# A Highly Regular and Scalable AES Hardware Architecture

Stefan Mangard, *Student Member*, *IEEE*, Manfred Aigner, and Sandra Dominikus

**Abstract**—This article presents a highly regular and scalable AES hardware architecture, suited for full-custom as well as for semi-custom design flows. Contrary to other publications, a complete architecture (even including CBC mode) that is scalable in terms of throughput and in terms of the used key size is described. Similarities of encryption and decryption are utilized to provide a high level of performance using only a relatively small area (10,799 gate equivalents for the standard configuration). This performance is reached by balancing the combinational paths of the design. No other published AES hardware architecture provides similar balancing or a comparable regularity. Implementations of the fastest configuration of the architecture provide a throughput of 241 Mbits/sec on a 0.6 $\mu$m CMOS process using standard cells.

**Index Terms**—Advanced Encryption Standard (AES), hardware architecture, IP module, VLSI, scalability, regularity.

## 1 Introduction

The symmetric block cipher Rijndael [1] was standardized by NIST[1] as Advanced Encryption Standard (AES) [2] in November 2001. Being the successor of the Data Encryption Standard (DES) [3], the AES is used in a wide range of applications.

The AES is the preferred algorithm for implementations of cryptographic protocols that are based on a symmetric cipher. It is not only used to secure data transfers between small, mobile consumer products, but it is also used in high-end servers. Consequently, the requirements for implementations of the AES differ significantly.

Applications with strict requirements concerning performance, power consumption, or side-channel leakage are, in practice, usually implemented by dedicated hardware. Hardware implementations of the AES are, for example, used in Internet servers as performance accelerators or in smart cards (besides other reasons) to increase the resistance against side-channel attacks.

Due to the practical importance of hardware implementations, the different AES candidates were implemented and compared on FPGAs (see [4], [5], and [6]) and on ASICs [7] before Rijndael was finally selected to become the AES. After this selection, more effort was dedicated toward the development of efficient hardware implementations of this particular algorithm (see [8], [9], [10], and [11]). The most recent proposal for an ASIC architecture of the AES is [12]. However, this architecture has very unbalanced combinational paths and requires a time and area-consuming selector function, which is not part of the actual AES algorithm.

1. National Institute of Standards and Technology.

• *The authors are with the Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria.*
*E-mail: {stefan.mangard, manfred.aigner, sandra.dominikus}@iaik.at.*

*Manuscript received 15 June 2002; revised 2 Dec. 2002; accepted 2 Dec. 2002. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117871.*

This article presents a highly regular and scalable AES hardware architecture that requires only 10,799 gate equivalents to provide a throughput of 128 Mbits/sec (for AES-128 encryption and decryption) on a 0.6 $\mu$m standard cell library. These numbers include an AMBA APB bus interface, a CBC register, and a key storage register.

The architecture uses similarities of encryption and decryption to provide a high level of performance while keeping the chip size small. The high performance is especially reached by keeping combinational paths balanced so that every clock cycle is fully utilized. The fact that the combinational paths are short compared to other published AES architectures makes the presented architecture a favorable choice for low-power applications. This is due to the fact that glitches, which occur more frequently in long combinational paths than in short ones, cause a significant power consumption.

Besides the small area requirements and the high performance, the presented architecture has another important property: It is highly regular. This helps to keep the size of the AES architecture small during place-and-route of a semi-custom design flow and facilitates the creation of full-custom designs. Full-custom approaches are particularly interesting for smart card implementations that are required to provide protection against power analysis attacks [13]. In a full-custom approach, the designer can balance the capacitive loads of differential nodes well as it is, for example, desired for logic styles like the one described in [14].

Another very important property of the presented architecture is its scalability. The performance of the architecture can be increased gradually at the cost of an increased chip size. Furthermore, the key size can easily be changed from 128 to 192 or 256 bits. However, the overall architecture does not change for versions with different performance and key sizes.

Section 2 gives a brief overview of the AES algorithm. In Section 3, the AES hardware architecture and the corresponding implementation options are described. The

| | | | |
|---|---|---|---|
| $D_{0,0}$ | $D_{0,1}$ | $D_{0,2}$ | $D_{0,3}$ |
| $D_{1,0}$ | $D_{1,1}$ | $D_{1,2}$ | $D_{1,3}$ |
| $D_{2,0}$ | $D_{2,1}$ | $D_{2,2}$ | $D_{2,3}$ |
| $D_{3,0}$ | $D_{3,1}$ | $D_{3,2}$ | $D_{3,3}$ |

Fig. 1. Alignment of an AES state.

performance of the architecture is summarized and compared with other AES hardware implementations in Section 4. Concluding remarks can be found in Section 5.

## 2 AES Algorithm

The AES is a round-based, symmetric block cipher. It is defined for a block size of 128 bits and key lengths of 128, 192, and 256 bits. According to the key length, these variants of the AES are called AES-128, AES-192, and AES-256. This article mainly focuses on implementing the AES-128, which is the most commonly used AES variant. However, the presented architecture can also be used for the other standardized key sizes.

The following subsection describes the AES transformations, which are the building blocks of AES encryptions and decryptions. In Section 2.2, the AES-128 key expansion is discussed.

### 2.1 AES Transformations

The AES takes a 128-bit data block as input and performs several different transformations on this block. In case of an encryption, the input block of the AES is called plaintext and the returned block is called ciphertext. All intermediate results of this block, as well as the input and the output block, are called states. For a discussion of the different transformations, executed on the 128-bit states in an AES encryption or decryption, it is best to picture a state as a 4-by-4 matrix of bytes (see Fig. 1). A 128-bit input/output block of the AES is mapped to an AES state by putting the first byte of the block in the upper left corner of the matrix and by filling in the remaining bytes column by column.

AES encryptions and decryptions are based on four different transformations that are performed repeatedly in a certain sequence. Each of these transformations, which are described in the following, maps a 128-bit input state to a 128-bit output state.

- **SubBytes:** The SubBytes transformation is a nonlinear substitution operation that works on bytes. Each byte of the input state is replaced using the same substitution function (called S-Box).

  The S-Box is defined as the multiplicative inverse in the Galois Field $GF(2^8)$ with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ followed by an affine transformation. The InvSubBytes transformation, which is needed for decryption, is the inverse of the affine transformation followed by the same inversion as in the SubBytes transformation.

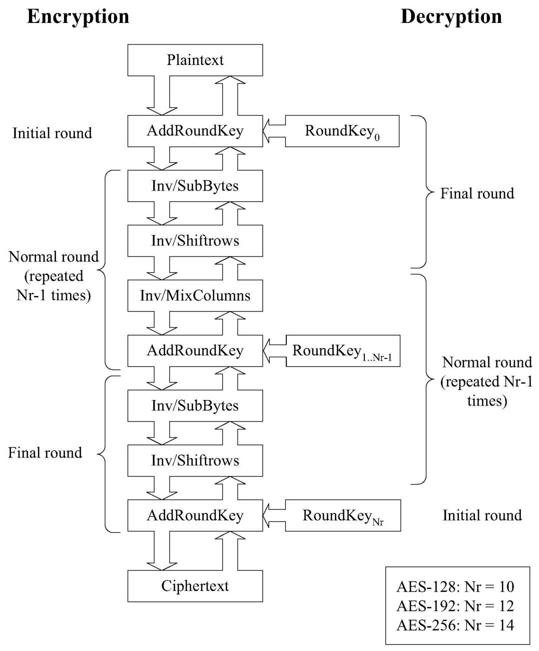


Fig. 2. Sequence of the execution of the four different transformations used in an AES encryption/decryption.

- **ShiftRows:** The ShiftRows transformation rotates each row of the input state to the left, whereby the offset of the rotation corresponds to the row number. For example, row one (the row consisting of the elements $D_{1,0}$, $D_{1,1}$, $D_{1,2}$, and $D_{1,3}$) is rotated by one position to the left. The inverse of this transformation is computed by performing the corresponding rotations to the right.
- **MixColumns:** The MixColumns transformation maps each column of the input state to a new column in the output state. Each input column is considered as a polynomial over $GF(2^8)$ and multiplied with the constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo $x^4 - 1$. The coefficients of $a(x)$ are also elements of $GF(2^8)$ and are represented by hexadecimal values in this equation. The InvMixColumns transformation is the multiplication of each column with $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ modulo $x^4 - 1$.
- **AddRoundKey:** The AddRoundKey transformation is self-inverting. It maps a 128-bit input state to a 128-bit output state by xoring the input state with a 128-bit round key.

These transformations are applied to a 128-bit input block in a certain sequence to perform an AES encryption or decryption. In both cases, the transformations are grouped to so-called rounds. There are three different types of rounds, namely, the initial round, the normal round, and the final round. The transformations of the different rounds and the sequence of the rounds are shown in Fig. 2. The rounds are slightly different for encryption and decryption and the number of rounds, $Nr$, depends on the key size.

The presented decryption algorithm is called Inverse Cipher. Compared to the encryption algorithm, it is simply

```
RC[1..10] = ('01','02','04','08','10',
             '20','40','80','1B','36')
Rcon[i]   = (RC[i],'00','00','00')

for(i = 0; i < 4; i++)
  W[i] = (key[0], key[1], key[2], key[3])

for(i = 4; i < 44; i++) {
  temp = W[i - 1]
  if (i mod 4 == 0)
    temp = SubWord(RotWord(temp)) xor Rcon[i / 4]
  W[i] = W[i - 4] xor temp
}
```

Fig. 3. Pseudocode for the AES-128 key expansion.

the execution of the inverse transformations in reversed order. Alternatively, the so-called Equivalent Inverse Cipher can be used for decryption. However, for the presented AES hardware architecture, the Inverse Cipher is more suitable.

### 2.2 AES-128 Key Expansion

For an AES-128 encryption, the 128-bit cipher key needs to be expanded to eleven 128-bit round keys. The principle idea of this key expansion is that the first round key, $Roundkey_0$, corresponds to the cipher key. All subsequent round keys are derived from their respective predecessor using a function $f$. So, $Roundkey_i = f(Roundkey_{i-1})$ for all $0 < i < 11$.

For an AES-128 decryption, the same round keys are used in reversed order. Using the inverse of the key expansion function, $f^{-1}$, the round keys can be derived recursively from $RoundKey_{10}$.

In Fig. 3, a pseudocode for the AES-128 key expansion is shown. This pseudocode is based on 32-bit key words and, so, the eleven 128-bit round keys are stored one after the other in the word array `W[0..43]`. The RotWord function, used in the pseudocode, rotates the input word by one byte to the left. The SubWord function applies the S-Box function to each byte of the input word. The RC values, finally, are the powers $x^{i-1}$ of $x$ in the same Galois field $GF(2^8)$ as used for the S-Box transformation.

Fig. 4 shows how the word array `W[0..43]` is mapped to the corresponding round keys. The key expansions for the AES-192 and for the AES-256 are very similar and described in detail in [2].

## 3 AES Hardware Architecture

The AES hardware architecture presented in this article is very modular and provides a high level of scalability. While the standard version of the architecture is suited for smart

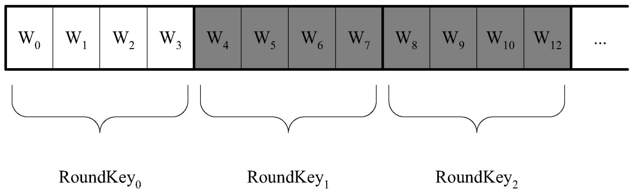


Fig. 4. Mapping of the key words to round keys.

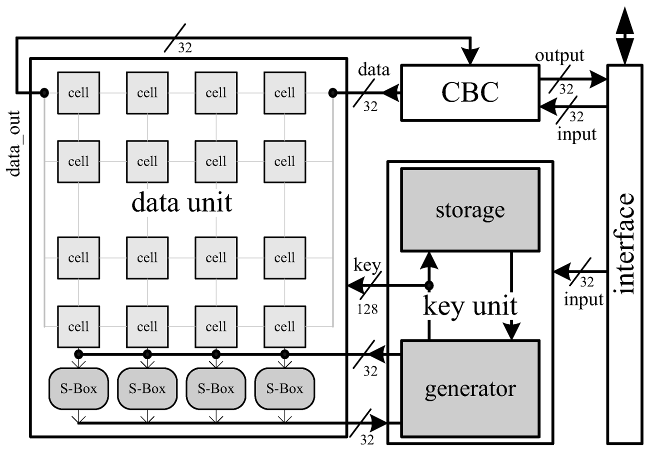


Fig. 5. Overall structure of the AES module.

cards, USB dongles, and similar devices, the high-performance version provides enough throughput to be used as an acceleration module in high-end servers. It is important to outline that, in both versions, the overall structure of the architecture remains the same—even for different key sizes.

This overall structure of the architecture, which is capable of performing AES encryptions and decryptions, is shown in Fig. 5. The AES hardware module consists of the following four components:

- **Interface:** The interface handles all communication of the AES module with its environment—it communicates based on 32-bit words with the other components of the AES module and via an AMBA APB bus with the environment of the module.
- **Data Unit:** The data unit is the main module of the architecture. It can perform any kind of AES encryption or decryption round using the round key that is assigned to its key input. Although the number of rounds is different for the three standardized key sizes, the types of rounds that are executed are always the same. Consequently, the data unit is independent of the key size.

  The data unit has a highly regular structure, as indicated in Fig. 5. It consists of 16 instances of a so-called data cell and a certain number of S-Boxes. The more S-Boxes are used, the higher is the performance of the AES module. The standard version of the data unit has four S-Boxes and is described in detail in Section 3.1. A high-performance version with 16 S-Boxes is presented in Section 3.2. In principle, it is also possible to implement a data unit with eight S-Boxes. This version can easily be derived from the description of the other two versions and is not presented separately.
- **Key Unit:** The key unit serves two main purposes: the storage of cipher keys and the calculation of the round keys. To save die size, the S-Boxes of the data unit are reused to perform the key expansion. In the presented architecture, this reuse is possible for any key size without loss of performance.

  Since 128 bit is currently the most commonly used key size, a key unit capable of performing the 128-bit key expansion is described in detail in this article (see Section 3.3). The overall structure of the AES module, however, allows the usage of key modules supporting multiple key sizes in parallel or any of the standardized key sizes on its own.

- **CBC Unit:** An AES module just consisting of a key unit, a data unit, and an interface can already perform the AES algorithm in ECB (Electronic Code Book) mode. However, because there exist certain attacks (e.g., reordering of blocks) against this mode, usually other modes of operation [15] are used. The most popular one is the CBC (Cipher Block Chaining) mode, where the result of an AES encryption is xored with the next 128-bit input block. This procedure needs to be reversed when performing a decryption. The CBC unit of the AES module implements the CBC mode without any negative influence on the overall performance of the AES module.

In the presented architecture, a 128-bit block of data is encrypted as follows: First, a cipher key needs to be loaded via the interface into the key unit. Once a key is loaded, it can be used for an arbitrary number of encryptions and decryptions. After loading the cipher key, the first 128-bit block of data is transferred via the interface and the CBC unit into the data unit. The data unit then iteratively performs the number of AES rounds that are required for the used key size.

In each round, the key unit provides the corresponding round key to the data unit. To calculate these round keys, the key unit uses the S-Boxes of the data unit during a clock cycle in which they are not used by the data unit. After the calculation of the AES rounds, the encryption result is passed in 32-bit words to the interface via the CBC unit. Decryptions are computed in a very similar way. In this case, the data unit performs the inverse AES transformations in reversed order and also the key unit provides the round keys in reversed order.

The remainder of this section presents the details of the standard data unit and those of the high-performance data unit. Additionally, an AES-128 key unit that can be used with both data units is described.

## 3.1 Standard Data Unit

The data unit is the biggest and the most important component of the AES architecture. It stores the current 128-bit state (see Fig. 1) of an encryption or decryption and is capable of performing any number and type of encryption/decryption rounds on this state. Consequently, all four AES transformations (SubBytes, ShiftRows, MixColumns, and AddRoundKey) and the corresponding inverse transformations are implemented within the data unit. For the AddRoundKey transformation, a round key needs to be provided by the key unit.

Fig. 6 shows the standard version of the data unit. Its structure is highly regular and closely related to the definition of the AES state. The standard data unit consists of 16 so-called data cells and four S-Boxes. An S-Box of the architecture is a circuit capable of performing the S-Box and the inverse S-Box transformation for an 8-bit input. The data

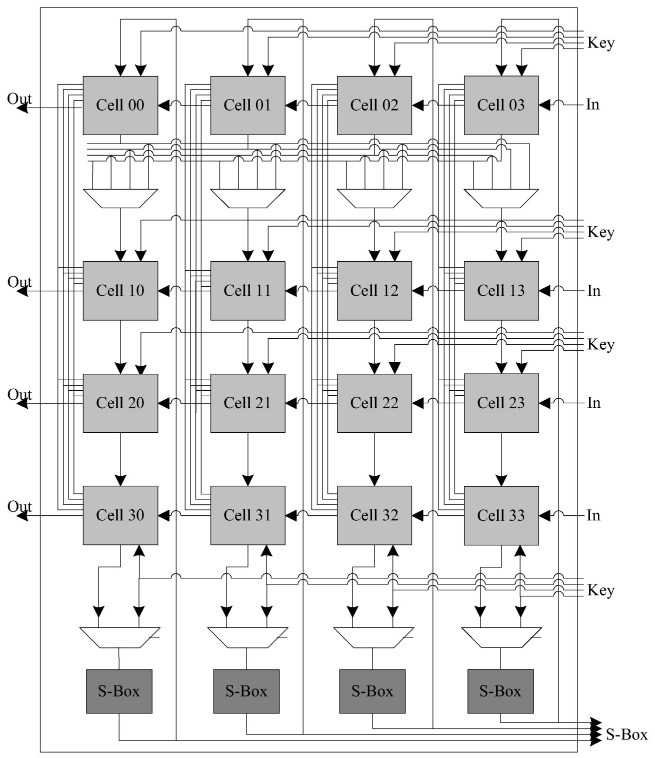


Fig. 6. Architecture of the standard data unit.

cells store eight bits per cell and perform all other AES transformations and the corresponding inverses, when connected appropriately. In full-custom designs, inputs and outputs of the data cells can be defined in a way that connection by abutment is possible when they are placed next to another.

However, the regular design not only facilitates full-custom designs. Also, for FPGA and standard-cell synthesis, a regular circuit is very desirable. If one improves the synthesis results of a single data cell by special attributes for the synthesizer, the overall area reduction is 16 times higher and therefore worth the effort.

Another distinguishing feature of the presented architecture is the fact that the combinational paths are relatively short and, what is even more important, very balanced. The commonly used approach to implement the AES in hardware is to store the 128-bit state in a register and to perform the AES transformations (except for the ShiftRows transformation) column by column. So, in order to perform a normal AES encryption round, first the ShiftRows transformation is done in one clock cycle. Then, the remaining transformations of an AES round are done column by column, whereby all transformations for one column are usually done within one clock cycle.

The problem of this approach is that the combinational path to perform a SubBytes, a MixColumn, and an AddRoundKey transformation in one clock cycle is very long. Additionally, the implementation of the ShiftRows transformation causes a significant wiring overhead. The data unit, presented in this section, solves both problems. It performs AES encryptions and decryptions in the following way:

In order to load a data block, the input data is shifted column by column from the right side (see Fig. 6) into the data cells. The inputs labeled "In" are connected via the CBC unit to the interface. The initial AddRoundKey transformation is done in the fourth clock cycle at the same time as the last column is loaded.

To compute a normal AES round, the registers are rotated vertically to perform the Inv-/SubBytes and the Inv-/ShiftRows transformation row by row. In the first clock cycle, the Inv-/SubBytes transformation starts for row three. Due to the fact that the implementation of the S-Boxes is pipelined (this will be motivated in Section 3.1.1), the result of this Inv-/SubBytes transformation is stored in row zero (see Fig. 6) two clock cycles later. Using the pipelined S-Boxes and the Barrel shifter between row zero and row one of the registers, the Inv-/SubBytes and the Inv-/ShiftRows transformations can be applied to all 16 bytes of the state within five clock cycles.

In the sixth clock cycle of a normal AES round, the Inv-/MixColumns and the AddRoundKey transformations are performed by all data cells in parallel. Since the S-Boxes are not used by the data unit during the sixth clock cycle, they can be utilized by the key unit to perform the key expansion for the next round key. In order to compute the final round of an encryption or decryption, the Inv-/Mixcolumns transformation is omitted by the data cells in this clock cycle.

In this way, the required number of encryption or decryption rounds can be executed by the data unit and the key unit until the 128-bit result is finally stored in the registers of the data unit. This result is then shifted column by column to the left (to the interface of the AES module). At the same time, a new input state can be loaded.

Using the standard data unit, the minimal number of clock cycles that are required to perform an AES-128 encryption or decryption is 64. Four clock cycles are required for the I/O of the data unit, 54 clock cycles are required to perform the nine normal AES rounds, and six are required for the final round.

The following two subsections present the architecture of the S-Boxes and the data cells.

### 3.1.1 S-Boxes

In hardware implementations, the SubBytes transformation and its inverse are the most expensive AES transformations. This is why the standard data unit does not contain as many S-Boxes as data cells.

In principle, there are two ways for implementing an S-Box in hardware that can be used for the SubBytes transformation and its inverse. It can either be implemented as ROM lookup or it can be calculated with combinational logic. The straightforward way to implement an S-Box is to store all possible output values for the S-Box function and its inverse in a ROM. However, this requires a small ROM with 512 bytes, where the overhead for address decoding and output signal conditioning outweighs the area requirements of the ROM matrix.

Alternatively, just the result of the inversion in $GF(2^8)$ could be stored in a 256 byte ROM and the affine transformation and its inverse could be calculated with combinational logic. This approach would only need half the ROM size of the first approach, but it would have an even worse overhead to matrix ratio.

The best way to implement an S-Box is to use combinational logic for the affine transformation, for its inverse and also for the computation of the inverse in $GF(2^8)$. This approach was first proposed by Rijmen in [16] and used by Rudra et al. in [11]. Implementations of S-Boxes that are particularly interesting for the presented architecture have been proposed by Satoh et al. in [12] and by Wolkerstorfer et al. in [17].

For the presented AES module, a pipelined (one stage) implementation of the S-Box as described in [17] is used. The main idea of this implementation is to build an efficient combinational circuit for the S-Box, which is based on the fact that $GF(2^8)$ can be seen as a quadratic extension of the field $GF(2^4)$. A pipelined version of the S-Box is used to accomplish that the combinational paths in the architecture are balanced (i.e., the paths of the S-Boxes and those of a MixColumns-and-AddRoundKey step are roughly the same).

### 3.1.2 Data Cells

The design of the data cells is crucial for the overall architecture of the data unit. The data cells serve as storage elements of the AES state and perform the Inv-/MixColumns and the AddRoundKey transformation. Each data cell consists of the following components:

- **Eight flip-flops:** Each data cell stores one byte of the current AES state (see Figs. 1 and 6).
- **One Multiplier:** The MixColumns transformation maps one column of the input state to a new column in the output state. The multiplier that is a part of each data cell computes one output byte of the MixColumns transformation based on a four byte input. This multiplier considers its four byte input as polynomial over $GF(2^8)$ and is capable of performing a multiplication of the input with the constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ and with its inverse, $a(x)^{-1}$, modulo $x^4 - 1$.

  The inputs of each multiplier are connected to the outputs of the four data cells that are in the same column as the multiplier itself (see Fig. 6). However, due to the definition of the MixColumns and the InvMixColumns transformation, the input connections are different in each row. The multipliers of the architecture are designed in a way that there is a maximum reuse of components between the multiplication with $a(x)$ and the one with $a(x)^{-1}$. A detailed description of this multiplier architecture can be found in [18].
- **Eight XOR-Gates:** The AddRoundKey transformation is performed in parallel in the presented architecture. Consequently, eight xor gates are required in each data cell.
- **Input Selection:** The data cells support unidirectional vertical and horizontal shifting. Consequently, each data cell consists of a multiplexor to select which input is loaded into the data cell.

In addition to these basic components, multiplexors are required in the data cells to switch the connections between the four major components. This is necessary because, for different types of AES rounds, the data cells need to perform different transformations.

The following paragraphs summarize how the components of the data cells need to be switched for AES encryption as well as for decryption rounds and to perform loading and reading operations in the presented architecture.

- **Encryption:**
  - *ShiftRows and SubBytes:* In a normal and in a final AES encryption round, first the ShiftRows and the SubBytes transformations need to be performed. As mentioned before, these transformations are done by a vertical rotation of the content of the data cells. The only capability a data cell needs to perform these transformations is vertical shifting.
  - *MixColumns followed by AddRoundKey:* In the last clock cycle of a normal encryption round, the MixColumns followed by the AddRoundKey transformation are applied. This requires that the output of the multiplier is fed into the xor gates.
  - *AddRoundKey:* In the final AES encryption round, no MixColumns transformation needs to be performed. Consequently, only the xor gates are used in the last clock cycle of a final encryption round.
- **Decryption:**
  - *InvShiftRows and InvSubBytes:* The S-Boxes and the Barrel shifter can be switched from encryption to decryption mode. Since the InvShiftRows and the InvSubBytes transformation can be performed in arbitrary order, these transformations can be performed by doing the same vertical rotation as for the encryption.
  - *AddRoundKey followed by InvMixColumns:* In a normal AES decryption round, the AddRoundKey transformation needs to be done before the InvMixColumns transformation. The output of the xor gates needs to be connected to the input of the multiplier. Compared to the encryption, the sequence of the multiplier and the xor gates simply needs to be switched. By implementing this sequence switching in the data cell, the hardware effort in the key unit is reduced (see Section 5.3.5 of [2]).
  - *AddRoundKey:* In the final decryption round, the InvMixColumns transformation is omitted. The same configuration of the data cell is used for the last clock cycle of a final decryption round as for the last clock cycle of a final encryption round.
- **Loading and Reading:**
  - *Loading and reading:* The content of the data cells is shifted horizontally to load and read data. During the first three clock cycles of loading and reading, the data cells only perform a horizontal shifting operation.
  - *AddRoundKey and load:* When the fourth 32-bit input word is loaded, additionally the initial AddRoundKey transformation is performed by using the xor gates of the data cells. This saves one clock cycle and therefore improves the overall performance.

  The reading of an output state and the loading of an input state can be done concurrently. While the new state is shifted in word-wise, the previous output state is shifted out.

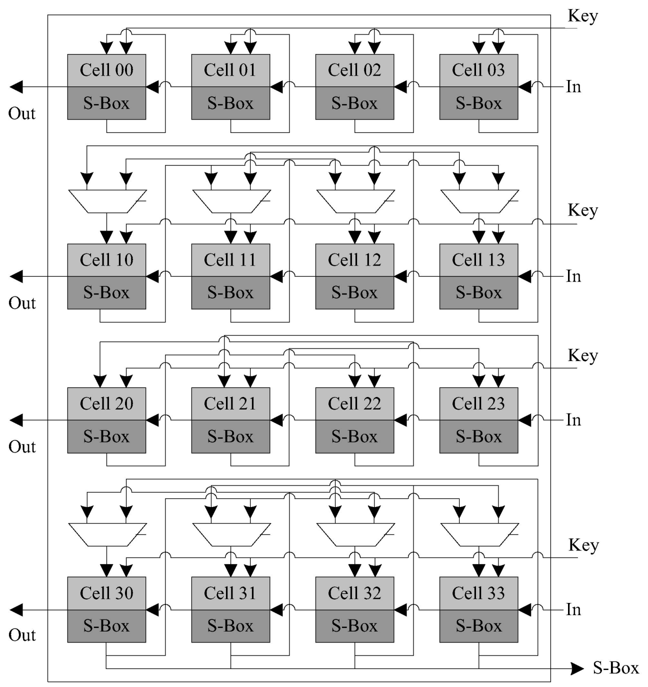


Fig. 7. High-performance data unit.

## 3.2 High-Performance Data Unit

In this section, the high-performance version of the data unit is presented. Like in the standard version, the high-performance data unit stores the current 128-bit state and can execute AES encryption and decryption rounds. It also has balanced paths and is highly regular, like the standard version.

As shown in Fig. 7, the high-performance version consists of 16 data cells and 16 S-Boxes. For the sake of clearness, the key wires are collapsed and the connections for the MixColumns transformation are not shown in this figure. However, the intercolumn connections for the MixColumns transformation and the key wires are the same as in the standard data unit.

In the high-performance version, there is an S-Box for each data cell. Consequently, no vertical rotation through all rows is necessary to perform the SubBytes transformation. The Barrel shifter performing the ShiftRows transformation cannot be reused and needs to be implemented separately for each row. Since each row only needs to be rotatable by a certain offset, simple multiplexors can be used instead of the Barrel shifters.

In fact, rows one and three only need to be rotatable by one position to the right and by one position to the left. Row two only needs to be rotatable by two positions to the right and row zero requires no circuit for rotation at all.

The important difference between the high-performance and the standard version is that normal and final AES rounds can be performed in three clock cycles (the standard data unit requires six clock cycles). The transformations that are done in these clock cycles are listed below:

1. In the first clock cycle, the first stage of the S-Box pipelines is loaded with the content of the data cells to perform the Inv-/SubBytes transformation.
2. In the second clock cycle, the second stage of the S-Box pipelines returns the results of the S-Box operations. These results are rotated according to the Inv-/ShiftRows transformation and stored in the data cells.
3. In the last clock cycle of a final round, only the AddRoundKey transformation is executed. In a normal encryption or decryption round, additionally, the Inv-/MixColumns transformation is performed. During encryption, the MixColumns transformation is done prior to the AddRoundKey transformation and, during decryption, the InvMixColumns transformation is executed afterward.

   In the third clock cycle, the S-Boxes are idle and can consequently be used by the key unit to calculate the key expansion for the next round key.

An AES-128 encryption or decryption of a 128-bit block of data can be done in 30 clock cycles. Including I/O, this sums up to 34 clock cycles compared to 64 clock cycles of the standard version. Section 4 analyzes the performance of both versions.

### 3.3 Key Unit

The key unit is used to store keys and to calculate the key expansion function. Due to the fact that the AES is standardized for 128, 192, and 256-bit keys, the interface between the key unit and the data unit is designed in a way that the key expansion for several different key sizes can be implemented on the same chip.

Independent of the key size, S-Box operations are required for the key expansion. Since the data unit does not perform any S-Box lookups while the MixColumns and AddRoundKey transformations are executed, the S-Boxes of the data unit are reused by the key unit during this clock cycle. Sharing the S-Boxes saves a significant amount of area.

AES-128 is currently the most widely used variant of the AES algorithm. Therefore, the description of the key unit is limited to this key size. Key units for other key sizes can be built in a similar way. For an AES-128 encryption, the key unit needs to generate $RoundKey_0$ to $RoundKey_{10}$. When doing a decryption, this needs to be done in reverse order.

Principally, there are two ways of implementing an AES-128 key expansion. Software implementations usually have sufficient RAM to precalculate and store all round keys. In hardware implementations, the key expansion is more efficient, when it is done on-the-fly. The key unit stores $RoundKey_i$ and is able to calculate $RoundKey_{i+1}$ or $RoundKey_{i-1}$, respectively.

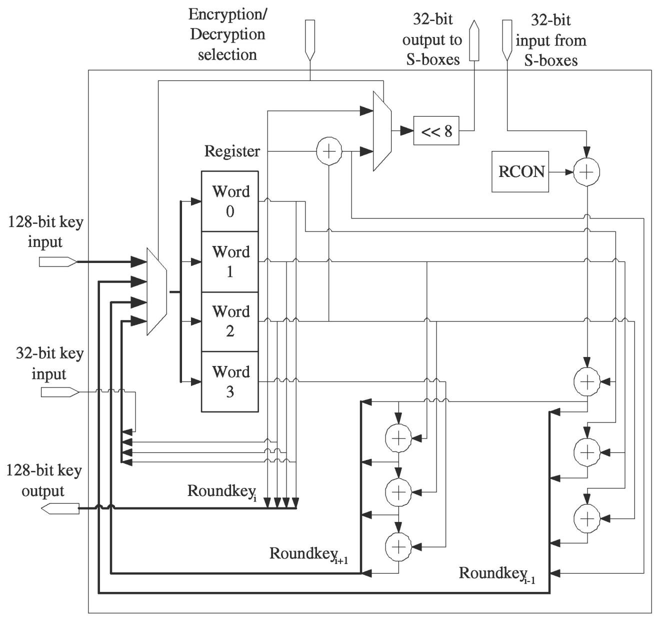


Fig. 8. AES-128 key generator.

The on-the-fly key expansion is done by the so-called key generator, which is the major part of the key unit. Besides the key generator, the key unit optionally consists of additional registers (the key storage) to provide a high level of key agility—this concept will be explained at the end of this section.

The key generator is able to compute the previous or the next round key within one or two clock cycles (depending on the number of pipeline stages in the S-Boxes). As shown in Fig. 8, this can be done quite efficiently—the critical paths are relatively short and the required area is small.

In case of an AES-128 encryption, the cipher key is loaded first and then, during the encryption process, $RoundKey_1$ to $RoundKey_{10}$ are calculated iteratively. For decryption, first a so-called key setup needs to be done. The reason for this is that a decryption uses $RoundKey_{10}$ in the initial round and the cipher key is required in the final round. During a key setup, the cipher key is expanded to $RoundKey_{10}$ and, so, $RoundKey_9$ to $RoundKey_0$ can be derived iteratively for decryption.

If two or more blocks of data need to be encrypted with the same key, there exist two possibilities to do this. In both cases, first a 128-bit cipher key needs to be loaded into the key generator. Then, for the encryption of the first block, the key expansion is performed until $RoundKey_{10}$ is stored in the register of the key generator. In order to encrypt the second block, this register needs to contain the corresponding cipher key again.

If the AES module is connected via a relatively slow bus, there is enough time to iteratively recalculate the original cipher key based on $RoundKey_{10}$, while the next data block is loaded. Using the pipelined S-Boxes of the data unit, this takes 20 clock cycles.

The alternative to this relatively slow, but area saving, procedure is the concept of key agility. For this purpose, the key unit not only consists of the key generator, but also of a number of 128-bit registers—the so-called key storage.

TABLE 1
Components and Their Complexity of the AES-128 Standard Module

| | Standard | |
|---|---|---|
| Module/Component | GE | % |
| S-Boxes | 4 x 392 | 14.5 |
| Multipliers | 16 x 212 | 31.4 |
| D. cells without mult. | 16 x 87 | 12.9 |
| Multiplexors | 384 | 3.6 |
| **Subtotal Data unit** | **6,736** | **62.4** |
| Key generator | 1,503 | 13.9 |
| Key storage | 691 | 6.4 |
| **Subtotal Key unit** | **2,194** | **20.3** |
| **CBC register** | **1,623** | **15.0** |
| **AMBA APB bus intf.** | **246** | **2.3** |
| **Total** | **10,799** | **100.0** |

TABLE 2
Components and Their Complexity of the AES-128 High-Performance Module

| | High-Performance | |
|---|---|---|
| Module/Component | GE | % |
| S-Boxes | 16 x 392 | 40.5 |
| Multipliers | 16 x 212 | 21.9 |
| D. cells without mult. | 16 x 87 | 9.0 |
| Multiplexors | 374 | 2.4 |
| **Subtotal Data unit** | **11,430** | **73.8** |
| Key generator | 1,503 | 9.7 |
| Key storage | 691 | 4.5 |
| **Subtotal Key unit** | **2,194** | **14.2** |
| **CBC register** | **1,623** | **10.5** |
| **AMBA APB bus intf.** | **246** | **1.6** |
| **Total** | **15,493** | **100.0** |

These registers are used to cache cipher keys. The content of each of them can be loaded directly into the register of the key generator. This has several advantages: First of all, several consecutive data blocks can be encrypted without any delay for key loading or a key recalculation in between. Second, in a scenario where multiple keys are used, the switching between the keys can be done without any delay. Finally, in a scenario where fast switching between encryption and decryption is required, the cipher key (needed as first key in an encryption) and $Roundkey_{10}$ (needed as first key in a decryption) can be cached so that there is no key setup latency.

The key unit that is used for both presented data units consists of a single key storage. This allows the encryption of multiple data blocks without any delay.

## 4 Performance

In this section, an AES-128 module built with a standard data unit and an AES-128 module based on a high-performance data unit are compared in terms of performance and area. Additionally, a comparison to related work is given.

### 4.1 Performance of the Presented Designs

Both versions of the AES-128 module have been implemented with VHDL and have been synthesized for a 0.6 $\mu$m CMOS process. The AES-128 module with the standard data unit has a complexity of 10,799 gate equivalents (GEs) and the module based on the high-performance data unit has 15,493 GEs. The complexity of each component of the standard version of the module is listed in Table 1. An analysis of the components of the high-performance module is listed in Table 2.

The multiplier, which is a part of the data cell, is listed separately because the multiplier accounts for most of the gates of the data cell. The high-performance AES-128 module essentially consists of 12 S-Boxes more than the standard module. These S-Boxes account for an increase of the total complexity by 43 percent. The critical path of both designs is more or less the same and it is determined by the delay of one pipeline stage of the S-Box—the maximum frequency for the AES-128 modules on the 0.6 $\mu$m technology is about 64 MHz. In Table 3, a summary of the performance is shown.

The standard version needs 64 clock cycles and the high-performance version 34 cycles to perform an AES-128 encryption or decryption. This improvement of the throughput by 88 percent is paid for by an increase of the complexity by 43 percent.

### 4.2 Related Work

This section compares the presented architecture with the one proposed in [12]. This was, so far, the most efficient published AES hardware implementation.

The design of Satoh et al. consists of 5,400 GEs and its maximum clock frequency is about 130 MHz on a 0.11 $\mu$m technology. The design requires 54 cycles to perform an encryption, which leads to a theoretical throughput (for the four-S-Box version) of 311 $Mbit/sec$. Unfortunately, the authors have not included the gates for loading and reading of data in the numbers presented in [12], and, so, their gate count seems very low at first glance.

The gate count is based on a core data path without mechanisms for I/O, CBC registers, or a key storage. So, for a comparison of complexity, the gate counts for these components need to be subtracted from the presented design—this leads to a gate count of about 8,200 GEs for the standard AES-128 module. However, an objective comparison is still not possible since the 0.11 $\mu$m technology used in [12] is more extensive than the technology we use. The big difference in the used technology also does not allow a reasonable comparison of the maximum frequencies or the throughput.

However, an important fact is that the critical path in [12] is very long: The SubBytes, the MixColumns, and the AddRoundKey transformation are done for one column

TABLE 3
Summary of the Performance of the AES-128 Module

| Version | Clock Cycles | Throughput [$Mbit/sec$] | Area [GE] |
|---|---|---|---|
| Standard | 64 | 128 | 10,799 |
| High perf. | 34 | 241 | 15,493 |

within one clock cycle. Additionally, in the same clock cycle, the data passes the so-called selector function, which seems to be another major cause of delay. This long critical path leads to the low clock frequency of 130 MHz on the 0.11 $\mu$m technology.

In the presented architecture, the critical path is only one pipeline stage of an S-Box, which is no more than a third of the critical path of the architecture presented in [12]. Therefore, if the same technology is used for both designs, we expect the maximum frequency of our module to be at least three times higher than the maximum frequency stated in [12]. This leads to a better overall performance.

## 5 Conclusion

In this paper, a highly regular and scalable AES hardware architecture is presented. Due to its modularity, the performance and the used key size can be changed easily. Although the design is very modular, a high level of resource sharing is done between encryption and decryption, as well as between the key expansion and the computation of the AES rounds.

The presented architecture differs significantly from other proposals because of its regularity and its relatively short and balanced combinational paths. These properties do not only lead to a high performance, but they particularly also support full-custom and low-power designs.

## Acknowledgments

This work originates from the European Commission funded project *Crypto Module with USB Interface (USB_ CRYPT)* established under contract number IST-2000-25169 in the Information Society Technologies (IST) program.

## References

[1] J. Daemen and V. Rijmen, *The Design of Rijndael.* Springer-Verlag, 2002.

[2] Nat'l Inst. of Standards and Technology, "Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[3] Nat'l Inst. of Standards and Technology, "Federal Information Processing Standard 46-3, The Data Encryption Standard (DES)," http://csrc.nist.gov/publications/fips/, 1999.

[4] A.J. Elbwirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," *Proc. Third Advanced Encryption Standard Candidate Conf.*, pp. 13-27, 2000.

[5] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," *Proc. Third Advanced Encryption Standard Candidate Conf.*, pp. 40-56, 2000.

[6] N. Weaver and J. Wawrzynek, "A Comparison of the AES Candidates Amenability to FPGA Implementation," *Proc. Third Advanced Encryption Standard Candidate Conf.*, pp. 28-39, 2000.

[7] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms," http://csrc.nist.gov/encryption/aes/round2/NSA-AESfinalreport. pdf, 2000.

[8] V. Fischer and M. Drutarovský, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," *Proc. Workshop Cryptographic Hardware and Embedded Systems—CHES 2001*, pp. 77-92, 2001.

[9] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82Gbits/Sec VLSI Implementation of the AES Rijndael Algorithm," *Proc. Workshop Cryptographic Hardware and Embedded Systems—CHES 2001*, pp. 51-64, 2001.

[10] M. McLoone and J.V. McCanny, "High Performance Single-Chip FPGA Rijndael Algorithm Implementations," *Proc. Workshop Cryptographic Hardware and Embedded Systems—CHES 2001*, pp. 65-76, 2001.

[11] A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar, J.R. Rao, and P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic," *Proc. Workshop Cryptographic Hardware and Embedded Systems—CHES 2001*, pp. 171-184, 2001.

[12] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. Advances in Cryptology—ASIACRYPT 2001*, pp. 239-254, 2001.

[13] P.C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Proc. Advances in Cryptology—CRYPTO 1999*, pp. 388-397, 1999.

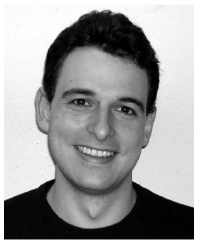
[14] K. Tiri, M. Akmal, and I. Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards," *Proc. 28th European Solid-State Circuits Conf.—ESSCIRC 2002*, 2002.

[15] Nat'l Inst. of Standards and Technology, "Special Publication 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation," http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf, 2001.

[16] V. Rijmen, "Efficient Implementation of the Rijndael SBox," http://www.esat.kuleuven.ac.be/rijmen/rijndael/sbox.pdf, 2000.

[17] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES S-Boxes," *Topics in Cryptology—CT-RSA 2002, Proc. RSA Conf. 2002*, Feb. 2002.

[18] J. Wolkerstorfer, "An ASIC Implementation of the AES-MixColumn Operation," *Proc. Austrochip 2001*, Oct. 2001.



**Stefan Mangard** received the Dipl.-Ing. degree in IT engineering from Graz University of Technology, Austria, in 2002. As a member of the VLSI & Security research group at the Institute for Applied Information Processing and Communication (IAIK), Graz, Austria, he is currently working toward the PhD degree. His research interests include side-channel attacks and development of corresponding countermeasures as well as design and implementation of cryptographic hardware in general. He is a student member of the IEEE and the IEEE Computer Society.

**Manfred Aigner** received the Dipl.-Ing. degree in IT engineering from Graz University of Technology, Austria, in 2000. He has been a part of the VLSI & Security research group at the Institute for Applied Information Processing and Communication (IAIK) since 1997. He works mainly in the design and development of cryptographic modules for smart cards from specification until prototype testing. His research interests also include side-channel attacks to cryptographic hardware and their countermeasures.

**Sandra Dominikus** obtained the Dipl.-Ing. degree in IT engineering from the Graz University of Technology, Austria, in 2002. In 2000, she joined the VLSI & Security research group at the Institute for Applied Information Processing and Communication (IAIK), Graz, Austria. She works in design and testing of hardware modules for cryptographic purposes. Her research interests include design methodology and design automation for hardware development.