Novel Processor Array Structure for Finite Field Inversion

Atef Ibrahim[1,2,a] Turki Alsomani[3,b] Fayez Gebali[4,c]

[1]Prince Sattam Bin Abdulaziz University, AlKharj, Saudi Arabia

[2]Electronics Research Institute, Cairo, Egypt

[3]Umm Al-Qura University, Makkah, Saudi Arabia.

[4]University of Victoria, Victoria, BC, Canada

[a]aa.mohamed@psau.edu.sa, [b]tfsomani@uqu.edu.sa, [c]fayez@ece.uvic.ca.

ABSTRACT

In this paper we present new processor array structure to perform inversion operation in $GF(2^m)$ based on the modified extended Euclidean algorithm. This array has simple structure with processing elements have local communication with each other. Also, it has low area and power complexities as well as a moderate speed compared to the previously reported designs. ASIC implementation results of the proposed design and the previously reported ones show that the proposed design reduces area complexity by ratios ranging from 18.3% to 56.0% and also reduces energy by ratios ranging from 11.4% to 77.3% over the previously reported design. This makes the proposed design more suited to the embedded applications that have more restriction on area and power consumption.

KEYWORDS

processor arrays, embedded applications, finite field inversion, information security, ASIC, digital circuits design.

1 INTRODUCTION

Power consumption limits the application of public key cryptosystems (PKC) in portable devices. Elliptic-curve cryptography (ECC) algorithms have the merit of giving the same level of security using smaller key sizes [1] comparing to other PKC algorithms. This resulted in using ECC algorithms in applications that have more restrictions on power/energy and timing [2]. Recently, more ECC hardware implementations that meet power/energy and timing limitations of these devices and applications have been reported in the literature [3, 4, 5, 6, 7, 8, 9]. Most of these implementations are concentrated on the efficient implementation of field multiplication and field inversion operations as they are the most expensive operations in ECC cryptography.

Compared to other schemes used to compute the field inversion (e.g., Fermats little theorem) the extended Euclidean schemes are considered the most efficient in terms of area and speed, but they may use variable size counters pre or post calculations [10, 11, 12] that makes them cumbersome for scalable VLSI implementations.

Processor arrays are the efficient hardware approach used to implement inversion in GF(2^m) [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 19, 26, 26, 27, 28]. Processor array architectures have some attractive features such as modularity, regularity, and concurrency that makes them more suitable for high-speed VLSI system design. Most of the previously reported processor array schemes target the high-performance applications with different time and area complexities and a few of them targets the resource-constrained embedded applications that have restrictions on area and power [28].

In this paper, we propose a processor array structure that is suited to the resource-constrained devices to perform inversion operation in $GF(2^m)$ based on a previously modified extended Euclidean algorithm. This ar-

ISBN: 978-1-941968-37-6

chitecture is composed of one-dimensional array of logic cells and have low area complexity of O(m). It is explored by applying a regular technique to the inversion algorithm. In this technique, the algorithm is first converted to a regular iterative algorithm, then the algorithm data dependence graph and a suitable affine scheduling function are obtained. The systolic architecture obtained have simple structure with processing elements that have local communication with each other.

The organization of this paper is as follows: Section 2 gives a brief discussion about the extended Euclidean-based finite field inversion algorithm over GF(2^m) and the conversion of this algorithm into bit-level form. Section 3 discusses the proposed systolic architecture for this iterative algorithm. Section 4 explains the complexity of the proposed design and compares it to the previous work. Finally, Section 5 concludes this work.

2 FINITE FIELD INVERSION

In GF(2^m), A finite field can be defined using the following irreducible polynomial:

$$Q(x) = x^m + q_{m-1}x^{m-1} + \cdots + q_2x^2 + q_1x + 1 \quad (1)$$

where $q_i \in GF(2)$ for $0 < i < m$.

In $GF(2^m)$, a field element A can be represented by the polynomial:

$$A(x) = \sum_{i=0}^{m-1} a_i\, x^i \quad (2)$$

where $a_i \in GF(2)$ for $0 \leq i < m$.

In GF(2), a multiplicative inverse of $A(x)$ is a polynomial $\hat{A}$ such that $\hat{A}(x)A(x) \equiv 1 \bmod Q(x)$, where $\hat{A}(x)$ is designated as $[A^{-1}(x) \bmod Q(x)]$. The most commonly used inversion algorithms are based on extended Euclidean algorithm (EEA) [29], Fermats little theorem [30] and Gaussian elimination [22]. EEA is widely used to perform inversion in practice.

The need for long polynomial division in each iteration of the conventional EEA-based inversion algorithm in $GF(2^m)$ makes it inefficient in both hardware and software implementations. This problem was solved in part by exchanging the degree comparison with a counter [31]. In [32], Yan et al. proposed a modified EEA-based inversion algorithm as shown in Algorithm 1. This algorithm computes four intermediate polynomials, $R(x), S(x), Y(x)$, and $H(x)$ that are stored in $m+1$-bit registers with bits numbered as $m, m-1, \cdots, 1, 0$. The most significant bits of registers R and S are the highest degree terms of the $R(x)$ and $S(x)$ (i.e., $R(x) = r_m x^m + \cdots + r_1 x^1 + r_0 x^0$ and similarly for $S(x)$), while the least significant bits of registers Y and H are the highest degree terms of $Y(x)$ and $H(x)$ (i.e., $Y(x) = y_m x^0 + \cdots + y_1 x^{m-1} + y_0 x^m$ and similarly for $H(x)$). Yan et al. [32] proved that there is no need to store the m^{th} bit of $R(x), S(x), Y(x)$, and $H(x)$. Thus, the registers can be shortened from $m+1$-bit to m-bit. Here we use R^i, S^i, H^i, Y^i and the ring counter D^i to denote the values of R, S, H, Y and D after i^{th} iteration. s_{m-1}^{i-1} and d_0^{i-1} represents the most significant bit (MSB) and least significant bit (LSB) of S^{i-1} and D^{i-1}, respectively. The ring counter D bits are ordered from right to left as $(d_{m-1}d_{m-2}\cdots d_0)$. The complement of control bit $c1$ is represented as $\overline{c1}$. In the initial step of the algorithm, the coefficients of the irreducible polynomial $Q(x)$ and the coefficients of polynomial $A(x)$ are assigned to the variables R^0 and S^0, respectively. Since the MSB of Q(x) is always equal to 1, this bit does not need to be computed or stored as mentioned in [32]. Thus, Q(x) coefficients can be stored in a register of size m. Through each iteration of the for loop, the control bits and variables are changed as follows:

- control bit $c1$ is set to the value of the MSB of S^{i-1}.
- control bit c2 is set to the value resulted from the logic anding of control bit $c1$ and control bit $sign^{i-1}$.
- control bit $sign^i$ is either set to $\overline{c1^i}$, if $sign^{i-1}$ bit have value of 1, or to the LSB of D (d_0^{i-1}), if $sign^{i-1}$ bit have value of 0.

- The variable H^i is either set to the value of Y^{i-1}, if $c2$ is set to 1, or to the value of the shifted left H^{i-1}, if $c2$ is set to 0. (In this case, dividing polynomial $H(x)$ over x is equivalent to shifting variable H to left that the highest degree terms of polynomial $H(x)$ is on the right).
- The variable R^i is either set to the value of the shifted left S^{i-1}, if $c2$ is set to 1, or remains unchanged, if $c2$ is set to 0.
- The variable S^i is either set to the value resulted from the logic xoring of R^{i-1} and shifted left S^{i-1}, if $c1$ is set to 1, or to the value of the shifted left S^{i-1}, if $c1$ is set to 0.
- The variable Y^i is either set to the value resulted from the logic xoring of shifted left H^{i-1} and Y^{i-1}, if $c1$ is set to 1, or to the value of Y^{i-1}, if $c1$ is set to 0.
- The variable D^i is either set to the value of the shifted left D^{i-1}, if $sign^i$ is set to 1, or to the value of the shifted right D^{i-1}, if $sign^i$ is set to 0.

This algorithm differers from many other EEA variants [22, 33, 34] in that it has no modular operations and this resulted in achieving short critical path delay, Besides making it more secure against side-channel attacks for fixed number of iterations [28].
Algorithm 2 shows a modification in terms of bits of Algorithm 1. In this algorithm, the terms r_j^i, s_j^i, y_j^i and h_j^i represent the j-th bit of R, S, Y and H at iteration i, respectively.

3 PROPOSED SYSTOLIC ARRAY STRUCTURE

Figure 1 shows the proposed processor array design when $m = 3$. The updated bits of $d_{m-j-2}^i,\ s_j^i,\ h_j^i,\ d_{m-j-1}^i$ are propagated between the processing elements (PEs), while the updated bits of $y_j^i,\ r_j^i$ are stored locally. Control bits ($c1^i,\ c2^i,\ sign^i$), $1 \leq i < 2m-1$, are generated inside PE$_m$ and broadcast to the PEs as shown in Figure 1. Since the input bits ($d_{m-j-2}^0,\ h_{j-1}^0,\ d_{m-j-1}^0,\ y_j^0$), $0 \leq j \leq m-1$,

Algorithm 1 Pseudo code of the EEA-based inversion algorithm [32].

Input: two polynomials $A(x)$ and $Q(x)$
Output: $\hat{A} = A^{-1}(x) \bmod Q(x)$
$R^0(x) \leftarrow Q(x),\ S^0(x) \leftarrow A(x),\ Y^0(x) \leftarrow x^m,\ H^0(x) \leftarrow 0,\ D^0 \leftarrow 2,\ sign^0 \leftarrow 1$
for $1 \leq i < 2m$ **do**
  $c1^i = s_{m-1}^{i-1}$
  $c2^i = c1^i\ AND\ sign^{i-1}$
  $sign^i \leftarrow \begin{cases} \overline{c1^i}, & \text{if } sign^{i-1} = 1 \\ d_0^{i-1}, & \text{if } sign^{i-1} = 0 \end{cases}$
  if $c1 = 1$ **then**
    $S^i \leftarrow (R^{i-1} + xS^{i-1})$
    $Y^i \leftarrow (H^{i-1}/x + Y^{i-1})$
  else
    $S^i \leftarrow xS^{i-1}$
    $Y^i \leftarrow Y^{i-1}$
  end if
  if $c2 = 1$ **then**
    $\{R^i, H^i\} \leftarrow \{xS^{i-1}, Y^{i-1}\}$
  else
    $\{R^i, H^i\} \leftarrow \{R^{i-1}, H^{i-1}/x\}$
  end if
  $D^i \leftarrow \begin{cases} 2D^{i-1}, & \text{if } sign^i = 1 \\ D^{i-1}/2, & \text{if } sign^i = 0 \end{cases}$
end for
$\hat{A} = H^{2m-1}$

have fixed values they can be generated by clearing or setting the flip flops of the processor array through the initial timing step as will be explained in the next paragraph. Therefore, there is no need to provide these input bits to the processor array. Only input bits that should be provided to the processor array are the bits of s_{j-1}^0 and r_j^0 as shown in Figure 2.
Figure 2 (a) shows the details of the control processing element PE$_m$. Figure 2 (b) shows the details of PE$_j$ when ($1 \leq j < m$) . Figure 2(c) shows the details of the simplified processing element PE$_0$. MUX $M3$ and the XOR gate next to it, shown in Figure 2 (b), are removed in Figure 2(c) that the input signals $s_{-1}^0,\ s_{-1}^{i-1}$ are equal to 0. MUXs $M4,\ M5,\ M6$ and $M7$, shown in Figure 2(b), are replaced by AND

Algorithm 2 Pseudo code of the bit-level EEA-based inversion algorithm.

INITIALIZE
$R^0 = (r^0_{m-1} \cdots r^0_1 r^0_0) \leftarrow (q^0_{m-1} \cdots q^0_1 1)$
$S^0 = (s^0_{m-1} \cdots s^0_1 s^0_0) \leftarrow (a^0_{m-1} \cdots a^0_1 a^0_0)$
$H^0 = (h^0_{m-1} \cdots h^0_1 h^0_0) \leftarrow (0 \cdots 00)$
$Y^0 = (y^0_{m-1} y^0_{m-2} \cdots y^0_1 y^0_0) \leftarrow (00 \cdots 001)$
$D^0 = (d^0_{m-1} \cdots d^0_2 d^0_1 d^0_0) \leftarrow (0 \cdots 010)$
$sign^0 \leftarrow 1$
$d^{i-1}_{-1}, s^{i-1}_{-1}, h^{i-1}_{-1}, d^{i-1}_m = 0$, for $1 \leq i < 2m$
for $1 \leq i < 2m$ **do**
  $c1^i = s^{i-1}_{m-1}$
  $c2^i = c1^i AND\ sign^{i-1}$
  if $sign^{i-1} = 1$ **then**
    $sign^i = \overline{c1^i}$
  else
    $sign^i = d^{i-1}_0$
  end if
  for $m-1 \leq j \leq 0$ **do**
    if $c1^i = 1$ **then**
      $s^i_j = r^{i-1}_j + s^{i-1}_{j-1}$
      $y^i_j = h^{i-1}_{j-1} + y^{i-1}_j$
    else
      $s^i_j = s^{i-1}_{j-1}$
      $y^i_j = y^{i-1}_j$
    end if
    if $c2^i = 1$ **then**
      $r^i_j = s^{i-1}_{j-1}$
      $h^i_j = y^{i-1}_j$
    else
      $r^i_j = r^{i-1}_j$
      $h^i_j = h^{i-1}_{j-1}$
    end if
  end for
  $D^i \leftarrow \begin{cases} 2D^{i-1}, & \text{if } sign^i = 1 \\ D^{i-1}/2, & \text{if } sign^i = 0 \end{cases}$
end for
output: $\hat{a}_j = h^{2m-1}_{m-j-1}$, for $0 \leq j \leq m-1$

gates as shown in 2(c) that the input signals s^{i-1}_{-1}, h^{i-1}_{-1}, d^{i-1}_m are equal to 0 and the input signal r^0_0 is equal to 1. Also, the lower XOR gate in Figure 2(b) is removed in Figure 2(c) that the value of h^{i-1}_{-1} is equal to 0.

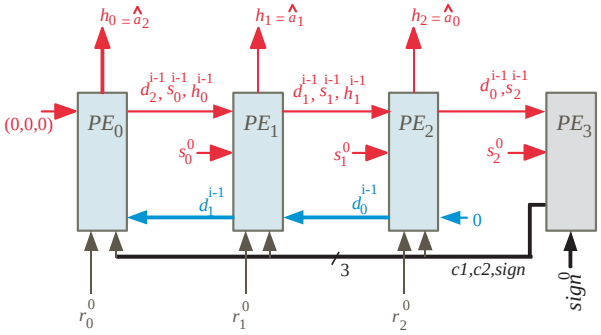

Figure 1. Proposed processor array architecture for $m = 3$.

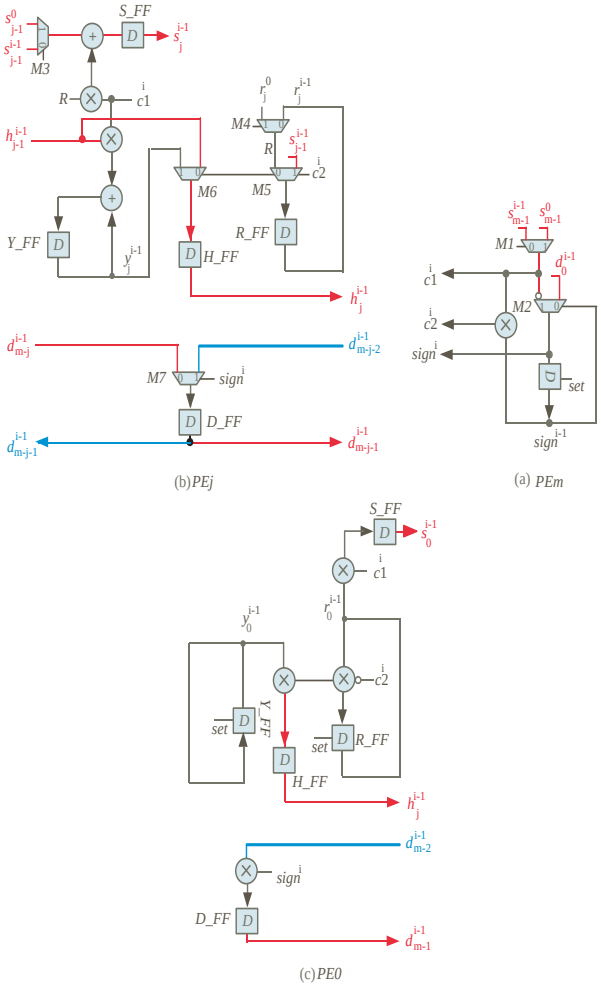

Figure 2. Design 1 when $m = 3$. (a) PE_m details. (b) PE_j details when $0 < j < m$. (c) PE_0 details. Squares labeled D are flip-flops with load and reset/set control inputs.

We summarize the operation of each PE for the proposed design as follows.

1. At time $t = 1$, the FF of PE_m in Figure 2(a) is set to have the initial bit $sign^0$ equal to 1.
2. At time $t = 1$, the (D_FF)'s of PE_j and PE_0 in Figures 2(b) and Figure 2(c) are cleared to have the input bits d_j^0 $(0 \leq j \leq m-1, j \neq m-2)$ equal to 0, except the D_FF of PE_{m-2} is set to have the input bit d_1^0 equal to 1.
3. At time $t = 1$, the (Y_FF)'s of PE_j in Figures 2(b) are cleared to have the input bits y_j^0 $(0 < j \leq m-1)$ equal to 0, except the Y_{FF} of PE_0 in Figure 2(c) is set to have the input bit y_0^0 equal to 1.
4. At time $t = 1$, the (R_FF)'s of PE_j in Figures 2(b) are cleared, except the R_{FF} of PE_0 in Figure 2(c) is set to obtain the initial value of r_0^0 equal to 1. Also, at the same time the MUX's $M4$ in Figures 2(b) are set to pass the input bits of r_j^0, $(0 < j \leq m-1)$.
5. At time $t = 1$, the (S_FF)'s of PE_j and PE_0 in Figures 2(b) and Figure 2(c) are cleared. Also, at the same time the MUX's $M3$ in Figures 2(b) are set to pass the input bits s_j^0, $(0 < j \leq m-1)$.
6. At time $t = 1$, the (H_FF)'s of PE_j and PE_0 in Figures 2(b) and Figure 2(c) are cleared to obtain the zero input bits h_j^0, $(-1 \leq j \leq m-2)$.
7. At time $t \geq 1$, control bits $c1^i$, $c2^i$, $sign^i$ are broadcasted to all PE's.
8. At time $t > 1$, the signals s_{j-1}^{i-1} are pipelined by selecting the lower input of MUX $M3$.
9. At time $t > 1$, the signals s_{m-1}^{i-1} and r_j^{i-1} are passed through MUXs $M1$ and $M4$, respectively.
10. At time $t = 2m-1$, the output result H is obtained where bit h_j is produced from PE_j, $(0 \leq j \leq m-1)$.

4 DELAY AND AREA COMPLEXITIES COMPARISON

The delay and area complexities of the proposed design can be estimated from Figure 1. Table 1 compares the proposed design to previously reported efficient designs of [24, 27, 28] in terms of area complexity (gates, muxs, and flip-flops), critical path delay, and latency. The following terms are used in Table 1:

1. T_{Inv} is the delay of inverter
2. T_A is the delay of AND gate
3. T_{MUX} is the delay of MUX
4. T_N is the delay of NAND gate
5. T_X is the delay XOR gate
6. $\tau_1 = 2T_A + T_{MUX}$
7. $\tau_2 = 2T_{MUX}$

We notice from Table 1 that the proposed design has area of 2 NOT gates, $2m+2$ AND gates, $2m-2$ XOR gates, $3m-2$ MUXs, and $4m$ flip flops. This area complexity is less than that required by the other designs of [24, 27, 28]. Also, the output of the proposed design is obtained after a latency of $2m-1$ clock cycles and the critical path duration is $\tau_2 = 2T_{MUX}$. The latency and the critical pass delays of the proposed design are less than that of the designs of [24, 27] and they are similar to the design of [28].

For real implementation, we described the proposed design and the previously efficient reported designs of [24, 27, 28] in VHDL at RTL level and synthesized them, using (45 nm, 1.1 V) standard-cell CMOS technology, to obtain the gate level for $m = 233$. The tool used for logic synthesis and power analysis is Synopsys tool package version 2005.09-SP2. All implementation results are obtained for 1.8 V and 25°C operating conditions. Table 2 compares the obtained ASIC implementation results of the efficient finite field inverters. In this table, the "Latency" is the total clock cycles needed to finish single inversion operation.

Table 1. Area and delay Comparison between different finite field inverters.

Design	NOT	AND	XOR	MUXs	Flip-Flops	Latency	Critical Path delay
Danesh-beh [27]	0	$6m$	$6m$	$18m$	$30m$	$5m-4$	τ_1
Yan [24]	$4m$	$6m$	$4m$	$12m$	$32m$	$5m-2$	τ_2
Fan [28]	1	$2m+3$	$2m+2$	$3m+4$	$5m+6$	$2m-1$	τ_2
Proposed Design	2	$2m+2$	$2m-2$	$3m-2$	$4m$	$2m-1$	τ_2

The "Area" is the area of the inverters in terms of 2-input NAND-gate equivalents; the "Clock frequency" measures the performance of the inverter; while "Inverter delay" and "Power" are the values of time and power consumed by the inverter to finish a single operation. We used the obtained synthesis results to calculate the energy and throughput rate to measure the optimization degree attained in each inverter. We notice from this table that the proposed design saves more area (ranging from 18.3% to 56.0%) and saving more energy (ranging form 11.4% to 77.3%) over the compared efficient designs that makes it very suitable for applications that have more restrictions on area and power consumption. Also, it has the same throughput as the design of Fan [28].

5 SUMMARY AND CONCLUSION

This paper presented a new processor array structure to perform inversion operation in $GF(2^m)$ based on a previously modified extended Euclidean algorithm. It has simple structure with processing elements that have local communication with each other. The implementation results showed that the proposed processor architecture is more efficient in area and power than the processor architectures previously reported in the literature.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the NSTIP (MAARIFAH) strategic technologies program number (13-INF2541-10) and the Science and Technology Unit (STU) at Umm Al-Qura University in the Kingdom of Saudi Arabia.

REFERENCES

[1] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York Inc., New York, USA, 2004.

[2] J. Kaps, *Cryptography for Ultra-Low Power Devices*. PhD thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, 2006.

Table 2. ASIC implementation results of the efficient finite field inverters for $m = 233$.

Inverter	Latency	Area [Kgates]	Clock Frequency [GHz]	Inversion delay (T) [μs]	Power [μW]	energy [PJ]	Throughput (m_{bits}/T) [Mbps]
Daneshbeh [27]	1161	11.08	2.911	0.399	10.30	4.1	583.9
Yan [24]	1163	10.28	2.920	0.398	8.8	3.50	585.4
Fan [28]	465	5.53	2.922	0.159	6.6	1.05	1465.4
Proposed Design	465	4.52	2.926	0.159	5.9	0.93	1465.4

[3] E. Ozturk, B. Sunar, and E. Savas, "Low-power elliptic curve cryptography using scaled modular arithmetic," 2004.

[4] G. Gaubatz, J. Kaps, E. Ozturk, and B. Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," in *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, pp. 146–150, 2005.

[5] J. Wolkerstorfer, "Scaling ecc hardware to a minimum," in *Austrochip 2005 Mikroelektronik Tagung* (N. K. und Peter Rössler, ed.), pp. 207 – 214, 2005.

[6] G. de Dormale, R. Ambroise, D. Bol, J. Quisquater, and J. Legat, "Low-cost elliptic curve digital signature coprocessor for smart cards," in *IEEE 17th Int. Conf. on Application-Specific Systems, Architectures and Processors*, pp. 347–353, 2006.

[7] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Public-key cryptography on the top of a needle," in *In Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS'07*, pp. 1831–1834, 2007.

[8] M. Feldhofer and J. Wolkerstorfer, "Strong crypto for rfid tags: a comparison of low-power hardware implementations," in *In Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS'07*, pp. 1839–1842, 2007.

[9] F. Frbass and J. Wolkerstorfer, "Ecc processor with low die size for rfid applications," in *In Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS'07*, pp. 1835–1838, 2007.

[10] C. T. Huang and C. W. Wu, "High-speed c-testable systolic array design for galois-field inversion," in *Proceedings of European Design and Test Conference 1997*, pp. 342–346, 1997.

[11] R. Schroeppel, H. Orman, S. OMalley, and O. Spatscheck, "Fast key exchange with elliptic curve systems," in *Advances in Cryptology: Proc. Eurocrypt'96*, pp. 43–56, 1995.

[12] N. T. Y. Watanabe and K. Takagi, "A vlsi algorithm for division in gf(2^m) based on extended binary gcd algorithm," *IEICE Transactions of the Institute of Electronics Information and Communication Engineers*, vol. E-85, no. 5, pp. 994–999, 2002.

[13] G. L. Feng, "A vlsi architecture for fast inversion in gf(2^m)," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1383–1386, 1989.

[14] C. C. Wang, T. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "Vlsi architectures for computing multiplications and inverses in gf(2^m)," *IEEE Transactions on Computers*, vol. 34, no. 8, pp. 709–719, 1985.

[15] K. Araki, I. Fujita, and M. Morisue, "Fast inverters over finite field based on euclidians algorithm," *IEICE Transactions of the Institute of Electronics Information and Communication Engineers*, vol. E-72, 1989.

[16] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in gf(2^m)," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 1010–1015, 1993.

[17] S. W. Wei, "Vlsi architectures for computing exponentiations, multiplicative inverses, and divisions in gf(2^m)," in *IEEE International Symposium on Circuits and Systems 1995, ISCAS '95*, pp. 203–206, 1995.

[18] J. H. Guo and C. L. Wang, "Hardware-efficient systolic architecture for inversion and division in gf(2^m)," *IEE Proceedings on Computers and Digital Techniques*, vol. 145, no. 4, pp. 272–278, 1998.

[19] Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in gf(2^m)," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1514–1519, 2003.

[20] C. L. Wang and J. H. Guo, "New systolic arrays for c + ab^2, inversion, and division in gf(2^m)," in *European Conference on Circuit Theory and Design 1995, ECCTD'95)*, pp. 431–434, 1995.

[21] C. L. Wang and J. L. Lin, "A systolic architecture for computing inverses and divisions in finite fields gf(2^m)," *IEEE Transactions on Computers*, vol. 42, no. 9, pp. 1141–1146, 1993.

[22] M. Hasan and V. Bhargava, "Bit-serial systolic divider and multiplier for finite fields gf(2^m)," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 972–980, 1992.

[23] S. T. Fenn, M. Benaissa, and D. Taylor, "Gf(2^m) multiplication and division over the dual basis," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 319–327, 1996.

[24] Z. Yan, D. V. Sarwate, and Z. Liu, "Hardware-efficient systolic architectures for inversions in gf(2^m)," *IEE Proceedings on Information Security*, vol. 152, no. 1, pp. 31–46, 2005.

[25] J. H. Guo and C. L. Wang, "Bit-serial systolic array implementation of euclids algorithm for inversion and division in gf(2^m)," in *Proceeding Technical Papers, Intl Symposum VLSI Technology, Systems, and Applications*, pp. 113–117, 1997.

[26] C. H. Kim, S. Kwon, C. P. Hong, and I. G. Nam, "Efficient bit-serial systolic array for division over gf(2^m)," in *In Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS'03*, pp. 252–255, 2003.

[27] A. K. Daneshbeh and M. A. Hasan, "A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over gf(2^m)," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 370–380, 2005.

[28] J. Fan, L. Batina, and I. Verbauwhede, "Design and design methods for unified multiplier and inverter and its application for hecc," *Integration, the VLSI Journal*, vol. 44, no. 4, pp. 280–289, 2011.

[29] D. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1981.

[30] Y. Asano, T. Itoh, and S. Tsujii, "Generalised fast algorithm for computing multiplicative inverses in gf(2^m)," *Electronics Letters*, vol. 25, no. 10, pp. 664–665, 1989.

[31] R. Brent and H. Kung, "Systolic vlsi arrays for polynomial gcd computation," *IEEE Transactions on Computers*, vol. 33, no. 8, pp. 731–736, 1984.

[32] Z. Yan, D. Sarwate, and Z. Lui, "High-speed systolic architectures for finite field inversion," *Integration, the VLSI Journal*, vol. 38, no. 3, pp. 383–389, 2005.

[33] G. Elias, A. Miri, and T. Yeap, "On efficient implementation of fpga-based hyperelliptic curve cryptosystems," *Computers and Electrical Engineering*, vol. 33, no. 5, pp. 349–366, 2007.

[34] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and K. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.