A note about binary finite fields multiplication on FPGA

F. Garcia Crespi, F. Vicedo, R. Gutiérrez , Katya G. Llamazares, P. Garrido, S. Alcaraz
Departamento de Física y Arquitectura de Computadores
Universidad Miguel Hermnandez
Elche, 03202, Spain

A. Grediaga
Universidad de Alicante
Dep. Tecnología Inform,ática y Computación
Ap. 99,Alicante, Spain

J.J. Climent
Dep. Ciencia de la Computació e Inteligencia Artificial
Ap. 99,Alicante, Spain

ABSTRACT
This paper present a notes about a hardware architecture over FPGAs for multiplication in binary fields $GF(2^m)$ using a matrix representation of the elements of $GF(2^m)$.

KEY WORDS
Finite Field Arithmetic, cryptography, finite field multiplication

1 Introduction

Finite fields are increasingly important for many applications in cryptography and algebraic coding theory [5]. Certain properties of the binary finite field $GF(2^m)$ like its "carry-free" arithmetic make it very attractive for hardware implementation. Another advantage of $GF(2^m)$ is the availability of different equivalent representations of the field elements, e.g. polynomial bases, normal bases, or dual bases.

According to the different basis representations, a variety of algorithms and architectures for multiplication in $GF(2^m)$ have been proposed. Efficient implementation of the field arithmetic in $GF(2^m)$ depends enormously on the particular basis used for the finite field.

From an architectural point of view, a polynomial basis multiplier can be realised in a bit-serial, digit-serial, or bit-parallel fashion. For area-restricted devices like smart cards, the bit-serial architecture offers a fair area/performance trade-off.

In this paper we presents a method for multiplication in $GF(2^m)$, where the fields elements are represented as matrices.

2 Finite Fields Arithmetic

2.1 Representation of the Field Elements

Abstractly, a finite field (or Galois field) consists of a finite set of elements together with the description of two operations (addition and multiplication) that can be performed on pairs of field elements. These operations must possess certain properties — associativity and commutativity of both addition and multiplication, distributivity, existence of an additive identity and a multiplicative identity, and existence of additive inverses as well as multiplicative inverses. The order of a finite field is the number of field elements it contains, and it is traditional to denote a finite field of order m as $GF(m)$. $GF(2)$ is the smallest possible finite field; it just contains the integers 0 and 1 as field elements. Addition and multiplication are performed modulo 2, therefore the addition is equivalent to the logical XOR, and the multiplication corresponds to the logical AND.

$GF(2^m)$ is called a *characteristic two* field or a binary finite field. It can be viewed as a vector space of dimension m over the field *GF(2)*. That is, there exist m elements $x_0, x_1, x_2, \ldots, x_{m-1}$ in $GF(2^m)$ such that each element $x \in GF(2^m)$ can be uniquely written in the form: $x = a_0x_0 + a_1x_1 \ldots + a_{m-1}x_{m-1}$ where $a_i \in GF(2)$.

The binary finite field $GF(2^m)$ contains 2^m elements, whereby *m* is a non-zero positive integer. Each of these 2^m elements can be uniquely represented with a polynomial of degree up to *m-1* with coefficients from GF(2). For example, if *a(x)* is an element in $GF(2^m)$, then one can have

$$a(x) = \sum_{i=0}^{m-1} a^i x_i$$

2.2 Addition and Multiplication

Such a set $\{ x_0, x_1, x_2 \ldots x_{m-1}\}$ is called a *basis* of $GF(2^m)$ over *GF(2)*. Given such a basis, a field element x can be represented as the bit string $(a_0a_1 \ldots a_{m-1})$. Addition of field elements is performed by bit-wise XOR-ing the vector representations. The multiplication rule depends on the basis selected. There are many different bases of $GF(2^m)$ over *GF(2)*. Some bases lead to more efficient software or hardware implementations of the arithmetic in $GF(2^m)$ than other bases. The most popular two kinds of bases used are the polynomial bases and the normal bases.

In the polynomial bases, the field arithmetic is implemented as polynomial arithmetic modulo $f(x)$. In

this representation, addition and multiplication of $a(x) = (a_0 a_1 \ldots a_{m-1})$ and $b(x) = (b_0 b_1 \ldots b_{m-1})$

- Field Addition: $a(x) + b(x) = (c_0 c_1 \ldots c_{m-1})$ where $c_i = a_i + b_i$ where $a_i \in GF(2)$
- Field Multiplication: $a(x).b(x) = (r_0 r_1 \ldots r_{m-1})$ where the polynomial $(r_0 + r_1 \ldots r_{m-1})$ is the remainder of the division of the polynomial result of multiply $a(x).b(x)$. That is, $c(x) = a(x).b(x) \pmod{} f(x)$.

Another possibility of representing the elements of $GF(2^m)$ is given by means of matrices. In general, the *companion matrix* of a monic polynomial $f(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + x^n$ of positive degree n over field is defined to be the n x n matrix:

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}$$

It is well known in linear algebra that A satisfies the equation $f(A) = 0$; that is $a_0 I + a_1 A + \ldots + a_{n-1} A^{n-1} + A^n = 0$ whee I is the nxn identity matrix. Thus, if A is the companion matrix of a monic irreducible polynomial $f(x) over F_p of degree n$, then $f(A) = 0$, and therefore A can play the role of root of $f(x)$. The polynomial in $A over F_p$ of degree less than n, yield a representation of the elements of F_p.

With F_p given in this way, calculation in this finite field are then carried out by the usual rules of matrix algebra, the sum and the multiplication decreases to the sum and multiplication of matrices, without necessity of reducing modulo $f(x)$, so we can multiply two elements of the field and the result is an element of the field too.

3 Previous Work

A number of software and hardware implementations have been reported for the computation of $GF(2^m)$ multiplication, which is the basic operation used by elliptic curve cryptographic systems. Among the most significant hardware implementations are [1, 3, 4, 5].

The implementations in [1, 3, 5] use normal basis representation. They use bit-serial multipliers, which require about m clock cycles to compute a multiplication in $GF(2^m)$ and compute squares with cyclic shifts.

The hardware implementation documented in [4] uses standard basis representation. This implementation is suitable for composite fields $GF((2^u)^v)$ where $u \cdot v = m$. Its core-processing element is a hybrid multiplier which computes a multiplication in v clock cycles.

3.1 Classical Methods for Multiplication

Multiplication using the polynomial representation of field elements is somewhat more difficult. To compute $a(x).b(x) \pmod{} f(x)$, we take the remainder $r(x)$ of the product $(a_0 + a_1 x + \ldots + a_{m-1} x^{m-1}).(b_0 + b_1 x + \ldots + b_{m-1} x^{m-1})$ when divided by $f(x)$, so first of all, we have $c(x) = a(x).b(x)$ and then the remainder of the division of $c(x)$ by $f(x)$.

Algorithm 1 in Handel-C:

```
unsigned int 1 A[16],B[16],C[16];
unsigned int i,j,m;

    m = 16;
    for( i=0; i<m; ++i)
      for( j=0; j<m; ++j)
         C[i+j] += A[i]*B[j];

// then we get the remainder of c(x)/f(x)
```

3.2 Bit-Serial The Shift-and-Add Method

Multiplication of field elements uses the same shift-and-add algorithm as is used for multiplication of integers, except that the 'add' is replaced with 'xor'. Algorithm 2 in Handel-C:

```
unsigned int 16 A,B,C,F;
unsigned int 16 IGUAL1[17];
unsigned int i,j;

A = 12;
B = 23;
F = 5;
C = 0;

//for(i=0;i<=7;i++) IGUAL1[i] = 1<<i;

for(i=0;i<=7;i++) {
if(A & IGUAL1[i]) C = C ^ B;
B = B << 1;
}
```

- Addition and subtraction in $GF(2)$ is a logical XOR.
- Multiplication consists of logical AND, logical XOR and 1-bit left-shift operations.

4 Proposed method

In this section we describe a reconfigurable hardware implementation of an algorithm for multiplication in $GF(2^m)$ using the matrix representation explain in 2 and Handel-C

4.1 Reconfigurable Hardware

Usually, the measure of the performance for hardware implementations of the arithmetic operations in the Galois field GF(2m) is the space and time complexities.

- Main performance criteria
- Space complexity
- Number of AND gates
- Number of XOR gates
- Time complexity
- Circuit's total gate delay

This work propose a hardware architecture over FPGAs for multiplication in binary fields $GF(2^m)$. One of its key features is its suitability for reconfigurable hardware. Unlike traditional VLSI hardware, reconfigurable devices such as Field Programmable Gate Arrays (FPGA) do not possess fixed functionality after fabrication but can be reprogrammed during operation.

4.2 DK1 Design Suite and Handel-C

Handel-C is a high level programming language designed to enable compilation of programs directly in to hardware, which is implemented on Field Programmable Gate Array (FPGA). Handel-C is a programming language designed to enable the compilation of programs into synchronous hardware. Handel-C is not a hardware description language though; rather it is a programming language aimed at compiling high level algorithms directly into gate level hardware. Handel-C is based on the syntax of conventional C (ANSI C), with additional extensions to take advantage of the features of hardware. As Handel-C is high level it is possible to convert a Software Algorithm in to a hardware implementation with the greatest of ease.

If Hardware Definition Languages are thought of as the Assembly Language equivalent in Hardware Design. Then Handel-C represents the equivalent of a High Level Language in Hardware Design. The Handel-C compiler and some detailed examples of its usage are described in the Handel-C Compiler Reference Manual.

Handel-C provides special constructs, which enable expressions to be evaluated in parallel. It also provides the ability to specify the width of a data variable.

```
int 3 a,b;
// a and b are integers of 3 bit
```

Sequential Expressions (this executes the two statements, one after the other sequentially)

```
.....
a = 1;
b = 2;
.....
```

Parallel Expressions (this executes both statements in parallel)

```
par {
a = 1;
b = 2;
}
```

The Celoxica DK1 design suite is a unique C direct-to-hardware solution that enables software engineers to migrate concepts directly to hardware without requiring the generation, simulation, or synthesis of hardware description languages. The DK1 design suite focuses on the design, validation, iterative refinement and implementation of complex algorithms in hardware. It includes built-in design entry, simulation and synthesis, driven directly by Handel-C. The output of the compiler is an architecture optimised EDIF netlist for FPGA's, or RTL VHDL for existing tool suites.

More information about Handel-C and DK1 in [2].

4.3 The Method

The matrix product $C = AB$ is formed by multiplying every row of A with every column of B, in the way described above. The resulting numbers are arranged in a new matrix: the m_{th} row in A times the n_{th} column in B gives the number at position (m, n) in AB.

Algorithm 3 in Handel-C:

```
set clock = external "P35";

/*
 * Function main
 */
void main(void)
{
// A, B and C are
// matrix_{4x4} of elements of one bit

unsigned 1 A[4][4];
unsigned 1 B[4][4];
unsigned 1 C[4][4];

A[0][0] = 1; A[1][0] = 1;
A[0][1] = 0; A[1][1] = 1; //....;
B[0][0] = 0; B[1][0] = 1;
B[0][1] = 1; B[1][1] = 0; //....;

par {
      C[0][0]=A[0][0]&B[0][0]
      |A[0][1]&B[1][0]
      |A[0][2]&B[2][0]
      |A[0][3]&B[3][0]
        .....
      C[3][3]=A[3][0]&B[0][3]
      |A[3][1]&B[1][3]
      |A[3][2]&B[2][3]
```

```
      |A[3][3]&B[3][3]
}
// & is the AND operation and
   | is the XOR operation
}
```

This algorithm reduce the CPU time to 2 cycles.

5 Results

Using Celoxica DK1, we implemented the three algorithms, and in debug mode simulates them. The result is the number of nand gates using by the algorithms.

	alg1		alg2		alg3	
m	cycles	gates	cycles	gates	cycles	gates
2	8	660	2	341	2	28
4	16	1350	4	523	2	98
8	32	2666	8	975	2	384
16	64	4486	16	4499	2	2866

Table 1. Results

6 Conclusion

We presented an architecture for parallel multiplication in the binary field $GF(2^m)$. However, in the literature we have not found anything paper adopting a technique similar to ours for performing finite field multiplication using Handel-C.

As we can see in Table 1, the first algorithm use $4m$ clock cycles and the second algorithm use m clock cycles. The third multiplier algorithm is fastest, so the utilization of this multiplier in a cryptography co-processors would increase their speed of operation.

References

[1] G.B. Agnew, R.C. Mullin, and S.A. Vanstone. An implementation of elliptic curve cryptosystems over F2155 . IEEE Journal on Selected Areas in Communications, 11(5):804-813, June 1993.

[2] Handel-C and DK1 manual available at http://www.celoxica.com

[3] [GSS99] L. Gao, S. Shrivastava, and G. Sobelman. Elliptic curve scalar multiplier design using FPGAs. In C. Koc and C. Paar, editors, Workshop on Cryptographic Hardware and Embedded Systems (CHES '99), volume LNCS 1717. Springer-Verlag, August 1999.

[4] M. Rosner. Elliptic curve cryptosystems on reconfigurable hardware. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1998.

[5] S. Sutikno, R. Enendi, and A. Surya. Design and implementation of arithmetic processor F2155 for elliptic curve cryptosystems. In the 1998 IEEE Asia-Pacific Conference on Circuits and Systems, pages 647-650, November 1998.