An Efficient Pipelined Multiplicative Inverse Architecture for the AES Cryptosystem

Mostafa Abd-El-Barr and Amro Khattab

***Abstract*—In this paper, we introduce an architecture for performing a recursive pipeline algorithm for use in optimizing the performance of the multiplicative inverse operations in Galois Field GF(2^k). The latter is extensively used in performing the S-Box byte-Substitution in the AES cryptosystem. Using composite fields the operations are recursively decomposed into lower level ones which are mapped into pipelines. Thus, several gate reductions will be obtained and gate sharing. Eventually, this enhances the performance of computing the multiplicative inverse.**

***Index Terms*—AES cryptosystem, S-Box substitution, multiplicative inverse, pipelined multiplicative inverse.**

I. Introduction

Finite fields play an important role in digital communication applications. These include the Advanced Encryption Standard (AES) cryptosystem and error correction codes. Performance enhancement of finite field operations, such as addition, multiplication, square and multiplicative inverse has been attempted by a number of researchers in the form of optimized VLSI implementations [1]-[4]. The attempted enhancements cover on-the-chip space cost, time delay and power consumption.

The AES is a symmetric-key block cipher algorithm used to encrypt/decrypt sensitive data worldwide. According to the AES data to be encrypted is divided into equally sized blocks each is called a *state*. The algorithm performs a series of mathematical operations on each state based on the Substitution-Permutation Network principle to produce a cipher text. The algorithm starts with an initial step in which it adds the round key to the state. After that, the state is entered into a loop consisting of four repeated operations: sub-byte, shift-row, mix-column, and add round key. This is followed by a final iteration that excludes the mix-column operation. Among the four loop operations, the byte substitution operation is performed using the S-Box. The S-Box performs a non-linear transformation on data by replacing each individual byte by a different byte. The main purpose of the byte substitution is to bring confusion to the data to be encrypted. The replacement bytes can be obtained on the fly by determining the multiplicative inverse of a given state in finite field GF(2^8) followed by an affine transformation in GF(2).

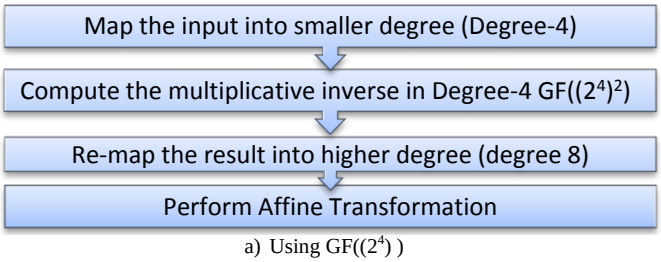
Manuscript received September 12, 2013; revised December 10, 2013. This work was supported by Kuwait University under Grant #WI 04/10.

Mostafa Abd-El-Barr is with the Department of Information Science, CCSE, Kuwait University, Safat 13060, Kuwait (e-mail: mostafa.abdelbarr@gmail.com).

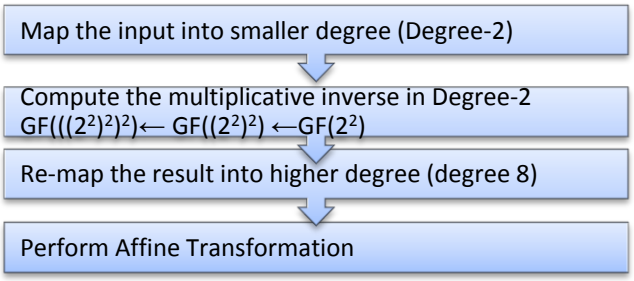
Amro Khattab is with the Computer Science Department, CCSE, Kuwait University, Safat 13060, Kuwait (e-mail: khattab@cs.ku.edu.kw).

There are several approaches to compute the multiplicative inverse. One possible approach is to use the Euclid's algorithm over the fields GF(2) and GF(2^m) [5]-[7]. The use of the normal bases in performing the multiplicative inverse operation is shown in [8]-[10]. The main advantage is the reduction of the square operation into a permutation of the binary representation. The main drawback is the complexity of the VLSI implementation. Other possible approaches to deal with multiplicative inverse include using elliptic curve technique [4] and using polynomial bases [1]-[3], [11], [9]. The main advantage of such approaches is their suitability to VLSI architecture.

One interesting approach for multiplicative inverse plays a crucial part in the S-Box byte-substitution computation of the AES. Fig. 1 provides illustrations of the S-Box computations using two composite filed approaches, i.e. GF(4) and GF(2).



a) Using GF((2^4))



b) Using GF($(((2^2)^2)^2$)

Fig. 1. S-Box computations using two different composite fields.

The use of the two composite fields GF(4) and GF(2) computation of the multiplicative inversion as means for producing compact AES hardware architecture with S-Box optimization has been shown in the literature, see for example [1]. A recursive efficient technique for computing the inversion in tower fields is introduced in [4] in which the authors have provided only an asymptotic estimate of the architecture without implementation.

In this paper we use the polynomial bases in developing a pipelined multiplicative inverse. The paper emphasizes using pipelined architectures in all Galois Field operations based on composite fields. The remaining parts are organized as follows. Section II provides background material on Galois

DOI: 10.7763/IJIEE.2014.V4.413

Fields and its basic operations. In Section III, we provide a brief coverage of the existing work conducted in implementing different techniques related to multiplicative inverse. Section IV provides details on the construction of the proposed recursive pipelined architecture for computing the multiplicative inverse. In Section V we present an analysis of the space and delay complexities of the proposed architecture with a comparison with other existing approaches in a way to demonstrate the effectiveness of the introduced architecture.

II. Background Material

The Galois Field GF(2^{2k}) is considered an extension field of degree 2 over GF(2^k). Assume the irreducible polynomial used in GF(2^{2k}) is $P(x) = x_2 + p_1x + p_0$ where p_1, $p_0 \in$ GF(2^k).

Take R as the root of $P(x)$, i.e. $R^2 + p_1R + p_0 = 0$. Then, for any element $A = a_1R + a_0$, the multiplicative inverse $B = b_1 R + b_0$ should satisfy the following condition:

$$A \times B = (a_0b_1 + b_0a_1 + a_1b_1p_1)R + (a_0b_0 + a_1b_1p_0) = 1,$$

where a_0, a_1, b_0, $b_1 \in$ GF(2^k) [2]. Solving this system of linear equations for b_0 and $b_1$gives:

$$b_0 = (a_0 + a_1p_1)\Delta^{-1}, \text{ and } b_1 = a_1\Delta^{-1},$$

where $\Delta = a_0(a_0 + a_1p_1) + p_0a_1^2$.

A. Composite Field GF((2^4)^2) Versus GF(((2^2)^2)^2)

Using Composite Field GF((2^4)^2), the following irreducible polynomials can be used: $x^2 + x + 1$ for GF(2^4) and $x^2 + x + \omega$ for GF((2^4)^2), $\omega = 1001$. Using Composite Field GF(((2^2)^2)^2), the following irreducible polynomials can be used: $x^2 + x + 1$ for GF(2^2), $x^2 + x + \varphi$ for GF((2^2)^2), and $x^2 + x + \gamma$ for GF(((2^2)^2)^2) where $\varphi = 10$, $\gamma = 1100$.

III. Existing Work

In [1] and [3] two architectures were introduced to implement the multiplicative inverse, one was based on the field GF((2^4)^2) and the other one based was based on the composite field GF(((2^2)^2)^2). Implementing the S-Box using these two architectures showed that GF(((2^2)^2)^2)used fewer circuit resources than GF((2^4)^2). According to [1], the implementation of the S-Box required a total of 294 gates in GF(((2^2)^2)^2) as area complexity which was 20% smaller than GF((2^4)^2) which required a total of 362 gates. Also, S-Box required 3.69 ns using GF(((2^2)^2)^2)as compared to 3.75ns using GF((2^4)^2). The gate implementation of the multiplicative inverse using the approach in [1] is shown in Fig. 2.

In GF((2^4)^2) $p_0 = \{1001\}_2$, while in GF(((2^2)^2)^2) $p_0 = \{1100\}_2$. Furthermore, The multiplicative inverse used in the computation in the field GF(((2^2)^2)^2) is implemented recursively in the field GF((2^2)^2). In the field GF((2^2)^2) the recursive implementation will be similar to the one shown in Fig. 2, using $p_0 = \{10\}_2$. The multiplication operation implementation in this case is shown in Fig. 3.

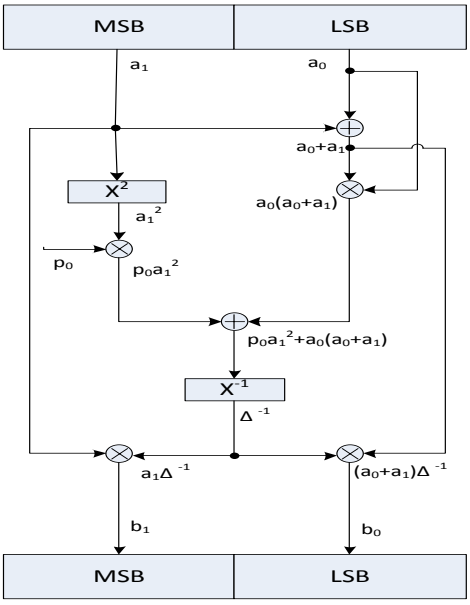


Fig. 2. Multiplicative inverse gate implementation over GF((2^4)^2).

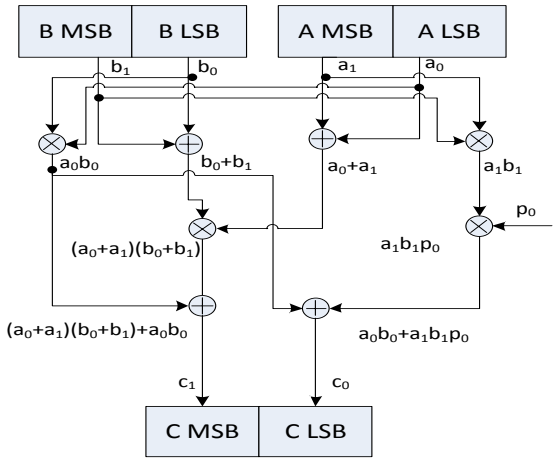


Fig. 3. Multiplication operation gate implementation.

IV. Proposed Architecture

In this paper, in order to distinguish the circuits used in different fields, we will use a notation according to which the data will be written with a postfix subscript number value indicating the data path size. Also, the figures will show the result of each operation typed next to the output arrows. The first improvement idea is to make efficient utilization of the resources available. Our main observation is that since some gates could be triggered concurrently, an improved circuitry should follow a pipelined approach. In a pipelined architecture, it is important to emphasize the order of the operations and hence, pipelined stages will be explicitly shown. Fig. 2 could now be redrawn as Fig. 4 to reflect the pipeline architecture. Here a subscript 4 indicates a data size of 4 bits, since the operations are implemented in the field GF((2^4)^2).

In Fig. 5 the pipeline had six pipeline stages. Compared to Fig. 4, the latter pipeline has less number of stages. Just by rewriting the equations, one could give the opportunity to execute more operations in parallel and minimize the dependency levels which leads to less delay. It should be noted in Fig. 4 that the inverse operation has 3 additions, 3 multipliers, two squares and on inverse operations in the

domain of GF(((2²)²)²).

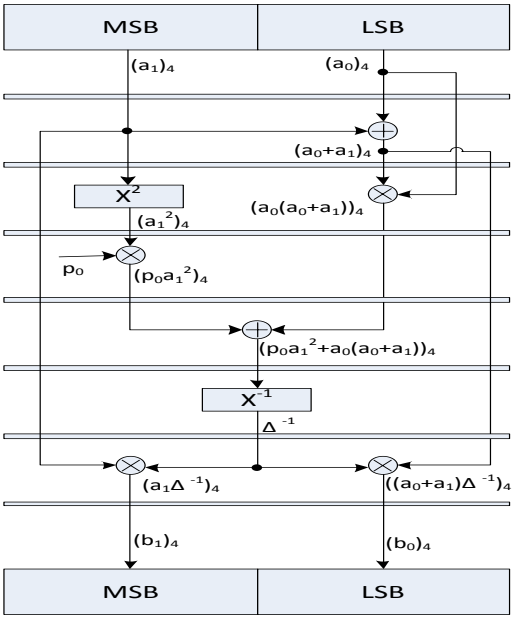


Fig. 4. Multiplicative inverse gate implementation over GF((2⁴)²) using a pipeline.

We observe that computing the value of:

$$\Delta = a_0(a_0 + a_1p_1) + p_0a_2^1 = a_0^2 + a_0a_1p_1 + p_0a_1^2$$

This is implemented as shown in Fig. 5.

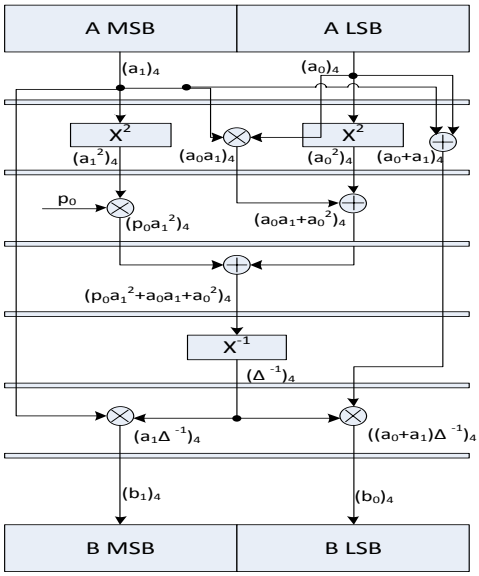


Fig. 5. Multiplicative inverse gate implementation over GF((2⁴)²) using a simplified pipeline.

Next, each operation in Fig. 5 will be investigated on its own to discuss its implementation closely. There are mainly four operations: addition, multiplication, square and multiplicative inverse. In Fig. 5 these operations are in the domain of field GF((2⁴)²), then each operation will be mapped into the domain of GF(((2²)²)²). The elements of first domain have a data path of size 4 bits, while the second is of size 2.

A. Addition

We start with the addition operation since it is the simplest to consider. It is implemented using exclusive or gates (Ex-OR) for each bit. So, an addition operation in GF((2⁴)²) requires four Ex-OR all executing in parallel. This is shown in Fig. 6. On the other hand one would perform an addition operation in GF(((2²)²)²) using only two Ex-OR gates.

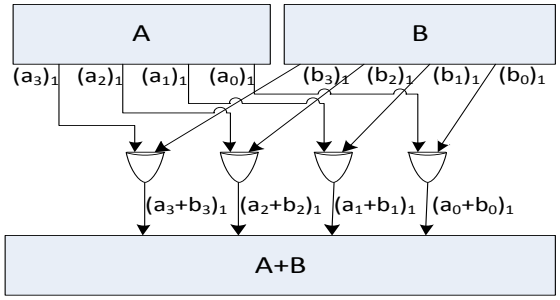
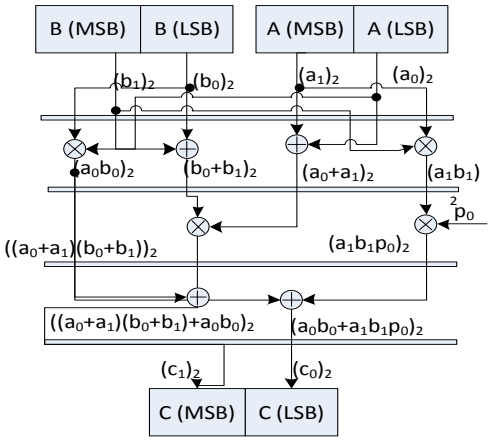


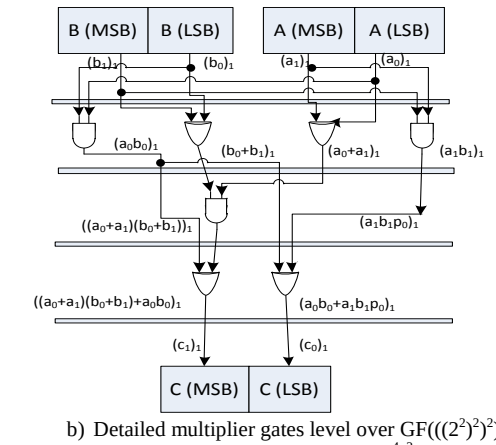
Fig. 6. Detailed adder implementation over GF(((2²)²)²).

B. Multiplication

With the multiplication operation, Fig. 3 is redrawn using a pipeline architecture which results Fig. 7 a). This figure shows that the multiplication over GF((2⁴)²) needs 3 multiplications, 4 additions and one constant multiplication in the domain of GF(((2²)²)²). Here, the operands are of size 2, so the operations could be interpreted to logical binary gates as shown in Fig. 7 b). The addition is done using Ex-OR and multiplication using AND gates. From this figure, there are three and gates and four Ex-OR.



a) Multiplier over GF((2⁴)²)



b) Detailed multiplier gates level over GF(((2²)²)²)

Fig. 7. Detailed multiplier implementation over GF((2⁴)²) and GF(((2²)²)²).

C. Square

For the square operation, one way of implementing it is by using the multiplier given in the previous section; just by multiplying the element by itself. This results in Fig. 8.

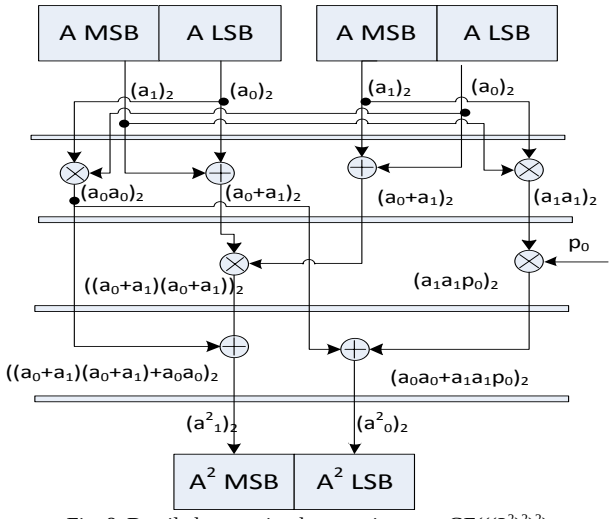
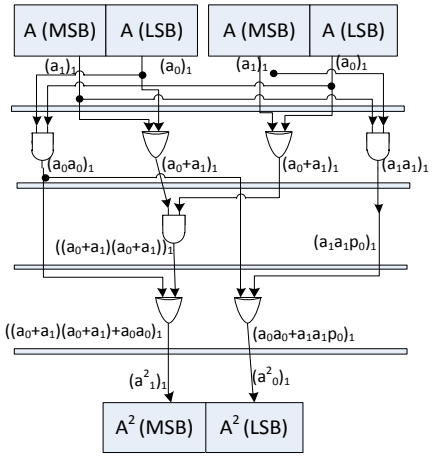
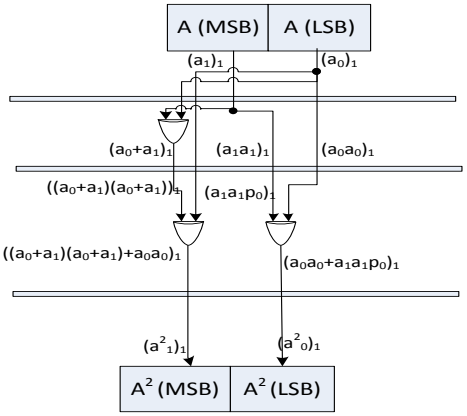


Fig. 8. Detailed square implementation over $GF(((2^2)^2)^2)$.

From this figure one can detect 3 square circuits and four additions in the $GF(((2^2)^2)^2)$. Then, all binary operations are mapped into the corresponding binary gates in Fig. 9 a). Next, several simplifications are done to get the square circuit of Fig. 9 b). For example, one of simplifications states that $a_0a_0 = a_0$; in other words a bit multiplied by itself results in the same bit. Also, this figure shows that the square in the field $GF(((2^2)^2)^2)$ contains only 3 Ex-OR.



a) Detailed Square implementation.



b) Simplified square operation gates level.

Fig. 9. Detailed simplified square implementation Over $GF(((2^2)^2)^2)$.

D. Multiplicative Inverse

The multiplicative inverse is the most expensive operation among the four operations discussed. In Fig. 5 the multiplicative inverse implementation was based on the domain of field $GF((2^4)^2)$ using a pipeline architecture. Notice that there is another inverse operation involved in stage 4, but in the domain of field $GF(((2^2)^2)^2)$. So, to implement this operation, one could use the same implementation of Fig. 5, recursively. This is shown in Fig. 10 a). Then, Fig. 10 b) reflects the binary gates level mapping. It contains 3 AND gates and 3 Ex-OR.

V. Experiments Analysis

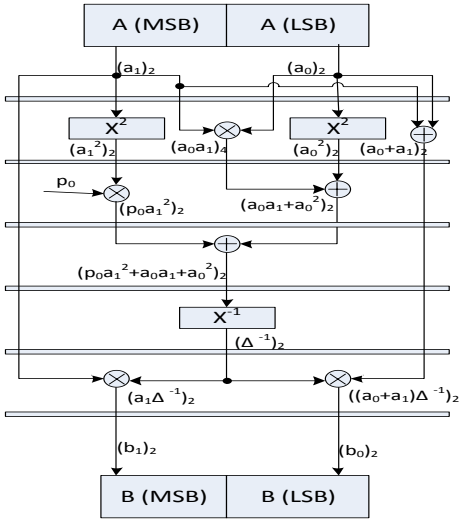
A. Multiplicative Inverse Space Complexity

In $GF(((2^2)^2)^2)$ area (in terms of Gate Count, GC) is computed as:

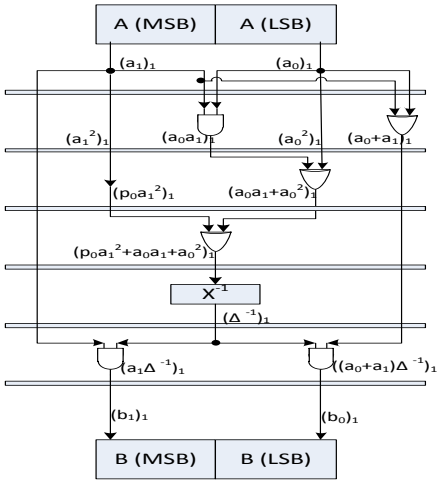
1) Addition (**A**) has: 4(*x*), in other words four multiplexers; check Fig. 6.
2) Multiplication (**M**) has: 3(*m*) + 4(*a*) + *c*; check Fig. 7 a).
3) Square (**S**) has: 3(*s*) + 4(*a*); as it is shown in Fig. 8.
4) Inverse (**I**) has: 1(*i*) + 2(*s*) + 3(*m*) + 3(*a*), as it is shown in Fig. 5.

Where the symbols represent:
(m): multiplication, (a) addition, (s) square, (i) inverse, and (c) constant multiplication on $GF((2^2)^2)$.

In $GF(2^2)$ these operations take the following area (in terms of GC):



a) Inverse implementation.



b) Detailed inverse operation gates level.

Fig. 10. Detailed simplified multiplicative inverse implementation Over $GF(((2^2)^2)^2)$.

1) (m) takes: 3 (•) + 4(x); check Fig. 7 b).
2) (s) takes: 3(x); as it is shown in Fig. 9 b).
3) (i) takes: 3(x) + 3(•), as it is shown in Fig. 10 b).
4) (a) takes: 2(x) , as it is shown in Fig. 6.
5) (c) takes: 1 (x)

where (•) represent AND gates and (x) Ex-OR gate. For the overall multiplicative inverse operation in GF(((2^2)^2)^2) there are 4(M), 2(S), 1(i) and 3(A). So the total area is: 48(•) and 152(x).

In order to convert this area cost in pure 2-way NAND gates; an AND gate takes two NAND gates and an Ex-OR gate takes 4 NAND gates as shown below.

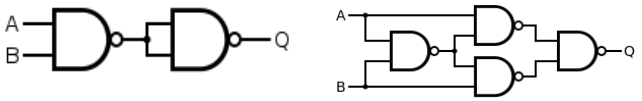


Fig. 11. An AND gate takes two NAND gates and an Ex-OR gate takes 4 NAND gates.

This way, the total area in NAND gate units for the multiplicative inverse operation is: 48(2)+152(4) = 704.

B. Multiplicative Inverse Time Complexity

There are 5 stages in the pipeline. The first stage has both multiplication and square operations. This takes a delay of 3 cycles, i.e. the delay taken by multiplication. Then, stage #2 contains addition and multiplication. Again, multiplication causes this stage to take 3 cycles delay. Next, stage #3 contains only addition operations, which would take one clock cycle. The following stage has the most expensive operation, i.e. the multiplicative inverse. It takes 5 cycles. The last stage contains multiplication operations running in parallel.

The total of the five stages in the pipeline, take 15 cycles. For m tasks using a pipeline of n-stages, the time required is

$$n+m\text{-}1$$

Speed-up $S(n)$ = Time using sequential processing/ Time using pipeline processing

$$= (m \times n \times t)/((n+m-1) \times t)$$
$$= (m \times n)/(n+m-1)$$

In our case the pipeline takes 5 stages, n=5. So, the speed-up could be rephrased as:

$$S(5) = 5m/(m+5-1) = 5m/(m+4).$$

VI. Comparison

This section provides a comparison with the performance results given in [1].

1) Comparison is based on area. The paper [1] shows that a multiplicative inverse based on a non-pipelined architecture takes over field GF(((2^2)^2)^2) 173 2-way NAND gates. On the other hand, the pipelined approach proposed takes 704 2-way NAND gates. This means that pipelined approach takes 4 times more space than a non-pipelined one. Fig. 12 shows a graph comparing the two results terms of area.

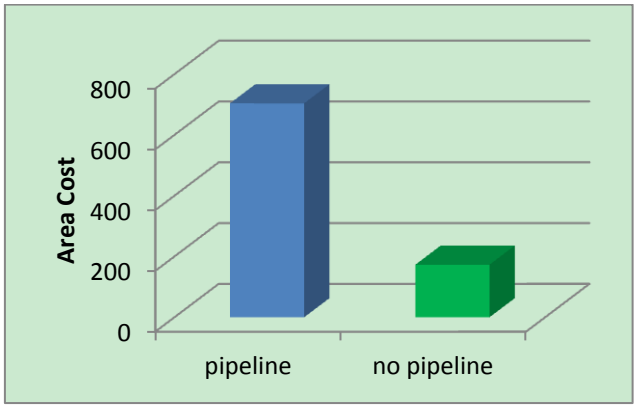


Fig. 12. Comparison between pipelined and a non-pipelined with respect to area.

2) Comparison is based on time delay. In the previous section it was shown that the non-pipelined approach of a multiplicative inverse takes: $m \times n \times t$ seconds whereas the pipelined approach takes $(n+m\text{-}1) \times t$ seconds.

Fig. 13 shows that as the time elapses, in the long run, the pipeline architecture outperforms the non-pipelined architecture.

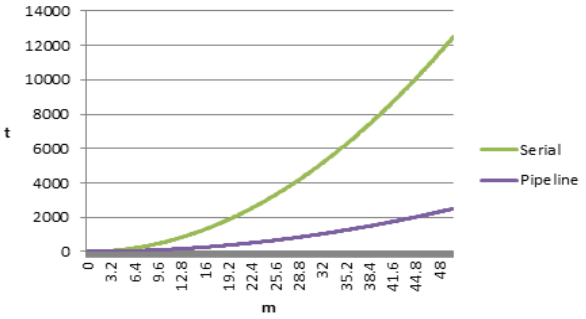


Fig. 13. Comparison between pipelined and a non-pipelined with respect to time delay.

VII. Conclusion

In this paper, a pipelined architecture for computing the multiplicative inverse for use in AES cryptosystem based on the pipeline techniques. This model is defined over the field GF(((2^2)^2)^2). The inverse operation was implemented recursively over the fields GF((2^2)^2). It was shown that the proposed pipeline approach achieves less time delay but at the expense of a bit more area.

Acknowledgment

The authors would like to acknowledge the financial support received from Kuwait University in the form of the funded research project WI 04/10.

References

[1] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with S-box optimization," in *Proc. ASIACRYPT*, 2001, pp. 239–254.
[2] J. Fan and C. Paar, "On efficient inversion in tower fields of characteristic two," in *Proc. IEEE International Symposium on Information Theory*, 1997, pp.20.
[3] A. Rudra, P. Dubey, C. Jutla, V. Kumar, J. Rao, and P. Rohatgi, "Efficient rijndael encryption implementation with composite field arithmetic," in *Proc. Cryptographic Hardware and Embedded Systems*, 2001, pp. 171-184.

[4] G. Jorge and P. Christof, "Efficient algorithms for elliptic curve cryptosystems," *Lecture Notes in Computer Science*, vol. 1294, pp. 342–356, August 1997.
[5] M. Hassan, "Efficient computation of multiplicative inverses for cryptographic applications," in *Proc. 15th IEEE Symposium on Computer Arithmetic*, June 2001, pp. 66-72.
[6] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in GF(2m)," *IEEE Transactions on Computers*, vol. 42, issue 8, pp. 1010–1015, August 1993.
[7] Z. Yan and D. Sarwate, "Systolic architectures for finite field inversion and division," *IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 789-792, 2002.
[8] T. Al-Somani and A. Amin, "Hardware implementations of GF(2m) arithmetic using normal basis," *Journal of Applied Sciences*, vol. 6, no. 6, pp. 1362-1372, 2006.
[9] C.-Y. Lee and C. Chiou, "New bit-parallel systolic architectures for computing multiplication, multiplicative inversion and division in GF(2m) under polynomial basis and normal," *Journal of Signal Processing Systems archive*, vol. 52, issue 3, pp. 313 - 324, September 2008.
[10] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2m) using normal bases," *Information and Computatio*n, vol. 78, issue 3, pp. 171–177, September 1988.
[11] N. Abu-Khader and S. Pepe,"Inversion/division in Galois field using multiple-valued logic," in *Proc. 37th International Symposium on Multiple-Valued Logic*, May 2007, pp. 16.

Mostafa Abd-El-Barr received his PhD degree in Computer Engineering, University of Toronto, Canada in 1986. He is with the Department of Information Science, College of Computing Sciences and Engineering (CCSE), Kuwait University since 2003 and an Adjunct Professor with the ECE Department, University of Victoria (UVic), BC, Canada. His research interests includeDesign and Analysis of Reliable & Fault-Tolerant Computer Systems, Computer Networks Optimization, Parallel Processing/Algorithms, Information Security, Multiple-Valued Logic (MVL) Design & Analysis, VLSI System Design, and Digital Systems Testing. He is the author and/or co-author of more than 160 scientific papers published in journals and conference proceedings/symposia. He has three books published (two are also translated to the Chinese Language). Professor Abd-El-Barr is a Senior IEEE Member and a member of the International Association of Engineers (IAENG). He is a Senior International Associate Editor of the International Journal of Information Technology & Web Engineering and a member of the Editorial of the International Journal of Computer Science and Security (IJCSS). Dr. Abd-El-Barr is a Certified Professional Engineer in Ontario, Canada. He is also an official IEEE/EAC/ABETaccreditation evaluator.

Amro Khattab has a MS degree in Computer Science, Kuwait University. He is working at the Computer Sciences Department, College of Computing Sciences and Engineering (CCSE), Kuwait University. His research interests include Cryptography and Computer Systems.
.