A High-Speed Hardware Implementation of the Hermes8-128 Stream Cipher

Paris Kitsos
Computer Science
School of Science and Technology
Hellenic Open University
Patras, Greece
e-mail: pkitsos@eap.gr

Ulrich Kaiser
Texas Instruments Deutschland GmbH
85350 Freising, Germany
e-mail: d-kaiser@ti.com

***Abstract*—An efficient high-speed hardware implementation of the Hermes8-128 stream cipher is presented in this paper. Hermes8-128 is proposed for hardware based implementations in the eSTREAM project [1]. Two FPGA devices are used for the hardware implementations. Especially, the XILINX (Spartan-2) 2S100-6 and (VIRTEX-4) 4VFX12-11 are used. A maximum throughput of 56.5 Mbps can be achieved with a clock frequency of 49 MHz with a XC2S100-6 device, while a throughput of 361 Mbps at 313 MHz is achieved with the 4VFX12-11 device. Since now only one previous reported Hermes8-128 hardware implementation exists, a comparison with the proposed one is given.**

I. INTRODUCTION

The continuous growing of mobility requires that engineers and developers design new cryptographic primitives with special care for speed, security and simplicity. RFID tags, smart cards and mobile pervasive-computing are typical examples of products where the amount of memory and power is very limited. The hardware implementations of today's algorithms, such as the AES cipher, are costly for devices with limited hardware resources, e.g. chip area or FPGA logic units. So, stream ciphers are useful in cases that low hardware complexity is needed.

The European Network of Excellence in Cryptology set up the eSTREAM project [1] with the main task to provide and recommend efficient stream ciphers for a wide variety of applications. One of the candidates is the Hermes8 [2] stream cipher. This cipher is proposed for both software (Profile-I) and hardware (Profile-II) byte-orientated implementations, e.g. Hermes8-128 with a key length of 16 bytes. Until now, one hardware implementation [3] of the Hermes8 has been presented in the literature. Its implementation is very compact, but with the drawback of performance. The proposed implementation has a different philosophy than that in [3] with the major goal to increase the performance for efficient use in applications with high throughput requirements.

While evaluating the performance of Profile-II candidates area requirements and time performance of an implementation are the most important metrics [4].

The organization of the paper is as following: In section 2, a brief introduction of the Hermes8-128 stream cipher is given. In section 3, the design methodology with the performance metrics are examined. The proposed architecture and VLSI implementation are presented in section 4. Implementation results and discussion (comparison with other works) are reported in section 5. Finally, section 6 concludes this paper.

II. HERMES8-128 STREAM CIPHER SPECIFICATIONS

Hermes8 is based on the Substitution-Permutation-Network (SPN) principle. The substitution (confusion) is performed by means of an S-BOX. The permutation and diffusion is performed by means of addressing the different state bytes, the different key bytes, and most importantly the chaining with help of the Accu. A basic block diagram for the Hermes8-128 cipher is illustrated in Fig. 1.

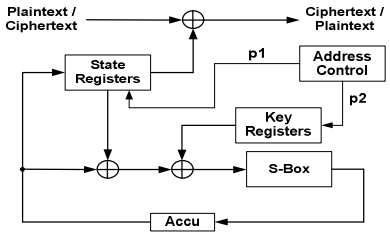

Figure 1. The basic Data Flow Diagram of the Hermes8 Stream Cipher

Hermes8-128 contains 16 key bytes and 37 state bytes. There are two pointers involved: p1 addresses one of the state bytes, p2 addresses one of the key bytes (see Fig. 1). The pointers obey modulo addition operation in order to assure that they always address valid register space. The use of pointers is favorable over shift register designs when low-power requirements are dominating the design.
The core state operation (called sub-round) consists of the following steps:
1. Select a certain state byte and EXOR it with Accu,
2. Select a certain key byte and EXOR it with previous result,
3. Take the previous result and apply the S-BOX function,

4. Store the previous result in Accu,
5. Copy Accu into the same state byte selected in step 1.

The S-BOX is 8-bit wide in order to provide a proper non-linear Boolean function needed for substitution, i.e. confusion. First choice is the known S-BOX of AES which is strong against differential cryptanalysis, however random number based S-BOXes are also suitable, if their differential distribution table (ddt) demonstrates good quality with respect to differential cryptanalysis attacks. The key bytes are modified every KEY_STEP3, i.e. seven steps, during the sub-round loops depending on the position of p2. Two temporary pointers p3 and p4 are addressing the key bytes following the byte addresses by p2. The byte k[p2] is not modified because it has to be used in the following sub-round. But the bytes k[p3] and k[p4] are 'rather old' and are therefore candidates for modification; they are replaced by SBOX[k[p3] exor k[p2]] and SBOX[k[p4] exor k[p2]] respectively. The exor'ing with k[p2] is advantageous over the direct application of the SBOX, because the inverse function of the SBOX does exist. Therefore, backtracking is hampered by means of this method. The dashed pointer in Fig. 2 represents the next p2 position (because KEY_STEP1=3) when addressing the next key byte needed for the next sub-round.

A similar method is followed for the key stream ks[] generation. The key stream bytes are derived from the state bytes state[]. Since the pointer p1 has been incremented after the last sub-round, it points to the 'oldest' available state byte. This is the first byte to be packed into the key stream block of sixteen bytes. Then further bytes follow by means of output pointer po that is incremented by two in order to separate consecutive sub-rounds from each other. Since a new output block of key stream bytes follows not earlier than the next STREAM_ROUNDS=3 are completed, the state byte contents corresponding to the same address are separated by 3 x 37 sub-rounds. During these 111 Hermes8-128 sub-rounds there are nearly 16 occurrences of key modification, i.e. about 32 key bytes are modified per output block in relation to 16 key byte registers. More information and also the Hermes8-128 cipher pseudo code can be found in the original specification and a related paper [2].

III. DESIGN METHODOLOGY

The design of Hermes8-128 is developed in VHDL with structural description logic such that it can be synthesized for FPGA devices. Especially two XILINX FPGA devices [5], the SPARTAN-II XC2S30 and VIRTEX-IV X4VFX12, are used in order to evaluate the performance of the proposed implementation.

To evaluate the performance of the proposed implementation the following performance metrics will be used in this paper.

- **Circuit Area (A)**

The term A represents the total circuit area that is required for the implementation, expressed in CLB numbers (# CLBs). The circuit area is obtained from synthesis results, and does not include buffers for clock distribution and additional overhead for placement and routing.

- **Maximum Clock Frequency (F)**

The maximum clock frequency, given in MHz, is determined by the critical path of the circuit.

- **Total Throughput (T)**

The total throughput of the algorithm expresses the number of cipher text bits simultaneously generated by the algorithm per second. It can be calculated from the following equation as:

$$T = \frac{\#bits \times F}{\#clock\ cycles} \qquad (1)$$

IV. HERMES8-128 HARDWARE ARCHITECTURE

Hermes8 is designed with a dedicated byte hardware implementation. The architecture that performs the Hermes8-128 stream cipher's key stream is shown in Fig. 2. This architecture mainly consists of the State Register, the Key Register, one S-Box and the Accu register. In addition, some multiplexers are there that support the correct operation of the Hermes8-128 cipher.

Two important modules are the Modulo Counters generator and the Control Unit. The Modulo Counters generate the appropriate count values (p1, p2, etc) used by the cipher.

For the initialization of these counters some predefined values (derived from the XOR of a number of key-bytes) must be loaded. The Control Unit produces all signals that are responsible for the correct synchronization and operation of the overall design.

Fig. 3 shows the implementation of the State Register. Actually this register is consisting of 37 byte registers, a codec circuit, and 37 2-input byte OR gates. This register initially stores the 37 IV bytes and each byte is updated by the output of the Accu register through the 2x1 byte OR gates. The circuit block codec has as input the p1 value and produces the proper byte register enable signals in order to update the right byte at the right time according to the p1 value.

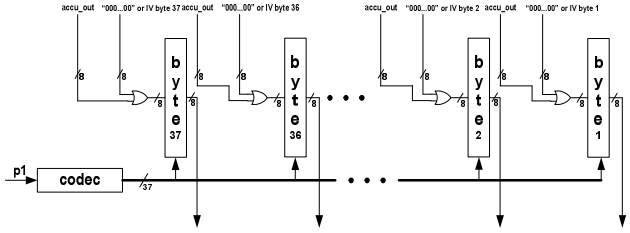

Figure 3. The Implementation of the State Register

Finally, the implementation of the Key Register is depicted in Fig. 4. This register consist of 16 byte registers, 16 2x1 8-bit Multiplexers (MUXes), three 16x1 byte Multiplexers (MUXes), 16 2-input byte OR gates, two S-Boxes and 16 3x1 OR gates. In this register initially the 16 key bytes are stored and each byte is updated either by the K[p3]new or K[p4]new values through the 2x1 byte MUXes. The byte registers' outputs are collected by the 16x1 byte MUX controlled by the pointer p2 in order to produce the K[p2].

In addition, the registers' outputs values are collected by two different 16x1 byte MUXes.

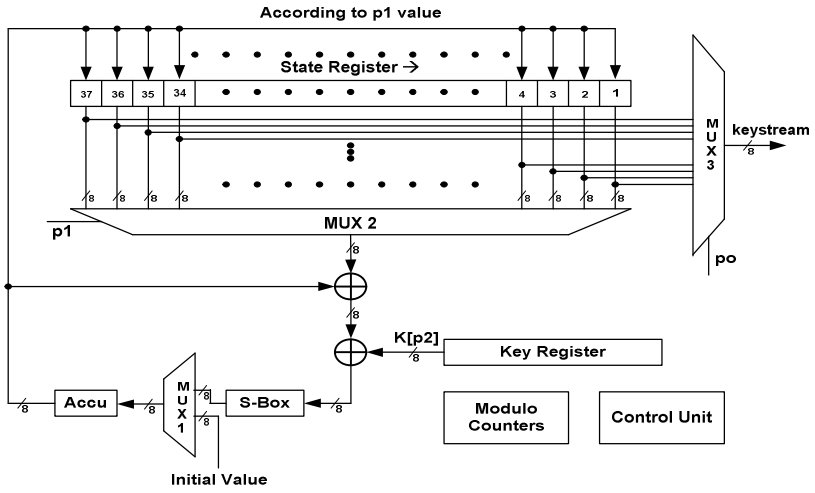

Figure 2. The Hermes8-128 Stream Cipher Architecture

The output of the first one, controlled by the p3 value, XORed with the K[p2] value and the result addresses the S-Box in order to produce the K[p3]new value. In a similar way the output of the second MUX, controlled by the p4 value, XORed with the K[p2] value and the result is used by the S-box in order to produce the K[p4]new value.

The values of K[p3]new and K[p4]new are used in order to update the new values of Key Register through the 2x1 byte MUXes supported by the enable16_p3 and enable16_p4 control signals respectively. The signal enable16 is used in order to initialize the output of each byte register with each key byte. By means of this design no dead clock cycles are needed in order to produce the new K[p2] values.

The operation of the proposed design (Fig. 2) starts with the initial parallel loading of the IV bytes into the State Registers and the Key bytes into the Key Register. This is why the OR gates between the byte registers and the MUXes are implemented as shown in Fig. 3 and Fig. 4. After the loading phase these inputs are forced with zeros and the system starts to operate due to the user's command. The MUX1 is used either to fetch the Accu register by its initial value or the output of the S-Box. The appropriate state byte, as the cipher specifications demands, is XORed with the Accu value selected by the MUX2. The result is XORed with the K[p2] value and then the new value is applied as address to the S-Box. Afterwards, the output value of the S-Box is stored in the Accu register and the Accu output value is copied into the State Register in that position showed by p1 value. Finally, the MUX3 is used in order to produce the keystream bytes according to the po address.

In this architecture no dead clock cycles are needed; the execution time for the initialization phase is 370 (init_rounds x 37) clock cycles and the execution time of the stream cipher is 481 clock cycles (init_rounds x 37 + stream_rounds x 37) for the generation of the first 16 cipher text bytes. Any further block of 16 output bytes to be delivered requires further 111 clocks.

V. HARDWARE IMPLEMENTATION RESULTS

The measurements of the hardware results and performance analysis are shown in Table 1. Two FPGA devices were used: Especially, the XILINX (Spartan-2) XC2S100-6 and (VIRTEX-4) 4VFX12-11.

Also, comparisons with the previous Hermes8 [3] cipher in term of area requirements and time performance is given. Finally, comparisons with other stream ciphers' implementations [6, 7] are added in order to have a fair and detailed comparison with the two proposed implementations. Besides, in the eSTREAM project, the hardware ciphers are dedicated for the low hardware resource environment [4] and the compactness of a hardware stream cipher is important in the evaluation in eSTREAM. Therefore, all FPGA stream ciphers' implementations are usually compared with two compact AES implementations [8, 9].

The Hermes8 implementation in [3] is a very compact hardware implementation. The data path consists of an 8-bit XOR operation, the AES S-box implemented is using composite field arithmetic in $GF(((2^2)^2)^2)$ with resource sharing of the 4-bit $GF((2^2)^2)$ multiplier and a dedicated unit to perform modulo reduction of an 8-bit value and is only used in the initialization phase. This implementation achieves a throughput up to 5.6 Mbps at a clock frequency equal to 45 MHz. The proposed implementation in this paper has an extremely different design philosophy compared with the implementation in [3]. The proposed implementation is efficient for applications with high throughput requirements – however, with a drawback in hardware resources. The throughput is measured after the initialization phase.

In [6] a small hardware implementation of the Edon80 stream cipher is also presented. Achieved is a throughput of 1.33 Mbps at 106 MHz for XC2S15-6 FPGA device and a throughput of 3.58 Mbps at 286 MHz for XC4VLX15-12 FPGA device.

In addition, in [7] an efficient implementation of MICKEY-128 cipher is presented. It achieves a throughput

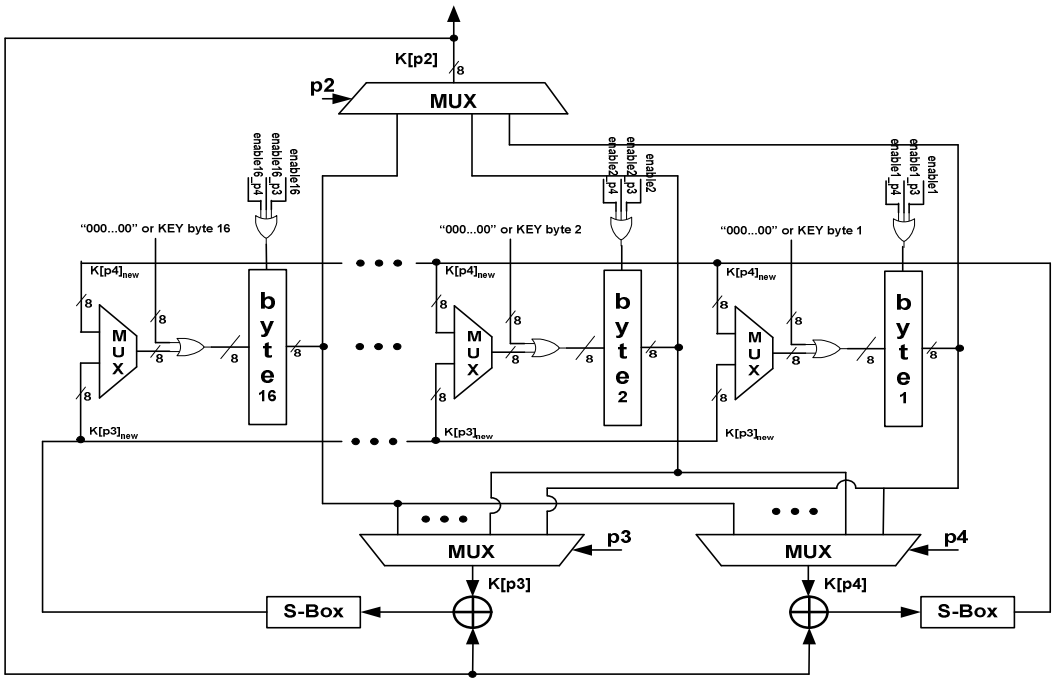

Figure 4. The Implementation of the Key Register

TABLE I. PERFORMANCE ANALYSIS AND COMPARISONS

Architecture	*FPGA Device*	*# CLBs*	*F (MHz)*	*Throughput (Mbps)*
Hermes8 [3]	Spartan-2 XC2S30-5	190	45	5.6
Edon80 [6]	Spartan-2 XC2S15-6	52	106	1.33
Edon80 [6]	Virtex4 XC4VLX15-12	45	286	3.58
MICKEY-128 [7]	Virtex XCV50ECS144	167	170	170
AES [8]	Spartan-2 XC2S30-6	222	60	69
AES [9]	Spartan-2 XC2S15-6	124	67	2.2
Hermes8 proposed	Spartan-2 XC2S100-6	697	49	56.5
Hermes8 proposed	Virtex-4 XC4VFX12-11	715	313	361

of 170 Mbps at 170 MHz clock frequency. In [8] and [9] two very compact FPGA implementations of the AES block cipher are shown. The implementation in [8] is based on a 32-bit architecture while the implementation in [9] is based on an 8-bit architecture.

As the above table illustrates the proposed cipher implementation achieves better time performance compared with the others - of course - with a drawback in hardware resources.

VI. CONCLUSIONS

An efficient hardware implementation of the new stream cipher Hermes8-128 is presented in this paper. The proposed implementation outperforms any previous hardware implementation of the same cipher. The synthesis results prove that the proposed implementation is suitable for FPGA implementation. In addition, it is suitable for applications with high throughput requirements.

REFERENCES

[1] The eSTREAM Call for Stream Cipher Primitives, http://www.ecrypt.eu.org/stream/call/

[2] U. Kaiser, "Hermes Stream Cipher", eSTREAM, ECRYPT Stream Cipher Project, Report 2006/057.

[3] T. Good, W. Chelton and M. Benaissa, "Review of stream cipher candidates from a low resource hardware perspective", SASC 2006 Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006.

[4] L. Batina, S. Kumar, J. Lano, K. Lemke, N. Mentens, C. Paar, B. Preneel, K. Sakiyama and I. Verbauwhede, "Testing Framework for eStream Profile-II Candidates", SASC 2006 Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006.

[5] Xilinx Incorporated. Silicon Solutions — Virtex Series FPGAs (2006).

[6] M. Kasper, S. Kumar, K. Lemke-Rust and C. Paar, "A Compact Implementation of Edon80", eSTREAM, ECRYPT Stream Cipher Project, Report 2006/057.

[7] P. Kitsos, "On the Hardware Implementation of the MICKEY-128 Stream Cipher", eSTREAM, ECRYPT Stream Cipher Project, Report 2006/059.

[8] P. Chodowiec and K. Gaj., "Very Compact FPGA Implementation of the AES Algorithm", Cryptographic Hardware and Embedded Systems - CHES 2004, volume 2779 of LNCS, pages 319-333. Springer, 2003.

[9] T. Good and M. Benaissa, "AES FPGA from the Fastest to the Smallest", Cryptographic Hardware and Embedded Systems - CHES 2005, volume 3659 of LNCS, pages 427- 440. Springer, 2005.