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.

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.