Sequential Multiplier with Sub-linear Gate Complexity

Anwar Hasan, Christophe Negre

Sequential Multiplier with Sub-linear Gate Complexity [1]

M. Anwar Hasan and Christophe Negre

Abstract

In this article, we present a new sequential multiplier for extended binary finite fields. Like its existing counterparts, the proposed multiplier has a linear complexity in flip-flop or temporary storage requirements, but a sub-linear complexity in gate counts. For the underlying polynomial multiplication, the proposed field multiplier relies on the Horner scheme.

Index Terms

Binary polynomial multiplication, sequential multiplier, Horner scheme, sub-linear gate complexity.

1 INTRODUCTION

Multiplication over finite fields is used in both symmetric and asymmetric key cryptosystems, and has been a subject of study, especially for its improved implementation, among many researchers as evidenced in their publications, see for example [8], [13], [5], [10], [14], [11], [4]. In this article, we consider a class of hardware architectures for multiplication over *extended binary* finite field $\mathbb{F}_{2^n}$.

For a small size field like the one used in the well known *symmetric* key system *Advanced Encryption Standard* (AES) [1], multiplication can be easily realized with a simple look-up-table (LUT), where the product of each combination of input is pre-stored, preferably in a read-only-memory (ROM). This table based method is however impractical for large size fields used in modern *asymmetric* key cryptosystems like Elliptic Curve Cryptography (ECC) [9], [6]. Asymptotically, a look-up table for multiplication can take as much as $O(n2^{2n})$ bits of ROM and the time delay would be essentially equal to the table access time, which is normally a function of the table size. Techniques exist to reduce the table size at the expense of an increased number of table accesses along with logic circuits.

For high speed multiplication over a large field, an alternative approach is to use only logic circuits or gates in some *parallel* way and to perform a multiplication on-the-fly, e.g., a fully bit parallel multiplier.

For ECC, where the value of n is only a few hundreds, a practical bit-parallel multiplier would take $O(n^{1+\epsilon})$ logic gates, where $0 < \epsilon \leq 1$, and have a time delay of $O(\log n)$ layers of gates.

In constrained environments, where silicon space is of prime concern, a fully bit parallel multiplier for a large field is generally too big to fit the design. On the other end of the spectrum of choices, it is possible, at least in theory, to realize a multiplier with as few as only two gates –one XOR and one AND– where the gates will be used repeatedly and many intermediate bit-level results will need to be stored. We will refer to this type of multipliers as *super-sequential*, since they would take $O(n^{1+\epsilon})$ iterations or clock cycles.The storage requirement for a super-sequential multiplier would be at least $O(n^{1+\epsilon})$. Because of large timing and memory requirements, super-sequential multipliers are likely to be least attractive for both constrained and high speed applications and this is perhaps why no practical design of the super-sequential multiplier has been reported in the literature.

Between the above two extremes, namely fully parallel and super-sequential multipliers, one can find various *sequential* multipliers, either at bit or digit-level. A sequential multiplier operates in an iterative way over a number of clock cycles, typically $O(n)$ cycles for bit sequential or $O(n/d)$ cycles for digit sequential architecture, where d is the digit size. A bit sequential multiplier has a shorter critical path than its digit sequential counterpart. For a bit or digit sequential multiplier, since intermediate results are stored in registers or read-write memories, the storage requirement is at least $O(n)$ bits. Besides registers, a sequential multiplier also requires logic gates, generally in the amount of $O(n)$, i.e., linear to n.

In our work, we differentiate ROM from registers in the usual way. In other words, while the content of a ROM is fixed and specific to a certain arithmetic operation, the content of a register can change. More importantly, registers are time-shared, i.e., a set of registers can be used by multiple operations, for example, additions, multiplication and inversion, provided that these operations are not executed simultaneously. On the other hand, like ROM but unlike registers, most of the logic gates are assumed to be specific to a certain operation and not shared. As a result, even if an arithmetic unit, such as a multiplier, can be designed to take advantage of time shared registers, there is still a need to reduce the amount of logic gates, especially for area constrained applications.

In this article, we propose a new sequential multiplier. Like its existing counterparts the proposed one requires $O(n)$ bits of registers, but only $O(n^{1-\epsilon})$, $0 < \epsilon \leq 1$, logic gates. To the best of our knowledge, no sequential multiplier has been reported earlier that has a gate count *sub-linear* to n. To give an idea about where the proposed multiplier fits with respect to various others, in Table 1 we list their space and time complexities in an asymptotic or general way.

The remainder of this article is organized as follows. In Section 2, we briefly review schemes for a fully bit parallel and a digit-sequential multiplier. In Section 3, we present our sequential multiplier which is based on the Horner method and has a sub-linear gate complexity. Finally, a complexity comparison and

concluding remarks are given Section 4.

TABLE 1

List of different types of multipliers and their complexities

Architecture	# Logic gates or ROM cells	# Flip-flops in registers	Critical Path Delay	# Clock Cycles
Full LUT	$n2^{2n}$	0	$O(n)$	1
Fully parallel circuit	$O(n^{1+\varepsilon})$	0	$O(\log(n))$	1
Digit-sequential circuit	$O(nd)$	$O(n)$	$O(\log(d))$	n/d
Bit-sequential circuit	$O(n)$	$O(n)$	$O(1)$	n
Proposed	$O(n^{1-\epsilon})$	$O(n)$	$O(\log(n))$	$O(n)$

2 Review of extended binary field multipliers

Field $\mathbb{F}_{2^n}$ can be viewed as the set of binary polynomials in t modulo an irreducible polynomial $P(t)$ of degree n. Let $A(t)$ and $B(t)$ be two elements of $\mathbb{F}_{2^n}$. The multiplication of these two elements is $C(t) = A(t) \times B(t) \mod P(t)$, for which we can use the following two steps:

$$C'(t) = A(t) \times B(t), \tag{1}$$

and then reduce $C'(t)$ modulo $P(t)$ to get

$$C(t) = C'(t) \mod P(t). \tag{2}$$

2.1 Bit parallel architectures

In this type of architectures, all the bits of the product $C = AB \mod P$ are generated in parallel. The design of the circuit is based on pure combinatorial logic and does not make any use of storage or flip-flops. Computations are done with no reuse of circuits. The approach we recall here performs the polynomial multiplication (1) and the reduction (2) separately. The reduction modulo P is generally quite simple when P is sparse (i.e., P is a pentanomial or a trinomial) and in this situation the reduction operation can be implemented with $O(n)$ XOR gates and a delay of $O(1)$. Consequently, we will focus here to the polynomial multiplication which is the most costly and the most complicated part of a finite field multiplier.

- *Quadratic multiplier*. Let $C' = \sum_{i=0}^{2n-2} c'_i t^i$ be the product $C' = A \times B$ where A and B are degree $n-1$

polynomials in t. The coefficients c'_i are given in terms of a_i and b_i as follows

$$\begin{aligned}
c'_0 &= a_0 b_0, \\
c'_1 &= a_0 b_1 + a_1 b_0, \\
&\vdots \quad\quad \vdots \\
c'_n &= a_0 b_n + a_1 b_{n-1} + \cdots + a_n b_0, \\
c'_{n+1} &= a_1 b_n + a_2 b_{n-1} + \cdots + a_n b_1, \\
&\vdots \quad\quad \vdots \\
c'_{2n-3} &= a_{n-2} b_{n-1} + a_{n-1} b_{n-2}, \\
c'_{2n-2} &= a_{n-1} b_{n-1}.
\end{aligned}$$

The computation of each c_i is performed in parallel with AND gates (for the products $a_\ell b_j$) and a binary tree of XOR gates. The resulting gate complexity of this polynomial multiplier is equal to n^2 AND gates and $n(n-1)$ XOR gates. The critical path of the multiplier is $\lceil \log_2(n) \rceil D_X + D_A$ where D_X and D_A are delays for a two input XOR and AND gates respectively.

- *Subquadratic multiplier [2].* A second strategy, which was first proposed in [4], is based on the method of Karatsuba. Specifically, the multiplication is performed by applying the following formula recursively

  Splitting: $A = A_0 + A_1 t^{n/2}$ and $B = B_0 + B_1 t^{n/2}$,
  Recursive products: $M_0 = A_0 B_0, M_1 = (A_0 + A_1) \times (B_0 + B_0)$ and $M_2 = A_1 B_1$,
  Reconstruction: $C' = M_0 + t^{n/2}(M_1 + M_0 + M_2) + t^n M_2$,

  As stated in [4], this approach requires $6n^{\log_2(3)} + 8n - 2$ XOR gates, $n^{\log_2(3)}$ AND gates and has a delay of $3\log_2(n)D_X + D_A$. Recently, some optimizations have been proposed: Leone in [7] has noticed that it is possible to slightly reduce the space complexity of the Karatsuba multiplier if we stop the recursion when we reach polynomials of degree 4 or 8 and then apply quadratic method. More recently, Fan *et al.* in [3] have proposed to use an odd/even splitting which reduces the delay to $2\log_2(n)D_X + D_A$.

In Table 2, we recall the complexity of the schoolbook method and then give the complexity of multipliers based on the Karatsuba combined with the optimization approaches of Leone [7] and Fan *et al.* [3].

2.2 Digit Sequential Multiplier

In sequential multipliers, computations are done with reuse of circuits and the final result is obtained over several clock cycles. Here, we review the digit sequential version of the multiplier of Song and Parhi [12]. Song and Parhi fix a digit size d and define $m = \lceil \frac{n}{d} \rceil$ and then they rewrite the two polynomials A and

TABLE 2

Complexity of bit parallel binary polynomial multiplier architectures

Method	Space Complexity		Delay
	# XOR	# AND	
Quadratic	n^2-1	n^2	$D_A+\lceil\log_2(n)\rceil D_X$
Subquadratic [3]	$6n^{\log_2(3)}-8n+2$	$n^{\log_2(3)}$	$D_A+(2\log_2(n))D_X$
Subquadratic [3] and [7] combined	$\frac{13}{3}n^{\log_2(3)}-8n+2$	$\frac{16}{9}n^{\log_2(3)}$	$D_A+(2\log_2(n)-2)D_X$

B as follows

$$\begin{aligned} A(t) &= \textstyle\sum_{i=0}^{m-1} A_i(t)t^{di} \text{ with } \deg_t A_i(t) < d, \\ B(t) &= \textstyle\sum_{i=0}^{m-1} B_i(t)t^{di} \text{ with } \deg_t B_i(t) < d. \end{aligned}$$

The left-to-right (L-to-R) method for digit-serial multiplication is based on the following expansion of the product $A \times B \mod P$

$$\begin{aligned} C &= A \times B \mod P \\ &= ((\ldots((AB_{m-1} \mod P)t^d + AB_{m-2} \mod P)t^d + \cdots)t^d + AB_1 \mod P)t^d + AB_0 \mod P. \end{aligned}$$

The previous expression results in the following L-to-R multiplication algorithm. It consists of a sequence of multiplications by t^d followed by the accumulation of AB_i in C modulo P .

Algorithm 1 Left-to-right digit sequential multiplication [12]

Require: $A(t)=\sum_{i=0}^{m-1} A_i(t)t^{di}$ and $B(t)=\sum_{i=0}^{m-1} B_i(t)t^{di}$.

Ensure: $C = A \times B$

$C \leftarrow 0$

for $i = m-1$ to 0 **do**

$C \leftarrow (t^d \times C + B_i A) \mod P$

end for

Return(C)

In Algorithm 1, the multiplication by t^d is performed by shifting the coefficients of C. The multiplication $B_i \times A$ is done using m parallel quadratic space complexity polynomial multipliers of size d. These digit multipliers compute the product B_iA_j for $j = 0, \ldots, m-1$. Each product B_iA_j has degree $2d-2$: thus the upper part of this product is added to C_{j+1} and the lower part to C_j. Then the updated value of C has $m+1$ digits, since in C_m we store the upper part of the product B_iA_{m-1}. If we assume that

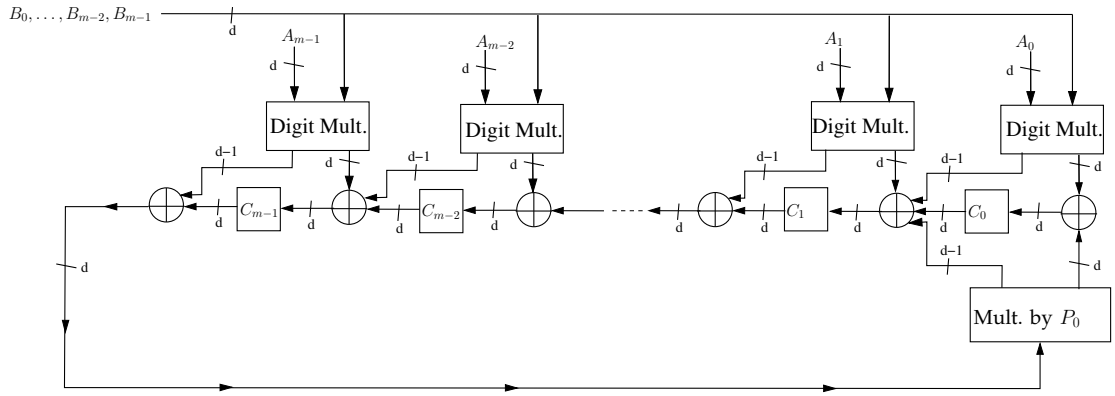
$P = t^{md} + \sum_{i=0}^{m-1} P_i(t)t^{id}$, then we reduce C_m modulo P using the following expression

$$C_m t^{md} = \sum_{i=0}^{m-1} P_i C_m t^{id}.$$

In practical applications, P can be taken as a pentanomial or a trinomial, so the P_i's are in general either equal to zero or sparse. Here, for the sake of simplicity, we will further assume that the only non-zero P_i is P_0. This means that the reduction step consists of $C_m t^{md} = P_0 C_m$ and can be performed with at most $3(d-1)$ XOR gates (indeed, if P is a pentanomial, then P_0 has 4 non-zero coefficients).

The resulting digit sequential multiplier is depicted in Fig. 1. The *Digit Mult.* boxes represent the quadratic parallel polynomial multipliers for degree d polynomials.

Fig. 1. Left-to-right digit sequential multiplier of [12]



The complexity of this digit sequential multiplier is given below. Note that $\mathcal{S}_\oplus$ represents the number of XOR gates, $\mathcal{S}_\otimes$ represents the number of AND gates and $\mathcal{D}$ represents the delay of the critical path of the architecture. We remark that the total computational delay is the number of clock cycles times the clock period, whose duration is at least the critical path delay.

$$\begin{aligned}
\mathcal{S}_\oplus &= m(d^2 + d - 1) + 5d - 4, \\
\mathcal{S}_\otimes &= md^2, \\
\mathcal{D} &= \lceil \log_2(d) + 5 \rceil D_X + D_A, \\
\#\text{Flip-flops} &= 3md, \\
\#\text{ Clock cycles} &= m.
\end{aligned}$$

Remark 1: We have restricted our study to a quite specific P but in general when P has degree n and is a pentanomial, a similar approach can be applied with no additional significant complexity. Our choice was motivated only by the simplicity of this special case in the digit sequential design.

3 SUB-LINEAR GATE COMPLEXITY USING HORNER'S METHOD

In this section, we present a multiplier with sub-linear gate complexity based on the Horner method. Our approach takes advantage of a subquadratic multiplier as well as a digit sequential multiplier. As before, let $A(t)$ and $B(t)$ be two polynomials of degree $< n$. Let m and d be two integers such that $m \times d = n$. We rewrite A and B in a digit form as follows:

$$\begin{aligned} A(t) &= \sum_{i=0}^{m-1} A_i(t)t^{di} \text{ with } \deg_t A_i(t) < d \\ B(t) &= \sum_{i=0}^{m-1} B_i(t)t^{di} \text{ with } \deg_t B_i(t) < d \end{aligned}$$

We multiply A and B using Horner's method. Here we assume that the irreducible polynomial P has the form $P = X^{dm} + P_0$ where P_0 has a degree less than d and has at most four non-zero coefficients. This assumption is not restrictive, a similar multiplier with the same order of area and delay complexity can be designed with a more general pentanomial P. The resulting method is described in Algorithm 2 and uses the Left-to-Right approach. We have exhibited all the m^2 digit multiplications and we have separated the operations $C + A_j B_i t^{jd},\ 0 \leq j < m-1$ and $0 \leq i < m$, from the operations $t^d(C + A_{m-1}B_i t^{d(m-1)}) \mod P$ and $(C + A_{m-1}B_i t^{d(m-1)}) \mod P$ which involve reduction modulo P.

The architecture based on Algorithm 2 is shown in Fig. 2. In this architecture, elements A and B are each stored in a register of m cells, each cell containing d bits. The product C is also stored in a register of m cells and is initialized with m zeros. Registers containing A and C are shifted cyclically once in every clock cycle. On the other hand, B is shifted once after every m clock cycles. At each clock cycle one of the following cases is performed and in Fig. 2 these three cases are indicated in red, blue and green.:

- *Case 1 (Red).* A digit A_j for $0 \leq j < m-1$ is output from the A register and the digit B_i is output from B. The A and C registers are left shifted but no shift is operated in the B register. The two digits A_j and B_i are multiplied in the digit multiplier. Then the data follows the blue lines. This corresponds to the addition of $A_j B_i$ to C_j and the result moves in the right most cell of the C register. The "carry" of $A_j B_i$ is added to C_{j+1} and the result is stored in the left most cell of C.
- *Case 2 (Blue).* The digit A_{m-1} and B_i are output from A and B respectively. The A register is left shifted and the B register is right shifted, but no left shifts are applied to the C register. The two digits A_{m-1} and B_i are multiplied and then, the data follows the red paths. The operation corresponds to the multiplication by t^d and the computation of $((C_{m-1} + A_{m-1}B_i)t^{md} \mod P)$.
- *Case 3 (Green).* This is the last step before the output of the product C. At this step, the C register is left shifted. The digits A_{m-1} and B_0 are output from the A and B registers and then multiplied through the digit multiplier. Then the lower part of the product is added to C_{m-1} and the upper part is reduced modulo P, i.e., is multiplied by P_0 and added to C_0 and C_1.

Algorithm 2 Horner's method with reduction modulo P

Require: $A = \sum_{j=0}^{m-1} A_j t^{dj}, B(t) = \sum_{i=0}^{m-1} B_i t^{di}$ in $\mathbb{F}_2[t]$ with $\deg A(t), \deg B(t) < n$ and $\deg A_j, \deg B_i < d$
Ensure: $C = A \times B$
$C \leftarrow 0$
for $i = m-1$ **to** 1
  for $j = 0$ **to** $m-2$
    $U_L + t^d U_H \leftarrow A_j B_i,$
    $C_j \leftarrow C_j + U_L, C_{j+1} \leftarrow C_{j+1} + U_H$
  end for
  Case 1: Comp. of $C + (\sum_{j=0}^{m-2} A_j t^{dj}) B_i$
  $temp \leftarrow C_{m-1}$
  for $j = m-1$ **to** 3
    $C_j \leftarrow C_{j-1}$
  end for
  $U_L + t^d U_M + t^{2d} U_H \leftarrow (temp + A_m B_i) \times P_0$
  $C_2 \leftarrow C_1 + U_H,\ C_1 \leftarrow C_0 + U_M,\ C_0 \leftarrow U_L$
  Case 2: Comp. of $(C + A_{m-1} t^{d(m-1)} B_i) \times t^d \mod P$
end for
for $j = 0$ **to** $m-2$
  $U_L + t^d U_H \leftarrow A_j B_0$
  $C_j \leftarrow C_j + U_L,\ C_{j+1} \leftarrow C_{j+1} + U_H$
end for
Case 1: Comp. of $C + (\sum_{j=0}^{m-2} A_j t^{dj}) B_0$
$U_L + t^d U_H \leftarrow B_{m-1} A_0$
$C_{m-1} \leftarrow C_{m-1} + U_L$
$V_L + t^d V_H \leftarrow U_H \times P_0$
$C_0 = C_0 + V_L, C_1 = C_1 + V_H$
Case 3: Comp. of $(C + A_{m-1} t^{d(m-1)} B_0) \mod P$
return(C)

To avoid any confusion between Case 2 and Case 3, in Fig. 2 we have used two *Mult. by* P_0 boxes which perform a multiplication by P_0 in the two distinct paths.

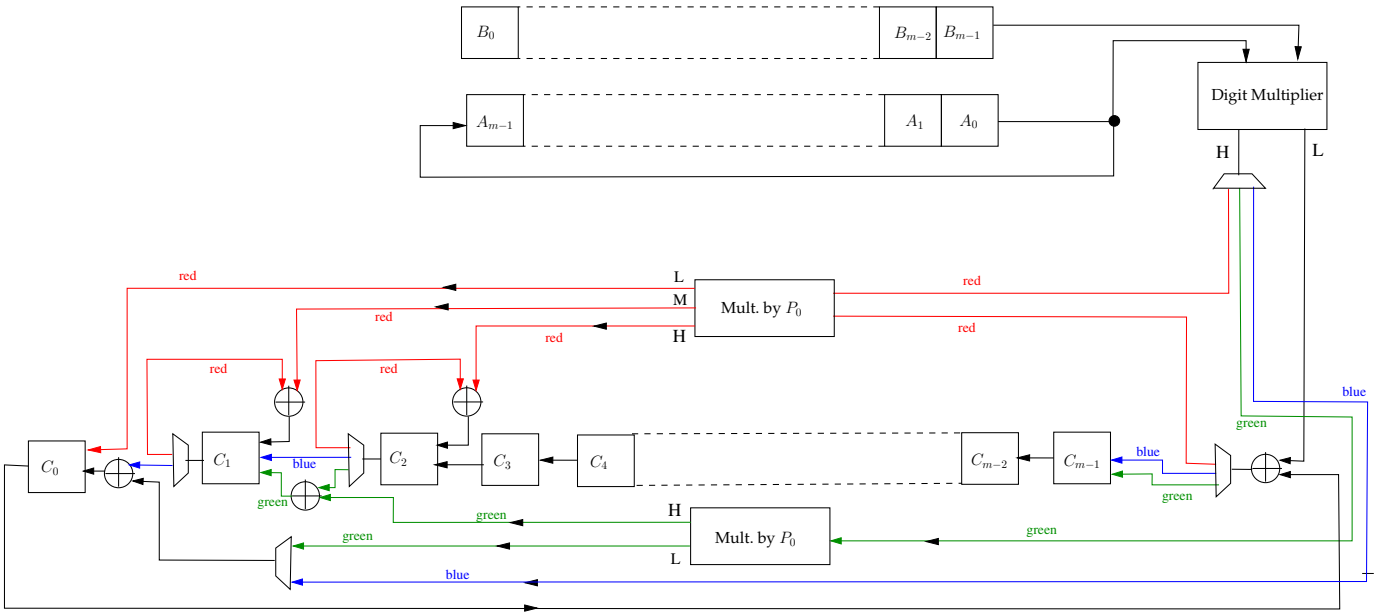
The main differences between Algorithm 1 and Algorithm 2 and their respective architectures are the following:

1) in Algorithm 2 and in the corresponding architecture in Fig. 2, the digit multiplications are done in sequence and not through m parallel digit multipliers.
2) In the proposed multiplier, the digit multiplication uses a subquadratic multiplier, and the digit size d in Fig. 2 is generally larger than the one in Fig. 1.

In Fig. 3, we illustrate the functioning of the proposed multiplier by presenting the first four cycles of the sequential multiplier for $n = 32$.

Complexity evaluation. Below we list, along with a brief explanation, the number of flip-flops, AND gates

Fig. 2. Sub-linear sequential multiplier based on the Horner method

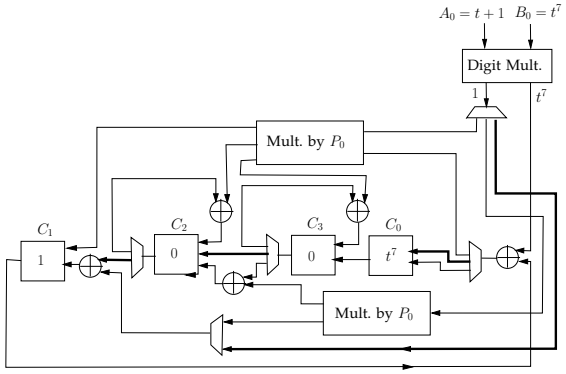


and XOR gates and the critical path of the architecture in terms of m and d. For the sake of simplicity, we assume that each path in Fig. 2 is of width d.

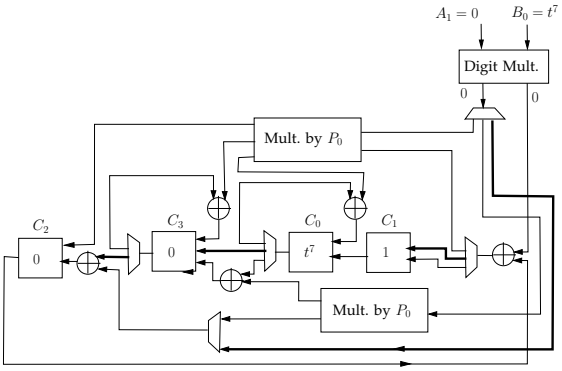
- $3m$ cells of d bits each for A, B and C. Thus, the architecture in Fig. 2 requires $3md$ flip flops in total.
- The number of AND gates is equal to $\mathcal{S}_{\otimes}(d)$ which is the number of AND gates of the digit multiplier. The number of XOR gates is equal to $\mathcal{S}_{\oplus}(d)+5d$, where we have counted the following: the XOR gates of one multiplier of size d, five adders of size d, and $2 \times 2d + 2 \times d$ XORs for the two multiplications by P_0.
- The architecture in Fig. 2 also requires one 1:2 and three 1:3 demultiplexers, and one 2:1 multiplexer where each input/output is d bits wide.
- The critical path delay is equal to $\mathcal{D}(d) + 4D_X + D_A$ where $\mathcal{D}(d)$ is the delay of the digit multiplier. The number of clock cycles is equal to m^2.

Using the subquadratic complexity results based on [3] and [7] as given in Table 2, we obtain the

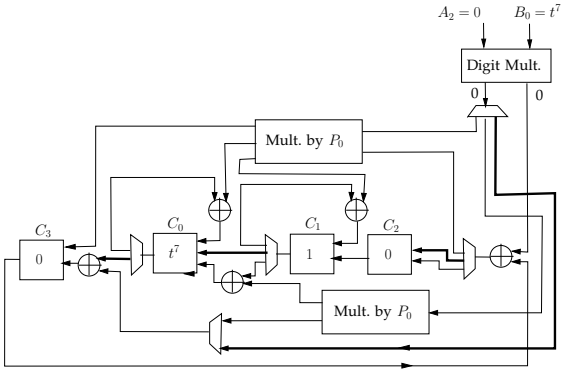
Fig. 3. First four steps of the multiplication of $A = (t+1) + (t+1) \times t^{24}$ and $B = t + t^7 \times t^{24}$ modulo $P = t^{4\times 8} + t^7 + t^6 + t + 1$ with $m = 4$ and $d = 32$ (note that after the demultiplexer, the followed paths are indicated with bold lines)



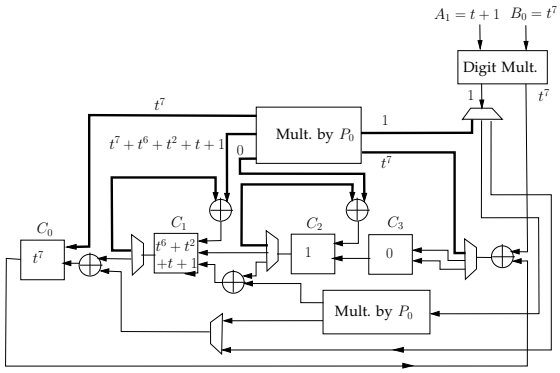
(a) Step 1. Case 1 applies and the data follows the red paths.



(b) Step 2. Case 1 applies and the data follows the red paths.



(c) Step 3. Case 1 applies and the data follows the red paths.



(d) Step 4. Case 2 applies (blue paths). In this step, the product $A_3 \times B_0 = (t^8 + t^7)$ is multiplied by P_0. The result is $t^{15} + t^{13} + t^{10} + t^9 + t^8 + t^7 = t^7 + (t^7 + t^6 + t^2 + t + 1)t^8 + 0 \times t^{16}$.

following complexity for the multiplier in Fig. 2

$$\begin{aligned} \#\text{XOR} &= \tfrac{13}{3} d^{\log_2(3)} + 3d + 2, \\ \#\text{AND} &= \tfrac{16}{9} d^{\log_2(3)}, \\ \#\text{Flip-flops} &= 3md, \\ \text{Delay} &= (2\log_2(d) + 2)D_X + D_A, \\ \#\text{Clock cycles} &= m^2. \end{aligned} \tag{3}$$

4 COMPLEXITY COMPARISON AND CONCLUSION

We finally obtain a sub-linear gate complexity by fixing the value of d and m as $d = \sqrt{n}$ and $m = \sqrt{n}$. The resulting complexity is given in Table 3. In the same table, we recall the complexity of the digit sequential multiplier of Song and Parhi [12] reviewed earlier in Section 2 (for which we assume that $m = n/d$).

TABLE 3

Comparison with the proposed sequential multiplier

Architecture	# AND	# XOR	#FFs	Critical Path Delay	# Clock cycles	Overall delay
Proposed	$1.77n^{0.78}$	$4.33n^{0.78} + 3\sqrt{n} + 2$	$3n$	$(\log_2(n) + 4)D_X + D_A$	n	$n((\log_2(n) + 2)D_X + D_A)$
Song and Parhi [12]	nd	$n(d + 1 - \frac{1}{d}) + 5d - 4$	$3n$	$\lceil \log_2(d) + 5 \rceil D_X + D_A$	$\frac{n}{d}$	$\frac{n(\lceil \log_2(d)+5 \rceil D_X + D_A)}{d}$

The results in Table 3 suggest that the proposed sequential multiplier has a gate count which is much smaller than that of the multiplier of [12]. On the other hand, the proposed multiplier has a longer critical path resulting in a much larger overall delay. Thus the proposed sub-linear gate complexity multiplier offers a new area-time trade-off. For example, if we consider $n = 256$ (i.e, $d = m = 16$), the multiplier of [12] would require 8596 XORs, 8192 ANDs, 16 clock cycles with 9 levels of gates on the critical path, whereas the corresponding values of the proposed multiplier are 397 XORs, 143 ANDs, 256 clock cycles and 11 levels of gates.

REFERENCES

[1] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, 2002.

[2] H. Fan and M. A. Hasan. A New Approach to Sub-quadratic Space Complexity Parallel Multipliers for Extended Binary Fields. *IEEE Trans. Computers*, 56(2):224–233, September 2007.

[3] Haining Fan, Jiaguang Sun, Ming Gu, and Kwok-Yan Lam. Overlap-free karatsuba-ofman polynomial multiplication algorithms. Cryptology ePrint Archive, Report 2007/393, 2007.

[4] G. Guajardo, T. Guneysu, C. Paar, , S. Kumar, and J. Pelzl. Efficient Hardware Implementation of Finite Fields with Applications to Cryptography. *Acta Applicandae Mathematicae*, 93(1-3):75–118, September 2006.

[5] M. A Hasan, M. Wang, and V. K. Bhargava. A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields. *IEEE Trans. Computers*, 42(10):1278–1280, 1993.

[6] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[7] M. Leone. A New Low Complexity Parallel Multiplier for a Class of Finite Fields. In *Proceedings of CHES'01*, pages 160–170, London, UK, 2001. Springer-Verlag.

[8] E. Mastrovito. VLSI Designs for Multiplication over Finite Fields *F* (2^m). In *6th International Conference on Applied Algebra, Algebraic Algorithm and Error-Correcting Codes (AAECC-6)*, pages 297–309, 1988.

[9] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology, proceeding's of CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer-Verlag, 1986.

[10] C. Paar. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields. *IEEE Trans. Comput.*, 45(7):856–861, 1996.

[11] A. Reyhani-Masoleh. A New Bit-Serial Architecture for Field Multiplication Using Polynomial Bases. In *CHES 2008*, pages 300–314, 2008.

[12] L. Song and K. K. Parhi. Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *J. VLSI Signal Process. Syst.*, 19(2):149–166, 1998.

[13] B. Sunar and C. Koc. Mastrovito Multiplier for All Trinomials. *IEEE Trans. Computers*, 48(5):522–527, 1999.

[14] M. Wang and I. F. Blake. Bit serial multiplication in finite fields. *SIAM J. Discret. Math.*, 3(1):140–148, 1990.