Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications

Gaël Rouvroy, François-Xavier Standaert,
Jean-Jacques Quisquater and Jean-Didier Legat
UCL Crypto Group
Laboratoire de Microélectronique
Université catholique de Louvain
Place du Levant, 3, B-1348 Louvain-la-Neuve, Belgium
rouvroy,standaert,quisquater,legat@dice.ucl.ac.be

Abstract

Hardware implementations of the Advanced Encryption Standard (AES) Rijndael algorithm have recently been the object of an intensive evaluation. Several papers describe efficient architectures for ASICs[1] and FPGAs[2]. In this context, the highest effort was devoted to high throughput (up to 20 Gbps) encryption-only designs, fewer works studied low area encryption-only architectures and only a few papers have investigated low area encryption/decryption structures. However, in practice, only a few applications need throughput up to 20 Gbps while flexible and low cost encryption/decryption solutions are needed to protect sensible data, especially for embedded hardware applications. This paper proposes an efficient solution to combine Rijndael encryption and decryption in one FPGA design, with a strong focus on low area constraints. The proposed design fits into the smallest Xilinx FPGAs[3], deals with data streams of 208 Mbps, uses 163 slices and 3 RAM blocks and improves by 68% *the best-known similar designs in terms of ratio* $Throughput/Area$*. We also propose implementations in other FPGA Families (Xilinx Virtex-II) and comparisons with similar DES, triple-DES and AES implementations.*

Keywords: Cryptography, AES, DES, FPGA, compact encryption/decryption implementation.

[1] ASIC: Application Specific Integrated Circuit.
[2] FPGA: Field Programmable Gate Array.
[3] Xilinx Spartan-3 XC3S50.

1. Introduction

In October 2000, NIST (National Institute of Standards and Technology) selected Rijndael [3] as the new Advanced Encryption Standard (AES), in order to replace the old Data Encryption Standard (DES). The selection process included performance evaluation on both software and hardware platforms and many hardware architectures were proposed. However, most of these architectures simply transcribe the algorithm into hardware designs, without relevant optimizations and tradeoffs. Moreover, the throughput and area constraints considered are often unrealistic as shown by the recently published results.

Speed efficient designs are not always relevant solutions. Many applications require smaller throughput (wireless communication, digital cinema, pay TV, ...). Sequential designs based on a 1-round loop may be judicious and attractive in terms of hardware cost for many embedded applications. Several such implementations have been published in the literature. For DES and triple-DES designs, the most efficient solution [5] encrypts/decrypts in 18 cycles with a fresh key. For AES, the best design based on 1-round loop [6, 7] produces a data rate of 1450 Mbps (Virtex-E) using 542 slices and 10 RAM blocks, but it does not support the decryption mode. Another efficient circuit [8] proposes a compact architecture that combines encryption and decryption. It executes one round in four cycles and produces a throughput of 166 Mbps (Spartan-II) using 222 slices and 3 RAM blocks.

The design proposed in this paper is also based on a quarter of round loop implementation and improves by 68% (in term of ratio $Throughput/Area$) the design detailed in [8]. We investigate a good combination of encryption/decryption and place a strong focus on a very low

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)
0-7695-2108-8/04 $ 20.00 © 2004 IEEE

area constraints. The resulting design fits in the smallest Xilinx devices (e.g. the Spartan-3 XC3S50 and Virtex-II XC2V40), achieves a data stream of 208 Mbps (using 163 slices, 3 RAM blocks) and 358 Mbps (using 146 slices, 3 RAM blocks), respectively in Spartan-3 and Virtex-II devices. It attempts to create a bridge between throughput and cost requirements for embedded applications.

The paper is organized as follows: the mathematical description of Rijndael is referred in section 2; section 3 describes our sequential AES encryptor/decryptor; finally, section 4 concludes this paper.

2. The AES algorithm

Due to space constraints, we do not have the opportunity to detail the AES algorithm. We recommend the reader to refer to [3].

3. Our sequential AES implementation

Some designs propose an implementation based on one complete round, and iteratively loop data through this round until the entire encryption or decryption is achieved. Only one $State^i$ is processed in one cycle. These designs are suited for feedback and non-feedback modes of operation.

As mentioned in [8], the AES round offers various opportunities of parallelism. The round is composed of 16 S-boxes and four 32-bit $MixColumn$ operations, working on independent data. Only $ShiftRow$ needs to deal with the entire 128-bit $State$.

Based on this observation, we propose an implementation using four S-boxes and one $MixColumn$ in order to compact the design. This decreases the area by a factor of four but increases the time of one round to four cycles. In practice, only the time-space tradeoff is modified. A similar approach was proposed in [8].

3.1. Implementation of ShiftRow and InvShiftRow operations

In our design, the way to access the $State^i$, for the first quarter of the round, is described in Figure 1. We read d^i_{15},d^i_{10},d^i_5,d^i_0 in parallel from the input memory, and execute $SubByte$, $MixColumn$ and $AddRoundKey$. Then we write results d^{i+1}_{15},d^{i+1}_{14},d^{i+1}_{13}, d^{i+1}_{12} to a different output memory. The second, third, and fourth quarters of the round are managed in a similar manner, depending on $ShiftRow$.

The best FPGA solution to implement such simultaneous read and write memory accesses is proposed in [8]. The solution is based on a shift register design. As described above, all calculations from the $AddRoundKey$ are written into adjacent locations of the output memory in consecutive cycles. We store first d^{i+1}_{15},d^{i+1}_{14},d^{i+1}_{13}, d^{i+1}_{12} in parallel, then d^{i+1}_{11},d^{i+1}_{10},d^{i+1}_9, d^{i+1}_8 in parallel, and so on. Therefore we can store the consecutive round results into shift registers (one shift register per row of the $State$, four shift registers for four rows). Xilinx FPGAs propose a very space efficient solution to achieve a 16-bit shift register with a dynamic variable access. Four slices can implement an 8-bit wide, 16-bit long shift register. The four dynamic variable accesses are used to read the input memory content at correct positions into the rows. Four 8-bit wide shift registers are needed, which corresponds to 16 slices. The $InvShiftRow$ operation can be done using the same shift registers modifying the way to access the input memory.

$$\begin{bmatrix} \mathbf{d^i_{15}} & d^i_{11} & d^i_7 & d^i_3 \\ d^i_{14} & \mathbf{d^i_{10}} & d^i_6 & d^i_2 \\ d^i_{13} & d^i_9 & \mathbf{d^i_5} & d^i_1 \\ d^i_{12} & d^i_8 & d^i_4 & \mathbf{d^i_0} \end{bmatrix}$$

$$\Downarrow$$

SB + MC + AK

$$\Downarrow$$

$$\begin{bmatrix} \mathbf{d^{i+1}_{15}} & d^{i+1}_{11} & d^{i+1}_7 & d^{i+1}_3 \\ \mathbf{d^{i+1}_{14}} & d^{i+1}_{10} & d^{i+1}_6 & d^{i+1}_2 \\ \mathbf{d^{i+1}_{13}} & d^{i+1}_9 & d^{i+1}_5 & d^{i+1}_1 \\ \mathbf{d^{i+1}_{12}} & d^{i+1}_8 & d^{i+1}_4 & d^{i+1}_0 \end{bmatrix}$$

Figure 1. Memory accesses involved in the first calculation step of the round i.

SB, MC and AK respectively mean the $SubByte$, $MixColumn$ and $AddRoundKey$ operations. Bellow, we also define SR, ISR, ISB, IMC and IAK as respectively $ShiftRow$, $InverseShiftRow$, $InverseSubByte$, $InverseMixColumn$ and $InverseAddRoundKey$ transformations.

3.2. Implementation of SubByte/MixColumn and InvSubbyte/InvMixColumn

Compared to the paper [8], we propose a more efficient combination of $SubByte$ and $MixColumn$ operations, i.e. we use less resources than separated block implementations. Our solution takes advantage of specific features of the new Xilinx devices and perfectly fits into the Spartan-3 or Virtex-II technologies[4].

The Spartan-3 and Virtex-II FPGAs have both dedicated 18-Kbit dual-port RAM blocks[5], that can be used to store tables for the combination of $SubByte$ and $MixColumn$.

As also mentioned in [3], the consecutive $SubByte$ and $MixColumn$ operations on the first quarter of the round can be expressed as $e^i_{15..12}$:

[4] It is not the case with Spartan-II.

[5] The Spartan-II has dedicated 4-Kbit dual-port RAM blocks.

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)
0-7695-2108-8/04 $ 20.00 © 2004 IEEE

$$\begin{bmatrix} e^i_{15} \\ e^i_{14} \\ e^i_{13} \\ e^i_{12} \end{bmatrix} = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \otimes \begin{bmatrix} SB(d^i_{15}) \\ SB(d^i_{10}) \\ SB(d^i_5) \\ SB(d^i_0) \end{bmatrix}$$

which is also equivalent to:

$$\begin{bmatrix} '02' \\ '01' \\ '01' \\ '03' \end{bmatrix} \otimes \left[SB(d^i_{15})\right] \oplus \begin{bmatrix} '03' \\ '02' \\ '01' \\ '01' \end{bmatrix} \otimes \left[SB(d^i_{10})\right] \oplus \begin{bmatrix} '01' \\ '03' \\ '02' \\ '01' \end{bmatrix} \otimes \left[SB(d^i_5)\right] \oplus \begin{bmatrix} '01' \\ '01' \\ '03' \\ '02' \end{bmatrix} \otimes \left[SB(d^i_0)\right]$$

If we define four tables (T_0 to T_3) with 256 4-byte data as:

$$T_0(a) = \begin{bmatrix} '02' \bullet SB(a) \\ SB(a) \\ SB(a) \\ '03' \bullet SB(a) \end{bmatrix}$$

$$\mathrm{T}_1(a) = \begin{bmatrix} '03' \bullet SB(a) \\ '02' \bullet SB(a) \\ SB(a) \\ SB(a) \end{bmatrix}$$

$$\mathrm{T}_2(a) = \begin{bmatrix} SB(a) \\ '03' \bullet SB(a) \\ '02' \bullet SB(a) \\ SB(a) \end{bmatrix}$$

$$\mathrm{T}_3(a) = \begin{bmatrix} SB(a) \\ SB(a) \\ '03' \bullet SB(a) \\ '02' \bullet SB(a) \end{bmatrix}$$

The combination of $SubByte$ followed by $MixColumn$ can be expressed as:

$$\begin{bmatrix} e^i_{15} \\ e^i_{14} \\ e^i_{13} \\ e^i_{12} \end{bmatrix} = T_0(d^i_{15}) \oplus T_1(d^i_{10}) \oplus T_2(d^i_5) \oplus T_3(d^i_0)$$

The size of one T_i table is 8 Kbits for encryption. The corresponding similar table for decryption also takes 8 Kbits (IT_0 to IT_3). It is therefore possible to achieve the complete $SubByte/MixColumn$ and $InvSubByte/InvMicolumn$ operations using two dual-port 18-Kbit RAM blocks.

The proposed solution significantly reduces the resources used in [8].

3.3. Encryption/Decryption design choices

One of the inconveniences of AES comes from the fact that the $AddRoundKey$ is executed after $MixColumn$ in the case of encryption and before $InvMixColumn$ in the case of decryption. Such encryption/decryption implementation will therefore require additional switching logic to select appropriate data paths, which can also affects the time performance. The paper [8] mentions this problem but chooses to design like that anyway.

AES decryption algorithm nevertheless allows $InvMixColumn$ and $AddRoundKey$ to be reordered if we perform an additional $InvMixColumn$ operation on most of the $RoundKeys$ (except the first and the last $RoundKeys$). More details about such scheduling of operations can be found in [3, 4]. At first sight, $InvMixColumn$ could seem to require much more area than the switching logic. This is especially true if the $InvMixColumn$ of the round is narrowly combined with the $InvSubByte$ in RAM blocks. Nevertheless, the subsection 3.4 proposes a solution using very few additional resources but some extra cycles to generate all inverse $Roundkeys$ ($InvRoundKeys$). Figure 2 summarizes our design choices concerning the data path round.

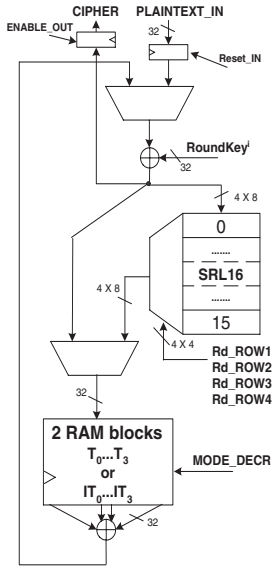

Figure 2. Our AES data path round.

3.4. Implementation of the key schedule

The implementation of our AES key schedule is based on precomputing $RoundKeys$ and $InvRoundKeys$ in advance and storing them in one RAM block. The difficult computation of the $InvRoundkeys$ on-the-fly[6] completely justifies this approach.

Our implementation of the key schedule is shown in Figure 3. First, it computes 32-bits of all the

[6]It is a real weak aspect of AES algorithm.

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)
0-7695-2108-8/04 $ 20.00 © 2004 IEEE

$RoundKey^i$ per clock cycle. The results are stored in one dual-port block RAM, thanks to the first port. This step takes 44 clock cycles. In the same time, we also store $SB(RoundKeys)$ data in the same RAM, but using the other port. It corresponds to the first step of the calculation process of $InvRoundKeys$. As mentioned in the subsection 3.3, $InvRoundKey^i$ equals to $IMC(RoundKey^{10-i})$, except for the first $InvRoundKey^0$ and last $InvRoundKey^{10}$, which equal to respectively $RoundKey^{10}$ and $RoundKey^0$.

If we need decryption processes, a second step has to be applied to $SB(RoundKeys)$. Therefore, we start to calculate $ISB(SB(RoundKey^{10}))$ and store it as $InvRoundKey^0$. Then, we evaluate the result $IMC(ISB(SB(RoundKey^{10-i})))$ which equals to $IMC(RoundKey^{10-i})$ and we store it as $InvRoundKey^{1..9}$. $InvRoundKey^{10}$ is generated as $InvRoundKey^0$. This optional decryption process takes 48 cycles to generate the complete $InvRoundKeys$.

Due to $InvRoundKey^0$ and $InvRoundKey^{10}$, tables (T_0 to T_3) need to be changed. $InvSubByte$ has to replace the duplicated $SubByte$. We define new 16-Kbit tables (CT_0 to CT_3) combined with (IT_0 to IT_3). CT_0 is illustrated below as an example:

$$CT_0(a) = \left[\begin{array}{c|c} '02' \bullet SB(a) & '0E' \bullet SB(a) \\ SB(a) & '09' \bullet SB(a) \\ ISB(a) & '0D' \bullet SB(a) \\ '03' \bullet SB(a) & '0B' \bullet SB(a) \end{array}\right]$$

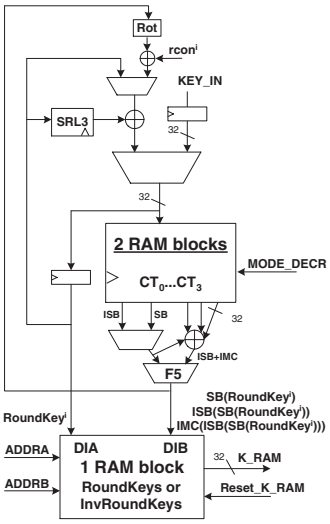

Figure 3. Our AES key schedule.

3.5. Implementation and results of our complete AES

Our final AES design combines the data path part and the key schedule part. Since the key schedule is done with precomputation, this part does not work simultaneously with the encryption/decryption process. It is therefore possible to share resources between both circuits. Both parts of the circuit were thought to perfectly fuse together without additional slices[7] and tri-state buffers. This allows reaching higher frequency than in [8]. The global design is shown in Figure 4. We fused the key and plaintext inputs to one register. The input and output registers are packed into IOBs to improve the resources used and the global frequency of the design.

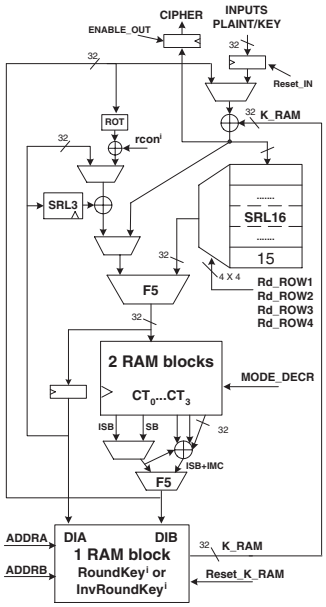

Figure 4. Our complete AES design.

The synthesis of our complete design was done using Synplify Pro 7.2 from Synplicity. The placing and routing were done using Xilinx ISE 6.1i package. The final results are given in Table 2 for Spartan-3 and Virtex-II.

As a comparison, we also set up a table with the previous AES [8], DES and 3-DES [5] results. Table 1 shows the results of these compact encryption/decryption circuits. Like others papers, we also define a ratio $Throughput/Area$ to facilitate comparisons. We finally achieve an implementation of AES which is 68% better in terms of $Throughput/Area$ assuming that Spartan-II and Spartan-3 are equivalent.

In comparison with the most efficient compact 3-DES

[7] Only new F5 multiplexers are required.

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)
0-7695-2108-8/04 $ 20.00 © 2004 IEEE

Algo.	Gaj's AES	**Our AES**	**Our AES**	**Our DES**	**Our 3-DES**
Device	XC2S30-6	XC3S50-4	XC2V40-6	XC2V40-6	XC2V40-6
Slices	222	163	146	189	227
Through. (Mbps)	166	208	358	974	326
RAM blocks	3	3	3	0	0
Throughput/Area (Mbps/slices)	0.75	**1.26**	**2.45**	**5.15**	**1.44**

Table 1. Comparisons with other sequential block cipher implementations.

Device	XC3S50-4	XC2V40-6
LUTs used	293	288
Registers used	126	113
Slices used	163	146
RAM blocks	3	3
Latency (cycles)	46	46
Out. every (cycles)	$1/44$	$1/44$
Frequency	71.5 MHz	123 MHz

Table 2. Final results of our complete sequential AES.

circuits in XC2V40-6, we can conclude that AES is more effective if we do not care about the use of three internal RAM blocks. However, 3-DES remains interesting for applications that need to regularly change the key for encryption or decryption. Indeed, our AES design takes 92 cycles, in the worst case, to calculate a new complete $InvRoundKeys$.

4. Conclusion

In this paper, we propose solutions for a very compact and effective FPGA implementation of the AES. We combine narrowly the non-linear S-boxes and the linear diffusion layer thanks to specific features of recent Xilinx devices. We also propose a low-cost solution to deal with the subkeys computed during the decryption step. In addition, we merge efficiently the key schedule and the data path parts.

The resulting implementations fits in a very inexpensive Xilinx Spartan-3 XC3S50 FPGA, for which the cost starts below $10 per unit. This implementation can encrypt and decrypt a throughput up to 208 Mbps, using 163 slices. The design also fits in Xilinx Virtex-II XC2V40 and produces data streams up to 358 Mbps, using 146 slices. In comparison with 3-DES, AES is more efficient if we do not care about the use of three internal FPGA RAM blocks.

The throughput, low-cost and flexibility of our solution make it perfectly practical for cryptographic embedded applications.

References

[1] Xilinx. The Spartan-3 and Virtex-II field programmable gate arrays data sheets,available from `http://www.xilinx.com`.

[2] National Bureau of Standards. *FIPS PUB 46*, The Data Encryption Standard. U.S. Departement of Commerce, Jan 1977.

[3] J. Daemen and V. Rijmen. The Block Cipher RIJNDAEL, NIST's AES home page, available from `http://www.nist.gov/aes`.

[4] P.Baretto, V.Rijmen, *The KHAZAD Legacy-Level Block Cipher*, Submission to NESSIE project, available from http://www.cosic.esat.kuleuven.ac.be/nessie/

[5] G. Rouvroy, FX. Standaert, JJ. Quisquater, JD. Legat. Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and TripleDES. In the proceedings of FPL 2003, Lecture Notes in Computer Science, vol 2778, pp. 181-193, Springer-Verlag.

[6] FX. Standaert, G. Rouvoy, JJ. Quisquater, JD. Legat. A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES Rijndael. In the proceedings of FPGA 2003, pp. 216-224, ACM.

[7] FX. Standaert, G. Rouvoy, JJ. Quisquater, JD. Legat. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In the proceedings of CHES 2003, Lecture Notes in Computer Science, vol 2779, pp. 334-350, Springer-Verlag.

[8] K. Gaj and P. Chodowiec. Very Compact FPGA Implementation of the AES Algorithm. In the proceedings of CHES 2003, Lecture Notes in Computer Science, vol 2779, pp. 319-333, Springer-Verlag.

Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)
0-7695-2108-8/04 $ 20.00 © 2004 IEEE