An Efficient Architecture for the AES Mix Columns Operation

Hua Li Zac Friggstad
huali@cs.uleth.ca zac.friggstad@uleth.ca
Department of Mathematics and Computer Science
University of Lethbridge
Canada T1K 3M4

Abstract ***In this paper, a compact architecture for the AES mix columns operation and its inverse is presented. The hardware implementation is compared with previous work done in this area. We show that our design has a lower gate count than other designs that implement both the forward and the inverse mix columns operation.***

Keywords: AES, cryptography, Galois field, mix columns

I. Introduction

Since the debut of the Advanced Encryption Standard (AES) [1], it has been thoroughly studied by hardware designers with the goal of reducing the area and delay of the hardware implementation of this cryptosystem. This paper presents an implementation of the AES mix column operation. This design proves to be low in gate count as well as outputs the desired result of the operation (forward or inverse) on the same wire array, depending on an input signal that selects between the two. This will eliminate the need for a multiplexor outside of the module to select between the results of the forward and the inverse operation.

This paper is organized in the following way. Section 2 presents the mathematical background of $GF(2^8)$ and the application to this mix columns design. Section 3 presents the hardware implementation. Section 4 compares this design with previous architectures and suggests appropriate uses for the presented design. Section 5 then concludes this paper.

II. Mathematical Background

The results of the mix columns operation are calculated using $GF(2^8)$ operations. Each element of $GF(2^8)$ is a polynomial of degree 7 with coefficients in $GF(2)$ (or equivalently $\mathbb{Z}_2$). Thus, the coefficients of each term of the polynomial can take the value 0 or 1. Given that there are 8 terms in an element of $GF(2^8)$, an element can be represented by bitstring of length 8, where each bit represents a coefficient. We will use the least significant bit to represent the constant of the polynomial, and going from right to left, represent the coefficient of x^i by the bit b_i where b_i is i bits to the left of the least significant bit. For example, the bitstring 10101011 represents $x^7+x^5+x^3+x+1$. For our convenience, a term x^i is found in the expression if the corresponding coefficient is 1. The term is omitted from the expression if the coefficient is 0. Addition of two elements in $GF(2^8)$ is simply accomplished using eight XOR gates to add corresponding bits.

Multiplication of two elements in $GF(2^8)$ requires a bit more work. The multiplication of two elements of $\mathbb{Z}_2$ is simulated with an AND gate. Multiplication in $GF(2^8)$ can then be accomplished by first multiplying each term of the second polynomial with all of the terms of the first polynomial. Each of these products should be added together. If the degree of the new polynomial is greater than 7, then it must be reduced modulo some irreducible polynomial. In the case of AES, the irreducible polynomial is $x^8+x^4+x^3+x+1$. For example:

$$\begin{array}{rr} & x^4+1 \\ \times & x^5+x^4 \\ \hline & x^8+x^4 \\ + & x^9+x^5 \\ \hline x^9 & +x^8+x^5+x^4 \end{array} \tag{1}$$

Since the degree of the result is greater than 7, then we must reduce it modulo $x^8+x^4+x^3+x+1$. This can be accomplished by multiplying the polynomial by x^{i-8}, where i is the degree of the polynomial that is to be reduced. Then, adding the multiplied irreducible polynomial to polynomial to be reduced. Taking this result, we continue this process until the degree of the polynomial we are reducing is not greater than 7. In our example, we perform:

$$\begin{array}{rr} & x^9+x^8+x^5+x^4 \\ + & x^9+x^5+x^4+x^2+x \\ \hline & x^8+x^2+x \\ + & x^8+x^4+x^3+x+1 \\ \hline & x^4+x^3+x^2+1 \end{array} \tag{2}$$

The mix column operation involves multiplying a 4×4 matrix of $GF(2^8)$ values by a column vector of $GF(2^8)$ values. Given a 32-bit input word $w=w_3w_2w_1w_0$ where each w_i is 8-bits, the mix column operation is given as:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} w_3 \\ w_2 \\ w_1 \\ w_0 \end{bmatrix} = \begin{bmatrix} w'_3 \\ w'_2 \\ w'_1 \\ w'_0 \end{bmatrix} \tag{3}$$

The inverse of this operation is:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} w'_3 \\ w'_2 \\ w'_1 \\ w'_0 \end{bmatrix} = \begin{bmatrix} w_3 \\ w_2 \\ w_1 \\ w_0 \end{bmatrix} \quad (4)$$

Where each element of the 4×4 matrices is a hexadecimal representations of the coefficients of an element in $GF(2^8)$. For example, $0E$ represents $x^3 + x^2 + x$.

Since multiplication is distributive over addition in $GF(2^8)$, we can rewrite the multiplication of two elements of $GF(2^8)$ as a linear combination of products of the first element and a single-termed polynomial in $GF(2^8)$. The sum of all of the single-termed polynomials must equal the second element in the original product. For example, we can express $x \cdot 0E$ as $(x \cdot 08) + (x \cdot 04) + (x \cdot 02)$, for any $x \in GF(2^8)$, because $08 + 04 + 02 = 0E$.

Because of the distributivity of $GF(2^8)$ and of matrices, we can rewrite the inverse of the mix columns operation as:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} w'_3 \\ w'_2 \\ w'_1 \\ w'_0 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} w'_3 \\ w'_2 \\ w'_1 \\ w'_0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0C & 08 & 0C & 08 \\ 08 & 0C & 08 & 0C \\ 0C & 08 & 0C & 08 \\ 08 & 0C & 08 & 0C \end{bmatrix} \cdot \begin{bmatrix} w'_3 \\ w'_2 \\ w'_1 \\ w'_0 \end{bmatrix} \quad (5)$$

Given these facts we note two things. First, for each of the four input bytes, we only need to calculate the products of the byte with 01, 02, 04 and 08. All elements of the matrices can be formed by summing these products. Also, since the inverse matrix can be expressed as the sum of the original matrix and some $\{0C, 08\}$ element matrix, where $0C = 08 + 04$, we see that 04 and 08 are never used in the forward operation, only the inverse.

For some $a \in GF(2^8)$ that is expressed by the bitstring $a_7a_6a_5a_4a_3a_2a_1a_0$, we can efficiently compute $a \times 2$ (or $a \times x$). The resulting value will be $a_6a_5a_4(a_3 + a_7)(a_2 + a_7)a_1(a_0 + a_7)a_7$, where $+$ is addition in $\mathbb{Z}_2$. This can be explained by understanding that multiplication by 02 is essentially multiplication by the single-termed polynomial x. This will result in all degrees of a being incremented by 1. If the resulting polynomial has degree 8, then the irreducible polynomial must be added. If the resulting polynomial is not 8, then nothing is done. We see that each bit in the above expression that is a sum of some a_i and a_7, if a is of degree 7, then the resulting degree is 8, and the irreducible polynomial is added. This will result in $a_i + a_7 = a_i + 1$. If a is not of degree 7, then the resulting degree will not be 8. This will result in $a_i + a_7 = a_i + 0 = a_i$ because 0 is the identity of addition in $\mathbb{Z}_2$.

III. A MixColumns Architecture

The implementation of the architecture based on the mathematics in the previous section can be presented in four simple modules. Diagrams of the first three modules can be found in Figures 1, 2 and 3.

The multiplication by the term x is simply a one-bit shift to the left, and an addition by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ if the coefficient of the term x^7 of the input is 1. This simple module is shown in Figure 1.

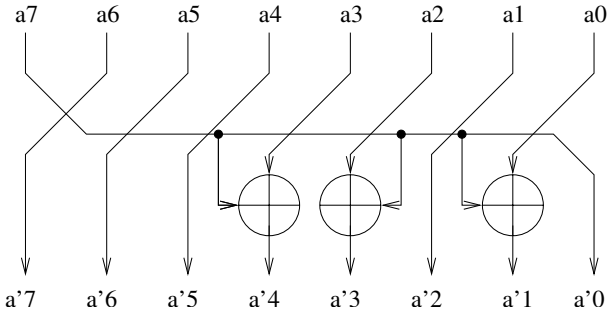

Fig. 1. Multiplication by constant polynomial x (module of $a \times 2$)

To generate the multiplications of $a \times 2^n$, i.e., $a \cdot x^n$, $(0 \leq n \leq 3)$ in $GF(2^8)$, we simply need to cascade various $a \times 02$ (i.e, $a \cdot x$) modules found in Figure 1. Figure 2 shows how these connections are made. The switch gates module in the figure denotes an eight bit array of switch gates, each of which takes the inv line as the base and one of wires in the wire array as the source. This is to propagate the signal of the wire array to the next $x2$ module if the inv line is set. If the line is not set, then logic 0 will be sent to the next $x2$ module from the sink of the switch gates. The interpretation of the function of this gate is that $a \times 2^2$ (i.e, $a \cdot x^2$) and $a \times 2^3$ (i.e, $a \cdot x^3$) for module input a will be calculated if the inverse of the mix columns operation is to be performed. If the operation is simply the forward operation, then multiplication by powers 2 and 3 are not used, and the input to the last two $a \times 2$ modules should be 0. Thus, the output of these two modules is then the identity element 0, and does not affect the XOR gates that follow in the next module.

The module $a \times 2^n$ (Figure 2) can be used to calculate two element multiplication in equations (3) and (4) as illustrated in Figure 3. Each $\oplus$ denotes an eight bit array of XOR gates that XOR the corresponding elements of the two input arrays. The hexadecimal numbers N_L/N_R in Figure 3 that mark the outputs of some of the $\oplus$ gates denote that the value $w_i \times N$ is output from the gate array. N_L is the number N used for encoding, while N_R is the number N used for decoding. We see that $N_R - N_L = t$ for some $t \in \{08, 0C\}$. It should be clearer now that if dec is set, then $N = N_R$, otherwise $N = N_L$.

Finally we can derive the logic circuit to compute the mix column operation as show in Figure 4. Four of the modules found in Figure 3 are used, one for each byte of the input word. We label the outputs from these modules as

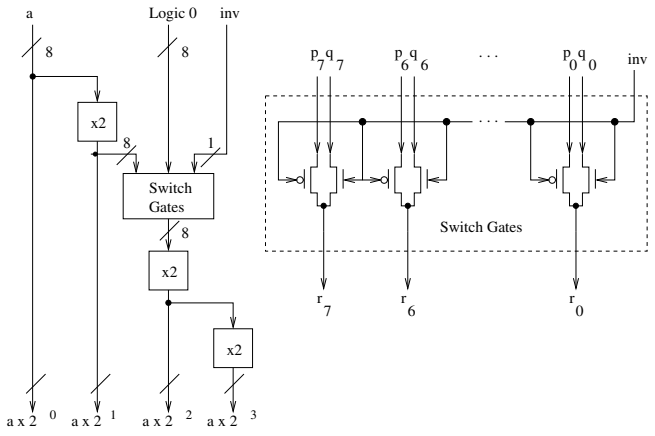

Fig. 2. Multiplications of $a \cdot x^n$ $(0 \leq n \leq 3)$ in $GF(2^8)$ (module of $a \times 2^n$)

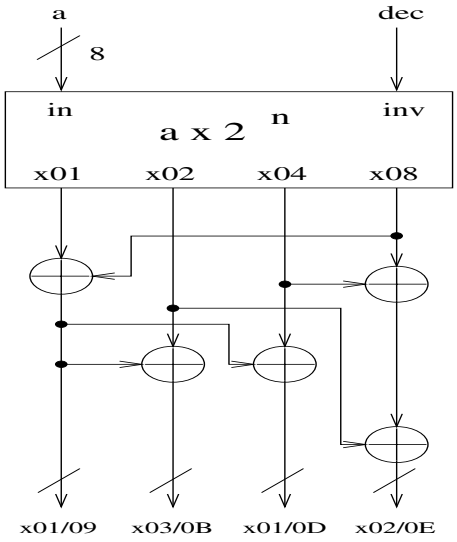

Fig. 3. Matrix product generator (module of product generator).

0109, $030B$, $010D$, and $020E$ where each output represents the product of the input w_i and some number n where $0 \leq i \leq 3$. The number n is the first number in the output label associated with that line if the dec line is not set, otherwise n is the second number. We calculate w'_i by adding, with XOR arrays, the appropriate outputs from the four modules that generate $w_i \times n$ for some $n \in GF(2^8)$. This is given by the following equations:

$$\begin{aligned} w'_3 &= w_3 \times E + w_2 \times B + w_1 \times D + w_0 \times 9 \\ w'_2 &= w_3 \times 9 + w_2 \times E + w_1 \times B + w_0 \times D \\ w'_1 &= w_3 \times D + w_2 \times 9 + w_1 \times E + w_0 \times B \\ w'_0 &= w_3 \times B + w_2 \times D + w_1 \times 9 + w_0 \times E \end{aligned} \quad (6)$$

Though the above equation is for the decode operation, the encode operation requires the exact same wiring of the exact same output from the four modules that generate the $w_i \times n$ outputs. Since these modules have already determined if the forward or the inverse values are output on the wire arrays, the connections described in the above equations will work

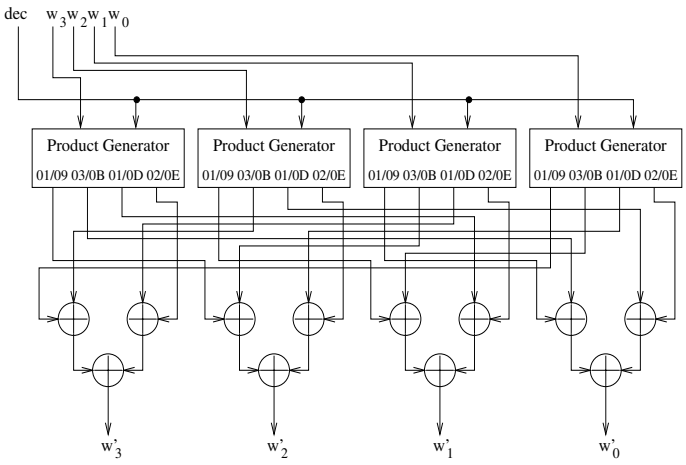

Fig. 4. Top-level view of the mix columns design.

for the forward operation as well. Therefore, the results of the forward operation will be output if the dec line is not set whereas the results of the inverse operation will be output in the same spot if the dec line is set.

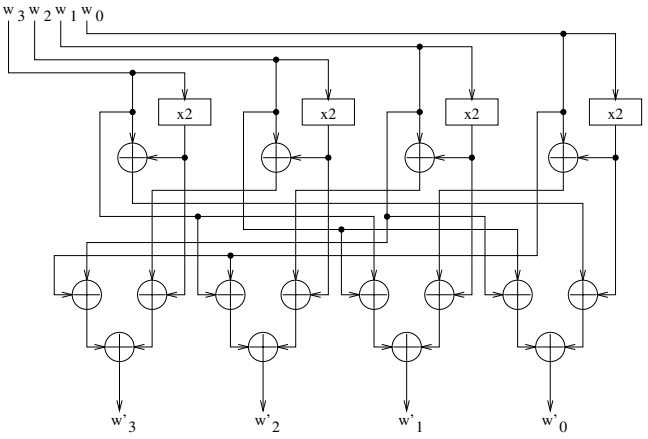

Fig. 5. A mix columns architecture that computes only the forward operation.

We also give a brief description of an architecture for the mix columns design that only computes the results of the forward operation. Some implementations of AES desire only the forward operation, such as the one found in [2]. We make special note of this design because it actually has less than half the amount of gates as in the design in Figure 4. Figure 5 shows the simple architecture of this encode-only module. We have removed the decode line, eight of the multipliers and many XOR arrays that were used to add in the $w_i \times 04$ and $w_i \times 08$ results, as they are not used in the forward operation.

IV. Comparison

The total number of gates in the presented design is $292_{XOR} + 32_{AND}$ and it results in a critical path of $6 \cdot T_{XOR} + T_{AND}$.

In [3], 16 multipliers are used for Mix Columns Operation, with 212 gates each. Replacing these multipliers with four

instances of our design (one for each column) can reduce the overall gate count by $2,096$ gates, which is a significant improvement considering that the paper claims $15,493$ gates in its high-performance design (14% of total gates are reduced). We also note that the design in [3] uses composite field arithmetic in the implementation of the s-box. Since the s-box and the mix columns (or its inverse) operations are performed in different clock cycles in the design in [3], and the s-box design has a critical path of at least $15 \cdot T_{XOR}$ (the AND gates were not considered in the analysis in the paper), we see that our design would not be the critical path in the whole AES design.

TABLE I
THE COMPARISONS OF SPACE AND TIME COMPLEXITY OF DIFFERENT MIX-COLUMN ARCHITECTURES FOR ENCRYPTION ONLY

Design	Gates	Critical Path	Results
Proposed Figure 5	140	4 GE	Encode Only
[2]	176	5 GE	Encode Only
[4]	176^{+}	6^{+} GE	Encode Only

Another design that is quite similar to our design is found in [2]. In this paper, only the forward operation is presented, not the inverse. Also, though they obtain the results with the same concept presented in our design, many unnecessary gates are used in the calculation of $w \times 2$ (i.e., $w \cdots x$) because of the multiplexing between the left-shifted result and the addition of the irreducible polynomial into the left-shifted result by using the left-most bit as the selection line. Our design avoids this by using that selection line as a direct XOR into corresponding x^i terms in the irreducible polynomial. This demands a maximum of $3XOR$ gates to multiply by x as opposed to the many that were used in their design when using the selector.

Table 1 illustrates the comparisons made between our design and another designs that implements only forward Mix Column operation, where GE refers to gate equivalents.

V. CONCLUSION

A fast and low complexity architecture for the AES Mix Column and Inverse Mix Column operations is proposed in this paper. The short critical path, small gate count and versatility (between encode and decode) is desirable for most architectural descriptions of AES. The performance comparisons indicate that the proposed mix-column architectures have less complexity than previous relevant work. In particular, the total number of gates in [4] can be reduced 14% off by applying our proposed mix-column architecture.

ACKNOWLEDGMENT

This research work is supported by the funding of Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] N. I. of Standards and Technology, *Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)*, http://csrc.nist.gov/publications/fips/fips197/fips197.pdf, 2001.

[2] H. Kuo, I. Verbauwhede, and P. Schaumont, "A 2.29 gbits/sec, 56 mw non-pipelined rijndael aes encryption ic in a 1.8v, 0.18 um cmos technology." [Online]. Available: citeseer.nj.nec.com/kuo02gbitssec.html

[3] S. Mangard, M. Aigner, and S. Moninikus, "A highly regular and scalable aes hardware architecture," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 483–491, April 2003.

[4] A. Rudra, P. Dubey, C. Jutla, V. Kumar, J. Rao, and P. Rohatgi, "Efficient rijndael encryption implementation with composite field arithmetic," in *Proc. Workshop Cryptographic Hardware and Embedded Systems*, ser. CHES, 2001, pp. 171–184.