A Fast Finite Field Multiplier*

Edgar Ferrer[1], Dorothy Bollman[2], and Oscar Moreno[3]

[1] PhD. CISE Program, University of Puerto Rico, Mayagüez, PR 00681
eferrer@cs.uprm.edu
[2] Department of Mathematical Sciences,
University of Puerto Rico, Mayagüez, PR 00681
bollman@cs.uprm.edu
[3] Department of Computer Science,
University of Puerto Rico, Rio Piedras, PR 00931
moreno@uprr.pr

Abstract. We present a method for implementing a fast multiplier for finite fields $GF(2^m)$ generated by irreducible trinomials of the form $\alpha^m + \alpha^n + 1$. We propose a design based on the Mastrovito multiplier which is described by a parallel/serial architecture that computes a multiplication in m clock cycles by using only bit-adders (XORs), bit-multipliers (ANDs), and shift registers. This approach exploits symmetries and subexpression sharing in Mastrovito matrices in order to reduce the number of operations, and hence computation time in our FPGA implementation. According to preliminary performance results, our approach performs efficiently for large fields and has potential for a variety of applications, such as cryptography, coding theory, and the reverse engineering problem for genetic networks.

1 Introduction

A Field-Programmable Gate Array or FPGA is a silicon chip containing an array of configurable logic blocks (CLBs). Over the last years new technologies have been developed in order to increase speed and the amount of CLBs in FPGAs, therefore they are becoming more feasible for implementing a wide range of applications. Nowadays FPGAs are becoming attractive in the High Performance Computing (HPC) community, because they can accelerate critical segments in high performance applications. HPC hardware vendors have begun to offer solutions that incorporate FPGAs into HPC systems where they can act as co-processors, accelerating key kernels within an application.

This work is concerned with accelerating finite field arithmetic in a high performance application by using FPGAs. Our motivation is an important problem in computational biology: the reverse engineering problem for genetic networks that are modeled by finite fields [1],[3]. However, our work has applications in other areas such as cryptography and error correction coding. These applications rely on intensive arithmetic computations over finite fields. Addition and

* This research is supported by grant NIH-MBRS (SCORE) S06-GM08102.

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 238–246, 2007.

multiplication are two basic operations. Addition is easily realized at very low computational cost, but other arithmetic operations on finite fields such as inversion, squaring, exponentiation and divisions are typically performed by repeated multiplications.

2 Background

An element in the finite field $GF(2^m)$ can be represented as a sequence of m bits in $GF(2)$ describing the coefficients of a binary polynomial. This representation is useful for manipulating finite field elements via bitwise operations, so we can exploit the hardware architecture of computers by carrying out finite field arithmetic by means of bit-level operations. In essence, the arithmetic computation over $GF(2^m)$ is suitable for FPGA implementations. We take advantage of reconfigurable hardware resources with the aim of accelerating computations considerably.

Multiplication is an expensive operation. Many solutions have been proposed for efficient multiplication over finite fields. Solutions are based on purely software approaches, purely hardware approaches, and more recently, on hardware/software using reconfigurable computing.

The representation of the field elements distinguishes the particular features of a finite field multiplier. The most common representations are dual basis, normal basis, and standard basis. In this work we deal with finite field elements represented in standard (or polynomial or canonical) basis, such that the finite field $GF(2^m)$ consists of a finite set of all binary polynomials of degree less than m. For example $GF(2^2) = \{0, 1, \alpha, \alpha + 1\}$, where α is a root in $GF(2^2)$ of the irreducible polynomial $\alpha^2 + \alpha + 1$.

A very natural approach for standard basis multiplication in $GF(2^m)$ is to multiply two elements in the field as polynomial multiplication modulo an irreducible polynomial. This operation is typically accomplished in two stages: polynomial multiplication and modular reduction.

Let $A(\alpha)$, $B(\alpha)$, $C(\alpha)$ elements in $GF(2^m)$ and $f(\alpha)$ the irreducible polynomial generating $GF(2^m)$. Then the finite field multiplication $C(\alpha) = A(\alpha)B(\alpha)$ is accomplished by calculating

$$C(\alpha) = A(\alpha) * B(\alpha) \mod f(\alpha) \tag{1}$$

where $*$ denotes polynomial multiplication. In a first stage the product $A(\alpha) * B(\alpha)$ is calculated, resulting in a polynomial $Q(\alpha)$ of degree at most $2m - 2$.

$$Q(\alpha) = A(\alpha) * B(\alpha) = \left(\sum_{i=0}^{m-1} a_i \alpha^i \right) \left(\sum_{i=0}^{m-1} b_i \alpha^i \right) \tag{2}$$

In a second stage the modular reduction is performed on $Q(\alpha)$, that is, $C(\alpha) = Q(\alpha) \mod f(\alpha)$, resulting in the polynomial $C(\alpha)$ of degree at most $m - 1$.

It is easy to show that the expansion of equation (2) can be expressed as a matrix-vector product $Q = MB$, where Q is a vector of dimension $2m - 1$,

which consists of the coefficients of $Q(\alpha)$. In the same way B is an m dimensional vector which consists of the coefficients of $B(\alpha)$, while the $(2m-1) \times m$ matrix M involves coefficients of $A(\alpha)$ (see for example [11]).

Notice that the last $m-1$ components of the vector Q (i.e. $[q_m, \ldots, q_{2m-2}]$) contain terms with degree greater than $m-1$. These terms must be reduced modulo the irreducible polynomial $f(\alpha) = \alpha^m + g(\alpha)$ in order to express them as polynomials in the field $GF(2^m)$. This reduction is obtained by using the reducing identity $\alpha^m = g(\alpha)$, so all the terms with degree greater than $m-1$ will be reduced to terms with degree in the proper range $[0, m-1]$. Each reduced term is added to the respective terms in $[q_0, \ldots, q_{m-1}]$, and so we get $C(\alpha)$. A particular term may need to be reduced several times. The maximum number of reductions is determined by:

$$N[m, n] = \left\lceil \frac{m-1}{\Delta} \right\rceil$$

where $\Delta = m - n$ [6].

For example, let $m = 3$ and $f(\alpha) = \alpha^3 + \alpha^2 + 1$, thus $\alpha^3 = \alpha^2 + 1$ and $\alpha^4 = \alpha^3 + \alpha$. Using these identities the term $q_3\alpha^3$ is reduced only once: $q_3\alpha^3 = q_3\alpha^2 + q_3$, while $q_4\alpha^4$ is reduced twice: $q_4\alpha^4 = q_4\alpha^3 + q_4\alpha = q_4\alpha^2 + q_4\alpha + q_4$, and so we get $C(\alpha) = q_4\alpha^4 + q_3\alpha^3 + q_2\alpha^2 + q_1\alpha + q_0 = (q_4 + q_3 + q_2)\alpha^2 + (q_4 + q_1)\alpha + (q_4 + q_3 + q_0)$. Notice that the maximum number of reductions is $N[3, 2] = 2$.

An alternative to the two-stage method, described above, for computing C is to perform the reduction directly on the matrix M, obtaining an already reduced $m \times m$ dimensional matrix Z, such that $C = ZB$. Z is called the Mastrovito matrix [8].

3 Multiplication Algorithm

A common approach to the design of multipliers that are based on the Mastrovito matrix Z is to compute Z and then do the multiplication in $GF(2^m)$ by means of matrix-vector multiplication. In our approach, we exploit the symmetry of Z without actually computing Z.

A method for constructing the Mastrovito matrix is proposed in [11], [6]. According to this method if $GF(2^m)$ is defined by the trinomial $\alpha^m + \alpha^n + 1$ then Z is given by

$$Z = \begin{bmatrix} U \\ L \end{bmatrix}$$

where U and L are Toeplitz matrices defined as follows:

Let $F = [\, 0 \; a_{m-1} \; a_{m-2} \; \ldots \; a_1 \,]$ and for each $i = 0, 1, \ldots, m-1$, let $F[i \rightarrow]$ be the result of shifting F i positions to the right (vacated positions on the left are filled with zeros). Also let $G = [\, a_n \; a_{n-1} \; \ldots \; a_1 \; a_0 \; a_m \; \ldots \; a_{n+1} \,]$

U is $n \times m$, its first column is $[\, a_0\ a_1\ \ldots\ a_{n-1}\,]^T$, and its first row is

$$[a_0]|| \sum_{i=0}^{N-1} F[i\Delta \rightarrow]$$

where $\Delta = m - n$, $||$ represents concatenation, and N is a short notation for $N[m, n]$.

L is $\Delta \times m$, its first column is $[\, a_n\ a_{n+1}\ \ldots\ a_{m-1}\,]^T$, and its first row is

$$G + \sum_{i=0}^{N-1} F[i\Delta \rightarrow]$$

Although the previously described method is used for constructing the entire Mastrovito matrix Z, in this work we construct only one row of Z which is sufficient in our approach for carrying out multiplications in $GF(2^m)$. By constructing the n-th row Z_n (where rows are numbered $0, 1, \ldots$), the remaining rows of Z can be obtained by means of right-shifts and concatenations over Z_n.

Example: If $GF(2^7)$ is defined by $\alpha^7 + \alpha^4 + 1$, then $\Delta = 3$, $N = 2$, and

$$G = [\, a_4\ a_3\ a_2\ a_1\ a_0\ a_6\ a_5\,]$$

$$\sum_{i=0}^{N-1} F[i\Delta \rightarrow] = F + F[\Delta \rightarrow] = [\, 0\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\,] + [\, 0\ 0\ 0\ 0\ a_6\ a_5\ a_4\,]$$

and so L_0 is

$$Z_4 = [\, a_4\ a_3 + a_6\ a_2 + a_5\ a_1 + a_4\ a_0 + a_3 + a_6\ a_6 + a_2 + a_5\ a_5 + a_1 + a_4\,]$$

The construction method previously described allows us to devise a template for the n-th row of the Mastrovito matrix(Z_n), which needs to be pre-computed as a first step in the multiplication algorithm (see Algorithm 1). In the for-loop, one output bit of C is obtained in each cycle by multiplying (via inner product) the current row Z_i by B. Each row is obtained as a result of right-shifting the previous row and filling the vacated position on the left with a_i.

Algorithm 1

Input: $A(\alpha), B(\alpha)$,template for Z_n; $A(\alpha), B(\alpha) \in GF(2^m)$
Output: $C(\alpha) = A(\alpha)B(\alpha)$; $C(\alpha) \in GF(2^m)$

Compute Z_n
$S \leftarrow Z_n$
for $i = 0$ to $m - 1$
  $c_{(i+n) \mod m} \leftarrow S \cdot B$
  $S \leftarrow$ right-shift(S)
  $s_0 \leftarrow a_i$
end for
return(C)

4 FPGA Implementation

The proposed multiplier is implemented in a parallel/serial architecture, which computes a multiplication in m clock cycles. One output bit of C is obtained in each cycle by multiplying (inner product) the current row Z_i by B, thus achieving the matrix-vector product C = ZB. According to this method, the finite field multiplication is carried out by means of m inner products. Hence, inner product is the main operation in our finite field multiplier

Since all the inner products are performed over fields of characteristic 2, they could be done by means of FIR (Finite Impulse Response) filters. FIR filters are widely used in various Digital Signal Processing (DSP) applications. A comprehensive treatment of the fundamentals of FIR filters and FPGA implementations can be found in [9].

A traditional architecture of a FIR filter is shown in figure 1. The m-bit input is shifted through m bit-registers (known as taps). Each output stage of a particular register is multiplied by a known factor. The resulting outputs of the multipliers are then summed to produce the filter output. A conventional FIR filter implementation consists basically of multiplication units and summation units. Inner product operations are carried out by a Multiply-and-Accumulate model, which compute and accumulate the binary partial products in a bit-register. This implementation requires m multiplications and $m - 1$ additions to compute an inner product over an m-bit input. Thus, m cycles are required before the next inner product can be processed.

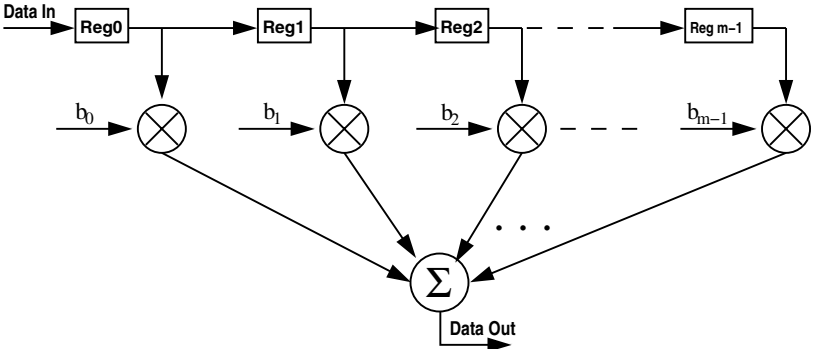

Fig. 1. An m-Tap FIR filter traditional architecture

Instead of using the aforementioned Multiply-and-Accumulate model, in this work we use a Multiply-and-Add design. By using this approach, two bit sequence can be multiplied in parallel, and afterwards, the sum of the resulting bit sequence has to be computed. This parallel implementation can speed-up the performance of the inner product. Here parallel multiplication is possible because the input vectors, namely, the current row of the Mastrovito matrix Z_i

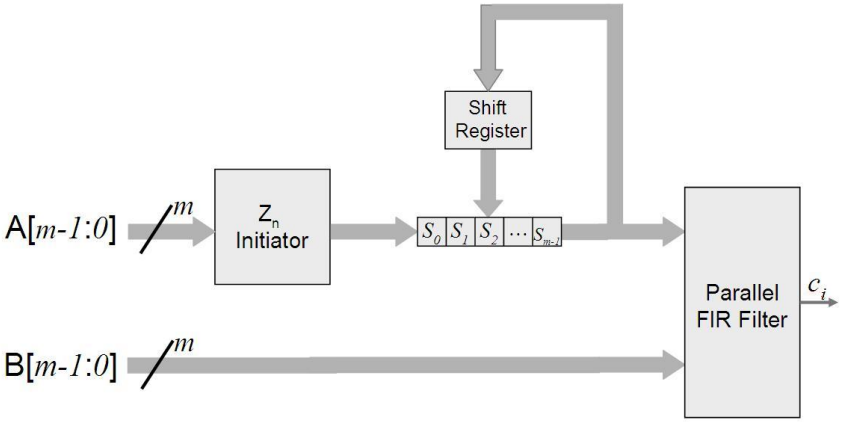

Fig. 2. Block diagram of the proposed multiplier

and the field element B are accessible at the same time. With this approach each inner product can be completed in one clock cycle.

In addition to the inner product, the entire multiplier implementation also includes the action of a shift register in each clock cycle, as is shown in figure 2. The initial row Z_n already includes the reductions required for the finite field multiplication, thus avoiding the modular reduction stage in the finite field multiplier. A template for Z_n is precomputed and then hardwired, taking advantage of subexpression sharing in order to reduce the number of operations.

One output bit of C is obtained in each cycle by multiplying (inner product performed by a FIR filter) the current row stored in register S by B. The current row is obtained as a result of right-shifting the previous row and filling the vacated position on the left with a_i.

5 Experimental Results

In this section we present a performance comparison between our finite field multiplier and other efficient FPGA implementations of finite field multiplication over $GF(2^m)$. Each of these multipliers represent elements in the polynomial basis.

Naturally, a finite field multiplication consists of a polynomial multiplication followed by a modular reduction. In [4], an efficient multiplier architecture of the type serial/parallel is presented where the modular reduction is made concurrently over each partial product, and finally all the partial products are added to obtain the final result. A similar, but more flexible architecture, is proposed in [7], where the value of the field degree can be changed and the irreducible polynomial can be configured and programmed; this feature can be achieved by implementing demultiplexers in the architecture design. In [5], the authors consider a hybrid-Karatsuba multiplier based on the Karatsuba multiplication

method which reduces the number of multiplication but at the cost of increasing the number of additions and the total propagation delay. To achieve a tradeoff between area and propagation delay, a hybrid model using Karatsuba formulas combined with the classical polynomial multiplication method is proposed.

In Table 1 we compare our approach with the mentioned polynomial basis multipliers reported in [4,5,7]. The field sizes used in this experiment are the same as those used in the cited references, the only suitable benchmarks for comparisons that are known to us. However, our approach can be implemented for larger finite fields.

The works cited in Table 1 are implemented on different platforms. We have synthesized our implementation over the same target devices used in the cited references for the purpose of comparison. However our work is focused on accelerating finite field arithmetic in a high performance hardware/software environment. Our target platform is a Cray XD1 system which includes six FPGAs units tightly integrated to 12 2.2 GHz Opteron AMD processors through a high bandwidth interconnection system. FPGA units are Xilinx Virtex II-Pro xc2vp50-7.

The times in Table 1 have been measured using FPGA synthesis results reported by Xilinx tool XST (Xilinx Synthesize Technology) included in the package ISE Foundation 7.1. Our implementations are synthesized without area and timing constraints.

Table 1. Comparison of multipliers

Field	Target FPGA	Implementation	Time (μs)	Space (slices)
$GF(2^{210})$	Xilinx Virtex xcv-300-6	Reference [7]	12.30	343
		This work	2.21	334
$GF(2^{233})$	Xilinx xc2v-6000-4	Reference [5]	2.58	not reported
		This work	2.42	415
$GF(2^{239})$	Xilinx Virtex xcv-300-6	Reference [4]	3.10	359
		This work	2.47	385

According to the given results, our implementation exhibits the best time performance, whereas the area is not the most favorable for some cases. However our main goal is to achieve very fast computation using reasonably the physical devices.

Higher acceleration rates are obtained using the Cray XD1 FPGA (see Table 2). According to our results, there are significant opportunities for speed up on the Cray XD1 using reasonably the FPGA's physical space, however the communication time between CPU and FPGA becomes an obstacle. The communication model that we have used is a simple push-model in which the CPU pushes the input data to the FPGA's registers, and reads the output data from a destination register on the FPGA. Our experimental results indicate that this is a costly communication model, for example the direct multiplier for $GF(2^{63})$ spent 2.77 μs for communications and 0.62 μs for computations. Other works such as [2] have reported similar communication problems with the Cray XD1.

However, in [10] it is shown that a savvy decision about the workload assigned to the FPGA can vastly improve the communication performance on the Cray Hyper-transport I/O bus over other communication interfaces technologies.

Table 2. Comparison of multipliers on the Cray XD1 FPGA

Field	Time (μs)	Space (slices)	Space Utilization
$GF(2^{210})$	1.85	305	1.29%
$GF(2^{233})$	2.02	369	1.56%
$GF(2^{239})$	2.04	363	1.53%

6 Conclusions and Future Work

We have presented a high performing FPGA implementation of a $GF(2^m)$ multiplier. The structure of our multiplication algorithm, has allowed us to enhance performance by exploiting Mastrovito matrix symmetries while avoiding modular reductions in the multiplication process. The main operation in our multiplier is the inner product which is accelerated by bit-level parallelism, this simple architecture uses a small amount of space, and does not present opportunity for performance improvement in terms of alternative implementations.

Although our approach has shown to be fast for the finite fields reported in the previous section, it promises more favorable results for multipliers on larger fields. In order to deal with high performance reconfigurable applications such as reverse engineering genetic networks, our FPGA implementation could be used as a co-processor for accelerating a CPU-based application. This requires a judicious partitioning of the problem between high performance CPUs and FPGAs. Here the communication overhead is an important issue and we have to consider alternative ways of communication in order to improve the overall performance. The most promising approach is to shift more of the computational burden to FPGAs by embedding our multiplier into a more complex FPGA-based high performance application. This is possible since our implementation uses only from 1 to 2 percent of the FPGA area. Communication overhead can also be reduced by taking advantage of FPGA Transfer Region of Memory using the I/O subsystem proposed by [2].

References

1. D. Bollman, E. Orozco, O. Moreno, "A Parallel Solution to Reverse Engineering Genetic Networks", in Gavrilova et al (eds), Lecture Notes in Computer Science, Springer-Verlag, Part III, 3045, pp. 490-497, 2004.
2. J.Fernando, D. Dalessandro, A. Devulapalli, and A. Krishnamurthy, "Enhancing FPGA Based Encryption", Ninth Workshop on High Performance Embedded Computing (HPEC). Sept. 2005.

3. E. Ferrer, D. Bollman, and O. Moreno, "Toward a Solution of the Reverse Engineering Problem Using FPGAs," to appear in Proceedings of the International Euro-Par Workshops, Lecture Notes in Computer Science, Springer-Verlag, Dresden, Germany, 2006.
4. M.A. Garcia-Martinez, R. Posada-Gomez, G. Morales-Luna, F. Rodriguez-Henriquez. "FPGA implementation of an efficient multiplier over finite fields $GF(2^m)$", Proceedings of International Conference on Reconfigurable Computing and FPGAs, 2005 (ReConFig'05), September 2005.
5. C. Grabbe, M. Bednara, J. Shokrollahi, J. Teich and J. Von Zur Gathen, "FPGA Designs of parallel high performance Multipliers", Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS-03), volume II, 268-271. Bangkok, Thailand.
6. A. Halbutogullari, Ç. Koç, "Mastrovito Multiplier for General Irreducible Polynomials", IEEE Transactions on Computers, volume 49, number 5, pp. 503-518, 2000
7. P. Kitsos, G. Theodoridis, and O. Koufopavlou, "An efficient Reconfigurable Multiplier Architecture for Galois Field $GF(2^m)$", Microelectronics Journal, volume 34, pp.975-980, 2003.
8. E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields", PhD thesis, Dept. of Electrical Eng., Linkvping Univ., Linkvping, Sweden, 1991.
9. U. Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays", Second Edition. Springer Verlag, Berlin, 2004
10. D. Strenski, "Computational Bottlenecks and Hardware Decisions for FPGAs". FPGA and Programmable Logic Journal, Nov 14, 2006.
11. B. Sunar and Ç. K. Koç "Mastrovito Multiplier for All Trinomials", IEEE Transactions on Computers", volume 48, number 5, pp. 522-527, May 1999.