ASIC Implementations of the Block Cipher SEA for Constrained Applications

François Macé*, François-Xavier Standaert**, Jean-Jacques Quisquater

UCL Crypto Group, Université Catholique de Louvain.

e-mails: mace,fstandae,jjq@uclouvain.be

Abstract. SEA is a scalable encryption algorithm targeted for small embedded applications. It was initially designed for software implementations in controllers, smart cards or processors. In this paper, we investigate its hardware performances in a 0.13 μm CMOS technology. For these purposes, different designs are detailed. First, a single clock cycle per round loop architecture is implemented. Beyond its low cost performances, a significant advantage of the proposed encryption core is its full flexibility for any parameter of the scalable encryption algorithm, taking advantage of generic VHDL coding. Second, a more realistic design with a reduced datapath combined with a serial communication interface is described in order to put forward the low-power opportunities of SEA. Finally, a minimum datapath is presented and its applicability to RFID encryption is discussed. Additionally to these results, performance comparisons with the AES Rijndael are proposed. They illustrate the interest of platform/context-oriented block cipher design and, as far as SEA is concerned, its low area requirements and reasonable efficiency.

1 Introduction

SEA is a parametric block cipher for resource constrained systems (*e.g.* sensor networks, RFIDs) that has been introduced in [17]. It was initially designed as a low-cost encryption/authentication routine (*i.e.* with small code size and memory) targeted for processors with a limited instruction set (*i.e.* AND, OR, XOR gates, word rotation and modular addition). Additionally and contrary to most present encryption algorithms (*e.g.* the DES [4] and AES Rijndael [3, 5]), the algorithm takes the plaintext, key *and* bus sizes as parameters and therefore can be straightforwardly adapted to various implementation contexts and/or security requirements. Compared to older solutions for low cost encryption like TEA (Tiny Encryption Standard) [21] or Yuval's proposal [22], SEA also benefits from a stronger security analysis, derived from recent advances in block cipher design/cryptanalysis. In practice, SEA has been proven to be an efficient solution for embedded software applications. In [15], the features of a low cost FPGA encryption/decryption core have also been detailed. But its hardware performances in a recent CMOS technology have not yet been investigated. Consequently, this paper explores the space *vs.* speed *vs.* power consumption tradeoffs of various designs for SEA. First, we consider a single cycle per round loop implementation.

* François Macé is a PhD student funded by the FRIA, Belgium.

** Postdoctoral researcher of the Belgian Fund for Scientific Research (FNRS).

In addition to its performance evaluation, we show that the algorithm's scalability can be turned into a *fully generic* VHDL design, so that any text, key *and* bus size can be straightforwardly re-implemented without any modification of the hardware description language, with standard synthesis and implementation tools. Then, we consider more realistic scenarios for constrained applications and investigate the low power capabilities of SEA. For this purpose, we detail a design combining a reduced datapath with a serial communication interface, executing each round in 15 clock cycles. Finally, we present a minimum datapath and discuss its application to RFID encryption constraints with respect to similar designs proposed for the AES Rijndael. These results illustrate the interest of platform/context oriented block cipher design for constrained applications. The rest of the paper is structured as follows. Section 2 describes the algorithm. Sections 3, 4 and 5 respectively detail our architectures and datapaths for SEA in different implementation contexts. Finally, conclusions are in Section 6.

2 Algorithm Description

In this section, we give a complete description of the algorithm, starting with the important parameters, then emphasizing its basic operation. Afterwards follows the round and key round description and finally the generic pseudo-C code for the whole execution of encryption and decryption.

2.1 Parameters and definitions

$\text{SEA}_{n,b}$ operates on various text, key and word sizes. It is based on a Feistel structure with a variable number of rounds, and is defined with respect to the following parameters:

- n: plaintext size, key size.
- b: processor (or word) size.
- $n_b = \frac{n}{2b}$: number of words per Feistel branch.
- n_r: number of block cipher rounds.

As only constraint, it is required that n is a multiple of $6b$[1]. For example, using an 8-bit processor, we can derive a 96-bit block ciphers, denoted as $\text{SEA}_{96,8}$.

Let x be a $\frac{n}{2}$-bit vector. We consider two representations:

- Bit representation: $x_b = x(\frac{n}{2} - 1) \ldots x(2)\ x(1)\ x(0)$.
- Word representation: $x_W = x_{n_b-1}\ x_{n_b-2} \ldots x_2\ x_1\ x_0$.

2.2 Basic operations

Due to its simplicity constraints, $\text{SEA}_{n,b}$ is based on a limited number of elementary operations (selected for their availability in any processing device) denoted as follows: (1) bitwise XOR $\oplus$, (2) addition mod 2^b $\boxplus$, (3) a 3-bit substitution box $S := \{0, 5, 6, 7, 4, 3, 1, 2\}$ that can be applied bitwise to any set of 3-bit words for efficiency purposes. In addition, we use the following rotation operations:

[1] since n_b must be integer and a multiple of 3 for the right use of the S-box

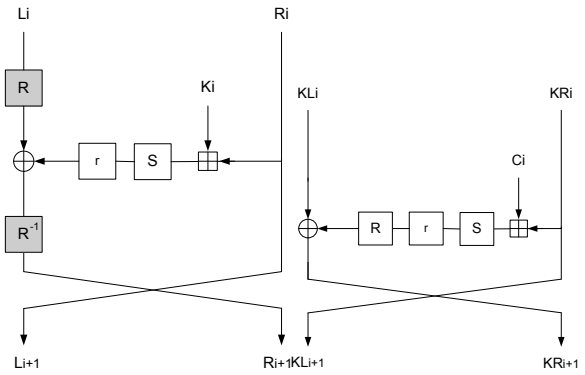

Fig. 1. Encrypt/decrypt round and key round.

(4) Word rotation R, defined on n_b-word vectors:

$$R : \mathbb{Z}_{2^b}^{n_b} \to \mathbb{Z}_{2^b}^{n_b} : x \to y = R(x) \Leftrightarrow \quad \begin{array}{l} y_{i+1} = x_i, 0 \le i \le n_b - 2, \\ y_0 = x_{n_b - 1} \end{array}$$

(5) Bit rotation r, defined on n_b-word vectors:

$$r : \mathbb{Z}_{2^b}^{n_b} \to \mathbb{Z}_{2^b}^{n_b} : x \to y = r(x) \Leftrightarrow \quad \begin{array}{ll} y_{3i} = x_{3i} \ggg 1, & \\ y_{3i+1} = x_{3i+1}, & \\ y_{3i+2} = x_{3i+2} \lll 1, & 0 \le i \le \frac{n_b}{3} - 1, \end{array}$$

where $\ggg$ and $\lll$ represent the cyclic right and left shifts inside a word.

2.3 The round and key round

Based on the previous definitions, the encrypt round F_E, decrypt round F_D and key round F_K are pictured in Figure 1 and defined as:

$$\begin{array}{lll} [L_{i+1}, R_{i+1}] = F_E(L_i, R_i, K_i) & \Leftrightarrow & R_{i+1} = R(L_i) \oplus r\big(S(R_i \boxplus K_i)\big) \\ & & L_{i+1} = R_i \\ [L_{i+1}, R_{i+1}] = F_D(L_i, R_i, K_i) & \Leftrightarrow & R_{i+1} = R^{-1}\Big(L_i \oplus r\big(S(R_i \boxplus K_i)\big)\Big) \\ & & L_{i+1} = R_i \\ [KL_{i+1}, KR_{i+1}] = F_K(KL_i, KR_i, C_i) & \Leftrightarrow & KR_{i+1} = KL_i \oplus R\Big(r\big(S(KR_i \boxplus C_i)\big)\Big) \\ & & KL_{i+1} = KR_i \end{array}$$

2.4 The complete cipher

The cipher iterates an odd number n_r of rounds. The following pseudo-C code encrypts a plaintext P under a key K and produces a ciphertext C. P, C and K have a parametric bit size n. The operations within the cipher are performed considering parametric b-bit words.

C=**SEA**$_{n,b}(P, K)$
{
  % initialization:
    $L_0 \& R_0 = P$;
    $KL_0 \& KR_0 = K$;

% key scheduling:
for i **in** 1 **to** $\lfloor \frac{n_r}{2} \rfloor$
$[KL_i, KR_i] = F_K(KL_{i-1}, KR_{i-1}, C(i))$;
switch $KL_{\lfloor \frac{n_r}{2} \rfloor}$, $KR_{\lfloor \frac{n_r}{2} \rfloor}$;
for i **in** $\lceil \frac{n_r}{2} \rceil$ **to** $n_r - 1$
$[KL_i, KR_i] = F_K(KL_{i-1}, KR_{i-1}, C(r - i))$;
% encryption:
for i **in** 1 **to** $\lceil \frac{n_r}{2} \rceil$
$[L_i, R_i] = F_E(L_{i-1}, R_{i-1}, KR_{i-1})$;
for i **in** $\lceil \frac{n_r}{2} \rceil + 1$ **to** n_r
$[L_i, R_i] = F_E(L_{i-1}, R_{i-1}, KL_{i-1})$;
% final:
$C = R_{n_r} \& L_{n_r}$;
switch KL_{n_r-1}, KR_{n_r-1};
},

where & is the concatenation operator, $KR_{\lfloor \frac{n_r}{2} \rfloor}$ is taken before the switch and $C(i)$ is a n_b-word vector of which all the words have value 0 excepted the LSW that equals i. Decryption is exactly the same, using the decrypt round F_D.

3 Generic loop Architecture

Our generic loop architecture supports both encryption and decryption and executes one round per clock cycle. It yields a straightforward implementation in which the round function and key schedule do not share any resources. Additionally, the left branch of the round function must be designed to support the `word rotate` operation and its inverse. The proposed implementation benefits from the same structure as the one detailed in [15] for FPGAs. Its purpose is to illustrate the high scalability of the algorithm and its achievements on throughput/area tradeoffs. Such a design is typically interesting when integrated in large scale systems. By contrast, because of its n-bit architecture and interface, it is not perfectly suited to low power, low cost applications.

3.1 Implementation details

The structure of our generic loop architecture of SEA is depicted in figure 2, with the round function details in the left part and the key schedule in the right part. It has a Feistel structure working on n-bit data blocks, each branch computing operations on $\frac{n}{2}$-bit operands. Resources consuming blocks are the S-boxes and the mod2^b adders. The `Word Rotate` and `Bit Rotate` blocks are implemented by swapping wires. Encryption and decryption are supported by two multiplexors controlled by the *Encrypt* signal. Two additional multiplexors in the key schedule allow to switch the right and left part of the round key ($KR_{\lfloor \frac{n_r}{2} \rfloor}$ and $KL_{\lfloor \frac{n_r}{2} \rfloor}$) at half the execution using the *Switch* command signal. Finally, the multiplexor controlled by *HalfExec* provides the round function with the right part of the Roundkey, transmitting its left part instead of the right one after the switch. Supplementary area consumption is caused by the routing pathes.

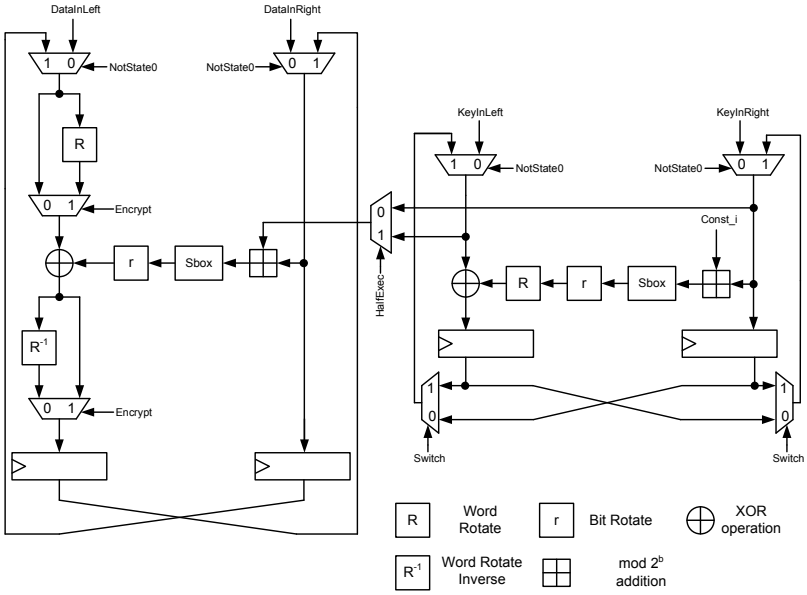

Fig. 2. Architecture of the generic loop implementation.

The algorithm can easily beneficiate of a modular implementation, taking as only mandatory parameters the size of the plaintexts and keys n and the word length b. The number of rounds n_r is an optional input that can be automatically derived from n and b according to the guidelines given in [17]. From the datapath description of Figure 2, a scalable design can then be straightforwardly obtained by using generic VHDL coding. A particular care only has to be devoted to an efficient use of the mod 2^b adders in the key scheduling part. In the round function, the mod 2^b adders are realized by using n_b b-bits adders working in parallel without carry propagation between them. In the key schedule, the signal `Const_i` can only take a value between 0 and $\frac{n_r}{2}$. Therefore, it may not be necessary to use n_b adders. If $\log_2(\frac{n_r}{2}) \leq b$, then a single adder is sufficient. If $\log_2(\frac{n_r}{2}) > b$, then $\lceil \frac{log_2(\frac{n_r}{2})}{b} \rceil$ adders will be required. In the next section, we detail the implementation results of this architecture for different parameters.

3.2 Implementation Results

We implemented SEA with the parameters n and b respectively ranging from 96 to 144 and from 4 to 12. The modular design was written in VHDL. The synthesis was realized using Synopsys Design Analyzer ® [19] and the place and route was performed using Cadence ® Soc Encounter^{TM}[2]. We used the 0.13 μm high density standard cell library from UMC[20] with a power supply voltage of $1.2V$. Timing constraints for synthesis and place and route were set to achieve a maximal frequency of 250 MHz. Power consumption figures were extracted using power tools provided by Soc Encounter^{TM}, using post place and route simulations of the circuit activity and a wireload model.

Our results are summarized in Table 1 for various parameters. For comparison purposes, we reported implementation results of the AES Rijndael. In addition, we also provide an ASIC implementation of the ICEBERG block cipher [18] that was specifically designed for efficient FPGA implementations. Both architectures have been chosen for comparison because of a similar "one clock cycle per round" implementation context. These results clearly illustrate the space

Algo.	n	b	n_r	Clock Frequency [MHz]	Throughput [Mbps]	Area [μm^2]	Gate Equivalent @ Synthesis	Gate Equivalent @ P& R	Total Power [μW]
SEA	96	8	93	250	258	22362	3758	4313	5102.64
SEA	108	6	111	250	243	23668	4003	4565	5844.02
SEA	126	7	117	250	269	28241	4770	5447	7216.96
SEA	132	11	121	250	273	29638	5071	5715	7894.62
SEA	144	4	149	250	242	32894	5764	6345	8029.56
SEA	144	6	139	250	259	32137	5525	6199	7789.28
SEA	144	8	135	250	267	31523	5427	6079	8201.22
SEA	144	12	133	250	271	31622	5550	6100	8183.44
AES [16]	128	-	10	224	2609.11	130 000	-	21337	-
AES [11]	128	-	10	295	3840	790 000	-	73200	86 000
ICEBERG	64	-	16	250	1000	45679	7732	8811	9577.11

Table 1. Implementation results for SEA, the AES and ICEBERG: n-bit loop architecture. (Total power evaluated at the maximum working frequency)

vs. throughput tradeoff between SEA and the AES, which also appears in the power consumption figures. Similarly, the table underlines the different optimization goals of SEA (minimum code size in software) and ICEBERG (maximum throughput/area ratio in reconfigurable hardware). We observe that, although not directly purposed for hardware implementations, SEA performs relatively well in this context. We also note that all these architectures allow both encryption and decryption facilities. Due to a particular structure of the key scheduling, SEA and ICEBERG additionally provide "on-the-fly" key derivation both for encryption and decryption (which is usually not achievable with other ciphers).

4 Reduced datapath with serial interface

The implementation results presented in the previous section may not be sufficient for resource constrained applications. In particular, the proposed n-bit architecture and interface for loading the key and data is not realistic in such contexts. As a consequence, this section investigates an alternative implementation, trading throughput for area and combined with a smaller block (b-bit) serial interface. For these purposes, we take advantage of a reduced datapath and modify the description of the round and key round with the following goals:

- Reduce the area consumption of the module.
- Reduce its power consumption.
- Support both encryption and decryption.
- Achieve a good tradeoff between area, power and throughput.

4.1 Re-scheduling of the algorithm

The proposed re-scheduling of the round function and key expansion operations are respectively shown in Algorithms 1 and 2. In order to better reduce the area requirements, we also had to abandon a part of the genericity in our VHDL codes: the parameter n_b is now fixed to 6, but the value of b is still parametric. In the re-scheduled algorithms, the `word rotate` operation is directly implemented as the registers R_i, L_i, KR_i and KL_i are accessed in read and/or write operations. Seven working registers (registers A to G) are needed in order to execute all the operations taking into account data dependencies. As it is easily observable, both round function and key schedule operation take 10 cycles to be fully executed.

Algorithm 1 Alternative implementation of SEA - Round Function

Input: $R_i, L_i, RK_i \in \mathbf{Z}_{2^b}^{n_b}$
Output: R_{i+1}, L_{i+1}

	E/D	Encryption	Decryption
1:	$A \leftarrow R_{i,0} + RK_{i,0}$;		
2:	$B \leftarrow R_{i,1} + RK_{i,1}$;		
3:	$C \leftarrow R_{i,2} + RK_{i,2}$;		
4:	$(D, E, F) \leftarrow r(S(A, B, C))$;	$A \leftarrow R_{i,3} + RK_{i,3}$;	$C \leftarrow R_{i,5} + RK_{i,5}$;
		$G \leftarrow L_{i,5}$;	$G \leftarrow R_{i,5}$;
5:	$B \leftarrow R_{i,4} + RK_{i,4}$;	$G \leftarrow L_{i,0}$;	$R_{i+1,5} \leftarrow L_{i,0} \oplus D$;
	$L_{i+1,0} \leftarrow R_{i,0}$;	$R_{i+1,0} \leftarrow D \oplus G$;	
6:	$L_{i+1,1} \leftarrow R_{i,1}$;	$R_{i+1,1} \leftarrow E \oplus G$; $G \leftarrow L_{i,1}$;	$A \leftarrow R_{i,3} + RK_{i,3}$;
		$C \leftarrow R_{i,5} + RK_{i,5}$;	$R_{i+1,0} \leftarrow L_{i,1} \oplus E$;
7:	$(D, E, F) \leftarrow r(S(A, B, C))$;	$R_{i+1,2} \leftarrow F \oplus G$;	$R_{i+1,1} \leftarrow L_{i,2} \oplus F$;
	$L_{i+1,2} \leftarrow R_{i,2}$;	$G \leftarrow L_{i,2}$;	
8 :	$L_{i+1,3} \leftarrow R_{i,3}$	$R_{i+1,3} \leftarrow D \oplus G$; $G \leftarrow L_{i,3}$;	$R_{i+1,2} \leftarrow L_{i,3} \oplus D$;
9 :	$L_{i+1,4} \leftarrow R_{i,4}$;	$G \leftarrow L_{i,4}$; $R_{i+1,4} \leftarrow E \oplus G$;	$R_{i+1,3} \leftarrow E \oplus L_{i,4}$;
10:		$R_{i+1,5} \leftarrow F \oplus G$;	$R_{i+1,4} \leftarrow L_{i,5} \oplus F$;
		$L_{i+1,5} \leftarrow R_{i,5}$;	$L_{i+1,5} \leftarrow G$;

Algorithm 2 Alternative implementation of SEA - Key Schedule

Input: KR_i, $KL_i \in \mathbf{Z}_{2^b}^{n_b}$, $Const_i \in \mathbf{Z}_{2^b}$
Output: kR_{i+1}, kL_{i+1}

	E/D	Encryption	Decryption
1:	$Ak \leftarrow KR_{i,0} + Const_i$;		
2:	$Bk \leftarrow KR_{i,1}$;		
3:	$Ck \leftarrow KR_{i,2}$;		
4:	$(Dk, Ek, Fk,) \leftarrow r(S(1k, Bk, Ek))$;	$Ak \leftarrow KR_{i,3}$; $Ck \leftarrow KR_{i,5}$;	
5:	$Bk \leftarrow KR_{i,4}$; $KR_{i+1,1} \leftarrow KL_{i,1} \oplus Dk$; $KL_{i+1,1} \leftarrow KR_{i,1}$		
6:	$KR_{i+1,2} \leftarrow KL_{i,2} \oplus Ek$; $KL_{i+1,2} \leftarrow KR_{i,2}$;	$Ck \leftarrow KR_{i,5}$; $Ak \leftarrow KR_{i,3}$;	
7:	$(Dk, Ek, Fk,) \leftarrow r(S(Ak, Bk, Ek))$; $KL_{i+1,3} \leftarrow KR_{i,3}$;		
	$KR_{i+1,3} \leftarrow KL_{i,3} \oplus Fk$;		
8:	$KR_{i+1,0} \leftarrow KL_{i,0} \oplus Fk$; $KL_{i+1,0} \leftarrow KR_{i,0}$;		
9:	$KR_{i+1,4} \leftarrow KL_{i,4} \oplus Dk$; $KL_{i+1,4} \leftarrow KR_{i,4}$;		
10:	$KR_{i+1,5} \leftarrow KL_{i,5} \oplus Ek$; $KL_{i+1,5} \leftarrow KR_{i,5}$;		

4.2 Implementation Structure

In order to achieve the goals of reducing the area and power consumption requirements, our proposed implementation shares resources between the round function and key schedule and works on b-bit operands. As a consequence they cannot be executed in parallel anymore. Let us denote the ith operation in the round function by r_i and the ith operation in the key round by k_i. Looking

carefully at Algorithms 1 and 2 reveals that certain operations can still be managed in parallel. Namely, k_1 can be executed together with r_8, k_2 with r_9 and k_3 with r_{10}. In the same manner, we can execute k_9 with r_1 and k_{10} with r_2. This results in an execution of the combined round/keyround taking only 15 cycles. Contrary to the implementation presented in section 3, we provide functionalities to load the cleartext and the key and to send back the ciphertext after encryption, through a b-bit serial interface. Taking these interfaces into account, both encryption and decryption can be concluded in $33 + 15 * n_r$ clock cycles.

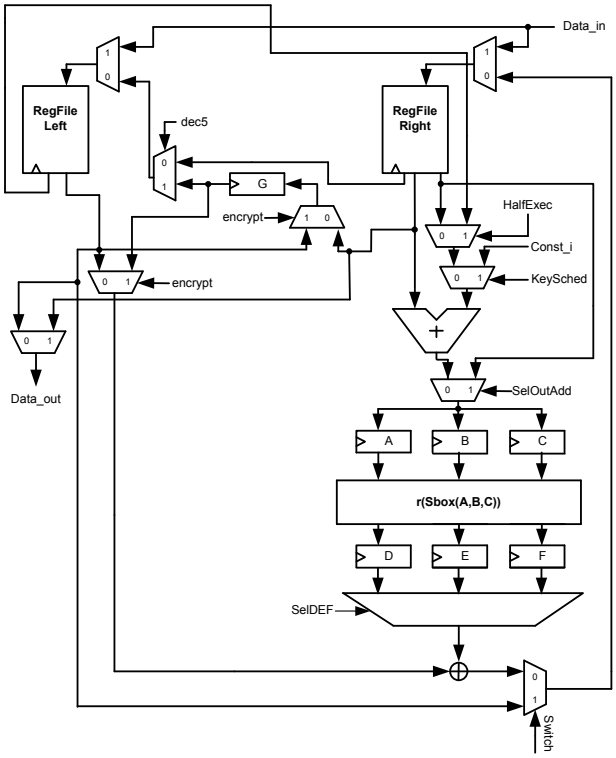

Fig. 3. Architecture of the alternative implementation (b bit operands).

An architecture realizing these implementation choices is presented in Figure 3. Again, a number of control signals are necessary to manage the execution of the algorithm. *encrypt* and *dec5* allow the selection of the right path for encryption and decryption. *KeySched* and *SelOutAdd* allows switching the input path during the round function or during the execution of the key schedule. *HalfExec* controls the selection of the right or left part of the round key during the first half or the second half of the encryption/decryption process. Finally, *Switch* allows directly accessing the register file of the right part during the switch between the two parts of the key. As some registers of the left and right register files have to be simultaneously accessed, we additionally need to generate single input - multiple outputs register files. The control part is realized using an FSM.

4.3 Implementation Results

Implementation results for the architecture of Figure 3 are given in table 2, for different values of the generic parameter b. Synthesis and place and route were realized using the same tools as in the previous section, with a maximal frequency of 80 MHz and optimized for area constraints. Compared to the previous generic loop architecture, a first observation is that the area gains of the modified architecture are negligible (when they exist). This is mainly due to the different interfaces these architectures are using. As a matter of fact, the serial interface considered in this section is more realistic for low cost, low power applications. Nevertheless, the alternative architecture saves a reasonable amount of power. For example, the generic loop and alternative architectures for $\text{SEA}_{144,12}$ respectively consume 59 μW and 25 μW at 100 KHz. We find 36 μW and 19 μW for $\text{SEA}_{96,8}$. The same trend holds at higher frequencies, where the leakage power becomes negligible. Of course, this reduced power consumption comes at the cost of a reduced throughput. Interestingly, a comparison with similar architectures for the AES Rijndael reveals that the space *vs.* throughput tradeoff is not as straightforward as for the previous loop architecture: specialized datapaths for the AES can achieve nice performances for low cost applications. These results illustrate the good ability of the AES Rijndael to fit to any kind of platform and application context. From these results, the AES implementations also look more power consuming. But this statement (as any comparison in the paper) probably has to be tempered by different design choices in the referenced implementations. These results are only purposed to underline intuitive facts regarding the implementation abilities of different algorithms. Let's finally remark that AES implementation providing encryption only lead to lower area consumption than our architecture while SEA saves more area than the proposed AES architectures when both encryption and decryption have to be integrated.

b	n	nr	♯ Cycles	Throughput 80 MHz [Mbps]	Area [μm^2]	Gate Equivalent @ Synthesis	Gate Equivalent @ P& R	Leakage Power [μW]	Total Power 80 MHz [μW]	Total Power 100kHz [μW]
8	96	93	1428	5.38	23186	3925	4472	17.453	1376	19.238
9	108	99	1518	5.69	25294	4281	4879	18.693	1546	20.527
10	120	113	1600	6	27606	4673	5325	19.911	1598	21.923
11	132	121	1712	6.17	29742	5035	5737	20.287	1664	23.101
12	144	133	1880	6.13	31342	5406	6046	22.351	1886	24.682

Table 2. Implementation results for SEA: b-bit architecture.

	Width [bit]	Equ. Gate	Process [μm]	Max Freq [MHz]	Latency [nr. cycles]	Throughput [Mbps]	Total Power 80 MHz [μW]	Enc/Dec
Satoh et al. [16]	32	5400	0.11	131	54	311	-	yes
Feldhoffer et al. [6]	8	3600	0.35	-	1016	-	-	no
Pramstaller et al. [14]	32	8500	0.6	50	92	70	-	yes
Hämäläinen et al. [10]	8	3200	0.13	130	160	104	2400	no
Hämäläinen et al. [10]	8	3100	0.13	152	160	121	2960	no

Table 3. Implementation results for the AES.

5 Towards a minimum datapath

Since SEA has been designed for small-code software implementations using a very limited instruction set, a final approach for its implementation would be to design a minimum co-processor allowing to run this minimum set of instructions. In contexts where throughput is not an issue, this would allow to reduced the cost of the datapath beyond what can be expected for standard algorithms such as the AES. As a matter of fact, the resulting implementations would be mainly similar to the ones initially presented in [17], *e.g.* for $\text{SEA}_{96,8}$ in AVR microcontrollers (code size: 412 bytes, registers use: 10, SRAM use: 24 bytes, number of cycles: <42.000, throughput at 100 MHz: 285 char/sec or 2.3 Kbit/sec).

An exemplary minimum datapath is pictured in Figure 4. Again, we investigated its implementation results for different word lengths b, with a fixed $n_b = 6$, as summarized in Table 4. An implementation with $b = 11$ achieves a total text/key length of $n = 132$ (and would necessitate a 32 word RAM: 24 for cleartext/ciphertext and roundkey storage and 8 for working register, as in a software implementation), which is close to the text/key length of the AES. Therefore, it is worth being compared with the smallest reported implementation of the AES [7]. In the latter implementation, 28% of the resources are taken by the datapath. In terms of gate equivalent after synthesis, this is a value of 952 gates. As a matter of fact, the minimum datapath of SEA is approximately 33% smaller. With respect to the power consumption, the figures given in [7] show that a power consumption of 4.5 μW is generated at 100 KHz for the whole circuit, while our datapath has a power consumption of around 4 μW. But no special low power techniques have been applied to the implementation of SEA and the 0.13 μm technology used in this paper generates much higher leakages (around 90% of the power consumption at 100 KHz). In general, these results illustrate that optimized implementations of SEA (*e.g.* using clock gating or V_{dd} scaling [8]) could give rise to very low power consumption results. Note that, roughly estimated, the execution of a combined round/keyround using the minimum datapath of Figure 4 would take 50 clock cycles, which still improves the performances of a purely software implementation. Additionally, the ability to reduce the datapath to a minimum number of gates also makes sense from a physical security point of view, *e.g.* if resistance against side-channel or fault attacks is to be considered. Moving from these estimations to actual (possibly physically secure) designs is a scope for further research.

b	Equ. Gate @ Synthesis	Area @ P and R [μm^2]	Equ Gate @ P and R	Leakage [μW]	Total Power 100kHZ [μW]	Total Power 80MHz [μW]
8	449	4753.7	917	2.865	3.218	293.5
9	507	5049.0	974	3.083	3.421	308.8
10	563	5332.6	1028	3.246	3.636	328.6
11	620	5624.6	1085	3.499	3.878	346.1
12	677	5920.1	1142	3.704	4.128	357.6

Table 4. Implementation results for the minimum datapath.

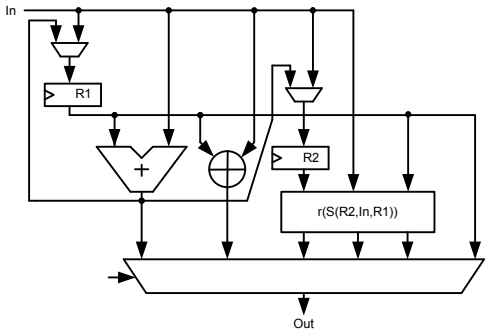

Fig. 4. Minimum datapath proposal for low cost, low power applications.

6 Conclusion

This paper details two implementations of SEA and evaluates their implementation cost using a standard cell EDA flow. The first implementation demonstrates the high scalability properties of the algorithm and illustrates its area *vs.* throughput tradeoff when compared to other block ciphers. The second implementation slightly reduces the scalability of the algorithm but provides a more realistic interface for low cost applications. It also reduces the power consumption figures. Finally, a minimum datapath is discussed with respect to its applicability to very constrained environments, *e.g.* RFIDs. If throughput constraints can be relaxed, it illustrates the good opportunities provided by SEA to decrease the implementation cost of a block cipher beyond what is possible with standard algorithms. Interestingly, SEA was not purposed for hardware implementations, but presents interesting features in this context (*e.g.* "on-the-fly" key derivation in both encryption and decryption). Improving the algorithm structure towards better hardware performances is therefore possible and is a scope for further research. It could then be compared with recent block and stream ciphers specifically designed for low cost hardware implementations, *e.g.* HIGHT [12], DESL [13], PRESENT [1], Trivium [9] or Grain [9]. Finally, SEA has interesting opportunities to trade energy for security by (possibly adaptively) reducing its (large) number of rounds. The resulting primitives would then range from secure ciphers to reasonable scrambling functions, which may be relevant in certain applications (*e.g.* sensor networks).

References

1. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe *PRESENT: An Ultra-Lightweight Block Cipher". Cryptographic Hardware and Embedded Systems*, to appear in the proceedings of CHES 2007, Vienna, Austria, September 2007.
2. `http://www.cadence.com/`
3. J. Daemen, V. Rijmen, *The Design of Rijndael*, Springer-Verlag, 2001.

4. FIPS 46-1, *"Data Encryption Standard"*, Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, January 22, 1988.
5. FIPS 197, *"Advanced Encryption Standard"*, Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, November 26, 2001.
6. M. Feldhoffer, S. Dominikus, J. Wolkerstorfer, *Strong Authentification for RFID Systems using the AES Algorithm*, in the proceedings of CHES 2004, LNCS, vol 3156, pp 347-370, Boston, Massachussets, USA, August 2004.
7. M. Feldhofer, J. Wolkerstorfer, V. Rijmen, *AES Implementation on a Grain of Sand*, in IEE Proceedings on Information Security, vol 152, issue 1, pp 13- 20, October 2005.
8. R. Gonzalez, B.M. Gordon, M.A. Horowitz, *Supply and Threshold Voltage Scaling for Low Power CMOS*, in the IEEE Journal of Solid-State Circuits, vol 32, num 8, pp 1210-1216, August 1997.
9. T. Good, W. Chelton, M. Benaissa, *Hardware Results for Selected Stream Cipher Candidates*, in the proceedings of SASC 2007, February 2007. Available for download via http://www.ecrypt.eu.org/stream/,
10. P. Hämäläinen, T. Alho, M. Hännikäinen, T.D. Hämäläinen, *Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core*, in the proceedings of DSD 2006, pp 577-583, Cavtat, Croatia, August 2006.
11. A. Hodjat, P. Schaumont, I. Verbauwhede, *Architectural Design Features of a Programmable High Throughput AES Coprocessor*, in the proceedings of ITCC 2004, vol 2, pp 498-502, Las Vegas, Nevada, USA, April 2004.
12. D. Hong *et al.*, *HIGHT: A New Block Cipher Suitable for Low-Resource Device*, in the proceedings of CHES 2006, Lecture Notes in Computer Science, vol 4249, pp 46-59, Yokohama, Japan, October 2006.
13. G. Leander, C. Paar, A. Poschmann, K. Schramm, *A Family of Lightweight Block Ciphers Based on DES Suited for RFID Applications*, to appear in the proceesings of FSE 2007, Luxembourg, March 2007.
14. N. Pramstaller, S. Mangard, S. Dominikus, J. Wolkerstorfer, *Efficient AES Implementations on ASICs and FPGAs*, in the proceedings of the 4^{th} Conference on the Advanced Encryption Standard - AES 2004, pp 98-112, Bonn, Germany, may 2004.
15. F. Macé, F.-X. Standaert, J.-J. Quisquater, *FPGA Implementation(s) of a Scalable Encryption Algorithm*, to appear in IEEE Transactions on VLSI.
16. A. Satoh, S. Morioka, K. Takano, S. Munetoh, *A Compact Rijndael Hardware Architecture with S-Box Optimization*, in the proceedings ASIACRYPT 2001, LNCS, vol 2248, pp 239-254, Gold Coast, Australia, December 2001.
17. F.-X. Standaert, G. Piret, N. Gershenfeld, J.-J. Quisquater,*SEA: A Scalable Encryption Algorithm for Small Embedded Applications*, in the Proceedings of CARDIS 2006, LNCS, vol 3928, pp 222-236, Tarragona, Spain, April 2006.
18. F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, *FPGA Implementations of the ICEBERG Block Cipher*, in the proceedings of ITCC 2005, vol 1, pp 556-561, Las Vegas, Nevada, April 2005.
19. `http://www.synopsys.com/`
20. `http://www.umc.com/`
21. D.J. Wheeler, R. Needham, *TEA, a Tiny Encryption Algorithm*, in the proceedings of FSE 1994, LNCS, vol 1008, pp 363-366, Leuven, Belgium, December 1994.
22. G. Yuval, *Reinventing the Travois: Encryption/MAC in 30 ROM Bytes*, in the proceedings of FSE 1997, LNCS, vol 1267, pp 205-209, Haifa, Israel, January 1997.