FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile

Philippe Bulens[⋆], Kassem Kalach[⋆⋆],
François-Xavier Standaert[⋆ ⋆ ⋆] and Jean-Jacques Quisquater

UCL Crypto Group, Université Catholique de Louvain,
Laboratoire de Microélectronique (DICE),
Place du Levant 3, B-1348 Louvain-La-Neuve, Belgium.
{bulens, kalach, fstandae, quisquater}@dice.ucl.ac.be

Abstract. Efficient cryptographic implementations are a fundamental factor in the achievement and dissemination of new computerized applications. In some recent environments with (very) limited resources such as smart cards, sensor networks or RFID tags, standard algorithms may not be completely adapted.Consequently, the design of new solutions for low-cost cryptography is sometimes necessary and is at least an interesting research direction. In this context, stream ciphers are usually believed to be an efficient alternative to block ciphers, because of a lower hardware (or even software) implementation cost. The eSTREAM research initiative was consequently established in order to investigate the possibility of building secure and efficient stream ciphers. Among a large number of submitted candidates, the project recently (September 2006) narrowed the "hardware profiled" stream ciphers to four focused candidates, namely Trivium, Grain-128, MICKEY-128 2.0 and Phelix. In this paper, we evaluate the hardware performance of these algorithms in the reconfigurable hardware Xilinx Virtex-II devices. Based on our implementations (that mainly confirm previous results), we discuss the respective interest of the focused candidates and suggest certain guidelines for their comparison.

1 Introduction

Stream ciphers are an important class of symmetric encryption algorithms. They contain internal states that vary with time and generate pseudo-random bit sequences, usually denoted as the keystream. The keystream is then bitwise XORed with a message in order to emulate the one-time-pad. By contrast, block ciphers tend to simultaneously encrypt/decrypt blocks of bits and act as a pseudo-random permutation of a fixed size, parametrized by a secret key. Although they received less attention than standard block ciphers in the recent years, stream

[⋆] Supported by Walloon Region, Belgium / First Europe Program.
[⋆⋆] Supported by Walloon Region, Belgium / IRMA project.
[⋆ ⋆ ⋆] François-Xavier Standaert is a post doctoral researcher funded by the FNRS (National Funds for Scientific Research, Belgium).

ciphers are more appropriate in certain contexts, *e.g.* where buffering is limited or when characters must be individually processed. Another interesting property is that the keystream can be generated prior to encryption and decryption when it is independent of the plaintext/ciphertext. In addition, it is also generally assumed that stream ciphers have a lower implementation cost and therefore are a sound solution for very constrained environments. Due to the proliferation of applications requiring low-cost cryptographic algorithms, *e.g.* smart cards, sensor networks or RFID tags, they appear as an interesting research direction[1].

In this context, the eSTREAM initiative [20] was launched as an effort to identify new stream ciphers that might be considered for widespread adoption. Since most known (old) stream ciphers have been the target of (more or less realistic) cryptanalytic attacks, a most important objective of the project was the design of a secure cipher and the derivation of sound (easily checkable) security criteria. As a reply to the eSTREAM call-for-papers, numerous algorithms have been submitted. They have been divided into three categories, namely software profiled, hardware profiled or suitable for both. Although the security evaluation of the different candidates is still running, a selection of focused candidates was posted on the ECRYPT page in September 2006. As for the Advanced Encryption Standard process, an important criteria for the discrimination of the remaining candidates is the performance evaluation.

In this paper, we consequently investigated the performances of the four ciphers selected as *Phase 2 Focus Candidate for Hardware Profile*, namely Grain-128 [15], Trivium [5], Mickey-128 2.0 [3] and Phelix [21]. The different ciphers will be shortly described in Section 2. Section 3 details our design methodology, our target platform for the evaluation as well as our architectures for the different ciphers. The implementation results are summarized in Section 4, together with a comparison/discussion between different stream and block ciphers from a hardware performance point of view and some guidelines for the discrimination of the four eSTREAM candidates. Finally, our conclusions are in Section 5.

2 Ciphers Description

In general, stream ciphers consist of two phases: an initialization phase in which the initial state of the cipher is derived from the key and initial vector (IV) and a generation phase in which the state is repeatedly updated and used to generate the keystream (*e.g.* via some non-linear function). In the next section a brief functional description of the different eSTREAM ciphers we consider in this paper will be given. For a more insight view of those ciphers, please refer to the original papers [2–5, 14, 15, 21].

[1] Note that the division stream cipher *vs.* block cipher may appear somewhat arbitrary since a block cipher can be turned into a stream cipher if used in Output Feedback Mode or Cipher Feedback Mode.

2.1 Trivium

Trivium is a synchronous stream cipher designed to provide a flexible trade-off between speed and gate count. It has one of the simplest architecture amongst the eSTREAM candidates and is consequently particularly easy to implement. Trivium supports an 80-bit secret key and an 80-bit IV. It generates up to 2^{64} bits of keystream. Trivium's 288-bit internal state may be seen as a concatenation of three shift registers of different lengths. The state is updated using an iterative process which extracts the values of 15 specific state bits and uses them to both update 3 bits of the state and compute 1 bit of keystream. The state is then updated and the process is repeated until the requested $N \leq 2^{64}$ bits of keystream have been generated. To initialize the cipher, the key and IV are written into two of the shift registers and the remaining bits are set to a fixed pattern of zeros and ones. The state is then updated over 4 full cycles (meaning 4×288 updates), in the same way as explained above, but without generating keystream bits. This initial step guarantees that every bit of the internal state depends on every bit of the key and of the IV in a complex nonlinear way.

2.2 Grain-128

Grain is a synchronous stream cipher targeted for hardware environments where gate count, power consumption and memory are limited. It supports a key size of 128 bits and IV size of 96 bits. Grain's 256-bit internal state consists of a 128-bit linear feedback shift register (LFSR) and a 128-bit non-linear feedback shift register (NFSR). The state is updated thanks to a nonlinear filter function that takes two inputs from the NFSR and seven from the LFSR. The NFSR is updated with a nonlinear 19-to-1 boolean function and a 1-bit linear input selected from the LFSR. The LFSR is updated with a 6-to-1 linear function. Before keystream is generated, the cipher must be initialized as follows: the 128-bit key is loaded into the 128-bit NFSR, the 96-bit IV is loaded into the low 96-bits of the LFSR and the remaining 32 high bits of the LFSR are filled with ones. The cipher is then clocked 256 times without producing any keystream. Instead, the output function is fed back and XORed with the input, both to the NFSR and LFSR update functions.

2.3 MICKEY-128 2.0

Just like Grain-128 and Trivium, MICKEY-128 2.0 is based on shift registers. The content of the two 160-bit registers defines the state of the cipher. It uses a fixed length 128-bit key and a variable length initialization vector (ranging from 0 to 128 bits). At startup, the two shift registers are set to "0". Then, the IV is used to fill the registers, followed by the key. The cipher is then clocked 160 times before producing any keystream bit. According to the authors, up to 2^{64} bits may be generated using a single (IV,key) pair. The main difference with Grain-128 and Trivium is that each single bit of the state of MICKEY-128 2.0 is computed, not just shifted (see Sec. 3.4). This allows replacing the use of a filter function by a simple XOR of two state bits to compute the keystream.

2.4 Phelix

Phelix is an enhanced version of Helix. Although it is presented as a stream cipher, it looks very much like a block cipher. It operates on a set of nine 32-bit words at a time, five of which being "active", that cycle through the Phelix structure. The basic operations involved are XORs, additions $\pmod{2^{32}}$ and rotations, allowing a fast execution on any kind of platform. Phelix works differently than the previously mentioned ciphers. For example, the first step is to derive the working key from the initial key (256-bit) and expand the nonce (128-bit) to its full size (256-bit). Those expanded words are then used to initialize and pre-clock the cipher before the keystream generation begins. Another difference is that the plaintext is incorporated during the computation. This gives the cipher the additional feature of a message authentication code (MAC) that can be derived at the cost of a fixed number of additional loops in the structure.

3 Hardware Implementation

3.1 Methodology

Since the selected designs are purposed for low-cost devices, we adopted a bit-serial style of interface for the inputs and outputs (plaintext, key, IV, ciphertext). Similarly, throughput was generally sacrificed in the favor of area requirements. For example, we did not take advantage of the possible parallelization tricks of Trivium and Grain-128 that would increase the throughput over area ratio. Consequently, our implementations are not necessarily the most efficient ones and we mainly aimed to evaluate the possibility to obtain low-cost architectures.

3.2 Targeted Platform

The chosen platform for implementation is a Xilinx Virtex-II FPGA. Such devices embed programmable logic blocks, RAMs and multipliers. In this description, the focus will be set on the logic elements as other resources will not be used. The slice is the logic unit that is used to evaluate a design's area requirements. Such a slice, depicted in Fig. 1 is made up of 2 Look-Up Tables (LUTs), 2 flip flops and a few additional gates. According to the user's choice, any of these LUTs can be configured in one out of three possible ways: RAM16 that act like a 16-bit RAM storage, SRL16 that implements a 16-bit linear shift register and LUT that is capable of computing any 4-to-1 boolean function. A more detailed view of half a slice is given in Fig. 2. An interesting thing to notice is the fast carry chain that cross the slice. As its designation suggests, it allows the efficient implementation of carry-propagate adders and, as will be shown, can sometimes be used to simplify the combinatorial cost of some designs.

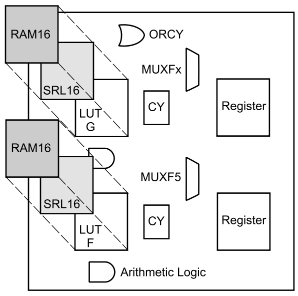

Fig. 1. Available Slice Configurations.

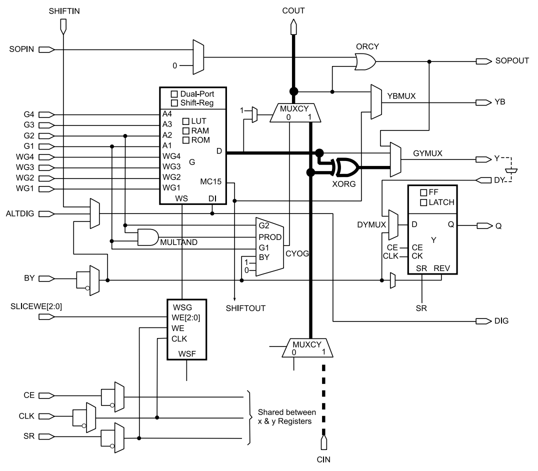

Fig. 2. Top Half Slice of a Xilinx Virtex-II.

3.3 Trivium and Grain-128

A straightforward approach was adopted to implement Grain and Trivium. We only used the platform-specific primitives SRL16 for the shift registers that provide an advantageous area optimization over a platform-independent design. It is worth noticing that the efficient exploitation of the SRL16 primitives highly depends on the required tap positions in the stream ciphers. Any time an intermediate value is extracted from the SRL16 primitives, a new LUT has to be used. As a consequence, a k bit register with $k < 16$ may occupy a complete SRL16 cell. Note also that this approach has the disadvantage of being less portable. The rest of our code is directly derived from the algorithm's specifications. Due to the simplicity of the operative parts in our designs, the cost of the control parts (*i.e.* state machines and counters) is not negligible.

3.4 MICKEY-128 2.0

The description of MICKEY-128 2.0 is somewhat software oriented. We consequently re-wrote the update functions of the R and S registers as follows:

- CLOCK_R :
  - if $i == 0 : r_0 \Leftarrow (r_0 \cdot \mathrm{CR}) \oplus r_{159} \oplus \mathrm{IR}$
  - if $i \in [1..159]$:

$$r_i \Leftarrow \begin{cases} r_{i-1} \oplus (r_i \cdot \mathrm{CR}) \oplus r_{159} \oplus \mathrm{IR} & \text{if } i \in \mathrm{RTAPS} \\ r_{i-1} \oplus (r_i \cdot \mathrm{CR}) & \text{if } i \notin \mathrm{RTAPS} \end{cases}$$

- CLOCK_S :
  - if $i == 0 : s_0 \Leftarrow s_{159} \oplus \mathrm{IS}$
  - if $i \in [1..158]$:

$$s_i \Leftarrow \begin{cases} s_{i-1} \oplus (\overline{s_i}' \cdot \overline{s_{i+1}}') & \text{if } \mathrm{FB0}_i \mid\mid \mathrm{FB1}_i == 00 \\ s_{i-1} \oplus (\overline{s_i}' \cdot \overline{s_{i+1}}') \oplus ((s_{159} \oplus \mathrm{IS}) \cdot \mathrm{CS}) & \text{if } \mathrm{FB0}_i \mid\mid \mathrm{FB1}_i == 01 \\ s_{i-1} \oplus (\overline{s_i}' \cdot \overline{s_{i+1}}') \oplus \big((s_{159} \oplus \mathrm{IS}) \cdot \overline{\mathrm{CS}}\big) & \text{if } \mathrm{FB0}_i \mid\mid \mathrm{FB1}_i == 10 \\ s_{i-1} \oplus (\overline{s_i}' \cdot \overline{s_{i+1}}') \oplus s_{159} \oplus \mathrm{IS} & \text{if } \mathrm{FB0}_i \mid\mid \mathrm{FB1}_i == 11 \end{cases}$$

    where $(\overline{s_i}' \cdot \overline{s_{i-1}}')$ is a shorthand for $((s_i \oplus \mathrm{COMP0}_i) \cdot (s_{i+1} \oplus \mathrm{COMP1}_i))$. The tabulated values of $\mathrm{COMP0}_i$ and $\mathrm{COMP1}_i$ allow to determine whether the $\overline{}$ operator has to apply[2].
  - if $i == 159 : s_{159} \Leftarrow s_{158} \oplus \big((s_{159} \oplus \mathrm{IS}) \cdot \overline{\mathrm{CS}}\big)$

As stated in Sec. 3.2, each LUT is capable of implementing any 4-to-1 boolean function. Fortunately, the update functions of the R and S registers of MICKEY-128 2.0 mainly use XOR operations, *i.e.* one of the additional gate available for each LUT. By setting either r_{159} or s_{159} on the carry-in pad of the slice, elements r_i and s_i requiring 5 inputs to be computed will fit in half a slice (see emphasized path in Fig. 2). For the remaining cases of s_i parameterized by $\mathrm{FB0}_i \neq \mathrm{FB1}_i$[3], the same trick can be used in order to fill a single slice. In extenso, the first LUT computes $\Big((s_{159} \oplus \mathrm{IS}) \cdot \overline{\mathrm{CS}}'\Big)$; the second, $s_{i-1} \oplus (\overline{s_i}' \cdot \overline{s_{i+1}}')$ and an additional XOR is used to complete the evaluation. Each of these bits consumes 2 LUTs.

This implementation can slightly be improved: if a group of 3 s_i with same $\mathrm{FB0}_i \mid\mid \mathrm{FB1}_i$ is stacked within 2 slices, the computation of $\Big((s_{159} \oplus \mathrm{IS}) \cdot \overline{\mathrm{CS}}'\Big)$ – taking place in one LUT – can be broadcasted to the upper 3 LUTs through the fast carry-chain. The increase in area for this part is now limited to a third of the 80 LUTs required to store this part of S. It is tempting to extend this idea to a group of 7 s_i stacked within 4 slices. However, this is likely to decrease the performance by increasing the complexity of the routing as the local availability of the data is reduced. Note that, as opposed to Trivium and Grain-128, each single bit of the state has to be computed which prevents benefiting from the SRL16 configured slices of the device. As a result, MICKEY-128 2.0 is expected to consume more area in our target FPGAs.

3.5 Phelix

The implementation of Phelix is quite straightforward. The designer is just left with the choice of the tradeoff best suiting his needs: a "big" design with a high throughput by implementing the whole Phelix block structure or something smaller, down to the "tiny" single round. From the previous implemented designs in [11], the best trade-off seems to be the one based on the H function (half the block structure) which exhibits the highest throughput over area. Therefore, we

[2] For example: let $i == 2$, then $s_i \Leftarrow s_{i-1} \oplus (\overline{s_i} \cdot s_{i+1}) \oplus \big((s_{159} \oplus \mathrm{IS}) \cdot \overline{\mathrm{CS}}\big)$.

[3] About half the length of the register S; that is around 80 bits.

selected this configuration. In order to implement the buffer storing Z_4^{i-1} to Z_4^{i-4}, the use of SRL16-configured LUTs again allows reducing the required space from 64 slices (128 registers) to 16 slices. In addition, the 512 memory points required to remember the expanded key and nonce words can be efficiently stored in LUTs configured as RAM (RAM16). This consumes 32 LUTs, that is 16 slices. Despite these efforts, the implementation seems to stay a little more expensive in area than presented in [11], which may be caused by a different interface.

4 Implementation Results

The four selected ciphers were implemented in VHDL. Synthesis and Place & Route were achieved on Xilinx ISE 8.2i. The selected device is a Xilinx Virtex-II XC2V6000-4ff1152. Table 1 summarizes our results whereas table 2 reminds some results of previous selected works.

Table 1. Implementation Results.

Cipher	**Device**	**Slices**	**Frequency (MHz)**	**Throughput (Mbps)**	**Thr./Area (Mbps/slice)**
Trivium	Virtex-II	41	207	207	5.05
Grain-128	Virtex-II	48	181	181	3.77
MICKEY-128 2.0	Virtex-II	190	200	200	1.05
Phelix	Virtex-II	1213	62.5	1000	0.82

Compared to the previous implementations of the same ciphers, the situation has not changed much. The specifications of Trivium and Phelix are the same as for eSTREAM *Phase 1* and consequently, they exhibit quite similar figures compared to the results in Table 2. In the case of Grain-128, the revision that, amongst others, has grown the state from 2×80-bit to 2×128-bit goes unnoticed in terms of FPGA hardware cost. Regarding MICKEY-128 2.0, the updated version has slightly increased its cost but keeps it reasonable. It should be noticed that the gap between MICKEY-128 2.0 on the one hand and Trivium and Grain-128 on the other hand could be reduced if ASIC implementations were considered. Indeed, MICKEY-128 2.0 does not benefit from SRL16-configured slices that Trivium and Grain-128 use abundantly.

Compared to the implementations of other stream ciphers, a number of additional observations can be mentioned. For illustration purposes, we also reported some implementation results for the AES Rijndael block cipher. Most interestingly, these performance evaluations highlight that Trivium and Grain-128 have extremely reduced implementation complexities that could be compared, *e.g.* to those of A5/1. MICKEY-128 2.0 has a slightly higher hardware cost in our target Xilinx FPGA. As already mentioned, this trend would probably be tempered if ASIC designs were considered. Finally, it seems that the performances of Phelix are better compared to those of a block cipher, as its structure also

Table 2. Previous Block and Stream Ciphers Implementations.

Cipher	**Device**	**Slices**	**Frequency (MHz)**	**Throughput (Mbps)**	**Thr./Area (Mbps/slice)**
Stream Ciphers					
A5/1 [9]	Virtex-II	32	–	188.3	5.88
Trivium [11]	Spartan-II	40	–	102	2.55
Grain [11]	Spartan-II	48	–	105	2.19
RC4 [9]*	Virtex-II	140	–	120.8	0.86*
MICKEY-128 [17]	Virtex-E	167	170	170	1.02
Phelix [11]	Spartan-II	1077	–	750	0.7
Block Ciphers					
AES [10]*	Spartan-II	124	–	2.2	0.02*
AES [19]*	Virtex-II	146	123	358	2.45*
AES [22]	Virtex-II	1780	77.91	1000	0.56

suggests. For example, the AES architecture in [22] iterates over a 128-bit width loop structure while the one in [19] uses a 32-bit datapath. Phelix ranges somewhere close to these results. Note that these observations have to be considered as general intuitions rather than fair comparisons since they consider different FPGA technologies: more recent FPGAs have higher work frequencies and thus throughput. In addition, the hardware efficiency (*i.e.* throughput/area ratio) of the *-marked implementations is not meaningful since they consumes FPGA RAM blocks. Finally, the hardware cost can only be compared if the respective implementation efficiencies (*e.g.* measured with the throughput/area ratio) are somewhat comparable. As a matter of fact, it is always possible to reduce the implementation cost, by considering smaller datapaths (*e.g.* [10] uses an 8-bit datapath for the AES) at the cost of a reduced throughput.

To strengthen these observations, let us finally mention that Trivium and Grain-128 offer the best opportunities to improve their throughput by parallelization techniques. In the case of Trivium (*resp.* Grain-128), any state bit is not used for at least 64 (*resp.* 16) iterations after it has been modified. This way, up to 64 (*resp.* 16) iterations can be computed at once, at some additional combinatorial cost. For example, [13] shows that a 64-bit Trivium is 54% larger and has a maximum clock frequency that is 10% lower than a 1-bit implementation.

5 Conclusion

This paper reports updated results on the FPGA implementations of the eSTREAM Phase-2 Focus Candidates with Hardware Profile. Compared to Phase 1, only the specifications of Grain-128 and MICKEY-128 2.0 have been slightly modified. We re-implemented Trivium and Phelix for comparison purposes. These results suggest that Trivium and Grain-128 have the smallest complexities for a hardware implementation. Since these ciphers also provide the best opportunities to improve their throughput by exploiting parallelization techniques, the strongest cryptanalytic efforts should be concentrated on these ciphers. Additional efforts could also be devoted to an ASIC implementation of MICKEY-128 2.0 since its structure is not perfectly suitable for FPGA implementations. Finally, Phelix appears as an interesting alternative among the implemented ciphers, due to its significantly different design principles.

References

1. G. Avoine. *Cryptography in Radio Frequency Identification and Fair Exchange Protocols*. PhD thesis, EPFL, Lausanne, Switzerland, December 2005.
2. S. Babbage and M. Dodd. The stream cipher MICKEY-128 (version 1). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/016, 2005. `http://www.ecrypt.eu.org/stream`.
3. S. Babbage and M. Dodd. The stream cipher MICKEY-128 2.0. eSTREAM, ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream`.
4. C. De Cannière and B. Preneel. Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005. `http://www.ecrypt.eu.org/stream`.
5. C. De Cannière and B. Preneel. Trivium Specifications. eSTREAM, ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream`.
6. IP Cores. `http://http://ipcores.com/index.htm`.
7. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithms. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, pages 357 – 370. Springer, August 2004.
8. K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification, Second Edition*. John Wiley & Sons, Ltd., 2003.
9. M. D. Galanis, P. Kitsos, G. Kostopoulos, and O. Koufopavlou. Comparison of the Performance of Stream Ciphers for Wireless Communications. In *International Conference on Computing, Communications and Control Technologies 2004 — CCCT'04*, Austin, Texas, USA, August 14 – 17 2004.
10. M. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, pages 427 – 440. Springer, August—September 2005.
11. T. Good, W. Chelton, and M. Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2006. `http://www.ecrypt.eu.org/stream`.
12. F. K. Gürkaynak and P. Luethi. Recommendations for Hardware Evaluation of Cryptographic Algorithms. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/028, 2006. `http://www.ecrypt.eu.org/stream`.
13. F. K. Gürkaynak, P. Luethi, N. Bernold, R. Blattmann, V. Goode, M. Marghitola, H. Kaeslin, N. Felber, and W. Fichtner. Hardware Evaluation of eSTREAM Candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST, ZK-Crypt. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015, 2006. `http://www.ecrypt.eu.org/stream`.
14. M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010, 2005. `http://www.ecrypt.eu.org/stream`.
15. M. Hell, T. Johansson, W. Meier, and A. Maximov. A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream`.

16. K. Järvinen, M. Tommiska, and J. Skyttä. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. In *IEE Proceedings on Information Security*, volume 152, pages 3 – 12, October 2005.
17. P. Kitsos. On the Hardware Implementation of the MICKEY-128 Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/059. `http://www.ecrypt.eu.org/stream`.
18. P. Léglise, F.-X. Standaert, G. Rouvroy, and J.-J. Quisquater. Efficient Implementation of Recent Stream Ciphers on Reconfigurable Hardware Devices. In *26th Symposium on Information Theory in the Benelux*, pages 261–268, May 2005.
19. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications. In *International Symposium on Information Technology: Coding and Computing — ITCC 2004*, pages 583 – 587. IEEE Computer Society, April 2004.
20. The eSTREAM web site. eSTREAM, ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream`.
21. D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/020, 2005. `http://www.ecrypt.eu.org/stream`.
22. J. Zambreno, D. Nguyen, and A. Choudary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and Applications*, pages 575 – 585. Springer, August–September 2004.