A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL

François-Xavier Standaert, Gael Rouvroy,
Jean-Jacques Quisquater, Jean-Didier Legat
{standaert,rouvroy,quisquater,legat}@dice.ucl.ac.be

UCL Crypto Group
Laboratoire de Microélectronique
Université Catholique de Louvain
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium

ABSTRACT

Reprogrammable devices such as Field Programmable Gate Arrays (FPGA's) are highly attractive options for hardware implementations of encryption algorithms and this report investigates a methodology to efficiently implement block ciphers in CLB-based FPGA's. Our methodology is applied to the new Advanced Encryption Standard RIJNDAEL and the resulting designs offer better performances than previously published in literature. We propose designs that unroll the 10 AES rounds and pipeline them in order to optimize the frequency and throughput results. In addition, we implemented solutions that allow to change the plaintext and the key on a cycle-by-cycle basis with no dead cycles. Another strong focus is placed on low area circuits and we propose sequential designs with very low area requirements. Finally we demonstrate that RAM-based implementations implies different constraints but our methodology still holds.

Categories and Subject Descriptors

B.7.1 [**Algorithms implemented in Hardware**]; E.3 [**Data encryption**]

General Terms

Algorithms, Security, Design, Performance

Keywords

AES RIJNDAEL, Reconfigurable hardware, FPGA, Cryptography, High encryption rates

FPGA'03, February 23–25, 2003, Monterey, California, USA.

1. INTRODUCTION

In September 1997, the NIST [1] issued a request for possible candidates for a new Advanced Encryption Standard (AES) to replace the Data Encryption Standard (DES). In August 1998, 15 candidates algorithms were selected and a year later, in August 1999, five finalists were announced: MARS, RC6, SERPENT, RIJNDAEL and TWOFISH. On 2 October 2000, the RIJNDAEL algorithm, developed by Joan Daemen and Vincent Rijmen was selected as the winner of the AES development race. In performance comparison studies carried out on all five finalists, RIJNDAEL proved to be one of the fastest and most efficient algorithms. It is also implemented on a wide range of platforms and is extendable to different key and block lengths.

As opposed to custom hardware or software implementations, little work exists in the area of block cipher implementations within existing FPGA's. Results available in the public literature sometimes mention encryption rates comparable with software ones. We believe that these performances can be greatly improved using today's technology as soon as inherent constraints of FPGA's are taken into account.

In this paper, we propose a methodology to efficiently implement block ciphers within commercially available FPGA's, based on similarities between configurable logic blocks available in FPGA's and encryption algorithms. It actually consists in simple digital design rules adapted to FPGA constraints. It is applied to RIJNDAEL and allows to improve previously reported results in terms of hardware cost, throughput or efficiency. We also suggest that different constraints have to be considered if some parts of the algorithm are implemented into the RAM blocks available in present FPGA's. Improved RAM-based implementations of RIJNDAEL are proposed in order to confirm this assumption.

This paper is organized as follows. The description of the hardware, synthesis tools and implementation tools is in sec-

[1] NIST : National Institute of Standards and Technology.

tion 2. Section 3 gives a short mathematical description of RIJNDAEL and we propose an efficient representation of the key scheduling algorithm by means of a key round. Our design methodology is proposed in section 4 and applied to RIJNDAEL in section 5. Comparisons between our implementation results and other published designs are in section 6 and conclusions are in section 7.

2. HARDWARE DESCRIPTION

All our implementations were carried out on XILINX VIRTEX1000BG560-6 and XILINX VIRTEX3200ECG1156-8 FPGA's. We chose these technologies 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). A 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 (LUT's). In addition to operate as a function generator, each LUT can provide a 16×1-bit synchronous RAM. Furthermore, the two LUT's 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 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 FPGA's 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 VIRTEX1000BG560-6 FPGA contains 12288 slices and 32 RAM blocks, which means 24576 LUT's and 24576 flip-flops. A VIRTEX3200ECG1156-8 FPGA contains 32448 slices and 208 RAM blocks, which means 64896 LUT's and 64896 flip-flops. In the next sections, we compare the number of LUT's, registers and slices. We also evaluate the delays and frequencies thanks to our synthesis tool. The synthesis was performed with FPGA Express 3.6.1 (SYNOPSYS) and the implementation with XILINX ISE-4. 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[2]. 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.

Bytesub, the non-linear layer γ: The Bytesub 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 bytesub 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 Shiftrow transformation δ: In Shiftrow, 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 Mixcolumn transformation θ: In Mixcolumn, 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:

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

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

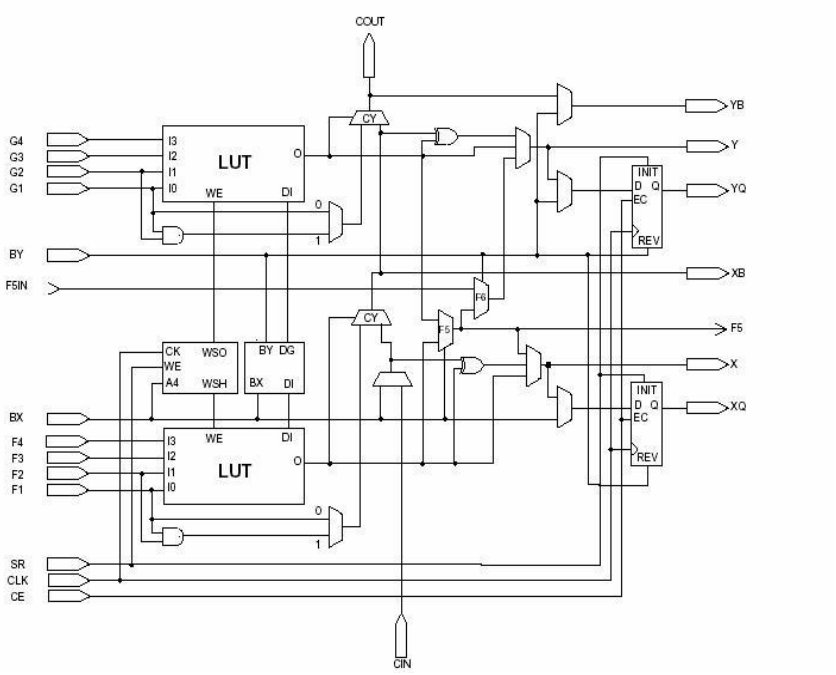

Figure 1: The VIRTEX slice.

Where (b_3, b_2, b_1, b_0) is a four-byte column of the state. An output byte of mixcolumn (for example b_0) can be expressed as[3]:

$$b_0 =' 02' \times a_0 +' 03' \times a_1 +' 01' \times a_2 +' 01' \times a_3 \tag{4}$$

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.

$$\sigma[k](a) = b \Leftrightarrow b_i = a_i \oplus k_i, 0 \leq i \leq 15 \tag{5}$$

[3] $\oplus$ is the bitwise EXOR operation.

The round transformation $\rho[K]$: The round transformation can be written as a composition of the four precedent transformations[4]:

$$\rho[K] = \sigma[K] \circ \theta \circ \delta \circ \gamma \tag{6}$$

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, Subsyte (SB) is a function that takes a 4-byte word in which each byte is the result of applying the RIJNDAEL s-box. The function Rotbyte (RB) 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 a 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:

$$K_{i+1} = \beta(K_i), i = 0, ..., 10 \tag{7}$$

Figure 2 illustrates the key round of Rijndael where registers needed for efficiency purposes are already mentioned.

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:

$$\alpha[K_0, K_1, ..., K_{10}] = \sigma[K_{10}] \circ \delta \circ \gamma \circ (\bigcirc_{r=1}^{9} \rho[K_r]) \circ \sigma[K_0] \tag{8}$$

[4] Read $\sigma[K]\big(\theta(\delta(\gamma))\big)$.

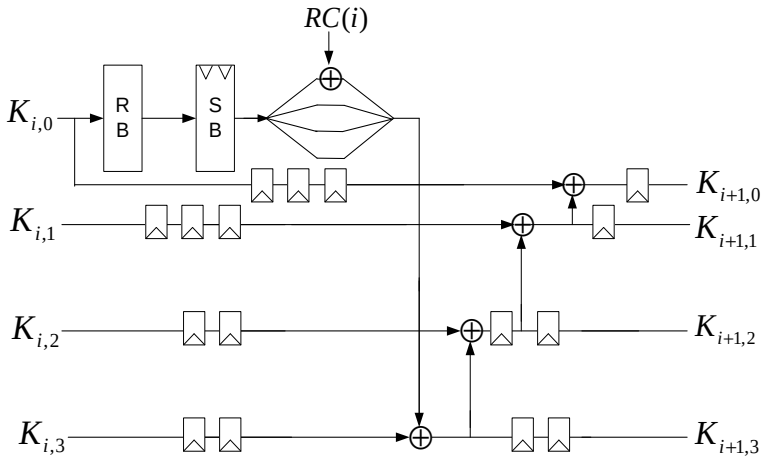

Figure 2: The key round β.

Our implementations are based on this description of AES Rijndael.

4. DESIGN METHODOLOGY:

FPGA's are very efficient devices and they are suitable for high work frequencies. However the CLB structure described in section 2 involves constraints to take 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 logic used, storage used and resulting performances. Typically, the designer tries to limit its critical path inside one CLB slice without consuming slices for register usage only.

Most modern block ciphers are iterated[5] and their round function usually consists of very simple algebraic or logic operations. As a consequence, they are suitable for efficient FPGA implementations by nature and the following methodology is a relevant tool to obtain good circuits. It actually consists in simple digital design rules applied to FPGA's.

1. Implement every basic component of the block cipher. Synthetise them and compute their LUT cost and critical path.

2. Insert registers in the basic components in order to limit the critical path inside one slice. The number of registers needed should never exceed the number of LUT's computed in step 1. An efficient usage of the slice implies to take advantage of additional EXOR gates and multiplexors available in the slice.

3. Combine basic components if they offer possible optimizations. Registers do not need to be placed between components only.

4. Build the round and the key round and adapt their number of pipeline stages if different. Use shift registers if needed because they allow more efficient implementations than repeated registers.

5. Build the complete cipher from these optimal components and implement the resulting design.

[5]The block cipher is defined as the application of a number of key-dependant transformations called round function.

At every step, efficiency can be checked by computing the ratio $Nbr\ of\ LUT's/Nbr\ of\ registers$. It always should be close to one. Let the efficiency of a block cipher be the ratio $Area\ (slices)/Throughput\ (Mbits/s)$. This methodology allows to get very efficient designs after synthesis. However, the implementation (specially the place and route step) of large designs is often difficult and sometimes implies additional constraints that can be overcome by modifying some parts of the design. This explain the frequently large difference between estimated frequency after synthesis and work frequency after implementation.

In the next section, we illustrate this methodology and apply it to the block cipher RIJNDAEL.

5. IMPLEMENTATION:

In order to allow relevant comparisons with existing FPGA implementations of RIJNDAEL, we performed different experiments with different technologies, depending on the use of RAM blocks to implement the substitution box or on the FPGA used: VIRTEX or VIRTEX-E. In the next section, the delay is estimated after synthesis[6], with a VIRTEX1000-BG560-6.

5.1 Components:

Bytesub, the non-linear layer γ: We implemented Bytesub as a large multiplexor, and took advantage of FPGA configurations to implement these ones. The upper part of Figure 6 illustrates the implementation of the Rijndael s-box. We pipelined γ by inserting two registers 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 transform γ where the

[6]FPGA Express (SYNOPSYS).

Component	Nbr of LUT	Nbr of registers	Estimated delay (ns)
γ	$144 \times 16 = 2304$	$42 \times 16 = 672$	5.8

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

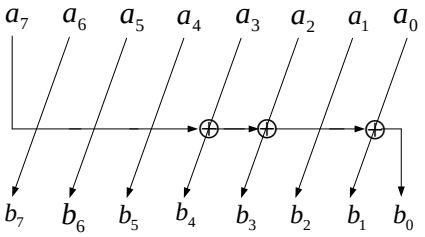

Figure 3: The function X.

s-box is repeated 16 times.

Another possibility is to use the RAM blocks available inside the VIRTEX to implement substitution boxes. The resulting bytesub transform uses 8 RAM blocks and is performed in one clock cycle. We discuss the RAM-based implementations of RIJNDAEL in section 6.

The shiftrow transformation δ: This is just routing information and takes no place in the design.

The Mixcolumn transformation 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:

$$b_0 =' 02' a_0 +' 03' a_1 +' 01' a_2 +' 01' a_3 \quad (9)$$

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 3 illustrates the function X. Note that output bits 0,2,5,6,7 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 4:

$$b_0 = X(a_0) \oplus X(a_1) \oplus a_1 \oplus a_2 \oplus a_3 \quad (10)$$

Interesting combinations between Mixcolumn 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. We will take advantage of this and try to keep our critical path inside one Virtex slice.

Combining Mixcolumn and Addroundkey: The Mixadd transform ϵ. On Figure 4, 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 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 4 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.

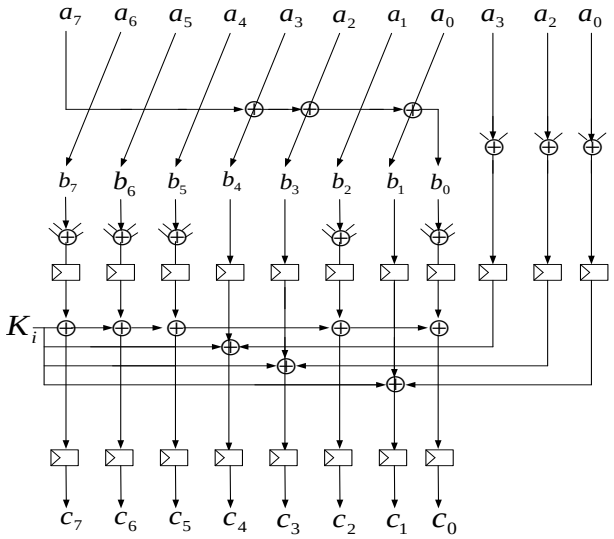

Figure 5: The Mixadd transform ϵ at the bit level.

Figure 5 illustrates the combination of Mixcolumn and Addroundkey at the bit level. Finally, Table 2 summarizes the synthesis results for the Mixadd transformation.

5.2 The round and key round functions:

Based on the above components, we can build the round and key round functions and evaluate their hardware cost. Figure 6 illustrates the round function of Rijndael. Figure 2 illustrates its key round. Both contain 4 registers levels. Note that Subbyte has the same inner structure as Bytesub and therefore the same number of register levels. The synthesis results for the round ρ and key round β functions are given in table 3.

6. COMPARISONS:

6.1 LUT-based implementations:

In [3, 4, 5, 6, 7], different LUT-based FPGA implementations of RIJNDAEL encryption are proposed within a VIRTEX1000. In the next section, we propose designs implemented on the same technology and compare them with existing results.

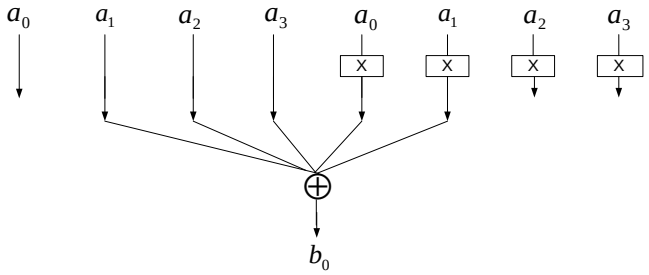

Figure 4: Output byte b_0 of the Mixcolumn transform θ.

Component	Nbr of LUT	Nbr of registers	Estimated delay (ns)
ϵ	304	304	4.8

Table 2: Synthesis of the Mixadd transform ϵ.

Component	Nbr of LUT	Nbr of registers	Estimated delay (ns)
LUT-based round	2608	976	5.8
LUT-based key round	768	488	5.8

Table 3: Synthesis of the round and key round.

Type	Nbr of LUT	Nbr of reg.	Nbr of slices	RAM blocks	Latency (cycles)	Output every (cycles)	Freq. after Synt. (Mhz)	Freq. after Impl. (Mhz)
Pipeline RIJNDAEL	33712	14592	17984	0	42	1	172	/
Sequential RIJNDAEL	3916	2132	2257	0	52	5/52	172	127

Table 4: RIJNDAEL encryption implementations on VIRTEX1000.

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 5: Comparisons with other LUT-based implementations.

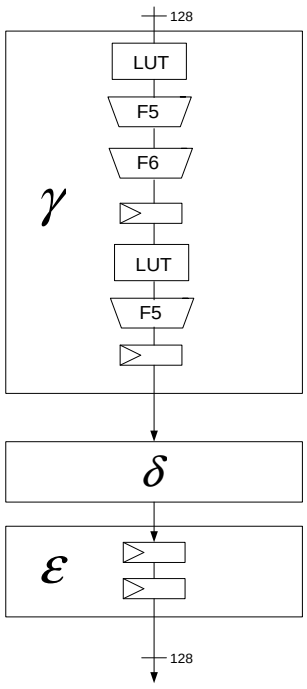

Figure 6: The round function ρ.

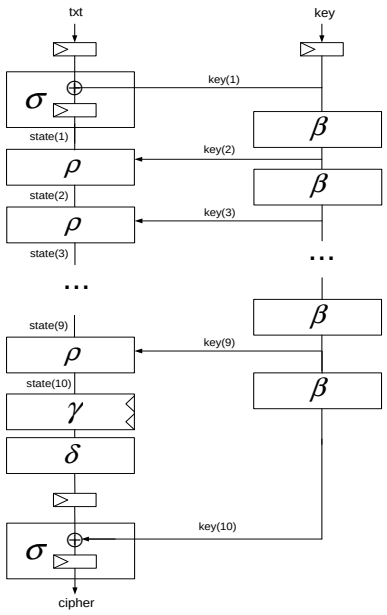

Figure 7: Pipeline AES Rijndael.

Our implementations of AES Rijndael directly results from the component descriptions. We decided to implement a pipeline version that unrolls the 10 RIJNDAEL rounds (illustrated on Figure 7) and a sequential version with only one unrolled round (illustrated on Figure 8 where the grey functions are actually included in the round ρ). We observed the hardware cost in terms of LUT's, registers and slices as well as the frequency results. In this section, the frequency is estimated after synthesis[7] and implementation[8].

On Table 4, we observe very high frequencies after synthesis. However, critical delays mainly occur when trying to place and route these synthesis results. The resulting implemented designs have surprising critical paths including 20% of logic and 80% of routes. We conclude that the real bottleneck of such large ciphers is the difficulty of having an efficient place and route. Actually, constraints come from shift registers and high fanout. Implementation could probably be improved by inserting registers but this additional degree of freedom for the routes would be balanced with additional ressources . We conclude that in case of critical routing delays, designs having logic paths over two (or more) slices can be considered. However, if our design methodology is applied, no slice should be used for register usage only and therefore this should not change the area requirements. Anyway, the resulting designs are very efficient as shown in Table 5 where we list the most efficient implementations within VIRTEX1000 FPGA's. Note that our pipeline design would be even more efficient but cannot fit into a VIRTEX1000.

[7]FPGA Express (SYNOPSYS).
[8]Xilinx ISE4.

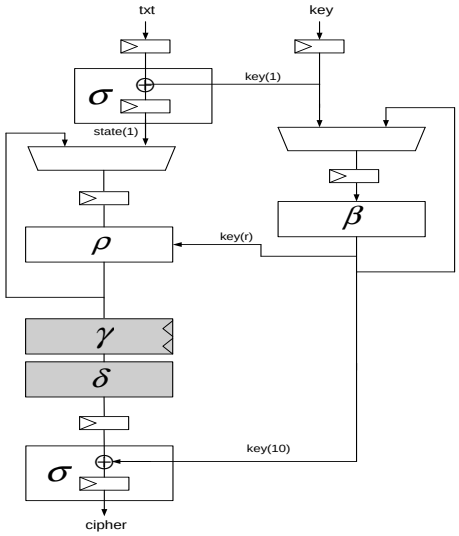

Figure 8: Sequential AES Rijndael.

6.2 RAM-based implementations:

In [9, 10, 11], RAM-based FPGA implementations of RIJNDAEL encryption are proposed within a VIRTEX-E FPGA. In the next section, we propose designs implemented on the same technology and compare them with existing results.

As we previously mentioned, it is possible to implement the substitution boxes of RIJNDAEL in the RAM blocks available in present VIRTEX-E. Although this could result in very efficient implementations, it also involves different design constraints. When RAM blocks are used, a strong bottleneck arises when trying to place and route large designs. This make the optimal pipelining of CLB's completely ineffective. In order to illustrate this assumption, we decided to implement different RAM-based solutions, where only the number of pipeline stages differs:

1. A 42-cycle pipeline or 52-cycle sequential version with a 4-cycle round: a register is added after the RAM-based substitution box.
2. A 32-cycle pipeline or 42-cycle sequential version with a 3-cycle round: the register after the RAM-based substitution box is removed.
3. A 21-cycle pipeline or 31-cycle sequential version with a 2-cycle round: the register inside the mixadd transform ϵ is also removed.
4. A 21-cycle sequential version with a 2-cycle round: the register after the multiplexor is removed.

Table 6 summarizes the implementation results of our different RAM-based RIJNDAEL encryption modules. It clearly illustrates that the high pipelining used in CLB-based implementations do not lead to optimal circuits. Actually, the most efficient solutions correspond to situations where logic paths cover two slices. The key difference with CLB-dominated designs is the lower area requirements of the design that causes slices to be used for register usage only if high pipelining is performed. However, our methodology still holds: step 2 mentioned that the number of registers of components should never exceed the number of LUT's needed. Optimal designs have well balanced logic and register requirements. Our 21-cycle solutions have this interesting property: their ratio $Nbr\ of\ LUT's/Nbr\ of\ registers$ is close to one.

Note that in case of sequential circuits, as high pipelining is no more wanted, it is also possible to modify the round structure so that initial and final key additions can be managed by the round function. This leads to very low area circuits as pictured on Figure 9.

The resulting designs improves the previously reported RAM-based implementations as shown in Table 7.

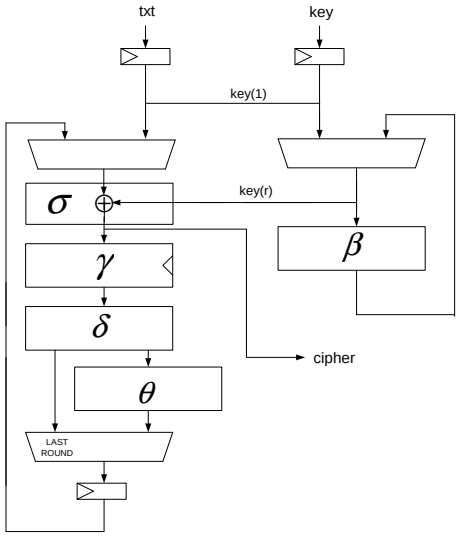

Figure 9: Modified sequential AES Rijndael.

7. CONCLUSIONS

We propose a methodology to implement block ciphers in reconfigurable hardware and applied it to RIJNDAEL. Inherent constraints of FPGA's are taken into account in order to get very efficient circuits. We investigated different possible implementations: CLB-dominated or RAM-based, pipeline or sequential. These implementations involve different constraints but the design methodology holds as long as we keep the ratio $Nbr\ of\ LUT's/Nbr\ of\ registers$ close to one. Efficiency is measured as the ratio $Area\ (slices)/Throughput\ (Mbits/s)$ and we obtain very efficient designs. A strong focus is placed on high throughput and low area and we implemented solutions for both criteria.

Upon comparison, our designs offer better results than previously reported in literature. Compact and high speed architectures are proposed and implemented on VIRTEX and VIRTEX-E technologies. Throughput is up to 14 Gbits/s and area requirements can be limited to 405 slices and 10 RAM blocks. Bottlenecks arises in the routing of our synthesis results. This could certainly be improved and this last point could deserve further analysis.

8. REFERENCES

[1] Xilinx: *Virtex 2.5V Field Programmable Gate Arrays Data Sheet*, http://www.xilinx.com.

[2] J.Daemen and V.Rijmen, *The Block Cipher 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)

Type	Nbr of LUT	Nbr of reg.	Nbr of slices	RAM blocks	Latency (cycles)	Output every (cycles)	Freq. after Synt. (Mhz)	Freq. after Impl. (Mhz)
Pipeline RIJNDAEL	4912	7792	5144	100	42	1	285	112
Pipeline RIJNDAEL	4272	6832	4032	100	32	1	232	92
Pipeline RIJNDAEL	3516	3840	2784	100	21	1	208	92
Sequential RIJNDAEL	1036	1452	866	10	52	5/52	285	147
Sequential RIJNDAEL	965	1372	739	10	42	4/42	232	135
Sequential RIJNDAEL	877	932	550	10	31	3/31	208	117
Sequential RIJNDAEL	877	668	542	10	21	2/21	208	119
Modified sequential	709	413	405	10	20	2/20	192	87

Table 6: RIJNDAEL encryption implementations on VIRTEX3200E.

Type	Nbr of LUT's	Nbr of slices	RAM blocks	Device	Throughput (Mbits/s)	Throughput/Area ($\frac{Mbits/s}{slices,LUT's}$)
McLoone et al.	/	2222	100	VIRTEX810E	6956	3.1
Our design	**3516**	**2784**	**100**	**VIRTEX3200E**	**11776**	**4.2**
Helion tech.	899	/	10	VIRTEX3200E	1187	1.32
Our design	**877**	**542**	**10**	**VIRTEX3200E**	**1450**	**1.65**

Table 7: Comparisons with other RAM-based implementations.

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 FPGA's*, 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] M.McLoone and J.V.McCanny, *Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm*, in the proceedings of FPL 2002: The Field Programmable Logic Conference, Lecture Notes in Computer Science, LNCS 2147, p.152ff.

[11] Helion Technology, *High Performance AES (Rijndael) Cores for XILINX FPGA*, *http : //www.heliontech.com.*

[12] 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.

[13] 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.

[14] A.Satoh et al, *A Compact RIJNDAEL Hardware Architecture with S-Box Optimization*, Advances in Cryptology - ASIACRYPT 2001, LNCS 2248, pp239-254, Springer-Verlag.

[15] CAST, *AES Encryption Cores*, *http : //www.cast − inc.com.*