Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

# Hardware-Efficient Systolic Array Implementations of Euclid's Algorithm for Inversion and Division in $GF(2^m)$

*Jyh-Huei Guo and Chin-Liang Wang*
Department of Electrical Engineering, National Tsing Hua University
Hsinchu, Taiwan 300, Republic of China
clwang@ee.nthu.edu.tw

**Abstract**

*In this paper, four parallel-in parallel-out systolic arrays are proposed for computing inversion or division in finite fields $GF(2^m)$ based on new variants of Euclid's algorithm with the standard basis representation. Two of these arrays involve $O(m^2)$ area-complexity and $O(1)$ time-complexity. The other two involve $O(m)$ area-complexity and $O(m)$ time-complexity. They are highly regular, modular, and thus well suited to VLSI implementation. As compared to existing related systolic architectures: 1) the former two and the one in [14] have the same area and time complexities, but our proposed arrays involve less hardware area; 2) the latter two with $O(m)$ area-complexity gains a significant improvement in area complexity.*

## 1. Introduction

Finite fields $GF(2^m)$ have found many applications in areas of communications, such as error-correcting codes [1]-[2] and cryptography [3]. In these applications, computing inverses and divisions in $GF(2^m)$ is usually required. Since such computations are quite hardware-consuming, it is thus desirable to design hardware-efficient architectures for them.

There have been a number of hardware structures available for computing inverses and/or divisions in $GF(2^m)$ (see, for example, [4]-[14]). Among them, the designs in [4]-[7] can only be used to compute inverses in $GF(2^m)$. They are not systolic design and suffer from the problem of signal broadcasting. It should be noted that the signal broadcasting problem should be avoided in designing high-speed circuits. On the contrary, the architectures in [8]-[14] can be used to compute both inverses and divisions in $GF(2^m)$ and are designed based on the concepts of systolic array [15]. The area-time complexity is $O(m^3)$ for those in [8]-[12], $O(m \cdot 2^m)$ for that in [13], and $O(m^2)$ for that in [14]. It should be noted that the existing systolic architectures involve at least $O(m^2)$ area complexity which is still too large for some applications, such as cryptography, etc.

In this paper, four parallel-in parallel-out systolic arrays are proposed for computing inversion or division in $GF(2^m)$ based on new variants of Euclid's algorithm with the standard basis representation. Two of these arrays involve $O(m^2)$ area-complexity and $O(1)$ time-complexity. The other two involve $O(m)$ area-complexity and $O(m)$ time-complexity. They are highly regular, modular, and thus well suited to VLSI implementation. As compared to existing systolic designs for the same problems: 1) the former two and the circuit in [14] have the same area and area-time complexities of $O(m^2)$, but our proposed arrays involve less hardware area; 2) the latter two with $O(m)$ area-complexity are the two with the least area-complexity so far. Thus, they are quite suited to those applications with very large value of $m$, such as cryptography.

## 2. A new variant of Euclid's algorithm for inversion/division in $GF(2^m)$

Let $A(\alpha)$ and $B(\alpha)$ be two elements in $GF(2^m)$, $G(x)$ be the primitive polynomial of degree $m$, and $C(\alpha) = A(\alpha) / B(\alpha) \bmod G(\alpha)$. Then we have

$$A(\alpha) = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \cdots + a_0 \quad (1)$$

$$B(\alpha) = b_{m-1}\alpha^{m-1} + b_{m-2}\alpha^{m-2} + \cdots + b_0 \quad (2)$$

$$G(x) = x^m + g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_0 \quad (3)$$

$$C(\alpha) = c_{m-1}\alpha^{m-1} + c_{m-2}\alpha^{m-2} + \cdots + c_0 \quad (4)$$

$$B(\alpha)C(\alpha) + G(\alpha)D(\alpha) = A(\alpha) \quad (5)$$

for some element $D(\alpha)$ in $GF(2^m)$, where each coefficient of the polynomials is in $\{0, 1\}$. All arithmetic operations in $GF(2^m)$ are performed by taking the results mod 2, and $C(\alpha)$ is called the inverse of $B(\alpha)$ when $A(\alpha) = 1$.

### *A. The original Euclid's algorithm*

To perform inversion/division operations defined above, the following Euclid's algorithm [2] can be used:

$R = B(\alpha)$; $S = G(\alpha)$; $U = A(\alpha)$; $V = 0$;
**while** $R \neq 0$, **do**
  $Q = S$ DIV $R$; (* DIV: polynomial division *)
  temp = $S - Q \cdot R$; $S = R$; $R$ = temp;
  temp = $V - Q \cdot U$; $V = U$; $U$ = temp;
**end** (* $V = C(\alpha)$; $S = 1$ *)

One disadvantage of this algorithm is that it does not involve a fixed number of iterations for computing $C(\alpha)$ in a given field. This makes it not easily realized using VLSI techniques.

### *B. A new variant of Euclid's algorithm*

#### *B.1 A new algorithm for computing polynomial division with remainder*

The above algorithm is composed of the operations of polynomial division with remainder - "$Q = S$ DIV $R$", "temp = $S - Q \cdot R$; $S = R$; $R$ = temp", and "temp = $V - Q \cdot U$; $V = U$; $U$ = temp". Here, we propose a new algorithm for performing the operations efficiently.

*Polynomial division with remainder*

(* assume $m$ = deg $S$ and $d$ = deg $S$ – deg $R > 0$ *)
$T = 0$; state = 0; count = 0;
**for** $i = 1$ to $2d$ **do**
  $R = \alpha \cdot R$;
  **if** state = = 0 **then**
    count = count + 1;
    **if** $r_m$ = = 1 **then**
      tmp = $R$;
      $R = R + S$;
      $S$ = tmp; $T = U$;
      state = 1;
    **end**
  **else**
    $T = \alpha \cdot T$ mod $G$;
    count = count – 1;
    **if** $r_m$ = = 1 **then**
      $R = R + S$; $T = T + U$;
    **end**
    **if** count = = 0 **then**
      $V = T + V$; $U \leftrightarrow V$;
      state = 0;
    **end**
  **end**
**end** (* $Q = S$ DIV $R$;
  temp = $S - Q \cdot R$; $S = \alpha^d \cdot R$; $R = \alpha^d \cdot$ temp;
  temp = $V - Q \cdot U$; $V = U$; $U$ = temp *)

where $r_m$ denotes the coefficient of $\alpha^m$ of $R$. This algorithm consists of $2d$ iterations. Instead of having the correct answers, $R$ and $S$ are multiplied by $\alpha^d$. This will not matter since we are not interested in the values of $R$ and $S$. TABLE I gives an example of the proposed algorithm, where $m = 4$, $R = \alpha^2 + \alpha$, $S = \alpha^4 + \alpha^2 + 1$, $U = \alpha + 1$, $V = \alpha^2 + \alpha$, and $d$ = deg $S$ – deg $R = 2$. With this case, $Q = S$ DIV $R = \alpha^2 + \alpha$. After $2d = 4$ iterations, we have the results $R$, $S$, $U$, and $V$.

#### *B.2 A new algorithm for inversion/division in GF($2^m$)*

From Sections 2.A and 2.B.1, we observe that

(1) At the beginning and end of the original Euclid's algorithm, the degree of $S$ is $m$ and 0, respectively ( deg $G(\alpha) = m$ and deg 1 = 0). That is, the total degree reduction of $S$ is $m$.
(2) Each iteration of the original Euclid's algorithm will decrease the degree of $S$ by $d$, where $d$ = deg $S$ – deg $R$.
(3) Performing each iteration of the original Euclid's algorithm using the proposed algorithm in Section 2.B.1 needs $2d$ iterations.

From above observations, we see that performing the original Euclid's algorithm using the proposed algorithm in Section 2.B.1 needs a constant number of $2m$ iterations. Thus, we have the new algorithm for computing divisions in $GF(2^m)$ as follows:

*Algorithm for division in GF($2^m$)*

$R = B(\alpha)$; $S = G = G(\alpha)$; $U = A(\alpha)$; $V = T = 0$;
state = 0; count = 0;
**for** $i = 1$ to $2m$ **do**
  $R = \alpha \cdot R$;
  **if** state = = 0 **then**
    count = count + 1;
    **if** $r_m$ = = 1 **then**
      tmp = $R$;
      $R = R + S$;
      $S$ = tmp; $T = U$;
      state = 1;
    **end**
  **else**
    $T = \alpha \cdot T$ mod $G$;
    count = count – 1;
    **if** $r_m$ = = 1 **then**
      $R = R + S$; $T = T + U$;
    **end**
    **if** count = = 0 **then**
      $V = T + V$; $U \leftrightarrow V$;
      state = 0;
    **end**
  **end**
**end** (* $V = C(\alpha) = A(\alpha) / B(\alpha)$ mod $G(\alpha)$ *)

TABLE II gives an example of the new algorithm, where $m = 4$, $G(\alpha) = \alpha^4 + \alpha + 1$, $A(\alpha) = \alpha^3 + \alpha^2 + \alpha$, and $B(\alpha) = \alpha^3 + \alpha + 1$. From this table, we see that at the step $i = 2m = 8$, $V$ has the result $C(\alpha) = A(\alpha) / B(\alpha)$ mod $G(\alpha) = \alpha + 1$.

By setting $A(\alpha) = 1$, the above algorithm can be used to compute $C(\alpha) = 1 / B(\alpha)$ mod $G(\alpha)$, i.e., inverse of

$B(\alpha)$. TABLE III gives an example of computing inverse of $B(\alpha)$, where $m = 4$, $G(\alpha) = \alpha^4 + \alpha + 1$, $A(\alpha) = 1$, and $B(\alpha) = \alpha^3 + \alpha + 1$. At the step $i = 2m = 8$, $V$ has the result $C(\alpha) = 1 / B(\alpha) \bmod G(\alpha) = \alpha^2 + 1$.

### *C. A new algorithm for inversion/division in GF($2^m$)*

From the algorithm in Section 2.B.2, we find that

(1) "count = 1" and "state = 1" always occur at the beginning of the iteration $i = 2m$. During the iteration, $U$ is not modified and "$U \leftrightarrow V$" is performed. That is, after $2m - 1$ iterations, $U$ already contains the result $C(\alpha)$.

(2) At the beginning of the iteration $i = 2m - 2$, either "count = 0" and "state = 0" or "count = 2" and "state = 1" occur. Under both situations, $U$ will not be modified during the iteration. Thus, $U$ already contains the result $C(\alpha)$ after $2m - 2$ iterations.

(3) When the algorithm is used to compute inverse of $B(\alpha)$, deg $T$ increases gradually. It will increase to $m$ at the iteration $i = 2m$. Thus, "mod $G$" is needed to perform only at the last iteration.

From above, we can discard the last two iterations and re-write the division algorithm as follows:

*New algorithm for division in GF($2^m$)*

$R = B(\alpha)$; $S = G = G(\alpha)$; $U = A(\alpha)$; $V = T = 0$;
state = 0; count = 0;
**for** $i = 1$ to $2m - 2$ **do**
  $R = \alpha \cdot R$; $T = \alpha \cdot T \bmod G$;
  **if** state = = 0 **then**
    count = count + 1;
    **if** $r_m$ = = 1 **then**
      tmp = $R$; (I)
      $R = R + S$; (II)
      $S$ = tmp; $T = U$; (I)
      state = 1;
    **end**
  **else**
    count = count − 1;
    **if** $r_m$ = = 1 **then**
      $R = R + S$; $T = T + U$; (II)
    **end**
    **if** count = = 0 **then**
      $V = T + V$; $U \leftrightarrow V$; (III)
      state = 0;
    **end**
  **end**
**end** (* $U = C(\alpha) = A(\alpha) / B(\alpha) \bmod G(\alpha)$ *)

Besides, it should be noted that "mod $G$" operation is no more needed when the algorithm is used to compute inverse.

## 3. Key operations of the proposed algorithm

Before implementing the new variant of Euclid's algorithm, we demonstrate how to compute its key operations: "$R = \alpha \cdot R$" and "$T = \alpha \cdot T \bmod G$". Since $R$ and $T$ are polynomials with degrees at most $m$ and $m - 1$ respectively, they can be expressed as follows:

$$R = r_m \alpha^m + r_{m-1} \alpha^{m-1} + \cdots + r_0 \tag{6}$$

$$T = t_{m-1} \alpha^{m-1} + t_{m-2} \alpha^{m-2} + \cdots + t_0 \tag{7}$$

Since $G(\alpha) = 0$, we have

$$\alpha^m = g_{m-1} \alpha^{m-1} + g_{m-2} \alpha^{m-2} + \cdots + g_0 \tag{8}$$

### *A. "$R = \alpha \cdot R$" operation*

Assume that

$$\begin{aligned} R' &\equiv r'_m \alpha^m + r'_{m-1} \alpha^{m-1} + \cdots + r'_0 \\ &= \alpha \cdot R \\ &= r_m \alpha^{m+1} + r_{m-1} \alpha^m + \cdots + r_0 \alpha \end{aligned} \tag{9}$$

Since $r_m = 0$ when the statement "$R = \alpha \cdot R$" is executed, the corresponding result can be reduced as

$$R' = r_{m-1} \alpha^m + r_{m-2} \alpha^{m-1} + \cdots + r_0 \alpha \tag{10}$$

Comparing (9) and (10), we have

$$r'_0 = 0 \tag{11}$$

$$r'_i = r_{i-1},\ 1 \le i \le m \tag{12}$$

It is interesting to see that if $R$ is expressed as an $(m + 1)$-bit word, the result of "$R = \alpha \cdot R$" can be obtained simply by shifting left the input data bits one position, where the rightmost bit is set to be zero and the leftmost bit is neglected.

### *B. "$T = \alpha \cdot T \bmod G$" operation*

Let

$$\begin{aligned} T' &\equiv t'_{m-1} \alpha^{m-1} + t'_{m-2} \alpha^{m-2} + \cdots + t'_0 \\ &= \alpha \cdot T \bmod G \\ &= t_{m-1} \alpha^m + t_{m-2} \alpha^{m-1} + \cdots + t_0 \alpha \bmod G \end{aligned} \tag{13}$$

Substituting (8) into (13) yields

$$\begin{aligned} T' &= t_{m-1}(g_{m-1} \alpha^{m-1} + g_{m-2} \alpha^{m-2} + \cdots + g_0) \\ &\quad + t_{m-2} \alpha^{m-1} + \cdots + t_0 \alpha \\ &= (t_{m-1} g_{m-1} + t_{m-2}) \alpha^{m-1} + (t_{m-1} g_{m-2} + t_{m-3}) \alpha^{m-2} \\ &\quad + \cdots + (t_{m-1} g_1 + t_0) \alpha + t_{m-1} g_0 \end{aligned} \tag{14}$$

Thus, the operation "$T = \alpha \cdot T \bmod G$" can be performed based on the following two equations:

$$t'_0 = t_{m-1} g_0 \tag{15}$$

$$t'_i = t_{m-1} g_i + t_{i-1},\ 1 \le i \le m - 1 \tag{16}$$

## 4. Hardware-efficient systolic array implementations of the proposed algorithm

### *A. Signal flow graph of the proposed division algorithm*

Fig. 1 shows a signal flow graph (SFG) array for implementing the proposed division algorithm in $GF(2^m)$, where $m = 3$. This array consists of $2m - 2$ Type-1 cells and $(2m - 2) \times m$ Type-2 cells. The functions of these two types of basic cells are illustrated in Figs. 2-3. The $i$th row of this array realizes the $i$th iteration. The Type-1 cell generates the control signals Ctrl1, Ctrl2, and Ctrl3 for the present iteration as well as computes the values of count and state for the next iteration (i.e., count' and state' in Fig. 2). The corresponding functions are given as follows:

$$\text{Ctrl1} = (\text{state} == 0)\ \&\ (r_m == 1) \tag{17}$$

$$\text{Ctrl2} = (r_m == 1) \tag{18}$$

$$\text{Ctrl3} = (\text{state} == 1)\ \&\ (\text{count} == 0) \tag{19}$$

$$\text{count}' = \begin{cases} \text{count} + 1, & \text{if state} == 0 \\ \text{count} - 1, & \text{if state} == 1 \end{cases} \tag{20}$$

$$\text{state} = \overline{\text{state}},\ \text{if} \begin{cases} ((r_m == 1)\ \&\ (\text{state} == 0)) \\ \text{or}\ ((\text{count} == 0)\ \&\ (\text{state} == 1)) \end{cases} \tag{21}$$

When the control signals Ctrl1, Ctrl2, and Ctrl3 are true, Type-2 cells in the corresponding row execute the operations of (I), (II), and (III), respectively. Otherwise, they skip the operations. The operations "+" and "$\leftrightarrow$" can be easily realized using XOR gates and multiplexers, respectively. The key operations "$R = \alpha \cdot R$" and "$T = \alpha \cdot T \bmod G$" always need to be executed at every iteration and are realized according to the methods described in Section 3. The $(i + 1)$th Type-2 cell from the right evaluates the $(i + 1)$th least significant coefficients of $R$, $S$, $U$, $V$, and $T$ (i.e., $r_i'$, $s_i'$, $u_i'$, $v_i'$, and $t_i'$ in Fig. 3), where $0 \le i \le m - 1$. Besides, it is noted that the coefficient $s_m$ needs not to be processed since it always equals to one. After $2m - 2$ iterations, the coefficients of result $C(\alpha)$ will emerges from the $(2m - 2)$th row of the array.

### *B. Systolic array implementations of the proposed division algorithm*

By applying the cut-set systolization techniques [16] to the SFG array in Fig. 1, a parallel-in parallel-out systolic array with maximum throughput rate can be derived. Fig. 4 shows the result, where "•" denotes 1-cycle delay element. This array can provide one division result at a rate of one every clock cycle after an initial delay of $5m - 4$ clock cycles.

The SFG array in Fig. 1 is in fact a dependence graph (DG) [17]. By projecting the DG along the south direction [17] to get a one-dimensional signal flow graph (SFG) array and applying the cut-set systolization techniques [16] to the corresponding SFG array, a parallel-in parallel-out systolic array with utilization efficiency of 50% can be derived. Fig. 5 shows the results, where "*" means "don't care". This array is controlled by a sequence $0*1*1*\cdots1*$ of length $4m - 4$, and is composed of 1 Type-3 cell and $m$ Type-4 cells. The functions of these two types of cells are illustrated in Figs. 6 and 7. This array performs one iteration every other clock cycle. After $5m - 4$ clock cycles, the coefficients of the result $C(\alpha)$ emerge from the bottom of this array in parallel form. The utilization efficiency of this array can be improved to 100% by interleaving the input data as shown in Fig. 8, where $b_i^*$'s, $g_i^*$'s, and $a_i^*$'s are the other input data. This array can produce two results every $4m - 4$ clock cycles (i.e. an average throughput rate of producing one result per $2(m - 1)$ clock cycles).

### *C. Systolic array implementations of the proposed inversion algorithm*

From Section 2.C, the difference between the proposed division and inversion algorithms is that the proposed inversion algorithm need not to perform the "mod $G$" operation. Thus, the arrays in Figs. 4 and 8 can be simplified when they are used to compute inverses in $GF(2^m)$. Figs. 9-10 and 11-12 show the corresponding simplified arrays and basic cells, respectively. The two simplified arrays in Figs. 9 and 10 have the same throughput performances and latency as those of the two arrays in Figs. 4 and 8, respectively.

## 5. Conclusions

In this paper, we have presented four parallel-in parallel-out systolic arrays for computing inverses or divisions in finite fields $GF(2^m)$ over the standard basis. They are highly regular, modular, and thus well suited to VLSI implementation. TABLE IV gives a comparison of the proposed arrays with those related in [8]-[12] and [14]. From this table, we can see that 1) the proposed arrays in Figs. 8 and 10 reach the least area-complexity of $O(m)$; 2) the array in [14] and the two in Figs. 4 and 9 have the same throughput performance and area-complexity. But according to the static CMOS transistor count estimation [18], the transistor counts of the arrays in Figs. 4 and 9 are about 66% and 53% of that of the array in [14], respectively. The above comparisons support that our arrays are very attractive for use in applications with large value of $m$, such as cryptography.

## References

[1] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA: MIT Press, 1972.

[2] E. R. Berlekamp, *Algebraic Coding Theory*. New York: Mcgraw-Hill, 1968.

[3] D. E. R. Denning, *Cryptography and Data Security*. Reading, MA: Addsion-Wesley, 1983.

[4] C. C. Wang, T. K. 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 Trans. Comput.*, vol. 34, pp. 709-719, Aug. 1985.

[5] G.-L. Feng, "A VLSI architecture for fast inversion in $GF(2^m)$," *IEEE Trans. Comput.*, vol. 38, pp. 1383-1386, Oct. 1989.

[6] K. Araki, I. Fujita, and M. Morisue, "Fast inverters over finite field based on Euclid's algorithm," *Trans. IEICE*, vol. E-72, pp. 1230-1234, Nov. 1989.

[7] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in $GF(2^m)$," *IEEE Trans. Comput.*, vol. 42, pp. 1010-1015, Aug. 1993.

[8] C.-L. Wang and J.-L. Lin, "A systolic architecture for computing inverses and divisions in finite fields $GF(2^m)$," *IEEE Trans. Comput.*, vol. 42, pp. 1141-1146, Sep. 1993.

[9] M. A. Hasan and V. K. Bhargava, " Bit-level systolic divider and multiplier for finite fields $GF(2^m)$," *IEEE Trans. Comput.*, vol. 41, pp. 972-980, Aug. 1992.

[10] S. T. J. Fenn, M. Benaissa, and D. Taylor, "$GF(2^m)$ multiplication and division over the dual basis", *IEEE Trans. Comput.*, vol. 45, pp. 319-327, Mar. 1996.

[11] S.-W. Wei, "VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF(2^m)$," in *Proc. 1995 IEEE Int. Symp. Circuits Syst.*, London, May 1995, pp. 4.203-4.206.

[12] C.-L. Wang and J.-H. Guo, "New systolic arrays for $C + AB^2$, inversion, and division in $GF(2^m)$," in *Proc. 1995 European Conference Circuit Theory Design*, Istanbul, Turkey, Aug. 1995, pp. 431-434.

[13] M. Kovac, N. Ranganathan and M. Varanasi, "SIGMA: A VLSI systolic array implementation of a Galois field $GF(2^m)$ based multiplication and division algorithm," *IEEE Trans. VLSI Systems*, vol. 1, pp. 22-30, Mar. 1993.

[14] J.-H. Guo and C.-L. Wang, "Systolic array implementation of Euclid's algorithm for inversion and division in $GF(2^m)$," in *Proc. 1996 IEEE Int. Symp. Circuits Syst.*, Atlanta, May 1996, pp. II.481-II.484.

[15] H. T. Kung, "Why systolic architectures?," *Computer*, vol. 15, pp. 37-46, Jan. 1982.

[16] S. Y. Kung, "One supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, pp. 867-884, July 1984.

[17] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[18] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*: A System Perspective. Reading, MA: Addison-Wesley, 1985.

**TABLE I**
**An example of our proposed algorithm for computing polynomial division with remainder**
($m = 4$, $R = \alpha^2 + \alpha$, $S = \alpha^4 + \alpha^2 + 1$, $U = \alpha + 1$, $V = \alpha^2 + \alpha$)

| $i$ | count | state | $R$ | $S$ | $U$ | $V$ | $T$ |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | $\alpha^2 + \alpha$ | $\alpha^4 + \alpha^2 + 1$ | $\alpha + 1$ | $\alpha^2 + \alpha$ | 0 |
| 1 | 1 | 0 | $\alpha^3 + \alpha^2$ | $\alpha^4 + \alpha^2 + 1$ | $\alpha + 1$ | $\alpha^2 + \alpha$ | 0 |
| 2 | 2 | 1 | $\alpha^3 + \alpha^2 + 1$ | $\alpha^4 + \alpha^3$ | $\alpha + 1$ | $\alpha^2 + \alpha$ | $\alpha + 1$ |
| 3 | 1 | 1 | $\alpha$ | $\alpha^4 + \alpha^3$ | $\alpha + 1$ | $\alpha^2 + \alpha$ | $\alpha^2 + 1$ |
| 4 | 0 | 0 | $\alpha^2$ | $\alpha^4 + \alpha^3$ | $\alpha^3 + \alpha^2$ | $\alpha + 1$ | $\alpha^3 + \alpha$ |

**TABLE II**
**An example of our proposed algorithm for computing divisions in $GF(2^4)$**
$(G(\alpha) = \alpha^4 + \alpha + 1, A(\alpha) = \alpha^3 + \alpha^2 + \alpha, B(\alpha) = \alpha^3 + \alpha + 1)$

| $i$ | count | state | $R$ | $S$ | $U$ | $V$ | $T$ |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | $\alpha^3 + \alpha + 1$ | $\alpha^4 + \alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha$ | 0 | 0 |
| 1 | 1 | 1 | $\alpha^2 + 1$ | $\alpha^4 + \alpha^2 + \alpha$ | $\alpha^3 + \alpha^2 + \alpha$ | 0 | $\alpha^3 + \alpha^2 + \alpha$ |
| 2 | 0 | 0 | $\alpha^3 + \alpha$ | $\alpha^4 + \alpha^2 + \alpha$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha$ | $\alpha^3 + \alpha^2 + \alpha + 1$ |
| 3 | 1 | 1 | $\alpha$ | $\alpha^4 + \alpha^2$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha$ | $\alpha^3 + \alpha^2 + \alpha + 1$ |
| 4 | 0 | 0 | $\alpha^2$ | $\alpha^4 + \alpha^2$ | $\alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^3 + \alpha^2 + 1$ |
| 5 | 1 | 0 | $\alpha^3$ | $\alpha^4 + \alpha^2$ | $\alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^3 + \alpha^2 + 1$ |
| 6 | 2 | 1 | $\alpha^2$ | $\alpha^4$ | $\alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha + 1$ |
| 7 | 1 | 1 | $\alpha^3$ | $\alpha^4$ | $\alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^2 + \alpha$ |
| 8 | 0 | 0 | 0 | $\alpha^4$ | 0 | $\alpha + 1$ | $\alpha^3 + \alpha^2 + \alpha + 1$ |

**TABLE III**
**An example of our proposed algorithm for computing inversion in $GF(2^4)$**
$(G(\alpha) = \alpha^4 + \alpha + 1, A(\alpha) = 1, B(\alpha) = \alpha^3 + \alpha + 1)$

| $i$ | count | state | $R$ | $S$ | $U$ | $V$ | $T$ |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | $\alpha^3 + \alpha + 1$ | $\alpha^4 + \alpha + 1$ | 1 | 0 | 0 |
| 1 | 1 | 1 | $\alpha^2 + 1$ | $\alpha^4 + \alpha^2 + \alpha$ | 1 | 0 | 1 |
| 2 | 0 | 0 | $\alpha^3 + \alpha$ | $\alpha^4 + \alpha^2 + \alpha$ | $\alpha$ | 1 | $\alpha$ |
| 3 | 1 | 1 | $\alpha$ | $\alpha^4 + \alpha^2$ | $\alpha$ | 1 | $\alpha$ |
| 4 | 0 | 0 | $\alpha^2$ | $\alpha^4 + \alpha^2$ | $\alpha^2 + 1$ | $\alpha$ | $\alpha^2$ |
| 5 | 1 | 0 | $\alpha^3$ | $\alpha^4 + \alpha^2$ | $\alpha^2 + 1$ | $\alpha$ | $\alpha^2$ |
| 6 | 2 | 1 | $\alpha^2$ | $\alpha^4$ | $\alpha^2 + 1$ | $\alpha$ | $\alpha^2 + 1$ |
| 7 | 1 | 1 | $\alpha^3$ | $\alpha^4$ | $\alpha^2 + 1$ | $\alpha$ | $\alpha^3 + \alpha$ |
| 8 | 0 | 0 | 0 | $\alpha^4$ | 0 | $\alpha^2 + 1$ | $\alpha$ |

**TABLE IV**
**Comparison of existing systolic arrays and the proposed systolic arrays for inversion/division in $GF(2^m)$**

| Item \ Circuits | [8]-[10] | [11]&[12] | [14] | Figs. 4 and 9 | Figs. 8 and 10 |
|---|---|---|---|---|---|
| Throughput (1/cycles) | $1/(2m-1)$ [8]<br>$1/m$ [9], [10] | 1 | 1 | 1 | $1/2(m-1)$ |
| Latency (cycles) | $7m-3$ [8]<br>$5m-1$ [9], [10] | $3m^2-2m$ [11]<br>$2m^2-3m/2$ [12] | $8m-1$ | $5m-4$ | $5m-4$ |
| Area complexity | $O(m^2)$ | $O(m^3)$ | $O(m^2)$ | $O(m^2)$ | $O(m)$ |
| Area-time product | $O(m^3)$ | $O(m^3)$ | $O(m^2)$ | $O(m^2)$ | $O(m^2)$ |
| I/O format | Serial-in Serial-out | Parallel-in Parallel-out | Parallel-in Parallel-out | Parallel-in Parallel-out | Parallel-in Parallel-out |
| # control signal | 2 [8], [10]<br>1 [9] | 0 | 0 | 0 | 1 |
| Operation | Division | Division | Division | Division Fig. 4<br>Inversion Fig. 9 | Division Fig. 8<br>Inversion Fig. 10 |

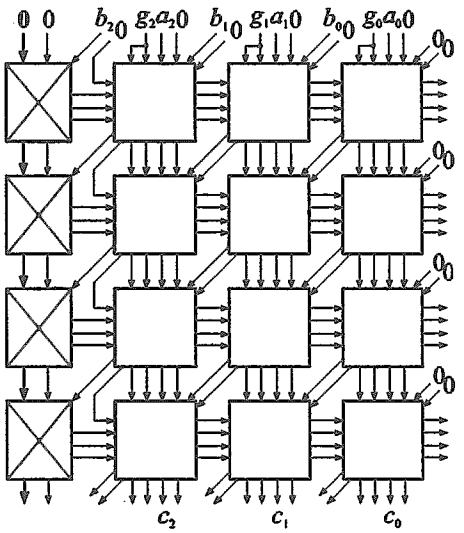


Fig. 1. Signal flow graph for the proposed division algorithm in $GF(2^m)$, where $m = 3$.

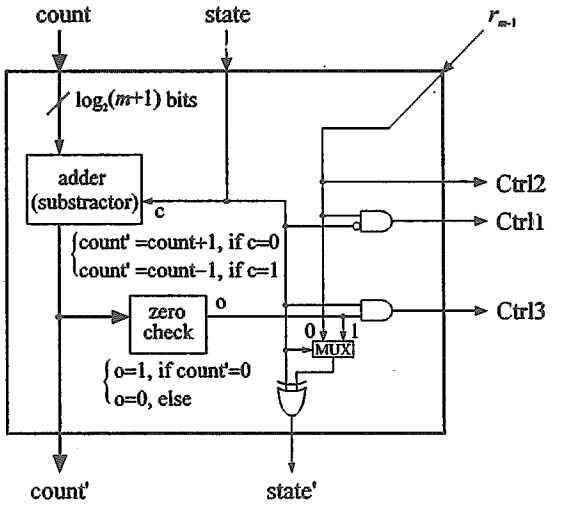


Fig. 2. The circuit of Type-1 cell in Fig. 1.

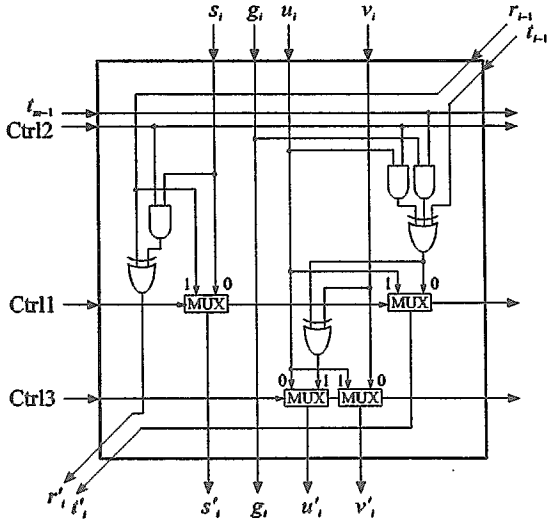


Fig. 3. The circuit of Type-2 cell in Fig. 1.

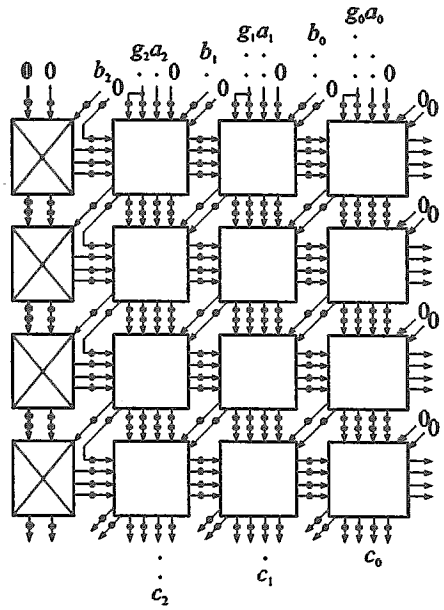


Fig. 4. Parallel-in Parallel-out systolic array for division in $GF(2^3)$ with the maximum throughput rate.

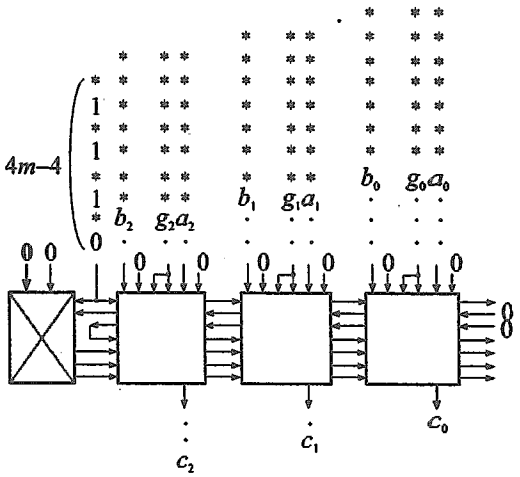


Fig. 5. Parallel-in Parallel-out systolic array for division in $GF(2^3)$ with utilization efficiency of 50%.

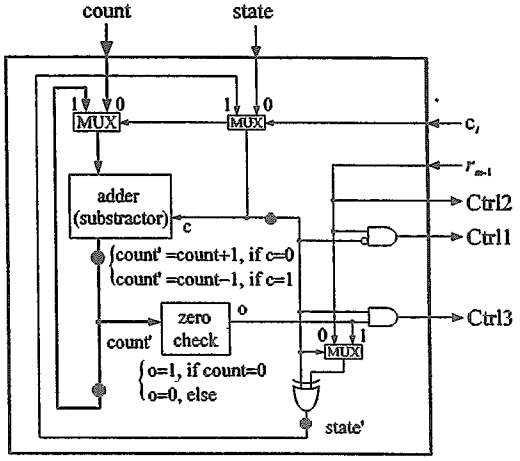


Fig. 6. The circuit of Type-3 cell in Fig. 5.

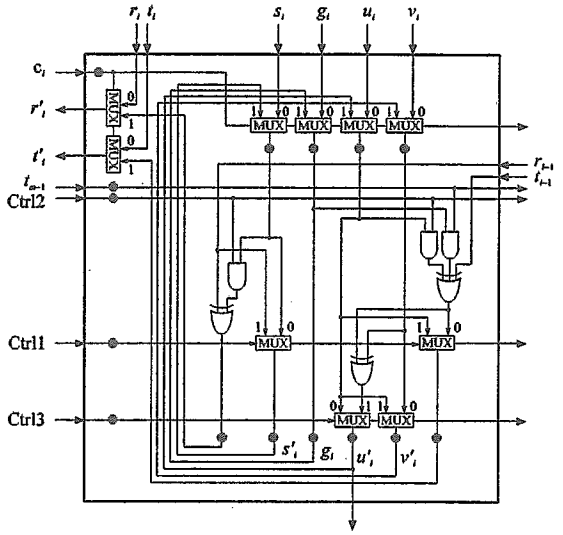


Fig. 7. The circuit of Type-4 cell in Fig. 5.

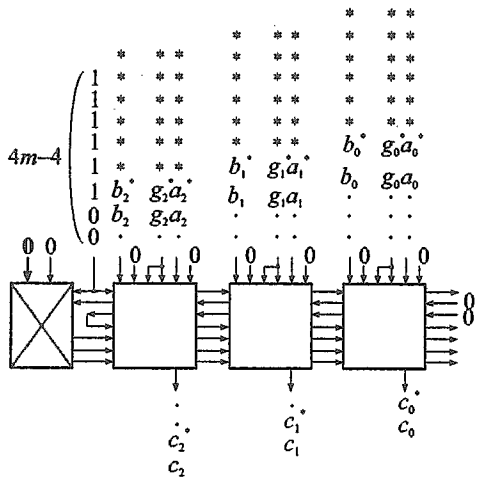


Fig. 8. Improving the utilization efficiency to 100% by interleaving the input data.

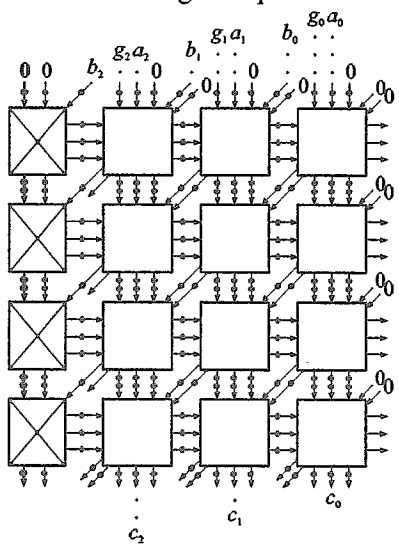

Fig. 9. Parallel-in Parallel-out systolic array for inversion in $GF(2^3)$ with maximum throughput rate.

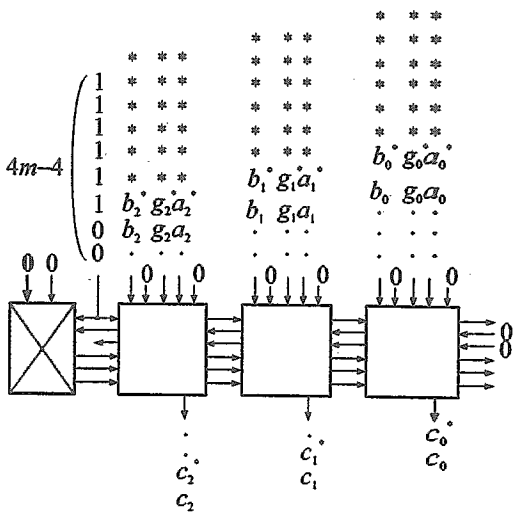


Fig. 10. Parallel-in Parallel-out systolic array for inversion in $GF(2^3)$ with utilization efficiency of 100%.

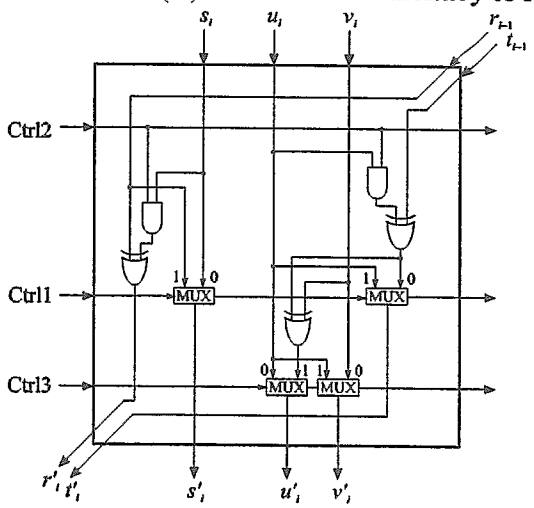


Fig. 11. The circuit of Type-5 cell in Fig. 9.

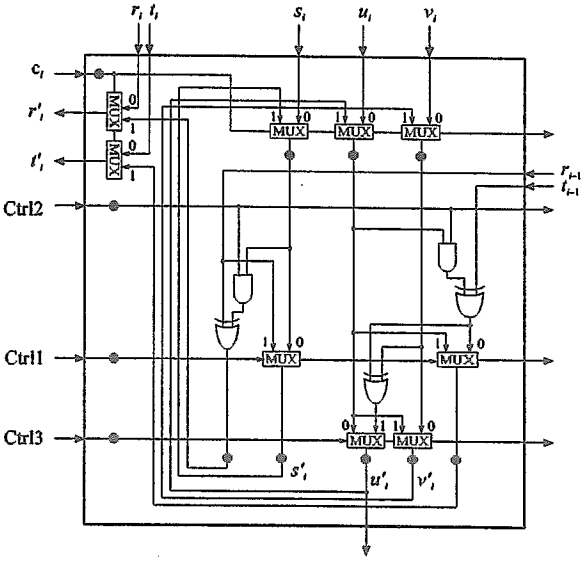


Fig. 12. The circuit of Type-6 cell in Fig. 12.