Computers and Electrical Engineering 35 (2009) 54–58

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

# An area/performance trade-off analysis of a GF($2^m$) multiplier architecture for elliptic curve cryptography

Miguel Morales-Sandoval, Claudia Feregrino-Uribe, René Cumplido *, Ignacio Algredo-Badillo

*Computer Science Department, National Institute for Astrophysics, Optics and Electronics, Luis Enrique Erro No. 1, Tonantzintla, Pue. 72840, Mexico*

ARTICLE INFO

*Article history:*
Received 24 January 2007
Received in revised form 26 November 2007
Accepted 27 May 2008
Available online 31 August 2008

ABSTRACT

A hardware architecture for GF($2^m$) multiplication and its evaluation in a hardware architecture for elliptic curve scalar multiplication is presented. The architecture is a parameterizable digit-serial implementation for any field order $m$. Area/performance trade-off results of the hardware implementation of the multiplier in an FPGA are presented and discussed.
© 2008 Elsevier Ltd. All rights reserved.

## 1. Introduction

Finite fields like the binary GF($2^m$) and the prime GF($p$) have been used successfully in error correction codes and cryptographic algorithms. In elliptic curve cryptography (ECC), the overall performance of cryptographic ECC schemes is hardly determined by arithmetic in GF($2^m$), being inversion and multiplication the most time consuming operations. According to the literature, arithmetic in GF($2^m$) binary fields using polynomial basis leads to efficient hardware implementations of ECC. Some works related to hardware implementation of ECC have reported parameterizable GF($2^m$) arithmetic units to compute the most time consuming operation in elliptic curve cryptography, the scalar multiplication. Those architectures are based on a diversity of multiplication algorithms, for example: Massey Omura multipliers [1], linear feedback shift registers multipliers [2], Karatsuba [3,4], and digit-serial multipliers [5]. Other works have studied and implemented GF($2^m$) multipliers using polynomial basis like [8,9]. Others have used different algorithms, like the Montgomery multiplication [10,11]. Although, from the architectural point of view, it is well known that the arithmetic unit has a big impact in the timing and area of hardware for scalar multiplication, it is not clear whether the architecture performance is due to the parallelism in the multipliers, the number of multipliers, or the kind of multipliers used. This technical communication presents the hardware architecture of a GF($2^m$) digit-serial multiplier and evaluates the area/performance trade off, considering various digit sizes $d$ and finite field orders $m$.

## 2. GF($2^m$) multiplication architecture

Multiplication in GF($2^m$) in polynomial basis is the operation $A(x) * B(x) \bmod F(x)$, that can be computed using a variety of proposed algorithms in the literature. On the one hand, serial or bit-serial algorithms, consider each individual bit of the operand $B(x)$ which implies a latency for multiplication of $m$ clock cycles. On the other hand, digit-serial multipliers consider a group of $d$ bits of operand $B(x)$ at time and perform the multiplication in $m/d$ cycles. However, it is not clear which is the

* Corresponding author. Tel.: +52 222 2663100; fax: +52 222 2663152.
*E-mail address:* rcumplido@inaoep.mx (R. Cumplido).

0045-7906/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2008.05.008

best size of $d$ for this kind of multiplier to achieve an appropriate performance that meets the constraints for a specific application. Varying the size of the digit allows to explore the cost in area and performance improvements from a serial implementation up to a parallel multiplication architecture. At each iteration, the operand $A(x)$ is multiplied by a group of $d$ bits of operand $B(x)$ and the result is reduced modulo $F(x)$. The result is added accumulatively to the result of the next iteration, considering the following $d$ bits of $B(x)$ until all $B(x)'$ bits are processed. The reduction in the operation latency comes with an increment in the complexity at each step of the multiplication. For our implementation, we consider the digit serial Algorithm 1 [6], the same algorithm used for the work reported in [5], and show the different area/time results when the digit size is varied. This will help designers to select suitable parameters when implementing architectures for high level applications like cryptographic algorithms or error correction code algorithms.

**Algorithm 1.** Digit-serial multiplication: multiplication in GF($2^m$)
**Require**: $A(x)$, $B(x)$ in GF($2^m$), $F(x)$ the $m+1$ grade irreducible polynomial
**Ensure**: $C(x) = A(x) * B(x) \bmod F(x)$

1: $C(x) \leftarrow B_{s-1}(x)A(x) \bmod F(x)$
2: **for** $k$ from $s-2$ down to 0 do
$C(x) \leftarrow x^d C(x)$
$C(x) \leftarrow C(x) + B_k(x)A(x) \bmod F(x)$
**end for**

Being $B(x)$ an element in GF($2^m$) using polynomial basis, this is viewed as the polynomial $b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0$. For a positive digit number $d < m$, the polynomial $B(x)$ can be grouped so that it can be expressed as $B(x) = x^{(s-1)d}B_{s-1}(x) + x^{(s-2)d}B_{s-2}(x) + \cdots + x^dB_1(x) + B_0(x)$, where $s = \lceil d/m \rceil$ and each word $B_i(x)$ is defined as follows:

$$B_i(x) = \begin{cases} \sum_{j=0}^{d-1} b_{id+j}x^j & \text{if } 0 \leqslant i < s-1, \\ \sum_{j=0}^{(m\%d)-1} b_{id+j}x^j & \text{if } i = s-1. \end{cases}$$

If $x^d$ is factored from the grouped representation of $B(x)$, the resulting expression is

$$B(x) = x^d(x^d(\cdots(x^d(x^dB_{s-1}(x) + B_{s-2}(x)) + \cdots) + B_1) + B_0).$$

This last representation of operand $B(x)$ is used in Algorithm 1 to compute the field multiplication. That is, $A(x)B(x) \bmod F(x) = x^d(x^d(\cdots(x^d(x^dB_{s-1}(x)A(x) + B_{s-2}(x)A(x)) + \cdots) + B_1A(x)) + B_0A(x)) \bmod F(x)$. At each iteration, the accumulator $C(x)$ is multiplied by $x^d$ and the result is added to the multiplication of $A(x)$ by each word $B_i(x)$ of $B(x)$. The partial result $C(x)$ is reduced modulo $F(x)$.

| | |
|---|---|
| $C(x) = B_{s-1}(x)A(x) \bmod F(x)$ | Initialization |
| $C(x) = x^dC(x) \bmod F(x) = x^dB_{s-1}(x)A(x) \bmod F(x)$ | Iteration $s-2$ |
| $C(x) = x^dB_{s-1}(x)A(x) + B_{s-2}(x)A(x) \bmod F(x)$ | |
| $C(x) = x^dC(x) \bmod F(x) = x^d(x^dB_{s-1}(x)A(x) + B_{s-2}(x)A(x)) \bmod F(x)$ | Iteration $s-3$ |
| $C(x) = x^d(x^dB_{s-1}(x)A(x) + B_{s-2}(x)A(x)) + B_{s-3}(x)A(x) \bmod F(x)$ | |
| ... | ... |

The proposed architecture for Algorithm 1 is shown In the left side of Fig. 1. A finite state machine controls the data flow executing the loop in Algorithm 1. At each iteration, a new digit of $d$ bits from $B(x)$ is processed so the operation is performed in $\lceil d/m \rceil$ cycles. The operations $x^dC(x)$ and $B_i(x)A(x)$ are computed using parallel combinatorial multipliers, that multiplies a $d-1$ grade polynomial with a $m-1$ grade polynomial. Being $U(x)$ a $d-1$ grade polynomial $u_{d-1}x^{d-1} + u_{d-2}x^{d-2} + \ldots + u_1x + u_0$, and $A(x)$ a $m-1$ grade polynomial, the parallel multiplication is

$$\begin{aligned} U(x)A(x) \bmod F(x) = {} & u_{d-1}x^{d-1}A(x) \bmod F(x) \\ & + u_{d-2}x^{d-2}A(x) \bmod F(x) \\ & + \cdots \\ & + u_1xA(x) \bmod F(x) \\ & + u_0A(x) \bmod F(x). \end{aligned}$$

The operation $xA(x) \bmod F(x)$ is a shift to the left operation of $A(x)$ together a reduction of $F(x)$. Thus, the value $x^iA(x) \bmod F(x)$ is the shifted and reduced version of $x^{i-1}A(x) \bmod F(x)$. So each value $x^iA(x) \bmod F(x)$ can be generated sequentially starting with $x^0A(x)$. Finally, each $x^iA(x) \bmod F(x)$ value is added depending on the bit value of $u_i$. These operations are executed by the parallel multiplier shown in the right side of Fig. 1.

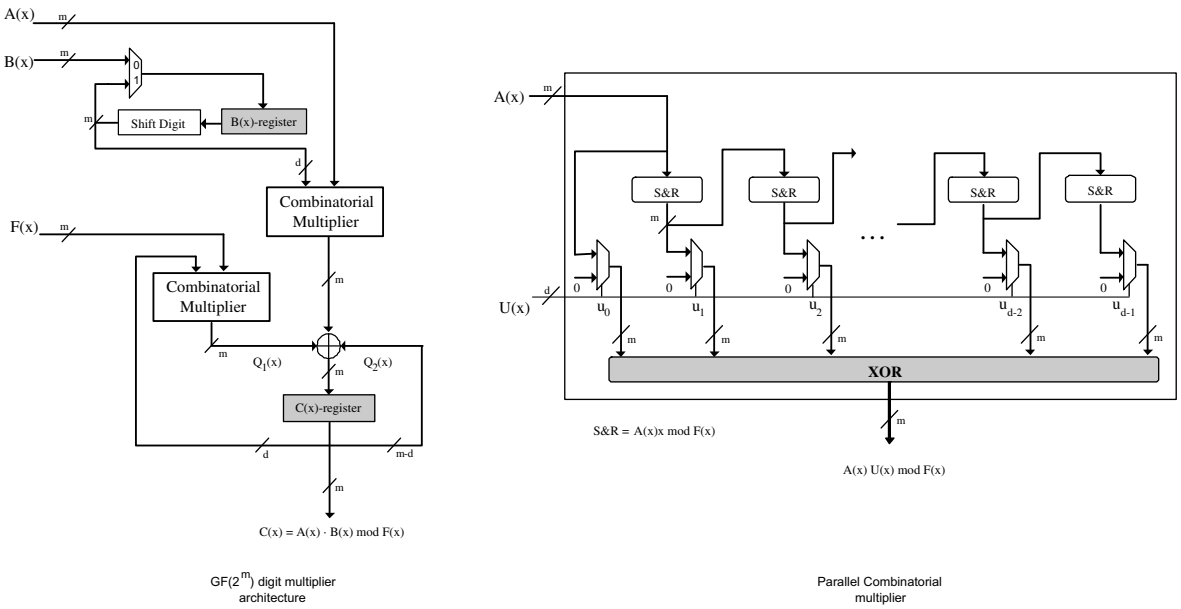


**Fig. 1.** Hardware architecture for digit serial finite field multiplication.

The operation $x^d C(x) \bmod F(x)$ is computed in two steps. Using the polynomial representation of $C(x)$,

$$\begin{aligned} x^d C(x) \bmod F(x) &= x^{\mathrm{d}}(c_{m-1}x^{m-1} + c_{m-2}x^{m-2} + \cdots + c_{m-d}x^{m-d} + c_{m-d-1}x^{m-d-1} + \cdots + c_1 x + c_0) \bmod F(x) \\ &= x^{\mathrm{d}}(c_{m-1}x^{m-1} + c_{m-2}x^{m-2} + \cdots + c_{m-d}x^{m-d}) \bmod F(x) + x^{\mathrm{d}}(c_{m-d-1}x^{m-d-1} + \cdots + c_1 x + c_0) \bmod F(x) \\ &= (c_{m-1}x^{\mathrm{m+d-1}} + c_{m-2}x^{m+d-2} + \cdots + c_{m-d}x^{m}) \bmod F(x) + (c_{m-d-1}x^{m-1} + \cdots + c_1 x^{d+1} + c_0 x^{d}) \bmod F(x) \\ &= Q_1(x) \bmod F(x) + Q_2(x) \bmod F(x). \end{aligned}$$

$Q_2(x)$ is a $m-1$ grade polynomial, corresponding to the $m-d$ least significant bits of $C(x)$ shifted $d$ positions to the left. $Q_2(x)$ does not need to be reduced.

By factoring $x^m$ from $Q_1(x)$, it is obtained $Q_1(x) = x^m(c_{m-1}x^{d-1} + c_{m-2}x^{d-2} + \cdots + c_{m-d})$. In this case, being $F(x)$ a $m+1$ trinomial or pentanomial of the form $F(x) = x^m + g(x)$, where $g(x)$ is a polynomial with grade $g \ll m$, the equivalence $x^m \equiv g(x)$ can be used. In this case, $g(x)$ corresponds to all bits of $F(x)$ except the $m$-bit. Thus, $Q_1(x) \bmod F(x) = g(x)\,(c_{m-1}x^{d-1} + c_{m-2}x^{d-2} + \cdots + c_{m-d})$. That is, the operation is a multiplication of $g(x)$ of grade $g$, and a polynomial of grade $d$, corresponding to the most significant $d$ bits of $C(x)$. The resulting polynomial is of grade $g+d$. In all the cases the polynomial $F(x)$ used in the tests for the finite fields $m \in \{163, 233, 283, 409, 571\}$, and digits $\{1, 4, 8, 16, 32\}$, the value $g + d \ll$ m, so no reduction is necessary. The polynomial $g(x)$ is expanded to a $m-1$ grade polynomial so $Q_1(x) \bmod F(x)$ be computed using the parallel combinatorial multiplier. All these computations are performed by the modules in the architecture for the multipliers, which includes the parallel multipliers, a shift to the left module of $d$-bits, two registers and a $3m$-input xor gate.

## 3. Implementation and results

The architecture was designed in VHDL, simulated and validated using Active-HDL and a test program in C. The architecture is parametrizable in the filed order for any $m$ value. The average system throughput of the architecture was obtained by synthesizing it to several finite field orders for the reconfigurable device xc2v2000 FPGA, using Xilinx's tools. The multiplier was implemented for the field orders $m = 163$, 233, 283, 409 and 571 recommended by NIST [7] for elliptic curve cryptography, and for the field $m = 277$ recommended by IPSec. Due the large number of I/O pins in the architecture, the GF($2^m$) multiplier was implemented together an I/O interface. This is a finite state machine that gets the input parameters $A(x)$ and $B(x)$ as 32-bit words and once the operation is computed, it delivers the results in several 32-bit words. The results presented in figures include the I/O interface.

We also investigated the performance of the multiplier considering the processing time. Fig. 2 shows the processing time for specific finite fields and digit sizes and Fig. 3 shows the area resources required for each one of these finite fields and digit sizes. From these figures, it can be observed that the bigger the digit, the better the performance, but the higher area requirements. Latency of the multiplier is mainly reduced by the size of the digit. From Fig. 2, it is seen that the difference in timing between the digit size 16 and 32 bits is not significant, thus the extra cost in terms of area for digit sizes greater that 32 bits is not justified.

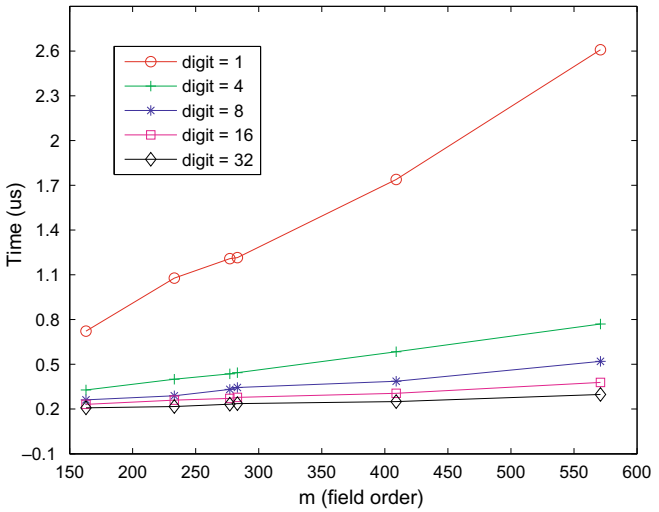


**Fig. 2.** Time ($us$) to compute GF($2^m$) multiplication using different parallelism grade and finite field orders.

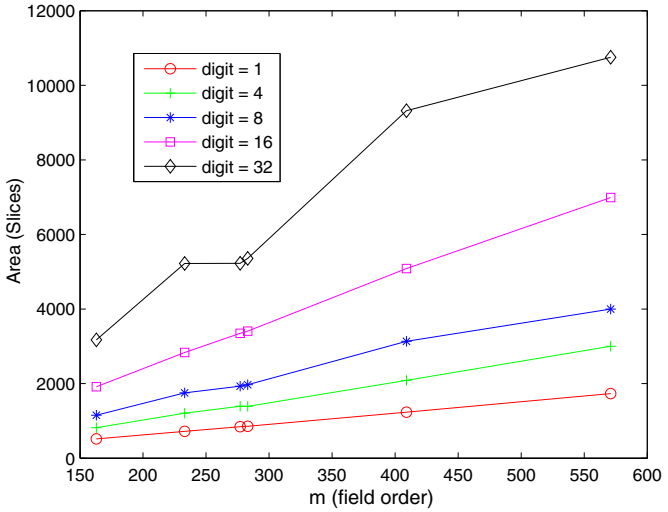


**Fig. 3.** Area (slices) resources for different parallelism grade and finite field orders.

The application constraints will guide the selection of the best implementation parameters. As an example of an application, consider a reconfigurable architecture for scalar multiplication in elliptic curve cryptography that manages several finite field orders, but only assigns fixed space for the field multiplier. For each specific finite field order, there is a digit size that maximizes the performance of the multiplier for a fixed area. For example, with 40 K gates the best performer is a 4-digit field multiplier for the field $m = 571$. In this area, we could also implement a 8-digit or a 16-digit multiplier for the fields $m = 409$ and $m = 277$, respectively, and so on.

It is worth to mention that the results presented in this technical communication were obtained from place and route optimized for speed and without keeping the hierarchical structure of the design. Finally, Table 1 shows a comparison of the area results and performance achieved in this work against the results presented in [8] for several kinds of parallel multipliers, using the field $m = 233$ and the same technology, a virtex2 FPGA xc2v6000-4. In this comparison, the I/O interface was not used. The results show that the digit serial solution requires less area, 10 times lower for $d = 32$ compared to the parallel implementation of the classical multiplier at the cost of six more clock cycles. In all the cases, the digit-serial multiplier has greater frequency which implies this module can be integrated to other designs working at high frequencies.

The multiplier using $d = 1$ achieves better timing (269 MHz, 0.60us) compared with the bit-serial implementation in [9] (42 MHz, 7.4 us) for the finite field $m = 163$. Other works have implemented finite field multipliers and used them in elliptic curve coprocessors or processors [1–5] but the results for the standalone multiplier are not available. Others have implemented the multiplier for the GF($p$) finite field so a direct comparison is not possible [10,11].

**Table 1**
Area and time comparison results

| Ref. | Multiplier | LUT/FF | Slices | Gate count | Clock period (ns) | Frequency (MHz) |
|---|---|---|---|---|---|---|
| [8] | Classical (estmd.) | 37,296/37,552 | – | 528,427 | 13.00 | 77 |
| [8] | HybridKaratsuba | 11,746/13,941 | – | 182,007 | 11.07 | 90.3 |
| [8] | MasseyOmura | 36,857/8,543 | – | 289,489 | 15.91 | 62.8 |
| [8] | SunarKoc | 45,435/41,942 | – | 608,149 | 10.73 | 93.2 |
| This work | ($d = 1$) | 484/477 | 246 | 6731 | 4.26 | 234.8 |
| This work | ($d = 4$) | 1188/634 | 766 | 12,905 | 4.82 | 207.2 |
| This work | ($d = 8$) | 2115/710 | 1384 | 18,394 | 5.32 | 187.7 |
| This work | ($d = 16$) | 4004/889 | 2436 | 31,139 | 6.19 | 161.5 |
| This work | ($d = 32$) | 7110/1349 | 4457 | 53,647 | 6.59 | 151.6 |

## 4. Conclusions

An area/performance trade off analysis for a digit-serial GF($2^m$) finite field multiplication was presented. The size of the digit to use in an application of the proposed multiplier architecture will be guided by the area assigned to the multiplier. Also, the required processing time and which other digits can be used to maximize the performance for other field order using greater digits should be taken into account.

## Acknowledgments

First author thanks the National Council for Science and Technology from Mexico (CONACyT) for financial support through the scholarship number 171577.

## References

[1] Ernest M, Klupsch S, Hauck O, Huss SA. Rapid prototyping for hardware accelerated elliptic curve public key cryptosystems. In: Proceedings of 12th IEEE workshop on rapid system prototyping, RSP'2001, Monterey, CA; June 2001, p. 24–31.
[2] Bednara M, Daldrup M, Gathen J, Shokrollahi J, Teich J. Reconfigurable implementation of elliptic curve crypto algorithms. In: IPDPS'02: Proceedings of the 16th international parallel and distributed processing symposium. Washington, DC, USA: IEEE Computer Society; 2002. p. 284–91.
[3] Ernest M, Jung M, Madlener F, Huss S, Blümel R. A reconfigurable system on chip implementation for elliptic curve cryptographyover GF($2^m$). In: Proceedings of the 4th international workshop on cryptographic hardware and embedded systems – CHES'2002. Lecture notes in computer science, vol. 2523. Redwood Shores, CA: Springer; 2002. p. 381–99.
[4] Saquib N, Rodriguez F, Diaz A. A parallel architecture for fast computation of elliptic curve scalar multiplication over GF($2^m$). In: Proceedings of 11th reconfigurable architectures workshop, RAW'04, Sta. Fe, USA; April 2004. p. 26–7.
[5] Lutz J, Hasan A. High performance FPGA based elliptic curve cryptographic co-processor. ITCC'04: international conference on information technology: coding and computing, vol. 2. IEEE Society Press; 2004. p. 486–92.
[6] Lutz Jonathan. High performance elliptic curve cryptographic co-processor. Master's thesis, University of Waterloo; 2003.
[7] NIST. Recommended elliptic curves for federal government use. <http://csrc.nist.gov/csrc/fedstandards.html>; 1999.
[8] Grabbe C, Bednara M, Teich J, von zur Gathen J, Shokrollahi J. FPGA designs of parallel high performance GF($2^{233}$) multipliers. In: Proceedings of IEEE ISCAS'03, vol. II; 2003. p. 268–71.
[9] Kitsos P, Theodoridis G, Koufopavlou O. An efficient reconfigurable multiplier architecture for galois field GF($2^m$). Microelectron J 2003;34(10).
[10] Savaş E, Tenca AF, Koç ÇK. A scalable and unified multiplier architecture for finite fields GF($p$) and GF($2^m$). Cryptographic hardware and embedded systems, LNCS no. 1965; August 2000. p. 281–96.
[11] Tenca AF, Koc Cetin K. A scalable architecture for modular multiplication based on Montgomery's algorithm. IEEE Trans Comput 2003;52(9):1215–21.