High-Speed Systolic Architectures for Finite Field Inversion and Division *

Zhiyuan Yan
Department of Electrical and Computer Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015 USA
yan@lehigh.edu

Dilip V. Sarwate
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, IL 61801 USA
sarwate@uiuc.edu

ABSTRACT

Based on a new reformulation of the extended Euclidean algorithm, systolic architectures suitable for VLSI implementations are proposed for finite field inversion and division in this paper. The architectures proposed in this paper can achieve $O(m^2)$ area-time complexity, $O(m)$ latency, and critical path delays of two logic gates. These architectures show improved performances when compared with previously proposed architectures.

Categories and Subject Descriptors

B.2.4 [**ARITHMETIC AND LOGIC STRUCTURES**]: High-Speed Arithmetic

General Terms

Algorithms, Design

Keywords

Finite field arithmetic, Galois field, Reed-Solomon codes, cryptography

1. INTRODUCTION

Finite fields have found applications in areas such as cryptography, digital signal processing, and error-control codes. Hence, efficient architectures for finite field arithmetics are important to the performances of the VLSI implementation of these applications. Inversion and division are the most complicated finite field arithmetics, and various algorithms and architectures (see, for example, [1, 2][4]-[7] and the references therein) have been proposed based on different approaches such as recursive construction approaches, the extended Euclidean algorithm (EEA), the extended Stein algorithm (ESA), Fermat's theorem, or solving linear equation systems. In this paper, we focus on inversion/division architectures based on the EEA. Recently, Yan and Sarwate [6] proposed systolic inversion/division architectures based on a modified Euclidean algorithm, which can achieve critical path delays[1] (CPDs) as small as two logic gates. Their architectures have some other advantages when compared with previously proposed architectures. Interested readers are referred to [6] for details. The architectures in [6] are referred to as the YS architectures henceforth.

In this paper, based on a *different* reformulation of the EEA, we propose new systolic architectures for inversions and divisions in $GF(2^m)$. Compared to the YS architectures, the new architectures proposed in this paper have similar structures and achieve the same area-time (AT) complexity[2], throughput, critical path delay, and latency. Furthermore, the algorithmic reformulation explained below results in smaller hardware costs: the new architectures have less inter-cell connections and fewer latches than the YS architectures. Detailed performance comparisons are provided in the paper.

For all our architectures, we propose to compute a division B/A by first inverting A and then multiplying by B. This allows our division architectures to still achieve the same CPDs as the component inversion architectures while maintaining $O(m^2)$ AT complexity and $O(m)$ latency.

2. REFORMULATION OF THE EXTENDED EUCLIDEAN ALGORITHM

Let $g(x)$ be an irreducible polynomial of degree m over $GF(2)$. The field $GF(2^m)$ can be viewed as the set of binary polynomials of degree less than m with addition and multiplication modulo $g(x)$. The inverse of $a(x)$ modulo $g(x)$ is the unique polynomial $\hat{a}(x)$ that satisfies

$$a(x) \cdot \hat{a}(x) \equiv 1 \pmod{g(x)}.$$

*This work was supported by the National Science Foundation under Grants CCR 99-79381 and ITR 00-85929.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

[1]The terminology of this paper mostly follows Parhi [3].
[2]The AT complexity of an architecture is simply the product of the respective orders of the gate count and the number of cycles between successive outputs.

The extended Euclidean algorithm can be used to compute $\hat{a}(x)$. There are certain drawbacks for architectures based on direct implementation of the extended Euclidean algorithm (see, for example, [1]), and our implementations are based on a *new* reformulation of the EEA that is more suitable for VLSI implementations:

The Reformulated Euclidean (RE) Algorithm

1. **Set** $\epsilon \leftarrow 0$, $u(x) \leftarrow g(x)$, $v(x) \leftarrow a(x)$, $s(x) \leftarrow x^m$, and $t(x) \leftarrow 0$.
2. **Repeat** $2m$ times:
   (a) **set** $v(x) \leftarrow xv(x)$, $t(x) \leftarrow t(x)/x$, and $\epsilon \leftarrow \epsilon - 1$.
   (b) if $v_m = 1$ and $\epsilon < 0$, **set** $\epsilon \leftarrow -\epsilon$ and **swap** $u \leftrightarrow v$ and $s \leftrightarrow t$.
   (c) if $v_m = 1$, **set**
   $$\begin{aligned} v(x) &\leftarrow v(x) + u(x), \\ s(x) &\leftarrow s(x) + t(x). \end{aligned}$$
3. **Output** $\hat{a}(x) = t(x)$.

The details of the reformulation have been omitted here due to the limitation of space. Note that the operations in each iteration of the reformulated algorithm are shifting, swapping, or addition, all of which are easy to implement.

3. IMPLEMENTATION ISSUES

The RE algorithm above keeps track of four binary polynomials, $u(x)$, $v(x)$, $s(x)$, and $t(x)$. These four polynomials can be stored in four $(m+1)$-bit registers since their degrees are no more than m. To simplify the notation, the bits of the registers u, v, s, and t are numbered from left to right as m, $m-1$, $\cdots$, 0.

Some further improvements are made to the RE algorithm. It can be shown that the last iteration of the RE algorithm simply shifts the t register by one position, and hence can be omitted. Also, in order to eliminate the precedence between Steps 2(a) and 2(b) and hence reduce the path delay, the operation $\epsilon \leftarrow \epsilon - 1$ in Step 2(a) of each iteration is incorporated into the previous iteration, and ϵ is initialized to -1 instead of 0. Finally, the operations in 2(a), 2(b), and 2(c) of the RE algorithm are serial, which lead to a longer critical path delay if implemented directly. It turns out that these operations can be carried out in parallel. These improvements result in the following algorithm, presented in a more hardware-oriented manner:

Algorithm I

1. **Initialization:**
   $u_i^{(0)} = g_{i-1}$, $v_i^{(0)} = a_{i-1}$, $s_i^{(0)} = 0$ for $i = 1, \cdots, m$;
   $t_i^{(0)} = 0$ for $i = 0, \cdots, m$; $u_0^{(0)} = 0$, $v_0^{(0)} = 0$,
   $s_0^{(0)} = 1$, $t_{-1}^{(0)} = 0$, $v_{-1}^{(0)} = 0$, and $\epsilon^{(0)} = -1$.
2. For $j = 1, 2, \cdots, 2m-1$ do
   compute the control signals $C1 \stackrel{\text{def}}{=} v_m^{(j-1)}$
   and $C2 \stackrel{\text{def}}{=} C1 \wedge (\epsilon^{(j-1)} < 0)$, and
   for $i = 0, 1, \cdots, m$, **compute**
   $$v_i^{(j)} = \begin{cases} v_{i-1}^{(j-1)} & \text{if } C1 = 0 \\ v_{i-1}^{(j-1)} + u_i^{(j-1)} & \text{if } C1 = 1 \end{cases} \quad (1)$$
   $$s_i^{(j)} = \begin{cases} s_i^{(j-1)} & \text{if } C1 = 0 \\ s_i^{(j-1)} + t_{i-1}^{(j-1)} & \text{if } C1 = 1 \end{cases} \quad (2)$$
   $$\left(u_i^{(j)}, t_i^{(j)}\right) = \begin{cases} \left(u_i^{(j-1)}, t_{i-1}^{(j-1)}\right) & \text{if } C2 = 0 \\ \left(v_{i-1}^{(j-1)}, s_i^{(j-1)}\right) & \text{if } C2 = 1 \end{cases} \quad (3)$$
   $$\epsilon^{(j)} = \begin{cases} \epsilon^{(j-1)} - 1 & \text{if } C2 = 0 \\ -\epsilon^{(j-1)} - 1 & \text{if } C2 = 1 \end{cases} \quad (4)$$
3. **Output** $\hat{a}_i = t_{m-1-i}^{(2m-1)}$, for $i = 0, \cdots, m-1$.

Note that the updates in Equations (1)-(3) in Algorithm I are different from those in Equations (1) and (2) in [6]. In particular, the updates of $u_i^{(j)}$, $v_i^{(j)}$, $s_i^{(j)}$, and $t_i^{(j)}$ in Equations (1) and (2) of [6] depend on *eight* values: $u_i^{(j-1)}$, $v_i^{(j-1)}$, $s_i^{(j-1)}$, $t_i^{(j-1)}$, $u_{i-1}^{(j-1)}$, $v_{i-1}^{(j-1)}$, $s_{i-1}^{(j-1)}$, and $t_{i-1}^{(j-1)}$, whereas the updates of $u_i^{(j)}$, $v_i^{(j)}$, $s_i^{(j)}$, and $t_i^{(j)}$ in Equations (1)-(3) above depend on only *four* values: $u_i^{(j-1)}$, $v_{i-1}^{(j-1)}$, $s_i^{(j-1)}$, $t_{i-1}^{(j-1)}$. This difference leads to fewer inter-cell connections and latches in the implementation based on Algorithm I. On the other hand, the control signals and the computation of ϵ and the control signals in Algorithm I are exactly the same as those in the modified Euclidean algorithm in [6]. Thus, the circuitry of control mechanisms in the new systolic architectures is the same as that in the YS architectures.

4. NEW SYSTOLIC PIPELINED ARCHITECTURES

The new systolic pipelined architectures that implement Algorithm I above have similar structures as the YS architectures in [6], all consisting of two-dimensional arrays of control and computing cells. Hence, there is a natural correspondence between the new systolic architectures and their component cells proposed herein and those in [6]. For convenience of comparison, all the computing and control cells in this paper are given the same names as their corresponding cells in [6]. For the reasons given above, the control cells in the new architectures are the same as their corresponding cells in the YS architectures, whereas the computing cells and the inter-cell connections for the new architectures proposed in this paper are different from those in the YS architectures.

The updates of ϵ in Equation (4) can be implemented using either adders or, as explained in [6], ring counters. Furthermore, it was pointed out in [6] that the ring counters can be distributed to the computing cells leading to modular architectures wherein the control cell and the critical path delay are completely independent of the size of the field. As explained above, these discussions about the control mechanisms are applicable to the new architectures proposed in this paper. As in [6], we present the architectures according to their control mechanisms.

4.1 Architectures with Centralized Control

An example ($m = 2$) of the new systolic division architecture with centralized control is shown in Figure 1. The architecture consists of $(2m-1)$ rows of $(m+1)$ type-1 computing cells with each row controlled by one type-2 control cell and an $m \times m$ array of type-3 cells. Despite their similar structures, the number of the inter-cell connections for

the architecture shown in Figure 1 is obviously smaller than that for the YS architecture shown in [6, Figure 1]. The circuitry for the type-1 is shown in Figures 2. The type-2 and type-3 cells are the same as the corresponding cells in [6]; two possible implementations of the type-2 cell are shown in [6, Figure 2(c)] and [6, Figure 2(d)] respectively and the circuitry of the type-3 cell is depicted in [6, Figure 2(b)]. The new type-1 cell uses the same number of gates as that shown in [6, Figure 2(a)], but the number of inputs of the new type-1 cell is *six* instead of *ten* for the type-1 cell shown in [6, Figure 2(a)].

The $m \times m$ array of type-3 cells is needed only for the architectures for divisions and should be removed for inversion architectures. Similar to the approach in [6], the division B/A is implemented as the concatenation of an inversion and a multiplication, $B \cdot (A^{-1})$, since the modifications necessary to directly compute divisions B/A increase the critical path delay. As noted in [6], computing B/A as $B \cdot (A^{-1})$ allows us to achieve higher speed for division architectures at the expense of more hardware and a longer latency.

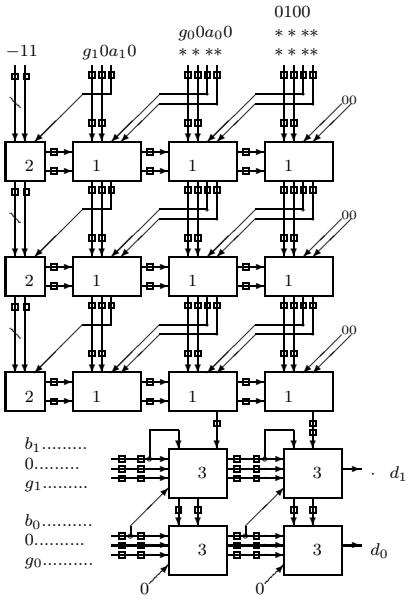

Figure 1: The pipelined architecture for division $d(x) = b(x)/a(x) \pmod{g(x)}$ in $GF(2^2)$

For continuous inputs, the architecture shown in Figure 1 produces one result per clock cycle after an initial latency of $5m-2$ and $7m-3$ clock cycles for inversion and division respectively. Obviously, the AT complexity of this architecture is $O(m^2)$. The CPD of the architecture in Figure 1 is given by $t_{cp} = \max\{t_{cp1}, t_{cp2}, t_{cp3}\}$ where t_{cp1}, t_{cp2}, and t_{cp3} are the CPDs of the type-1, type-2, and type-3 cells respectively. Due to the correspondence between the cells of the new architecture and the YS architectures, the CPD of the architecture shown in Figure 1 is exactly the same as that of the architecture shown in [6, Figure 1].

4.2 Architectures with Distributed Control

An example ($m = 2$) of the new systolic division architecture with distributed control is shown in Figure 3. The architecture consists of $(2m-1)$ rows of $(m+1)$ type-4 computing cells with each row controlled by one type-5 control cell. Again the number of the connections for the architecture shown in Figure 3 is smaller than that for the architecture in [6, Figure 3]. The circuitry for the type-4 computing cell is shown in Figures 4. The type-5 control cell is the same as the corresponding cell shown in [6, Figure 4(b)]. The new type-4 computing cell uses the same number of gates as that shown in [6, Figure 4(a)], but the number of inputs of the new type-4 cell is *nine* instead of *thirteen* for the type-4 cell shown in [6, Figure 4(a)].

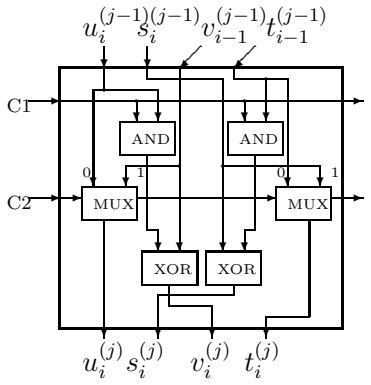

Figure 2: The architecture for the type-1 cells

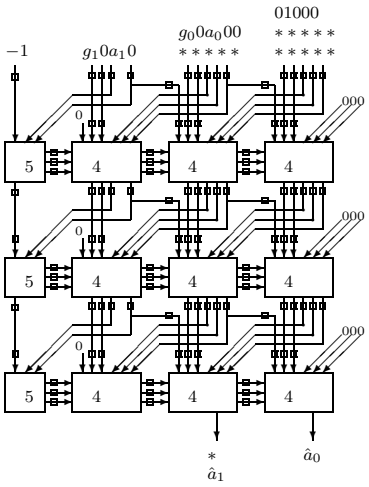

Figure 3: The pipelined architecture with distributed control for inversion in $GF(2^2)$

The throughput, latency, and AT complexity of the architecture in Figure 3 are the same as the architecture in Figure 1. The CPD of the architecture shown in Figure 3 is $t_{\text{XOR}} + t_{\text{AND}}$ where t_{AND} and t_{XOR} denote the delays of 2-input AND and XOR gates respectively. Note that this is exactly the same as the CPD of the corresponding YS architecture shown in [6, Figure 3]. Furthermore, this critical path delay is, as pointed out for the YS architecture in Figure 3 of [6], truly independent of m since all its cells are of small and fixed sizes independent of m. The small fixed sizes of all cells also lead to better systolic architectures.

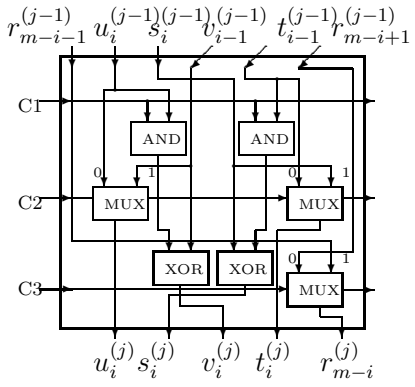

Figure 4: The architecture for the type-3 cells

5. PERFORMANCE COMPARISONS

In this section, the new systolic pipelined architectures are explicitly compared *only* with the YS architectures for the following reasons. The YS architectures are the best pipelined architectures based on the EEA known to us. Recall the one-to-one correspondence between the new architectures proposed in this paper and the YS architectures in [6] and note that the performances between the corresponding architectures are easily related. Since the YS architectures are compared with pipelined architectures proposed in [2] and [5] in Tables 1 and 2 of [6], comparisons between the new architectures and those in [2] and [5] are easily obtained through the correspondence between the new architectures in this paper and the YS architectures.

Throughput, Critical Path Delay, and Latency

The throughput of all the new architectures and the YS architectures in [6] are 1. For each of the new architectures proposed in this paper, the critical path delay and latency are the same as those of the corresponding YS architecture in [6] respectively.

Gate Counts and Total Hardware Costs

Each of the new systolic architectures uses $4m^2$ fewer latches and, if all the connections are counted equally, $8m^2$ fewer inter-cell connections than its corresponding YS architecture in [6]. The numbers of all the other gates used by each of the new systolic architectures are the same as those of the corresponding YS architecture. Hence, the hardware costs of the new systolic architectures we propose in this paper are clearly less than those of the corresponding YS architectures.

In summary, the new systolic architectures proposed in this paper use less hardware than the architectures in [6], while maintaining the same CPDs, throughputs, and latencies. As shown in Tables 1 and 2 of [6], the YS architectures achieve improved CPDs and hardware savings over the architectures in [2] and [5]. Hence, the new systolic architectures presented in this paper achieve the same improved CPDs and even *greater* hardware savings over the architectures in [2] and [5].

6. REFERENCES

[1] K. Araki, I. Fujita, and M. Morisue, "Fast Inverters over Finite Field Based on Euclid's Algorithm," *Trans. of IEICE*, vol. 72E, no. 11, pp. 1230–1234, November 1989.

[2] J.-H. Guo and C.-L. Wang, "Hardware-efficient Systolic Architecture for Inversion and Division in GF(2^m)," in *IEE Proceedings on Computers and Digital Techniques*, 1998, pp. 272–278.

[3] K. K. Parhi, *VLSI Digital Signal Processing Systems*, John Wiley and Sons, New York, 1999.

[4] 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 of Electronics, Communications and Computer Sciences*, vol. E85-A, no. 5, pp. 994–999, May 2002.

[5] C. H. Wu, C. M. Wu, M. D. Shieh, and Y. T. Wang, "Systolic VLSI Realization of a Novel Iterative Division Algorithm over $GF(2^m)$: a High-Speed, Low-Complexity Design," in *Proceedings of ISCAS'01*, 2001, pp. 33–36.

[6] Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 42, pp. 1515–1520, November 2003.

[7] Z. Yan and D. V. Sarwate, "Systolic Architectures for Finite Field Inversion and Division," in *Proceedings of ISCAS'02*, 2002, pp. 789–792.