IST-2002-507932

ECRYPT

European Network of Excellence in Cryptology

Network of Excellence

Information Society Technologies

D.VAM.2
State of the Art in Hardware Architectures

Due date of deliverable: 31. July 2005
Actual submission date: 05. September 2005

Start date of project: 1. February 2004 Duration: 4 years

Lead contractor: Institute for Applied Information Processing and Communications (IAIK)

Revision 1.0

Project co-funded by the European Commission within the 6th Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	

State of the Art in Hardware Architectures

Editor
Elisabeth Oswald (IAIK)

Contributors
Martin Feldhofer (IAIK), Kerstin Lemke (RUB), Elisabeth Oswald (IAIK),
François-Xavier Standaert (UCL), Thomas Wollinger (RUB)
and Johannes Wolkerstorfer (IAIK)

05. September 2005
Revision 1.0

The work described in this report has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Contents

Chapter 1

Executive Summary

This is the first of two deliverables that survey state-of-the-art hardware architectures for cryptographic algorithms. Hardware implementations of cryptographic algorithms have a long history. Traditionally, algorithms were implemented in hardware to achieve a higher speed than with implementations in software. The requirements of contemporary and future applications however, demand often other properties of hardware implementations.

Today we can identify two application scenarios where hardware implementations are advantageous over software implementations. Firstly, these are high-speed applications where a cryptographic co-processor performs the cryptographic operations in order to relieve the rest of the system. Secondly, these are applications where low power and low area requirements are stringent. In both application scenarios, the secure storage of keys is important.

In this deliverable we survey hardware architectures that are suitable for cryptographic applications. In particular, we analyze various hardware implementations of the AES algorithm. The focus of this survey will be on throughput-optimized circuits and on circuits designed for operation in very constricted environments where the power budget and the silicon area are sparse resources.

In order to provide a profound analysis of existing AES hardware we describe the AES algorithm briefly. The analysis of existing AES hardware shows considerations for light-weight implementations and for high-performance implementations. For both implementation goals, pointers and references to state-of-the-art implementations are given. Concepts and design considerations behind these implementations, which allow to push the limits of AES hardware implementations, are summarized in a compact manner.

Chapter 2

Introduction to Hardware Architectures

This is the first of two deliverables that survey state-of-the-art hardware architectures for cryptographic algorithms. Hardware implementations of cryptographic algorithms have a long history. Traditionally, algorithms were implemented in hardware to achieve a higher speed than with implementations in software. The requirements of contemporary and future applications however, demand often other properties of hardware implementations. Low power and low area are becoming increasingly important in the smart card and RFID field.

Numerous cryptographic algorithms exist and a growing number of algorithms is included in standards. However, there is a small number of algorithms that is actually used in practice. RSA and DES are prominent examples of algorithms that are frequently used. RSA and DES are also examples of algorithms that are gradually becoming replaced by other algorithms. Algorithms based on elliptic curve cryptography have become popular in context of electronic signatures. The Advanced Encryption Standard (AES) has been selected as successor of DES and an increasing number of applications and products are switching now from DES to AES encryption.

We take these developments into account in our deliverables. In this deliverable we give an overview of state-of-the-art hardware architectures in this chapter. Then we review hardware architectures for AES in Chapter 3. Thereby, we concentrate on architectures that are suitable for high-speed applications and we concentrate on architectures that are suitable for low-power applications. We conclude this deliverable in Chapter 4.

2.1 Historical Development

The implementation of cryptographic algorithms in hardware has a long history. From 1930 on, rotor machines were frequently used in military applications to encrypt and decrypt sensitive information. Rotor machines were electro-mechanical devices that implemented complex poly-alphabetic substitution ciphers—those ciphers were already too complex for calculations by hand. Consequently, one of the reasons for the popularity of rotor machines was that they made encrypting (and decrypting) messages easy.

With the computer era, a new tool (the computer) became available to simplify complex calculations. As a consequence, cryptanalysis became more powerful and new algorithms had to be designed. The designs changed; algorithms were now intended to run efficiently on

computers.

However, in the 1970s when modern cryptosystems such as DES and RSA were invented, it became attractive again to implement algorithms in dedicated hardware. Like in the case with rotor machines, devices were needed that would make using the new and strong cryptographic methods easy. As the standard computers at that time were mainly lacking the speed for fast executions of cryptographic algorithms, it was necessary to make special implementations of those algorithms in dedicated hardware.

The situation changed again over the years. Due to Moore's law, both computing power and memory become dramatically faster and cheaper and therefore, modern personal computers (PCs) are fast enough for most applications. Nevertheless, with the growing use of cryptographic methods in different applications, the need to securely store keys became apparent. Unfortunately, PCs were not designed with security in mind, neither were the operating systems or the Internet. Over the past years it has become clear that the PC platform is insecure; it is not a suitable platform to store cryptographic keys.

However, for example in applications that involve advanced electronic signatures, the signatory is required to store her signature creation data securely. Consequently, implementations of cryptographic algorithms in hardware that allow to store the key in a secure manner as well became attractive. This need triggered the era of so-called hardware security modules—devices that can create keys and perform cryptographic algorithms such that the keys never have to leave the device.

This idea of have cryptographic functionality and a secure storage has found many more applications. Smart cards, that we use as cash cards or that are used as SIM cards in mobile phones, for instance, have gained widespread acceptance, and in the recent years Radio-Frequency IDentification (RFID) chips have received significant attention. There desire of embedding security in all kinds of mobile and ubiquitous devices has triggered both the industry as well as the research community to get back to hardware implementations of cryptographic algorithms.

Today we can identify two application scenarios where hardware implementations are advantageous over software implementations. Firstly, these are high-speed applications where a cryptographic co-processor performs the cryptographic operations in order to relieve the rest of the system. Secondly, these are applications where low power and low area requirements are stringent. In both application scenarios, the secure storage of keys is important.

2.2 Cryptographic Hardware

We consider cryptographic hardware as any hardware that can perform cryptographic algorithms. This is a rather broad concept and includes standard processors that simply run a cryptographic algorithm as well. In the following sections we give a brief overview on the different kinds of cryptographic hardware that have gained widespread use and elaborate on the properties that are important for typical use-cases.

2.2.1 Cryptographic (Co-)Processors

In the most general case we can regard a general purpose processor that runs cryptographic algorithms as cryptographic hardware. A cryptographic (co-)processor is a device that is specifically designed to run certain cryptographic algorithms. Typically, one or more cryptographic co-processors are placed next to a general purpose processor. The co-processor is

then responsible for the cryptographic operations. Systems using such an approach are of PCI cards, which are are often used as hardware security modules (HSMs) or SSL accelerators. An increasing number of state-of-the-art SSL accelerators use an FPGA to implement the co-processor functionality.

For an HSM when used as signature creation device or within banking applications, the secure storage of secret keys is of utmost importance. Thus, for such devices not only the speed is a concern, but also its resistance to implementation attacks ([1], [16]) is important. If such a device needs to provide answers to requests over an open network (for example in an authentication protocol), additional care must be taken with respect to timing attacks [15].

Cryptographic processors as described in this section require software such that they can be used by the outside world. This software is called Application Programming interface (API) and it has turned out that flaws in APIs open the door for attacks [4].

2.2.2 Smart Cards and USB Devices

The term *smart card* is commonly used for cards that contain an integrated circuit (IC), the so called *Integrated Circuit Cards* (ICC)[13]. Nowadays, smart cards are supposed to be tamper proof hardware and in this function serve as security token in various applications. Depending on their specific interface, smart cards communicate directly i.e., with their metallic contacts with a card reader, which supplies them with power and with a clock signal. Some smart cards are equipped with an RF-interface and thus, do not need a physical connection with a reader.

Two types of smart cards can be distinguished. *Memory cards* have besides an *Electrically Erasable Programmable Read-Only Memory* (EEPROM) also some security logic and a read/write memory. The security logic allows to protect certain memory areas by implementing a simple kind of authentication mechanism. *Microprocessor cards* contain a full microprocessor mostly consisting of a Central Processing Unit (CPU), memory (Read-Only and Random Access), Input/Output (IO) and mostly also some cryptographic modules. Typical modules are random number generators, DES and RSA accelerators. Contemporary smart cards have an 8-bit, 16-bit or 32-bit CPU, up to 64 Kbyte ROM, up to 64 Kbyte EEPROM and up to 1 Kbyte of RAM. High-end smart cards generate their clock internally to improve the performance.

Microprocessor cards are the type of smart cards that are used for security sensitive applications. Due to the limited space and the limited power supply, low power and low area implementations of cryptographic algorithms are necessary for these platforms. Resistance against all kinds of implementation attacks is a further requirement for smart cards.

Security enhanced USB devices have become increasingly popular over the previous years. A typical security enhanced USB device essentially is a smart card with a USB interface. Implementations of cryptographic algorithms must therefore fulfill similar requirements as for smart card implementations.

2.2.3 RFID tags

RFID tags [14] are gaining widespread use already today as bar-code replacement. These tags have limited or no security components. If RFID tags are supposed to be used also in applications other than supply-chain management, for example as means to assure the authenticity of a product, security functionality will be required.

One of the advantages of RFID tags is that they do not need physical connection to a reader. They can be used up to several meters (the limit depends on the type of the tag) away from the reader. Nevertheless, these tags get their power supply from the reader. It is therefore mandatory to have very low power implementations of cryptographic algorithms for such devices.

2.3 FPGA vs. ASIC

There are two common methods in conventional computing for the execution of algorithms. The first is to use a purpose-built hardwired technology, for example an Application Specific Integrated Circuit (ASIC) to perform the operations in hardware. ASICs are specifically designed to perform a given computation, and are thus very efficient. The term efficient can have several meanings, for instance, the design can be very fast, or very small or can require only very little power. However, the circuit cannot be altered after production. This forces a redesign and remanufacturing of the chip if any part of the circuit needs to be modified.

The second method is to use a software-programmed microprocessor, a far more flexible solution. Processors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system is altered without changing the hardware. The downside of this flexibility is that the performance can suffer, if not in clock speed then in work rate, and is far below that of an ASIC.

Reconfigurable computing intends to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfigurable devices, including Field Programmable Gate Arrays (FPGAs), contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, the so-called logic blocks, are connected using a set of routing resources that are also programmable. As a consequence, the implementation of FPGA designs can be performed at the user site. Synthesis and implementation tools allow the high level description of a design to be translated into the programming language for an FPGA.

The reconfigurability of FPGAs offers several advantages when using them for cryptographic applications.

Algorithm Agility: This term refers to the switching of cryptographic algorithms during operation of the targeted application. One can observe that the majority of modern security protocols, such as SSL or IPsec, are algorithm independent and allow for multiple encryption algorithms. The encryption algorithm is negotiated on a per-session basis and a wide variety may be required.

Algorithm Upload: It is perceivable that fielded devices are upgraded with a new encryption algorithm. A reason for this could be that the product has to be compatible to new applications. From a cryptographical point of view, algorithm upload can be necessary because a current algorithm was broken (e.g., Data Encryption Standard - DES, a standard expired (e.g. DES), a new standard was created (e.g. Advanced Encryption Standard - AES, and/or that the list of ciphers in an algorithm independent protocol was extended. Assuming there is some kind of (temporary) connection to a network such as the Internet, FPGA-equipped encryption devices can upload the new configuration code. Notice that the upgrade of ASIC-implemented algorithms is practically

infeasible if many devices are affected or if the systems are not easily accessible, for instance in satellites.

Algorithm Modification: There are applications which require the modification of standardized cryptographic algorithms, e.g., by using proprietary S-boxes or permutations. Such modifications are easily made with FPGAs. One example, where a standardized algorithm was slightly changed, is the UNIX password encryption where DES is used 25 times in a row and a 12-bit salt modifies the expansion mapping. It is also attractive to customize block cipher such as DES or AES with proprietary S-boxes for certain applications. Furthermore, in many occasions cryptographic primitives or their modes of operation have to be modified according to the application.

Throughput: General-purpose processors are not optimized for fast execution especially in the case of public-key algorithms. Mainly because they lack instructions for modular arithmetic operations on long operands. Modular arithmetic operations include for example exponentiation for RSA and multiplication, squaring, inversion, and addition for elliptic curve cryptosystems. Although typically slower than ASIC implementations, FPGA implementations have the potential of running substantially faster than software implementations.

Cost Efficiency: There are two cost factors, that have to be taken into consideration, when analyzing the cost efficiency of FPGAs: cost of development and unit prices. The costs to develop an FPGA implementation of a given algorithm are much lower than for an ASIC implementation, because one is actually able to use the given structure of the FPGA (e.g. look-up table) and one can test the reconfigured chip endless times without any further costs. This results in a shorter time-to-market period, which is nowadays an important cost factor. The unit prices are not so significant when comparing them with the development costs. However, for high-volume applications, ASIC solutions are the more cost-efficient choice.

2.4 Hardware-Design Methodology

Design strategies are techniques that help creating digital integrated circuits concise and efficiently. Basically, design strategies are rules of thumb, which turned out to be useful in practice. These best practices help to reduce design complexity. Neil Weste numerates four techniques that should be applied during the development of any digital circuit: hierarchy, regularity, modularity, and locality [31]. This section discusses these design strategies, which any design flow should consider. Achieving hierarchy is the most important strategy when designing complex circuits. By enforcing hierarchy, it is possible to bring in abstraction into the design. Abstraction is necessary to handle complexity by hiding distracting details. Hierarchy is obtained by subdividing hardware modules into a set of smaller sub-modules. Sub-modules are more comprehensible than larger modules.

The decomposition of a circuit into sub-modules corresponds to the divide-and-conquer approach used in software development. It helps to lower the complexity of (sub-)modules and improves their reusability. Regularity is a design strategy that tries to keep the number of sub-modules resulting from hierarchical decomposition within limits. A small number of submodules might be used multiple times in the same project using instantiation techniques.

Instantiating a single leaf cell multiple times in regular array structures helps to keep the design effort of datapaths low compared to creating a large number of different cells. Optimizing a single leaf cell will improve the whole datapath. Standard cells used in application specific ICs are another example for regularity. Optimizing them is beneficial because they are reused in many projects. Modularity demands well-defined interfaces for sub-modules. Interfaces should unambiguously define several properties of a cell. Properties may comprise the name of the cell, its functionality, its interface, and may include electrical characteristics. Well defined interfaces facilitate assembling larger modules from submodule instances. Good examples for modularity are standard cells, which obey rigid schemes for their interfaces. For example, the abutment of standard-cell instances generates automatically power rails because all standard cells use the same approach for power routing. Locality is a design strategy that hides details of modules. Internal construction details should be hidden inside a cell to abstract its functionality and other characteristics. In case locality is met by sub-modules, characteristics of modules can be predicted by the characteristics of its sub-modules. Examples for locality are submodules with registered output that allow predicting the maximum clock frequency of assembled modules by analyzing sub-modules. Contrary, modules having combinational paths will inhibit this and thus violate the locality design strategy.

2.4.1 Top-Down Design Approach

In order to develop a complex digital system it is necessary to apply a structured design methodology that is capable to detect potential flaws and weaknesses as early as possible to shorten the design time. For multi-million transistor designs, this is only possible by applying a top-down approach that subdivides the problem of finding good hardware into several layers of abstraction. The highest level of abstraction will define the intended functionality and some boundary conditions under which the circuit should work. The lowest level will represent the physical implementation of the circuit in silicon. A top-down design methodology creates hardware by defining the highest level of abstraction first and refines it using some intermediate abstraction layers until the physical implementation is obtained. The top-down methodology with its different abstraction levels keeps the complexity of each abstraction level within limits.

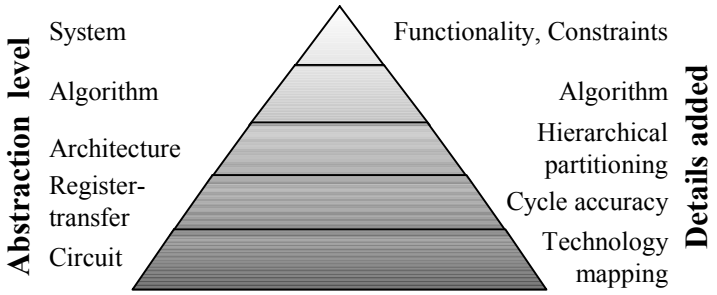

Figure 2.1: Abstraction Levels

Using a top-down approach to design a complex digital system will automatically subdivide the overall problem into a number of smaller problems. This decomposition of problems follows a divide-and-conquer strategy, which is known to be efficient for solving problems in the field of computer science. The decomposed problems are smaller and their solutions add

up to solve the overall problem. The smaller problems can be classified into several layers of abstraction as depicted in Figure 2.1. Figure 2.1 lists five abstraction levels that are commonly differentiated in the domain of digital circuit design.

The 5 levels of abstraction are the system level, the algorithmic level, the architectural level, the register-transfer level, and the circuit level (see Figure 2.1). A short description of these levels is given subsequently.

System Level: The system level defines the functionality of a digital system. Boundary conditions, under which the circuit should work, are defined on this layer too. The system level is the highest abstraction level and hence describes the digital systems without many details. Basically, the system level should describe the type of application and the intended functionality of the circuit rather than defining the means by which the functionality is achieved.

Algorithmic Level: The algorithmic level explores different alternatives to implement desired functionality of a circuit. It can be a refinement of a previous executable specification. But most often, the functional model of the circuit is written from scratch using a programming language that allows implementing the functionality quickly.

Architectural Level: The architectural level of a circuit defines a circuit by its modules and their submodules. These modules implement functionalities defined in the high-level model but the modules do not necessarily comply with the functions (or classes) of the high-level model. There are two typical approaches for finding efficient hardware architectures. Either, one searches for a hardware module that can implement many functions of the high-level model. Or, one tries exploit the immanent parallelism of hardware. Many algorithms allow to process data in parallel. Multiple instances of (sub-)modules accelerate the computation when operating concurrently.

Register-Transfer Level: The register-transfer level refines the architectural level by defining the modules and sub-modules more precisely. The register-transfer level defines on one hand the functionality more precisely by fixing which operations are executed in which clock cycle. Thus, a hardware model on register-transfer level is a cycle-accurate description of the circuit. Digital hardware is usually synchronous, which means that a global clock orchestrates all computations. The register-transfer level uses only datatypes that resemble hardware signals. Any data has to be represented by digital signalseither single signals or by groups of digital signals. Such busses have to have a determined width. Thus, models on register-transfer-level are also bit-accurate.

Circuit Level: The circuit (or gate) level represents the physical implementation of a circuit. It is obtained by mapping the desired functionality onto the target technology. There are two major options for target technologies: either reconfigurable logic or custom silicon implementation. The first option—reconfigurable logic—is typified by FPGAs. These off-the-shelf integrated circuits get their functionality by configuration. However, the cost of FPGAs and their considerable power consumption admit their use mainly in limited-lot production or for use in prototypes. Custom silicon implementation is appropriate for high-volume products and for optimized implementations where high clock frequencies are important or low power

consumption is crucial. The state-of-the-art method to produce digital circuits on CMOS technology is to use a semi-custom design flow, which implements circuits by placing and routing supplied standard cells.

2.4.2 Semi-Custom Design

The design flow for generating a circuit's representation on circuit-level is much the same for FPGA and ASIC technologies. A synthesizer transforms the HDL model written on register-transfer level into a netlist on gate-level. Place-and-route tools take this netlist as input. They place the cells offered by the target technology and generate interconnects by routing wires.

2.5 Hardware Security

It has become obvious in the recent years that implementations of cryptographic algorithms can be attacked in various ways. Invasive attacks require direct electrical access to the internal components of the device unter attack. Semi-invasive attacks require to access the device. However, neither the the passivation layer gets damaged nor does the attacker make electrical contact to the internals of the device. Non-invasive attacks require an attacker to manipulate a device by only using the standard inputs and outputs. These attacks are active attacks in the sense that the attacker actively abuses a device (i.e. uses it not in its intended way). Passive attacks are also known. Passive implementation attacks are commonly referred to as side-channel attacks. They are also non-invasive and typically require an attacker to monitor side channels that leak sensitive information from a device. The security of implementations against non-invasive attacks is discussed in two recent deliverables ([3] and [2]) of the VAM3 working group.

Chapter 3

AES Hardware Architectures

3.1 Introduction

In 2001, the National Institute of Standards and Technologies NIST adopted the Rijndael algorithm as the Advanced Encryption Standard AES [19]. The AES algorithm began immediately to replace the Data Encryption Standard DES which was in use since 1976. AES excels DES at improved long-term security because of larger key sizes (128, 192, and 256 bits). Another major advantage of AES is its ability of efficient implementation on various platforms. AES is suitable for small 8-bit microprocessor platforms, common 32-bit processors, and it is appropriate for dedicated hardware implementations. Hardware implementations can reach throughput rates in the Gigabit range. The efficiency of implementation and the free availability of the algorithm pave the way for the use of AES in applications like wireless LAN according to the IEEE 802.11i standard and further standards like the ISO 18033-3, IPSec, and TLS.

Although AES is used in many different applications, hardware implementations of the algorithm focus mostly on throughput optimization. Early hardware implementations—the first attempts were undertaken during the selection process of the algorithm—were straightforward implementations that had no optimization goal in mind. Often, these architectures even lacked complete functionality. Many circuits were pure encryption devices without support for decryption.

In the meantime, more mature reports about AES hardware implementations are available [5, 17, 23, 24, 27, 29]. Most of them stress throughput optimization with no hardware resource restrictions [29]. Only a few implementations try to realize resource-efficient hardware which squeeze out adequate performance from limited silicon area and limited power budgets. Most of the published AES architectures are intended for use on configurable platforms like FPGAs [5, 24]. Only a few architectures tailored for silicon implementation are published [17, 27].

This document will analyze various hardware implementations of the AES algorithm. The survey will focus on very efficient implementations. The efficiency of an AES hardware implementation depends on the field of application. In one case efficiency could mean tremendous throughput for instance in networking applications. Efficiency could mean resistance against side-channel attacks for smartcard applications. Or efficiency could mean power efficiency for radio-powered devices. Within the VAM2 working group, we maintain a web-site, the so-called *AES Lounge* see `http://www.iaik.tugraz.at/research/krypto/AES`, that gives an up-to-date overview of AES implementations.

The focus of this survey will be on throughput-optimized circuits and on circuits designed for operation in very constricted environments where the power budget and the silicon area are sparse resources.

In order to provide a profound analysis of existing AES hardware it is necessary to describe the AES algorithm briefly (§3.2). The analysis of existing AES hardware shows considerations for light-weight implementations (§3.3) and for high-performance implementations (§3.4). For both implementation goals pointers and references to state-of-the-art implementations are given. Concepts and design considerations behind these implementations, which allowed to push the limits of AES hardware implementations, are summarized in a compact manner.

3.2 Description of the Advanced Encryption Standard AES

The Advanced Encryption Standard AES is a symmetric block cipher [19]. It operates on 128-bit blocks of data. The algorithm can encrypt and decrypt blocks using secret keys. The key size can either be 128-bit, 192-bit, or 256-bit. The actual key size depends on the desired security level. The different versions are most often denoted as AES-128, AES-192, or AES-256. Today, AES-128 is predominant and supported by most hardware implementations.

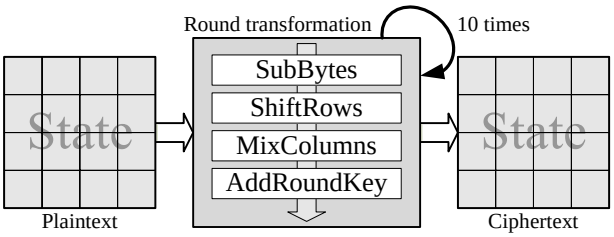

Figure 3.1: AES iterates a round transformation.

3.2.1 AES Design Considerations

The central design principle of the AES algorithm is simplicity [7]. Simplicity facilitates implementations on different platforms under different sets of constraints. The simplicity is achieved by two means: the adoption of symmetry at different levels and the choice of basic operations.

The first level of symmetry lies in the fact that the AES algorithm encrypts 128-bit blocks of plaintext by repeatedly applying the same round transformation, outlined in Figure 3.1. AES-128 applies the round transformation 10 times, AES-192 uses 12, and AES-256 uses 14 iterations. While many other encryption algorithms share this aspect of symmetry, there are some notable exceptions. For example, Serpent and MARS, two other candidate AES algorithms use 8, respectively 4 different round transformations.

Symmetry can also be found within the definition of the AES round transformation. This is discussed in the next section. The symmetry in the structure allows to reuse hardware components multiple times and hence to produce very small implementations.

The basic operations used in the AES algorithm can all be described very easily in terms of operations over the finite field GF(2^8). This property allows to reason about the algorithm using established mathematical techniques, facilitating security analysis as well as the construction of optimal implementations. Secondly, finite-field operations can be implemented

very efficiently in hardware in comparison to integer arithmetic, where the presence of a carry requires special precautions.

3.2.2 AES Round Transformation

The round transformation modifies the 128-bit State. The initial State is the input plaintext and the final State—after the round transformations—is the output ciphertext. The State is organized as a 4×4 matrix of bytes. The round transformation scrambles the bytes of the State either individually, row-wise, or column-wise by applying the functions SubBytes, ShiftRows, MixColumns, and AddRoundKey sequentially. An initial AddRoundKey operation precedes the first round. The last round differs slightly from the others: the MixColumns operation is omitted.

The functions of the round transformation are linear and non-linear operations that are reversible to allow decryption using their inverses. Every function affects all bytes of the State. The function SubBytes is the only non-linear function in AES. It substitutes all bytes of the State using table look-up. The content of the table can be computed by a finite-field inversion followed by an affine transformation in the binary extension field GF(2^8). The resulting look-up table is often called S-Box. The same S-Box is used for all 16 bytes of the State. The ShiftRows function is a simple operation. It rotates the rows of the State by an offset. The offset equals the row index: the first row is not shifted at all, the last row is shifted three bytes to the left. The MixColumns function accesses the State column-wise, working on each column in the same way. It interprets a column as a polynomial over GF(2^8), with degree smaller than 4. The State bytes are the coefficients of the polynomial. The output column corresponds to the polynomial obtained by multiplication by a constant polynomial and reducing the result modulo $x^4 + 1$. The AddRoundKey function adds a round key to the State. It is a 128-bit XOR operation. A new round key is derived in every iteration from the previous round key. The initial round key equals the original secret key. The computation of the round keys is based on the SubBytes function and uses additionally some simple byte-level operations like XOR.

Decryption computes the original plaintext of an encrypted ciphertext. During the decryption, the AES algorithm reverses encryption by executing inverse round transformations in reverse order. The round transformation of decryption uses the functions AddRoundKey, InvMixColumns, InvShiftRows, and InvSubBytes—in this order. AddRoundKey is its own inverse function because the XOR function is its own inverse. The round keys have to be computed in reverse order. InvMixColumns needs a different constant polynomial than MixColumns does. InvShiftRows rotates to the right instead of to the left. InvSubBytes reverses the S-Box look-up table by an inverse affine transformation followed by the same inversion over GF(2^8) which was used for encryption.

3.2.3 Hardware Aspects of AES

Most operations of AES are byte-oriented. So, they can be executed efficiently on 8-bit processors. On 32-bit processors, AES is efficient too because some 8-bit operations can be combined to 32-bit operations. In hardware implementations, any word size is suitable. Most hardware implementations prefer 128-bit architectures. This offers the greatest degree of parallelism to increase concurrency of AES computations. A higher degree of concurrency allows higher throughput. Some implementations even unroll the ten iterations of the round

transformation. By using extra hardware for each of the ten rounds, a pipelined version of AES hardware is formed. This comes at the price of tenfold hardware resources. Such an approach obviously neglects the symmetry present in the AES design. Furthermore, pipelining is not always useful because there are modes of operation, which are used for bulk encryption that are not able to exploit the parallelism given by a pipeline architecture. For instance during encryption, the cipher block chaining (CBC) mode requires the result of the previous encryption as input. As a consequence, the pipeline is stalled until the result is available. Thus, most AES hardware implementations have only one round realized in hardware that is reused to compute all ten iterations. For throughput rates in the Megabit range, it is not necessary to implement the whole round in hardware using a 128-bit architecture. Smaller architectures can fulfill the requirements. For instance, 32-bit architectures reach 70 Mbps at 50 MHz clock frequency [23].

The size of the architecture (32-bit, 128-bit) defines the size of the circuit. A 32-bit architecture needs four S-Boxes to compute the SubBytes function of a 32-bit word, while a 128-bit architecture requires 16 S-Boxes. S-Boxes are the most spacious parts of an AES hardware implementation. The number of S-Boxes determines the overall size of an AES hardware module. Thus, using an efficient approach for implementing S-Boxes is crucial for AES hardware. Basic options for implementing S-Boxes are look-up tables using ROMs. ROMs have to store $8 \times 256 = 2048$ bits. Alternatively, it is possible to compute the S-Box output by calculating the finite-field inversion and the subsequent affine transformation. J. Wolkerstorfer et al. pointed out that combinational logic can do this calculation as efficient as ROMs can do table look-ups [33]. In particular, using combinational logic is superior when decryption is needed too. In this case, the size of a ROM will double to store 4096 bits, while the computational approach can reuse the inversion circuitry for InvSubBytes. Only circuitry for the inverse affine transformation has to be added.

The finite-field multiplications of MixColumns can be combined with the table look-up of SubBytes into so-called T-Boxes [7]. However, the MixColumns function is also suitable for realization with combinational logic. Hardware implementations of MixColumns are smaller than T-Boxes, have a shorter critical path, and consume less power. A combined implementation of MixColumns and InvMixColumns can reuse many terms that appear in both computations.

The storage requirement of an AES implementation has much impact on the overall size of the circuit. An AES-128 implementation needs at least 256 bits for storage: 128 bits to store the State and 128 bits to store the actual round key. On some platforms it is more efficient to use additional memory. On Xilinx FPGAs, N. Pramstaller et al. [24] showed that a duplication of the State can reduce the overall hardware costs. In standard-cell circuits, storage using flip-flops is more costly. Thus, standard-cell circuits usually store the State only once. Memory considerations are also of interest for the round-key generation. Software implementations on 32-bit platforms compute all round keys beforehand. This saves time when several blocks have to be encrypted and facilitates decryption during which the round keys are used in reverse order. On area-efficient hardware implementations, the key schedule is preferably done on-the-fly to lower storage requirements. In every iteration of the round transformation, the according round key is computed from the previous one. For decryption, the so-called inverse cipher key has to be computed before decryption can start. This key is the last round key of encryption.

3.3 Lightweight Implementations

The definition of "lightweight implementations" of the AES algorithm is important for the specification of the requirements. Different aspects like die size and power consumption for hardware implementation as well as code size and performance for software implementations have to be considered. In this document, the term lightweight is understood in respect to low die size and low power consumption for hardware implementations. The fields of application are passively-powered and battery-powered devices in pervasive and ubiquitous computing. Especially the field of RFID devices and wireless sensor networks are of major interest for us.

3.3.1 Design Principles

An optimized silicon implementation of a cryptographic algorithm requires a stringent design methodology which adheres to approved design principles. It has to evaluate different design options carefully. In the presented design, low die size and low power consumption are the major goals while high data throughput is of minor importance. It is inevitable to study the pros and cons of a measure to reach all design objectives. For example, a specific step can reduce the silicon area of the chip but can increase the power consumption significantly. In this section various methods are described which require attention during the design phase.

The chosen hardware architecture of a chip design mainly determines the properties of an implementation. Well considered design decisions on architectural level help to meet requirements like low power consumption, low die size, and acceptable throughput rates. Nearly all published hardware implementations of AES have Gigabit throughput rates as optimization goals. This is contrary to implementation presented here because the priority objective of this work is to reduce the die size. The die size directly influences the cost of production of high-volume digital circuits. Additionally, low power consumption is important to meet the constraints for contactless devices like smart cards or RFID tags.

Along with optimizations on the architectural level it was very important to consider various implementation details to achieve the ambitious design objectives concerning low die size and low power consumption. In VLSI design, it is always an option to optimize critical parts of a circuit using a full-custom approach instead of using standard cells. Using full-custom cells can cause some problems. First, the development time increases significantly which makes the chip much more expensive. Additionally, a quick migration to a more recent CMOS technology is not possible. Therefore, all optimizations are done within the VHDL description. This facilitates a fast design flow including synthesis, place and route for different CMOS technologies and produced a first-time-right silicon.

Design for Low Die Size

The design of a VLSI circuit for low die size using standard cells involves considerations from algorithmic level down to the circuit level. For the AES algorithm, the decision was made to support the 128-bit version for encryption and decryption. The smallest useful word size, namely an 8-bit architecture, was chosen and the algorithm was implemented such that hardware resources are reused as much as possible.

Using a dedicated RAM instead of a flip-flop based approach came into question during the design. Normally, RAM macros are more area efficient because of their regular layout. Dedicated RAM is only more efficient when the memory size is larger than the 256 bits.

Additionally, a dedicated RAM circuit would consume much more power because RAMs use power-consuming pre-charging of bit lines for every access.

Design for Low Power Consumption

Power consumption has become a major optimization goal of today's VLSI design. The differences between power consumption and energy consumption might be important to notice. For battery-powered devices the energy consumption per operation is the optimization goal. This means the power-delay product should be minimized. In contrary, for passively powered devices like contactless smart cards the mean power consumption is the critical concern. The duration of the operation does not matter. It is of importance that the power consumption per clock cycle is limited although the total energy consumption of an operation might be larger. Here it is often necessary to serialize operations because the concurrent calculation would exceed the available power. The implementation of the AES presented here minimizes mean power consumption.

The total power consumption of a CMOS circuit is the sum of static and dynamic power consumption. The static power consumption caused by the leakage current, mainly depends on the size of the chip. It is very small and can be more or less ignored here. The dynamic power consumption consists of loading and unloading the total capacitance (C_L) of the chip. Equation 3.1 presents the influences on dynamic power consumption. The design measures for lowering the power consumption result from minimizing the factors in this equation.

$$P_{dyn} = C_L \cdot {V_{DD}}^2 \cdot f_{CLK_{eff}} \cdot E_{sw} \tag{3.1}$$

The load capacitance on the chip C_L increases as more gates are placed on the die. This means that lowering the die size as well as reducing the supply voltage (V_{DD}) to a minimum directly reduces power consumption. These two coefficients are somehow predetermined by the low die-size constraint and the operating conditions of the chip. Assuming a fixed supply voltage, the best option for a low-power design is reducing the effective clock frequency $f_{CLK_{eff}}$ of the circuit. It reduces the power consumption linearly.

Clock gating is a very effective measure for reducing the effective clock frequency. In the presented architecture all datapath registers and nearly all control logic registers are clocked only when there is a potential signal change. For example, in the RAM only one 8-bit register can be written at a time. This has the advantage that only one register consumes power at the active clock edge. Additionally, there is no need to have multiplexers at the input of a register to store its old value. This reduces the silicon area of the circuit. In the AES hardware implementation, this measure was applied rigorously in all submodules. It turned out to lower the power consumption significantly. Thereby, parts of the circuit are virtually switched off when they are not in use.

The switching activity E_{sw} of the circuit can be reduced by using a method called sleep logic. Whenever the output of a combinational circuit is not needed changes of the input data will nevertheless cause switching activity and hence power consumption inside the module although the computed data is not needed. In order to prevent this undesired switching activity the inputs of the combinational circuit are masked using AND gates. A sleep signal that disables the AND gates prevents all switching activities of the combinational logic behind the gate because the input is constantly zero.

3.3.2 Related Work

The first hardware implementations of the AES algorithm showed up during the selection process of the AES standard. In the meantime, a wide spectrum of implementations was published. Most of these AES modules use reconfigurable hardware (FPGAs) as target technology. AES implementations optimized for FPGA platforms can differ substantially from standard-cell implementations because cost functions of FPGAs are very different to standard cells as the previous discussion about storage showed. Therefore, the choice of related work in this section is mostly limited to lightweight CMOS realizations which have area efficiency and low power consumption as major design goal.

The AES hardware of A. Satoh et al. [27] is a 32-bit architecture. It has a separated encryption/decryption block and a key expansion circuit. The four S-Boxes are used for the SubBytes operation as well as for key scheduling. There are two very large multiplexers. The first one is used for selection of 32-bit words from the State or the key storage, respectively. The second multiplexer is required at the output of the data path for selecting the appropriate result. Because of their internal structure they need one MixColumns multiplier and two InvMixColumns multipliers. The fully functional encryption and decryption module supports 128-bit keys. It has a hardware complexity of 5,400 gates and reaches a throughput of 311 Mbps.

A highly regular approach was presented by Mangard et al. [17]. This 32-bit architecture performs encryption and decryption for various key sizes. It uses 16 instances of a data cell that stores eight bits per cell. The cell contains a MixColumns multiplier and performs all transformations except SubBytes. The parameterizable number of S-Boxes, which can be four or 16, makes the design scalable concerning data throughput. In a follow-up article the authors describe a minimum version of their architecture where only four MixColumns multipliers are used [23]. This approach is comparable to the work in this section but requires a chip area of 8,500 gate equivalents while having a higher data throughput of 70 Mbps. In addition, it supports the CBC mode.

Feldhofer et al. described in an article the application of an encryption-only implementation of the AES algorithm [8]. It has application in RFID technology and uses an 8-bit architecture. The implementation details described in this section mainly originate from the work of Feldhofer et al. in [9] which describes a manufactured chip implementation.

3.3.3 Implementation Details

The design of the AES hardware implementation used a flexible methodology which put forth a lot of possible optimization ideas. All ideas were evaluated regarding their impact on the silicon size and on the power efficiency. The evaluation is based on synthesis results and circuit-level simulations. These in-depth analysis assure that the circuit achieves the requirements for passively powered devices. In the following, the architecture of the AES module is presented and details of the circuit that contribute to its efficiency are discussed.

Hardware Architecture

In [9], the AES was implemented for encryption and decryption using a fixed key size of 128 bits. This reduces the number of rounds to ten and the required memory for the State plus the round key does not exceed 256 bits. The low-power requirements of the chip are too restrictive to allow using 128-bit operations. Even a 32-bit implementation of AES would

not fit to the requirements. Therefore, the decision was to implement an 8-bit architecture of AES where all operations consume significantly less power than 32-bit operations do. The architecture of the AES can be seen in Figure 3.2.

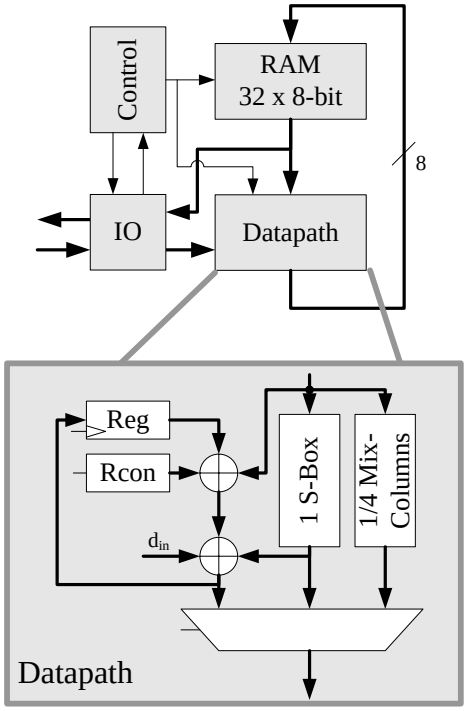

Figure 3.2: Architecture of 8-bit AES module.

The main parts of the AES are the controller, the RAM, the datapath, and the IO module. The IO module has a microcontroller interface that allows to use the AES module as a coprocessor. The controller accepts commands from the IO module and provides control signals for RAM and the datapath to sequence AES operations. The controller is realized as a hard-wired finite-state machine. This allows to optimize the efficiency concerning low-power consumption and low die size. It mainly consists of a 4-bit round counter and address registers for addressing rows and columns of the RAM. These counters are implemented as shift registers using one-hot encoding. One-hot encoding ensures that changes of the state cause only two signal transitions. Moreover, one-hot encoding reduces undesired glitching activity of control signals.

The finite-state machine sequences the ten rounds consisting of the operations AddRoundKey, ShiftRows, SubBytes, MixColumns, or their inverse operations. Additionally, all round keys are generated in time for every round of the AES. This on-the-fly round key generation helps to reduce the necessary storage capacity of the RAM block to 256 bits. The first 128 bits store the actual State and the second 128 bits store the current round key. As no spare memory is present for storing intermediate values, the controller has to assure that no State byte nor key byte is overwritten if it is needed again during calculation.

The RAM is single ported to ease silicon implementation. It is realized as a flip-flop based memory. The extensive use of clock gating lowers the power consumption. Additionally, this standard-cell based approach eases the physical realization compared to using a dedicated

RAM macro block.

Datapath Implementation

The datapath of the AES module contains combinational logic to calculate the AES transformations SubBytes, MixColumns, AddRoundKey, and their inverse operations (see Figure 3.2). The ShiftRows/InvShiftRows transformation is implemented by appropriate addressing of the RAM. It is executed when results of the S-Box operation are written back.

The remaining components of the datapath are the submodule Rcon, some XOR gates, and an 8-bit register to store intermediate results during key scheduling. Rcon is a circuit which provides constants needed for the key schedule. The XOR gates are needed for round key generation and are reused to add the State with the round key during the AddRoundKey transformation. Additionally, the data input and key input are handled by the data path.

A design goal was to equalize the power consumption of all datapath operations occurring during the execution of the AES algorithm. The equalization is very important for contactless devices because the most power demanding operation might cause a reset of the whole circuit. This reset may be triggered by dropping of the supply voltage below a defined minimum. As a consequence, submodules of the datapath like the S-Box or MixColumns were designed such that their power consumption is nearly the same.

The encryption or decryption of 16-byte blocks works as follows. The 16 bytes of input data are successively written to the RAM through the 8-bit microcontroller interface followed by the 16 bytes of the key. The initial AddRoundKey operation is performed during the loading of the key. For decryption, the inverse cipher key must be loaded because all round keys are calculated in reverse order. Issuing the start command to the control input starts encryption or decryption. The ten AES rounds with the functions SubBytes, ShiftRows, MixColumns for encryption and the functions InvSubBytes, InvShiftRows, InvMixColumns for decryption are performed according to the algorithm specification. During the computation of AddRoundKey, which is equal for encryption and decryption, the subsequent round key is derived from its predecessor using the S-Box, Rcon, and the XOR functionality of the datapath. Encryption can be done within 1032 clock cycles including the IO operation. Decryption needs 1165 clock cycles because of its different key schedule.

S-Box Implementation A significant advantage of the 8-bit architecture of the design is to reduce the number of S-Boxes from at least four of a 32-bit implementation to one instance. This reduces the required silicon resources. The single S-Box is used for the SubBytes and the InvSubBytes operation as well as for key scheduling. The S-Box is the biggest part of the AES datapath. There are several options for implementing an AES S-Box. The most obvious option is a 512×8-bit ROM to implement the 8-bit table look-up for encryption and decryption. Unfortunately, ROMs do not have good properties regarding low-power design.

A particularly suitable option is to calculate the substitution values using combinational logic as presented in [33]. One feature of this S-Box is that it can be pipelined by inserting register stages. The selected S-Box implementation uses one pipeline stage which shortens the critical path of the S-Box and lowers glitching activity. Furthermore, this pipeline register is used as intermediate storage during the ShiftRows operation. During the substitution of one byte, the next byte is read from the memory. The substituted byte is written to the current read address. By choosing the read addresses properly, the SubBytes and the ShiftRows operation are combined. ShiftRows degrades to mere addressing.

In order to reduce the signal activity of the S-Box circuit an advanced variant of sleep logic is applied. When the output of the S-Box is not needed the inputs are switched off to put the S-Box in an idle mode. This is done because signal changes at the input of a combinational circuit cause power consumption although no useful computation is performed. During idle mode, the value 0x52 for encryption or 0x63 for decryption is applied to the S-Box input. This has the advantage that the output of the S-Box is equal to zero because S-Box(0x52)=0x0 and InvS-Box(0x63)=0x0. In addition to the lower signal activity during the idle mode a simple XOR suffices to multiplex the output of the S-Box and other outputs of datapath components (MixColumns, ...). Otherwise, a dedicated multiplexer is needed for this task. The XOR approach has area and power advantages.

MixColumns Implementation Another innovative solution to achieve low power consumption is the calculation of the MixColumns and InvMixColumns operations. A submodule which calculates one fourth of the MixColumns and InvMixColumns operation in one cycle was developed. Instead of using four multipliers as stated in [32] only one modified multiplier is used.

The columns of the State are considered as polynomials over GF(2^8) that are multiplied modulo $x^4 + 1$ by the fixed polynomial $c(x)$ for encryption. The inverse polynomial $c^{-1}(x)$ is used for decryption. Equation 3.2 shows the derivation of both equations to reduce the effort for a combined computation of encryption and decryption. When subtracting the encryption polynomial $c(x)$ from the decryption polynomial $c^{-1}(x)$ it can be seen that after extracting the common coefficients only the coefficients $\{08\}$ and $\{0c\}$ are used in addition to the encryption-only result.

$$\begin{aligned} c(x) &= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \\ c^{-1}(x) &= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \\ c^{-1}(x) - c(x) &= \{08\}x^3 + \{0c\}x^2 + \{08\}x + \{0c\} \\ c^{-1}(x) - c(x) &= \{08\}(x^3 + x) + \{0c\}(x^2 + 1) \\ c^{-1}(x) &= c(x) + \{08\}(x^3 + x) + \{0c\}(x^2 + 1) \end{aligned} \tag{3.2}$$

This novel approach led to an efficient multiplier implementation as shown in Figure 3.3. It calculates one byte per cycle after a pre-loading phase of three cycles. The processing of one column of the State takes seven clock cycles. A complete MixColumns or InvMixColumns operation takes 28 clock cycles to transform the entire State. The critical path of the multiplier circuit is even shorter than the one of the S-Box.

3.3.4 Characteristics of Lightweight Implementations

The silicon implementation of the chip described in [9] was realized using the 0.35 μm CMOS standard-cell library from Philips Semiconductors. A strict design methodology was applied to ensure first-time right silicon. The circuit was described and verified in VHDL on register-transfer level. Some circuit elements very were modeled precisely to enforce desired results of the synthesizer. Synthesis used the PKS shell from Cadence and clock tree generation was done using CT-Gen. Continuous testing and verification eliminated errors during the design steps. Placement and Routing were done with Silicon Ensemble from Cadence. Back-end verification ensured manufacturability. These tests consisted of static timing analysis,

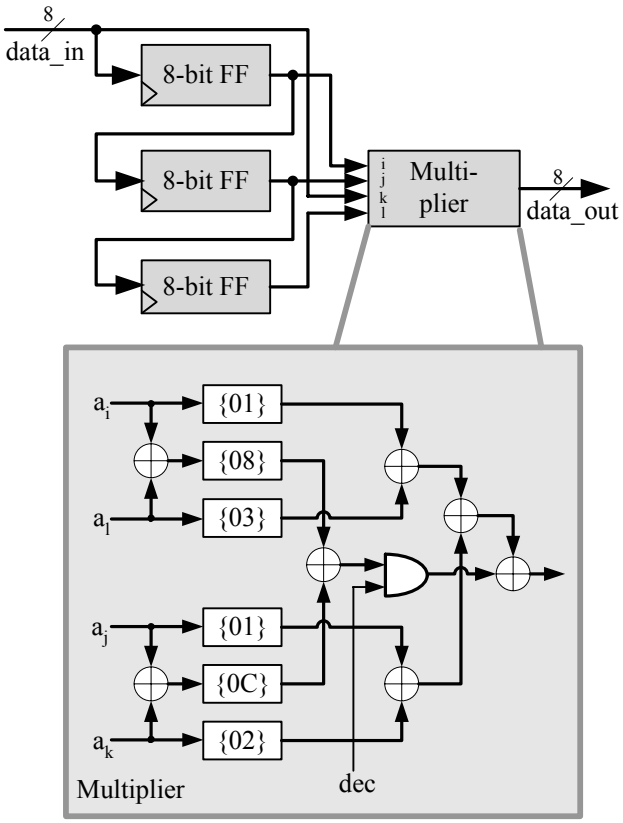

Figure 3.3: MixColumns submodule.

power simulations, LVS, and DRC. After manufacturing the chip, verification of the correct functionality of the circuit was done on a HP82000 chip tester environment. Test vector sets supplied from NIST were used to make functional tests as well as for parameterizing the chip concerning power consumption. The region of operativeness regarding supply voltage and clock frequency was evaluated using sweep tests.

Die Size

The die size of the chip is the smallest known implementation of AES-128 supporting encryption and decryption. The core needs a silicon area of 0.25 mm^2 on a 0.35 μm CMOS which roughly compares to 4,400 gate equivalents. Synthesis results indicate a complexity of 3,400 gates. The difference to the layout complexity is due to the clock tree, filler cells and other layout overhead. Figure 3.4 shows the die photo using a magnification factor of roughly 40. The largest part of the circuit is the flip-flop based RAM. It consumes about 60% of the chip area. The controller including the finite-state machine, the round counter, and the addressing registers need as well as the S-Box 12% each. MixColumns takes 9% of the total silicon area. The remaining area is occupied by the Rcon submodule, the XOR gates, and the multiplexer at the output of the datapath.

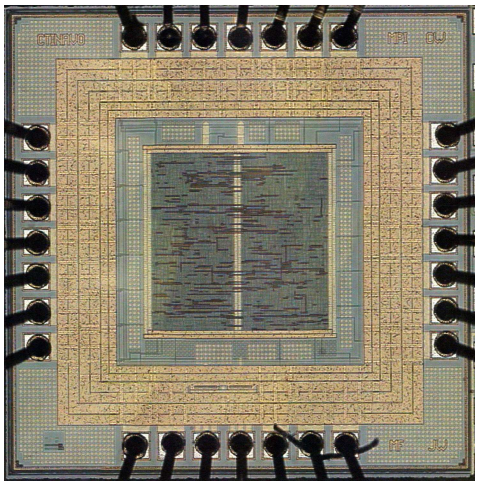

Figure 3.4: Die photo of AES-128 IC.

Performance

Measurement results on the chip tester indicated correct functionality even at very low supply voltages. The chip works correct with a supply voltage higher than 0.65 V. At this voltage, a clock frequency of nearly 2 MHz is reached. At the maximum supply voltage of 3.3 V, the maximum clock frequency of 80 MHz is reached. The encryption of one 128-bit block requires 1032 clock cycles including loading data and reading data. The maximum throughput rate for encryption is 9.9 Mbps. The performance for decryption is nearly the same as for encryption.

Power Consumption

A common method to measure the power consumption of digital circuits is to measure their supply current and assuming constant supply voltage. Therefore, the voltage drop of the supply current is measured at a small ohmic resistor using a digital oscilloscope. This method is invariably error-prone because of current spikes that are common in digital systems. Moreover, very small supply currents are difficult to keep apart from noise. Therefore, an alternative method to measure the power consumption of the AES chip was applied. The charge transfer method was used. Especially for low-power circuits this method is commonly applied. Thereby, the voltage drop of a capacitance that supplies the chip is measured while the chip performs its operation. The measured mean current consumption of the AES-128 chip is $3.0\,\mu A$ when operated at the target clock frequency of 100 kHz and a supply voltage of 1.5 V. In comparison, the simulated current consumption using the power estimation tool DIESEL from Philips Semiconductors is $3.3\,\mu A$ for the same clock period and the same supply voltage. These power figures do not include power dissipated by IO pads because they have a separated supply voltage.

The distribution of the current consumption is as follows. The RAM circuit consumes approximately 52% of the power followed by the controller using 18%. The datapath requires the remaining power. The S-Box needs 14%, the MixColumns circuit 8%, and the rest of the circuit 8%. The rest comprehends the output multiplexer, the temporary storage register, the XOR gates, and the Rcon module.

Comparison of some Related Work

This section shows a comparison of the presented work of Feldhofer et al. [9] with the AES modules in [27] and [23]. Note that we compare architectures that are suitable for low power and low die size implementations in this section. Table 3.1 shows that Satoh's circuit requires 5,400 gates on a 0.11 μm technology. The reference circuit needs 3,400 gates. This is 40% less. The work of Mangard et al. has gate count of 8,500. It includes CBC functionality and an AMBA interface on a 0.6 μm technology. Disregarding the CBC functionality, a gate count of 7,000 can be assumed. The reference solution needs only half of the resources.

A reasonable comparison of the maximum clock frequency and the data throughput is difficult to accomplish. The work in [27] reaches 130 MHz—60% faster than the presented circuit. This excellent clock frequency is more due to the more recent process technology rather than to architectural reasons. The critical path of [27] is much longer than the one in [9] because it comprehends a complete S-Box, two large multiplexers, a MixColumns multiplier, and an XOR gate. In comparison, the critical path in [9] is approximately one third because it consists mainly of one half of the pipelined S-Box and some minor logic which is similar than [23].

Unfortunately, many other publications do not include power consumption results since their designs were not manufactured in silicon. The current consumption in [9] should be magnitudes lower than all others because they did not apply any measures for low power consumption.

AES-128 Version	Tech. [μm]	Area [GEs]	Throughput [Mbps]	Max. Frequ. [MHz]	Power [μW]
Feldhofer [9]	0.35	3,400	9.9	80	4.5
Satoh [27]	0.11	5,400	311	130	–
Mangard [23]	0.6	7,000	70	50	–

Table 3.1: Performance comparison of related work.

3.4 High-speed Implementations

The applications of high-speed AES hardware are various. Data rates in the Gigabit range are necessary in network protocols like IPsec or TLS. While in wireless computing the data rates increase fast, traditional wired networks and optical networks still use data rates up to 20 Gbps.

3.4.1 Design Principles

The design methodology for high-speed AES hardware implementations requires a careful evaluation of the different design options. The major optimization goal in this section is high data throughput while the required hardware resources and the power consumption is of minor importance. In contrast, it is not useful to spend large quantities of hardware for a minimal speedup of the circuit. The required performance also depends on the application the AES is designed for. In this section, various methods are highlighted which require attention during the design phase.

The chosen hardware architecture of an implementation mainly determines the properties of the circuit. In addition to optimizations on the architectural level it is important to consider various implementation details to achieve the objectives concerning high data rate. In VLSI design, a common approach is to optimize critical parts of the circuit in a full-custom design. This measure can increase the performance up to a certain level but has the disadvantage of time-consuming development and verification phases. Shrinking of a circuit to the next process technology is not easily possible. Therefore, the decision for a full-custom design should be considered carefully.

The target technology for a piece of hardware is mainly determined by the required number of units. The production costs of an ASIC using a standard-cell technology are often too high for a small number of servers in the high-speed backbone. Therefore, the use of Field Programmable Gate Arrays (FPGAs) is a common alternative. Besides the advantages like the possibility for adaption (changing key size or the used mode of operation) the price per unit for very large FPGAs decreases. For this reason the total costs for development and production might be smaller.

Concepts for System Architecture

The chosen architecture mainly determines the performance of a circuit. Although 32-bit implementations reduce the required hardware resources and the throughput to area ratio is quite comparable to 128-bit architectures, highest data rates in the Gigabit range can only be reached when the calculation is done using 128-bit operations. For the AES round transformation this has the effect that the ShiftRows and InvShiftRows operation comes for free because the bytes are simply rewired. SubBytes and InvSubBytes is a bytewise transformation. Therefore, fully parallel processing is only possible when the S-Box is instantiated 16 times. A similar argument can be given for the MixColumns and InvMixColumns operation where four multiplier instances are required for parallel computing. Round key generation using a 128-bit architecture requires four S-Boxes and other minor logic elements. Optionally, the S-Boxes for data and key unit can be shared.

The AES algorithm requires to calculate a certain round transformation 10, 12, or 14 times depending on the used key size. This allows to implement the algorithm either iteratively or to duplicate a single round multiple times. This measure is called loop unrolling. In an iterative design the whole round transformation is instantiated only once. The same piece of hardware is used for all round transformations while the result is iteratively stored in a register and used as input for the next round. Round unrolling can be done for all rounds or only for several rounds. Loop unrolling is mostly combined with pipelining and sub-pipelining. Pipelining is a technique which is used to increase the performance of digital systems by processing multiple blocks of data in parallel. Pipelining is achieved by placing registers between each round transformation. In general, a pipelined architecture provides the highest throughput because the clock frequency can be increased. Sub-pipelining inserts pipeline stages within the round transformation itself. It decreases the delays between pipeline stages but increases the number of clock cycles that are required to perform an encryption.

Hardware Optimizations for Different Modes of Operation

In literature, pipelining and loop unrolling are said to be the concepts which increases the performance of a circuit per se. That this argument is not necessarily correct should be

explained here in more detail. The recommended modes of operation for a symmetric key encryption algorithm like the AES are defined by NIST [20]. Depending on the application the best known modes are: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR). Other newly standardized modes are Cipher-based MAC (CMAC) and Counter with Cipher Block Chaining-message Authentication Code (CCM) [21, 22]. The two basic classifications can be made whether the encryption algorithm is used in feedback (CBC, CFB, OFB) or non-feedback (ECB, CTR) mode.

In feedback modes the result of the previous encryption is involved in the calculation of the next ciphertext value. It is not possible to start encryption of the next block of data until the result of the previous one is ready. Hence, loop unrolling and pipelining is in general unnecessary because only one round is active at a time and the pipeline never stays filled. The performance gain is only small and hardware resources are wasted. The time from the start of the encryption until the result is available (latency) stays the same. The data rate is determined by the latency of the circuit. An exception to this argument is the encryption or decryption of multiple unrelated data streams. The number of pipeline stages defines the number of independent data streams that can be processed in parallel. This is possible because the result of one stream is not influenced by the other ones. Although inner-round pipelining stages do not improve the performance in feedback mode there are reasons for including them into an architecture anyway. The critical path of the combinational logic of a whole round of the AES might be too long to reach a specified clock frequency. Insertion of a registers shortens the critical path, allows higher clock frequencies, and reduces signal activity. This is a measure to reduce the power consumption of a circuit significantly.

For non-feedback modes like ECB or CTR all above explained hardware optimizations like loop unrolling and inner and outer-round pipelining can be applied. The fastest possible architecture of AES can be reached when all rounds are fully unrolled and many pipelining stages are inserted.

The calculation of the throughput of an AES implementation depends on various arguments. Equation 3.3 explains that the throughput of a circuit depends on the block size and the latency. Due to the fixed block size of 128 bits in the AES algorithm the latency according to Equation 3.4 is to minimize for highest data throughput. The time T_{clk} is the clock period of the circuit and depends on the longest critical path through the circuit. The value $\#Rounds\_per\_block$ is the number of rounds required to calculate one block of data. $\#Pipeline\_stages$ presents the number of pipeline stages in the architecture and $\#Utilized\_stages$ is the number of pipeline stages which can be used in parallel.

$$Throughput = \frac{Block\_size}{Latency} \tag{3.3}$$

$$Latency = \frac{T_{clk} \cdot \#Rounds\_per\_block \cdot \#Pipeline\_stages}{\#Utilized\_stages} \tag{3.4}$$

Table 3.2 shows the performance analysis of different high-speed architectures of the AES algorithm. The first column contains the reference implementation where an iterative approach without pipelining is used. The last two lines show the data throughput rate related to the reference data T_h of the basic architecture. As explained above in non-feedback modes like ECB the more pipeline stages are included the higher is the performance. For feedback modes like CBC the data throughput can not be increased when only a single data stream

is encrypted. For multiple independent blocks of data the same throughput is possible for all modes of operation. The hardware resources are listed in the last row. Thereby, all values are related to A_{ref} which are the requirements of the iterative reference implementation. Additional 128-bit register for pipelining are indicated with R.

Architecture	Iterative	Iterative piped	Sub-unrolling	Sub-unrolling piped	Fully unrolled	Fully unrolled piped
Unrolled rounds	1	1	k	k	10	10
Inner-round stages	0	n	0	n	0	n
$\#Pipeline\_stages$	1	n	k	k·n	10	10·n
$\#Rounds\_per\_block$	10	10	10/k	10/k	1	1
T_{clk}	T_{ref}	T_{ref}/n	T_{ref}	T_{ref}/n	T_{ref}	T_{ref}/n
$\#Utilized\_stages$	1	n	k	k·n	10	10·n
$Latency$	10·T_{ref}	10·T_{ref}/n	10·T_{ref}/k	10·T_{ref}/(k·n)	T_{ref}	T_{ref}/n
$Throughput$ ECB	Th_{ref}	n·Th_{ref}	k·Th_{ref}	k·n·Th_{ref}	10·Th_{ref}	10·n·Th_{ref}
$Throughput$ CBC	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}	Th_{ref}
Resources	A_{ref}	A_{ref}+n·R	k·A_{ref}	k·(A_{ref}+n·R)	10·A_{ref}	10·(A_{ref}+n·R)

Table 3.2: Performance analysis of hardware optimizations.

3.4.2 Related Work

A lot of research work has been done on different high-speed implementations of the AES. Most of them use reconfigurable hardware (FPGAs) as target technology [5, 6, 11, 18, 24, 26, 28, 25, 30, 34, 35]. Only a few concentrate on CMOS standard-cell implementations [10, 12, 29]. The achieved data rates range from 150 Mbps for 32-bit implementations and reach 68 Gbps in 128-bit architectures. Unfortunately, not all of the presented articles find practical applications in real world scenarios. Fully unrolled and pipelined architectures suffer from the fact that they can only be applied in special modes of operation. In the non-feedback modes ECB and CTR it is possible to keep the pipeline fully filled but in general the feedback modes CBC, CFB, and OFB are more practical for encrypting high-speed data streams. Some of related articles even do not implement key scheduling and hence are not functional.

The FPGA implementations of Chodowiec [5] and Pramstaller [24] are 32-bit architectures for encryption and decryption. They have maximum data rates of 150 Mbps and 215 Mbps, respectively. Both use an innovative State representation where storage for two States is used. One State stores the current values and the other State contains the newly calculated output. This has the advantage that ShiftRows is merely addressing the right registers and no State transposition between column and row operations is required. Both implementations calculate key scheduling in advance. Chodowiec [5] uses a serial datapath where four clock cycles are needed per round. The S-Box is implemented in dedicated Block RAMs. MixColumns and InvMixColumns are implemented efficiently by exploiting shared logic using LUTs. This results in an FPGA implementation which uses 222 CLB slices and 3 Block RAMs and has a data throughput of 166 Mbps at a maximum clock frequency of 60 MHz. The implementation of Pramstaller [24] supports CBC mode, has an AMBA APB bus interface, and has a datapath where SubBytes and MixColumns works in parallel. This makes the critical path shorter but

requires ten clock cycles per round. The S-Box circuit [33] and the MixColumns circuit [32] are based on the work of Johannes Wolkerstorfer. The State memory is realized by configuring the CLBs to a synchronous dual-port RAM. This proposed solution requires in total 1,125 CLB slices and no Block RAM. At a clock frequency of 161 MHz the data throughput rate is 215 Mbps for ECB and CBC mode.

One of the first really fast FPGA implementations of AES was presented by McLoone [18]. The fully unrolled 128-bit encryption architecture has, depending on the AES key size, 10, 12, or 14 pipeline stages. After an initialization phase, in every clock cycle one encryption result is ready in non-feedback modes. The matrix multiplication of MixColumns is directly implemented and the S-Box values are stored in Block RAM. The required FPGA resources are 2,222 CLB slices and 100 Block RAMs. the data throughput of 7 Gbps can be reached at a frequency of 54.35 MHz. Additionally, an approach for encryption and decryption is presented. Using 7,576 CLB slices and 102 Block RAMs the throughout of 3,238 Mbps at a frequency of 25.3 MHz can be reached. Saggese et al. [26] presents four 128-bit encryption-only architectures. They compare fully unrolled and deeply pipelined architectures to least area iterative designs. Modes of operations are addressed as they suggest to interleave multiple data streams for keeping the pipeline in feedback modes fully filled. On-the-fly round key generation is applied and the S-Boxes are implemented in Block RAMs. The MixColumns circuit is a net of XOR gates. The iterative approach without pipelining has a data throughput of 1 Gbps at a frequency of 79 MHz and requires 446 CLBs and 10 Block RAMs. The fastest implementation has five pipeline stages per round and is fully unrolled. A throughput of 20.3 Gbps at a frequency of 158 MHz is possible using 5810 CLBs and 100 Block RAMs. In 2004, Hodjat [11] also presented an unrolled and fully pipelined implementation of the AES encryption on a 128-bit architecture. Four pipeline stages are used for each round. The byte substitution of the S-Box circuit is implemented using GF(2^4) operations which uses three stages of pipeline registers. Four MixColumns multiplier are used per round where each consists of a chain of XORs. The round key generation is not addressed in this work. The maximum data throughput rate is 21.54 Gbps at a clock frequency of 168 MHz and can only be achieved in non-feedback modes. The required resources are 12450 CLB slices a no Block RAM. Further interesting publications of AES for FPGAs can be found in [25, 28, 30, 34, 35].

Publications of high-speed CMOS implementations are rare. The first 128-bit implementation of a manufactured chip is from Verbauwhede [29] in 2003. Their encryption-only architecture has no fixed block size as defined in the AES standard but implements the Rijndael algorithm with key and data block size of 128, 192, and 256 bits. The fully parallel, non-pipelined version requires one clock cycle per round. The architecture has 32 S-Box instances for the encryption unit and 16 instances in the key-scheduling part which are directly implemented according to the AES standard [19]. The MixColumns multiplication circuit is segmented into XOR operations and instantiated four times for each column. The on-the-fly key scheduling supports all key sizes. The implementation was done on a 0.18 μm CMOS standard-cell technology and has a maximum clock frequency of 154 MHz at a supply voltage of 1.8 V. For the data block size of 128 bits the data throughput is 1.6 Gbps while the chip area is 3.96 mm^2 which compares to 173 K gates. The same authors describe in [10] a new version of a non-pipelined, iterative, encryption-only architecture. It has a data throughput of 3.83 Gbps using a chip area of 0.79 mm^2 also on a 0.18 μm CMOS standard-cell technology. In the work of Hodjat [12], the fastest AES implementation is presented which achieves data throughput rates of up to 70 Gbps. The application of such an AES core might be in optical links where a data stream is encrypted using the non-feedback CTR mode. Different concepts

with on-the-fly round key generation and pre-computing of round keys are presented. They used a fully unrolled and pipelined concept with four inner-loop pipelining stages. The 32 instances of the S-Box per round are three-stage pipelined and use the concept of composite field computations [33]. Four MixColumns multipliers are used per round and implemented as chains of XOR gates. The maximum achieved data rate is 68 Gbps with a chip area of 250 K gates on a 0.18 μm CMOS standard-cell technology.

3.4.3 Implementation Details

The possible implementations of the AES for high-speed requirements are manifold. According to the considerations above two different high-speed architectures of the AES algorithm were implemented. The first implementation can be used for non-feedback and feedback modes of operations. It is an iterative approach for encryption and decryption targeted for standard cell technology. The second implementation is a fully unrolled architecture for maximum data throughput for non-feedback modes of operations. The target technology is an FPGA. Both architectures are designed for 128-bit key size and described in the following.

Iterative Encryption and Decryption AES Architecture

The target applications of high-speed AES hardware modules implemented with standard-cell technology are for example IPsec solutions in mass products like routers. The analysis of the requirements for different Internet protocols led to an architecture which is a tradeoff between data throughput and die size. The support for all modes of operations (feedback and non-feedback) was a major design goal while the number of independent data streams is variable. While other implementations can only compute encryption, the presented architecture also supports decryption with on-the-fly key scheduling.

The maximum data throughput can be achieved when the bit width of the AES architecture is 128 bits. The iterative approach allows the computation of one AES round per clock cycle. This brings the maximum hardware utilization while the required hardware resources are relatively small compared to an unrolled architecture. Besides the architecture for the data unit, the key unit implements the key scheduling. Here also one round key is generated per clock cycle. The architecture of the data unit and the key unit can be seen in Figure 3.5 and Figure 3.7 and are describe in detail in the following.

Data Unit The data unit of the AES module contains combinational logic to calculate the AES transformations SubBytes, ShiftRows, MixColumns, AddRoundKey, and their inverse operations InvSubBytes, InvShiftRows, InvMixColumns. Figure 3.5 shows the interconnection of the different parts in the circuit which allows the calculation of one round in a single step.

The SubBytes transformation operates independently on each byte of the State using a substitution table (S-Box). Sixteen instances of the S-Box circuit are used in this architecture. Each one is realized using a composite field implementation according to Wolkerstorfer [33], see Figure 3.6. For encryption a combination of the multiplicative inverse in the finite field GF(2^8) and an affine transformation is used. Decryption works similar but the inverse affine transformation is applied before the multiplicative inverse. In 128-bit architectures the ShiftRows and InvShiftRows transformations degrade to a simple rewiring where a multiplexer selects whether shift left for encryption or shift right for decryption are applied. As the

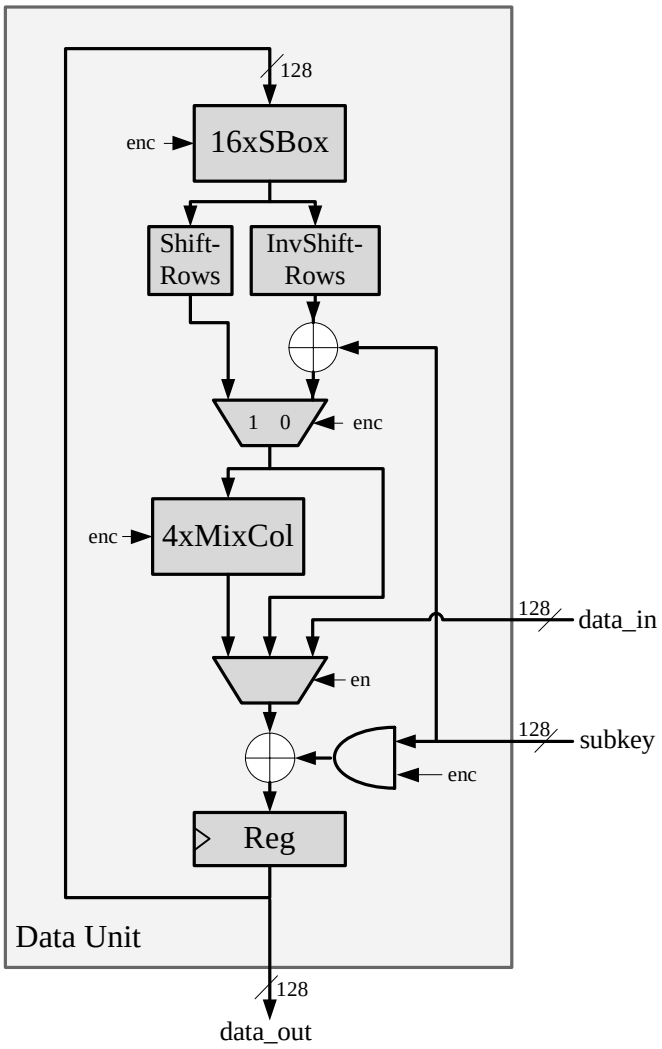

Figure 3.5: Data unit of 128-bit AES architecture.

sequence for AddRoundKey and MixColumns are exchanged for encryption and decryption, the XOR operation with the key follows for decryption immediately after the InvShiftRows operation while for encryption it is applied as a last step in the round transformation. A multiplexer circuit is used which allows to select the output of the MixColumns circuit, skip this MixColumns circuit, or using the data input port. The MixColumns and the InvMixColumns operations are implemented using four multipliers. Each of it transforms one 32-bit column of the State to the appropriate 32-bit output of this column. The implementation of the multiplier is based on the work of Wolkerstorfer [32]. Here some improvements are introduced where some terms in the derivation can be reused.

Key Unit The key unit performs the key-scheduling algorithm. Starting with the secret key as input it delivers one round key per clock cycle. For decryption either the last round key has to be supplied or the last key is calculated during a pre-computation phase of ten cycles. In each step of the key schedule the first 32-bit word is rotated and the S-Box lookup

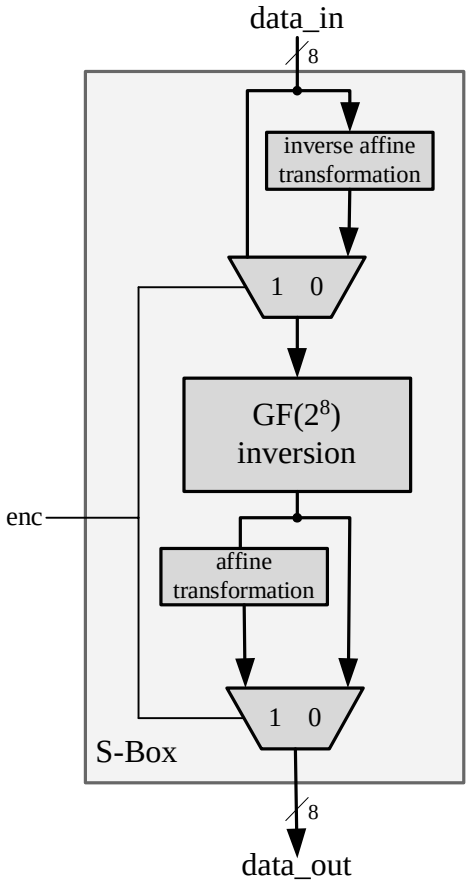

Figure 3.6: AES S-Box circuit using composite field arithmetic.

has to be applied. Therefore four instances of the S-Box are required. A constant called Rcon is also added to the output of the S-Box circuit with an XOR gate. This result is combined with the last 32-bit word with an XOR. The other three new 32-bit words are calculated from the old values and an XOR operation with the other inputs according to the algorithm. The different order of the inputs for the XOR gates of encryption and decryption is realized using multiplexers. One 128-bit register stores the actual round key.

Fully Unrolled AES Architecture

For applications in backend routers or in server where many data streams have to be encrypted simultaneously data rates of Gigabits are necessary. Because of the small number of such high-speed modules FPGAs are commonly used as target technology. The highest data throughput for AES encryption can be reached when all ten rounds are unrolled and pipelining registers are introduced between each round. Therefore the fully unrolled architecture which can be seen in Figure 3.8 is proposed. Because of this unrolled architecture only non-feedback modes of operation like CTR mode or the encryption of independent data streams are useful. The decision was made to implement an encryption-only module because in CTR mode decryption can also be done with this module.

The architecture presented in Figure 3.8 consists of an initial round which is an XOR gate

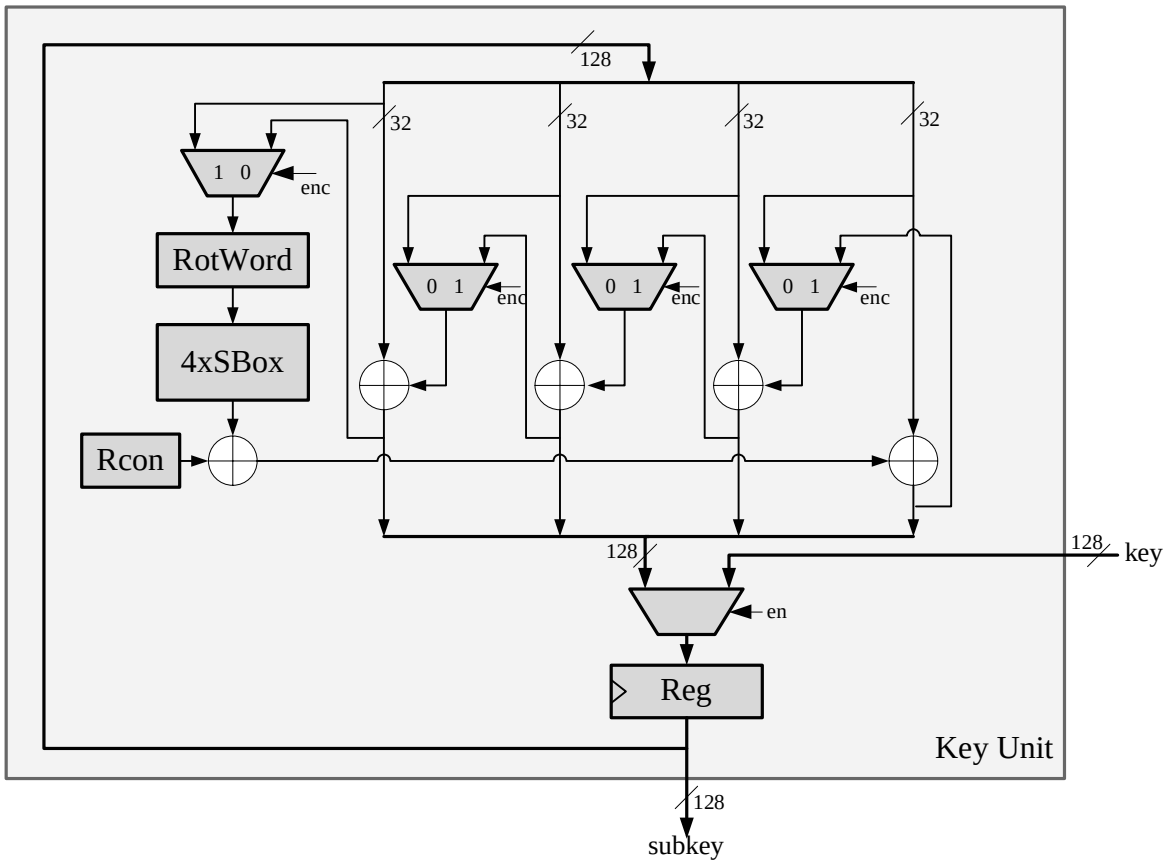

Figure 3.7: Key unit of 128-bit AES architecture.

of the original key with the data input followed by nine instances of the round transformation. The last round instance does not include the MixColumns circuit because the operation is not required in the last round. Because of the FPGA target technology the 16 S-Box circuits are realized as a table lookup in dedicated Block RAMs. The ShiftRows operation is again a fixed rewiring from the outputs of the S-Box to the input of the following circuit. The MixColumns architecture is the same as explained above but realized in LUTs. The ten different round keys have to be generated in advance and stored on the FPGA in Block RAMs.

3.4.4 Characteristics of High-speed Implementations

Depending on the desired target technology for FPGAs or standard cell technology the results are measured in LUTs, Block RAMs or in gate equivalents. The comparison is therefore only possible for the same target technology. In the following the presented design is compared with some related work.

Chip Area Estimation for Standard Cell Approach

The chip area estimation for the iterative standard cell solution is shown in Table 3.3. It is based on the synthesis results of the individual parts. It does not include the clock tree nor the placement overhead. The total chip area for the data unit and the key unit is approximately

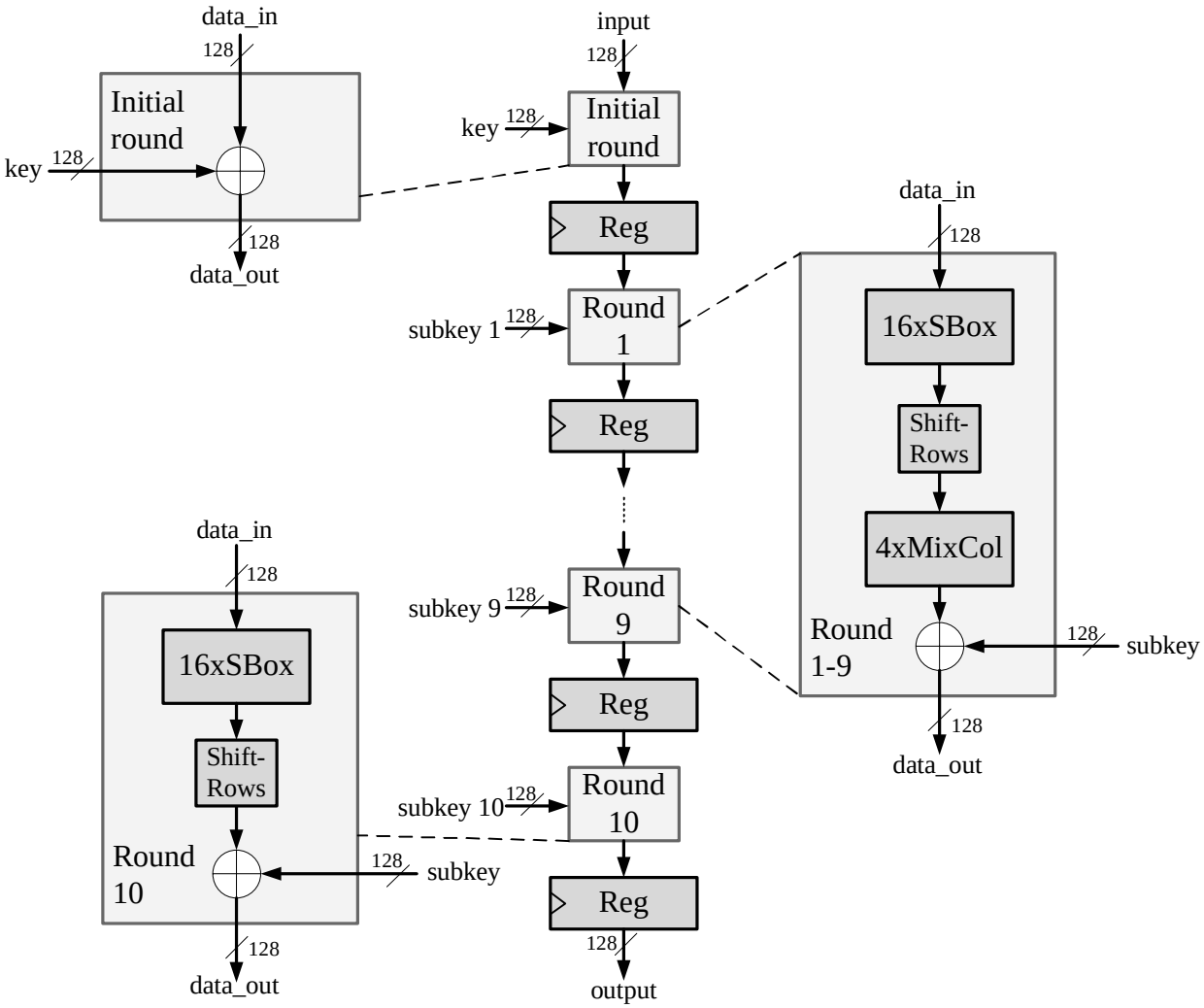

Figure 3.8: Fully unrolled AES architecture.

20,000 gate equivalents. In comparison Hodjat [10] presents an iterative, encryption-only AES processor with a gate count of 34,300 GEs. This approach includes a circuit for modes of operations and has higher data throughput because the 0.18 μm CMOS technology is faster than the 0.35 μm CMOS process from Austriamicrosystems.

Comparison of some Related Work for FPGAs

The architecture of an AES implementation mainly defines the required hardware resources on an FPGA. Additionally, the used synthesis tool and the target device influences this result. Table 3.4 gives an overview of existing FPGA solutions. Because of the different FPGAs, most of the use XILINX FPGAs, the values have to be seen as a relative comparison of resource requirements and data throughput.

	Area/Instance [GEs]	#Instances	Area [GEs]
S-Box	473	20	9,460
MixColumns	582	4	2,328
Flip flops	6	256	1,536
Multiplexer	3	512	1,536
XOR	3	384	1,152
misc.	1	4,000	4,000
AES iterative		**Total**	≈20,000
Hodjat [10]			34,300

Table 3.3: Chip area estimation of iterative AES architecture.

Authors	LUTs	Block RAMs	Throughput [Gbps]
Chodowiec [5]	222	3	0.166
Chodowiec [6]	12,600	80	12.16
Chodowiec [6]	2,057	8	1.265
Chodowiec [6]	2,507	0	0.414
Hodjat [11]	9,446	0	21.64
Hodjat [11]	5,177	84	21.54
McLoone [18]	2,222	100	7.0
Pramstaller [24]	1,125	0	0.215
Rouvroy [25]	146	3	0.358
Saggese [26]	446	10	1.0
Saggese [26]	648	10	1.82
Saggese [26]	2,778	100	8.9
Saggese [26]	5,810	100	20.3
Standaert [28]	1,769	0	2.085
Standaert [28]	15,112	0	18.560
Wang [30]	1,857	0	1.604
Zambreno [34]	387	10	1.41
Zambreno [34]	1,254	20	4.44
Zambreno [34]	2,206	50	10.88
Zambreno [34]	3,766	100	22.93
Zambreno [34]	16,938	0	23.57
Zhang [35]	9,406	0	11.965
Zhang [35]	11,022	0	21.556

Table 3.4: Resource requirements and performance of different AES architectures for FPGAs.

Chapter 4

Conclusions and Future Work

In this deliverable we have surveyed hardware architectures that are suitable for cryptographic applications. In particular, we have analyzed various hardware implementations of the AES algorithm. The focus of this survey has been on throughput-optimized circuits and on circuits designed for operation in very constricted environments where the power budget and the silicon area are sparse resources.

In order to provide a profound analysis of existing AES hardware we have describe the AES algorithm briefly. The analysis of existing AES hardware has shown considerations for light-weight implementations and for high-performance implementations. For both implementation goals, pointers and references to state-of-the-art implementations have been provided. Concepts and design considerations behind these implementations, which allow to push the limits of AES hardware implementations, have been summarized in a compact manner. We have shown that for both implementation goals practical solutions do exists. Hence, there state-of-the-art hardware architectures of AES that are suitable for high-speed applications, and there are state-of-the-art hardware architectures of AES that are suitable for low power applications, such as RFID tags.

This deliverable has been the first of two deliverables dealing with hardware architectures. In the second deliverable, we will focus on hardware architectures for public-key cryptosystems. In particular we will provide a summary of high-speed and of lightweight implementations of elliptic curve based cryptographic algorithms.

Bibliography

[1] Ross J. Anderson and Markus G. Kuhn. Tamper Resistance - a Cautionary Note. In *Second Usenix Workshop on Electronic Commerce*, pages 1–11, November 1996.

[2] Leijla Batina, Kerstin Lemke, Elke de Mulder, Nele Mentens, Elisabeth Oswald, Eric Peeters, and François-Xavier Standaert. D.VAM.5—Side-channel attacks on FPGAs, August 2005.

[3] Lejla Batina, Elke De Mulder, Kerstin Lemke, Stefan Mangard, Elisabeth Oswald, Gilles Piret, and François-Xavier Standaert. D.VAM.4—State of the art of side-channel attacks other than power and timing attacks, August 2005.

[4] Mike Bond. Attacks on Cryptoprocessor Transaction Sets. In Çetin Kaya Koç and David Naccache and Christof Paar, editor, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2001.

[5] Pawel Chodowiec and Kris Gaj. Very Compact FPGA Implementation of the AES Algorithm. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2003.

[6] Pawel Chodowiec, Po Khuon, and Kris Gaj. Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining. In *ACM/SIGDA ninth international symposium on Field programmable gate arrays - FPGA 2001, Montery, California, USA, 2001, Proceedings*, pages 94–102. ACM Press, 2001.

[7] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.

[8] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems using the AES Algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.

[9] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings on Information Security, to appear*, November 2005.

[10] Alireza Hodjat, David Hwang, Bo-Cheng Lai, Kris Tiri, and Ingrid Verbauwhede. A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-um CMOS Technology. In John Lach, Gang Qu, and Yehea I. Ismail, editors, *Proceedings of the 15th ACM Great Lakes Symposium on VLSI 2005, Chicago, Illinois, USA, April 17-19, 2005*, pages 60–63. ACM, ACM Press, April 2005.

[11] Alireza Hodjat and Ingrid Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. In *12th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004), 20-23 April 2003, Napa, CA, Proceedings*, pages 308–309. IEEE Computer Society, 2004.

[12] Alireza Hodjat and Ingrid Verbauwhede. Minimum Area Cost for a 30 to 70 Gbits/s AES Processor. In *2004 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2004), Emerging Trends in VLSI Systems Design, 19-20 February, 2004, Lafayette, LA, USA*, pages 83–88. IEEE Computer Society, 2004.

[13] International Organisation for Standardization (ISO). ISO/IEC 7816: Identification cards - Integrated circuit(s) cards with contacts, 1989.

[14] International Organization for Standardization (ISO). ISO/IEC 18000-3: Information Technology AIDC Techniques — RFID for Item Management, March 2003.

[15] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer, 1996.

[16] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[17] Stefan Mangard, Manfred Aigner, and Sandra Dominikus. A Highly Regular and Scalable AES Hardware Architecture. *IEEE Transactions on Computers*, 52(4):483–491, April 2003.

[18] Màire McLoone and John V. McCanny. High Performance Single-Chip FPGA Rijndael Algorithm Implementations. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2001.

[19] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. Available online at `http://www.itl.nist.gov/fipspubs/`.

[20] National Institute of Standards and Technology (NIST). Special Publication 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation - Methods and Techniques, December 2001. Available online at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`.

[21] National Institute of Standards and Technology (NIST). Special Publication 800-38C 2004, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004. Available online at `http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf`.

[22] National Institute of Standards and Technology (NIST). Special Publication 800-38B 2005, Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005. Available online at `http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf`.

[23] Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer. Efficient AES Implementations on ASICs and FPGAs. In H. Dobbertin, Vincent Rijmen, and A. Sowa, editors, *Proceedings of the Fourth Workshop on the Advanced Encryption Standard, AES4 - State of the Crypto Analysis', Bonn, Germany, May 10-12, 2005.*, volume 3373 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2004.

[24] Norbert Pramstaller and Johannes Wolkerstorfer. A Universal and Efficient AES Co-Processor for Field Programmable Logic Arrays. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Antwerp, Belgium, August 30 - September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 565–574. Springer, August 2004.

[25] Gaël Rouvroy, François-Xavier Standaert, Jean-Didier Legat, and Jean-Jacques Quisquater. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications. In *International Conference on Information Technology: Coding and Computing (ITCC 2004), Special Session on Embedded Cryptographic Hardware, Washington, DC, USA, 5-7 April, 2004, Proceedings*, volume 2, pages 583–587. IEEE Computer Society, 2004.

[26] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio G. M. Strollo. An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm. In Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, editors, *Field Programmable Logic and Application, 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003, Proceedings*, volume 2778 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2003.

[27] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.

[28] François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 334–350. Springer, 2003.

[29] Ingrid Verbauwhede, Patrick Schaumont, and Henry Kuo. Design and Performance Testing of a 2.29 Gb/s Rijndael Processor. *IEEE Journal of Solid-State Circuits*, 38(3):569–572, March 2003.

[30] Shuenn-Shyang Wang and Wan-Sheng Ni. An Efficient FPGA Implementation of Advanced Encryption Standard Algorithm. In *International Symposium on Circuits and Systems (ISCAS 2004), Vancouver, British Columbia, Canada, May 23-26, 2004, Proceedings.*, volume 2, pages 597–600. IEEE Computer Society, May 2004.

[31] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design - A Systems Perspective*. Addison-Wesley, 2nd edition, 1993. ISBN 0-201-53376-6.

[32] Johannes Wolkerstorfer. An ASIC Implementation of the AES-MixColumn operation. In Peter Rössler and Andreas Döderlein, editors, *Austrochip 2001*, pages 129–132, 2001. ISBN 3-9501517-0-2.

[33] Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES SBoxes. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference 2002, San Jose, CA, USA, February 18-22, 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2002.

[34] Joseph Zambreno, David Nguyen, and Alok N. Choudhary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings.*, volume 3203 of *Lecture Notes in Computer Science*, pages 575–585. Springer, 2004.

[35] Xinmiao Zhang and Keshab K. Parhiter. High-Speed VLSI Architectures for the AES Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(9):957–967, September 2004.