Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-Box

Stefan Tillich, Martin Feldhofer, and Johann Großschädl

Graz University of Technology,
Institute for Applied Information Processing and Communications,
Inffeldgasse 16a, A–8010 Graz, Austria
{stillich,mfeldhof,jgrosz}@iaik.tugraz.at

Abstract. Cryptographic substitution boxes (S-boxes) are an integral part of modern block ciphers like the Advanced Encryption Standard (AES). There exists a rich literature devoted to the efficient implementation of cryptographic S-boxes, whereby hardware designs for FPGAs and standard cells received particular attention. In this paper we present a comprehensive study of different standard-cell implementations of the AES S-box with respect to timing (i.e. critical path), silicon area, power consumption, and combinations of these cost metrics. We examined implementations which exploit the mathematical properties of the AES S-box, constructions based on hardware look-up tables, and dedicated low-power solutions. Our results show that the timing, area, and power properties of the different S-box realizations can vary by more than an order of magnitude. In terms of area and area-delay product, the best choice are implementations which calculate the S-box output. On the other hand, the hardware look-up solutions are characterized by the shortest critical path. The dedicated low-power implementations do not only reduce power consumption by a large degree, but they also show good timing properties and offer the best power-delay and power-area product, respectively.

1 Introduction

The Internet of the 21st century will consist of billions of non-traditional computing systems like cell phones, PDAs, sensor nodes, and other mobile devices ("gadgets") with wireless networking capability. Wireless networking, along with the fact that many of these devices (e.g. sensor nodes) are easily accessible, have raised a number of security concerns. Sophisticated security protocols, in combination with well-established cryptographic primitives, can ensure privacy and integrity of communication over insecure networks. Consequently, there is an increasing demand to implement cryptographic algorithms on resource-limited embedded devices like sensor nodes, smart cards, or mobile phones. Even some extremely constrained systems like radio frequency identification (RFID) tags may require cryptographic processing.

S. Vassiliadis et al. (Eds.): SAMOS 2006, LNCS 4017, pp. 457–466, 2006.

The Advanced Encryption Standard (AES), which has been announced by the NIST in 2001, defines one of the most important symmetric ciphers for the next decades [11]. The AES algorithm is a variant of the Rijndael cipher [4] and can be implemented efficiently in both software and hardware. Common AES hardware implementations take the form of cryptographic ASICs and coprocessors. In addition, hardware/software co-design techniques like instruction set extensions have also been proposed in recent years [14]. Due to the high performance of modern microprocessors, AES software implementations can reach throughput rates which are sufficient for most applications. Therefore, hardware implementations of the AES algorithm are mainly important for high-end server systems with extreme performance requirements and for low-power and low-energy environments.

Most of the published AES hardware designs focus on high speed and high throughput for implementation on FPGAs [3,12]. In addition, some ASIC implementations have been reported in the recent literature. For instance, Hodjat et al. developed a 3.84 Gbits/s coprocessor based on a 0.18-μm CMOS technology with intended usage in high-end server applications [6]. Another high-speed implementation can be found in [16]. A completely different approach is necessary for low-power and low-energy devices. Feldhofer et al. published an AES implementation suited for passively-powered devices like RFID tags [5]. It is the smallest and most power-saving implementation known so far.

Modern symmetric ciphers require non-linear functions in order to defend against linear cryptanalysis. Substitution is a popular function for introducing non-linearity. A substitution function is commonly referred to as S-box and can be defined on basis of arithmetic operations or as an arbitrary mapping. Different cipher algorithms also use different numbers of S-boxes, e.g. DES uses eight S-boxes which map six to four bits, while AES uses a single S-box which is a bijective mapping from eight to eight bits.

The AES algorithm makes use of its S-box in the SubBytes round transformation as well as in the key expansion. From a mathematical point of view, the AES S-box is defined as an *inversion* in the finite field GF(2^8) with a specific reduction polynomial [7], followed by an affine transformation [4]. The inverse S-box, which is required for the InvSubBytes round transformation for decryption, is simply the inverse of the affine transformation, followed by an inversion in GF(2^8). The finite field inversion is the only non-linear operation of the AES algorithm. Since there exist many design options for the S-box in hardware, it is challenging to find an optimal implementation for a particular purpose. On the one hand, the main criterion for high-speed implementations is a short critical path, which allows to reach high clock frequencies[1]. On the other hand, S-box implementations for embedded devices call for small silicon area and low power consumption. In this paper we analyze and compare silicon area, critical path delay, and power consumption characteristics of the most common standard-cell implementations of the AES S-box. We hope that our results will help system designers to find the optimal S-box for their application.

[1] The S-box normally lies on the critical path of AES hardware implementations.

The remainder of this paper is organized as follows. In Section 2 the different implementation strategies for the AES S-box are discussed. In Section 3 we describe the particular implementations which we have examined. In Section 4 we discuss our experimental results and we conclude in Section 5.

2 Implementation Strategies for the AES S-Box

The AES is based on rounds consisting of linear and non-linear transformations [4]. All transformations operate on a two-dimensional array of 4×4 bytes (128 bits), called the *State*. One of the strengths of the AES algorithm is its simplicity, which facilitates the implementation on a wide range of different platforms under different constraints. Various hardware designs have been reported in the literature, whereby the efficient implementation of the S-box received particular attention [1,2,8,9,10,13,15].

The SubBytes transformation substitutes all 16 bytes of the State independently using the S-box. Furthermore, the S-box is also used in the AES key expansion. In software, the S-box is typically realized in form of a look-up table since the inversion in the finite field $\mathrm{GF}(2^8)$ can not be calculated efficiently on general-purpose processors. In hardware, on the other hand, the implementation of the S-box is directed by the desired trade-off between area, delay, and power consumption. The most obvious implementation approach for the S-box takes the form of hardware look-up tables. However, since encryption and decryption require different tables, and each table contains 2048 bits, the overall hardware cost of this approach is relatively high.

An implementation option related to standard cells is the usage of ROM compilers to produce hardware macros. For the standard-cell technology that we have used, a ROM macro of sufficient size would require a considerable amount of silicon area. The critical path delay is very similar to a hardware look-up approach, but the power consumption of generated ROMs is about 2–3 orders of magnitude higher. Therefore, we do not consider the implementation of the S-box as ROM macro in this paper

More advanced approaches calculate the S-box function in hardware using its arithmetic properties. The focus of such implementations is the efficient realization of the inversion in $\mathrm{GF}(2^8)$, which can be achieved by decomposing the finite field into the sub-fields $\mathrm{GF}(2^4)$ and $\mathrm{GF}(2^2)$. An inversion in a finite field of characteristic 2 can be carried out in different ways, depending on the basis which is used to represent the field elements [7]. The two most common types of bases for $\mathrm{GF}(2^m)$ are the *polynomial basis* and the *normal basis*. A polynomial basis is a basis of the form $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ where α is a root of an irreducible polynomial $p(t)$ of degree m with coefficients from $\mathrm{GF}(2)$. On the other hand, a normal basis can be found by selecting a field element $\beta \in \mathrm{GF}(2^m)$ such that the elements of the set $\{\beta, \beta^2, \beta^4, \ldots, \beta^{2^{m-1}}\}$ are linearly independent.

A third approach for implementing the AES S-box was proposed by Bertoni et al. in [1]. By using an intermediate one-hot encoding of the input, arbitrary logic functions (including cryptographic S-boxes) can be realized with minimal

power consumption. The main drawback of this approach is that it results in relatively large silicon area.

3 Examined Implementations

All surveyed AES S-box implementations can perform forward and inverse byte substitution for encryption and decryption respectively. We have implemented all analyzed solutions either by ourselves or have obtained them from the authors of the respective publications[2]. All implementations just consist of combinatorial logic, i.e. no pipelining stages have been inserted.

The simplest design in our comparison is a straight-forward implementation of a hardware look-up table. The synthesizer transforms the behavioral description of the look-up table into a mass of unstructured standard cells. This approach will be denoted as **hw-lut**. A modification of that approach is to use sub-tables in order to minimize switching activity in the look-up tables to reduce power consumption. We have examined such solutions with sub-tables of size 16, 32, 64, 128, and 256 bytes, but in this paper we only cite results for size 16 (**sub16-lut**).

Implementations which calculate the S-box transformation in hardware have been first proposed by Wolkerstorfer et al. [15] and Satoh et al. [13]. The first approach decomposes the elements of $GF(2^8)$ into polynomials over the sub-field $GF(2^4)$ and performs inversion there. Our implementation of this solution is denoted as **wolkerstorfer**. Satoh's solution decomposes the field elements further into polynomials over the sub-field $GF(2^2)$, where inversion is a trivial swap of the lower and higher bit of the representation. This implementation is called **satoh** in the following. Both of these approaches represent the field elements by using a polynomial basis. Canright improved the calculation of the S-box by switching the representation to a normal basis [2]. Like in Satoh's solution, the finite field element's representation is mapped to a polynomial over the sub-field $GF(2^2)$. This approach will be denoted as **canright**.

A compromise between hardware look-up and calculation has also been examined. In this implementation (denoted as **hybrid-lut**) only the inversion in $GF(2^8)$ is implemented as look-up table. As this inversion is used for both encryption and decryption, the size of the look-up table is halved in relation to the **hw-lut** approach. The affine and inverse affine transformations are done in logic just as in the calculating implementations of **wolkerstorfer**, **satoh**, and **canright**.

The low-power approach of Bertoni et al. [1] uses a decode stage to represent the 8 bits of the input byte and the control bit which selects encryption or decryption into a one-hot encoding on 512 lines. The substitution itself is just a simple rewiring of these lines. As two of the lines always map to the same 8-bit result (one for encryption and one for decryption), these line pairs are combined with a logical OR to result in a one-hot encoding of the result on 256

[2] We would like to thank Johannes Wolkerstorfer and David Canright for providing their HDL source code.

lines. A subsequent encoder stage transforms this result back to an 8-bit binary value. Due to this decoder-permute-encoder structure, there is only very little signal activity within the circuit at a change of the input, resulting in low power consumption. Note that the structure of Bertoni's approach makes it in principle easily possible to introduce pipeline stages. However, it may be necessary to add a large number of additional flip-flops when the pipeline stage is placed between the decoder and encoder, i.e. on the one-hot encoded signal lines. These flip-flops will increase power consumption considerably and can easily mitigate the low-power advantages of this solution. For design scenarios where both power consumption and silicon area are of minor importance, Bertoni's approach can offer the best opportunity for reaching very high clock frequencies.

We have tested two implementations of Bertoni's approach: One implementation used a decoder with four stages as proposed in the original publication for minimal power consumption (**bertoni**). The second realization, denoted as **bertoni-2stg**, used a different decoder structure with only two stages in order to reduce the critical path of the circuit.

In the remainder of this paper we will refer to **wolkerstorfer**, **satoh**, and **canright** as *calculating implementations*. We will denote **hw-lut** and **hybrid-lut** as *look-up implementations*, and **sub16-lut**, **bertoni**, and **bertoni-2stg** as *low-power implementations*.

4 Experimental Results

For our experiments we have used a 0.35 μm CMOS standard cell library from austriamicrosystems. We synthesized all implementations described in Section 3 using the Physically Knowledgeable Synthesis (PKS) tool from Cadence. For each implementation numerous synthesis runs with different target values for maximal critical path delay have been performed. Each synthesis run provided the actual critical path delay, the area of the synthesized design, and the total power consumption. However, for our evaluation we only used results where the timing constraints were met by the synthesizer.

Most of the following figures use a logarithmic scale on the Y-axis, as otherwise it would not be possible to display the results for all implementations within a single figure. Each legend cites the functions in the same top-down order as they are contained in the respective figure.

In Figure 1 the area of synthesized designs with a specific critical path delay are shown. The area is given in gate equivalents (GE), calculated as total area divided by the size of a 2-input NAND with the lowest driving strength, which is the NAND20 standard cell of the library we used.

Amongst the three calculating implementations (at the bottom of the figure), **canright** is clearly the best. It has the smallest size, but suffers from a longer critical path than the hardware look-up implementations and the low-power implementations. The calculating implementations are smaller than the other two approaches because they make use of the algebraic structure of the S-box to implement the substitution. On the other hand, this structure has a relatively

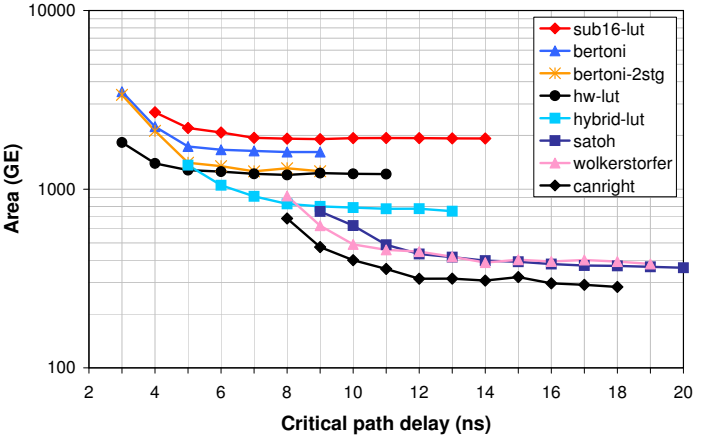

Fig. 1. Area vs. critical path delay

long critical path. The shortest critical path can be achieved with **hw-lut** but its size is 2–3 times that of **canright**. Look-up implementations neglect the algebraic structure of the S-box and just aim at a straightforward realization of the boolean equations constituted by the input-output relation. Hence, the synthesizer has a much higher degree of freedom for optimizing the circuit, which allows for a much shorter critical path at the expense of silicon area. Also good speed can be achieved with **bertoni** and **bertoni-2stg**, but they require even more silicon area.

The low-power implementations also neglect the arithmetic properties of the substitution and just implement the boolean equations of the input-output relation. However, they use a specific structure (decode-permute-encode) to reduce signal activity. Although the critical path is similarly short as for look-up implementations, the one-hot encoding requires more silicon area than the look-up implementations. The **sub16-lut** approach also has a significant area overhead introduced by the address decoding of the sub-tables, which makes it the solution requiring the most silicon area. Moreover, the address decoding logic leads to a longer critical path. As expected, the compromise between hardware look-up and calculation (**hybrid-lut**) lies roughly between **hw-lut** and the calculating implementations in regard of both critical path delay and area.

Figure 2 shows the total power consumption in relation to the critical path delay. All power figures have been normalized to the power consumption of **hw-lut** for 5.0 ns delay. The low-power implementations based on the approach by Bertoni (**bertoni**, **bertoni-2stg**) show the lowest power consumption. The original implementation **bertoni** has the best characteristics while the modified version **bertoni-2stg** is slightly worse. Bertoni's approach is solely directed towards low power consumption with a minimal level of signal activity in the circuit. Therefore, it is better than the **sub16-lut** approach, which tries to improve a straightforward look-up table implementation (**hw-lut**) with low-power

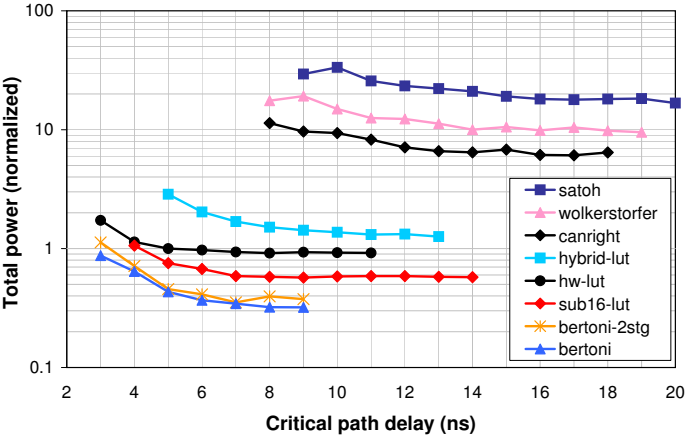

Fig. 2. Total power consumption vs. critical path delay

measures. The **sub16-lut** implementation requires almost twice as much power as **bertoni**, while **hw-lut** consumes 2–3 times more power. The **hybrid-lut** approach requires even considerable more power than **hw-lut**.

The power consumption of the calculating implementations is much higher than that of the low-power and look-up versions. The algebraic evaluation of the S-box function in calculating implementations requires re-computation of all intermediate values even if only a few number of input bits toggle. This behavior entails very high signal activity. In look-up implementations a change of a few input bits affects the calculation of all output bits separately. As some output bits will be left unchanged, the signal activity within this particular path is low and hence limits the power consumption. The most power-efficient variant among the calculating implementations is **canright**, which has 6–10 times the power consumption of **hw-lut**. The power consumption of **wolkerstorfer** is about 9–20 times higher and those of **satoh** is even 17–34 times higher.

Figure 3 shows our results in terms of the power-area product. This metric is particularly relevant for applications which require both small silicon area and low power consumption, e.g. cryptographically enhanced RFID tags or sensor nodes. Due to their high power consumption, the calculating implementations have the worst power-area products.

For relaxed critical path conditions, **hybrid-lut**, **sub16-lut**, and **hw-lut** have similar characteristics, with slight advantages for **hybrid-lut** if a longer critical path is tolerable. When further reducing the critical path, **hybrid-lut** shows rather bad properties compared to **sub16-lut**. Finally, if an extremely short critical path is desired, **hw-lut** and **bertoni** have the best power-area product, followed by **bertoni-2stg**. Among these three designs, **bertoni** and **bertoni-2stg** should be preferred over **hw-lut** if a critical path delay of more than 4 ns is acceptable. Our results show slight advantages for **bertoni-2stg** compared to **bertoni** for some critical path targets (5–7 ns).

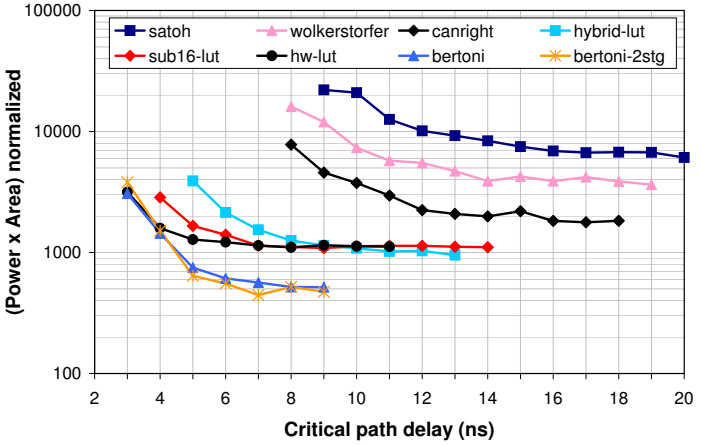

Fig. 3. Power-area product vs. critical path delay

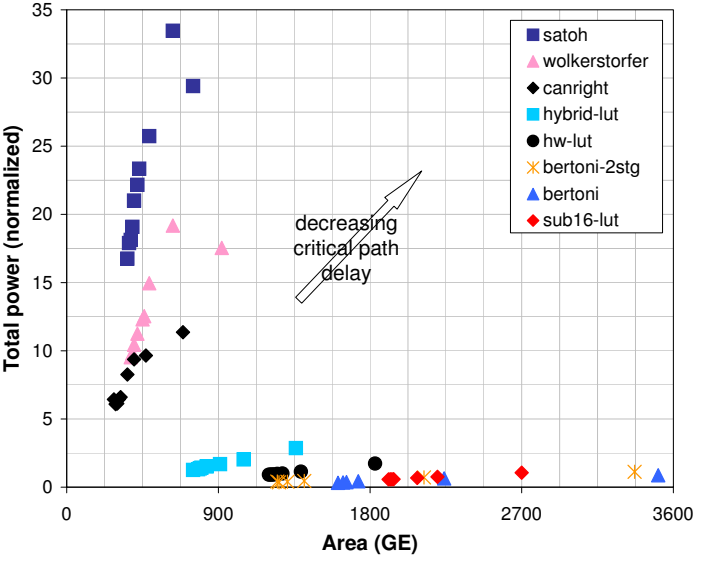

Fig. 4. Total power consumption vs. area

Figure 4 displays total power consumption in relation to required silicon area. Generally, the points farther away from the point of origin belong to synthesis results for shorter critical path delays. The figure shows that calculating implementations tend to sacrifice power efficiency to achieve higher speed. On the other hand, look-up and low-power implementations trade silicon area for a shorter critical path. To minimize the critical path delay, the synthesizer uses optimization techniques like the utilization of standard cells with higher driving

strengths and duplication of logic paths, which results in a considerable higher power consumption for signal switches. Calculating implementations have an inherently high number of signal switches and therefore incur an over-proportional increase in power consumption for reduced critical path delays. Look-up and low-power implementations, on the other hand, have much lower levels of signal activity which only leads to moderate increases in power consumption for shorter critical paths.

5 Conclusions

In this paper we have examined eight AES S-box implementations which follow three different design strategies. We have analyzed and compared various cost metrics like critical path delay, silicon area, and power consumption of these implementations based on synthesis runs with a 0.35 μm CMOS standard cell library. To our knowledge this is the first comprehensive survey of all possible standard cell implementations of the AES S-box published so far. While the results for the calculating implementations only apply to the AES S-box, the insights from the other two implementation strategies (look-up except **hybrid-lut** and low-power) are also useful for other cryptographic S-boxes.

Acknowledgements

The research described in this paper has been supported by the Austrian Science Fund under grant P16952–N04 and by the FIT-IT initiative of the Austrian Federal Ministry of Transport, Innovation, and Technology (project SNAP).

The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. G. Bertoni, M. Macchetti, L. Negri, and P. Fragneto. Power-efficient ASIC Synthesis of Cryptographic Sboxes. In *Proceedings of the 14th ACM Great Lakes Symposium on VLSI (GLSVLSI 2004)*, pp. 277–281. ACM Press, 2004.
2. D. Canright. A very compact S-Box for AES. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 441–455. Springer Verlag, 2005.
3. P. Chodowiec and K. Gaj. Very compact FPGA implementation of the AES algorithm. In *Cryptographic Hardware and Embedded Systems — CHES 2003*, vol. 2779 of *Lecture Notes in Computer Science*, pp. 319–333. Springer Verlag, 2003.
4. J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer Verlag, 2002.

5. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings Information Security*, 152(1):13–20, Oct. 2005.
6. A. Hodjat, D. D. Hwang, B.-C. Lai, K. Tiri, and I. M. Verbauwhede. A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-μm CMOS technology. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI (GLSVLSI 2005)*, pp. 351–356. ACM Press, 2005.
7. R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, second edition, 1996.
8. M. Macchetti and G. Bertoni. Hardware implementation of the Rijndael SBOX: A case study. *ST Journal of System Research*, 0(0):84–91, July 2003.
9. N. Mentens, L. Batina, B. Preneel, and I. M. Verbauwhede. Systematic evaluation of compact hardware implementations for the Rijndael S-box. In *Topics in Cryptology — CT-RSA 2005*, vol. 3376 of *Lecture Notes in Computer Science*, pp. 323–333. Springer Verlag, 2005.
10. S. Morioka and A. Satoh. An optimized S-Box circuit architecture for low power AES design. In *Cryptographic Hardware and Embedded Systems — CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 172–186. Springer Verlag, 2002.
11. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). Federal Information Processing Standards (FIPS) Publication 197, Nov. 2001.
12. N. Pramstaller and J. Wolkerstorfer. A universal and efficient AES co-processor for field programmable logic arrays. In *Field Programmable Logic and Application — FPL 2004*, vol. 3203 of *Lecture Notes in Computer Science*, pp. 565–574. Springer Verlag, 2004.
13. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-Box optimization. In *Advances in Cryptology — ASIACRYPT 2001*, vol. 2248 of *Lecture Notes in Computer Science*, pp. 239–254. Springer Verlag, 2001.
14. S. Tillich, J. Großschädl, and A. Szekely. An Instruction Set Extension for Fast and Memory-Efficient AES Implementation. In *Communications and Multimedia Security — CMS 2005*, vol. 3677 of *Lecture Notes in Computer Science*, pp. 11–21. Springer Verlag, 2005.
15. J. Wolkerstorfer, E. Oswald, and M. Lamberger. An ASIC implementation of the AES SBoxes. In *Topics in Cryptology — CT-RSA 2002*, vol. 2271 of *Lecture Notes in Computer Science*, pp. 67–78. Springer Verlag, 2002.
16. X. Zhang and K. K. Parhi. High-speed VLSI architectures for the AES algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(9):957–967, Sept. 2004.