Efficient AES Implementations on ASICs and FPGAs

Norbert Pramstaller, Stefan Mangard, Sandra Dominikus, and Johannes Wolkerstorfer

Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
{firstname.surname}@iaik.at

Abstract. In this article, we present two AES hardware architectures: one for ASICs and one for FPGAs. Both architectures utilize the similarities of encryption and decryption to provide a high throughput using only a relatively small area. The presented architectures can be used in a wide range of applications. The architecture for ASIC implementations is suited for full-custom as well as for semi-custom design flows. The architecture for the FPGA implementation does not require on-chip block RAMs and can therefore even be used for low-cost FPGAs.

Keywords: Advanced Encryption Standard (AES), FPGA, ASIC.

1 Introduction

The symmetric block cipher Rijndael [4] has been standardized by NIST[1] as Advanced Encryption Standard (AES) [10] in November 2001. Today, AES is the most widely used symmetric block cipher and it is implemented in many different devices to secure wired as well as wireless connection links.

The requirements for an implementation of AES strongly depend on the application running on the device and of course also on the type of the device. In many scenarios, it is sufficient to implement AES in software. However, there are also many very relevant scenarios, where the requirements concerning the implementation of AES can only be met by dedicated hardware implementations.

Encryption engines for high speed communication links, for example, often need to be implemented in hardware due to the high throughput requirements. In applications that need to be resistant against side-channel attacks, AES is also often implemented in special hardware modules [11]. The reason for this is simply that it is easier to secure a small AES module against side-channel attacks rather than to secure an entire processor. In devices where the power consumption is critical, hardware implementations are also the preferred choice, because they consume considerably less power than software implementations.

[1] National Institute of Standards and Technology.

H. Dobbertin, V. Rijmen, A. Sowa (Eds.): AES 2004, LNCS 3373, pp. 98–112, 2005.

In this article, we present efficient hardware implementations for ASICs and FPGAs that can be used for a wide range of applications. Both architectures use similarities of encryption and decryption to provide a high level of performance using only a relatively small area. While most publications on implementations of AES only provide performance and area figures without interfaces and registers for CBC mode, the architectures presented in this article are complete. Both architectures include an AMBA APB [1] interface and can perform encryptions and decryptions in CBC mode.

Our architecture for ASICs is presented in Section 2 and the one for FPGAs is discussed in Section 3. Conclusions about both architectures can be found in Section 4.

2 ASIC Implementation of AES

During the last years, several proposals [8, 12, 13, 16, 19] on how to implement AES on an ASIC have been published. Most of these publications focus mainly on the throughput of the implementation. In this article however, we describe an architecture that in addition is highly regular and scalable. Therefore, it can be used for a wide range of applications.

The architecture discussed in this article has balanced combinational paths in order to fully utilize every clock cycle. 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.

The regularity of the presented architecture 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 interesting for smart card implementations that are required to provide protection against power analysis attacks [7]. In a full-custom approach, the designer can well balance the capacitive loads of output nodes, as it is for example desired for logic styles like the one described in [14, 15].

The performance of our AES architecture can gradually be increased at the cost of an increased chip size. The overall structure of this architecture, which is capable of performing AES encryptions and decryptions, is shown in Fig. 1. The AES hardware module consists of the following four components:

- **The Interface:** The AMBA APB interface handles all communication of the AES module with its environment.
- **The 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 which need to be executed are the same for all key sizes. Consequently, the data unit is independent of the key size.

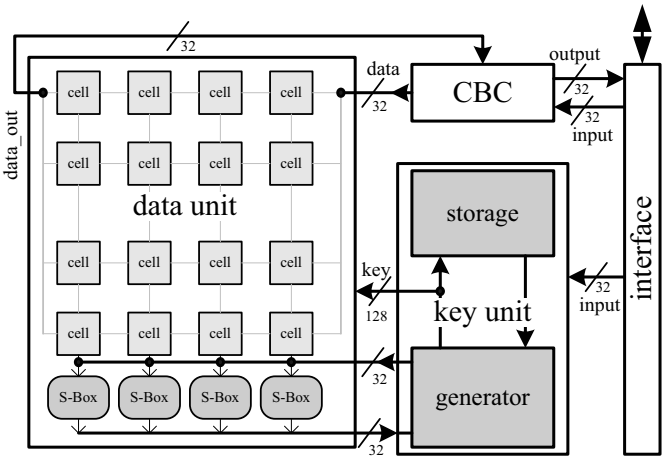

Fig. 1. Overall structure of the AES module

- **The 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.
- **The CBC Unit:** 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 AMBA APB interface into the key unit. Once a key is loaded, it can be used for an arbitrary number of encryptions and decryptions. After the loading of 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 required 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 data and the key unit.

2.1 The Data Unit

The data unit is the biggest and the most important component of the AES architecture. It stores the current 128-bit data block of an encryption or de-

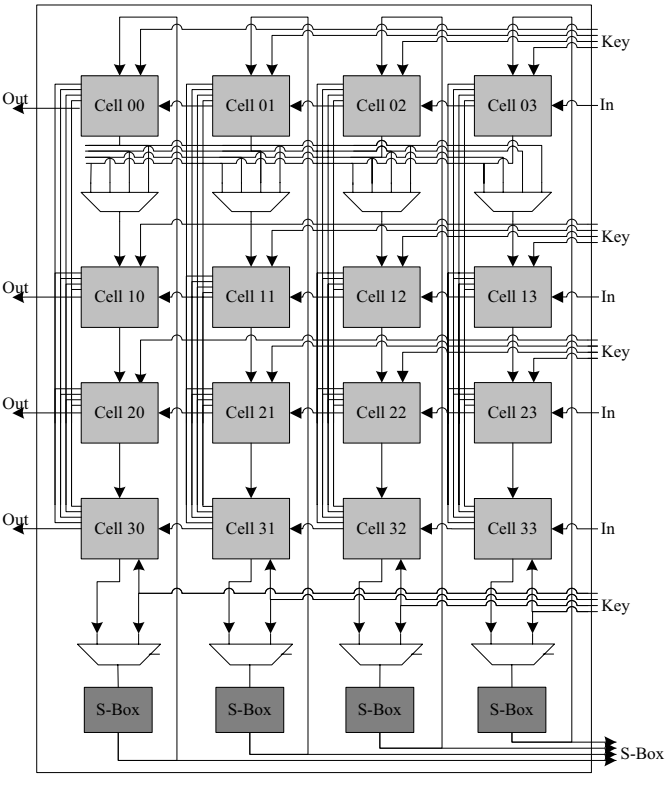

Fig. 2. The architecture of the standard data unit

cryption (referred to as the "State") and is capable of performing any number and type of en-/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 corresponding round key needs to be provided by the key unit.

Figure 2 shows the standard version of the data unit. It consists of sixteen so-called data cells and four S-Boxes. An S-Box of the architecture is a circuit capable of performing the SubBytes and the inverse SubBytes transformation for an 8-bit input. The data 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 such that connection by abutment is possible when they are placed next to another.

Another distinguishing feature of the presented architecture is the fact that the combinational paths are relatively short and, more important, very balanced. The commonly used approach to implement 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. Therefore, 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 applied 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:

To load a data block, the input data is shifted column by column from the right side (see Fig. 2) into the data cells. The inputs labelled "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 our design, we use an S-Box with one pipeline stage. Therefore, the (Inv)SubBytes and the (Inv)ShiftRows transformations can be applied to all sixteen bytes of the State in 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.

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 65. 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 seven are required for the final round.

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

S-Boxes. In hardware implementations, the SubBytes transformation and its inverse are the most expensive AES transformations. For the presented AES module, a pipelined (one stage) implementation of the S-Box as described in [18] 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 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).

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. Besides some input selection circuit, each data cell consists of eight flip-flops, a multiplier [17] for the MixColums transformation (which can also be omitted in order to scale the design) and XOR gates for the AddRoundKey transformation.

2.2 The 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 such that the key expansion for several different key sizes can be implemented on the same chip.

The key unit used in our design stores the key loaded via the interface and is capable of calculating all round keys for encryption and decryption iteratively. 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. Details about the key unit can be found in [8].

2.3 Performance of the ASIC AES Implementation

As mentioned before, the presented AES module is built up very regular and is highly scalable. Three different scaled versions of the module have been implemented and tested. They are named "standard version", which was described in the previous sections, "minimum version", which is the smallest, but slowest one, and the "high-performance version", which is the fastest, but most area intensive version.

As shown in Fig. 2, the data unit of the standard version consists of 16 data cells (including 16 MixColumns multipliers) and four S-Boxes. The en-/decryption of an 128-bit data block requires 65 clock cycles. By using 16 S-Boxes instead of four, the performance can be increased. The S-Box lookup can be done for all 128 bits of the State in parallel. With this configuration (high-performance version), the AES module requires only 35 clock cycles to en-/decrypt a 128-bit data block. In the minimum version of the AES module, only four S-Boxes and four MixColumns multipliers are used. Only the four "leftmost" data cells contain MixColumns multipliers. In the other data cells the multipliers are omitted. Here, 92 clock cycles are needed to process a 128-bit data block.

In this section, we give a comparison of the three different scaled versions of the AES module in terms of performance and area. Additionally, a comparison to related work is given.

2.4 Performance of the Presented ASIC Design

The three versions of AES-128 have been implemented in VHDL and have been synthesized for a 0.6 μm CMOS process. Table 1 shows the complexity in gate

Table 1. Complexity of AES components in GE

Component	Complexity [GE]
S-Box	392
Multiplier	212
Data cell (without Multiplier)	87
Key generator	1,633
Key store	691
AMBA Bus Interface	267
CBC Register	1,599

Table 2. Complexity of the AES-128 modules

Component	#	Minimum	#	Standard	#	High Perf.
S-Boxes	4	1,568	4	1,568	16	6,272
Multipliers	4	848	16	3,392	16	3,392
D. cells without mult.	16	1,392	16	1,392	16	1,392
Multiplexors	96	224	192	384	224	374
Data unit		**4,032**		**6,736**		**11,430**
Key generator	1	1,633	1	1,633	1	1,633
Key store	1	691	1	691	1	691
Key unit		**2,324**		**2,324**		**2,324**
AMBA + CBC	1	1,866	1	1,866	1	1,866
Control logic		319		279		230
Additional		**2,185**		**2,145**		**2,096**
Total		**8,541**		**11,205**		**15,850**

equivalents (GE) of each component used for the AES module. In Table 2, the complexity of the three different modules is calculated by adding the size of the components used for the different versions. In the first column for each version the used number of components is given. The standard version of the module has a complexity of 11,205 GE, whereas the minimum version needs 8,541 GE. The high performance version requires 15,850 GE.

The high-performance module essentially consists of 12 S-Boxes more than the standard module. This causes an increase of the complexity by 41%. On the other hand, the minimum version consists of 12 MixColumns multipliers less than the standard version and is therefore 24% smaller. The critical path of all three designs is more or less the same and is determined by the delay of one pipeline stage of the S-Box—the maximum frequency for the complete AES-128 modules (including AMBA interface, CBC, and control logic) on the 0.6 μm technology is about 50 MHz. In Table 3, a summary of the performance is shown.

The standard version needs 65, the high-performance version 35, and the minimum version 92 clock cycles to perform an AES-128 encryption or decryption. In the high-performance version, the improvement of the throughput by 87% is paid by an increase of the complexity by 41%. In the minimum version, the reduced complexity (-24%) accounts for a 29% loss in throughput.

Table 3. Summary of the performance of the different AES-128 modules

Version	Clock Cycles	Throughput@50 MHz [*Mbps*]	Area [GE]
Minimum	92	70	8,541
Standard	65	98	11,205
High perf.	35	183	15,850

2.5 Related Work

This subsection compares the presented architecture with the one proposed in [13]. The design of Satoh et al. consists of 5,400 GE and was implemented on a 0.11 μm technology. The design consists of a core data path and a key generator. It does not include mechanisms for I/O, CBC registers or a key store. Its maximum clock frequency is about 130 MHz. The design requires 54 cycles to perform an encryption, which leads to a theoretical throughput (for the four-S-Box version) of 311 Mbps.

For a comparison of complexity, the gate count for our design has to be reduced by the additional components our design offers (key store, CBC registers, AMBA interface)—this leads to a gate count of about 8,600 GE for the standard AES-128 module. This comparison is still not completely fair, since different technologies are used for synthesis. The 0.11 μm technology used in [13] allows smaller structures and offers different leaf cells. It seems to be more extensive than our technology, because similar components of the designs are claimed to be smaller in the architecture in [13]. For example, the S-Box proposed by Satoh et al. was reconstructed and synthesized with our technology. The result was a 15% bigger S-Box than used in our design, whereas the results with the 0.11 μm technology in [13] show an S-Box that needs 25% less GE than our S-Box design in the 0.6 μm technology.

The big difference in the used technology also does not allow a reasonable comparison of the maximum frequencies or the throughput. When comparing the two proposed S-Boxes synthesized in our 0.6 μm technology in terms of delay, our proposed S-Box is about 30% faster. An attempt to compare the throughput of both designs starts with comparing the critical paths. In [13] the critical path is very long: The SubBytes, the MixColumns and the AddRoundKey transformation are done for one column 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.

In our presented architecture, the critical path consists only of one pipeline stage of an S-Box. This is approximately a third of the critical path of the architecture presented in [13]. Using the same technology for synthesis of the compared designs, we expect the maximum frequency of our module to be at least three times higher than the maximum frequency stated in [13]. This leads to a better overall performance.

3 FPGA Implementation of AES

Reconfigurable devices like Field Programmable Logic Arrays (FPGA) gain more and more importance in hardware, software and hardware/software co-designs. In the beginning often seen only as devices for rapid prototyping, FPGAs are increasingly used for final applications. One of the mostly used arguments for using FPGAs rather than ASICs, is the reduced time-to-market and the cost advantages of standard devices.

Due to the importance of reconfigurable devices, numerous FPGA AES implementations have been published within the last years. These implementations mainly focus on high throughput rates [3, 9], and use techniques like loop unrolling and pipelining. They are able to report throughput rates up to 12,160 Mbps [3]. Applying such techniques leads to AES hardware implementations that require a huge amount of FPGA resources that are only available for expensive devices and cannot be implemented in low-end FPGAs.

In this section we present a new universal architecture that is supported by several FPGA product families and can also be implemented using inexpensive low-end FPGAs. It is the first known AES FPGA implementation that does not require on-chip block RAMs. Furthermore, it implements the complete AES encryption standard and features the Cipher Block Chaining (CBC) mode.

3.1 Related Work

Gaj et al. [3] published the fastest known FPGA implementation. For encryption and decryption with 128-bit keys, a throughput of 12,160 Mbps on a Xilinx Virtex XCV1000BG560-6 device is reported. McLoone et al. [9] achieve a throughput of 6,956 Mbps for 128-bit keys only. They also presented encryption engines for 192-bit or 256-bit keys with accordingly lower throughput. Their combined encryption and decryption implementation can handle 128-bit keys and achieves a throughput of 3,239 Mbps on a Xilinx Virtex-E XCV3200E-8CG1156 device. The third implementation published by Dandalis et al. [5] also provides encryption and decryption for 128-bit keys. They achieve a throughput of 353 Mbps on a Xilinx Virtex XCV1000BG560-6 device. Fischer et al. [6] published a non-pipelined design supporting encryption and decryption for 128-bit keys. They report a throughput of 451 Mbps of their fast configuration and 115 Mbps for an economic configuration. A drawback of their design is the missing on-chip round-key generation. Chodowiec et al. [2] presented an implementation for low-end devices. Using only few resources they achieve a throughput ranging from 139 Mbps to 166 Mbps depending on the used FPGA device.

All implementations (except [6, 2]) use a considerable amount of hardware resources. For instance, [9] requires 138 block RAMs for 256-bit keys. This demands the use of expensive million-gate FPGA devices.

As shown above, most published hardware implementations focus on high throughput rates and do not provide a non-parameterizable design to support the complete AES standard. Furthermore, the high throughput implementations [3, 9] do not support the Cipher Block Chaining mode (CBC).

3.2 Architecture of the AES FPGA Implementation

This section describes the architecture of the AES co-processor. Starting with a swift overview, we will present details to highlight some innovative improvements that make it possible to present a resource-efficient AES co-processor suitable for low-end FPGA devices.

Basic components of the AES co-processor as shown in Fig. 3 are the AMBA APB interface, the data unit, the key unit, and the control unit. The key unit calculates the KeyExpansion function. All round keys are pre-calculated and stored in the key unit. Pre-calculated round keys allow fast en-/decryption of different data blocks for the same cipher key because no additional KeyExpansion is required. The data unit holds the State and performs all AES transformations: AddRoundKey, (Inv)SubBytes, (Inv)ShiftRows and (Inv)MixColumns. When encryption or decryption has completed the ciphertext (plaintext in case of decryption, respect.) is stored in the data unit. The control unit receives commands from the AMBA interface and generates control signals for all other modules. In addition to control round-key calculation, encryption and decryption, it also sequences data loading and unloading.

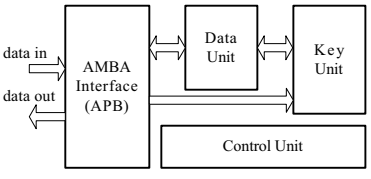

Fig. 3. Architecture of the AES co-processor

The architecture is similar to the architecture presented in Section 2. Differences are a modified State representation and a modified round-key calculation scheme. Due to a non-pipelined approach, the same performance for all modes of operations (ECB and CBC) is reached. Next we describe the AES data unit and the AES State representation in detail.

Data Unit. The data unit stores the State, all intermediate results of the round function applied to the State and the output data when encryption or decryption has completed. The major difference to all other published AES implementations is the innovative State representation that consists of two States. One State contains the actual State values and the other State stores newly calculated values. Figure 4 depicts the two States, referred to as StateA and StateB. In each cycle, 32 bits (one row or one column) of either StateA or StateB are altered. Using a second State provides a lot of benefits without the need of additional recourses: (Inv)ShiftRows comes for free and no State transposition between column and row operations is required.

Storage elements in FPGAs can be efficiently implemented by using synchronous RAMs because the basic logic elements of FPGAs, called slices, can

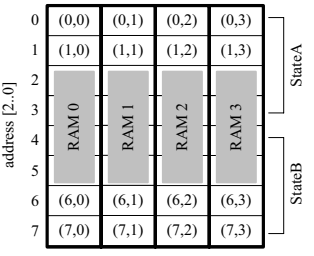

Fig. 4. The State-RAM

be configured as 16×1 bit synchronous RAM. Two slices (= one Configurable Logic Block - CLB) provide 16×1 bit synchronous dual-port RAM functionality (see [20]). Dual-port RAMs allow concurrent reading and writing to the RAM. Due to these technology features, the State-RAM as depicted in Fig. 4 is implemented as four slices of 8×8 bit synchronous dual-port RAMs to allow addressing the slices independently.

The data unit performs all transformations of the round function: (Inv)ShiftRows, (Inv)SubBytes, (Inv)MixColumns and AddRoundKey. AddRoundKey and (Inv)MixColumns are applied to the State column by column, whereas (Inv)-ShiftRows and (Inv)SubBytes are applied to the State row by row. Due to the slice architecture of the RAM which holds the State, it is not possible to read/write from/to the RAM column by column. Hence, a transposition of the State is necessary if a row-oriented operation follows a column-oriented operation, or vice versa. Transposition would require a reorganization of the State before further operations can be performed. By using two States, transposition can be implemented by accordingly addressing the State-RAM. Furthermore, (Inv)ShiftRows can be combined with transposing the State. As a consequence of this, (Inv)ShiftRows and transposition come for free. In the sequel we describe the memory organization and State transposition for encryption. The same approach can easily be modified for decryption.

When a row-oriented operation follows a column-oriented operation (or vice versa), the State must be transposed. Combining row and column transformations minimizes the number of required transpositions: ShiftRows is combined with SubBytes and AddRoundKey is combined with MixColumns. This approach requires only one transposition per round. Encryption requires SubBytes followed by ShiftRows. Since ShiftRows does not affect the byte values and SubBytes is applied to each byte of the State individually, the order of both operations does not matter. This fact eases the address generation for the State-RAM.

For explaining the State transposition we consider the State as 4×4 matrix: $\mathbf{S} = (s_{i,j})_{j=0..3}^{i=0..3}$. The ShiftRows transformation described in [10] can then be expressed as follows:

$$\mathbf{S}' = ShiftRows(\mathbf{S}) = (s_{i,j-i \bmod 4})_{j=0..3}^{i=0..3} \tag{1}$$

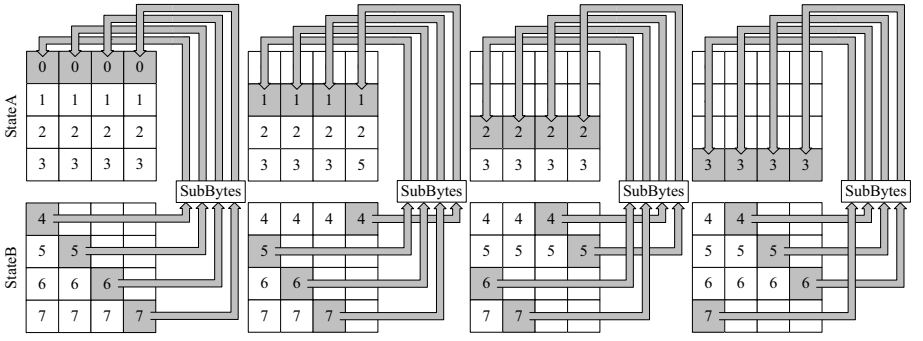

Fig. 5. ShiftRows and SubBytes for encryption

If we replace the State by the transposed State, we obtain:

$$\mathbf{S'^T} = ShiftRows(\mathbf{S^T}) = (s_{i+j\,mod\,4,j})_{j=0..3}^{i=0..3} \tag{2}$$

With the result of (2) the addressing of the StateB-RAM can be determined: the indices (i, j) must be substituted with $(i + j\,mod\,4, j)$. Due to the even number of AES rounds for all key lengths, ShiftRows is always applied to StateB only. Thus, the resulting index tuples can be directly mapped to the RAMs. The first part of the tuple index specifies the RAM slice and the second part specifies the RAM address. Since we operate on StateB, we must add an offset of 4 to the index value to get the correct address. Figure 5 shows the transposition of the State, including ShiftRows and SubBytes for encryption.

Implementation of (Inv)SubBytes and (Inv)MixColumns. The (Inv)-SubBytes transformation is based on [18]. One difference is that the byte inversion in GF(2^8) is implemented by using a synchronous ROM. (Inv)MixColumns is similar to the architecture presented in [17]. For further details refer to [18, 17].

Key Unit. An innovative aspect of our implementation is that the key unit can handle 128-bit, 192-bit and 256-bit keys with minimal additional hardware requirements. Supporting all key lengths increases the needed hardware resources for the key unit by only 7.8%. The size of the key memory for 256-bit keys is the same as for 128-bit keys. For 128-bit keys, the KeyExpansion function derives 44 32-bit round-key parts from the cipher-key. This requires a 64×32 bit RAM. 256-bit keys produce 63 32-bit round-key parts fitting the 64×32 bit RAM.

3.3 Performance of the FPGA AES Implementation

This section compares the proposed AES co-processor with the works referred to in Section 3.1. In order to provide comparable results, we implemented our co-processor on a Xilinx Virtex-E XCV1000EBG560-8 device.

Table 4. Hardware resources and throughput comparison

Work	**Device**	**#CLB-slices**	**#BRAM**	ECB mode **Throughput** [Mbps]
Gaj et al. [3]	Xilinx XCV1000	12,600	80	12,160
McLoone et al. [9] (I)	Xilinx XCV812E	2,222	100	6,956
McLoone et al. [9] (II)	Xilinx XCV3200E	2,577	112	5,800
McLoone et al. [9] (III)	Xilinx XCV3200E	2,995	138	5,000
McLoone et al. [9] (IV)	Xilinx XCV3200E	7,576	102	3,239
Dandalis et al. [5]	Xilinx XCV1000	5,673	?	353
Fischer et al. [6] (I)	FLEX 10KE200-1	2,530	24	451
Fischer et al. [6] (II)	ACEX 1K50-1	1,213	10	115
Chodowiec et al. [2]	Xilinx XC2S30-6	222	3	166
Our proposal	Xilinx XCV1000E	**1,125**	**0**	**215**

[9]: enc.: (I)AES-128, (II)AES-192, (III)AES-256, enc./dec.:(IV)AES-128
[6]: AES-128 enc./dec.: (I) fast configuration, (II) economic configuration

The performance results given in Table 4 are for the ECB mode. Most of these implementations claiming high throughput rates will have similar performance figures when operating in the CBC mode. The CBC mode is strictly recommended and commonly used for encrypting high-speed data streams (*e.g.* as it is used for encrypting data transfers over networks) and hence, the above-listed high throughput rates lose their significance.

As shown in Table 4 our implementation is the only one that does not require any block RAMs. The presented AES co-processor supports the complete AES standard and the CBC mode. Additionally, it is equipped with a 32-bit AMBA APB interface that eases the integration with processors used in System-on-Chip designs [1]. If we do not consider the CBC mode and the AMBA bus interface, our approach is still comparable with the above-listed works but we would require less hardware resources (-26 %).

Our implementation utilizes 9.16% of the available logic cells on a Xilinx Virtex-E XCV1000EBG560-8 device. 90.8% of the logic resources and 100% of the on-chip BRAMs can be used by other circuits like a LEON2 or an ARM processor. For a stand-alone application a low-end FPGA (*e.g.* Xilinx SpartanII XC2S100-6) is sufficient for implementing the complete AES co-processor— the other approaches (except [2]) do not fit on a SpartanII device. The high throughput designs do not support this flexibility and require expensive multi-million gate FPGAs. Another important fact is that the other works do not provide an en-/decryption engine that supports all defined key lengths.

The maximum clock frequency on a XCV1000 FPGA is 161 MHz. At this frequency, a throughput of 215 Mbps for AES-128, 180 Mbps for AES-192, and 156 Mbps for AES-256 is achieved for both ECB mode and CBC mode.

4 Conclusions

In this article we presented two designs of a compact AES co-processor, one suitable for ASIC implementations, the other one suitable for FPGAs. Both designs are able to implement the whole functionality of the AES standard: encryption and decryption with all key lengths (128-bit, 192-bit, and 256-bit). In addition to covering the complete AES standard they support the Cipher Block Chaining mode CBC. The AES co-processors also have a standard 32-bit interface (AMBA) that facilitates the integration in System-on-Chip designs.

Our ASIC implementation is very regular, which makes it well suited for full custom designs, and highly scalable. By scaling, the ASIC AES module it can be adapted for many different applications with different requirements. With this architecture high performance (up to 198 Mbps) as well as low area requirements (down to 8,500 GE) can be reached on a 0.6 μm technology.

For the FPGA implementation, we have shown that due to an innovative State representation the complete AES co-processor can be implemented on inexpensive low-end FPGA devices. An implementation on a Xilinx Virtex-E device uses only 1,125 CLB-slices and no block RAMs. Our FPGA implementation reaches a throughput of 215 Mbps at a clock frequency of 161 MHz for encryption and decryption.

References

1. ARM Limited. AMBA 2.0 Specification. `"http://www.arm.com/armtech/"`, 2001.
2. P. Chodowiec and K. Gaj. Very Compact FPGA Implementation of the AES Algorithm. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science (LNCS)*, pages 319–333. Springer, 2003.
3. P. Chodowiec, P. Khuon, and K. Gaj. Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. In *Symposium on Field Programmable Gate Arrays – FPGA 2001*, pages 94–102. ACM Press, 2001.
4. J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.
5. A. Dandalis, V. Prasanna, and J. Rolim. A Comparative Study of Performance of AES Final Candidates Using FGPAs. `"http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3agenda.html"`, 2000.
6. V. Fischer and M. Drutarovský. Two Methods of Rijndael Implementation in Reconfigurable Hardware. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science (LNCS)*, pages 77–92. Springer-Verlag, 2001.
7. P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
8. S. Mangard, M. Aigner, and S. Dominikus. A Highly Regular and Scalable AES Hardware Architecture. In *IEEE Transactions on Computers*, volume 52, pages 483–491, April 2003.

9. M. McLoone and J.V. McCanny. High Performance Single-Chip FPGA Rijndael Algorithm Implementations. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science (LNCS)*, pages 65–76. Springer-Verlag, 2001.
10. National Institute 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.
11. N. Pramstaller, F.K. Gürkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner. Towards an AES Crypto-chip Resistant to Differential Power Analysis. In *Proccedings of ESSCIRC 2004, to appear*, 2004.
12. A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar amd J.R. Rao, and Pankaj Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science (LNCS)*, pages 171–184. Springer-Verlag, 2001.
13. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science (LNCS)*, pages 239–254. Springer-Verlag, 2001.
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. In *Proceedings of ESSCIRC 2002*, 2002.
15. K. Tiri and I. Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science (LNCS)*, pages 125–136. Springer, 2003.
16. 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.
17. J. Wolkerstorfer. An ASIC implementation of the AES-MixColumn operation. In *Proceedings of Austrochip 2001*, October 2001.
18. J. Wolkerstorfer, E. Oswald, and M. Lamberger. An ASIC implementation of the AES S-Boxes. In *Topics in Cryptology – CT-RSA 2002, Proceedings of the RSA Conference 2002*, volume 1965 of *Lecture Notes in Computer Science*. Springer-Verlag, February 2002.
19. S-Y Wu, S-C Lu, and C-S Laih. Design of AES Based on Dual Cipher and Composite Field. In *Topics in Cryptology – CT-RSA 2004, Proceedings of the RSA Conference 2004*, volume 2964 of *Lecture Notes in Computer Science*. Springer-Verlag, February 2004.
20. Xilinx Incorporated. Silicon Solutions — Virtex Series FPGAs. "http://www.xilinx.com/products/".