A Unidirectional Bit Serial Systolic Architecture for Double-Basis Division over GF(2^m)

Amir K. Daneshbeh and M.A. Hasan
Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada
E-mail: akdanesh@engmail.uwaterloo.ca, ahasan@ece.uwaterloo.ca

Abstract

A unidirectional bit serial systolic architecture for division over Galois field GF(2^m) is presented which uses both triangular and polynomial basis representations. It is suitable for hardware implementations where the dimension of the field is large and may vary. This is the typical case for cryptographic applications. This architecture is simulated in Verilog-HDL and synthesized for a clock period of 1.4 ns using Synopsys. The time and area complexities are truly linear, since no carry propagation structures are present, and the complexity measures are equivalent or excel the best designs proposed so far.

1 Introduction

Finite field arithmetic has many applications in coding theory [1], and cryptography [10]. Among the arithmetic operations over finite fields the multiplicative inversion and division are the most costly ones. Most hardware proposals to compute inversion and division over GF(2^m), *e.g.*, [2, 5, 6, 13, 14] suffer from a common problem. In applications where the dimension of the field, m, is large or may vary, their implementation becomes cumbersome or even impractical. In particular, in public key cryptographic applications, the field dimension may exceed 4000 for discrete logarithm cryptosystems or 500 for elliptic curve cryptosystems [10].

In general, three major schemes to compute multiplicative inverses exist: Fermat's little theorem, variants of extended Euclidean algorithm (EEA) or a solution of a set of linear equations. First scheme is efficient if a fast squaring or multiplication scheme is available [9]. The extended Euclidean algorithm (EEA) based schemes to compute inversion and division are the most efficient in time and area but may require variable size counter-like structures with carry propagation chain to keep track of the difference of the degree of polynomials, *e.g.*, [5, 14, 11], in which case they are not suitable for high-performance and scalable VLSI implementations. The third method is generally inefficient for large values of field dimension. However, it is shown that a set of linear equations formed upon triangular basis representation can be solved by schemes similar to EEA [6].

Many systolic array proposals for multiplicative inversion or division over GF(2^m) based on EEA or its dual (extended Stein's algorithm), *i.e.*, [4, 8, 15, 16], all require counter-like structures with carry propagation delays. Since the carry propagation chain depends on the field dimension m, in general, it dominates the critical delay path. Only counter or comparator architectures with no carry propagation chain can be considered dimension independent. Moreover, a counter with no carry propagation chain can be easily transformed into a distributed structure.

In this paper we expand the double-basis multiplicative inversion scheme over GF(2^m) proposed in [6] to present a unidirectional bit serial systolic implementation of a divider with no carry propagation structure suitable for large values of m where the divisor is received in triangular basis, and both dividend and quotient, *i.e.*, result, are in polynomial basis. We will show that the extra cost of division in systolic implementations is minimal. Most importantly, an efficient stepwise restoring of the result polynomial is proposed as well.

2 Preliminaries

2.1 Polynomial and Triangular Basis Representations

Given an irreducible polynomial $F(x)$ of degree m over the finite field GF(2) and considering one of the roots of $F(x)$ such as ω, *i.e.*, $F(\omega) = 0$, it is well known that the set of elements $\Omega = \{1, \omega, \omega^2, \cdots, \omega^{m-1}\}$ is linearly independent and forms a basis referred to as a polynomial or canonical basis. Then, any element $A \in GF(2^m)$ can be

represented as $A = \sum_{i=0}^{m-1} a_{\Omega i}\omega^i$, where $a_{\Omega i} \in \{0, 1\}$ are the coordinates of A with respect to the polynomial basis Ω, or in a column vector form $\underline{a}_\Omega = [a_{\Omega 0}, a_{\Omega 1}, \cdots, a_{\Omega m-1}]^T$ over GF(2).

Given the polynomial basis Ω, the set $\Delta = \{\delta_0, \delta_1, \cdots, \delta_{m-1}\}$ is linearly independent and forms a basis if $\delta_i = \sum_{j=0}^{m-1-i} f_{i+j+1} w^j,\ 0 \le i \le m-1$, where f_i's are coefficients of $F(x)$ [7]. The set Δ is referred to as a triangular basis of Ω. Next, the element A of GF(2^m) can also be represented as $A = \sum_{i=0}^{m-1} a_{\Delta i}\delta_i$, where $a_{\Delta i}$ is the ith coordinate of A w.r.t. Δ. The change of coordinates of element A between Ω and Δ bases is given by [7]

$$a_{\Omega j} = \sum_{i=0}^{m-j-1} f_{i+j+1} a_{\Delta i} \quad 0 \le j \le m-1 \qquad (1)$$

and

$$a_{\Delta k} = \begin{cases} a_{\Omega m-1-k} & k = 0, \\ a_{\Omega m-1-k} + \sum_{i=0}^{k-1} f_{m-k+i} a_{\Delta i} & 1 \le k \le m-1. \end{cases} \qquad (2)$$

2.2 Double-Basis Inversion Algorithm

In [6], it is said that the inversion can be seen as the solution to a set of linear equations written as

$$\underline{1}_\Delta = \mathbf{H}(\underline{a}_\Delta)\underline{b}_\Omega \qquad (3)$$

where $\underline{1}_\Delta$ is a column vector representing the multiplicative identity element of GF(2^m) in triangular basis and $\mathbf{H}(\underline{a}_\Delta) = [\underline{a}_\Delta, \underline{a\omega}_\Delta, \cdots, \underline{a\omega}_\Delta^{m-1}]$ is a Hankel matrix built upon $\underline{a}_\Delta$, the column vector representation of element A with entries as its triangular basis coordinates. Then, the coordinates of the inverse element B in the polynomial basis can be computed by solving Equation (3). The Hankel matrix in (3) has $2m-1$ constant coefficients. Let these be $h_i,\ 0 \le i \le 2m-2$, then the matrix entries at (i, j) is h_{i+j}. Algorithm 1 as proposed in [6] can be used to solve Equation (3). Algorithm 1 is an optimization based on a variant of one of the algorithms discussed in [12].

Two aspects of Algorithm 1 should be noted. First, the *exit* condition of Algorithm 1 is ensured by monotonically decreasing value of $R^{(i)}(x)$. Second, an invariant similar to the invariant of the basic EEA [1, 2],

$$W^{(i)}(x)x^{2m-1} + U^{(i)}(x)x^{2m-2}H(x^{-1}) = R^{(i)}(x), \quad (4)$$

is preserved at each iteration. In the following we do not discuss the $W^{(*)}(x)$ polynomials since they are not relevant to the inversion. It can be noted that the polynomial updating step follows an extended Euclidean algorithm (EEA) transformation using a long-division scheme. Basically, the polynomial updating step of Algorithm 1 performs a series of multi-stepwise long division iterations where the invariant of Equation (4) is preserved.

Algorithm 1 Double-Basis Inversion Algorithm.

input: $H(x) = h_{2m-2}x^{2m-2} + h_{2m-3}x^{2m-3} + \cdots + h_1 x + h_0$.
output: The inverse element.
  Initialization:
  $i \leftarrow 0$; $U^{(-1)}(x) \leftarrow 0$; $U^{(0)}(x) \leftarrow 1$;
  $R^{(-1)}(x) \leftarrow x^{2m-1}$; $R^{(0)}(x) \leftarrow x^{2m-2}H(x^{-1})$;
  Polynomial Updating:
  while $(\deg R^{(i)}(x) > m-1)$ **do**
    $i \leftarrow i+1$;
    $d \leftarrow \deg R^{(i-2)}(x) - \deg R^{(i-1)}(x)$;
    $R^{(i)}(x) \leftarrow R^{(i-2)}(x) - x^d R^{(i-1)}(x)$;
    $U^{(i)}(x) \leftarrow U^{(i-2)}(x) - x^d U^{(i-1)}(x)$;
    if $(\deg R^{(i)}(x) \ge \deg R^{(i-1)}(x))$ **then**
      swap $(R^{(i)}(x), R^{(i-1)}(x))$;
      swap $(U^{(i)}(x), U^{(i-1)}(x))$;
return $U^{(i)}(x)$

Algorithm 1 can be extended to perform *division* over finite fields by initializing $U^{(0)}(x)$ with the dividend whose degree is $\le m-1$. In this case, while computing $U^{(i)}(x)$, a modulo reduction may be required, since the degree of $x^d U^{(i-1)}(x)$ may become m or more. Thus, the irreducible polynomial $F(x)$ must be input as well.

Solving Equation (3) and consequently Algorithm 1 can be considered as a two step process: the Hankel matrix entry formation in the initialization step and the polynomial updating step. In general, the formation of $2m-1$ entries of the Hankel matrix requires recursive formulae such as

$$h_i = \begin{cases} a_{\Delta i} & 0 \le i \le m-1, \\ \sum_{j=0}^{m-1} h_{i-m+j} f_j & m \le i \le 2m-2. \end{cases} \qquad (5)$$

In [6], it is shown that by a suitable choice of irreducible polynomial, such as trinomials of type $x^m + x + 1$, the formation of the entries of the Hankel matrix requires no recursion and they can be easily generated using a single row of XOR gates. It can be shown that for all practical cases the Hankel matrix entry formation has a lower computational complexity compared with the polynomial updating step of Algorithm 1. In what follows a systolic implementation of the polynomial updating step of Algorithm 1 is proposed.

3 Systolization of the Polynomial Updating Step

For large values of m, the complexity of a centralized implementation of the polynomial updating step of Algorithm 1 is dominated by the presence of structures (counters or comparators) with carry propagation delay and long control interconnects. For a VLSI implementation of Algorithm 1 with large values of m a bit serial systolic archi-

tecture will be more suitable, not only because of bit-level pipeline data processing but most importantly because in this case a distributed control structure with no carry propagation can be devised.

Using a bit serial architecture of Algorithm 1, the computation of $d \leftarrow \deg R^{(i-2)}(x) - \deg R^{(i-1)}(x)$ may be implemented implicitly by a series of bitwise shifts of polynomial $R^{(i-1)}(x)$ up to an alignment condition of the leading coefficients of polynomials $R^{(i-2)}(x)$ and $R^{(i-1)}(x)$. Such an implementation has two major consequences discussed next.

3.1 Upper Bound of Number of Iterations of Algorithm 1

The *exit* condition of Algorithm 1 occurs after a variable number of iterations. However, by imposing one single degree shift of polynomial $R^{(i-1)}(x)$ per iteration, it is easy to show that in the worst case after exactly $2m$ iterations the exit condition is satisfied, that is the degree of both polynomials $R^{(i-1)}(x)$ and $R^{(i-2)}(x)$ are less than or equal $m-1$.

Hence, in a modified version of Algorithm 1, an exact number of iterations can be defined as a **for** loop of $2m$ iterations. In a bit serial systolic implementation of such an algorithm where each iteration is executed in a different PE, no explicit exit condition checking is required.

The $2m$ iterations is a worst-case. In all other cases, the exit condition may occur sooner, *i.e.*, $i < 2m$. Then a modified variant of Algorithm 1 must ensure that the returned value of $U^{(i)}(x)$ is not modified in consecutive iterations up to $2m$. This will be shown later. However, first, we must introduce the second major advantage of a single degree shift of polynomial $R^{(i-1)}(x)$ in evaluating the **swap** condition $(\deg R^{(i)}(x) \geq \deg R^{(i-1)}(x))$.

3.2 Distributed Relative Degree Tracking of Algorithm 1

In Algorithm 1, the *swap* condition ensures that in the next iteration the polynomial $R^{(i-1)}(x)$ has lower degree (relative to $R^{(i-2)}(x)$) always in order to be shifted. According to Algorithm 1, at each iteration after a new polynomial $R^{(i)}(x)$ is computed, it is necessary that the pre-shifted polynomial $R^{(i-1)}(x)$ and its degree to be known in the next iteration. However, in a bit serial implementation with single degree shifts, no effective restoring of the shifted polynomial is necessary in order to decide which polynomial will be shifted next if a mechanism to keep track of the relative degree of two polynomials is devised as shown below.

Let δ represent an up/down counter which counts the number of shifts of the shifting polynomial $R^{(i-1)}(x)$, and initialized to zero. Initially polynomial $R^{(-1)}(x) = x^{2m-1}$, and if no restoring of polynomials is needed, then in all successive iterations when an alignment of two polynomials $R^{(i-1)}(x)$ and $R^{(i-2)}(x)$ occurs, the degree of both polynomials will be $2m-1$. However, δ keeps track of their relative degree.

At all iterations while the polynomial $R^{(i-1)}(x)$ is shifted, δ is increased. At an *alignment* of polynomials $R^{(*)}(x)$, if $\delta \geq 0$, an *overwriting* occurs and the sign of δ is swapped. The *overwriting* is the only iteration when both polynomials are updated. Afterwards, while δ is negative, a shift of polynomial $R^{(i-1)}(x)$ increments δ towards zero and again into positive until the next *overwriting* condition. Hence, only the sign of δ is needed to decide whether an *overwriting* is required or not. Furthermore, such an up/down counter can be easily implemented by using a distributed ring-counter without any carry propagation.

3.3 Shifted Result Problem

If Algorithm 1 is implemented by one shift per iteration as discussed in Section 3.1, then at the iteration where i becomes $2m$ (inside the while loop),

$$\deg R^{(2m)}(x) \leq m-1 \quad \text{(restoring case)}.$$

An inherent problem to implement the polynomial updating step of Algorithm 1 without an effective restoring of polynomials $R^{(*)}(x)$ is what we may call the *shifted result* problem. In Sections 3.1, 3.2, it has been shown that by keeping track of the degree of polynomials $R^{(*)}(x)$, both the *exit condition* checking and the *swap condition* can be done while one of the two polynomials are left shifted bitwise continuously. Mathematically, this means that the degree of a non-restored polynomial $R^{(i-1)}(x)$ always equals to $2m-1$ or approaching it. Hence, in this non-restoring scheme, at the iteration where i becomes $2m$, we have

$$\deg R^{(2m)}(x) \leq 2m-1 \quad \text{(non-restoring case)}.$$

As a consequence, in a non-restored scheme, not the correct result $U^{(2m)}(x)$ but rather a shifted result $U^{(2m)}(x)x^m$ will be returned unless a corrective action is applied. Different strategies to tackle a similar problem while computing multiplicative inverses using extended Euclidean algorithm are proposed: initializing $U^{(0)}(x) = x^{-m}$ [8]; initializing $U^{(-1)}(x) = x^m$, $U^{(0)}(x) = 0$, but at each iteration divide $U^{(i-1)}(x)$ instead of multiply such that $U^{(i)}(x) \leftarrow U^{(i-2)}(x) - U^{(i-1)}(x)/x^d$, ([1], Section 2.3), [16]; multiplying or dividing $U^{(i-1)}(x)$ alternatively [2]; or using an auxiliary polynomial [4]. In a bit serial systolic implementation, a more efficient method appears to be the use of double delay elements to perform a stepwise restoring of $U^{(*)}(x)$ as will be proposed in the following variant of Algorithm 1.

3.4 A Division Algorithm suitable for systolization

Using the optimizations suitable for an efficient systolization of Algorithm 1 introduced in Sections 3.1, 3.2, 3.3, Algorithm 2 is proposed which can perform the division as well.

Algorithm 2 Division Variant of Algorithm 1 with Restoring Result Polynomial.

input: irreducible polynomial $F(x)$,
dividend in polynomial basis $C(x)$, and
$H(x) = h_{2m-2}x^{2m-2} + h_{2m-3}x^{2m-3} + \cdots + h_1 x + h_0$
which is a function of $A(x)$.
output: $C(x)/A(x) \mod F(x)$.
Initialization:
$\delta \leftarrow 0$; $U \leftarrow 0$; $V \leftarrow C(x)$;
$R \leftarrow x^{2m-1}$; $S \leftarrow x^{2m-2}H(x^{-1})$;
Polynomial Updating:
for $2m$ times **do**
  if $(s_{2m-1} = 0 \ \& \ \delta \geq 0)$ **then**
    $[R,\ S] \leftarrow [R,\ xS]$;
    $[U,\ V] \leftarrow [U/x,\ V]$;
  else if $(s_{2m-1} = 0 \ \& \ \delta < 0)$ **then**
    $[R,\ S] \leftarrow [R,\ xS]$;
    $[U,\ V] \leftarrow [U,\ xV \mod F(x)]$;
  else if $(s_{2m-1} = 1 \ \& \ \delta < 0)$ **then**
    $[R,\ S] \leftarrow [R,\ x(R-S)]$;
    $[U,\ V] \leftarrow [U,\ (x(U-V)) \mod F(x)]$;
  else if $(s_{2m-1} = 1 \ \& \ \delta \geq 0)$ **then**
    $\delta \leftarrow -\delta$; // overwriting case //
    $[R,\ S] \leftarrow [S,\ x(R-S)]$;
    $[U,\ V] \leftarrow [V,\ (U-V) \mod F(x)]$;
  $\delta \leftarrow \delta + 1$;
return U

Algorithm 2 receives as input the field defining irreducible polynomial $F(x)$, the dividend polynomial $C(x)$, and $2m-1$ entries of the Hankel matrix as defined in Section 2.2. Unlike in Algorithm 1, the indices (in the superscript) of $R^{(*)}(x)$ and $U^{(*)}(x)$ polynomials are dropped in Algorithm 2 and two consecutive $R^{(*)}(x)$ polynomials (respectively $U^{(*)}(x)$) are renamed as a pair of distinct polynomials $[R,\ S]$ (respectively $[U,\ V]$).

In Algorithm 2, the *alignment* condition is shown as $(s_{2m-1} = 1)$, since $(r_{2m-1} = 1)$ always, and where s_{2m-1} is the $(2m-1)$st degree coefficient of S. Also, the *overwriting* condition is shown as $(s_{2m-1} = 1 \ \& \ \delta \geq 0)$, where $\&$ represents a logical AND operator. The **for** loop of $2m$ represents the worst-case number of iterations to have the correct result in U, *i.e.*, at the $2m$th iteration, an overwriting occurs and $U \leftarrow V$. In all cases where U holds the final results before the $2m$th iteration, Algorithm 2 ensures that the value of U is not modified in consecutive iterations while $\delta < 0$.

The innovation of Algorithm 2 is the restoring transformation of $[U,\ V]$ versus the non-restoring transformation of pair $[R,\ S]$.

3.5 Bit Serial Systolic Architecture for Division

It is said that Algorithm 2 is optimized for systolization. A bit serial unidirectional systolic architecture for the updating step of Algorithm 2 is depicted in Figure 1. A single type PE is used and no latches external to PEs are required. Furthermore, a distributed ring-counter without any carry propagation is used.

In Figure 1, there are three sets of inputs to each PE: *datapath* signals $(r,\ s,\ u,\ v,\ f)$; *state* signals $(dseq,\ dec,\ update)$; and the *control* signal $start$. The *datapath* inputs are defined in Algorithm 2. The $start$ is defined as a one following by m zeroes. The $update$ is the latched value of incoming $s_{in} = s_{2m-1}$ at $(start = 1)$. Signals dec and $dseq$ represent the sign and magnitude of the counter δ. The *state* signals are initialized to 1, 0, 0 respectively. The latency of this architecture is $5m-2$ cycles. The throughput which also depends on the size of inputs is 1 division per $2m$ cycles. Hence, as a common feature of all systolic architectures two back-to-back computations can be processed.

In Figure 2, one processing element (PE) of architecture in Figure 1 is shown. Two of the *state* signals, $update$ and dec are precomputed and saved for the next PE. The signed integer δ may be represented by a sign bit, dec, and by a $(m+1)$-bit vector $dseq$ where $dseq = 2^{|d|}$. At *initialization* $\delta = 0$, hence, $dec = 0$, and $dseq = 00\cdots001$. At each shift of S while $dec = 0$, $dseq$ is left shifted. At *overwriting*, dec is set, and $dseq$ is right shifted. In all consecutive iterations until $dseq$ returns back to 1, dec remains set. It will reset at $dseq = 1$. The dec in Figure 2 represents a boolean simplification of this set-reset mechanism , exactly, $dec \leftarrow \overline{dseq}_{in} \ \& \ (dec_{in} \mid s_{in})$. The overline represents a logical NOT and "|" a logical OR operator. The dec is used to check for an *overwriting* event. An internal non-latched *state* signal $ovrw$ is computed as $(update_{in} \ \& \ \overline{dec}_{in})$.

The *reduction* condition for the division algorithm is defined as $((v_{in} = v_m = 1) \ \& \ (dec_{in} = 1) \ \& \ (start_{in} = 1))$, however, mapped more efficiently by a balanced delay path for the input of three latches u_1, v_1, and $reduce_{lat}$. Accordingly, the critical delay path is $t_{cp} = max(3T_{A2},\ 2T_{A2} + T_{X2},\ T_{A2} + 2T_{X2}) + T_{M2}$, where T_{A2}, T_{M2} and T_{X2} are defined as the delay of a 2-input AND gate, MUX and XOR gate respectively.

4 Implementation Considerations and Comparison

In this section some implementation features and possible enhancements of the proposed bit serial systolic archi-

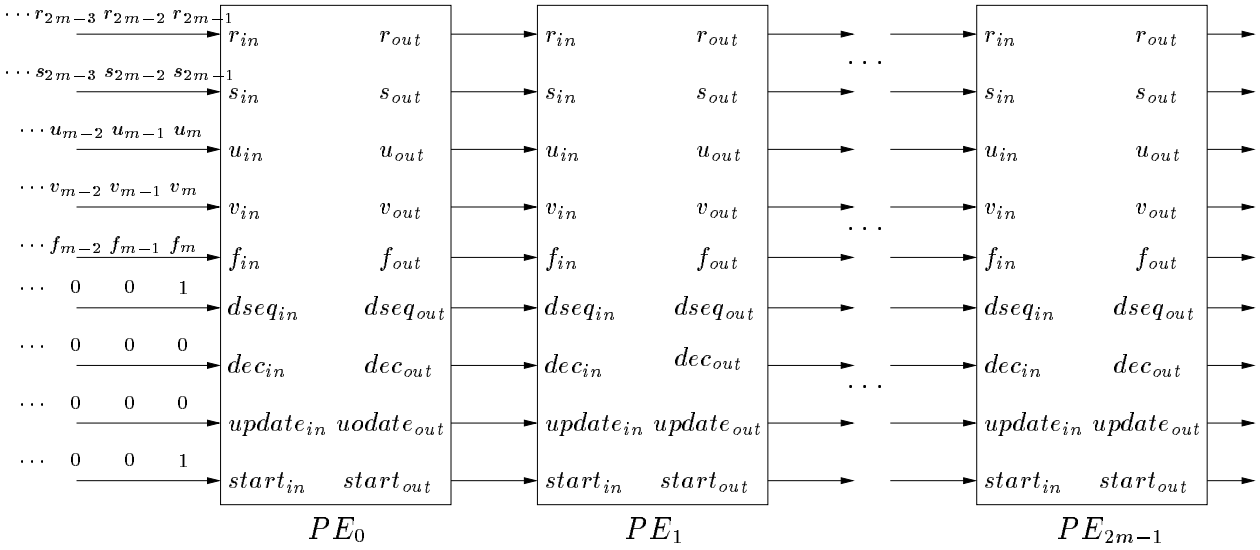

Figure 1. Bit serial unidirectional systolic architecture for Division.

tecture are discussed and a comparison with two systolic architectures are provided.

4.1 Dealing with Varying Dimension

This architecture may process any field dimension up to the size defined by the number of the PEs where no external buffering is available. Let N represent the number of PEs in this architecture. In a bit serial unidirectional systolic architecture as described, it is easy to see that if $N \geq 2m$, the coefficients of the inverse element start to appear at the output of the $2m$th PE after $4m$ cycles. Hence, using the architecture in Figure 1, it is possible to process any field dimension $m \leq N/2$ as far as the correct result can be captured at the right cycle.

Two such schemes can be considered. Either the output of the $2m$th PE should be accessible directly, or a mechanism should be in place to de-activate all remaining PEs from $2m+1$ to N, and to capture the result at the output of Nth PE always. The former does not require any inner PE modification and provides the optimal latency [3].

4.2 Implementation Results

The divider as described in Section 3.5 is coded in Verilog-HDL, simulated and verified. The Verilog model of Figure 2 has been synthesized using Synopsys with a standard CMOS 0.18μ library for a clock period of 1.4 ns. The average setup time of the flip-flops was 0.32 ns and the clock skew set at 10%. The area reports, including the flip-flops, are equivalent to 120 2-input NAND gates. It should be pointed out that non-combinational area is the dominant part, more than 75%.

4.3 Comparison

Table 1 shows a comparison of proposed divider with two representative dividers, namely [4], and [5]. The design in [4] is chosen since it is a divider with best overall performance figures known to us and it is also in the same category as ours (a bit serial unidirectional systolic architecture). The second design [5] is chosen to show why single row parallel-in designs in particular when they use counter-like structures are not suitable for large values of m.

In Table 1, the deteriorating effect of a carry propagation counter in both area and timing complexity in the architecture of [5] w.r.t. the design in [4] and ours can be seen. Further, the single row systolic architectures with parallel inputs as in [5] are impractical for large values of m, not only due to large IO pins but mainly due to large number of IO latches for synchronization purposes. On the other hand, the architecture in [4] requires an auxiliary polynomial to restore the shifted result problem. This can be seen by an extra number of IO pin and extra MUXs required in each PE.

5 Conclusions

In this paper a unidirectional bit serial systolic architecture of a double-basis divider over GF(2^m) is presented. Main contribution is the restoring mechanism introduced in Algorithm 2 and implemented by using double delay elements in Figure 2. Also, a distributed counter structure

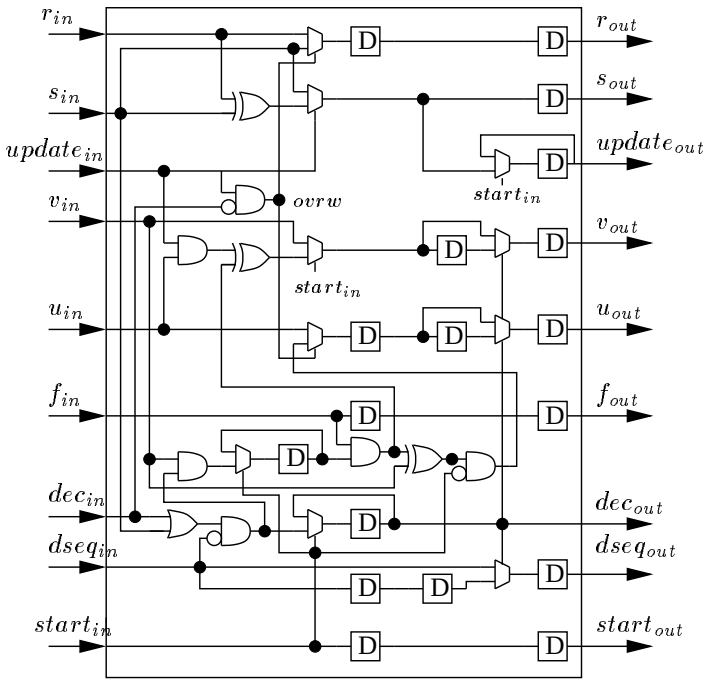

Figure 2. Processing element for the divider.

again implemented by using double delay elements is proposed. This bit serial divider is useful in applications where other field operations are performed using double-basis, and moreover, the field dimension may be large or variable such as cryptographic applications. This architecture is well suitable for high performance (high clock rates) applications. It is seen that delay and area are highly dependent on the delay elements used. Delay or area optimizations by using special latch design may be explored further.

Since the polynomial updating step of Algorithm 1 follows an extended Euclidean algorithm transformation, this architecture can be used in all applications where such algorithm is used. In particular, it can compute division over GF(2^m) when the element is represented in polynomial basis as well.

References

[1] E. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill Book Company, 1968.

[2] H. Brunner, A. Curiger, and M. Hofstetter. "On Computing Multiplicative Inverses in GF(2^m)". *IEEE Transactions on Computers*, 42(8):1010–1015, Aug. 1993.

[3] A. Daneshbeh and M. Hasan. "A Class of Scalable Unidirectional Bit Serial Systolic Architectures for Multiplicative Inversion and Division over GF(2^m)". `http://www.cacr.math.uwaterloo.ca`, Dec 2002. Technical Report CORR 2002-35.

[4] J.-H. Guo and C.-L. Wang. "Bit-serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in $GF(2^m)$". *Proceedings of Technical Papers, International Symposium on VLSI Technology, Systems, and Applications*, pages 113–117, June 1997.

[5] 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*, 47(10):1161–1167, Oct. 1998.

[6] M. Hasan. "Double-Basis Multiplicative Inversion over GF(2^m)". *IEEE Transactions on Computers*, 47(9):960–970, Sep. 1998.

[7] M. Hasan and V. Bhargava. "Architecture for a Low Complexity Rate-Adaptive Reed-Solomon Encoder". *IEEE Transactions on Computers*, 44(7):938–942, July 1995.

[8] C.-T. Huang and C.-W. Wu. "High-Speed C-Testable Systolic Array Design for Galois-field Inversion". *Proceedings of the European Design and Test Conference*, Paris:342–346, March 1997.

[9] T. Itoh and S. Tsujii. "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Basis". *Information and Computation*, 78:171–177, 1988.

[10] A. Menezes, P. van Ooreschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[11] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. "Fast Key Exchange with Elliptic Curve Systems". *Advances in Cryptology, EUROCRYPT '95, LNCS 921, Springer-Verlag*, pages 43–56, 1995.

[12] Y. Sugiyama. "An Algorithm for Solving Discrete-Time Wiener-Hopf Equations based on Euclid's Algorithm".

Table 1. Comparison of bit serial systolic dividers.

Circuits	Guo et al. [5]	Guo et al. [4]	Figure 2
Time Complexity	$O(m)$	$O(m)$	$O(m)$
Area Complexity	$O(m \log m)$	$O(m)$	$O(m)$
Throughput	1	$\frac{m}{m+1}$	$\frac{m}{2m}$
Latency	$8m-1$	$5m-4$	$5m-4$
IO pins	$O(m)$	10	9
Unidirectional	yes	yes	yes
Critical Path	$2T_{A2}+T_{X3}+2T_{M2}$	$T_{A2}+T_{X2}+T_{X3}+T_{M2}$	$2T_{A2}+T_{X2}+T_{M2}$
Single Cell	no	yes	yes
Gate Count:	total of different cells	per cell	per cell
IO Synch. Latch	$4m$	-	-
inner-inter Latch	$46m+4m\lceil \log_2 m+1 \rceil$	22	18
MUX	$35m+2$	11	10
XOR	$11m$	5	3
AND/OR	$26m$	8	7
others	adder, zero-check	-	-

IEEE Transactions on Information Theory, 32(5):394–409, May 1986.

[13] Y. Watanabe, N. Takagi, and K. Takagi. "A VLSI Algorithm for Division in GF(2^m) Based on Extended Binary GCD Algorithm". *Transactions of The Institute of Electronics, Information and Communication Engineers, 2002. IEICE'02*, E85(5):994–999, 2002.

[14] C.-H. Wu, C.-M. Wu, M.-D. Shieh, and Y.-T. Hwang. "Systolic VLSI Realization of a Novel Iterative Division Algorithm over GF(2^m): A High Speed Low-Complexity Design". *Proceedings of the IEEE International Symposium on Circuits and Systems. ISCAS'01*, 4:33–36, 2001.

[15] C.-H. Wu, C.-M. Wu, M.-D. Shieh, and Y.-T. Hwang. "An Area-Efficient Systolic Division Circuit over GF(2^m) for Secure Communication". *Proceedings of the IEEE International Symposium on Circuits and Systems. ISCAS'02*, 5:733–736, 2002.

[16] Z. Yan and D. Sarwate. "Systolic Architectures for Finite Field Inversion and Division". *Proceedings of the IEEE International Symposium on Circuits and Systems. ISCAS'02*, V:789–792, June 2002.