Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs

Francois-Xavier Standaert, Gael Rouvroy,
Jean-Jacques Quisquater, Jean-Didier Legat

UCL Crypto Group
Laboratoire de Microelectronique
Universite Catholique de Louvain
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium
standaert,rouvroy,quisquater,legat@dice.ucl.ac.be

Abstract. Performance evaluation of the Advanced Encryption Standard candidates has led to intensive study of both hardware and software implementations. However, although plentiful papers present various implementation results, it seems that efficiency could still be greatly improved by applying good design rules adapted to devices and algorithms. This paper addresses various approaches for efficient FPGA implementations of the Advanced Encryption Standard algorithm. As different applications of the AES algorithm may require different speed/area tradeoffs, we propose a rigorous study of the possible implementation schemes, but also discuss design methodology and algorithmic optimization in order to improve previously reported results. We propose heuristics to evaluate hardware efficiency at different steps of the design process. We also define an optimal pipeline that takes the place and route constraints into account. Resulting circuits significantly improve previously reported results: throughput is up to 18.5 Gbits/sec and area requirements can be limited to 542 slices and 10 RAM blocks with a ratio throughput/area improved by at least 25% of the best-known designs in the Xilinx Virtex-E technology.

1 Introduction

In October 2000, NIST (National Institute of Standards and Technology) selected Rijndael [2] as the new Advanced Encryption Standard. The selection process included performance evaluation on both software and hardware platforms. Many hardware architectures were proposed [3 − 16], but most of them were simple implementations according to the Rijndael specification. More recently, design strategies and implementation approaches were proposed for the implementation of block ciphers in reconfigurable hardware [17, 18] while other papers focused on some interesting algorithmic optimizations, specially for the highly expensive substitution box of Rijndael [19–21]. This paper addresses various approaches for FPGA implementations of the Advanced Encryption Standard algorithm and combines recent observations about Rijndael in efficient designs. As different applications of the AES algorithm may require different speed/area

* This work has been funded by the Walloon region (Belgium) through the research project TACTILS.

tradeoffs, we propose a rigorous study of the possible implementation schemes, but also discuss design methodology and algorithmic optimization in order to improve previously reported results. We first discuss the implementation of the substitution box and linear diffusion layer at the algorithmic level. Then we examine different possible architectures and optimizations. Finally, we present heuristics allowing to evaluate the efficiency of our architectures at different steps of the design process. Synthesis and implementation constraints of FPGAs are taken into account in order to define maximum and optimal pipeline. We apply these notions to loop and unrolled architecture in order to improve circuits performances and compare our results to the best designs reported in literature. The main contribution of this paper has to be found in the improvement of hardware efficiency that we define as the ratio throuhput/area: efficiency of best-known unrolled architectures is improved by 35% while efficiency of best-known loop architectures is improved by at least 25% in the Xilinx Virtex-E technology.

This paper is structured as follows. The description of the hardware, synthesis tool and implementation tool is in section 2. Section 3 gives a short mathematical description of Rijndael and we propose an efficient representation of the key schedule by means of a key round. The main contribution of this paper lies in section 4 where we discuss the possible implementation tradeoffs. Section 4.1 deals with design methodology and defines hardware efficiency and maximum pipeline for FPGAs. Section 4.2 presents possible algorithmic optimization of Rijndael. Different schemes for the substitution box are proposed and the diffusion layer is combined with the key addition. Section 4.3 proposes different architectures for various speed/area tradeoffs: loop architectures and unrolled architectures are studied and implemented. Finally, section 4.4 defines optimal pipeline for FPGAs as well as a heuristic rule to reach it. Practical results and comparisons with best known published designs are in section 5 and conclusions are in section 6.

2 Hardware description

All our implementations were carried out on a XILINX VIRTEX3200ECG1156-8 FPGA. We chose this technology in order to allow relevant comparisons with the best-known FPGA implementations of Rijndael. In this section, we briefly describe the structure of a VIRTEX FPGA as well as the synthesis and implementation tools that were used to obtain our results.

Configurable Logic Blocks (CLB's): The basic building block of the VIRTEX logic block is the logic cell (LC). An LC includes a 4-input function generator, carry logic and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each VIRTEX CLB contains four LC's, organized in two similar slices. Figure 1, shows a detailed view of a single slice. Virtex function generators are implemented as 4-input look-up tables (LUTs). In addition to operate as a function generator, each LUT can provide a 16×1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16×2-bit or 32×1-bit synchronous RAM or a 16×1-bit dual port synchronous RAM. The VIRTEX LUT

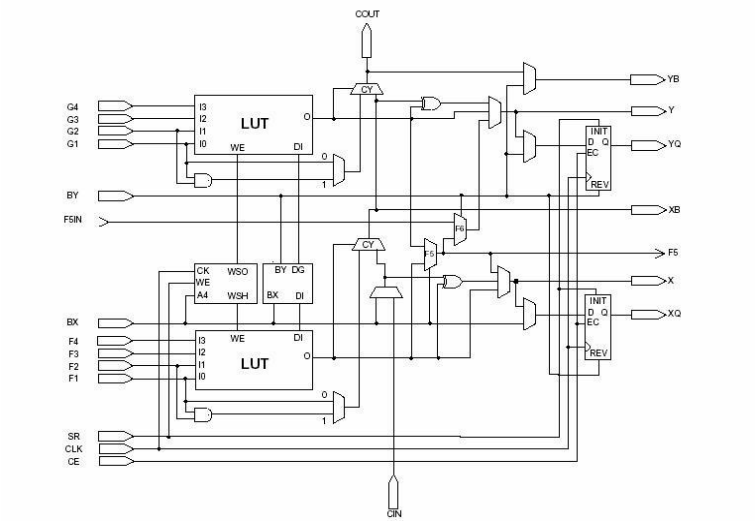

Fig. 1. The VIRTEX slice.

can also provide a 16-bit shift register.

The storage elements in the VIRTEX slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing function generators.

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine bits. Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions up to 19 bits. The arithmetic logic also includes a XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementations.

Finally, VIRTEX FPGAs incorporate several large RAM blocks. These complement the distributed LUT implementations of RAM's. Every block is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently.

Our hardware: A VIRTEX3200ECG1156-8 FPGA contains 32448 slices and 208 RAM blocks, which means 64896 LUTs and 64896 flip-flops. In the next sections, we compare the number of LUTs, registers and slices. We also evaluate the delays and frequencies thanks to our synthesis tool. The synthesis was performed with FPGA Compiler 2 3.7.1 (SYNOPSYS) and the implementation with XILINX ISE-5. Finally, our circuit models were described using VHDL.

3 Block cipher description

Rijndael is an iterated block cipher that operates on a 128-bit cipher state and uses a 128-bit key[1]. It consists of a serie of 10 applications of a key-dependent round transformation to the cipher state. In the following, we will individually define the component mappings and constants that build up Rijndael, then specify the complete cipher in terms of these components.

Representation: The state and key are represented as a square array of 16 bytes. This array has 4 rows and 4 columns. It can also be seen as a vector in $GF(2^8)^{16}$. Let s be a cipher state or a key $\in GF(2^8)^{16}$, then s_i is the i-th byte of the state s and $s_i(j)$ is the j-th bit of this byte.

SubBytes, the non-linear layer γ: The SubBytes transformation is a non-linear byte substitution, operating on each byte independently. The substitution table (or s-box) is invertible and is constructed by the composition of two operations:

1. The multiplicative inverse in $GF(2^8)$.
2. An affine transform over $GF(2)$.

Every byte is therefore considered as a polynomial with coefficients in $GF(2)$:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$
$$b_7b_6b_5b_4b_3b_2b_1b_0 \rightarrow b(x) \tag{1}$$

Then SubBytes consists of the parallel application of this s-box S:

$$\gamma(a) = b \Leftrightarrow b_i = S[a_i], 0 \leq i \leq 15 \tag{2}$$

The ShiftRows transformation δ: In ShiftRows, the rows of the state are cyclically shifted over different offsets. Row 0 is not shifted, row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes.

The MixColumns transformation θ: In MixColumns, the columns of the state are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by:

$$c(x) =' 03'x^3 +' 01'x^2 +' 01'x +' 02' \tag{3}$$

The polynomial is coprime to $x^4 + 1$ and therefore is invertible. This can be written as a matrix multiplication:

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

[1] Actually, there exist several versions of Rijndael with different block and key lengths, but we focus on this one.

Where (b_3, b_2, b_1, b_0) is a four-byte column of the state. An output byte of MixColumns (for example b_0) can be expressed as:

$$b_0 =' 02' \times a_0 \oplus' 03' \times a_1 \oplus' 01' \times a_2 \oplus' 01' \times a_3$$

We also define a function X, corresponding to the multiplication with '02' modulo the irreductible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$: $X : GF(2^8) \rightarrow GF(2^8) : X(a) = b \Leftrightarrow$

$$\begin{aligned}
b(7) &= a(6) \\
b(6) &= a(5) \\
b(5) &= a(4) \\
b(4) &= a(3) \oplus a(7) \\
b(3) &= a(2) \oplus a(7) \\
b(2) &= a(1) \oplus a(7) \\
b(1) &= a(0) \\
b(0) &= 0 \oplus a(7)
\end{aligned}$$

The round key addition $\sigma[K]$: In this operation, a round key is applied to the state by a simple bitwise EXOR. The round key is derived from the cipher key by means of the key schedule. The round key length is equal to the block length.

$$\sigma[k](a) = b \Leftrightarrow b_i = a_i \oplus k_i, 0 \leq i \leq 15 \tag{4}$$

The round transformation $\rho[K]$: The round transformation can be written as a composition of the four previous transformations:

$$\rho[K] = \sigma[K] \circ \theta \circ \delta \circ \gamma = \sigma[K]\big(\theta(\delta(\gamma))\big) \tag{5}$$

The key schedule: The round keys are derived from the cipher key by means of the key schedule. This consists of two transformations: the key expansion and the round key selection. In our description, SubWord (SW) is a function that takes a 4-byte word in which each byte is the result of applying the Rijndael s-box. The function RotWord (RW) returns a word in which the bytes are a cyclic permutation of those in its inputs such that the input word (a, b, c, d) produces the output word (b, c, d, a). Finally, $RC(i)$ is an 8-bit round constant for the round i.

The key schedule can be easily described by the use of a key round β that takes four 4-byte input words, corresponding to a 128-bit key, and produces four 4-byte output words. The first round key K_0 is the cipher key, then, we have:

$$K_{i+1} = \beta(K_i), i = 0, ..., 10 \tag{6}$$

Figure 2 illustrates the key round of Rijndael.

The complete cipher: Rijndael is defined for the cipher key K as the transformation Rijndael[K]= $\alpha[K_0, K_1, ..., K_{10}]$ applied to the plaintext where:

$$\alpha[K_0, K_1, ..., K_{10}] = \sigma[K_{10}] \circ \delta \circ \gamma \circ (\bigcirc_{r=1}^{9} \rho[K_r]) \circ \sigma[K_0] \tag{7}$$

Our implementations are based on this description of AES Rijndael.

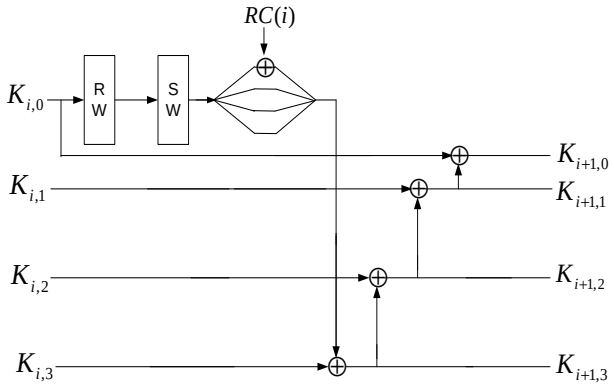

Fig. 2. The key round β.

4 Implementation tradeoffs

The optimization methods and the resulting implementation tradeoffs for the implementation of AES Rijndael can be divided into two classes: architectural and algorithmic optimization. Algorithmic optimization exploits algorithmic strength inside each round unit. Architectural optimization exploits design techniques such as pipelining, loop unrolling and sub-pipelining.

This paper first considers **loop architectures**, where only a small number m (typically $m = 1$) of rounds are independently implemented in hardware. Loop architectures enables small area circuits but have low throughput. Then we improve the throughput at the cost of increased area by the combination of loop unrolling and pipelining. **Unrolled architectures** have a large number m of rounds (typically all) that are independently implemented in hardware. **Pipelining** increases the encryption speed by processing multiple blocks of data simultaneously. It is achieved by inserting rows of registers among combinatorial logic. Parts of logic between two consecutive registers form pipeline stages. In case of block ciphers, each round constitutes a pipeline stage. Finally, **sub-pipelining** is similar to pipelining but also inserts registers inside the round functions.
Concerning algorithmic optimizations, we focused on the critical parts of Rijndael. Different schemes for the substitution box are proposed and compared. We also underline interesting combinations of the MixColumns θ with the key addition $\sigma[K]$. In this section, we propose a rigorous study of the possible tradeoffs for implementing Rijndael.

4.1 Design methodology

In [18], a methodology to implement block ciphers in reconfigurable hardware is presented, based on simple digital design rules applied to iterated block ciphers. Looking at the round functions of iterated block ciphers, it is observed that they are mainly built on simple algebraic or logic operations. Therefore, the sub-pipelining of round functions is mandatory if efficient designs are wanted. Practically, the designer can easily keep his critical path inside one CLB slice. Moreover, looking at the CLB strcture, it can be seen that FPGAs involve specific

constraints that have to be taken into account if an optimal design is wanted. As the slice of Figure 1 is divided into logic elements and storage elements, an efficient implementation will be the result of a better compromise between combinatorial logic used, sequential logic used and resulting performances. These observations lead to different definitions of implementation efficiency:

1. In terms of performances, let the efficiency of a block cipher be the ratio $Throughput\ (Mbits/s)/Area\ (slices)$.
2. In terms of resources, the efficiency is easily tested by computing the ratio $Nbr\ of\ LUTs/Nbr\ of\ registers$: it should be close to one.

Our implementations of Rijndael were designed in order to maximize these notions of hardware efficiency. It practically results in the sub-pipelining of every component of the round functions. The next section studies algorithmic optimizations combined with good sub-pipelining. For this purpose, we define the **maximum pipeline** as the pipeline of which number of stages implies that the ratio $Nbr\ of\ LUTs/Nbr\ of\ registers$ is the closest to one (and lower than one).

4.2 Algorithmic optimizations: a first tradeoff

A. Implementing the substitution box: The Rijndael S-box is a non-linear byte substitution used 200 times in Rijndael with 128-bit block length and key length. It is invertible and is constructed by the composition of two transformations:

1. The mapping $x \rightarrow x^{-1}$, where x^{-1} represents the multiplicative inverse in the field GF(2^8).
2. An affine transformation over GF(2): $x \rightarrow Ax + b$, where A and b are constants.

In terms of hardware resources, the substitution box is the most expensive part of Rijndael. As a consequence, its implementation is a critical part in the design of an efficient encryption core. This transform can be implemented following different schemes. We propose to observe three possibilities and the resulting constraints.

A1. The multiplexor model: A first and obvious solution is to consider SubBytes as a large multiplexor and take advantage of special FPGA configurations to implement these ones. Figure 3 illustrates the implementation of an output bit of the Rijndael s-box. We pipelined γ by inserting two register levels so that the critical path corresponds to one 4-input LUT, one multiplexor F5 and one multiplexor F6. Table 1 summarizes the synthesis results for the non-linear

Component	Nbr of LUT	Nbr of registers
γ	$144 \times 16 = 2304$	$42 \times 16 = 672$

Table 1. Synthesis of the non-linear layer γ.

transform γ where the s-box is repeated 16 times.

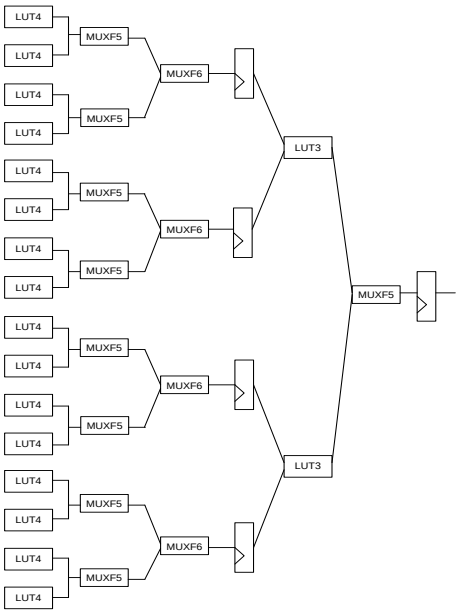

Fig. 3. The substitution box γ.

A2. RAM-based implementation: Another possibility is to use the RAM blocks available inside the VIRTEX to implement substitution boxes. The resulting SubBytes transform uses 8 RAM blocks and is performed in one clock cycle.

A3. Composite field solution: In AES Rijndael, every byte represents an element in the finite field GF(2^8). It can also be represented as a polynomial of degree 8 in the field GF(2): $b_7.x^7+b_6.x^6+b_5.x^5+b_4.x^4+b_3.x^3+b_2.x^2+b_1.x^1+b_0$. Addition and substraction of polynomials are given by the sum modulo 2 of the coefficients of both terms (bitwise XOR). Multiplication in GF(2^8) corresponds to multiplication of polynomials modulo an irreductible binary polynomial of degree 8. Rijndael uses $m(x) = x^8+x^4+x^3+x+1$. As the irreductible polynomial is used to construct the field and there are different irreductible polynomials of degree 8, several finite fields can be considered and generate different representations of Rijndael. These fields are isomorphic which means that there is a one-to-one mapping from one representation of Rijndael to another. Finally, the multiplicative inverse of a polynomial $b(x)$ is defined such that:

$$b(x).b^{-1}(x) = 1.mod.m(x) \tag{8}$$

In [19], subfield arithmetics are used to propose efficient implementations of Galois Field arithmetic, especially in the context of the Rijndael block cipher. Computations in the field GF(2^8) are replaced by computations in the composite

field $\mathrm{GF}(2^4)^2$ in order to reduce the size of the tables needed for the inversion. Basically, the idea is to consider our polynomial of degree 8 in the field $\mathrm{GF}(2)$ as a polynomial of degree 2 in the field $\mathrm{GF}(2^4)$, say $a_1x + a_0$, where $a_0, a_1 \in GF(2^4)$. The multiplicative inverse of $a_1x + a_0$ is computed in the field $GF(2^4)^2$ as a polynomial $b_1x + b_0$ such that:

$$(a_1x + a_0) \times (b_1x + b_0) = 1.mod.P(x) \tag{9}$$

Where $P(x)$ is an irreductible polynomial and coefficients b_0, b_1 can be expressed as follows:

$$\begin{aligned} b_1 &= a_1.(a_0^2 + a_1a_0 + \Delta a_1^2)^{-1} \\ b_0 &= (a_0 + a_1).(a_0^2 + a_1a_0 + \Delta a_1^2)^{-1} \end{aligned} \tag{10}$$

[19] gives details about parameter Δ and polynomial $P(x)$ as well as an affine transform that maps elements of $GF(2^8)$ to elements of $GF(2^4)^2$. We implemented the resulting composite field s-box as represented in Figure 4. We in-

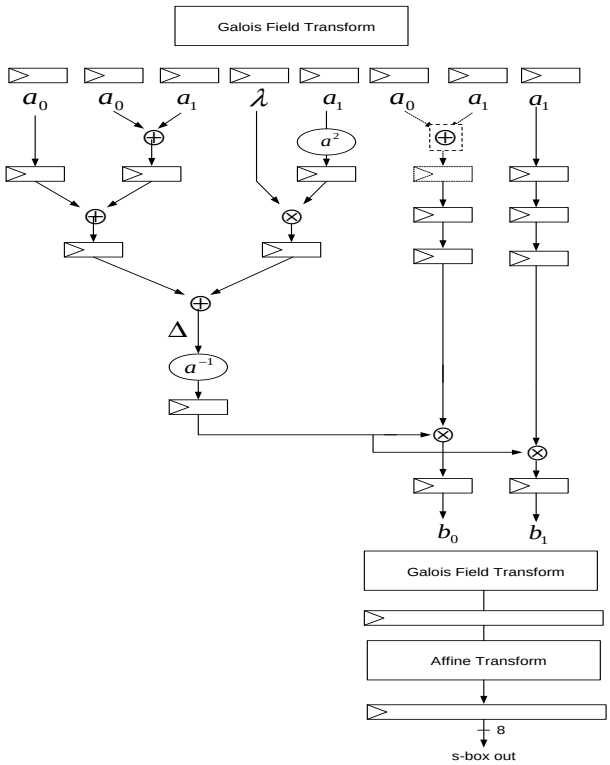

Fig. 4. The composite substitution box.

serted seven pipeline levels in order to get the ratio *Nbr of LUTs/Nbr of registers* close to one. Remark that this representation of the substitution box allows to keep the whole design unchanged as the Galois Field transform is used

Component	Nbr of LUT	Nbr of registers
γ	$84 \times 16 = 1344$	$76 \times 16 = 1216$

Table 2. Synthesis of the composite non-linear layer γ.

twice in order to be compatible with other transforms. Table 2 summarizes the synthesis results for the composite non-linear transform γ where the s-box is repeated 16 times. Compared to the multiplexor model, we have traded LUTs for registers, and obtained a better efficiency.

B. Implementing the other components: the Mixadd combination.

B1. The ShiftRows transform δ: This is just routing information and takes no place in the design.

B2. The MixColumns transform θ: Mixcolum operates on a 4-byte column and corresponds to multiplications and additions in $GF(2^8)$. For example, for the output byte b_0, we have:

$$b_0 =' 02' a_0 +' 03' a_1 +' 01' a_2 +' 01' a_3 \tag{11}$$

We implemented multiplications with a function X that corresponds with the multiplication with '02', modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Figure 5(a) illustrates the function X. Note that output bits 0,2,5,6,7

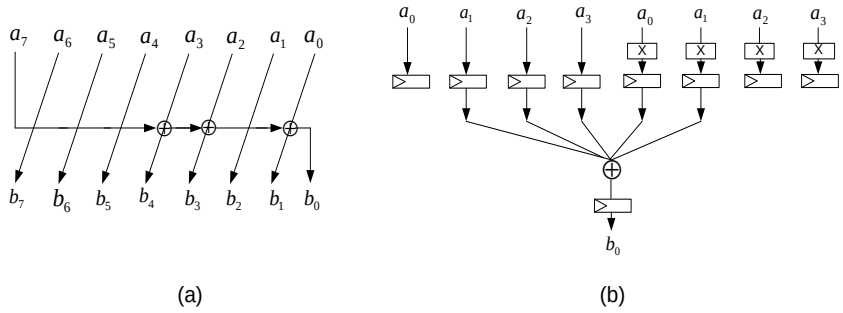

Fig. 5. (a) The function X. (b) Output byte b_0 of MixColumns.

just correspond to input bits shifted. Only 3 bits are modified by an EXOR operation. From this, we can easily represent an output byte of θ as shown in Figure 5(b):

$$b_0 = X(a_0) \oplus X(a_1) \oplus a_1 \oplus a_2 \oplus a_3 \tag{12}$$

Interesting combinations between MixColumns and the key addition can be performed when observing the structure of the Virtex slice (see Figure 1). Indeed, we observe that a slice offers the possibility to perform an EXOR between 5 bits: four bits are managed by the LUT and the last one by an EXOR gate next to the LUT. Our Mixadd transform takes advantage of this configuration and keeps the critical path inside one Virtex slice.

B3. The Mixadd transform ϵ: In Figure 5(b), we observe that an output byte of θ is obtained by a bitwise EXOR between 5 bytes: 3 are input bytes and the remaining ones are output bytes of function X. However, looking at the bit level, we know that 5 output bits of X are just shifted input bits. For these ones, only one register is needed to pipeline the diffusion layer.

For the 3 remaining bits, there is an additional EXOR inside the function X. Therefore, for these bits, we compute the bitwise EXOR between the 3 left bytes of Figure 5(b) and the output bits of X independently. Then we insert a register. A bitwise EXOR operation remains to be carried out and we combine it with the key addition. The resulting Mixadd transformation only needs two register levels to keep a critical path inside one slice.

Figure 6(a) illustrates the combination of MixColumns and Addroundkey at the bit level. Finally, Table 3 summarizes the synthesis results for the Mixadd

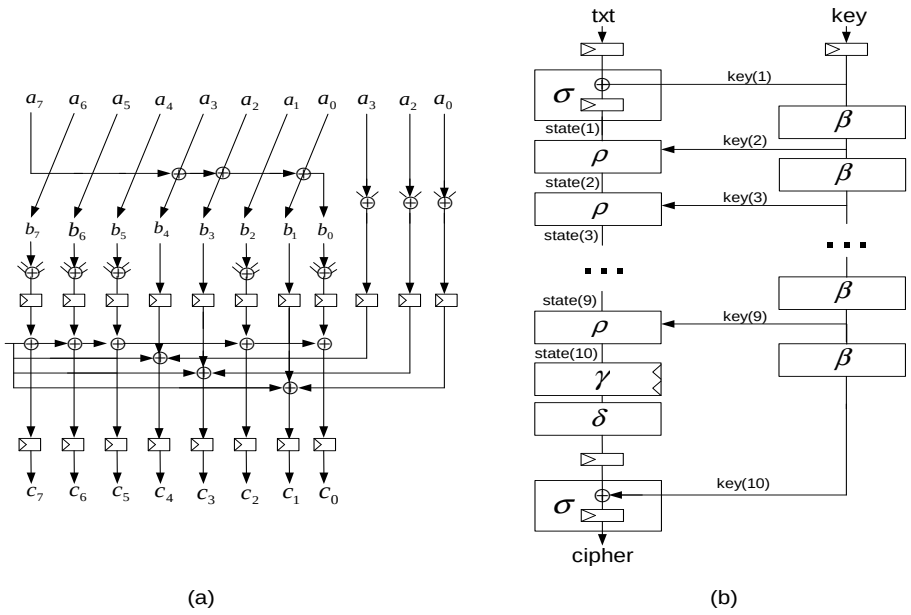

Fig. 6. (a) Mixadd transform at the bit level. (b) AES Rijndael: unrolled architecture.

transformation.

Component	Nbr of LUT	Nbr of registers
ϵ	304	304

Table 3. Synthesis of the Mixadd transform ϵ.

4.3 Implementation schemes: a second tradeoff

Depending on different optimization criteria, different architectures can be employed. Optimization for maximum speed can be achieved by a fully pipelined unrolled architecture. In the applications requiring minimum area, a loop architecture with only one round implemented seems to be the best choice. In both cases, we tried to maximize the efficiency defined in section 4.1.

Our implementations of AES Rijndael directly results from the previous component descriptions. For high throughput constraints, we implemented a pipeline version that unrolls the 10 AES rounds illustrated in Figure 6(b). For low

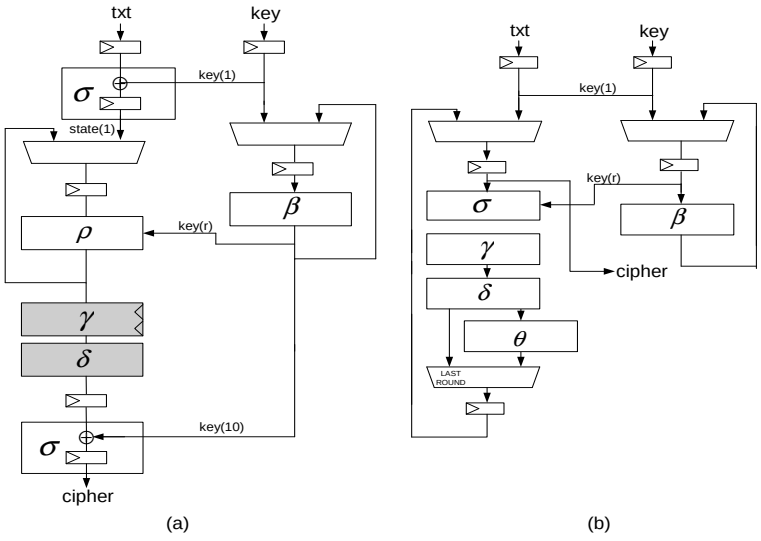

Fig. 7. AES Rijndael: loop architecture (a) and (b).

area constraints, we propose sequential implementations with only one unrolled round. Figure 7(a) uses the optimized combination of MixColumns and addkey and its grey functions are actually included into the round ρ. Figure 7(b) modifies the round structure so that the initial and final key additions are managed inside the round. It is important to remark that the modification of the round structure implies the loss of our mixadd combination and needs an additional multiplexor for the last round of the algorithm. As a consequence it presents no practical advantage in our FPGA implementations but would probably be the best choice for ASICs where the CLB structure does not exist and for which the mixadd optimization is therefore not relevant.

For all our proposals, we evaluated the hardware cost in terms of LUTs, registers and slices as well as the frequency results. These results are estimated after implementation, using XILINX ISE5.

4.4 Optimal pipeline: a third tradeoff

The design methodology and algorithmic optimization of previous sections allowed us to reach very interesting frequencies after synthesis. However, the implementation (and specially the routing task) of large designs was a critical constraint in our designs. Practically, our most pipelined circuits presented surprising delays including 20% of logic and 80% of routes. We concluded that the real bottleneck of such large ciphers is the difficulty of having an efficient place and route: in case of complex circuits, high pipelining is not mandatory. Moreover, as the difficulty of the place and route task is hardly evaluated, a new practical problem is to find the best tradeoff between good synthesis results and good implementation results. We propose the heuristic of Algorithm 1 to solve this last optimization problem. This heuristic led us to the optimized results of the next section where we mention the optimal number of pipeline stages.

Algorithm 1 Optimal pipeline search

1. Start from the maximal pipeline defined in section 4.1, i.e. implement Rijndael with the best ratio $Nbr\ of\ LUTs/Nbr\ of\ registers$;
2. After implementation, compute the efficiency $E_{cur} = Throughput\ (Mbits/s)/Area\ (slices)$;
3. $OK = 0$;

While $OK = 0$ **do** {

1. Remove the pipeline stage that involves the lowest frequency reduction and re-implement Rijndael;
2. After implementation, compute the efficiency $E_{nxt} = Throughput\ (Mbits/s)/Area\ (slices)$;
3. **If** $E_{cur} \geq E_{nxt}$ **then** $OK = 1$;
   else $E_{cur} = E_{nxt}$;

}

4. The final efficiency E_{cur} specifies the optimal pipeline;

5 Practical results and comparisons

In order to take every possible tradeoff into account in this section, we list our results for different architectures and different substitution boxes. The tables presented are based on the optimal pipeline defined in previous section. Loop architectures (Figure 7(a)) are in Table 4. Unrolled architectures (Figure 6(b)) are in Table 5.

Type	Nbr of LUT	Nbr of reg.	Nbr of slices	RAM blocks	Latency (cycles)	Output every (cycles)	Freq. after Impl. (Mhz)	Throughput (Mbits/sec)
LUT-based γ	3846	2517	2257	0	52	5/52	169	2008
RAM-based γ	877	668	542	10	21	2/21	119	1450
Composite γ	2524	2185	1767	0	82	8/82	167	2085

Table 4. Rijndael encryption: loop architectures on VIRTEX3200E.

Type	Nbr of LUT	Nbr of reg.	Nbr of slices	RAM blocks	Latency (cycles)	Output every (cycles)	Freq. after Impl. (Mhz)	Throughput (Mbits/sec)
LUT-based γ	33712	14592	19072	0	42	1	86	11008
RAM-based γ	3516	3840	2784	100	21	1	92	11776
Composite γ	19752	13479	15112	0	72	1	145	18560

Table 5. Rijndael encryption: unrolled architectures on VIRTEX3200E.

Type	Nbr of LUTs	Nbr of slices	RAM blocks	Throughput (Mbits/s)	Throughput/Area ($\frac{Mbits/s}{slices,LUTs}$)
McLoone et al. [9]	/	2222	100	6956	3.1
Our design	**3516**	**2784**	**100**	**11776**	**4.2**
Helion tech. [10]	899	/	10	1187	1.32
Our design	**877**	**542**	**10**	**1450**	**1.65**
Satoh et al. composite [15]	/	1880	0	589	0.31
Our design	**2524**	**1767**	**0**	**2085**	**1.17**
Satoh et al. mux [15]	/	2529	0	833	0.33
Our design	**3846**	**2257**	**0**	**2008**	**0.88**

Table 6. Comparisons with other implementations on VIRTEX-E technology.

Finally, in Table 6, we compare our results with the best implementations of Rijndael encryption on VIRTEX-E technology found in literature. For RAM based substitution boxes, McLoone and McCanny had the best unrolled implementation in CHES 2001 [9] while Helion Technologies [10] had the best loop architecture. For LUT-based substitution boxes, we have no knowledge of any unrolled architecture but Satoh and Morioka presented in ASIACRYPT 2001 and in the Third NESSIE workshop the best results for loop implementations [15, 20]. They studied mux-modeled s-boxes as well as composite ones. Finally, concerning older technologies[2], we report in Table 7 an old result of our LUT-based loop architecture and compare it with results of the last AES conference [3–6]. It is obvious that the methodology applied allowed us to significantly improve previously reported performances of Rijndael implemented in FPGAs.

6 Conclusion

When implementing block ciphers, several strategies can produce effective designs. Based on recently published works and observations about Rijndael, we studied different possible implementation tradeoffs. Inherent constrainst of FPGAs were taken into account in order to define an efficient methodology. We defined notions of hardware efficiency and optimal pipeline and our circuits were designed in order to optimize different possible architectures: loop and unrolled. Inside these architectures, we proposed algorithmic optimizations for the substitution box but also efficient combinations between the diffusion layer and the key addition.
Upon comparison, our circuits offer better performance than previously reported

[2] Most of the AES performance evaluation was done on VIRTEX1000 FPGAs.

in literature. Compact and high speed architectures are proposed and implemented on VIRTEX-E technology. Throughput is up to 18.5 Gbits/sec and area requirements can be limited to 542 slices and 10 RAM blocks with an improved ratio throughput/area. Optimized efficiency was obtained by applying heuristic rules in order to deal with place and route constraints.

Type	Nbr of slices	Device	Throughput (Mbits/s)	Throughput/Area ($\frac{Mbits/s}{slices}$)
Gaj et al.	2900	VIRTEX1000	331.5	0.11
Dandalis et al.	5673	VIRTEX1000	353	0.06
Elbirt et al.	9004	VIRTEX1000	1940	0.22
Our design	**2257**	**VIRTEX1000**	**1563**	**0.69**

Table 7. Comparisons with the last AES conference.

References

1. Xilinx: *Virtex 2.5V Field Programmable Gate Arrays Data Sheet*, http://www.xilinx.com.
2. J.Daemen and V.Rijmen, *AES Proposal: Rijndael*, NIST's AES home page, *http : //www.nist.gov/aes*.
3. A.J.Elbirt et Al, *An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists*, The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.
4. K.Gaj and P.Chodowiec, *Comparison of the Hardware Performance of the AES Candidates using Reconfigurable Hardware*, The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.
5. P.Chodowiec et al, *Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple-DES Using SLAAC-1V FPGA Accelerator Board*, in the proceedings of ISC 2001: Information Security Workshop, LNCS 2200, pp.220-234, Springer-Verlag.
6. A.Dandalis et al, *A Comparative Study of Performance of AES Candidates Using FPGAs*, The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.
7. T.Ichikawa et al, *Hardware Evaluation of the AES Finalists*, The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.
8. O.Kwon et al, *Implementation of AES and Triple-DES Cryptography using a PCI-based FPGA Board*, in the proceedings of ITC-CSCC 2002: The International Technical Conference On Circuits/Systems, Computers and Communications.
9. M.McLoone and J.V.McCanny, *High Performance Single Ship FPGA Rijndael Algorithm Implementations*, in the proceedings of CHES 2001: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.
10. Helion Technology, *High Performance AES (Rijndael) Cores for XILINX FPGA*, *http : //www.heliontech.com*.
11. V.Fischer and M.Drutarovsky, *Two Methods of Rijndael Implementation in Reconfigurable Hardware*, in the proceedings of CHES 2001: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.
12. CAST, *AES Encryption Cores*, *http : //www.cast − inc.com*.
13. Amphion Semiconductor, *CS5210-40: High Performance AES Encryption Cores*, 2001. http://www.amphion.com/cs5210.html
14. N.Sklavos, O.Koufopavlou, *Architecutre and VLSI Implementations of the AES-Proposal Rijndael*, in IEEE Transactions on Computers, Volume 51, Number 12, pp1454-1459, December 2002.
15. A.Satoh et al, *Compact Hardware Architecture for 128-bit Block Cipher Camellia*, in the Proceedings of the Third NESSIE Workshop, november 6-7, 2002, Munich, Germany.
16. N.Weaver and J.Wawrzynek *High Performance Compact AES Implementations in Xilinx FPGAs*, http://www.cs.berkeley.edu/ nweaver/Rijndael.
17. Xinmiao Zhang, Parhi K.K., *Implementation approaches for the advanced encryption standard algorithm*, in IEEE Circuits and Systems Magazine, pp 24-46, Fourth Quarter 2002.
18. FX.Standaert, G.Rouvroy, JD.Legat, JJ.Quisquater, *A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES Rijndael*, in the proceedings of FPGA 2003: the Field Programmable Logic Array Conference, February 23-25 2003, Monterey, California.
19. A.Rudra et al, *Efficient Rijndael Encryption Implementation with Composite Field Arithmetic*, in the proceedings of CHES 2001: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.
20. A.Satoh et al, *A Compact Rijndael Hardware Architecture with S-Box Optimization*, Advances in Cryptology - ASIACRYPT 2001, LNCS 2248, pp239-254, Springer-Verlag.
21. J.Wolkerstorfer, E.Oswald, M.Lamberger, *An ASIC Implementation of the AES SBoxes*, in the proceedings of CT-RSA 2002, LNCS 2271, pp67-78, Springer-Verlag.