EFFICIENT IMPLEMENTATION OF RECENT STREAM CIPHERS ON RECONFIGURABLE HARDWARE DEVICES

Philippe Léglise, François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater
UCL Crypto Group, Microelectronics laboratory, Université catholique de Louvain,
Place du Levant, 3 - 1348 Louvain-la-Neuve, Belgium
e-mails: {leglise, fstandae, rouvroy, jjq}@dice.ucl.ac.be

Stream ciphers have the reputation to be very efficient when implemented in hardware, much more efficient than any block cipher. However, although plenty of papers and books claim it, few results of hardware implementations of stream ciphers are available. In this paper, we provide FPGA implementation results of recent stream ciphers in order to evaluate their actual hardware efficiency. In addition, we compare these results with those of standard block ciphers (AES, 3DES, Rijndael, Misty1, ...). The selected stream ciphers are LILI-II, Helix and SNOW 2.0 and the implementation platform is a Virtex-II FPGA from Xilinx. On the basis of these results, it may be argued that while present stream ciphers allow us to obtain efficient implementations, they are not overwhelmingly more efficient than block ciphers. In general, their efficiency is comparable. However, stream ciphers are made of arguably low cost primitives which could provide really compact designs if correctly combined together.

INTRODUCTION

Stream ciphers are an important class of symmetric encryption algorithms. They contain internal states that vary with time and generate pseudo-random key bits, the keystream. The keystream is then bitwise XORed with the message to encrypt/decrypt. By contrast, block ciphers tend to simultaneously encrypt/decrypt blocks of bits of a message using a fixed encryption transformation. Stream ciphers are also more appropriate, and in some cases mandatory (*e.g.* in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may finally be advantageous in situations where transmission errors are highly probable.

Since 1999 and the creation of the European NESSIE project [10], stream ciphers have known a growing interest but they remain fewer and less investigated (at least from the hardware implementation point of view) than block ciphers. With regards to their usual efficiency claim: "*Stream ciphers can be conceived in order to be very efficient, more efficient than the block ciphers and this, more particularly in hardware*" [9] and in spite of the possible commercial applications resulting from this efficiency, this situation seems astonishing.

The goal of this contribution is to quantify this claim ("Is it true?"). As until now, few results of hardware implementations are available [1, 8], various types of stream ciphers will be implemented. The selected stream ciphers are LILI-II [3], Helix [6] and SNOW 2.0 [4] and the implementation platform is a Virtex-II FPGA. These designs are based on different principles and have been recently proposed (after 2002). Then, these implementation results will be compared with those of standard block ciphers. To this end, we have defined the hardware efficiency as the ratio throughput/area (the area corresponding to the number of slices used on an FPGA).

HARDWARE DESCRIPTION

All our implementations were carried out on a Xilinx Virtex XC2v6000ff1152-6 FPGA [13] which contains 33792 slices and 144 RAM blocks, which means 67584 LUTs and 67584 flip-flops. In the next sections, we compare the number of slices and RAM blocks of the different implementations. We also evaluated the delays and frequencies after place and route thanks to our implementation tool (Xilinx ISE-6).

LILI-II

Most of the time, modern stream ciphers are built with only one LFSR as basic element. LILI-II [3] is based on two bitwise LFSRs. The way in which non-linearity is introduced into LILI-II follows two principles. First, a completely irregular synchronization of the second LFSR (127-bit long) is used. It is at least synchronized once and maximum 4 times between the consecutive production of two keystream bits. This LFSR is controlled by the first LFSR (128-bit long) (see Figure 1). Second, a non-linear filter is used, which is reduced to a 12:1 truth table. The presence of these two LFSRs combined with a relatively simple introduction of non-linearity makes of LILI-II an "intuitive" stream cipher. For these reasons, it was logically retained for an hardware implementation. LILI-II uses a 128-bit key and a 128-bit IV (Initialization Vector). For further details about the algorithm, see [3].

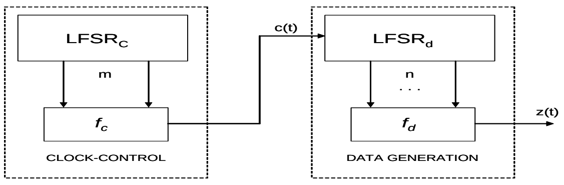

Figure 1: Overall view of LILI-II

HELIX

Helix [6] was retained for a hardware implementation because it is basically different from the traditional stream ciphers : no LFSR was used. All operations in Helix are on 32-bit words. These operations are addition modulo 2^{32}, XOR and left rotation by fixed numbers of bits. These operations are efficient in hardware. Helix combines the stream cipher and MAC (Message Authentication Code) functionalities and its design philosophy can be summarized as "many simple rounds". The Helix state is composed of 5 words (Z_0 to Z_4) of 32 bits each. In figure 2, half a block of Helix is

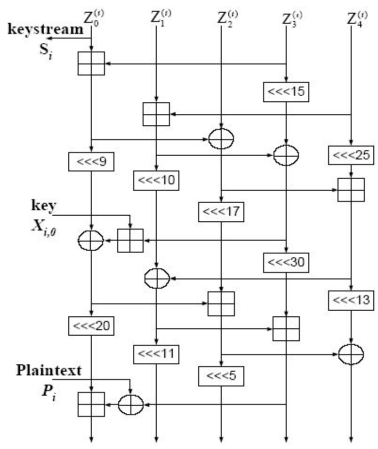

Figure 2: Half block of Helix

illustrated. Helix uses a 256-bit key and a 128-bit IV. Its key scheduling is complex and cannot be explained within this paper. For further details, see [6].

SNOW 2.0

SNOW 2.0 [4] is the evolution of SNOW 1.0 [5] and has been designed to improve performances and security. SNOW 2.0 (see Figure 3) is based on only one LFSR (contrary to LILI-II which uses two of them). It has the characteristic to work on 32-bit words rather than on a single bit one. It is thus interesting to verify whether that leads to an efficient implementation in hardware. This LFSR is 512 bits long. The non-linearity is provided by an FSM (Finite State Machine) based on the Rijndael S-Box. SNOW 2.0 uses a 128-bit or a 256-bit key and a 128-bit IV. For further details about the LFSR, the FSM and the key scheduling, see [4].

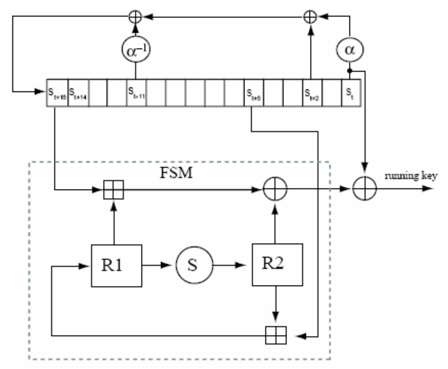

Figure 3: Design of SNOW 2.0

DESIGN ISSUES

Of the two styles of LFSRs, the usual style is called a Fibonacci LFSR. To shift a Fibonacci LFSR, you simply copy each bit to its neighbor on the right. The original rightmost bit is considered as the output. The bit that is shifted in at the left is the parity of some specific subset of the bits (the taps) of the register. The other style of LFSR is called a Galois LFSR, and has the same properties as the Fibonacci LFSR, but is shifted differently. To shift a Galois LFSR, each bit is copied to its neighbor on the right, except for the taps, for which the rightmost bit of the register is XORed in before the copy is done. The bit that is shifted in at the left is the original rightmost bit, which is also considered the output [7]. We now briefly describe our implementations of LILI, Helix and SNOW 2.0.

For LILI-II, both styles of LSFRs have been implemented. For the regularly clocked LFSR, the Galois LFSR needs 59 slices for a throughput of 384.6 Mbits/sec compared to 44 slices and a throughput of 273.6 Mbits/sec for the Fibonacci LFSR. Their efficiency ratios are 6.52 and 6.22 respectively. So, the Galois is the more efficient. The advantage of a Galois LFSR over a Fibonacci LFSR when being implemented in hardware is that a Galois LFSR usually has an even lower gate delay than a Fibonacci LFSR, resulting in a potentially lower clock cycle time. For the second LFSR of LILI-II, during the production of two consecutive keystream bits, 1 to 4 shifts have to be performed. To this end, in only one clock cycle, each one of its 127 registers has 4 possible different inputs. They correspond to the value that this same register should have if this LFSR was clocked 1, 2, 3, or 4 times. This value is selected by the output of the first LFSR. The resulting implementation uses 127 flip-flops and 127 multiplexers (4:1). In this case, the Galois LFSR needs 385 slices for a throughput of 238.1 Mbits/sec compared to 245 slices and a throughput of 203.5 Mbits/sec for the Fibonacci LFSR. Their efficiency ratios are 0.6 and 0.83 respectively. The Fibonacci LFSR is now more efficient due to the fact that for the generation of this style of LFSRs, only four equations have to be stored for the left-most taps whereas the Galois LFSR needs to store 4 equations per tap. The truth table has been achieved in a Virtex RAM block. The key scheduling requires 963 cycles of latency and uses a large memory for storing its intermediate states.

The hardware implementation of one Helix block is straightforward. It requires 329 slices for 2009 Mbits/sec which gives a ratio of 6.1. On the other hand, although efficient in software, the generation of the key words is cumbersome to deal with in hardware. For this reason, they are sometimes assumed to be precomputed, as in [8]. As a consequence, we provide the results of both a single block of Helix and the complete cipher with embedded generation of the key words. Helix then requires a latency of 26 cycles for encrypting/decrypting the first 32 bits of the message.

Finally, two versions of SNOW 2.0 have been implemented : one using the Virtex RAM blocks, the other not. The multiplication of a tap of the LFSR by α or α^{-1} can be done with the help of one dual-port RAM block and this is performed as explained in the original paper [4]. The only difference is that RAM blocks of the

Virtex are synchronous. So we have to take the taps one state before in order to get the good values at the appropriate moment (see Figure 4). The implementation of the whole LFSR requires 488 slices to reach 7,990 Mbits/sec. For the FSM, the synchronization problem is resolved as shown in Figure 4 where $\oplus$ is a XOR, $\boxplus$ an addition modulo 2^{32}, R1 and R2 two 32-bit registers. The FSM requires 90 slices and two RAM blocks for a throughput of 5,970 Mbits/sec.

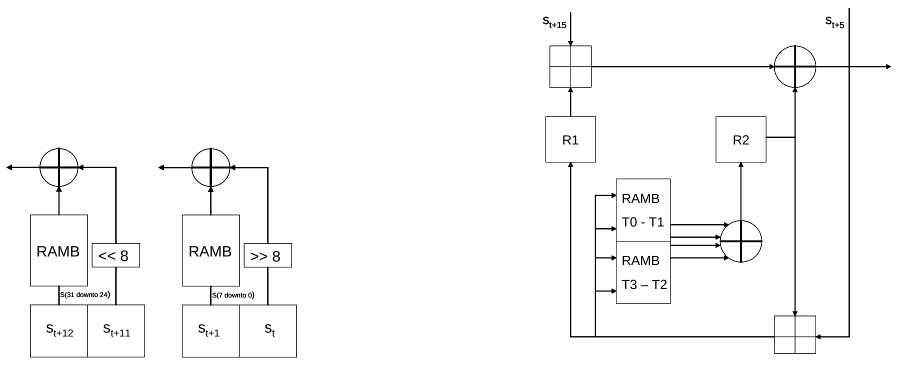

Figure 4: Multiplication by α or α^{-1} and FSM implemenation

The version of SNOW 2.0 implemented without RAM blocks stores the tables in the Virtex look-up tables which are used as ROMs. This version of the LFSR needs 795 slices for a throughput of 13,781 Mbits/sec. The FSM has been implemented as shown on the Figure 3. It requires 2,420 slices for a throughput of 5,351 Mbits/sec.

CONCLUSIONS

In this paper, four representative stream ciphers have been implemented. Table 1 summarizes our results and compares them with certain recent block ciphers on Xilinx FPGAs. Remark that strict comparisons are made difficult since these designs relate to different contexts (*e.g.* encryption/decryption designs, loop architectures or unrolled architectures for block ciphers). Looking at these results, the most efficient of all the ciphers is A5/1 which is also one of the weakest. With regard to other stream ciphers, Helix appears to be efficient as well, but requires some software precomputations, which may not be a practical solution for any context where the complete cipher has to be embedded on a single platform. LILI-II is not competitive with modern block ciphers and its efficiency is mainly limited by its expensive synchronization process. Finally, SNOW 2.0 allows the best implementation opportunities

Algorithm	**Nbr. of slices**	**Nbr. of RAMs**	**Throughput (Mbits/sec)**	**Efficiency Mbits/(sec.slices)**
STREAM CIPHERS - Virtex-II				
A5/1 [8]	32	0	188.3	5.88
E0 [8]	895	0	189	0.21
LILI-II	**866**	**1**	**243**	***0.28***
Helix (prec. key words) [8]	418	0	1,024	2.45
Helix block	**329**	**0**	**2,009**	**6.1**
Helix complete	**3,367**	**0**	**1,707**	**0.51**
RC4 [8]	140	3	120.8	*0.86*
SNOW 1.0 [1]	752	3	2,128	*2.83*
SNOW 2.0	**1,015**	**3**	**5,659**	***5.57***
SNOW 2.0	**2,420**	**0**	**5,351**	**2.21**
BLOCK CIPHERS - Virtex				
Twofish [12]	21,000	0	15,200	0.72
Serpent [12]	19,700	0	16,800	0.85
BLOCK CIPHERS - Virtex-E				
Camelia [12]	9,692	0	6,750	0.7
Khazad [12]	7,175	0	7,872	1.10
Misty1 [12]	6,322	0	10,176	1.61
Rijndael [12]	2,524	0	2,085	1.17
BLOCK CIPHERS - Virtex-II				
RC6 [12]	7,456	0	4,800	0.64
IDEA [12]	9,793	0	6,800	0.69
SHACAL-1 [12]	13,729	0	17,021	1.24
3DES [12]	604	0	917	1.51
ICEBERG [12]	4,946	0	17,344	3.51
BLOCK CIPHERS - Virtex-II + RAMBs				
Rijndael [12]	146	3	358	*2.45*
ICEBERG [12]	3,132	64	13,440	*4.29*
AES [11]	146	3	358	*2.45*

Table 1: Performances of block and stream ciphers on Xilinx FPGAs

and offers better efficiency than most recent block ciphers (excepte ICEBERG [12] that was specifically designed for FPGA implementations). As SNOW was originally software-oriented, we may expect the future design of an even better stream cipher dedicated to hardware. Remark that most stream ciphers have limited area requirements compared to block ciphers. Therefore, the main difference between block and stream ciphers may not be in their respective effectiveness, but rather in their ability to provide compact solutions for constraint contexts.

REFERENCES

[1] K. Alexander, R. Karri, I. Minkin, K. Wu, P. Mishra, X. Li, *Towards 10-100 Gbps Cryptographic Architectures*, in CATT/WICAT Annual Research Review, available from http://wicat.poly.edu/tech_report/tr/02-005.pdf, 2003.

[2] L. Batina, J. Lano, N. Mentens, B. Preneel, I. Verbauwhede, S. B. Örs, *Energy, Performance, Area versus Security Trade-offs for Stream Ciphers*, in ECRYPT Workshop, SASC - The State of the Art of Stream Ciphers, pp. 302-310, 2004.

[3] A. Clark, E. Dawson, J.Fuller, J.Golic, H-J. Lee, W. Millan, S-J.Moon, L. Simpson, *The LILI-II Keystream Generator*, ACISP'2002, 2002.

[4] P. Ekdahl, T. Johansson. *A new version ot the stream cipher SNOW*, available from http://www.it.lth.se/cryptology/snow/, 2002.

[5] P. Ekdahl, T. Johansson, *SNOW - a new stream cipher*, available from http://www.it.lth.se/cryptology/snow/, 2000.

[6] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, T. Kohno, *Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive*, in FSE'2003, 2003.

[7] I. Goldberg, D. Wagner, *Architectural Considerations for Cryptanalytic Hardware*, CS252 technical report, Berkeley, May 1996.

[8] M. D. Galanis, P. Kitsos, G. Kostopoulos, O. Koufopavlou, *Comparison of the Performance of Stream Ciphers for Wireless Communications*, proceedings of CCCT'04, Austin, Texas, USA, August 14-17, 2004.

[9] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[10] *NESSIE: New European Schemes for Signatures, Integrity, and Encryption*, available from http://www.cryptonessie.org, 2004.

[11] G. Rouvroy, *Secure and Reconfigurable Hardware Decoder for Digital Cinema Images*, PhD Thesis, UCL, June 2004.

[12] F.-X. Standaert, *Secure and efficient use of reconfigurable hardware devices in symmetric cryptography*, PhD Thesis, UCL, June 2004.

[13] Xilinx, *Virtex-II Data sheets*, available from http:// www.xilinx.com, 2003.