Synthesis Optimization on Galois-Field Based Arithmetic Operators for Rijndael Cipher

Petrus Mursanto

Faculty of Computer Science – University of Indonesia
Kampus UI, Depok 16424, Indonesia
Email: santo@cs.ui.ac.id

Abstract. A series of experiments has been conducted to show that FPGA synthesis of Galois-Field (*GF*) based arithmetic operators can be optimized automatically to improve Rijndael Cipher throughput. Moreover, it has been demonstrated that efficiency improvement in *GF* operators does not directly correspond to the system performance at application level. The experiments were motivated by so many research works that focused on improving performance of *GF* operators. Each of the variants has the most efficient form in either time (fastest) or space (smallest occupied area) when implemented in FPGA chips. In fact, *GF* operators are not utilized individually, but rather integrated one to the others to implement algorithms. Contribution of this paper is to raise issue on *GF*-based application performance and suggest alternative aspects that potentially affect it. Instead of focusing on *GF* operator efficiency, system characteristics are worth considered in optimizing application performance.

Keywords: *FPGA; Galois Field; Rijndael Cipher; VHDL.*

1 Introduction

Galois Field (*GF*) arithmetic plays an important role in modern communication system, particularly in two important aspects of information exchange, i.e. security and data correctness. *GF* is utilized in cryptography algorithm [1],[2] and error correction codes (ECC) [3],[4]. Performance of applications in these two fields is determined by the efficiency of GF arithmetic operators involved in the system [5]. There has been found in the literatures research efforts in improving *GF* operators' efficiency, e.g. multiplication [6], division [7] and inversion [8]. In fact, *GF* operators are not performing their functions individually and independently, rather they are parts of a functional integration at the system level. Is operator efficiency beneficial to the application level performance?

This paper reports an experimental result of implementing Rijndael encryption and decryption algorithms based on six variants of *GF* operator. The purpose of the experiment is to obtain a Rijndael configuration whose throughput is the most optimum. The Rijndael algorithm was implemented using VHDL by

Received November 24th, 2011, Revised June 10th, 2011, Accepted for publication June 27th, 2011.
Copyright © 2011 Published by LPPM ITB, ISSN: 1978-3086, DOI: 10.5614/itbj.ict.2011.5.2.2

means of two synthesis tools: the **Xilinx** ISE 8.2i and the **Altium** ProChip Designer.

2 Previous Research

Similar to the ordinary algebra, GF algebra has a number of arithmetic operations, such as: addition, subtraction, etc. Variants of GF arithmetic operators are characterized by:

1. operation types: multiplication, division, inversion, square or square root
2. representation basis: standard/polynomial (PB), normal (NB) or dual (DB)
3. processing types: parallel or serial.

In digital circuit, *GF* addition and subtraction are simply implemented by exclusive-OR logic operation. The advance of digital technology has shifted performance measurement mechanism from the running time of software algorithm [9] to VLSI complexity, i.e. the number of components and their total delay [6].

The first circuit structure of *GF* arithmetic was proposed by Berlekamp in 1982, i.e. polynomial and dual based multiplication [10]. Normal based multiplier was introduced firstly by Massey-Omura in 1986 [11], which is known afterward as MO multiplier. In 1988, Mastrovito proposed a more modular multiplier with higher regularity of the structure that suits systolic cells in VLSI [12]. However, speed, size and modularity of Mastrovito's multiplier depend much on the irreducible polynomial *P(x)* used to generate the field elements. By selecting the right *P(x)*, parallel multiplication has at most 2^m-1 logical gates and occupies 55% of the space required for implementing Bartee and Schneider's algorithm [9]. In 1991, Mastrovito's dissertation reported an experimental investigation on multiplication using more than one representation basis [13]. It was concluded that PB multiplier is the most versatile form for the most arithmetic computational problems of $GF(2^m)$ in VLSI. In addition, PB solution also possess conversion cost that can compensate the efficiency gained by the other representation basis [14] and occupies a half space of the one required by MO multiplier. Mapping problem for inter-basis conversion is the concern of Wu, et al. [15] which introduced an efficient conversion method from PB to NB specifically for squaring. Furthermore, Sunar, et al. proposed conversion matrix for any form of generator polynomial [16].

Several improvements of multiplication algorithm were also reported by Afanasyev [17] and similarly by Hasan, et al. [18] that proposed a modification of the architecture by defining the irreducible polynomial as all-one polynomial (AOP). By applying the AOP, Hasan claimed the complexity of multiplication

decreases by 50%. Meanwhile, Lee-Lim also reported a performance improvement by applying circular dual basis (CDB) [19]. Lee's method is very efficient for trinomial with composite $GF((2^n)^m)$ where m is primary relative over n, or $gcd(m,n) = 1$. However, defining certain form of irreducible polynomial is considered as limitation, inflexible and low reusability [20].

A comprehensive study on *GF* arithmetic was reported by Paar's dissertation [21], in which he proposed a decomposition algorithm from $GF(2^k)$ to $GF((2^n)^m)$ where $k = n.m$, called **composite field**. In addition, Paar also explored inversion after the first algorithm introduced by Itoh-Tsujii in 1988 [22]. Paar's further research in [23] reported composite field multiplication and inversion in $GF(2^8)$. Composite field implementation in FGPA showed components saving by 25% and acceleration by 10% [24]. The composite field inversion requires 29% of AND and XOR gates compared to the standard one. Rudra [25] and Jutla [26] also developed a method for linear transformation of *GF* binary elements to composite field representation.

Combination of serial and parallel processes were reported by Choi, et al. [27] that introduced hybrid multiplier by forming irreducible polynomial $x^m + x^n + 1$ where $n \leq m/2$. This hybrid multiplier has flexible structure to compromise space and time complexity and is proven having less complexity than Wu and Hasan's multipliers [15]. Several other methods were proposed to support hardware implementation, such as Huang-Wu [28] that has systolic array architecture approach to ease the testing process.

Previous implementation of Rijndael cipher has been reported, such as improvement of arithmetic efficiency based on composite field by Rudra et.al. [25], optimization of transformation using Look Up Table by Lee [29],[30], and performance improvement of SubBytes algorithm specifically on S-Box module by Rijmen [31].

3 Motivation

Previous research focused on efficiency improvement of GF operators to obtain better performance in term of speed or occupied space when implemented in digital circuits. However, literatures on GF-based implementation at system level are merely experience sharing with specific features without any analysis on the consequences of GF operator variants involved in the system. Triggered by the issues discussed in the available literatures, the experiments were designed to answer the following research questions:

1. Can performance improvement gained at operator level be obtained linearly at the application level?

2. How can *GF*-based circuit optimization be achieved at application level?
3. Will an application take benefit by employing all best variants of *GF* operators?

4 Methodology

A set of experiments was designed to examine whether the best variants of GF operator can be combined to construct the most efficient application. In other words, which configuration of GF operator variants produces an application with the greatest throughput? Arithmetic operator types were taken as suggested by the algorithm of Rijndael cipher 128-bit. The operators were implemented as the most efficient variant from each combination of two parameters: representation basis (PB, NB or DB), and processing structure (parallel or serial). For further reference in the next discussion, we use six variants of operators as follows:

Table 1 GF operator variants.

Variant	Structure	Basis
1	Parallel	Polynomial
2	Serial	
3	Parallel	Normal
4	Serial	
5	Parallel	Dual
6	Serial	

The Rijndael cipher is implemented into six versions, each of which was constructed based on the variants in Table 1. They were implemented using structural VHDL and synthesized with Xilinx ISE 8.2i and Altium ProChip Designer. Synthesis parameters were set for speed as the goal, optimized the period of the entire design with normal optimization effort. The synthesis process results in maximum combinational path or total delay that defines the maximum frequency possibly supplied to the system. Hence, the system throughput can be calculated based on the data capacity proceeded per time unit, expressed in Mega Byte per second (MBps). Throughput is then used as an indicator of the system performance. An optimal configuration is defined as the one having biggest throughput among the six versions of the system.

5 *GF* Operator Architectures

This section briefly discusses the six variants of *GF* operator, in particular the one widely used in encryption and decryption processes, i.e. the multiplication. Variant 1 of multiplier implements Mastrovito's circuit in [12]. Multiplication is

processed partially in variant 2 by the circuit shown in Figure 1. Variant 3 and 4 are realized based on the NB multiplier in [13]. Multiplier variant 5 and 6 are implemented using parallel and serial structures in [13].

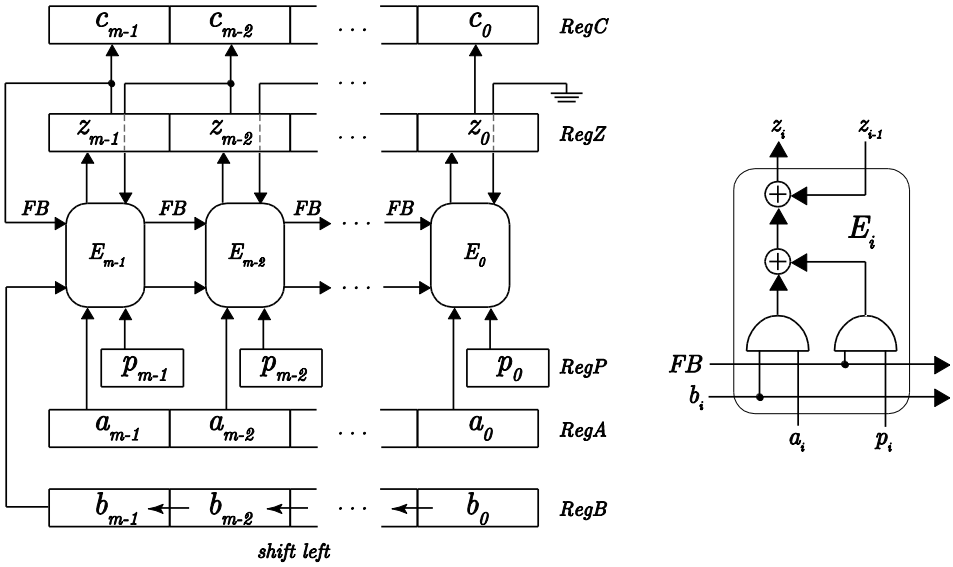

Figure 1 Serial PB Multiplier *GF(2m)*.

The implementation of six variants of multiplier results in delays shown in Table 2. It can be seen that the parallel multiplier in dual basis has the biggest combination delay. The best variant is the one having the smallest delay, i.e. polynomial based parallel multiplier or variant 1.

Table 2 Delay of $GF(2^4)$ multiplier in ns.

Structure	Tools	PB	NB	DB
Parallel	Xilinx	12.69	13.49	17.61
	Altium	11.37	11.94	14.25
Serial	Xilinx	15.96	15.18	15.31
	Altium	15.24	16.26	15.14

6 Rijndael Cipher

This session describes the implementation of Rijndael cipher with focus on hardware structure involving *GF* algebra. Decipher is a reverse process of cipher and vice versa. More details on analytic theory and algorithm's philosophic background, its strength and weakness as well as limitation are

covered in [32].

According to Rijndael encryption scheme, one block of data is converted to ciphertext by means of a number of transformation algorithms. Temporary result of internal process in delivering ciphertext is called *State*. State is represented in a square of byte arrays. It has four rows and a number of columns. Number of columns is defined using a variable that equals to block length divided by 32. This session describes the implementation of Rijndael cipher with data block and key length of 128 bits.

Encryption algorithm is arranged in several steps of computation such as shown in Figure 2. In initial phase (iteration 0), State is computed by XORing data block and the cipher key. Furthermore, State is going through 10 iterations consisting of computation phase SubBytes (SB), ShiftRows (SR), MixColumns (MC) and AddRoundKey (ARK). Specific for the 10^{th} iteration, State skips the MC process. In every iteration, result of MC is XORed with Round Key which is unique for corresponding iteration. The Round Key is produced by Round Key Generator from a number of transformations over the cipher key.

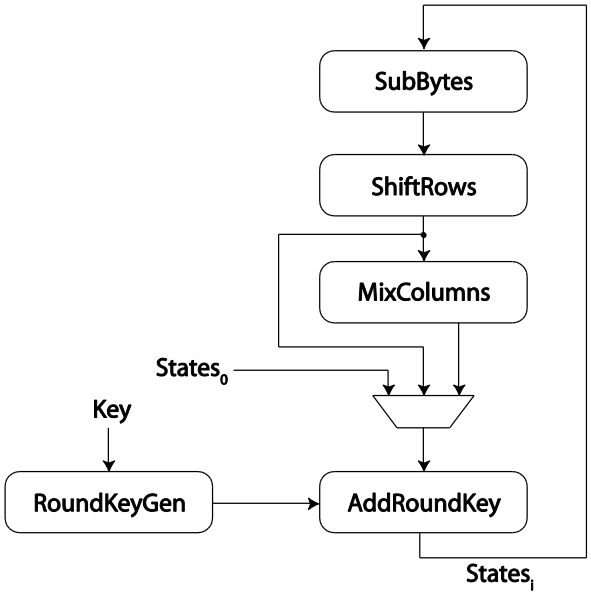

Figure 2 Rijndael Cipher Process.

Detail of internal process in each block within Figure 2 has been discussed in [31]. This paper presents the structure of Rijndael's blocks and their throughput as consequences of *GF* operator variants selected in the implementations.

6.1 SubBytes

SubByte (SB) transformation is a byte non-linear substitution, operates on each State independently. Substitution table (known as S-Box) is invertible and built with composition of two main steps, i.e.: multiplicative inversion and Affine transformation. *GF(2^8)* inversion can be accomplished in either parallel or serial. Direct parallel inversion for *GF(2^4)* has been discussed in [12] with subfield in [13]. Affine transformation delay is duration time required by XOR operation. Considering additional cycles for Affine transformation, throughput of SubBytes is obtained as shown in Table 3.

Table 3 Throughput SubBytes in MBps.

Structure	Tool	PB	NB	DB
FulPar	Xilinx	105.56	92.00	80.00
	Altium	113.56	98.40	86.36
Parallel	Xilinx	102.40	149.20	n.a.
	Altium	112.00	162.80	n.a.
Serial	Xilinx	116.60	131.40	120.48
	Altium	125.60	135.08	130.68

6.2 ShiftRows

In ShiftRows (SR), the row of State is shifted in *cyclic* or left rotated with offset varies from 0 to 3. SR is directing input bytes to different row of output bytes. Implementation delay equals to zero or 'cost-free' since there is no gate involved.

6.3 MixColumns

In MixColumn, column of State is considered as a polynomial of *GF (2^8)* and multiplied by modulo of x^4+1 with specific polynomial:

$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

MixColumns delay is duration time of multiplication which is implemented in parallel or serial. In this case, there are multiplication with constant operands, i.e. 01, 02 and 03. Performance evaluation is conducted over the throughput of MixColumns if the multiplier is implemented in one of the six variants in Table 1. Result of throughput measurement using Xilinx and Altium tools is shown in Table 4.

Table 4 MixColumns throughput in MBps.

Structure	Tool	PB	NB	DB
Parallel	Xilinx	819.6	746.8	620.0
	Altium	897.2	814.0	669.6
Serial	Xilinx	842.0	554.8	858.4
	Altium	907.2	570.4	931.2

6.4 AddRoundKey

This is a simple process, i.e. XORing State with the Round Key resulted from RKG in corresponding iteration. Delay of this process comes from the XOR gates.

6.5 Round Key Generator

RKG is to produce Round Key for every iteration. The original cipher key is only for iteration 0, and then used by RKG for delivering Round Key in iteration 1. Round Key 1 is processed for delivering Round Key in iteration 2, and so on. Circuit delay is the duration time required for multiplicative inversion. Round Key Generator has similar structure with SubBytes involving only inversion and XOR. Performance measurement of ARK can be seen in Table 5. It is shown that normal based inversion variant has the highest performance as demonstrated in [33].

Table 5 Throughput Round Key Generator in MBps.

Structure	Tool	PB	NB	DB
FulPar	Xilinx	211.12	183.96	161.84
	Altium	227.16	196.8	172.72
Parallel	Xilinx	117.08	186.76	n.a.
	Altium	128.20	203.44	n.a.
Serial	Xilinx	118.44	134.96	122.64
	Altium	127.56	138.72	133.00

6.6 Rijndael Cipher Performance

The overall performance of Rijndael encryption requires a uniform of symbol representation basis. For that reason, the best performance must be identified for every representation basis. In summary, Table 6, Table 7 and Table 8 show the best throughput of the Rinjdael's modules produced by the three representation basis.

Table 6 The best throughput of SubBytes in MBps.

Tool	Modul Architecture	Throughput	Δ × clk
Xilinx	SymbSerial Serial PB	116.60	2.11 × 65
	SymbSerial Parallel NB	149.20	21.42 × 5
	SymbSerial Serial DB	120.48	2.33 × 57
Altium	SymbSerial Serial PB	125.60	1.96 × 65
	SymbSerial Parallel NB	162.80	19.66 × 5
	SymbSerial Serial DB	130.68	2.15 × 57

Round Key Generator (RKG) runs in parallel with three iterative processes, they are: SubBytes (SB), ShiftRows (SR) and MixColumns (MC). Results of those parallel processes are XOR-ed with Add Round Key (ARK) to produce new State that becomes the input for next iteration. Hence, number of cycles required in iteration is defined by the biggest one of the two processes whose period follows the slowest module. Throughput is calculated based on parallel processes in four columns; in this case the total data is 16 bytes.

Table 7 The best throughput of MixColumns in MBps.

Tool	Module Architecture	Throughput	Δ × clk
Xilinx	Serial Multp PB	842.0	2.11 × 9
	Parallel Multp NB	746.8	21.42 × 1
	Serial Multp DB	858.4	2.33 × 8
Altium	Serial Multp PB	907.2	1.96 × 9
	Parallel Multp NB	814.0	19.66 × 1
	Serial Multp DB	931.2	2.15 × 8

Table 8 The best throughput of Round Key Generator in MBps.

Tool	Modul Architecture	Throughput	Δ × clk
Xilinx	FullPar Inv PB	211.12	75.79 × 1
	Parallel Inv NB	186.76	85.68 × 1
	FullPar Inv DB	161.84	98.87 × 1
Altium	FullPar Inv PB	227.16	70.44 × 1
	Parallel Inv NB	186.76	78.64 × 1
	FullPar Inv DB	172.72	92.64 × 1

Table 9 The highest throughput of encryption.

Tool	Basis	Δ/clk	#clock	Δ(ns)	Thrgput (MBps)
Xilinx	PB	75.79	65+9+1	5684.25	2.82
	NB	85.68	5+1+1	599.76	26.68
	DB	98.87	57+8+1	6525.42	2.45
Altium	PB	70.44	65+9+1	5283	3.03
	NB	78.64	5+1+1	550.48	29.07
	DB	92.64	57+8+1	6114.24	2.62

It is shown in Table 9 that encryption performance is very low since it has to accommodate the delay of RKG which is built in parallel structure. It is therefore required to build a serial structure of RKG as long as the number of cycles in total does not exceed the number of cycles accumulated by SB, SR and MC. PB and DB can have operational structures in serial per symbol with serial operators; whereas NB should have serial structure per symbol with parallel operators. In this case, serial operators result in number of cycles exceeds the number of cycles for encryption. Performance of serial structured RKG is presented by Table 10.

Table 10 Throughput serial Round Key Generator (RKG).

Tool	Modul Architecture	Thrghput (MBps)	Δ × clk
Xilinx	Serial-Serial Inv PB	29.61	2.11 × 64
	Serial-Parallel Inv NB	46.69	21.42 × 4
	Serial-Serial Inv DB	30.66	2.33 × 56
Altium	Serial-Serial Inv PB	31.89	1.96 × 64
	Serial-Parallel Inv NB	50.86	19.66 × 4
	Serial-Serial Inv DB	33.25	2.15 × 56

By changing RKG structure from parallel to serial, performance of the system increases. It is evidently shown by the increasing throughput of the application significantly in Table 11.

Table 11 Throughput of Rijndael cipher with serial RKG.

Tool	Basis	Δ/clk	#clock	Δ(ns)	Thrgput (MBps)
Xilinx	PB	2.11	65+9+1	158.33	101.06
	NB	21.42	5+1+1	149.94	106.71
	DB	2.33	57+8+1	153.78	104.04
Altium	PB	1.96	65+9+1	147	108.84
	NB	19.66	5+1+1	137.62	116.26
	DB	2.15	57+8+1	141.77	112.86

It can be seen on Table 11, Rijndael cipher with normal based representation has the highest throughput. It is shown consistently by both Xilinx and Altium tools. It proves that the most efficient at PB operators does not deliver the most optimal Rijndael application. Performance degradation of RKG modularly in fact improves the throughput of integrated system in the application.

7 Automatic Synthesis Optimization

Experiment results show that employing the best operators does not automatically produce the most efficient application. This phenomenon is shown consistently by Xilinx and Altium synthesis tools. Simple and common logic saying that parallel process is faster than serial does not hold. This is caused by the fact that the synthesis tools have initiated automatic improvement or limited optimization to the VHDL structural design.

Multiplication and inversion are obviously the dominant process in Rijndael encryption. Examination on the results shows that multiplication with fixed bit operands is optimized by removing unnecessary components in the system. For specific configuration of Rijndael cipher, *P(x)* and *g(x)* are fixed during the system lifecycle. *g(x)* is one of the operand performing as *a(x)* in Figure 1. Fixed values of p_i and g_i cause the content of block E_i in Fig.1 can be simplified becoming the circuits shown in Figure 3. All combinations of a_i and p_i within internal block E_i can be optimized without any AND gate. By omitting AND gates, Xilinx reduces significantly the delay so that the minimum period is 2.5 ns. Hence, processing 4 bit requires 10 ns, which is faster than parallel multiplication delay 12.69 ns.

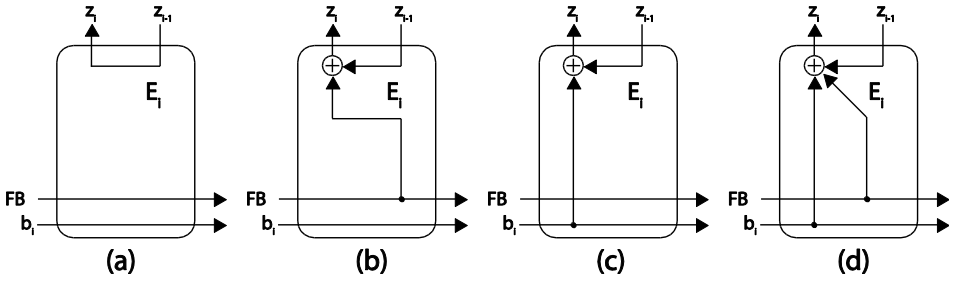

Figure 3 Internal component of E_i in PB Serial Multiplier for (a) $a_i=0$; $p_i=0$ (b) $a_i=0$; $p_i=1$ (c) $a_i=1$; $p_i=0$ (d) $a_i=1$; $p_i=1$.

It is examined that optimization was not applied to parallel multiplications although they also have constant operands. It is due to the fact that the tools optimize constant bit only, whereas parallel multiplier signal is implemented as an *m*-bit bus.

DB multiplier is superior over the PB and NB variants in term of fully serial delivery of the product. With constant values of *P(x)* and *A(x)*, DB serial multiplier delivers the product bit by bit as the operand b_i enters from MSB to LSB. Therefore, variant 6 saves 1 cycle compared to variant 2 and 4 that delivering the product after the whole cycle for *m*-bit completes. For that reason, variant 6 based systems can process addition serially as the product is delivered from index *m-1* to *0*.

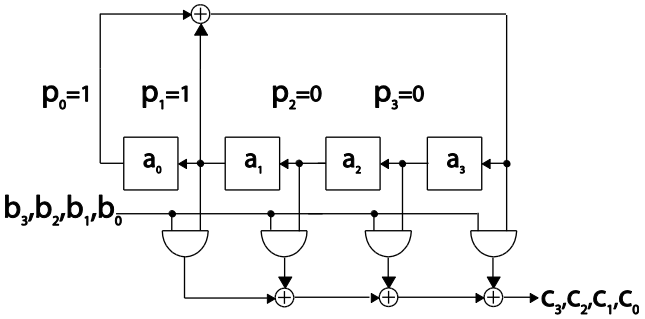

Figure 4 DB serial multiplier with constant *P(x)*.

With constant $P(x) = x^4 + x + 1$, optimization steps applied to serial DB multiplication is shown by Figure 4. In general this finding supports several statements in the literature that multiplication with constant operands can be more efficient [10]. It was examined as well that optimization does not apply to serial NB multiplier. In serial NB multiplication, the values of both operands change dynamically during the lifecycle of the system due to the rotation of internal registers.

8 Conclusions and Further Research

This paper reports that an optimal performance of Rijndael cipher does not always require the best variants or the most efficient *GF* operators. Combining all best operators, where each is the most efficient variant, is not a simple mechanistic conversion process. In addition, when implemented in FPGA, efficiency improvement is also contributed by synthesis tools that optimize serial operators whose operands are constant. This explains why the experimental results show the superiority of serial based system over parallel ones. This result supports previous finding with similar experiments on Reed Solomon Codec [34].

Obtaining synergic efficiency at system level requires careful considerations on several aspects of the system. Based on the facts obtained in this paper, it is worth to investigate system characteristics as the main contributors for the

application performance, i.e. the operator composition and distribution, interaction between them and types of internal process within the system.

It is interesting to examine further the consistency of this optimization in other *GF*-based applications. Explorative experiments are required for Rijndael AES with cipher-key 128-bit [35] or RS (255,223) 8-bit such as the one used by NASA [36]. Hypothetical prediction suggests that higher performance ratio would be obtained by serial variants over the parallel ones. It is because of additional combinational path in parallel operators that results in bigger delays. Meanwhile, serial operators requires only several additional cycles with constant minimum periods.

Acknowledgement

Part of this material is based upon work supported by European Union Project VN/Asia-Link/001 (79754). The author would like to thank Prof. R.G. Spallek and Mr. Jörg Schneider for providing tools and facilities in the Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur, Technische Universität Dresden, Germany.

References

[1] van Tilborg, H., *An Introduction to Cryptology*, Kluwer Academic Publishers, 1988.

[2] Schneier, B. *Applied Chryptography*, Wiley&Sons, 1993.

[3] Wicker, S.B., *Error Control Systems for Digital Communication and Storage*, Prentice Hall, New Jersey 07458, 1995.

[4] Sweeney, P., *Error Control Coding from Theory to Practice*, John Wiley & Sons, Inc., 2002.

[5] Lin, S. & Costello, D.J. *Error Control Coding*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[6] Wang, C., Truong, T., Shao, H., Deutsch, L., Omura, J. & Reed, I., *VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)*, IEEE Transactions on Computers, **34**(8), pp. 709–717, August 1985.

[7] Hasan, M., Wang, M. & Bhargava, V., *Division and Bit-Serial Multiplication over GF(q^m)*, IEEE Transactions on Computers, **41**(8), 972-980, August 1992.

[8] Zhou, T., Wu, X., Bai, G. & Chen, H., *Fast GF(p) Modular Inversion Algorithm Suitable for VLSI Implementation*, Electronics Letters, **38**(14), pp. 706-707, 4 July 2002.

[9] Bartee, T. & Schneider, D., *Computation with Finite Fields*, Information and Computers, **6**, 79–98, March 1963.

[10] Berlekamp, E., *Bit-Serial Reed-Solomon Encoders*, IEEE Transactions on Information Theory, **28**, 869-874, November 1982.

[11] Massey, J. & Omura, J., *Computational Method and Apparatus for Finite Field Arithmeti*c, US Patent No.4, 587, 627, 1986.

[12] Mastrovito, E.D., *VLSI Designs for Computations Over Finite Field GF (2^m)*, Master's Thesis No: 159, Linkping Studies in Science and Technology Linkping, Sweden, Dec 1988.

[13] Mastrovito, E., *VLSI Architectures for Computations in Galois Fields*, PhD Thesis, Dept of Electrical Eng, Linkping Univ, Sweden, 1991.

[14] Gollman, D., *Algorithmenentwurf in der kryptographie Habil*. University of Karlsruhe Preprint, 1990.

[15] Wu, H., Hasan, M.A., Blake, I.F. & Gao, S., *Finite Field Multiplier Using Redundant Representation*, IEEE Transactions on Computers, pp. 1306-1316, Nov 2002.

[16] Sunar, B., Savas, E. & Koç, C.K., *Constructing Composite Field Representations for Efficient Conversion*, IEEE Transactions on Computers, **52**(11), 1391-1398, November 2003.

[17] Afanasyev, V., *Complexity of VLSI Implementation of Finite Field Arithmetic*, In Proc of II Int'l Workshop on Algebraic and Combinatorial Coding Theory Leningrad, USSA., pp. 6-8, 1990.

[18] Hasan, M., Wang, M. & Bhargava, V., *A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields*, IEEE Transactions on Computers, **42**(10), pp. 1278–1280, October 1993.

[19] Lee, C.-H. & Lim,J.-I., *A New Aspect of Dual Basis for Efficient Field Arithmetic*, Samsung Advanced Inst of Technology, 1998.

[20] Fenn, T.S., Benaissa, M. & Taylor, D., *GF(2^m) Multiplication and Division Over the Dual Basis*, IEEE Transactions on Computers, **45**(3), pp. 319–327, March 1996.

[21] Paar, C., *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*, PhD Thesis, Univ of Essen, Germany, June 1994.

[22] Itoh, T. & Tsujii, *A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) using Normal Basis*, Journal Information and Computation, **78**(3), pp. 171–177, Sep. 1988.

[23] Paar, C. & Rosner, M., *Comparison of Arithmetic Architectures for Reed-Solomon Decoders in Reconfigurable Hardware*, IEEE Symposium on

FPGA-Based Custom Computing Machines (FCCM), pp. 219–225, April 1997.

[24] Mursanto, P., *Comparison of Galois Field Mutlipliers in Standard and Composite Field Architectures*, In National Conference on Computer Science & Information Technology, Depok Fasilkom UI, 29-30 January 2007.

[25] Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J. & Rohatgi, P., *Efficient Rijndael Encryption Implementation with Composite Field Arithmetic*, Proc of 3rd Int'l Workshop on Cryptographic Hardware and Embedded Systems, Springer-Verlag, **2162**, pp. 171-184, 2001.

[26] Jutla, C., Kumar, V. & Rudra, A., *On The Circuit Complexity of Isomorphic Galois Field Transformations*, Technical Report RC22652 (W0211-243) IBM Research Division, Nov 2002.

[27] Choi, Y., Chang, K.-Y., Hong, D. & Cho, H., *Hybrid Multiplier for GF(2^m) Defined by Some Irreducible Trinomials*, Electronics Letters, **40**(14), pp. 852–853, July 8, 2004.

[28] Huang, C.-T. & Wu, C.W., *High-Speed Easily Testable Galois-Field Inverter*, IEEE Transactions on Circuit and Systems, **48**(9), pp. 909–918, September 2000.

[29] Li, H., *A Parallel S-Box Architecture for AES Byte Substitution*, International Conference on Communications, Circuits and Systems, **1**(1), pp. 1–3, 27-29 June 2004.

[30] Li, H., *A New CAM Based S-Box Look Up Table in AES*, IEEE International Symposium on Circuits and Systems, **5**, pp. 4634–4636, 23-26 May 2005.

[31] Rijmen, V., *Efficient Implementation of the Rijndael S-box F.W.O*. Post-Doctoral report, Dept. ESAT, Belgium, 2001.

[32] Daemen, J. & Rijmen, V., *AES Proposal: Rijndael*, March 9, 1999 Ver 2.

[33] Mursanto, P., *Manfaat Representasi Elemen Berbasis Normal dalam Meningkatkan Kinerja Operator Aritmetika Galois Field*, In Proc 6th National Conf. Design and Application of Technology, Widya Mandala Univ., pp. 121–127, Jul 19, 2007.

[34] Mursanto, P., *Performance Evaluaton of Galois Field Arithmetic Operators for Optimizing Reed Solomon Codec*, In Proceeding International Conference on Instrumentation, Communication, Information Technology & Biomedical Engineering (ICICI-BME) Bandung, Nov 23-25, 2009.

[35] Daemen, J. & Rijmen,V. , Rijndael, *The Advanced Encription Standard*, Dr.Dobb's Journal, **26**, pp. 137–139, Mar 2001.

[36] Sklar, B., *Digital Communications Fundamentals and Applications*, Prentice Hall, Inc., New Jersey 2nd ed, 2003.