# Implementation of the AES-128 on Virtex-5 FPGAs

Philippe Bulens[1]*, François-Xavier Standaert[1]**, Jean-Jacques Quisquater[1], Pascal Pellegrin[2], Gaël Rouvroy[2]

[1] UCL Crypto Group, Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium
[2] IntoPIX s.a., Place de l'Université, 16, B-1348 Louvain-La-Neuve, Belgium

**Abstract.** This paper presents an updated implementation of the Advanced Encryption Standard (AES) on the recent Xilinx Virtex-5 FPGAs. We show how a modified slice structure in these reconfigurable hardware devices results in significant improvement of the design efficiency. In particular, a single substitution box of the AES can fit in 8 FPGA slices. We combine these technological changes with a sound intertwining of the round and key round functionalities in order to produce encryption and decryption architectures that perfectly fit with the Digital Cinema Initiative specifications. More generally, our implementations are convenient for any application requiring Gbps-range throughput.

## 1 Introduction

Reprogrammable hardware devices are highly attractive options for the implementation of encryption algorithms. During the selection process of the AES [1], an important criterion was the efficiency of the cipher in different platforms, including FPGAs. Since 2001, various implementations have consequently been proposed, exploring the different possible design tradeoffs ranging from the highest throughput to the smallest area [2]. Each of those implementations usually focuses on a particular understanding of "efficiency". Furthermore, every time a new hardware platform is introduced, a new implementation is to be made in order to comply with and take advantage of its specificities.

Therefore, this paper aims to provide an update on the performances of the AES, taking the new Xilinx's Virtex-5 FPGAs as evaluation devices. Our results show how the modified slice structure (*i.e.* the 6-input Look-Up-Tables combined with multiplexors) allows an efficient implementation of the AES substitution box (S-box). We include these technological advances in a state-of-the art architecture for an encryption module. The resulting IP core typically complies with the Digital Cinema Initiative specifications [3]: the presented encryption and decryption designs can be used to decrypt the incoming compressed data stream in a digital cinema server and re-encrypt the uncompressed data between the server and the projector. More generally, it is convenient for any application requiring

* Supported by Walloon Region, Belgium / First Europe Program.
** Postdoctoral Researcher of the Belgian Fund for Scientific Research.

Gbps-range throughput and compares positively with most recently published FPGA implementations of the AES. Although the improvements described in this paper straightforwardly derive from technological advances, we believe they are of interest for the cryptographic community in order to keep state-of-the-art implementation results available and detailed in the public literature.

The rest of the paper is structured as follows. Section 2 briefly reminds how the AES cipher processes the data. Then, we discuss the specificities of our target platform before fully describing the architecture developed in Section 4. The implementation results are summarized in Section 5, together with some selected results from the literature. Finally, our conclusions are in Section 6.

## 2 Cipher Description

The AES is a substitution permutation network (SPN) allowing the encryption/decryption of data by blocks of 128-bits and supporting key lengths of 128, 192 and 256 bits. In the following, we focus on the 128-bit key version. Its internal state, usually represented as a $4 \times 4$ matrix of bytes, is updated by iterating through the round structure (10, 12 or 14 times according to the key size). The round is described as four different byte-oriented transformations.

First, **SubBytes** introduces the non-linearity by taking, for each byte, the modular inverse in GF($2^8$) and then applying an affine transformation. Instead of computing distinctly these two steps, the full transformation is achieved by passing each byte through an S-box (Figure 1). Then **ShiftRows** modifies the state. It simply consists of a circular left shift of the state's rows by 0, 1, 2 and 3 bytes respectively (Figure 2). Third, **MixColumns** applies a linear transformation to the state's columns (Figure 3). Each of them is regarded as a polynomial and is multiplied by a fixed polynomial $c(x) = 3 \cdot x^3 + x^2 + x + 2 \pmod{x^4 + 1}$. Finally, the **AddRoundKey** transform mixes the key with the state. As each subkey has the same size as the state, the combination is performed by a simple bitwise XOR between subkey bytes and their corresponding state bytes (Figure 4). A first key addition is performed before entering the first round, and the last round omits the **MixColumns** transformation.

Prior to the en/de-cryption process, the subkeys have to be generated.

The key schedule takes the main key $K_0$ and expand it as shown in Fig. 5 for the case of a 128-bit key, where **SubWord** applies the S-box to the 32-bit input word, **RotWord** rotates the word one byte to the left and *RC(i)* is an 8-bit constant associated to each round $i$ [1].

[1] In encryption mode, this can easily be performed "on-the-fly", *i.e.* in parallel to the rounds execution in order to get the subkey at the exact time it is needed. In decryption mode, the round keys generally have to be derived prior to the decipher. Solutions allowing "on-the-fly" derivation of the decryption subkeys require specific features (*i.e.* knowledge of a decryption key that corresponds to the last encryption subkeys) and hardware overhead. Therefore, these are not considered in this paper.

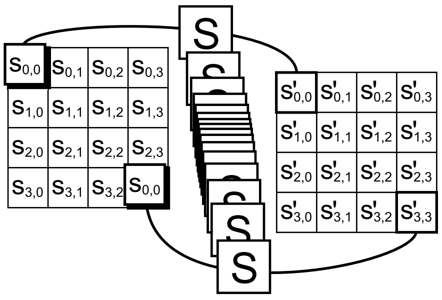
**Fig. 1. SubBytes** Transformation.

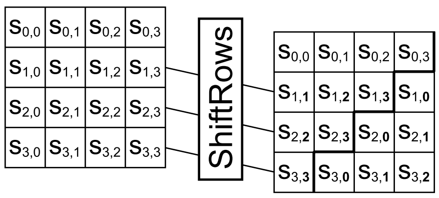
**Fig. 2. ShiftRows** Transformation.

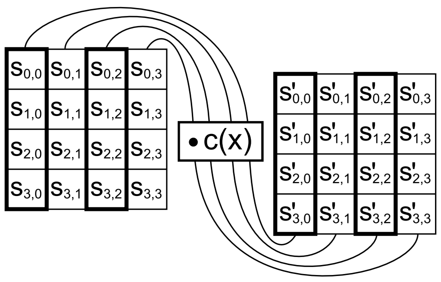
**Fig. 3. MixColumns** Transformation.

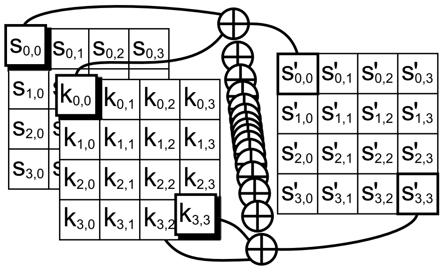
**Fig. 4. AddRoundKey** Transform

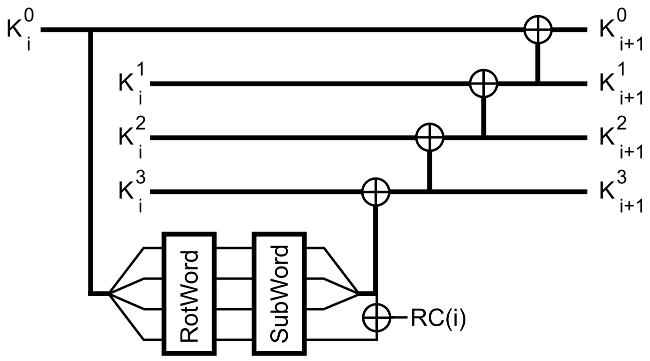

**Fig. 5.** AES 128-bit Key Expansion Round.

Let us finally mention that the decryption process slightly differs from the encryption one. In order to decipher data blocks, the inverse transformations have to be applied to the state. These operations are respectively called **Inv**-*Transform*, except for the **AddRounKey** as it is its own inverse, and are applied in the same order as described above. This may result in a different performance of the AES in encryption and decryption. Regarding the key schedule, the operations remain the same as for encryption but the subkeys have to be introduced in reverse order. For more details, we refer to [1].

## 3 Target Platform

The chosen platform for implementation is a Xilinx Virtex-5 FPGA. Nowadays, such devices embed programmable logic blocks, RAM memories and multipliers[2]. Compared to the early reconfigurable devices, these features allow recent FPGAs to provide interesting solutions for a wide range of applications. In this description, the focus will be set on logic elements as other resources will not be used. In Xilinx FPGAs, the slice is the logic unit that is used to evaluate a design's area requirement. The Virtex-5 platform exhibits two kinds of slices. Each of these contains 4 Look-Up Tables (LUTs), 4 flips-flops and some additional gates. These elements, defining the "basic" slice (sliceL), provide the user with logic, arithmetic and ROM functions. In addition, another version of the slice (sliceM) adds the capability to be configured as a distributed RAM or as a shift register. Those enhanced slices represent about 25% of the total amount of available slices. Figure 6 shows the difference between sliceLs and sliceMs.

## 4 Architecture

A lot of architectures for the AES have been presented in the open literature. Each of those target a specific application and accordingly, various tradeoffs of pipelining, unrolling, datapath width and S-boxes designs have been presented. These contributions do generally agree that the most critical part in an AES design is the S-box. To our knowledge, three different methods have been exploited in order to achieve efficient implementations:

**Logic.** In this first proposal, one 256×8-bit S-box is required for each byte of the state. Implementing this as a large multiplexor on platforms where LUTs provide 4-to-1 computation and by taking advantage of special FPGA configurations (*i.e.* MuxF5s and MuxF6s), led the authors of [4] to consume 144 LUTs for one S-box. In the case of a full length datapath (128-bit), 2304 LUTs (144×16) are required to perform the whole substitution. This method results in a logic depth of 2 slices. Those two levels can be advantageously pipelined to prevent frequency reduction.

[2] Additionally, certain devices also embed microprocessors (PowerPC).

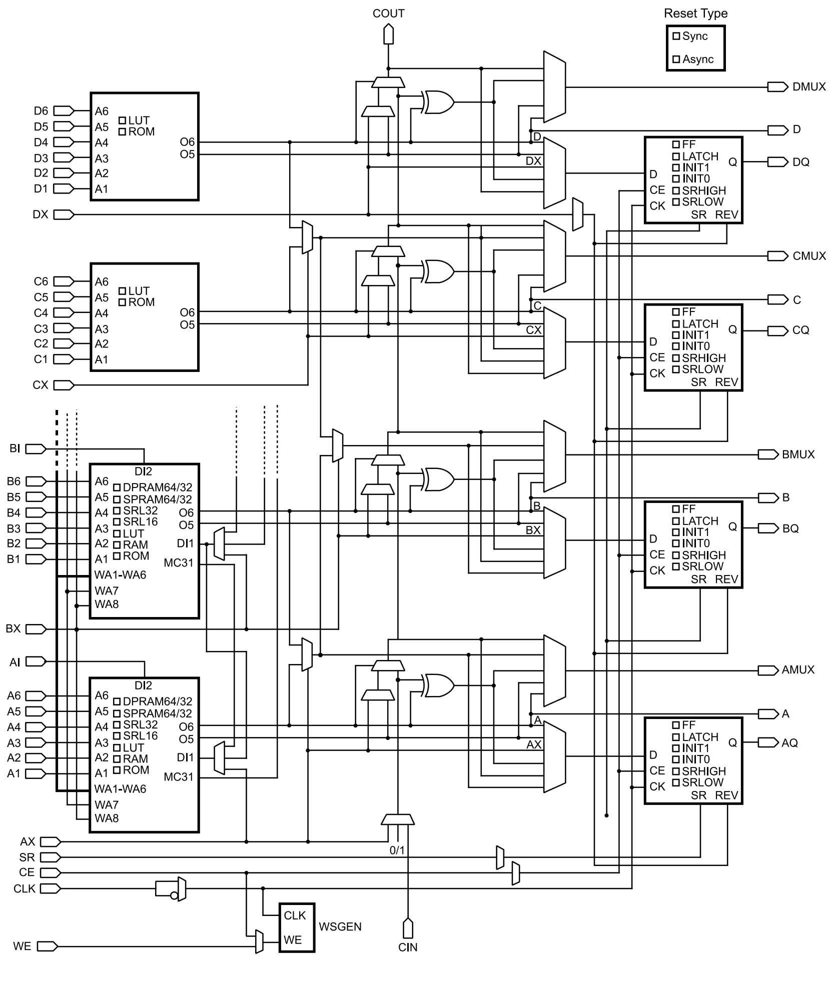

**Fig. 6.** Virtex-5 mixed view of top-half sliceL and bottom-half sliceM.

**Algorithmic.** Another approach to compute **SubBytes** is to directly implement the multiplicative inverse and affine transforms. In order to make it more efficient, Rijmen suggested in [5] to move computations from GF($2^8$) to the composite field GF($(2^4)^2$). The main advantage relies on the reduced size of the inversion table: $2^4 \times 4$ instead of $2^8 \times 8$. However, some logic is required to implement transformations to and from such composite field. This type of implementation has been exploited in [6, 7] for example.

**RAM.** Embedded BlockRAMs on FPGA platforms can also be used to store the S-boxes. Such an approach achieves high-throughput performances in [8]. If enough of these memories are available, another idea is to combine **SubByte** and **MixColumns** in a single memory table, as sometimes proposed in software implementations. Examples of such designs are in [9, 10]. Depending on the size and availability of RAM blocks, these solutions may be convenient in practice for recent FPGA devices.

In our context, the choice is straightforward. Due to the technology evolution, a $256\times8$-bit S-box fits in 32 LUTs. Using a similar approach as in [4], four LUTs make four 6-to-1 tables from which the correct output is chosen thanks to the F7MUXs and F8MUX of the slice. It allows packing a $256\times1$-bit table in four LUTs, that is a single slice. This solution has the significant advantage of both reducing the area requirements in LUTs and performing the S-box computation in a single clock cycle, thus reducing the latency.

The architecture developed for 128-bit key encryption is in Figure 7. It has a 128-bit datapath for both data and keys. The state of the cipher iterates over a single round structure. The **ShiftRows** operation is not shown on the figure below as it simply consists in routing resources.

As far as the key expansion is concerned and when dealing with large amount of data, like in the Digital Cinema context, computing all subkeys prior to en/decryption seems a better alternative than an “on-the-fly” key schedule. Indeed, the overhead due to a master key change quickly vanishes as the number of messages using this same key increases. It also allows us to reduce the area requirements of the complete design. As the key schedule and the encryption module use the same S-boxes, these are shared in our architecture. Multiplexors allow the S-box inputs to change between the state and the subkey. These multiplexors do not increase the implementation cost as they are packed in LUTs together with the key addition. The remaining of the subkeys'computation proceeds as explained in Section 2. Each subkey is written in a RAM configured sliceM. They are brought back when needed for en/de-cryption. We note that the key schedule must be performed before decryption takes place anyway, as the subkeys have to be applied in reverse order. Also, the S-Boxes sharing does not hold for the decryption architecture. Indeed, the S-boxes implementing **SubBytes** for the key schedule can not be reused for deciphering. This is because decryption involves the **InvSubBytes** operation that do not yield the same S-boxes.

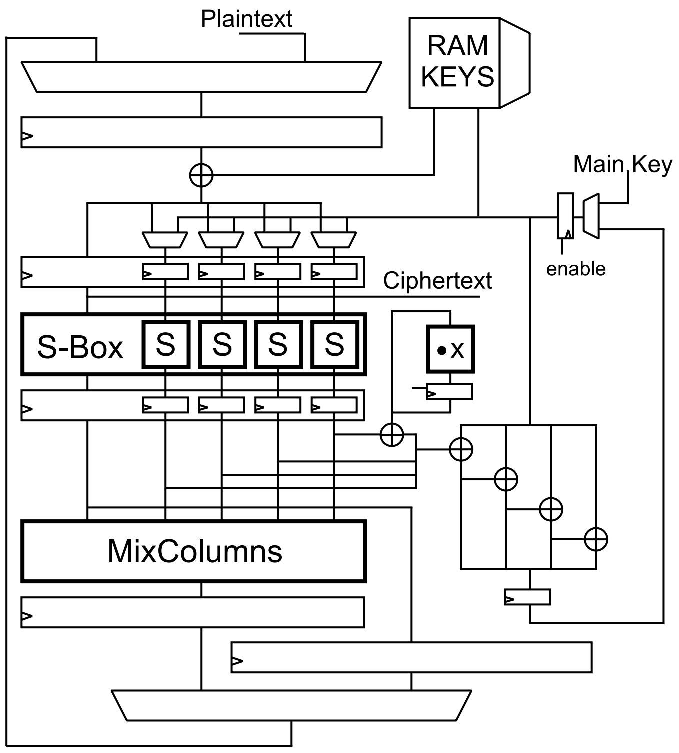

**Fig. 7.** AES Encryption Architecture.

This architecture perfectly suits the needs of ECB and Counter modes of operations. It could also be tuned to handle the CBC mode, at the cost of a reduced efficiency. Indeed, as the plaintext block is to be XORed with the previous ciphertext before being encrypted, the four pipeline stages of the round do not allow encryption of four plaintext blocks at the same time. Note that in the case of CBC decryption, this restriction does not hold as this additional XOR is performed after looping through the round.

Although the focus of this paper is the implementation of an AES en/decryption module on a Virtex-5, we also investigated this IP core in Virtex-4 and Spartan-3 FPGAs, for illustration/comparisons purposes. The architecture remains the same as the one presented here for the Virtex-5. The only difference relies on the way S-boxes are implemented. In the case of a Virtex-4, the S-boxes are made up of BlockRAMs. Each of the blockRAM has a datapath width of 32-bit that allows the output of the S-box to be stored times 0, 1, 2 and 3. That is, a part of the **MixColumns** computation is made while passing through the RAMs. To make things clear, let us consider the combination of SubBytes and MixColumns in the AES. An output column of this transform equals:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \times \begin{bmatrix} SB(a_0) \\ SB(a_1) \\ SB(a_2) \\ SB(a_3) \end{bmatrix},$$

where the $b_i$'s represent the transform output bytes and the $a_i$'s its input bytes. The $b_i$ vector is equivalent to:

$$\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \times [SB(a_0)] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \times [SB(a_1)] \oplus \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \times [SB(a_2)] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \times [SB(a_3)]$$

Therefore, if we define four tables as:

$$T_0(a) = \begin{bmatrix} 02 \times SB(a) \\ SB(a) \\ SB(a) \\ 03 \times SB(a) \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \\ SB(a) \end{bmatrix},$$
$$T_2(a) = \begin{bmatrix} SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} SB(a) \\ SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \end{bmatrix},$$

the combination of SubBytes and MixColumns equals:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = T_0(a) \oplus T_1(a) \oplus T_2(a) \oplus T_3(a)$$

In our Virtex-4 implementation, these $T$ tables are stored in RAM and all what is left to complete the **MixColumns** transform is a single level of logic handling the XOR of the four bytes. On Spartan-3 devices, the situation is different since the BlockRAMs do not provide a dedicated latch at their output. Reproducing the behavior of Virtex-4 requires 24-bits to be stored using the slice flip-flops which consumes much more area. Since XORing the table outputs without using the slice flip flops causes a reduction of the work frequency, the most efficient solution is to implement **MixColumns** and **SubBytes** independently.

## 5 Results

The AES designs were described using VHDL. Synthesis and Place & Route were achieved on Xilinx ISE 9.1i. The selected devices are Xilinx's Virtex-5, Virtex-4 and Spartan-3. Table 1 summarizes the results achieved for both the encryption and decryption (Enc/Dec) modules. Moreover, some previous results are summarized in Table 2. As it is generally true for any comparison of hardware performances, those results have to be taken with care since they relate to different FPGA devices. In the Virtex-5 FPGAs, a slice is made up of 4 LUTs instead of 2 for previous Xilinx devices. In order to allow fair(er) comparison, it then makes sense to double the figures as if a slice was 2 LUTs. This is taken into account into the parenthesis of Table 1. Compared to previous devices, the benefit of Virtex-5 is easily underlined. It corresponds either to the removal of the blockRAMs from the design on the Virtex-4 or a 50% slice reduction from a full logic design on Spartan-3 FPGAs. This strongly emphasized the advantage of technology evolution shifting from 4 to 6 input bits LUTs.

**Table 1.** Implementation Results: encryption/decryption designs.

| Device | Slices | BRAM | Freq. (MHz) | Thr. (Gbps) | Thr. / Area (Mbps/slice) |
|---|---|---|---|---|---|
| Virtex-5 | 400 / 550 (800 / 1100) | 0 | 350 | 4.1 | 10.2 / 7.4 |
| Virtex-4* | 700 / 1220 | 8 | 250 | 2.9 | 4.1 / 2.3 * |
| Spartan-3 | 1800 / 2150 | 0 | 150 | 1.7 | 0.9 / 0.8 |

Additional insights on our implementation results can be obtained by looking at Table 2. Namely, the proposed architectures range among the efficient ones found out in the literature. Again, these observations have to be considered as general intuitions rather than fair comparisons since they consider different FPGA technologies: more recent FPGAs have higher work frequencies and thus throughput. In addition, the hardware efficiency (*i.e.* throughput/area ratio) of the *-marked implementations is not meaningful since they consumes FPGA RAM blocks. Finally, the hardware cost can only be compared if the respective implementation efficiencies (*e.g.* measured with the throughput/area ratio) are somewhat comparable. As a matter of fact, it is always possible to reduce the

**Table 2.** Previous Implementations.

| Device | Datapath | Slices | BRAM | Freq. (MHz) | Thr. (Gbps) | Thr./Area (Mbps/slice) |
|---|---|---|---|---|---|---|
| Spartan-2 [11]* | 8 | 124 | 2 | – | 0.002 | 0.02* |
| Virtex-2 [10]* | 32 | 146 | 3 | 123 | 0.358 | 2.45* |
| Virtex-E [12]* | 128 | 542 | 10 | 119 | 1.45 | 2.67* |
| Virtex-E [4] | 128 | 2257 | 0 | 169 | 2.0 | 0.88 |
| Virtex-2 [13]* | 128 | 387 | 10 | 110.16 | 1.41 | 3.64* |
| Virtex-2 [13] | 128 | 1780 | 0 | 77.91 | 1.0 | 0.56 |
| Virtex-4 [14] | 128 | 18400 | 0 | 140 | 17.9 | 0.97 |
| Virtex-5 [15] | 128 | 349 | 0 | 350 | 4.1 | 11.67 |

implementation cost, by considering smaller datapaths (*e.g.* [11] uses an 8-bit datapath for the AES, [10] uses a 32-bit datapath, all the others use 128-bit architectures) at the cost of a reduced throughput.

In the case of Helion Technology's implementation [15], the comparison is more interesting since it relates to the same Virtex-5 platform as ours. At first sight, their Fast AES Encryption core seems to consume less area than the proposed architecture. However, the gap can be reduced if we assume that their core uses an "on-the-fly" key schedule. In such a case, the distributed RAM used to store subkeys is to be removed from our presented design (along with its control logic) which allows to earn at least 32 slices. This makes both designs very close. As a matter of fact, the differences between these cores mainly relate to different optimization goals. Our was to design encryption and decryption IPs exploiting a very similar architecture with a key scheduling algorithm executed prior to the encryption/decryption process. We note that not using the "on-the-fly" key scheduling for encryption makes sense for power consumption reasons. If the implementation context does not require frequent key changes, there is no need to re-compute these keys for every plaintext.

## 6 Conclusion

This paper reports implementation results of the AES algorithm on the new Virtex-5 devices. It exhibits the (straightforward but significant) benefits that can be drawn from the technology evolution within recent FPGAs. In particular it is shown how the AES substitution box perfectly suits the new Virtex-5 slice structure using 6-bit LUTs. This enables reducing the cost of a single S-box from 144 down to 32 LUTs ! Compared to 4 input bit LUTs-based designs, this advantage roughly corresponds to either the removal of blocks of embedded RAM memory or a slice count reduction of 50%, depending on the design choices. The proposed architectures range among the most efficient ones published in the open literature. Their reasonable implementation cost make them a suitable solution for a wide range of application requiring Gbps-range throughput, including digital cinema and network encryption.

## References

1. National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standards Publications – FIPS 197, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, November 2001.
2. K. Järvinen, M. Tommiska, and J. Skyttä. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. In *IEE Proceedings on Information Security*, volume 152, pages 3 – 12, October 2005.
3. Digital Cinema Initiative. DCI System Specification, V1.0, July 20 2005.
4. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. Efficient Implementation of Rijndael in Reconfigurable Hardware: Improvements and Design Tradeoffs. In *Cryptographic Hardware and Embedded Systems — CHES 2003*, pages 334–350. Springer, September 2003.
5. Vincent Rijmen. Efficient implementation of the Rijndael S-box. `www.citeseer.ist.psu.edu/rijmen00efficient.html`.
6. Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi. Efficient rijndael encryption implementation with composite field arithmetic. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 171–184, 2001.
7. A. Hodjat and I. Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES coprocessor on FPGA. In *12th. Annual IEEE Symposium on Field Programmable Custom Computing Machine, FCCM'04*, pages 308–309, April 2004.
8. G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo. An FPGA-based Performance Analysis of the Unrolling, Tiling and Pipelining of the AES Algorithm. In *13th. International Conference on Field Programmable Logic and Applications — FPL'03*, pages 292–302, September 2003.
9. M. McLoone and J. V. McCanny. Rijndeal FPGA Implementation Utilizing Look-Ups Tables. In *IEEE Workshop on Signal Processing Systems — SIPS'01*, pages 349–360, September 2001.
10. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Well Suited for Small Embedded Applications. In *International Symposium on Information Technology: Coding and Computing, ITCC 2004*, pages 583 – 587. IEEE Computer Society, April 2004.
11. M. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, pages 427 – 440. Springer, August—September 2005.
12. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL. In *11th. ACM International Symposium on Field-Programmable Gate Arrays — FPGA'03*, pages 216–224, February 2003.
13. J. Zambreno, D. Nguyen, and A. Choudary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and Applications*, pages 575 – 585. Springer, August–September 2004.
14. Stefan Lemsitzer, Johannes Wolkerstorfer, Norbert Felbert, and M. Braendli. Multi-gigabit GCM-AES Architecture Optimized for FPGAs. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES'07*, pages 227–238, 2007.
15. Helion. `www.heliontech.com`.

16. D. Canright. A Very Compact S-box for AES. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, pages 441–456. Springer, August – September 2005.
17. P. Chodowiec and K. Gaj. Very Compact FPGA Implementation of the AES Algorithm. In *Cryptographic Hardware and Embedded Systems — CHES 2003*, pages 319–333. Springer, September 2003.
18. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithms. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, pages 357 – 370. Springer, August 2004.
19. IP Cores. `http://http://ipcores.com/index.htm`.
20. M. McLoone and J. V. McCanny. High Performance Single-Chip FPGA Rijndael Algorithm Implementation. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES'01*, pages 65–76, May 2001.
21. M. McLoone and J. V. McCanny. Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm. In *11th. International Conference on Field-Programmable Logic and Applications — FPL'01*, pages 152–161, August 2001.
22. A. Satoh and S. Morioka. Unified Hardware Architecture for the 128-bit Block Ciphers AES and Camellia. In *Cryptographic Hardware and Embedded Systems — CHES 2003*, pages 304–318. Springer, September 2003.
23. N. A. Saqib, F. Rodríquez-Hendríquez, and A. Díaz-Pérez. AES Algorithm Implementation–An Efficient Approach for Sequential and Pipeline Architectures. In *4th. Mexican International Computer Science — ENC'03*, pages 126–130, September 2003.