Power-efficient ASIC Synthesis of Cryptographic Sboxes

Guido Bertoni
STMicroelectronics
Agrate Brianza, Italy
guido.bertoni@st.com

Marco Macchetti
Politecnico di Milano
Milan, Italy
macchett@elet.polimi.it

Luca Negri
Politecnico di Milano
Milan, Italy
lnegri@elet.polimi.it

ABSTRACT

In this paper we present a novel methodology that can be used to design efficient hardware structures for a certain class of combinatorial functions. The methodology is primarily intended to achieve low-power synthesis of non-linear one-to-one functions on ASIC technology libraries and fits well for the synthesis of small cryptographic substitution box (Sbox) functional components; the latter are found in most secret key cryptographic algorithms, and usually represent their most relevant part in terms of required computational power. We also describe an extension that allows us to apply the method to general vectorial Boolean functions.

Categories and Subject Descriptors

B.6.1 [**Design styles**]: Combinational logic

General Terms

Performance, Design, Security.

Keywords

Low power logic, cryptography, S-box implementation.

1. INTRODUCTION

The problem of synthesizing Boolean functions with high non-linearity has received attention in recent years, due to the deployment of dedicated cryptographic engines and to the raised interest in symmetric key cryptography that followed the AES contest [1]. The functions which are used as substitution boxes in block ciphers are relatively small functions, where the number of inputs rarely exceeds 10; they must also have particular properties of maximum non-linearity and minimum input-output correlation, see for instance the design rationale of Rijndael [2].

Such functions can or cannot have a clear mathematical structure, independently of the non-linearity properties; for instance, the inverse function in the finite field GF(2^8) was chosen as the basic non-linear core of the AES algorithm. However, it is still possible to generate random functions with the desired properties that do not have a straightforward mathematical description; the reader may refer to [3] and [4] for the description of an algorithm to generate such functions on a bit-by-bit fashion.

When synthesizing these functions on a given technology library, one must typically aim at minimizing chip area and latency, and usually power consumption is a secondary metric of cost. Nonetheless, power consumption is a problem of increasing relevance, especially if the dedicated hardware is going to be included on a smart card, in a mobile system or in general in battery-powered systems. There are several known methodologies to design low power structures at system level, architectural level and register transfer level (RTL), but few contributions refer to the logic level, especially when combinatorial-only circuits are considered; some results are reported in [5] and [6] where two-level and multilevel logic optimizations are discussed.

In this paper, we propose a methodology that can be used to synthesize cryptographic Sboxes (with input size ranging between 1 and 16 lines) on custom silicon (ASIC) libraries with energy-efficiency as a primary goal. Moreover, we extend the method to cover the case of more generic vectorial Boolean functions, showing its general applicability.

This paper is organized as follows: Section 2 describes the theoretical basis for our design approach; in Section 3 we give results from synthesis on a real technology library; in Section 4 we discuss an extension that covers the case of general vectorial Boolean functions; Section 5 concludes the paper.

2. THE PROPOSED METHODOLOGY

For our discussion, we will initially consider permutation functions, where obviously the number of outputs equals that of inputs; in Section 4 the more general case of non bijective functions and of N to M functions will be addressed.

Generally speaking, a permutation function is defined on a set of elements S and constitutes an isomorphism over this set. Different representations are possible for the elements of S and the choice of the representation can have a great impact on the hardware complexity of the permutation function; for instance, if we choose a simple binary representation, we need to synthesize a non-linear random-like $N \times N$ vectorial Boolean function. Instead, we decide to choose a *one-hot* representation of the elements of S, like in [7]; this means that number of inputs and outputs will grow and we have to synthesize a $2^N \times 2^N$ function; on the other hand we can view the isomorphism over S as a simple rearrangement of the bits which represent each element: the one and only active line will change place inside the string of 2^N bits and so the function virtually consumes zero power. We need, however, to interface the hardware block with the other circuitry, where typically the simple binary representation is used. This is achieved by means of a decoder and of an encoder; the whole architecture is depicted in Fig. 1.

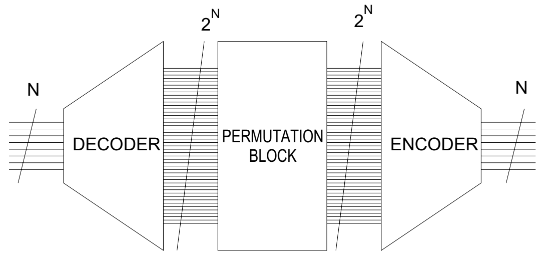

Figure 1: Diagram of the proposed architecture

2.1 The cost function

Traditional synthesis methods, both exact and heuristic, aim at minimizing area, sometimes adding low power consumption as a secondary objective. We approach the problem from a completely different perspective, by setting low dynamic power consumption as the primary goal, and by releasing any constraint on area. What we intend to reduce is the average energy consumption associated with transitions on the primary inputs of the circuit. This is proportional, for each gate in the circuit, to the switching activity (probability) and to fan-out (due to increasing capacitance). Eventually, the cost function to be minimized is

$$\sum_{all\ gates} (P_{sw,out} F) \qquad (1)$$

with the initial assumption that all inputs have a .5 probability of being 1 at a certain time, and that they are completely uncorrelated one with the other and over time. This is likely the case for Sbox circuits used inside a cryptographic module, since for almost all round iterations of a typical block cipher the correlation between successive values is very low. This assumption can be used to infer the output probability for first level gates (using the gate's truth table) which is then the input one for the following gates.

2.2 The decoder structure

A considerable part of our work has to deal with finding the optimal structure of the decoder, with regard to energy-efficiency. Some related work was presented in [8], but at a lower level, tuning the size of the CMOS gates. We performed our optimization at logic gate level, trying to break up the decoder into a number of levels that perform partial decoding; a possible multi-level structure is depicted in Fig. 2a for $N = 7$.

The idea is that the first level performs a partial decoding of S subsets of the input lines (components labeled *Partial Decoder* (PD) in Fig. 2), yielding S groups of lines, while the following levels take care of combining, by means of AND gates, the output lines from different groups of the previous level in all possible ways. This is accomplished with the components labeled *Cross Decoder* (CD) in Fig. 2; in this

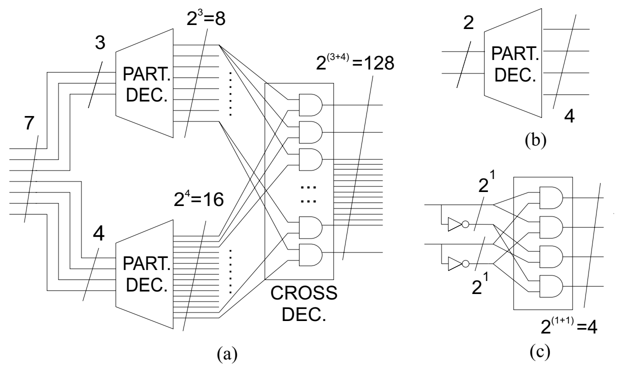

Figure 2: Example of multi-level decoder

example, the CD ANDs each output line of the upper PD with each output line of the lower one, yielding the $2^7 = 128$ completely decoded lines. In general, there can be an arbitrary number of levels, each containing more than one CD, each connecting an arbitrary number of PDs or CDs from the previous level.

Actually, the distinction between PDs and CDs can be eliminated, if we consider that a PD for L input lines (by all means, a normal L-bit decoder) can be seen as a CD crossing L 2-line groups, being each of these groups the union of an input line and its complement (this is exemplified for $L = 2$ in Fig. 2b and Fig. 2c): this cuts down the whole structure to a tree of Cross Decoders. Considering any two consecutive

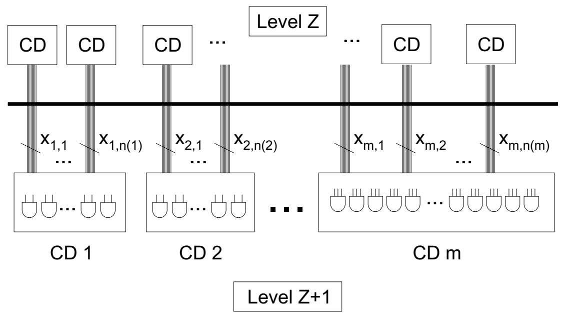

Figure 3: Calculation of the cost (average energy) for a generic level in the decoder

levels Z and $Z + 1$ in this tree of CDs (see Fig. 3), we can define the following quantities for level $Z + 1$:

- $x_{k,t}$ is the size of the t^{th} group of lines that enters the k^{th} CD. Actually, it does not represent the number of lines in the group, but rather the $\log_2$ of that number[1] (e.g. $x = 3$ for a group of 8 lines).
- $n(k)$ is the number of groups that enter the k^{th} CD.
- y_k is the sum of the sizes of the groups entering the k^{th} CD, and also represents the $\log_2$ of the number of output lines for the CD. More precisely, $y_k = \sum_{t=1}^{n(k)} x_{k,t}$
- m is the number of CDs at level $Z + 1$.

[1]This will hereinafter apply to all references to the size of a group of lines, if not elsewhere specified.

For level Z, the cost function in (1) becomes

$$\sum_{k=1}^{m}\sum_{t=1}^{n(k)}\left[2^{(y_k-x_{k,t})}\cdot 2^{x_{k,t}}\cdot 2\cdot\frac{1}{2^{x_{k,t}}}\cdot(1-\frac{1}{2^{x_{k,t}}})\right] \qquad (2)$$

which is the sum over all groups of lines entering the CDs at level $Z+1$ of the product of 5 factors. Among these, $2^{x_{k,t}}$ is the number of lines in the group and $2^{(y_k-x_{k,t})}$ is the fan-out of these lines (each line in the group is crossed with all possible combinations of $n(k)-1$ lines coming from the other groups entering the same CD); the remaining factors represent the switching probability of the line. In fact, for any AND gate with N inputs, the switching probability $P_{sw,out}$ of its output line can be written as

$$P_{sw,out} = 2P_{out,1}(1-P_{out,1}) \qquad (3)$$

where $P_{out,1}$ is the probability of the output being 1. Now, since we are working with uniformly distributed and uncorrelated inputs, the probability of being 1 for a line in a group of X lines that results from a partial decoding of a set of primary input lines is always $1/X$. Moreover, this is also true for *all* AND gates in *all* CDs, since the n input lines of an n-input AND gate result from partial decodings of different sets of primary input lines, and we assumed that all input lines are not correlated with each-other. Hence, by substituting $1/2^{x_{k,t}}$ in (3), we obtain the last three factors of (2).

We wrote a C program to find the structure of the decoder that gives the absolute minimum of (1), intended as the sum of (2) over all levels in the tree. The program exhaustively evaluates the cost of all possible trees of CDs, exploiting the large number of degrees of freedom of the structure, which can be roughly grouped into: the number of levels in the decoder; at each level, how many partial decoding groups to allocate; at each level, how to connect the groups from the previous level to the groups in this level.

The program was useful up to the case of $N = 16$ input lines, after which execution time became unacceptable due to complexity. However, this produced some very interesting results: the best solution (in terms of energy cost) is surprisingly symmetric and can be obtained by means of a simple algorithm. Starting from the final level of the decoder (one group with 2^N lines, coming out of one CD), and working backwards toward the inputs, the following rules should be applied, level-by-level:

- The generic level Z is built as follows: for each CD at level $Z+1$, two CDs are allocated at level Z, and connected to the inputs of the CD at level $Z+1$. Let S be the size of the level $Z+1$ CD (the size of its output); the size of the two level Z CDs must be $\lfloor S/2\rfloor$ and $\lceil S/2\rceil$. Note that if S is even, the two CDs will both have size $S/2$.
- This must be applied all the way back to the inputs, until a level is allocated at which the biggest CDs have size 2. In fact, the inputs to a size-two CD will be, according to the algorithm, two groups of size one; these are practically two input lines of the circuit, each taken with its complement.

The thus built structure will have $\lceil\log_2 N\rceil$ levels. Fig. 4 shows the resulting decoder structure for $N = 8$ (a symmetric case) and $N = 11$. From what we have learned using our program, we can conclude that:

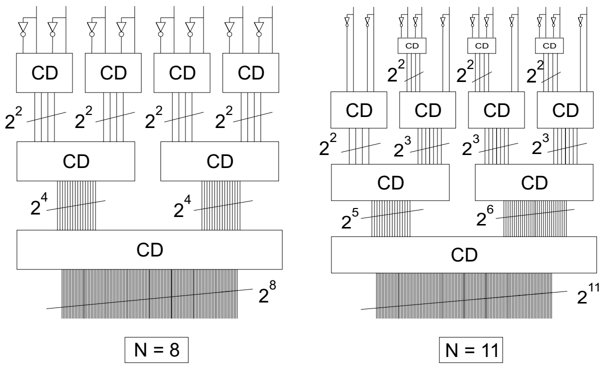

Figure 4: Some optimal decoder structures

- For $N \leq 16$ the cost-optimal decoder structure, *among those than can be built with a tree of CDs (and this also includes the classical one-level AND-array decoder)*, is the one obtained with the above algorithm. Although we do not include a theoretical proof for it, this has been proved by exhaustive search.
- For $N > 16$ we have no reason to believe that the same algorithm cannot be used to build the optimal structure, although we have no proof for it. However, the problem of the synthesis of unstructured permutation functions for $N > 16$ becomes a rather theoretical one, as most applications (not only cryptographic Sboxes) employ smaller functions.

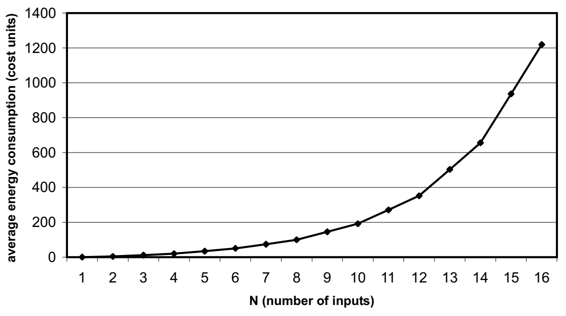

Figure 5: Decoder power consumption

The cost of the optimal decoder for $N \leq 16$ is plotted in Fig. 5. The energy consumption is here reported in cost units, which represent the energy used by one gate with fan-out 1 when switching between 0 and 1 or vice versa. This graph has the only purpose of illustrating how the decoder energy usage grows with N, in relative terms. For more precise simulation results, given in terms of power consumption and also including the encoder power, see Section 3. In Section 2.4, a subset of the values of the curve in Fig. 5 is analytically derived to analyze its asymptotic behavior.

2.3 The encoder structure

The function of the encoder is to perform a positional to binary encoding of the 2^N permuted lines over the N output lines. As N grows, it is impossible to implement this component with a single level of OR gates: there would be too many inputs per gate. Thus, each OR gate is split into a certain number of cascaded OR gates, of smaller size: this yields an encoder that is made up of N OR trees. However, these trees are conceptually very different from the decoder tree, since the number of gates decreases as we move from the inputs to the outputs (this implies a very low average fan-out for the gates) and the OR gates can actually be *reused* among different encoding trees.

For these reasons, there is less scope for energy-efficiency improvement of the encoding side, and this can be anyway obtained with traditional area-based methods, which will actually try to reuse the ORs as much as possible. Therefore, our work on the encoder was not as extensive as on the decoder, and we adopted the structure provided by the synthesis tool (Synopsys Desing Compiler).

2.4 Analytical derivation of decoder cost

The total cost of the optimal decoder tree can be easily captured in a single expression, by summing the result of (2) for each level in the tree, in highly symmetric cases. This happens when N is a power of 2, which implies the two groups of input lines to all Cross Decoders at a specific level have the same size. The total cost is obtained by simplifying (2) according to the symmetry and then summing it up over the $\log_2 N$ levels of the structure; after some intermediate steps, the final expression of the cost is:

$$4N \sum_{K=1}^{\log_2 N} (2^{(-K)}(2^{(2^{(K-1)})} - 1)) \tag{4}$$

As expected, the value of this expression for $N = 2, 4, 8, 16$ equals that of the graph in Fig. 5. For the other (asymmetric) cases, no simple expression can be derived.

Trying to expand the sum in (4) for different values of N, it is easy to recognize that the last term of the sum is the dominant one as N grows. Thus, for $N \to \infty$, (4) is asymptotic to:

$$4N \cdot 2^{-\log_2 N} \cdot 2^{2^{\log_2 N - 1}} = 4 \cdot 2^{\frac{N}{2}} \tag{5}$$

Hence, even though the total area of the decoder grows with 2^N, its power cost only grows with $2^{\frac{N}{2}}$. This is possible if we think that not all the gates in the decoder are active at the same time. In fact, when an arbitrary number of inputs switch, only a certain number of paths through the decoder are activated (i.e. have some switching activity) and only two of these paths actually reach the final level of the decoder (that is, by definition of decoder, only two outputs of the decoder switch).

3. SYNTHESIS RESULTS

We have first applied the proposed methodology to the AES algorithm: the Rijndael Sbox has been synthesized with Synopsys synthesis tools version 2002.05 on a Solaris platform; the target technology library is the STMicroelectronics HCMOS8 library, featuring 0.18 μm process and 1.8V core voltage. All post-synthesis simulations have been done at the logic gate level with 1 ps resolution, using the VHDL simulator by Synopsys and automatically annotating the switching activity information.

For reasons of privacy all power consumption figures are normalized, in order to not reveal the absolute values.

The input patterns were chosen according to the fact that each Sbox component is part of a cryptographic block cipher; since data redundancy in the input bits is soon destroyed by the iterations of the algorithm, we can assume that the real inputs to these components are random string of bits, coherently with the discussion of Section 2. We have implemented the Mother of All random number generator [9] in VHDL to generate the different input patterns.

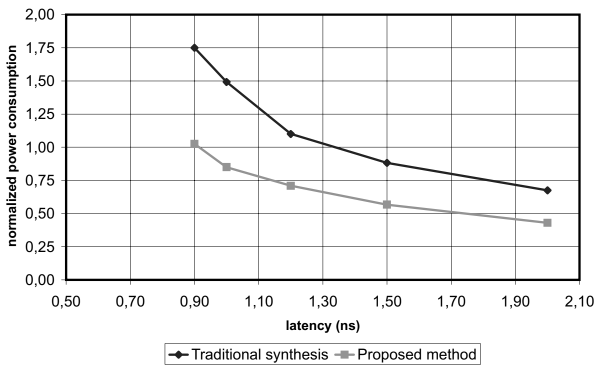

Figure 6: AES Sbox power cons. vs. latency

Fig. 6 shows synthesis results: there is a clear advantage in using the proposed method over the built-in heuristics of the Synopsys tools in terms of reduced power consumption for a given latency. The results labeled *traditional synthesis* were obtained starting from a Look-Up-Table like description, as discussed in [10]. Note that an acceptable speed for this component is around 1 GHz, and at this frequency the proposed circuit structure consumes only 57% of the power consumed by the circuit obtained from a trivial table-like description. Area occupation, on the contrary, is increased by 11%.

Other implementations are possible for the Rijndael Sbox, such as the one based on composite fields discussed in [11] and [12], but latency-power product is worse in that case, as latency is increased by a factor of 3. A low-power variant of the composite fields architecture was proposed in [13], but the same consideration on the latency-power product applies. Note also that the composite fields approach is only applicable to a small fraction of Sboxes.

We have also tested the proposed methodology versus the classical approach in the case of randomly-generated 8×8 permutation functions. Results from synthesis of 10 such functions on the aforementioned technology library are reported in Fig. 7. Note that here the power consumption of the proposed circuit architectures is always the same; this is obvious, because the logic structure is the same and only the permutation of the internal lines is different for each case. Results from the classical mapping approach show small differences and are always more power consuming. In these mappings the latency is constrained to 1 ns.

4. THE CASE OF GENERAL FUNCTIONS

The majority of vectorial Boolean functions are not expressible as permutations. From our perspective we can introduce a little modification in the methodology to be able to synthesize these functions; this modification will not affect the decoder, so the above considerations on the optimal structure (which depends only on the number of inputs N) are still valid.

A non-bijective Boolean function can have N inputs and

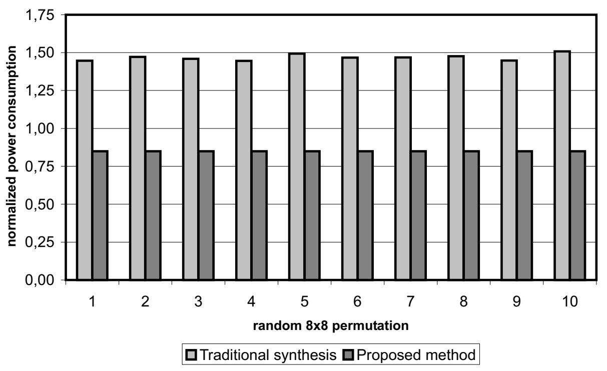

Figure 7: Power cons. of random permutations

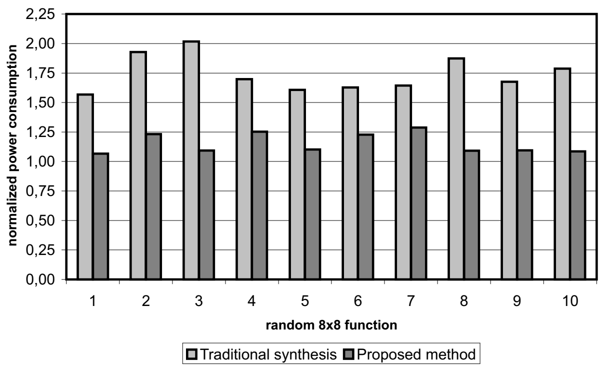

Figure 8: Power cons. of random functions

M outputs ($N \times M$) and both the cases $N = M$ or $N \neq M$ are possible. Since, as already explained in Section 2, the area is exponential with the number of inputs, we pose a limit of 16 to the number of input bits. However, if we keep the number of inputs fixed, area grows linearly with the number of outputs, because the encoder is made up by M OR trees and the worst size of the OR trees only depends on the number of inputs (we can state that, on average, each output bit is the logical OR of half of the intermediate lines). For this reason M can be left unconstrained.

The permutation block is substituted with a new component whose purpose is to implement the actual Boolean function and which has 2^N inputs (the decoded lines from the decoder) and 2^M outputs (the lines to be fed into the encoder). If the function is not a permutation, then it is possible that for some input lines the outputs will be exactly the same: then we simply OR the two internal lines inside of the new component. If the function is $N \times M$ with $M \geq N$ this means that there are some output lines from this component which are not used at all, because there are some output configurations that simply never happen. These lines are left unconnected.

To test the methodology in this new case, we generated 10 random 8×8 Boolean functions and the synthesis results are given in Fig. 8. As we can infer from the graph, the proposed method is still better than the Synopsys Design Compiler built-in heuristics in all cases; however power consumption is increased with regards to the permutations case, it is not constant anymore, and apparently there is no correlation between the fluctuations in the classical series and those in the proposed method series.

We have also applied the method to the eight DES Sboxes, and the average power consumption is decreased to only 57% with respect to the classical mapping. Moreover, as a second result, we have that power consumption is the same for every DES Sbox, which is a good way to minimize hot regions in the operating chip. Area occupation for each DES Sbox is about 46% greater than the original implementation.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented a synthesis methodology that can be used to map power-efficient dedicated hardware for cryptographic Sboxes. The method is shown to give good results for several cryptographic Sboxes and is extended to cover the case of general vectorial Boolean functions.

Future work potentially includes the extension of the method to $N \times M$ random Boolean functions with $N > 16$; for this purpose one could try to decompose the bigger functions into smaller cores with no more than 16 inputs and use the presented methodology to map these cores.

6. ADDITIONAL AUTHORS

This paper is co-authored by Pasqualina Fragneto, STMicroelectronics, Italy, pasqualina.fragneto@st.com.

7. REFERENCES

[1] AES homepage, available at http://csrc.nist.gov/aes.

[2] J. Daemen and V. Rijmen. AES proposal: Rijndael. NIST AES Proposal, June 1998.

[3] C. Adams and S. Tavares. Structured design of cryptographically good S-boxes. *Journal of Cryptology*, 3(1):27–41, 1990.

[4] L. O'Connor. An analysis of a class of algorithms for S-box construction. *Journal of Cryptology*, 7(3):133–151, 1994.

[5] S. Iman and M. Pedram. Multi-level network optimization for low power. In *ICCAD Proceedings*, pages 372–377, Nov. 1994.

[6] S. Iman and M. Pedram. Two-level logic minimization for low power. In *ICCAD Proceedings*, pages 433–439, Nov. 1995.

[7] L. Xiao and H. M. Heys. Hardware design and analysis of block cipher components. In *Proc. 5th ICISC*, 2002.

[8] B. S. Amrutur and M. A. Horowitz. Fast low-power decoders for rams. *IEEE Journal of Solid State Circuits*, 36(10):1506–1515, oct 2001.

[9] G. Marsaglia. Mother of all pseudo random number generator, http://www.agner.org/random/mother/.

[10] N. Sklavos and O. Koufopavlou. Architectures and vlsi implementations of the aes-proposal rijndael. *IEEE Trans. on Computers*, 51(12):1454–1459, dec 2002.

[11] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In *Proc. CHES 2001*, pages 171–184, 2001.

[12] M. Macchetti and G. Bertoni. Hardware implementation of the rijndael sbox: a case study. *ST Journal of System Research*, (0):84–91, jul 2003.

[13] S. Morioka and A. Satoh. An optimized s-box circuit architecture for low power aes design. In *Proc. CHES 2002*, pages 172–186, 2003.