Implementation of the AES Algorithm Using a Reconfigurable Functional Unit

G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, S. Pontarelli, M. Re , A. Salsano
University of Rome ”Tor Vergata”
Via del Politecnico 1, 00133, Rome, ITALY
Email: {cardarilli, di.nunzio, fazzolari, pontarelli, re, salsano}@ing.uniroma2.it

***Abstract*—Nowadays programmable devices (microprocessors and DSPs) are based on complex architectures optimized for obtaining maximum speed performances that degrades when the implemented application is mostly based on operations on single bit or subset of bits. This kind of data processing and bit manipulation operations can be accelerated by using a Reconfigurable Functional Unit (RFU). In this paper the benefits of using the ADAPTO RFU (Adder-Based Dynamic Architecture for Processing Tailored Operators) [1] [2] to speed up the Advanced Encryption Standard algorithm (AES) is investigated. The paper shows how the ADAPTO architecture is useful for the acceleration the AES algorithm due the efficient implementation of the most complex operations of the algorithm. A comparison in terms of number of assembly instructions is given.**

I. Introduction

The spreading diffusion of digital techniques for data and signal processing is pushing toward microprocessor and DSP architectures of increasing efficiency.

An important limitation for such new structures is their efficiency in computing operations based on a reduced data parallelism. Examples of these operations are bit permutations, where input bits are rearranged individually or on the basis of short subwords [4], and polynomial multiplication in Galois Field (GF), where only some input bits are multiplied (XORed) by the polynomial coefficients [10].

Of course, these nonstandard operations can be performed by conventional processors, but their implementation requires several standard instructions. Despite its large use, this approach is not efficient and reduces the processor speed performance.

Several solutions have been proposed in the literature to overcome this drawback, either in software [4] or hardware. Among the hardware solutions the most interesting ones, in terms of flexibility and performance, are based on Reconfigurable Functional Units (RFUs). Proposed RFUs are similar to small FPGAs (array of LUTs and pass-transistors for the programmable interconnect) and are connected in parallel to the ALU, sharing the Register File (RF) and working as an hardware Instruction Set (IS) expansion [6].

RFUs are very different from conventional coprocessors in expanding the core instruction set, but coprocessors are not integrated in the datapath unit requiring the use of the system bus to exchange data.

In this scenario, great efforts are devoted to develop efficient processors for embedded systems applications. In this case, apart from evaluations based on computational performance, very critical constraints are represented by power consumption and cost factors that are strongly related to the complexity of the architecture and consequently to the silicon area.

To face these constraints in [1], [2] the authors proposed a new architecture named ADAPTO in the which LUTs [5] used for the implementation of general purpose logic have been replaced by Full-Adders (FAs). The resulting architecture is less expensive with respect to those proposed in the literature, but this favorable characteristic is counterbalanced by a reduced flexibility.

In order to evaluate the trade-off between complexity and flexibility, the authors defined a set of experiments based on typical embedded systems applications. Those applications have been executed by using ADAPTO and a general purpose microprocessor (by using an architecture emulator) in order to verify the speed-up factor [3].

In this paper we shows that ADAPTO can also be used to accelerate the Rijndael AES algorithm [10], that is based on operations on $GF(2^n)$.

The paper is organized as follows. In Section II ADAPTO is briefly described, while Section III illustrates the the AES algorithm. In section IV the ADAPTO implementation of the AES algorithm is presented with the evaluation of the obtained speed-up. Finally, in Section V the conclusions are drawn.

II. The ADAPTO architecture

The ADAPTO RFU architecture is based on three alternated stripes of Logic Blocks (LBs) and interconnect with a parallelism of 32 bits (both for inputs and outputs). The architecture has been conceived to be connected to the main processor Register File (RF). LBs are based on FAs that perform both logical and arithmetical operations meanwhile the interconnect is based on pass transistors (as shown in Fig. 1). Multicontext is implemented by context memories (LBs and interconnect programming).

FAs can be configured to execute one bit addition, NOT and PASS, 2 input AND, 2 input OR, 2 and 3 input XOR, and 3 Majority.

The structure of the interconnect is shown in Fig. 2, and is based on a multicontext approach (for an high reconfiguration speed). Each LB output can be linked with any inputs of the LBs of the bottom row. In addition to the 32 inputs coming from the upper LBs, two additional lines directly connected 0

978-1-61284-943-0

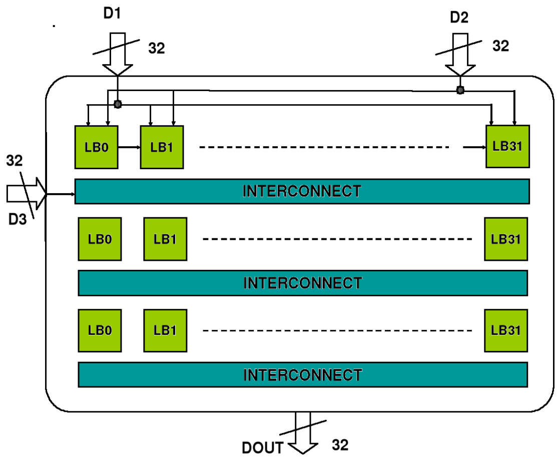

Fig. 1. ADAPTO Reconfigurable Array Architecture

and 1 have been added to the interconnect in order to easily implement shift operations with 1 or 0 insertion and operations on constant values. Multicontext configuration bits are stored in local memories. Carry chain uses a direct connection (linking adjacent LBs) to speed-up the carry propagation in multibit adders. A more detailed description of the ADAPTO architecture can be found in [1] and [5].

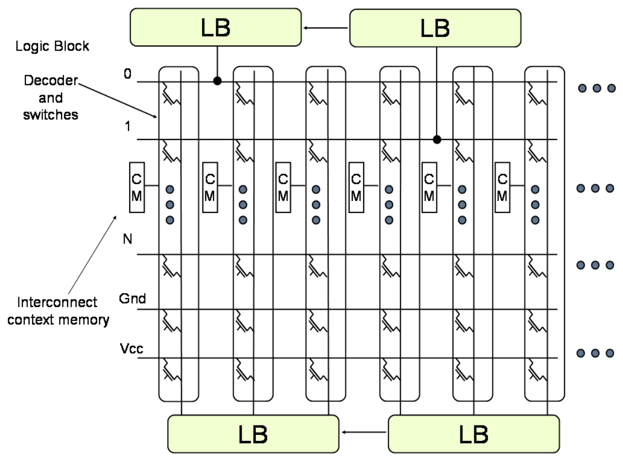

Fig. 2. Reconfigurable Interconnect Network.

III. The AES algorithm

In the AES algorithm (that is based on byte operations) the encryption of a data block is composed by: a XOR step, several round transformations (in the following simply rounds), and an final round (different by the previous ones). In the case of 128 bits blocks, the data are arranged as 16 bytes (the state) and are organized in a matrix as follows

$$\mathbf{I} = \begin{pmatrix} A & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{pmatrix} \quad (1)$$

The encryption of a 128 bits block requires 9 rounds, each one composed by four processing steps

1) ***SubBytes:*** a non-linear substitution step where each byte is replaced by addressing a Look Up Table (LUT).
2) ***ShiftRows:*** a transposition step where each row of the state is cyclically shifted a for a number of steps.
3) ***MixColumns:*** a mixing operation operating on the columns of the state, combining the four bytes in each column
4) ***AddRoundKey:*** each byte of the state is combined with the round key.

In the final round, the Mixcolumn transformation is not performed. For the decryption, the inverse transformations ***InvSubBytes***, ***InvShiftRowsInvMixColumns*** and a slogtly dofferent ***AddRoundKey*** are used.

In a software implementation of the AES encryption/decryption, *SubBytes* and *InvSubBytes* operations are efficiently implemented by using 256 bytes LUTs, while *ShiftRows* and *InvShiftRows* are byte reorderings and correspond to a simple software implementation. Moreover, *ShiftRows* and *InvShiftRows* can be merged with *SubBytes* and *InvSubBytes* and therefore their impact on the performance of the AES algorithm is negligible. Also *AddRoundKey* a two inputs 32-bit XOR operation can be efficiently implemented in standard software.

The operations that can take advantage by using an RFU are *MixColumns/InvMixColumns* due to their implementation complexity that on a **ARM926EJ-S RISC** requires about 50 assembly instructions. The *MixColumn* transformation is a based on matrix multiplication in $GF(2^8)$ where each column of $\mathbf{I}$ is multiplied by each row of

$$\mathbf{M} = \begin{pmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{pmatrix} \quad (2)$$

obtaining 16 results. Each element of the matrix (hexadecimal notation) represents a polynomial of degree 7 with coefficients corresponding to the its binary representation ($e.g.$ 0x0A =0b00001010 = $x^3 + x$). We remark that constant multiplications are performed on $GF(2^8)$ by using $x^8 + x^4 + x^3 + x + 1$ as generator polynomial.

For the *InvMixcolumn* transformation each column of $\mathbf{I}$ is multiplied by each column of the inverse of $\mathbf{M}$

$$\mathbf{M^{-1}} = \begin{pmatrix} 0x0E & 0x0B & 0x0D & 0x09 \\ 0x09 & 0x0E & 0x0B & 0x0D \\ 0x0D & 0x09 & 0x0E & 0x0B \\ 0x0B & 0x0D & 0x09 & 0x0E \end{pmatrix} \quad (3)$$

Multiplications involved in *InvMixcolumn* are more complex than multiplications involved in *Mixcolumn* and represents the more expensive task in the AES algorithm. In the next section we show how these operations are implemented by using ADAPTO.

IV. ADAPTO implementation

A. Data allocation strategy

The allocation of data has been performed considering the RF of a 32 bit microprocessor. The implementation of *Mixcolumn* and *InvMixcolumn* operations in ADAPTO requires to choose an appropriate organization to store the 16 bytes of **I** in the RF [1]. In our approach, **I** has been memorized by column (Table I). In this way ADAPTO can load an entire column at each clock cycle.

	Stored bytes			
Register/part	[31..24]	[23..16]	[15..8]	[7..0]
R1	A	B	C	D
R2	E	F	G	H
R3	I	J	K	L
R4	M	N	O	P

TABLE I
RF ALLOCATION OF AES MATRIX

B. $GF(2^8)$ constant multiplication in ADAPTO

In this subsection, $GF(2^8)$ constant multiplication is illustrated (GF multiplication corresponds to a conventional polynomial multiplication followed by a division by the polynomial generator). In order to illustrate the use of ADAPTO, we consider the following example. Take into account the multiplication of a generic 8-bit polynomial $P = p_7 \cdot x^7 + p_6 \cdot x^6 + p_5 \cdot x^5 + p_4 \cdot x^4 + p_3 \cdot x^3 + p_2 \cdot x^2 + p_1 \cdot x^1 + p_0$ by the constant polynomial $(x + 1)$, corresponding to the hexadecimal number 0x03. The operation 0x03 $\cdot P$ in the GF gives 0x03$\cdot P = (p_7 \cdot x^7 + p_6 \cdot x^6 + p_5 \cdot x^5 + p_4 \cdot x^4 + p_3 \cdot x^3 + p_2 \cdot x^2 + p_1 \cdot x + p_0) + (p_6 \cdot x^7 + p_5 \cdot x^6 + p_4 \cdot x^5 + p_3 \cdot x^4 + p_2 \cdot x^3 + p_1 \cdot x^2 + p_0 \cdot x) + p_7 \cdot (x^4 + x^3 + x + 1)$. Shortly we can write

$$0x03 \cdot P = P + (P << 1) + p_7 \cdot (x^4 + x^3 + x + 1)$$

Figure 3 shows the ADAPTO implementation of the constant multiplication by 0x03. The shift operations, as the left shift $A << 1$ required in the above multiplication, are performed directly by the ADAPTO interconnection network. Also $a_7 \cdot (x^4 + x^3 + x + 1)$ can be implemented by the interconnection network. In fact, let us call Z=$z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0$ the byte representing the result of $a_7 \cdot (x^4 + x^3 + x + 1)$. We have $z_7 = z_6 = z_5 = z_2 = 0$ and $z_4 = z_3 = z_1 = z_0 = a_7$. Therefore the interconnect can compute Z by imposing some LB inputs to zero and connecting a_7 to the remaining LBs. The stripe following the interconnection is configured as a three input XOR, performing the required three additions on $GF(2^8)$.

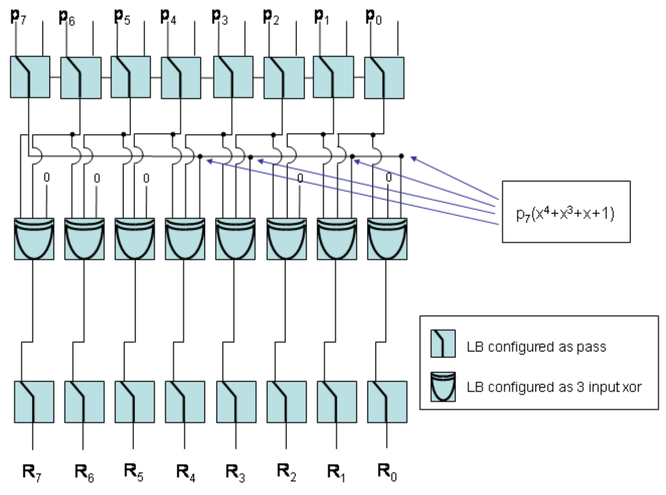

Fig. 3. Implementation of 0x03$\cdot B$ in ADAPTO

Any constant multiplications that can be translated in a bit rearrangement and a sum of three terms can be performed with the above method. In our work any constant multiplication is computed using the set of basic multiplications shown in the Table II.

C	Implementation $C * \cdot P$
$0x02$	$(P << 1) + p_7 \cdot (x^4 + x^3 + x + 1)$.
$0x03$	$P + (P << 1) + p_7 \cdot (x^4 + x^3 + x + 1)$
$0x04$	$(P << 2) + p_7 \cdot (x^5 + x^4 + x^2 + x) +$ $+ p_6 \cdot (x^4 + x^3 + x + 1)$

TABLE II
BASIC CONSTANT MULTIPLICATIONS

C. ADAPTO implementation of *Mixcolumn*

Using the above results, in this subsection we describe the implementation on ADAPTO of the multiplication of a row of **M** by a column of **I**. We suppose that each column of **I** is stored in the RF, according to above discussed data allocation strategy. D1, D2, D3 are the ADAPTO input operands coming from the RF, and R is the final result that will be returned to the RF. Moreover α, β, and γ are the partial results present at the output of the three LB stripes of ADAPTO.
As an example, we consider the multiplication of the first row of **M** by the first column of **I** .
Algorithm 1 describes the implementation of this multiplication. The first stripe of ADAPTO is configured as a pass-thru and connects directly A, B, C, D (output α) to the first level of interconnect . The constant multiplications $0x02 \cdot A$, and $0x03 \cdot B$, with the sum $C + D$ are computed by the interconnect and the second LB stripe (output β). It must be noticed that this mixed operations require 24 LBs configured as three inputs XOR, while the eight rightmost LBs are unused. In the algorithm we represent these unused output as don't care '-'. Starting β, the last LB stripe, configured as three input XOR, provides the final mod. 8 sum γ.

Algorithm 1 Multiplication 1st row of **M** by the 1st data column

```
Input: D1(A,B, C, D); D2=(-.-.-.-); D3=(-.-.-.-)
Output: $R[7:0] = 0x02 \cdot A + 0x03 \cdot B + C + D$.
  $\alpha \leftarrow (A,\ B,\ C,\ D)$
  $\beta \leftarrow (0x02 \cdot A,\ 0x03 \cdot B,\ C + D, -)$
  $\gamma \leftarrow (-,\ -,\ -,\ 0x02 \cdot A + 0x03 \cdot B + C + D)$
  return $R[7:0] \leftarrow \gamma[7:0]$
```

The product of the same column by a different row requires the reconfiguration of ADAPTO. Therefore the entire MixColumn operation requires four different ADAPTO contexts (of the 16 available in the current architecture). However, the computation of the product of different columns by the same row is computed by using the same context.

By using **Algorithm 1** each row by column multiplication requires 1 ADAPTO instruction. Consequently the whole *MixColumn* operation is computed in 16 assembly instructions (corresponding to 16 clock cycles, since one clock cycle is required for each ADAPTO context [1], [2]). This implementation has been compared with a *Mixcolumns* software implementation present in the benchmark suite described in

[11]. This function, compiled on a **ARM926EJ-S RISC**, architecture requires about 200 assembly instructions. Thus the speed-up obtained with ADAPTO is about 12.5x

D. ADAPTO implementation of ***InvMixcolumn***

The InvMixColumn operation $\mathbf{I} \times \mathbf{M^{-1}}$ is more complex due to the structure of the entries of the matrix $\mathbf{M^{-1}}$ (here $\mathbf{I}$ is different from that used in Mixcolumn). To simplify the computation, the constant coefficient of multiplications are expressed in terms of the elementary constants of Table II. Table III is shows the decomposition of the multiplications by complex constants.

C	Decomposition of $C \cdot P$
$0x08$	$0x02 \cdot (0x04 \cdot P)$
$0x09$	$0x08 \cdot P + 0x01 \cdot P$
$0x0B$	$0x08 \cdot P + 0x03 \cdot P$
$0x0C$	$0x03 \cdot (0x04 \cdot P)$
$0x0D$	$0x0C \cdot P + 0x01 \cdot P$
$0x0E$	$0x0C \cdot P + 0x02 \cdot P$

TABLE III
DECOMPOSITION OF COMPLEX CONSTANTS

The decomposition of Table III can require more contexts for the computation of a constant multiplication. For example, the multiplication by $0x0E$ is performed in two phases (corresponding to 2 ADAPTO contexts). For the multiplication of first row of $\mathbf{M^{-1}}$ by the first column of data matrix $\mathbf{I}$ we decompose the computation it in two terms

$$R = 0x0E \cdot A + 0x0B \cdot B + 0x0D \cdot C + 0x09 \cdot D =$$
$$= (0x0C \cdot A + 0x08 \cdot B + 0x0C \cdot C + 0x08 \cdot D) +$$
$$(0x02 \cdot A + 0x03 \cdot B + 0x01 \cdot C + 0x01 \cdot D)$$

The result R is evaluated in two phases. In the first phase, the first partial results are computed by ADAPTO as constant multiplications of A, B, C, D by $0x04$, (corresponding to α) followed by four multiplications by $0x03$ or $0x02$ (γ computation). These operations are implemented in the first ADAPTO context. In the second phase the input D1 contains A, B, C, D, and the inputs D2 and D3 store the results of the previous phase. We use the ADAPTO inputs D1 and D2 to compute $0x0C \cdot C + C$ and $0x08 \cdot D + D$. Instead D3 is used to input the previous results to the input of the second stripe of LB. So we compute the three terms $0x0C \cdot A + 0x02 \cdot A$ $0x03 \cdot B$ and $0x08 \cdot B + 0x0D \cdot C + 0x09 \cdot D$. The third strip of the ADAPTO sums (XOR) these three terms. The two phases are shown in **Algorithm 2**.

The two constants of **Algorithm 2** are valid until the row of $\mathbf{M^{-1}}$ is unchanged. When we go to another row the constants change and two other contexts must be used. Consequently, the computation of the whole *InvMixColumn* in principle requires 8 contexts. Some simplification can be carried out observing the properties of the entries of $\mathbf{M^{-1}}$. For example, PHASE 1 can be shared between the first and third rows, and between the second and the fourth rows. In fact the first and the third rows have the common term $(0x0C \cdot A + 0x08 \cdot B + 0x0C \cdot C + 0x08 \cdot D)$, while the second and the fourth rows have $(0x08 \cdot A + 0x0C \cdot B + 0x08 \cdot C + 0x0C \cdot D)$. This property allows to reduce the number of contexts and number of time the contexts must be reconfigured Therefore the computation of

Algorithm 2 multiplication of a row of the matrix $\mathbf{M^{-1}}$ by a column

```
Input: D1(A,B, C, D); D2=(-,-,-,-); D3=(-,-,-,-)
Output: $R[7:0] = 0x0E \cdot A + 0x0B \cdot B + 0x0D \cdot C + 0x09 \cdot D$.
  PHASE 1 (context 1)
  $\alpha \leftarrow (A,\ B,\ C,\ D)$
  $\beta \leftarrow (0x04 \cdot A,\ 0x04 \cdot B,\ 0x04 \cdot C,\ 0x04 \cdot D)$
  $\gamma \leftarrow (0x0C \cdot A,\ 0x08 \cdot B,\ 0x0C \cdot C,\ 0x08 \cdot D)$
  PHASE 2 (context 2)
  $D1 \leftarrow (A,\ B,\ C,\ D);\ D2 \leftarrow \gamma;\ D3 \leftarrow \gamma;$
  $\alpha \leftarrow (A,\ B,\ 0x0D \cdot C,\ 0x09 \cdot D)$
  $\beta \leftarrow (0x0E \cdot A,\ 0x03 \cdot B,\ 0x08 \cdot B + 0x0D \cdot C + 0x09 \cdot D, -)$
  $\gamma \leftarrow (-, -, -,\ 0x0E \cdot A + 0x0B \cdot B + 0x0D \cdot C + 0x09 \cdot D)$
```

the complete output matrix requires 16 runs of PHASE 2 and 8 runs PHASE 1, for a total of 24 ADAPTO reconfigurations corresponding to 24 assembly instructions. On the other hand, the software implementation of *InvMixColumn* requires again 200 instructions. So, a speed-up of about 8.3x is obtained.

V. CONCLUSIONS

This paper describes how ADAPTO can be used in a low cost microprocessors or DSP architecture to accelerate the execution of the AES algorithm. The combined use of slices of FAs instead of LUTs and a suitable structure for the interconnect allows realizing a flexible architecture using a limited silicon area with respect to other solutions presented in the literature. The use of ADAPTO allows an high speed execution of the most time consuming operations of the AES algorithm. In particular, we show that *MixColumns* and *InvMixColumns* can be performed by ADAPTO in 16 and 24 clock cycles respectively, exploiting the 32-bits parallelism for processing multiple bytes in parallel, obtaining a speed-up in the range 8.3-12.5x with respect to the software implementation on an ARM926EJ-S RISC architecture.

REFERENCES

[1] G. C. Cardarilli, L. Di Nunzio, M. Re, "High Performance Reconfigurable blocks for real-time reconfigurable unit (ADAPTO)", Proc. ReCoSoc 2008, Barcelona, July 9-11, 2008

[2] G. C. Cardarilli, L. Di Nunzio, M. Re, "A full-adder based reconfigurable architecture for fine grain applications: ADAPTO", Proc. IEEE Int. Conf. ICECS 2008, Malta, 31August- 3 September 2008

[3] G. C. Cardarilli, L. Di Nunzio, M. Re "Speed-up of RISC processor computation using ADAPTO" ,Proc. IEEE Int. Symp. on Circuits and Systems, ISCAS 2009, Taipei, Taiwan, 24-27 May 2009

[4] Bengu Li, Rajiv Gupta, "Bit Section Instruction Set Extension of ARM for Embedded Applications", Proc. Int. Conf. CASES 2002, Greenoble, France, October 8-11, 2002

[5] G. C. Cardarilli, L. Di Nunzio, M. Re, "Arithmetic/Logic Blocks for Fine-Grained Reconfigurable Units", Proc. IEEE Int. Symposium on Circuits and Systems, ISCAS 2009, Taipei, Taiwan, 24-27 May 2009

[6] M. D. Razdan, R. Smith, "A high-performance microarchitecture with hardware programmable functional units", Proc. of MICRO-27, Nov. 1994

[7] A.J. Menezes, and P.C. Van Oorschot, S.A.Vanstone, "Handbook of applied cryptography", 1997, CRC press

[8] W.Peterson, and E.Weldon, "Error-correcting codes", 1972, MIT Press

[9] E.Di Claudio, F.Piazza, G.Orlandi, "Fast combinatorial RNS processors for DSP applications", IEEE Trans. on Computers, vol. 44, no. 5, 1995

[10] J.n Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002.

[11] M.. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown "MiBench: A free, commercially representative embedded benchmark suite", Proc. 4th Ann. IEEE Intl Work. Workload Characterization, 2001