# 4.2 Gbits/sec Single-Chip FPGA Implementation of the AES Algorithm.

F. Rodríguez-Henríquez, N. A. Saqib and A. Díaz-Pérez

*This letter presents a high performance encryptor/decryptor core of the Advanced Encryption Standard (AES). The proposed architecture is implemented on a single-chip -FPGA using a fully pipelined approach. The results obtained show that our design offers up to 25.06% less area and yields up to 27.23% higher throughput than the fastest AES FPGA implementations reported to date.*

*Introduction*: Rijndael block cipher algorithm was chosen by NIST as the new Advanced Encryption Standard (AES) [1, 2]. The AES encryption and decryption processes definitions are not identical [2]. This characteristic poses a challenge for efficient AES encryptor/decryptor core hardware implementations, as the implementation of a separate architecture for encryption and decryption processes would imply costly area requirements. On the other hand, while efficient hardware implementation was one of the main evaluation criteria for selecting the AES winner, few FPGA encryption designs have been reported to date [3-7], and not all of them implement encryptor/decryptor cores [7]. In this letter a fully pipelined AES encryptor/decryptor core is presented. First a description of the AES algorithm and proposed optimizations are discussed, and then performance results are presented and compared with previous reported designs.

*The AES algorithm*: Figure 1 shows the encryption architecture of the AES cipher algorithm for encrypting one 128-bit block of data using a 128-bit user key. The algorithm treats the input 128-bit block as a group of 16 bytes organized in a 4×4 matrix called *State* matrix. The algorithm consists of an initial transformation, followed by a main loop where nine iterations called *rounds* are executed. Each round is composed of a sequence of four transformations: Byte Substitution (BS), ShiftRows (SR), MixColumns (MC) and AddRoundKey (ARK). For each round of the main loop, a round key is derived from the original key through a process called *Key Scheduling*. Finally, a last round consisting of three transformations, BS, SR and ARK, is executed. The AES decryption algorithm operates similarly by applying the inverse of all the transformations described above in reverse order. In the rest of this section we will briefly describe the four AES rounds transformations BS, SR, MC and ARK.

*ByteSubstitution (BS):* Each input byte of the State matrix is independently replaced by another byte from a look-up table called S-box. The AES S-box is a 256-entry table composed of two transformations: First each input byte is replaced with its multiplicative inverse in $GF(2^8)$ with the element {00} being mapped onto itself; followed by an affine transformation over $GF(2)$ [1, 2]. For decryption, inverse S-box is obtained by applying inverse affine transformation followed by multiplicative inversion in $GF(2^8)$ [2].

*ShiftRows (SR):* is a cyclic shift operation where each row is rotated cyclically to the left using 0,1,2 and 3-byte offset for encryption while for decryption, rotation is applied to the right.

*MixColums (MC):* In this transformation each column of the *State* matrix is multiplied by a constant fixed matrix as follows,

$$\begin{bmatrix} c'_{0,i} \\ c'_{1,i} \\ c'_{2,i} \\ c'_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,i} \\ c_{1,i} \\ c_{2,i} \\ c_{3,i} \end{bmatrix} \qquad (1)$$

Where the two column vectors above represent the i-esime column of the State matrix and the i-esime column of the transformed MC State matrix, for $i$ = 0, 1, 2, 3, respectively. Similarly, for the decryption process, we compute Inverse

MixColumns, by multiplying each column of the State matrix by a constant fixed matrix as shown below

$$\begin{bmatrix} c'_{0,i} \\ c'_{1,i} \\ c'_{2,i} \\ c'_{3,i} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} c_{0,i} \\ c_{1,i} \\ c_{2,i} \\ c_{3,i} \end{bmatrix} \tag{2}$$

*AddRoundKey (ARK):* The output of MC is XOR-ed with the corresponding round sub-key derived from the user key. The ARK step is essentially the same for encryption and decryption processes.

*Design:* A pipelined approach for the AES algorithm implementation is shown in Figure 2. Keys are generated and fed to each round. After 10 cycles, encrypted 128-bit data blocks appear at each clock cycle. As it has been already mentioned, the encryption and decryption processes of AES are not symmetric. Therefore, the inverse of each step must be implemented separately for decryption. This approach is usually not practical as it implies the allocation of large amounts of memory. In order to alleviate the area requirements for an AES encryptor/decryptor core, we propose the following design optimizations for FPGA implementations:

*S-box and Inverse S-box implementation::* Both of them, the S-box and the inverse S-box may be computed by implementing the affine (AF) and Inverse affine (IAF) transformations together with a look-up table for Multiplicative Inverse (MI). In this way, the combination MI + AF provides S-box for encryption, while IAF + MI computes the Inverse S-box needed for decryption. An additional multiplexer block is used to switch the data path for encryption/decryption. This approach is advantageous since the same look-up table can be reused for encryption and decryption.

*MC and IMC Implementation:* MC for encryption can be computed efficiently by using only 3 steps [1]: a sum step, a doubling step and a final sum step. Let the elements of State matrix's column one be *a*[0], *a*[1], *a*[2] and *a*[3] and let *k*[0], *k*[1], *k*[2] and *k*[3] represent the first, second, third and fourth bytes of a 4-byte key block, respectively. Then the transformed MC column *a*'[0], *a*'[1], *a*'[2] and a'[3] can be efficiently obtained by computing

$$\begin{array}{lll} v = a[1]\oplus a[2]\oplus a[3] & xt_o = xtime(a[0]) & a'[0] = k[0]\oplus v \oplus xt_0 \oplus xt_1; \\ v = a[0]\oplus a[2]\oplus a[3] & xt_1 = xtime(a[1]) & a'[1] = k[1]\oplus v \oplus xt_1 \oplus xt_2; \\ v = a[0]\oplus a[1]\oplus a[3] & xt_2 = xtime(a[2]) & a'[2] = k[2]\oplus v \oplus xt_2 \oplus xt_3; \\ v = a[0]\oplus a[1]\oplus a[2] & xt_3 = xtime(a[3]) & a'[3] = k[3]\oplus v \oplus xt_3 \oplus xt_0; \end{array} \quad (3)$$

Where xtime(*v*) represents the finite field multiplication of $02 \times v$, and 02 stands for the constant polynomial *x* in *GF*($2^8$). Notice also that in the above equation we have embedded for efficiency reasons, the ARK transformation into the MC step. The same strategy utilized above for MC would yield seven steps to compute IMC (four sum steps and three doubling steps). The complexity difference is due to the fact that coefficients in equation (2) have a higher Hamming weight than the ones in equation (1). We overcome this drawback by observing that it should exist a 4×4 byte matrix *D*(*x*) in *GF*($2^8$) such that,

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} D(x) \quad (4)$$

Using the fact that both constant matrixes in equation (4) are the inverse of each other in the field *F* = *GF*($2^8$) generated by the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ [2]. It can be easily proved that equation (4) has a unique solution in the finite field *F* given by: $d_{0,0} = 5; d_{1,0} = 0; d_{2,0} = 4; d_{3,0} = 0.$ Where $d_{i,0}$, i = 0, 1, 2, 3 represent the four coefficients of the first column of *D*(*x*). Hence, equation (4) can be rewritten as,

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \quad (5)$$

The above equation suggests an efficient way to compute IMC by reusing the MC transformation to obtain the IMC constant matrix. This is useful since the second matrix in the right side of equation (5) has considerably less Hamming weight than the original constant matrix for IMC. A small modification (ModM) is therefore introduced before MC+ARK steps for decryption. Figure 3 shows the proposed architecture for AES encryption/decryption processes using the following data path,

*Encryption*: MI + AF + SR + MC + ARK

*Decryption*:: ISR + IAF + MI + ModM + MC + ARK

*Performance Results*: The design shown in figure 3, was implemented on a XCV2600E Xilinx Virtex-E FPGA device. The implementation occupies 80 BRAMs (43 %), 386 I/O Blocks (48 %) and 5677 slices (22.3 %). Table 1 details the total area required for each block shown in figure 3. The design uses a system clock of 34.2 MHz and the data is processed at a rate of 4121 Mbits/sec.

As is shown in Table 2, the results obtained compare quite favorably with existing FPGA implementations The area requirements have been reduced up to 25.06% while the expected throughput has been increased more than 27.23% compared to the fastest FPGA design reported to date ([3]).

*Conclusions*: In this letter, a highly optimized fully pipelined AES encryptor/decryptor core architecture has been presented. The total number of slices required by the design are 5677 while achieved throughput is 4121

Mbits/sec. Future work includes possible improvements of the design's performance figures by using different optimization techniques.

## References

1. Daemen, J. and Rijmen, V.: 'The Design of Rijndael'. AES-The Advanced Encryption Standard, (Springer-Verlag Berlin Heidelberg, New York, 2002).

2. Trappe, W. and Washington, L.C.: 'Introduction to Cryptography with Coding Theory' (Prentice-Hall, Upper Saddle River, 2002)

3. McLoone, M. and McCanny, J.V.: 'High Performance FPGA Rijndael Algorithm Implementations` (Springer-Verlag, 2001), CHES2001, LNCS 2162, pp. 65-76.

4. Ichikawa, T., Kasuya, Matsui, M.: 'Hardware Evaluation of the AES Finalists'. The Third Advanced Encryption Standard (AES3) Candidate Conference, New York, USA, 13-14 April, 2000.

5. Weeks, B., Bean, M., Rozylowicz, T. and Ficke, C.: 'Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms'. The Third Advanced Encryption Standard (AES3) Candidate Conference, New York, USA, 13-14 April, 2000.

6. Elbirt, J., Yip, W., Chetwynd, B. and Paar, C.: 'A FPGA implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists'. The Third Advanced Encryption Standard (AES3) Candidate Conference, New York, USA, 13-14 April, 2000.

7. Lutz, K. *et al.*: '2 Gbit/s Hardware Realizations of RIJNDAEL and SERPENT'. A comparative analysis (Springer-Verlag, 2002), CHES2002, LNCS 2523, pp. 144-158.

**Authors' Affiliations:**
F. Rodríguez-Henríquez, N. A. Saqib and A. Díaz-Pérez (Computer Science Section, CINVESTAV- IPN, Av. IPN No. 2508, 07360 México D.F.)
E-mail: francisco@cs.cinvestav.mx

**Table Captions:**

Table 1: Area requirements for AES encryption/decryption.
Table 2. AES algorithm FPGA implementation.

Table 1

| **Block** | **CLB Slices** | **BRAMs** |
|---|---|---|
| KeyBlock | 1278 | |
| BS | ------- | 80 |
| SR | ------- | ----- |
| Affine Transformation | 80 x 10 = 800 | |
| Inverse Affine Transformation | 64 x 10 = 640 | |
| MixColumn + ARK (combined) | 152 x 9 = 1368 | |
| Modification (for decryption) | 120 x 9 =1080 | |
| ARK (1st + last round) | 128 | |
| Misc. (Timing + I/O registers) | 383 | |
| **Total** | **5677** | **80** |

Table 2

| | **Device** | **CLB Slices** | **Throughput (Mbits/sec)** |
|---|---|---|---|
| Ichikawa et al[4] | VLSI | ------- | 1950 |
| Weeks et al [5] | VLSI | ------- | 5163 |
| Lutz et a [7] | VLSI | ------- | 2260 |
| Elbirt et al [6] | XCV1000 | 9004 | 1940 |
| McLoone et al [3] | XCV3200E | 7576 | 3239 |
| This design | XCV2000E | 5677 | 4121 |

**Figure Captions:**

Fig.1. Basic algorithm flow.

Fig. 2. Pipeline approach for 128 bit  Rijndael Encryption.

Fig. 3. Proposed AES implementation for a pipeline design.

Figure 1

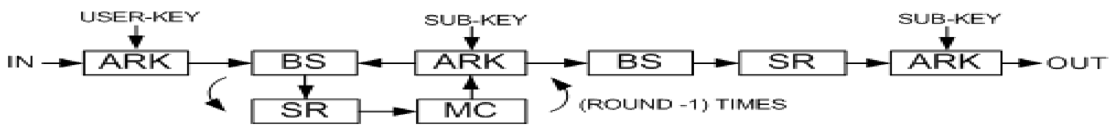

USER-KEY
SUB-KEY
SUB-KEY
IN
ARK
BS
ARK
BS
SR
ARK
OUT
SR
MC
(ROUND -1) TIMES

Figure 2

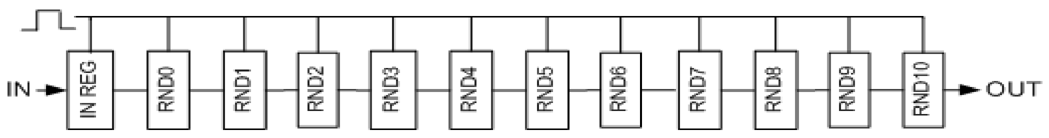
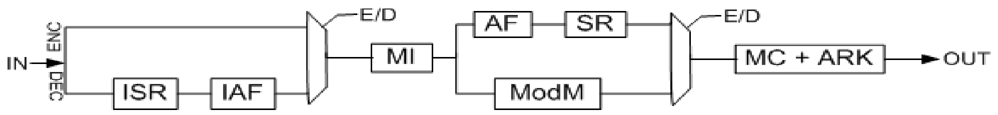

IN
IN REG
RND0
RND1
RND2
RND3
RND4
RND5
RND6
RND7
RND8
RND9
RND10
OUT

Figure 3


IN
ENC
DEC
ISR
IAF
E/D
MI
AF
SR
ModM
E/D
MC + ARK
OUT