SYSTOLIC ARRAY IMPLEMENTATION 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 – **This paper presents a new systolic VLSI architecture for computing inverses and divisions in finite fields $GF(2^m)$ based on a variant of Euclid's algorithm. It is highly regular, modular, and thus well suited to VLSI implementation. It has $O(m^2)$ area complexity and can produce one result per clock cycle with a latency of $8m$-2 clock cycles. As compared to existing related systolic architectures with the same throughput performance, the proposed one gains a significant improvement in area complexity.**

I. 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 time-consuming, it is desirable to design high-speed circuits for them to meet the real-time requirements.

There have been a number of hardware structures available for fast inversion and/or division in $GF(2^m)$ (see, for example, [4]-[12]). Among them, the designs in [4]-[9] employ serial-form input and/or serial-form output. Basically, such circuits with serial-form data transmission involve small hardware complexity but might have unsatisfactory throughput performance when m gets large. In contrast, the designs in [10]-[12] make use of parallel-in parallel-out I/O and achieve higher throughput rates with more hardware complexity. All these parallel architectures are designed based on the concept of systolic arrays [13] and are able to provide the maximum throughput in the sense of producing new results at a rate of one per clock cycle, i.e., the time complexity is O(1). However, their hardware requirements seem too high for large fields; the area complexity is $O(m*2^m)$ for the circuit in [10] and is $O(m^3)$ for those in [11] and [12].

In this paper, a new parallel-in parallel-out systolic array with O(1) time complexity and $O(m^2)$ area complexity is proposed for inversion and division in $GF(2^m)$. The architecture is designed based on a variant of Euclid's algorithm for computing the greatest common divisor of two polynomials. It is highly regular, modular, and thus well suited to VLSI implementation. As compared to previous systolic architectures for inversion and division with the same throughput performance, the proposed one saves a significant amount of chip area.

This work was supported by the National Science Council of the Republic of China under Grant NSC-85-2215-E-007-017.

II. VARIANTS OF EUCLID'S ALGORITHM FOR COMPUTING INVERSIONS AND DIVISIONS IN $GF(2^m)$

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

$$A(\alpha)=a_{m-1}\alpha^{m-1}+\cdots+a_1\alpha+a_0 \quad (1)$$
$$B(\alpha)=b_{m-1}\alpha^{m-1}+\cdots+b_1\alpha+b_0 \quad (2)$$
$$G(\alpha)=\alpha^m+g_{m-1}\alpha^{m-1}+\cdots+g_1\alpha+g_0 \quad (3)$$
$$C(\alpha)=c_{m-1}\alpha^{m-1}+\cdots+c_1\alpha+c_0 \quad (4)$$
$$B(\alpha)C(\alpha)+G(\alpha)D(\alpha)=A(\alpha) \quad (5)$$

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
$U = C(\alpha)$.
```

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. The Modified Euclid's Algorithm in [9]

To overcome the above-mentioned problem, Brunner et al. [9] proposed a modification of Euclid's algorithm that always involves $2m$ iterations to compute an inverse or division in $GF(2^m)$. The algorithm can be summarized as follows:

```
$R = B(\alpha)$; $S = G = G(\alpha)$; $U = A(\alpha)$; $V = 0$;
count = 0;
for $i = 1$ to $2m$ do
    if $r_m == 0$ then (*occurring $m$ times*)
        $R = \alpha \cdot R$; $U = \alpha \cdot U \bmod G$;
        count = count + 1;
    else
        if $s_m == 1$ then
            $S = S + R$; $V = V + U$;
        end
        $S = \alpha \cdot S$;
        if count = = 0 then
            $R \leftrightarrow S$; $U \leftrightarrow V$; (*exchange operations*)
            $U = \alpha \cdot U \bmod G$;
            count = count + 1;
        else (*occurring $m$ times*)
            $U = U/\alpha \bmod G$;
            count = count - 1;
        end
    end
end (*$U=C(\alpha) = A(\alpha)/B(\alpha) \bmod G(\alpha)$; count = 0*)
```

where r_m and s_m denote the most significant coefficients of R and S, respectively. This algorithm involves $2m$ iterations (i=1 to $2m$) and "count = 0" always occurs at the end of the last iteration [9]. In other words, both of the statements "count = count + 1" and "count = count - 1" run m times during the $2m$ iterations. To realize the algorithm, a serial-in serial-out pipelined architecture was given in [9]. This circuit possesses good area-time performance, but it is not a systolic design and suffers from broadcasting problems. The reason why the algorithm is not amenable to systolic array implementation is that its arithmetic operations are not uniform during the $2m$ iterations. It performs "$U = \alpha \cdot U \bmod G$" for some iterations, and performs "$U = U/\alpha \bmod G$" for the others.

C. A New Variant of Euclid's Algorithm

Note that, if the statement "$U = U/\alpha \bmod G$" is removed from the algorithm given above, the final result will become $U = C(\alpha)\alpha^m$, instead of the desired answer $U = C(\alpha)$. To obtain the correct answer, we can execute the operation "$U = U/\alpha \bmod G$" m times after the $2m$ iterations have been completed. It can also be seen from the statements "temp = $V - Q \cdot U$; $V = U$, U = temp" given in Section II.A that removing the statement "$U = U/\alpha \bmod G$" is equivalent to executing the statement "$V = \alpha \cdot V \bmod G$". Moreover, the statements "$R \leftrightarrow S$; $U \leftrightarrow V$; $U = \alpha \cdot U \bmod G$" are equivalent to "$V = \alpha \cdot V \bmod G$; $R \leftrightarrow S$; $U \leftrightarrow V$". With these observations, we can derive the following algorithm for inversion/division in $GF(2^m)$:

```
    $R = B(\alpha)$; $S = G = G(\alpha)$; $U = A(\alpha)$; $V = 0$;
    count = 0;
Part A:
    for $i = 1$ to $2m$ do
        if $r_m == 0$ then
            $R = \alpha \cdot R$; $U = \alpha \cdot U \bmod G$;          (I)
            count = count + 1;
        else
            if $s_m == 1$ then
                $S = S + R$; $V = V + U$;                   (II)
            end
            $S = \alpha \cdot S$; $V = \alpha \cdot V \bmod G$;             (III)
            if count = = 0 then
                $R \leftrightarrow S$; $U \leftrightarrow V$;                     (IV)
                count = count + 1;
            else
                count = count - 1;
            end
        end
    end (*$U = C(\alpha) \cdot \alpha^m \bmod G(\alpha)$.; count = 0*)
Part B:
    for $i = 2m+1$ to $3m$ do
        $U = U/\alpha \bmod G$;
    end (*$U = C(\alpha) = A(\alpha)/B(\alpha) \bmod G(\alpha)$*)
```

Apparently, the new variant of Euclid's algorithm consists of two parts; Part A first generates a temporary result $C(\alpha) \cdot \alpha^m \bmod G(\alpha)$., and then Part B divides it by α^m to yield the correct answer. Table I demonstrates a procedure of the proposed algorithm for computing inverses/divisions in $GF(2^4)$, where $G(\alpha) = \alpha^4+\alpha+1$, $A(\alpha) = \alpha^3+\alpha^2+\alpha$, and $B(\alpha) = \alpha^3+\alpha+1$. At step $i = 2m = 8$, $U = \alpha^2+1$ is the temporary result $C(\alpha) \cdot \alpha^m \bmod G(\alpha)$., and at step $i = 3m = 12$, $U = \alpha+1$ is the correct answer $C(\alpha) = A(\alpha)/B(\alpha) \bmod G(\alpha)$. As compared to the algorithm described in Section II.B, the new algorithm involves more uniform arithmetic operations during the recursively computing process, and is thus easier to realize using a systolic architecture.

III. SYSTOLIC IMPLEMENTATION OF THE PROPOSED ALGORITHM

Fig. 1 shows a systolic architecture to implement the proposed algorithm for computing inverses and divisions in $GF(2^m)$, where '•' denotes a one-cycle delay element. It consists of a subarray of $2m$ Type-I cells and $2m \times m$ Type-II cells for realizing the Part-A operations and a subarray of $m \times m$ Type-III cells for realizing the Part-B operations. The ith row of each subarray performs the ith-iteration operations of the corresponding part. The functions of these three types of basic cells are illustrated in Figs. 2 to Fig. 4.

TABLE I
An Example of Computing Inverses/Divisions in $GF(2^4)$ Based on the Proposed Algorithm
($G(\alpha)=\alpha^4+\alpha+1$, $A(\alpha)=\alpha^3+\alpha^2+\alpha$, $B(\alpha)=\alpha^3+\alpha+1$)

i	count	R	S	U	V
	0	$\alpha^3+\alpha+1$	$\alpha^4+\alpha+1$	$\alpha^3+\alpha^2+\alpha$	0
1	1	$\alpha^4+\alpha^2+\alpha$	$\alpha^4+\alpha+1$	$\alpha^3+\alpha^2+\alpha+1$	0
2	0	$\alpha^4+\alpha^2+\alpha$	$\alpha^3+\alpha$	$\alpha^3+\alpha^2+\alpha+1$	$\alpha^3+\alpha^2+1$
3	1	$\alpha^4+\alpha^2$	$\alpha^4+\alpha^2+\alpha$	α^3+1	$\alpha^3+\alpha^2+\alpha+1$
4	0	$\alpha^4+\alpha^2$	α^2	α^3+1	$\alpha^3+\alpha^2$
5	1	α^3	$\alpha^4+\alpha^2$	$\alpha^3+\alpha+1$	α^3+1
6	2	α^4	$\alpha^4+\alpha^2$	α^2+1	α^3+1
7	1	α^4	α^3	α^2+1	$\alpha^3+\alpha+1$
8	0	α^4	α^4	α^2+1	α^2+1
9				$\alpha^3+\alpha+1$	
10				$\alpha^3+\alpha^2$	
11				$\alpha^2+\alpha$	
12				$\alpha+1$	

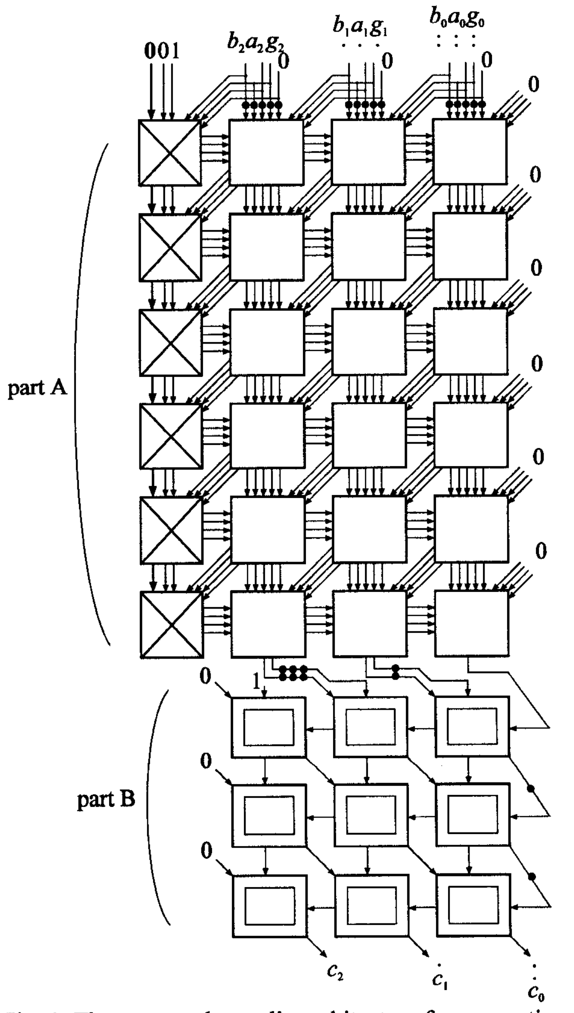


Fig. 1. The proposed systolic architecture for computing inversions/divisions in $GF(2^m)$. m=3.

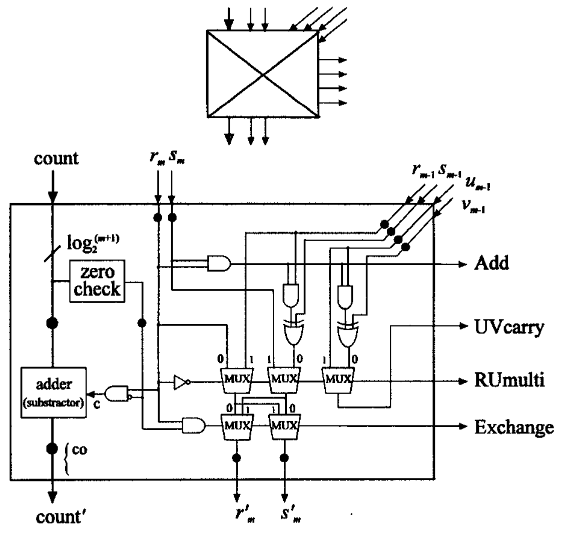
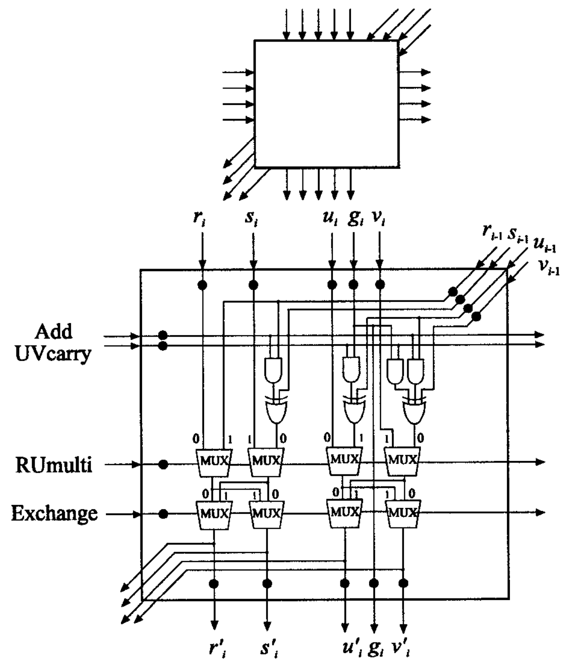


Fig. 2. The circuit of Type-I cell in Fig. 1.



r_i s_i u_i g_i v_i
r_{i-1} s_{i-1} u_{i-1} v_{i-1}
Add
UVcarry
RUmulti
Exchange
MUX
r'_i s'_i u'_i g_i v'_i

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

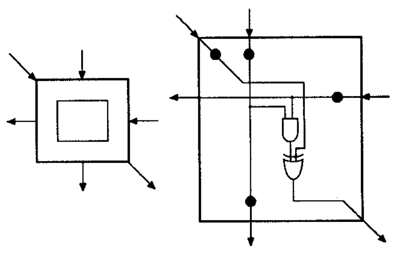


Fig. 4. The circuit of Type-III cell in Fig. 1.

Each Type-I cell is used to generate the following control signals:

RUmulti = (r_m = = 0)
Add = (r_m = =1) & (s_m = = 1)
Exchange = (r_m = = 1) & (count = = 0)
count′ = count - 1, if (count $\neq$ 0) & (r_m ==1)
count′ = count + 1, else

When RUmulti = 1, the corresponding row of Type-2 cells executes the operations given in (I); otherwise, it does the operations of (III). The Add and Exchange signals are used to determine whether the operations of (II) and (IV) are performed or skipped. The Part-A subarray generates the temporary result $C(\alpha)\cdot\alpha^m$ mod G at its bottom row, and then sends it to the Part-B subarray for further processing. With little effort, one can check the inversion/division results will emerge from the bottom of the Part-B subarray at a rate of one per clock cycle. It can also be seen that the proposed systolic architecture has area complexity of O(m^2) and a latency of 8m-2 clock cycles.

IV. CONCLUSIONS

Table II gives a comparison of the proposed parallel-in parallel-out systolic array for inversion and division in $GF(2^m)$ with those in [11] and [12]. We can see from this table that all the architectures compared reach the same throughput rate of one result per clock cycle, but the proposed one has much smaller area requirement, much shorter latency, and much better area-time product performance.

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. C-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] 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.

[7] 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.

[8] 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.

[9] 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.

[10]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.

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

TABLE II
Comparison of Some Parallel-In Parallel-Out Systolic Arrays for Computing Inversions/Divisions in $GF(2^m)$

Item \ Circuits	Wei [11]	Wang & Guo [12]	Proposed
Number of Cells	$m^2(m-1)$	$m^2(m-1)/2$	Type I: $2m$ Type II: $2m^2$ Type III: m^2
Throughput (1/cycle)	1	1	1
Latency (cycles)	$3m^2-2m$	$2m^2-3m/2$	$8m-2$
Maximum Cell Delay	T_{AND2} $+T_{XOR3}$	T_{AND2} $+T_{XOR4}$	$T_{AND2}+T_{XOR3}+2T_{MUX2}$
Cell Complexity	3 AND$_2$'s 1 XOR$_2$ 1 XOR$_3$ 13 latches	6 AND$_2$'s 2 XOR$_4$'s 17 latches	**Type I:** 5 AND$_2$'s 2 XOR$_2$'s 5 MUX$_2$'s 1 INV $\log_2(m+1)$ bits adder zero-check circuit $9+2\log_2(m+1)$ latches **Type II:** 4 AND$_2$'s 2 XOR$_2$'s 1 XOR$_3$ 8 MUX$_2$'s 18 latches **Type III:** 1 AND$_2$ 1 XOR$_2$ 4 latches
AT-product	O(m^3)	O(m^3)	O(m^2)

AND$_i$: i-input AND gate; XOR$_i$: i-input XOR gate.
INV : inverter; MUX$_i$: i-input multiplexer.
T_{ANDi} : the propagation delay through an i-input AND gate.
T_{XORi} : the propagation delay through an i-input XOR gate.
T_{MUXi} : the propagation delay through an i-input multiplexer.