JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 953-967 (2011)

Design of High-Speed Iterative Dividers in GF(2^m)*

MING-DER SHIEH, WEN-CHING LIN AND CHIEN-MING WU+
Department of Electrical Engineering
National Cheng Kung University
Tainan, 701 Taiwan
+*Chip Implementation Center*
National Applied Research Laboratories
Hsinchu, 300 Taiwan

Fast algorithms for high-speed divider design in finite fields GF(2^m) are very crucial in applications like cryptosystems. In this paper, we reformulated the conventional iterative division algorithm by changing the pre-defined variable and then updating its initial value accordingly. The reformulated division algorithm allows a restructuring of the divider architecture to further improve its operating speed without increasing latency or area cost. Using the proposed fast algorithm, we developed two high-speed iterative dividers based on the semi-systolic and bit-serial systolic architectures. Analytical results show that the cost of the initial value update and variable transformation in the reformulated algorithm is almost negligible in the hardware implementation. Our divider designs improve the critical path delay. Compared with related divider designs, the proposed designs have time and area advantages.

Keywords: division algorithm, finite field, high-speed, Stein's algorithm, systolic array

1. INTRODUCTION

The theory of finite fields plays a very important role in many practical applications, such as error control codes [1] and public-key cryptosystems [2]. Of the basic arithmetic operations in finite fields GF(2^m), the addition/subtraction is simply a bit-wise exclusive-or (XOR) operation, and the multiplication can be efficiently performed based on Horner's rule [3, 4]. The modular division algorithm, however, is relatively complicated and deserves special design effort.

Division algorithms over GF(2^m) can be classified into four types based on the underlying equations or algorithms: (1) Fermat's Theorem [5], (2) discrete-time Wiener-Hopft equation [6], (3) Euclid's algorithm [7-14], and (4) Stein's algorithm [15-19]. Among them, the extended Euclid and Stein algorithms have attracted a lot of attention because dividers designed based on these two algorithms can achieve lower area-time (*AT*) complexity ($O(m^2)$) [8-14, 16-19]. In terms of hardware realization, different schemes have been proposed to reduce the critical path delay. For example, the idea [20] of introducing a new variable δ to represent the difference of upper bounds of polynomials has been applied to relax the time-consuming operation of comparing degrees of polynomials. The operating speed can be further improved by replacing the 2's-complement counter with a one-hot coded shift register for large values of *m* [11-13, 16, 18]. Alternatively, a division opera-

Received October 6, 2009; revised February 26, 2010; accepted April 20, 2010.
Communicated by Chung-Ping Chung.
* This work was supported in part by the National Science Council of Taiwan, R.O.C. under contract No. NSC 96-2220-E-006-008.

tion can also be accomplished by using an inversion followed by a multiplication [12-14]. The main concerns of employing a direct divider design or an inversion-multiplication based design are the trade-offs among the operating speed, area, latency, and throughput rate. Intuitively, the inversion-multiplication based design can provide higher operating speed at the price of increasing the hardware cost and latency.

This paper explores techniques to reduce the critical path delay of conventional divider designs in GF(2^m) based on either Stein's or Euclid's algorithms without sacrificing area or latency. For ease of explanation, we focus on improving the operating speed of dividers employing Stein's algorithm. For modular division based on Stein's algorithm, two auxiliary variables, often denoted by U and V, are used with the dividend S and divisor R to determine the value of (S/R) mod G, where G is an irreducible polynomial. The variables U and V are updated in a way similar to update S and R in each iteration except for the reduction operation on U with respect to G. Thus, the critical path delay of conventional dividers is often dominated by the updating of variable U.

By changing the pre-defined variable U in conventional iterative division algorithms and then updating its initial value accordingly, we can relax the data dependency that exists between determining whether or not V is added to U and performing the reduction operation on the newly updated U. The reformulated division algorithm thus provides the opportunity of restructuring the divider architecture to further improve its operating speed. In fact, such a transformation or change on a variable can be viewed as a kind of pre-processing step. Note that different techniques may be adopted in different applications to achieve a hardware-efficient design; *e.g.*, a pre-process that transforms a real number into Montgomery's domain to simplify modular operations [21]. The main concern will be the cost of carrying out the pre-processing steps.

Using the reformulated iterative division algorithm, we developed two array architectures, the bit-serial and semi-systolic divider architectures, for very high-speed operation. Analytical results show that our development yields lower time and area complexities than other divider designs [8-11, 17-19], and can operate as fast as the most-significant-bit first (MSB-first) multiplier [3, 4]. The cost of the initial value update and variable transformation in the reformulated algorithm is almost negligible in the hardware implementation. Using algorithm manipulation, our design reveals an obvious area advantage over Kim's work [19] designed based on retiming technique. Although the proposed bit-serial divider design cannot be operated as fast as the inversion-multiplication design in [14], it has area, latency, and area-time (AT) advantages.

The rest of this paper is organized as follows. Section 2 describes the proposed iterative division algorithm. Section 3 presents the corresponding semi-systolic and bit-serial divider architectures. Section 4 shows the performance analysis and comparison with the related work. Finally, there are our concluding remarks about this work.

2. MODIFYING STEIN'S ALGORITHM FOR HIGH-SPEED DESIGN

2.1 Preliminary

Let $G(x)$ be an irreducible polynomial in GF(2^m) expressed as:

$$G(x) = \sum_{j=0}^{m} g_j x^j, \tag{1}$$

where $g_0 = g_m = 1$ and $g_j \in \{0, 1\}$ for $j = 1, \ldots, m - 1$, or an equivalent vector form $G = (g_m, \ldots, g_1, g_0)$. Any element $A(x) \in \text{GF}(2^m)$ can then be uniquely represented as a vector $A = (a_{m-1}, \ldots, a_1, a_0)$ or in the polynomial form:

$$A(x) = \sum_{j=0}^{m-1} a_j x^j, \tag{2}$$

where $a_j \in \{0, 1\}$ for $j = 0, \ldots, m - 1$.

In the following, we review the division algorithm [18] to obtain $V(x) = (A(x)/B(x))_G$ based on the extended Stein's algorithm, where $(C(x))_G$ denotes the operation $C(x)$ mod $G(x)$. Notably, in Algorithm *DA*_1, (1) i is used to count the number of iterations, which implies that the division algorithm converges in $2m - 1$ iterations; (2) r_0 denotes the least significant bit (LSB) of the polynomial $R(x)$; (3) δ represents the difference of upper bounds on deg(R) and deg(S), where deg(C) denotes the degree of $C(x)$; (4) $R(x)/x$ indicates the right shift operation, $R(x)/x \equiv (0, r_{m-1}, \ldots, r_1)$, which decreases the degree of $R(x)$ by one.

Algorithm *DA*_1: Division Algorithm [18]
Initialization: $(R(x), S(x), U(x), V(x), \delta) \leftarrow (B(x), G(x), A(x), 0, -1)$
Result: $V(x) \equiv (A(x)/B(x))_G$
Algorithm: for $i = 1 : 2m - 1$
  if $r_0 = 1$
    if $\delta < 0$
      $(R(x), S(x), U(x), V(x)) \leftarrow (R(x) + S(x), R(x), U(x) + V(x), U(x))$
      $\delta \leftarrow -\delta$
    else
      $(R(x), U(x)) \leftarrow (R(x) + S(x), U(x) + V(x))$
  $(R(x), U(x)) \leftarrow (R(x)/x, (U(x)/x)_G)$
  $\delta \leftarrow \delta - 1$

2.2 Proposed Division Algorithm

From Algorithm *DA*_1, it can be observed that the resulting critical path delay is dominated by computing either δ^{i+1} or $U^{i+1}(x)$, where the superscript denotes the iteration. Different ways of speeding up the computation of δ^{i+1}, such as the high-speed counter design and the shift-register-based design [11-13, 16, 18] for large m, have been presented in the literature. As a result, the critical path delay becomes dominated by computing $U^{i+1}(x)$, *i.e.*, performing the operation $U^{i+1}(x) = ((U^i(x) + r_0^i \cdot V^i(x))/x)_G$. This paper aims at relaxing the data dependency existing on updating $U^{i+1}(x)$ and, therefore, offers an opportunity to further increase the operating speed of existing dividers.

Iterative update of $U(x)$ involves sequential execution of the following two statements: (1) if $r_0 = 1$, $U(x) = U(x) + V(x)$; otherwise $U(x)$ is unchanged, and (2) if $u_0 = 1$, $U(x) = (U(x) + G(x))/x$; otherwise $U(x) = U(x)/x$, where u_0 denotes the LSB of $U(x)$. Thus, the value of u_0 in statement (2) is dependent on the result of performing the operation in statement (1). This, in turn, increases the resulting critical path delay. To estimate the critical path delay,

we rewrite the recurrent equation for updating $U(x)$ as follows,

$$\begin{aligned} U^{i+1}(x) &= ((U^i(x) + r_0^i \cdot V^i(x))/x)_G \\ &= ((U^i(x) + r_0^i \cdot V^i(x) + (u_0^i + r_0^i \cdot v_0^i) \cdot G(x))/x). \end{aligned} \tag{3}$$

Assuming that the two operations, $U^i(x) + r_0^i \cdot V^i(x)$ and $u_0^i + r_0^i \cdot v_0^i$, are executed concurrently, the critical path delay of Eq. (3) can be estimated as $2T_{AND} + 2T_{XOR}$, where T_{AND} and T_{XOR} denote the delay times of 2-input AND and XOR gates, respectively. In the following, we show how to relax the data dependency existing between the two statements for reducing the critical path delay.

Making the change of variable $W(x) = (U(x)x)_G$, equivalently $U(x) = (W(x)/x)_G$, we can rewrite Eq. (3) as:

$$(W^{i+1}(x)/x)_G = (((W^i(x)/x)_G + r_0^i \cdot V^i(x))/x)_G. \tag{4}$$

Therefore,

$$\begin{aligned} W^{i+1}(x) &= (W^i(x)/x)_G + r_0^i \cdot V^i(x) = (W^i(x) + w_0^i \cdot G(x))/x + r_0^i \cdot V^i(x) \\ &= W^i(x)/x + w_0^i \cdot (G(x)/x) + r_0^i \cdot V^i(x) \end{aligned} \tag{5}$$

where w_0^i denotes the LSB of $W(x)$ at the ith iteration. According to Eq. (5), the resulting critical path delay is now reduced to $T_{AND} + 2T_{XOR}$. Similarly, substituting $(W(x)/x)_G$ for $U(x)$, we obtain the following iterative equations:

$$(R^{i+1}(x), W^{i+1}(x)) \leftarrow ((R^i(x) + r_0^i \cdot S^i(x))/x, (W^i(x)/x)_G + r_0^i \cdot V^i(x)), \tag{6}$$

$$(S^{i+1}(x), V^{i+1}(x)) \leftarrow (\overline{\eta} \cdot S^i(x) + \eta \cdot R^i(x), \overline{\eta} \cdot V^i(x) + \eta \cdot (W^i(x)/x)_G), \tag{7}$$

where the controlling variable η is asserted when $r_0^i = 1$ and $\delta^i < 0$. Note that the superscript will be removed hereafter whenever no confusion occurs.

To check the time delay needed to update the value of $V^{i+1}(x)$, we re-express the iterative equation of $V^{i+1}(x)$ as:

$$V^{i+1}(x) = \overline{\eta} \cdot V^i(x) + \eta \cdot (W^i(x)/x)_G = \overline{\eta} \cdot V^i(x) + \eta \cdot ((W^i(x) + w_0^i \cdot G(x))/x). \tag{8}$$

From Eq. (8), the resulting path delay of updating $V^{i+1}(x)$ can be estimated as $T_{AND} + T_{XOR} + T_{MUX}$, where T_{MUX} denotes the delay of 2-to-1 multiplexers. Therefore, the critical path delay of the divider is determined by $\max\{T_{AND} + 2T_{XOR}, T_{AND} + T_{XOR} + T_{MUX}\}$. Note that we assume the controlling circuit does not affect the final critical path delay as mentioned previously. Because T_{MUX} is smaller than T_{XOR} when using the TSMC 0.18 μm process, the critical path delay of the divider is then dominated by updating $W^{i+1}(x)$ and expressed as $T_{AND} + 2T_{XOR}$.

From Eqs. (6) and (7), we observe that the operation $(W^i(x)/x)_G$ is now computed before updating W^{i+1} and V^{i+1}. Therefore, we reformulate Algorithm *DA*_1 and derive a fast division algorithm, named Algorithm *DA*_2, to further improve the operating speed of iterative dividers.

Algorithm *DA*_2: The Developed Iterative Division Algorithm
Initialization: $(R(x), S(x), W(x), V(x), \delta) \leftarrow (B(x), G(x), (A(x)x)_G, 0, -1)$
Result: $V(x) \equiv (A(x)/B(x))_G$
Algorithm: for $i = 1 : 2m - 1$

$W(x) \leftarrow (W(x)/x)_G$
if $r_0 = 1$
  if $\delta < 0$
    $(R(x), S(x), W(x), V(x)) \leftarrow (R(x) + S(x), R(x), W(x) + V(x), W(x))$
    $\delta \leftarrow -\delta$
  else
    $(R(x), W(x)) \leftarrow (R(x) + S(x), W(x) + V(x))$
$R(x) \leftarrow R(x)/x$
$\delta \leftarrow \delta - 1$

Compared to traditional division algorithms based on the extended Stein algorithm [17, 18], the underlying operations for $W(x)$ and $U(x)$ are very similar. $V(x)$ is added to both if r_0 is 1 and they are both divided by x mod $G(x)$. However, for $U(x)$, the addition must be performed before the division and thus data dependency exists. In contrast, for $W(x)$, the two operations can be performed in parallel. As a result, the proposed division algorithm has higher parallelism than those in [17, 18].

Note that the shifter-register-based designs [11-13, 16, 18] should be applied for high-speed divider designs when the value of m increases. Because $W(x) = (U(x)x)_G$, the initial value of $W(x)$ is preset to $(A(x)x)_G$. This implies that there is a need to pre-compute the value of $A(x)x + a_{m-1} \cdot G(x)$ in conventional design methods. To reduce design complexity, we can set $W(x)$ to be an $(m + 1)$-bit vector with its value initialized as $A(x) \cdot x = (a_{m-1}, a_{m-2}, \ldots, a_0, 0)$. In fact, the effect of multiplying $A(x)$ by x will be cancelled by the following operation $W(x)/x$ at the beginning of the first iteration. Therefore, the cost of removing the required pre-computation of $(A(x)x)_G$ is just a 1-bit register.

Table 1 compiles an example of modular division in GF(2^m) using the developed Algorithm *DA*_2 assuming that $G(x) = x^4 + x + 1$, $A(x) = x^3 + x^2 + 1$, and $B(x) = x^2 + x + 1$, *i.e.*, $G = (10011)$, $A = (1101)$ and $B = (0111)$. As can be seen in Table 1, Algorithm *DA*_2 takes seven iterations to find the solution $V = (A/B)_G = (1000)$, which corresponds to $V(x) = x^3$. Note that the second row, $i = 0$, contains the initial values.

Table 1. An example for the developed algorithm *DA*_2.

i	δ	R	S	W	V
0	– 1	0111	1 0011	1 1010	0000
1	0	1010	0 0111	0 1101	1101
2	– 1	0101	0 0111	0 1111	1101
3	0	0001	0 0101	0 0011	1110
4	– 1	0010	0 0101	0 0110	1110
5	– 2	0001	0 0101	0 0011	1110
6	1	0010	0 0001	0 0110	1000
7	0	0001	0 0001	0 0011	1000

3. DEVELOPED ITERATIVE DIVIDERS

Starting from the dependence graph (DG) of the systolic array implementation of the proposed Algorithm *DA*_2, we describe the underlying base cell designs and then show the mapped semi-systolic and bit-serial array architectures. The semi-systolic design is more area-efficient than the bit-serial version, while the bit-serial design can operate faster than the semi-systolic one.

3.1 Base Cell Design

Fig. 1 (a) shows the two-dimensional DG array of the proposed algorithm with $m = 3$. Generally speaking, the array consists of $2m - 1$ rows corresponding to the number of iterations. Each row has $m + 1$ cells including one A-cell, $(m - 1)$ B-cells, and one C-cell. The iterative division operations are performed row by row, *i.e.*, the ith row for the ith iteration, with the division result $V(x)$ available at the bottom row after $2m - 1$ iterations. For clarity, we define three controlling signals $Ctrl2 \equiv r_0^i$, $Ctrl3 \equiv w_0^i$, and $Ctrl4 \equiv \eta = (r_0^i = 1)$ & $(\delta^i < 0)$, and rewrite Eqs. (6) and (7) for carrying out Algorithm *DA*_2 as follows,

$$R^{i+1}(x) \leftarrow (R^i(x) + Ctrl2 \cdot S^i(x))/x, \tag{9}$$

$$W^{i+1}(x) \leftarrow (W^i(x) + Ctrl3 \cdot G(x))/x + Ctrl2 \cdot V^i(x), \tag{10}$$

$$S^{i+1}(x) \leftarrow \overline{Ctrl4} \cdot S^i(x) + Ctrl4 \cdot R^i(x), \tag{11}$$

$$V^{i+1}(x) \leftarrow \overline{Ctrl4} \cdot V^i(x) + Ctrl4 \cdot (W^i(x) + Ctrl3 \cdot G(x))/x, \tag{12}$$

$$\delta^{i+1} \leftarrow \overline{Ctrl4} \cdot (\delta^i - 1) + Ctrl4 \cdot (-\delta^i - 1). \tag{13}$$

Note that the addition is a bit-wise XOR operation, $R(x)/x \equiv (0, r_{m-1}, \ldots, r_1)$ is a right shift operation that decreases the degree of $R(x)$ by one, and the operation $(W^i(x)/x)_G \equiv (W^i(x) + w_0^i \cdot G(x))/x$ is to perform $w_j^i \leftarrow w_{j+1}^i + w_0^i \cdot g_{j+1}$ for $0 \le j \le m - 1$, where $w_m^i = 0$.

The A-cell in Fig. 1 (c) works as the right boundary cell for updating the LSB's of $W(x)$ and $V(x)$, and generates the three controlling signals *Ctrl*2, *Ctrl*3, and *Ctrl*4. The B-cell in Fig. 1 (b) is employed to implement the operations defined in Eqs. (9)-(12), excluding the LSB's of the four variables and MSB's of $S(x)$ and $R(x)$. The C-cell in Fig. 1 (d) is a simplified B-cell used to compute the MSB's of $S(x)$ and $R(x)$; *i.e.*, s_m^{i+1} and r_{m-1}^{i+1}. Finally, Eq. (13) is generally carried out by the D-cell depicted in Fig. 1 (e). For high-speed operation with large m, the traditional 2's-complement counter in Fig. 1 (e) is replaced by a one-hot $(m + 1)$-bit counter, also referred to as an $(m + 1)$-bit left/right (L/R) shifter [18], as depicted in Fig. 1 (f) to increase its operating speed.

An $(m + 1)$-bit L/R shifter is derived by introducing a pair of variables (d, f) and defining $d = 2^{|\delta|} = 2^{\delta(-1)^f}$, where $f \in \{0, 1\}$ and $d \in \{1, 2, 2^2, \ldots, 2^m\}$ [18]. Using the (d, f) notation, the conditions $\delta = 0$, $\delta > 0$, and $\delta < 0$ can then be respectively represented by $(d_0, f) = (1, 0)$, $(0, 0)$, and $(0, 1)$, where d_0 denotes the LSB of d, and f is the sign of δ. Note that since the combination $(d_0, f) = (1, 1)$ never occurs, we can use $f = 1$ to denote $\delta < 0$ to simplify the controller design [13]. When substituting (d, f) into the proposed Algorithm *DA*_2, we can replace the conditional expressions with the following analogies: (1) when $\delta < 0$, the operation $\delta \leftarrow -\delta$ can be simply described by setting $f = 0$; (2) the operation $\delta \leftarrow \delta - 1$

can be performed by setting $d \leftarrow d/2$ if $(d_0, f) = (0, 0)$, and $(d, f) \leftarrow (d \cdot 2, 1)$ otherwise. Thus, the original 2's-complement counter can be replaced by an $(m + 1)$-bit L/R shifter as shown in Fig. 2.

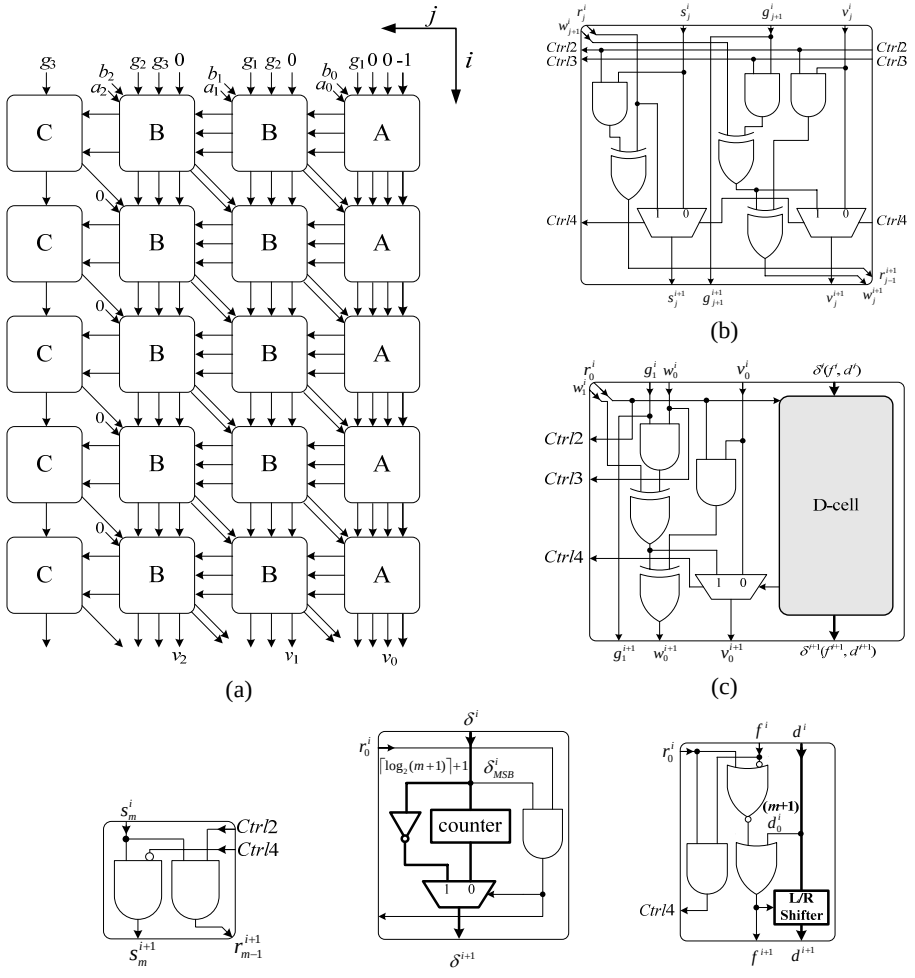

Fig. 1. (a) Dependence graph of algorithm *DA*_2; (b) B-cell; (c) A-cell; (d) C-cell; (e) D-cell using the traditional counter; (f) D-cell using the L/R shifter.

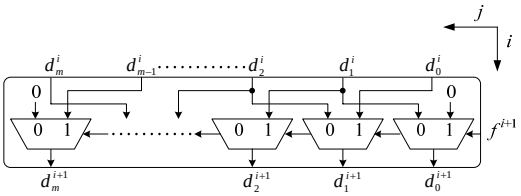

Fig. 2. L/R shifter (one-hot counter) design.

3.2 Semi-Systolic Array Design

From the DG shown in Fig. 1 (a), a semi-systolic architecture can be derived by choosing the projection vector $\mathbf{D} = [i, j] = [1, 0]^T$ and the scheduling vector $\mathbf{S} = [1, 0]^T$ [22]. Fig. 3 (a) illustrates the resulting circuit design in which v_j^i, $0 \le j \le m - 1$, represents the output at the ith iteration. The three base cells, A*-cell, B*-cell, and C*-cell, are modified from the corresponding A-cell, B-cell, and C-cell in Fig. 1, and the array divider can output the final result after $2m$ clock cycles with one extra cycle to load initial values. Note that the gray multiplexers in Figs. 3 (b), (c), and (d) are used to select the input data. That is, the array divider will access input data with $Ctrl1 = 0$ in the first cycle (iteration) and latch the intermediate results with $Ctrl1 = 1$ in the remaining cycles. The small black rectangular represents a one-cycle delay element or a flip-flop.

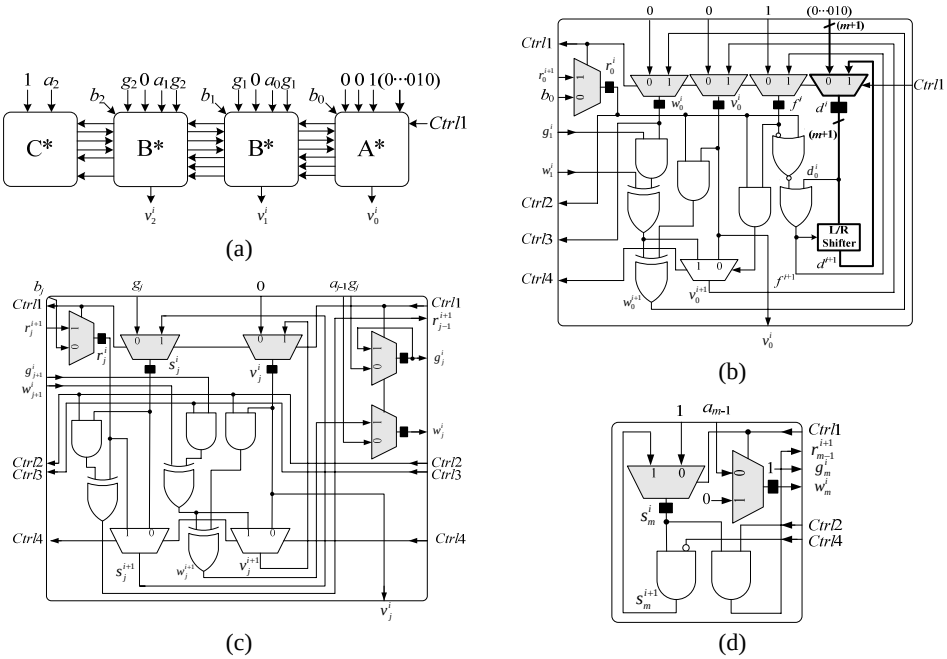

Fig. 3. (a) The derived semi-systolic array divider; (b) A*-cell; (c) B*-cell; (d) C*-cell.

3.3 Bit-Serial Systolic Array Design

The bit-serial architecture can be implemented by choosing the projection vector $\mathbf{D} = [0, 1]^T$ and the scheduling vector $\mathbf{S} = [2, 1]^T$ for the DG in Fig. 1 (a). However, directly mapping from Fig. 1 (a) using $\mathbf{D} = [0, 1]^T$ will result in $O(m)$ counters in the mapped linear array because of the counter in the original A-cell. To solve this problem, we employed the L/R shifter design, used in Algorithm *EBdf* [18], and uniformly distributed the content of the shifter, as done in [13], into those cells in a row before performing array mapping on the DG. In this manner, the area complexity of the required counter can be reduced from $O(m^2)$ to $O(m)$.

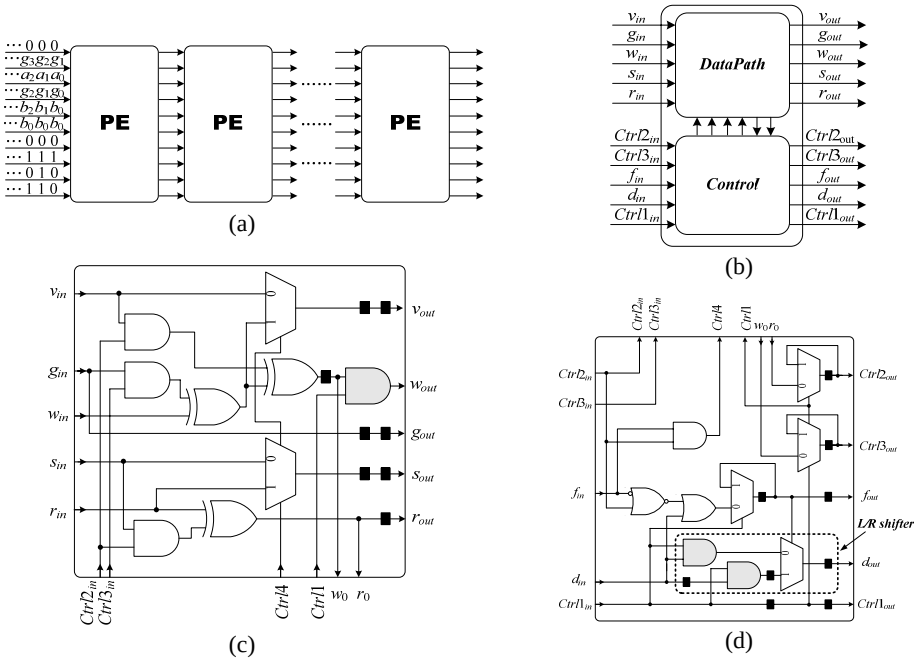

Fig. 4. (a) Bit-serial systolic array divider; (b) Inputs/outputs of PE; (c) Datapath of PE; (d) Control circuit of PE.

Fig. 4 (a) shows the mapped bit-serial array divider made up of $2m - 1$ identical processing elements (PEs), each consisting of two distinct components (the datapath and control circuit) as shown in Figs. 4 (b), (c), and (d). The datapath is used to update $R(x)$, $S(x)$, $W(x)$, and $V(x)$, while the control circuit is to generate or bypass the required controlling signals for the current PE and the following PE. Note that the gray AND gates in Figs. 4 (c) and (d) are employed to ensure that two consecutive or back-to-back division operations can be performed correctly. These added AND gates do not increase the critical path delay of the divider.

The sequence of the controlling signals shown on the left-hand side of Fig. 4 (a) is as follows: (1) the *Ctrl*1 signal consists of a zero followed by m ones with the leading zero used to initialize the divider and to sample the values of r_0, w_0, and f at the beginning of a division operation; (2) since $\delta = -1$ initially, we have $f = 1$ and $d = (00\ldots010)$; (3) *Ctrl*2 and *Ctrl*3 are signals defined in the base cell design in Fig. 1. Because $\mathbf{S} = [2, 1]^T$, the latency of our bit-serial design is $5m - 3$, consisting of $2(2m - 1)$ cycles for *Ctrl*1 to pass through the array and $(m - 1)$ cycles for the last PE to complete its task. The throughput of our bit-serial design is one division per $m + 1$ cycles because two consecutive division operations can be processed seamlessly.

4. PERFORMANCE ANALYSIS AND IMPLEMENTATION RESULTS

In this section, we first compare the proposed semi-systolic array design with the related work in the literature. Table 2 lists the area requirement and performance of our semi-

systolic array design and the work based on extended Euclid's algorithm [8] and Stein's algorithm [17, 18], respectively. To have a fair comparison, we make the following assumptions: (1) each high fan-in gate is expressed in terms of the equivalent 2-input gates for area estimation; (2) the multiplexers used to select between the initial values and intermediate results are included in our comparison; (3) dedicated flip-flops (FFs) are employed to store the irreducible polynomial; (4) the gate count of the control circuit is not included in our comparisons since the related work [8, 17] did not show their control circuit design. In Table 2, the symbols T_{AND}, T_{XOR}, and T_{MUX} denote the delays of 2-input AND gates, XOR gates, and multiplexers (MUXs), respectively.

Table 2. Comparison of different semi-systolic array architectures.

		[8]	[17, 18]	Ours_S (Fig. 3)
Time Complexity		$O(m)$	$O(m)$	$O(m)$
Throughput		$1/2m$	$1/2m$	$1/2m$
Latency		$2m$	$2m$	$2m$
Critical path delay*		$3T_{AND} + 2T_{XOR} + 3T_{MUX}$	$2T_{AND} + 3T_{XOR} + T_{MUX}$	$T_{AND} + 2T_{XOR} + T_{MUX}$
Area Complexity		$O(m)$	$O(m)$	$O(m)$
Gate count	FF*	$5m$	$5m$	$5m$
	XOR	$3m$	$3m$	$3m$
	AND	$3m$	$3m$	$3m$
	MUX*	$13m$	$7m$	$7m$

* Including the effects of assumptions (2) and (3) above.

Analytical results reveal that our semi-systolic array design can operate faster than the related designs [8, 17, 18] without increasing the resulting latency. The area cost of our design is as low as the previous work [17, 18]. Brunner's work [8] has the highest complexity because of the complicated data flow. Instead of using the variable U in conventional Stein's algorithm [17, 18], the proposed algorithm uses redefined variable W to reduce the resulting critical path delay. The semi-systolic array (Ours_S) based on our proposed Algorithm *DA*_2 is not only high-speed but also area-efficient. Moreover, the developed semi-systolic array comprises three internal global signals (*Ctrl*2, *Ctrl*3, and *Ctrl*4) and one external global signal (*Ctrl*1 for input selection). The number of global signals is the same as the result in [17, 18] and is one less than that in [8].

Table 3 shows the area requirements and critical path delays of the proposed bit-serial array design and the related work [10, 11, 14, 19]. Note the symbol T_{XOR3} denotes the delay of a 3-input XOR gate. In [9], Guo modified Brunner's inversion algorithm [8] by postponing the operation $(U(x)/x)_G$ to implement a bit-serial systolic array structure. However, the non-uniform cell structure and the 2's-complement adder/subtractor result in high area complexity. Although the improved unidirectional bit-serial version [10] used distributed degree tracking control to reduce the area complexity from $O(m \log_2 m)$ to $O(m)$, the hardware requirement is still higher than that of our design because of the extra register required to store the temporary T variable. Daneshbeh [11] proposed a unidirectional bit-serial systolic array with the ring-counter structure, which is the best bit-serial systolic divider design, to our knowledge, based on the extended Euclid algorithm. In [19], Kim presented a bit-serial divider based on the extended Stein algorithm, improving its critical

Table 3. Comparison of different bit-serial systolic array architectures.

		[10]	[11]	[19]	[14] + [23]	Ours_B (Fig. 4)
Time Complexity		$O(m)$	$O(m)$	$O(m)$	$O(m)$	$O(m)$
Throughput		$1/m$	$1/(m + 1)$	$1/m$	$1/(m + 1)$	$1/(m + 1)$
Latency		$5m - 4$	$5m - 4$	$5m - 2$	$7m - 2$	$5m - 3$
Critical path delay		$T_{AND} + T_{XOR} + T_{XOR3} + T_{MUX}$*	$2T_{AND} + T_{XOR} + T_{MUX}$	$T_{AND} + 2T_{XOR}$	$T_{AND} + T_{XOR}$	$T_{AND} + 2T_{XOR}$
Area Complexity		$O(m)$	$O(m)$	$O(m)$	$O(m)$	$O(m)$
Gate count	FF	$44m - 44$	$36m - 18$	$40m - 19$	$50m$**	$34m - 17$
	MUX	$22m - 22$	$20m - 10$	$14m - 7$	$14m$	$12m - 6$
	XOR*	$10m - 10$	$6m - 3$	$8m - 4$	$6m$	$6m - 3$
	AND/OR	$16m - 16$	$14m - 7$	$26m - 13$	$9m$	$18m - 9$
	INV	$2m - 2$	$4m - 2$	$14m - 7$	$2m$	$4m - 2$
Identical Cell		yes	yes	yes	no	yes
(I/P, O/P) pins		(10, 10)	(9, 9)	(9, 9)	(10, 5)	(10, 10)

* A 3-input XOR gate is estimated as two 2-input XOR gates.
** Assuming that $8m$ FF's are used to bypass the dividend and the irreducible polynomial through the bit-serial inversion systolic array.

path delay using the retiming technique. Starting from the algorithm-level derivation, our design has an obvious area advantage over Kim's work although both methods lead to the same critical path delay.

In [14], Yan proposed a bit-serial systolic array for realizing the inversion operation based on the extended Euclid algorithm with a distributed ring counter; *i.e*., the shift-register-based counter. A division operation can then be achieved by using an inversion followed by a multiplication. As stated in Yan's previous work [13], the main concerns of employing a direct divider design or an inversion-multiplication based design are the trade-offs among the operating speed, area, latency, throughput rate, and area-time (AT) complexity. For a fair comparison, the multiplier design in [23], also mentioned in [14], is adopted in conjunction with Yan's inversion design to carry out the division operation. Note that to keep a consistent input format, *i.e*. all the input operands appearing at the same time, it is assumed that $8m$ ($2 \times 4m$) FF's are used to bypass the dividend and the irreducible polynomial through the bit-serial inversion systolic array. As shown in Table 3, although the bit-serial divider design based on our work cannot operate as fast as the inversion-multiplication design in [14], it has area and latency advantages.

For more insight into the analytical results, the gate counts, clock periods (CLK), throughputs, latency, and AT-complexity listed in Tables 2 and 3 are tabulated quantitatively in Table 4 assuming $m = 191$. To make our comparison independent of the underlying processes (or technologies) and the effects of synthesis tools, the area cost and critical path delay in Tables 2 and 3 were obtained from the summations of equivalent gate count and delay based on the cell library information in Table 5, as shown in Table 4. In this table, AT is defined as area divided by throughput and the clock period represents the summation of the critical path delay, $T_{\text{CLK}\rightarrow\text{Q}}$ and T_{setup}, where $T_{\text{CLK}\rightarrow\text{Q}}$ and T_{setup} are the propagation delay and setup time of flip-flops, respectively. Note that the total gate count is expressed by the equivalent number of used 2-input NAND gates based on the TSMC 0.18 μm cell library; the detailed information is listed in Table 5. As can be seen from

Table 4. Comparison of different designs over GF(2^{191}).

	Area (gates)	CLK (ns)	Throughput (Gb/s)	Latency (ns)	AT**
semi-systolic array					
[8]	14057.6*	1.509	0.33	576	42598
[17, 18]	10963.4*	1.258	0.39	480	28111
Ours_S	10963.4*	0.969	0.51	370	21496
bit-serial systolic array					
[10]	64942	1.122	0.89	1067	72968
[11]	53721	0.952	1.04	905	51654
[19]	60012.8	0.835	1.19	795	50430
[14] + [23]	63431.1	0.682	1.45	910	43745
Ours_B	48577.5	0.835	1.19	794	40821

* The area of the L/R shifter is not included.
** AT is defined as the ratio of area to throughput.

Table 5. Delay and area information of the standard cells.

	AND/OR	MUX	XOR	INV	FF
Area (gate)	1.3	2.7	2.7	0.7	5.3
Delay (ns)	0.136	0.134	0.153	0.025	–

Note: $T_{XOR3} \approx 2T_{XOR}$, $T_{\text{setup}} \approx 0.211$ns and $T_{\text{CLK}\rightarrow\text{Q}} = 0.182$ns.

Table 4, the semi-systolic design has smaller area and latency than the bit-serial version, while the bit-serial design can operate faster and has higher throughput than the semi-systolic one. Moreover, there is no global controlling signal in the bit-serial systolic architecture, but the semi-systolic one has to consider the effects of broadcasting signals for large values of m. According to Table 4, the proposed bit-serial divider design has the lowest AT complexity compared with the related bit-serial division design.

Both the semi-systolic and bit-serial array designs have been coded in the Verilog hardware description language and then synthesized using Synopsys tools based on the TSMC 0.18 μm library, including exhaustively verifying both designs for $m = 13$. We also included the provided wireload model and allocated 10% clock skew margin in the synthesis script. The input and output delays are both assigned 0.2ns. In our experiments, since the speed of a semi-systolic array depends on the field size m, we selected $m = 13$ and $m = 191$ as the sample designs for small and large fields, respectively. Experimental results show that the clock period and the total gate count are 1.26ns and 868 gates for $m = 13$, respectively, and 1.48ns and 12,665 gates for $m = 191$. For the same field sizes, our bit-serial systolic array design can operate as fast as 806 MHz (1.24ns) with total gate counts of 2,955 and 46,900 gates for $m = 13$ and 191, respectively. The operating speed would be independent of the field size because there is no global signal in the bit-serial array design. For clarity, The experimental results and the computed throughput, latency, and AT complexity over GF(2^{191}) are listed in Table 6. The discrepancy between the numbers in Tables 4 and 6 is due to the following reasons. The evaluated CLK period in Table 6 is larger than that in Table 4 because the wire delay and clock skew were included in our experiment. The area of the semi-systolic array in our implementation is larger than that in Table 4 since the area complexity analysis in Table 2 did not include the area of the L/R shifter.

Table 6. Area and performance of our designs over GF(2^{191}).

	Area (gates)	CLK* (ns)	Throughput (Gb/s)	Latency (ns)	AT
Ours_S	12665**	1.48	0.33	565	38378
Ours_B	46900	1.24	0.80	1180	58625

* The provided wireload model and 10% clock skew margin were included.
** The area of the L/R shifter was included.

The area of the bit-serial array in Table 6 being slightly smaller than that in Table 4 might result from the optimization strategy of the synthesis tool.

5. CONCLUSION

By making a change of variable and then updating its initial value accordingly, we derived a fast iterative division algorithm over GF(2^m) based on Stein's algorithm. Using our reformulated division algorithm, we developed two very high-speed array architectures: the bit-serial and semi-systolic divider architectures. Compared with other direct divider designs, our developments are better in both time and area complexities. Although the bit-serial divider design based on our development cannot be operated as fast as the inversion-multiplication based design, it has area, latency, and area-time advantages. In fact, such a variable change and transformation can be viewed as a simple pre-processing scheme that can be possibly applied to many algorithms, such as Euclid's algorithm.

REFERENCES

1. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, Prentice-Hall, New Jersey, 1995.
2. W. Stallings, *Cryptography and Network Security*, Englewood Cliffs, Prentice-Hall, New Jersey, 1999.
3. S. K. Jain, L. Song, and K. K. Parhi, "Efficient semisystolic architectures for finite-field arithmetic," *IEEE Transactions on Very Large Scale Integral Systems*, Vol. 6, 1998, pp. 101-113.
4. C. L. Wang and J. L. Lin, "Systolic array implementation of multipliers for finite fields GF(2^m)," *IEEE Transactions on Circuits and Systems*, Vol. 38, 1991, pp. 796-800.
5. T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis," *Information and Computation*, Vol. 78, 1988, pp. 171-177.
6. C. L. Wang and J. L. Lin, "A systolic architecture for computing inverses and division in finite field GF(2^m)," *IEEE Transactions on Computers*, Vol. 42, 1993, pp. 1141-1146.
7. D. E. Knuth, *Seminumerical Algorithms, The Art of Computer Programming*, Reading, Addison-Wesley, Mass., Vol. 2, 1981.
8. H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in GF(2^m)," *IEEE Transactions on Computers*, Vol. 42, 1993, pp. 1010-1015.

9. J. H. Guo and C. L. Wang, "Systolic array implementation of Euclid's algorithm for inversion and division in GF(2^m)," *IEEE Transactions on Computers*, Vol. 47, 1998, pp. 1161-1167.
10. J. H. Guo and C. L. Wang, "Bit-serial systolic array implementation of Euclid's algorithm for inversion and division in GF(2^m)," in *Proceedings of International Symposium on VLSI Technology, Systems, and Applications*, 1997, pp. 113-117.
11. 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, 2005, pp. 370-380.
12. Z. Yan and D. V. Sarwate, "Systolic architectures for finite field inversion and division," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2002, pp. 789-792.
13. Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in GF(2^m)," *IEEE Transactions on Computers*, Vol. 52, 2003, pp. 1514-1519.
14. Z. Yan, D. V. Sarwate, and Z. Liu, "Hardware-efficient systolic architectures for inversion in GF(2^m)," in *Proceedings of IEE Information Security*, Vol. 152, 2005, pp. 31-45.
15. J. Stein, "Computational problems associated with Racah algebra," *Journal of Computational Physics*, Vol. 1, 1967, pp. 397-405.
16. Y. Watanabe, N. Takagi, and K. Takagi, "A VLSI algorithm for division in GF(2^m) based on extended binary GCD algorithm," *IEICE Transactions on Fundamentals*, Vol. E85-A, 2002, pp. 994-999.
17. C. H. Kim and C. Y. Hong, "High-speed division architecture for GF(2^m)," *IEE Electronic Letters*, Vol. 38, 2002, pp. 835-836.
18. C. H. Wu, C. M. Wu, M. D. Shieh, and Y. T. Hwang, "High-speed, low-complexity systolic designs of novel iterative division algorithms in GF(2^m)," *IEEE Transactions on Computers*, Vol. 53, 2004, pp. 375-380.
19. C. H. Kim, S. Kwon, C. P. Hong, and I. G. Nam, "Efficient bit-serial systolic array for division over GF(2^m)," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2003, pp. 252-255.
20. R. P. Brent and H. T. Kung, "Systolic VLSI arrays for GCD computation," *IEEE Transactions on Computers*, Vol. 33, 1984, pp. 731-736.
21. P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, Vol. 44, 1985, pp. 519-521.
22. K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley and Sons Inc., New York, 1999.
23. S. Kwon, C. H. Kim, and C. P. Hong, "A systolic multiplier with LSB first algorithm over GF(2^m) which is as efficient as MSB first algorithm," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2003, pp. 633-636.

Ming-Der Shieh (謝明得) received the B.S. degree in Electrical Engineering from National Cheng Kung University, in 1984, the M.S. degree in Electronic Engineering from National Chiao Tung University, Taiwan, in 1986, and the Ph.D. degree in Electrical Engineering from Michigan State University, East Lansing, in 1993. From 1988 to 1989, he was an engineer at United Microelectronic Corporation, Taiwan. From 1993 to 2002, he was with the faculty of Department of Electronic Engineering, National Yunlin University of Science and Technology (NYUST). He received the teaching award from NYUST in 1998 and was the department chairman from 1999 to 2002. Since 2002, he has been with the Department of Electrical Engineering, National Cheng Kung University, where he is currently a full Professor. His research interests include VLSI design and testing, VLSI for signal processing, digital communication, and computer-aided design.

Wen-Ching Lin (林文景) received the B.S. and M.S. degree in Electrical Engineering from National Cheng Kung University, Taiwan, in 2005 and 2007 respectively. He is pursuing his Ph.D. degree in Electrical Engineering from National Cheng Kung University, Taiwan, since 2007. His primary research areas were VLSI design and architecture, cryptography and finite field theory.

Chien-Ming Wu (吳建明) obtained his B.S. and M.S. degrees, both in Electronic Engineering, from National Yunlin University of Science and Technology, Taiwan, in 1997 and 1999, respectively. He received the Ph.D. degree from the Graduate School of Engineering Science and Technology at National Yunlin University of Science and Technology, Taiwan, in 2003. He is currently an associate researcher at National Chip Implementation Center (CIC), Taiwan. His research interests include VLSI design in communication, coding theory, and digital signal processing.