bits architectures respectively. Our initial machine states with more initial clockings needs more states and this is a reason, why architectures with 1, 2 and 4 round unrolled has bigger area than 8 round unrolled architecture.

There are critical paths for chosen architectures inside control units. We were looking for the best suited to this algorithm control unit implementation. Key schedule requires 1152 clocks, and control unit can be implemented in many ways:

- based on 11-bits counter (Trivium-1) or 5-bits counter (Trivium-64),
- based on two one-hot machine states: 18 and 64-states (Trivium-1) and 18 states (Trivium-64)
- based on many small one-hot machine states: seven 2-states and two 3-states (Trivium-1 needs 1152 initial clocks – $2^7 3^2$) and one 2-state and two 3-states machine states (Trivium 64 needs 18 initial clocks – $2*3^2$) This architecture we call Trivium-64 enhanced one hot architecture.

  Moreover, Trivium has very good documentation and C-code.

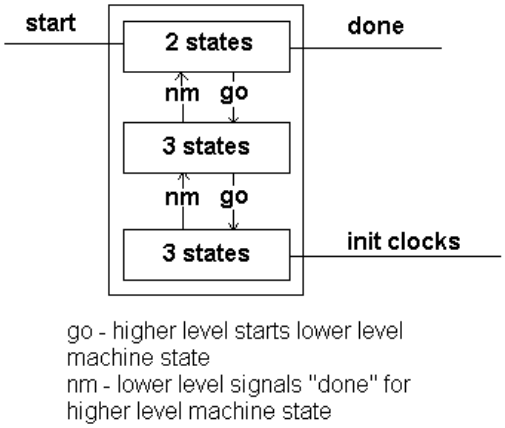

Fig.5: Key Init Control Unit for Trivium-64 enhanced "one hot" architecture

4. Results

	area (LE)	power drain (mW)	throughput (Mb/s)	thr/area (Mb/(s*LE))
AES – 128 bit	5053	1191,01 (105 Mhz)	611	0.12
Grain-1 (comb)	219	341,12 (242 Mhz)	242	1.11
Grain-2 (comb)	254	349,34 (230 Mhz)	460	1.81
Grain-4 (comb)	302	358,12 (221 Mhz)	884	2.93
Grain-8 (comb)	360	369,63 (215 Mhz)	1720	4.78
Grain-16 (comb)	508	406,24 (215 Mhz)	3440	6.77
Grain-1 (tab)	261	347,94 (205 Mhz)	205	0.79
Grain-2 (tab)	326	332,29 (160 Mhz)	320	0.98
Grain-4 (tab)	442	341,94 (160 Mhz)	640	1.45
Grain-8 (tab)	679	356,01 (150 Mhz)	1200	1.77
Grain-16 (tab)	1138	385,52 (143 Mhz)	2288	2.01
LEX	5378	1578,31 (100 Mhz)	1454	0.27
Mickey-128	537	366,63 (220 Mhz)	220	0.41
Salsa20-qr	2356	415,23 (55 Mhz)	343	0.15
Salsa20-sr	3400	390,00 (40 Mhz)	931	0.27
Salsa20-dr	3510	450,14 (30 Mhz)	1280	0.36
Trivium-1	393	381,99 (295 Mhz)	295	0.75
Trivium-2	368	381,26 (290 Mhz)	580	1.58
Trivium-4	364	393,27 (300 Mhz)	1200	3.30
Trivium-8	380	435,00 (350 Mhz)	2800	7.36
Trivium-16	424	457,15 (290 Mhz)	4640	10.94
Trivium-32	518	540,08 (280 Mhz)	8960	17.30
Trivium-64	710	669,78 (245 Mhz)	15680	22.08
Trivium-64-enhanced	700	683,13 (255 Mhz)	16320	23.31

Table: Summary of results for eSTREAM candidates

	Init clocks	Init time	Crytical path
AES – 128 bit	22	210ns	SubByte, ShiftRow, MixColumn, AddKey
Grain-1 (comb)	322	1330ns	Combinational version of g(x)
Grain-16 (comb)	19	92ns	Combinational version of g(x)
Grain-1 (tab)	322	1570ns	Table version of g(x)
Grain-16 (tab)	19	168ns	Teble version of g(x)
LEX	22	220ns	SubByte, ShiftRow, MixColumn, AddKey
Mickey-128	603	2660ns	Update function in S register
Salsa20-qr	180	3240ns	Quarterround
Salsa20-dr	12	444ns	Doubleround
Trivium-1	1578	5350ns	Control Unit
Trivium-64	29	120ns	Control Unit
Trivium-64-enhanced	38	147ns	Round function

Table: Summary of Initialisation results and crytical path

Compactness

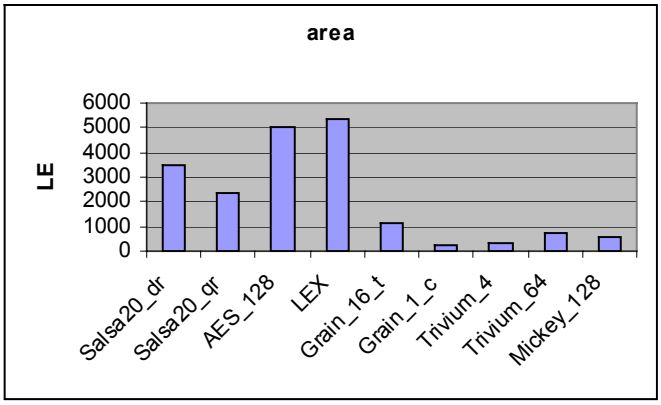

Fig.6: Compactness summary

Selected reference AES implementations divided the set of chosen eSTREAM candidates into 3 classes:
- small area (no more than 2k LE used)
- medium area (2k – 5k LE)
- large area (more than our biggest ref. AES implementation > 5k LE).

Three of chosen algorithms (Trivium, Grain, Mickey) have very compact characteristic and it is possible to implement them with not more than 1000 LE. The biggest (and the fastest) version of Trivium needs almost twice as much as the smallest one, but still can be regarded as one of the very compact implementations. Grain algorithm can be described in the same way. The biggest version needs five times as much as the smallest one, but similarly it is still very compact.

Mickey-128 algorithm is based on two shift registers but his update functions in linear and non-linear register seems to be not very scalable. That is the reason why it is easy to implement compact version of Mickey, but our experiences with looking for high speed version are similar to the results from [10] and [8].

Salsa20 fits very well to our second class. Our implementations of this algorithm requires between 2,4k. LE and 3,5k. LE.

Only LEX seems to be always regarded as a more demanding as an implementation of AES, but it is always possible to use all flexibility and good implementation features of AES [26].

Performance

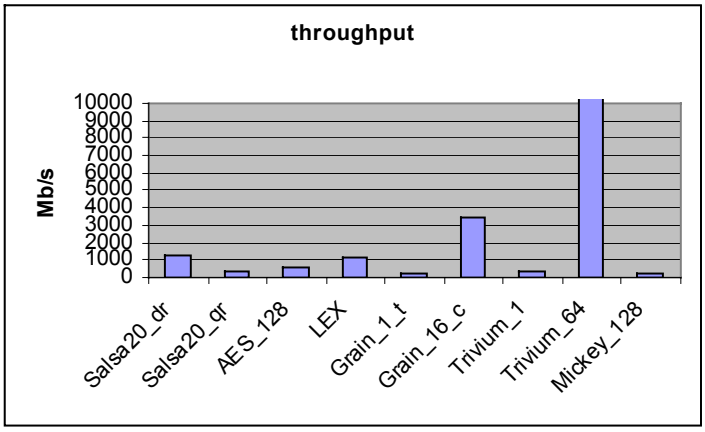

Fig.7: Throughput summary

Throughput is not a most important matter in our disscustion, but speed of Trivium-64 and Grain-16 architecture have to be mentioned. Our results confirm authors expectations that this algorithms suits hardware implementations very well.

Salsa20, Lex also can be implemented in such a way to achieve throughput better that 1Gb/s.
As mention above Mickey seems to be very difficult scalable and our implementation is regarded as a medium speed.

Power Consumption

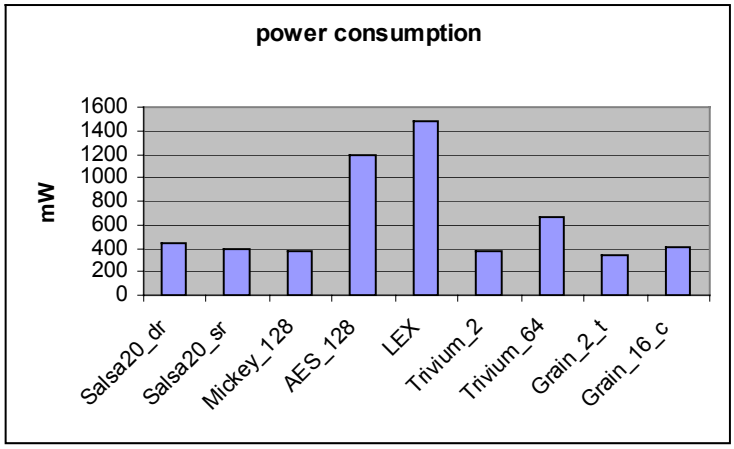

Fig.8: Power consumption summary

We used Power Play Analyzer from Quartus II and Power Calculator for Cyclone family from altera.com site. All chosen algorithms, except LEX has more than twice smaller power consumption than AES.

Throughput/Area

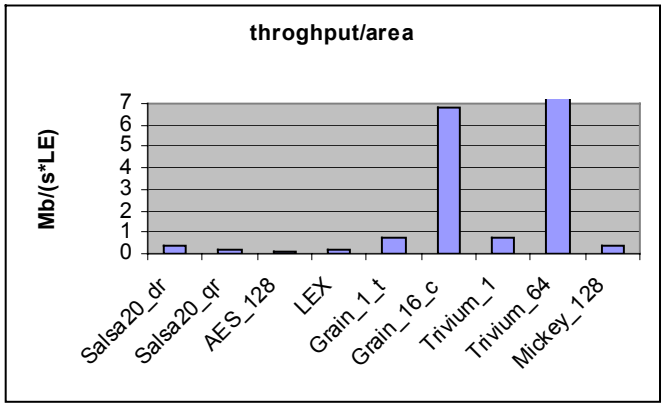

Fig.9: Throughput/area ratio summary

All our implementation have better throughput/area ratio than our AES ref. implementation. Grain and Trivium have significantly better ratio.

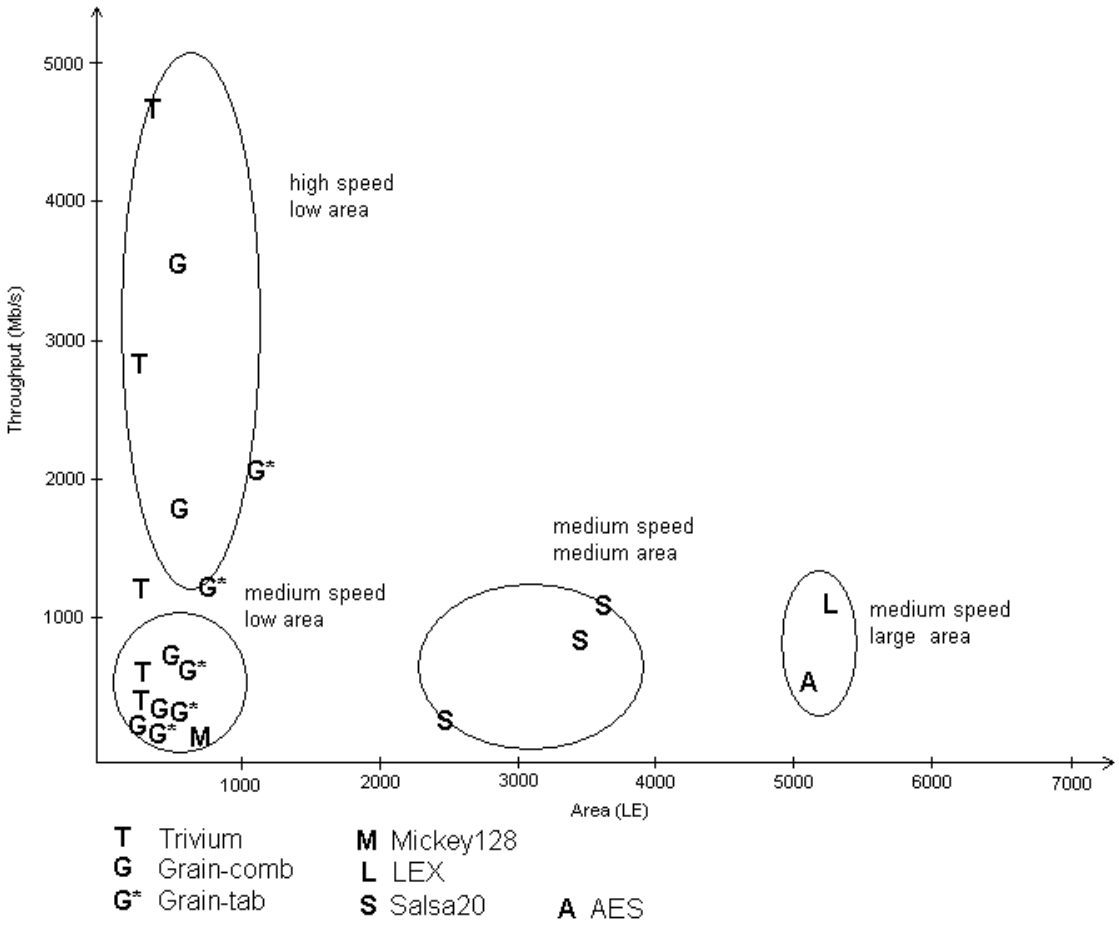

Fig.10: Hardware characteristics of chosen algorithms

5. Conclusions

We believe that the large differences among parameters of all chosen eSTREAM algorithms in hardware resulted primarily from internal structure of these algorithms, and were not significantly affected by our implementation decisions. On the other hand, we could not completely eliminate or predict the influence of the FPGA design tools and the HDL design entry method on the results of the comparison. Assessed exclusively from the hardware performance point of view, the five chosen algorithms fall into the four distinct classes with different performance characteristics (Fig.10).

First class (large area, medium/high speed): LEX, AES (ours) (Lex is better than 1Gb/s but it needs more resources than AES)

Second class(medium area medium/high speed): Salsa20 (Salsa20 is insignificantly better than 1Gb/s and it is very close to be threated as “small area” algorithm type)

Third class (small area, medium speed): Mickey

Fourth class (small area, high speed): Trivium and Grain (great scalability, wide range possibilities of implementation)

The expectations from an efficient cryptographic algorithm will differ depending on the specific application. It is very difficult to expect that a single implementation will satisfy all requirements. Our opinion is that the most important aspect for a hardware/software-efficient cryptographic algorithm is flexibility. It must be possible to implement wide range of algorithm's architectures.

6. Acknowledgements

We would like to thank Kris Gaj for valuable comments, advise and encuregement in implementations of eSTREAM proposals. We also would like to thank Dariusz Świderski and Piotr Żarkiewicz for their valuable comments about technology and importance of low cost applications.

7. References

[1]. De Canniere Ch., Preneel B. – "Trivium Specifications" – eSTREAM proposal at http://www.ecrypt.eu.org/stream/triviump2.html
[2]. Hell M., Johansson T., Meier W. – "Grain – a stream cipher for constrained environments" – eSTREAM proposal at http://www.ecrypt.eu.org/stream/grainp2.html
[3]. Babbage S., Dodd M. – „The stream cipher Mickey-128"- eSTREAM proposal at http://www.ecrypt.eu.org/stream/mickey128p2.html
[4]. Whiting D., Schneier B., Lucks S., Muller F. – „Phelix Fast Encryption and Autentication in a single Cryptographic Primitve" – eSTREAM proposal at http://www.ecrypt.eu.org/stream/phelixp2.html
[5]. Bernstein D. – "Slasa20 specification" – eSTREAM proposal at http://www.ecrypt.eu.org/stream/salsa20p2.html
[6]. Biryukov A. – "A new 128-bit key stream cipher LEX" – eSTREAM proposal at http://www.ecrypt.eu.org/stream/lexp2.html
[7]. Batina L., Kumar S., Lano J., Lemke K., Mentens N., Paar C., Preneel B., Sakiyama K., Verbauwhede I. – „Testing framework for eSTREAM Profile II Candidates", SASC 2006 - Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006 available at http://www.ecrypt.eu.org/stream/papersdir/2006/014.pdf
[8]. Gurkayank F., Luethi P., Bernold N., Blattmann R., Goode V., Marghitola M., Kaeslin H., Fleber N., Fichtner W. – "Hardware evaluation of eSTREAM Candidates: Achterbahn, Grain, Mickey, Mosquito, Sfinks, Trivium, Vest, ZK-Crypt", SASC 2006 - Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006 available at http://www.ecrypt.eu.org/stream/papersdir/2006/059.pdf
[9]. Good T., Chelton W., Benaissa M. – "Review of stream cipher candidates from a low resource hardware perspective", SASC 2006 - Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006 available at http://www.ecrypt.eu.org/stream/papersdir/2006/059.pdf
[10]. Kitsos P. – "On the hardware implementation of the Mickey-128 Stream Cipher" http://www.ecrypt.eu.org/stream/papersdir/2006/059.pdf
[11]. Gaj K., Chodowiec P. – „Comparison of the hardware performance of the AES candidates using reconfigurable hardware", 3. AES Candidate Conference 2000: New York, New York, USA, available at http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/22-kgaj.pdf
[12]. Fips197, , available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[13]. Misztal M. – lectures in subject "Cryptanalysis of Stream Ciphers"
[14]. Mank K. – lectures in subject "Stream Ciphers"
[15]. Mitchell C., Dent A. – "International standards for stream ciphers: A progress report", SASC2004, 14-15.10.2004, available at http://www.isg.rhul.ac.uk/cjm/isfsca.pdf
[16]. Lano J., Mentens N., Preneel B., Verbauwhede I. – „Power Analysis of Stream Cipher",SASC2004, 14-15.10.2004,
[17] Rechberger C., Oswald E. – „Stream Cipher and Side Channel Analysis" http://www.iaik.tugraz.at/research/sca-lab/publications/pdf/Rechberger2004SCAandStream.pdf
[18]. Kumar S., Lemke K., Paar C. – „Some thoughts about Implementation properties of stream ciphers", SASC2004, 14-15.10.2004, http://www.ecrypt.eu.org/stvl/sasc/slides25.pdf
[19]. Babbage S. – "Stream cipher – what does the industry want ?", SASC2004, 14-15.10.2004, http://www.ecrypt.eu.org/stvl/sasc/slides21.pdf
[20]. Biryukov A. – "Block and Stream ciphers – state of art" http://eprint.iacr.org/2004/094.pdf
[21]. Bernstein D.J. – „Comparision of 256 bit stream ciphers" - SASC 2006, Leuven, Belgium - 2006.02.02 http://cr.yp.to/talks/2006.02.02/slides.pdf
http://cr.yp.to/streamciphers/stream256-20060123.pdf
[22]. Galanis M., Kitsos P., Kostopoulos G., Sklavos N., Goutis C. – „Comparision of Hardware Implementation of Stream ciphers", The International Arab Journal of Information Technology, Vol.2, No. 4, October 2005, p.275-274, http://www.vlsi.ee.upatras.gr/~mgalanis/pubs/iajit.pdf
[23]. http://www.altera.com
[24]. Bora P. – "Methods of hardware implementations of crypto-algoritms" – lecture in Polish Academy of Science
[25]. Gaj K., Chodowiec P. – „Very Compact FPGA Implementation of AES algorithm", CHES 2003, Proceedings , LNCS Vol. 2779 , pp. 319-333 http://islab.oregonstate.edu/ches/SLIDES/chodowiec_gaj.pdf
[26]. Good T., Benaissa M. – "AES FPGA from the Fastest to the smallest" – Proceedings of CHES 2005, pp. 427-440, LNCS 3659, Springer, 2005 http://class.ece.iastate.edu/tyagi/cpre681/papers/AESCHES05.pdf
[27]. www.altera.com
[28] Menezes J. A., van Oorschot C., Vanstone A. S. – "Handbook of apllied of cryptography" available at http://www.cacr.math.uwaterloo.ca/hac/