Toward a Solution of the Reverse Engineering Problem Using FPGAs*

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 Mathematics, 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. An important issue in computational biology is the reverse engineering problem for genetic networks. In this ongoing work we consider reverse engineering in the context of univariate finite fields models. A solution to the reverse engineering problem using multipoint interpolation relies on intensive arithmetic computations over finite fields, where multiplication is the dominant operation. In this work, we develop an efficient multiplier for fields $GF(2^m)$ generated by irreducible trinomials of the form $\alpha^m + \alpha^n + 1$. We propose a design described by a parallel/serial architecture that computes a multiplication in m clock cycles. This approach exploits symmetries in Mastrovito matrices in order to improve time complexities of an FPGA (Field Programmable Gate Array) implementation. According to preliminary performance results, our approach performs efficiently for large fields and has potential for an efficient solution of the reverse engineering problem for large genetic networks, as well as other finite fields applications such as cryptography and Reed-Solomon decoders.

1 Introduction

An important problem in computational biology is modeling gene regulatory networks in order to determine gene behavior in biological systems and how they interact with each other. The reverse engineering problem for genetic networks is the problem of determining the network that describes functional relations between genes, given a set of experimental data.

In this ongoing work we consider the reverse engineering problem in the context of univariate finite fields models [1,12]. The reverse engineering problem can then be stated more precisely as follows:

Given a time series $s_0, s_1, \ldots, s_{k-1}$ of measurements of gene expression data representing the states of m genes at times $t_0, t_1, \ldots t_{k-2}$, and a set of conditions χ, the reverse engineering problem is the problem of finding a function f such

* This research is supported by grants NSF-CISE EIA-0080926 and NIH-MBRS (SCORE) S06-GM08102.

W. Lehner et al. (Eds.): Euro-Par 2006 Workshops, LNCS 4375, pp. 304–312, 2007.

that $f : GF(q) \rightarrow GF(q)$ has the property that $f(s_j) = s_{j+1}$, where $s_j = (a_0, a_1, \ldots, a_{m-1})$, and f satisfies the conditions in χ.

Our solution $f(x)$ to the reverse engineering problem then involves the determination of a polynomial $P(x)$, such that $f(x) = P(x) + g(x)$, and $P(s_i) = P(s_{i+1})$, and $g(x)$ is a polynomial such that $g(s_i) = 0$, for $i = 0, 1, \ldots, k-2$. The polynomial $P(x)$ can be determined interpolating over the points s_i. Once having determined $P(x)$, the polynomial $g(x)$ can be used to adjust the model in order to satisfy the conditions in χ.

A classical method such as Lagrange interpolation formula can be used, but it has computational complexity $O(n^2)$, where n is the number of points to be interpolated. In contrast, Lipson's algorithm has complexity $O(n \log^2 n)$. Bollman *et al* [1] have shown that a parallel version of this algorithm is efficient for reverse engineering univariate genetic networks.

2 Dealing with Large Genetic Networks

Various approaches have been taken for modeling gene regulatory networks, including linear models, Bayesian networks, neural networks, nonlinear ordinary differential equations, stochastic models and Boolean models. One model that has received considerable attention is the Boolean model e.g., [6]. Recently, several researchers have pointed out advantages in generalizing the Boolean model to finite fields. Two types of finite field models have emerged, a multivariate model as described by Laubenbacher [8] et al and a univariate model as described by Moreno et al [12], [1]. The multivariate model gives local information at each gene, whereas the univariate model gives global information about the network.

In a finite field model for genetic networks we assume that gene expression is discretized so that there are a prime number p of levels. There are several ways to discretize the real-valued microarray data. One way is by thresholding. Another way is to normalize gene expressions and use the deviation from the mean to discretize the data. Inconsistencies due to either noise or biological variance can be resolved by using information theoretic error correction [13].

If there are m genes and p levels of expression, then there are p^m states and we model such a network by the elements of $GF(p^m)$. In this work we consider univariate finite field models with $p = 2$, so that each gene assumes just two states, either *on* or *off*. Thus we restrict interpolation to fields $GF(2^m)$.

In practice, m can be quite large. For example, [10] outlines a study of gene regulatory networks in yeast. Yeast has 6000+ genes. This study includes a subset of 106 transcription factors and 2343 genes for which strong empirical evidence of interaction was found using the experimental technique outlined in the paper. Advances in techniques should yield data on all 6270 genes in yeast, and eventually similar data will be available for all 20,000+ human genes. It is thus of vital interest to develop algorithms to reverse engineering very large networks.

A solution to the reverse engineering problem for large values of m using multipoint interpolation relies on intensive arithmetic computations over finite fields. Addition and multiplication are two basic operations. Addition is easily realized at very low computational cost, but multiplication is costly in terms of computation time and circuit complexity. Other arithmetic operations on finite fields used for reverse engineering such as inversion, squaring, exponentiation and divisions are performed by repeated multiplications. Thus, in order to solve the reverse engineering problem for the very large genetic networks that biologists would like to consider, it is essential to develop capacity for performing fast and efficient arithmetic over very large finite fields, especially multiplication.

One very fast method for performing arithmetic on $GF(2^m)$ involves the use of Zech logarithm tables. By using lookup tables we can perform arithmetic operations at "almost no cost", but the memory space becomes a great limitation. For instance, a 32-bit word length for storing the elements of $GF(2^{30})$ in a table, requires $2^2 \cdot 2^{30}$ bytes $= 4$ GB in main memory. This method is efficient for small finite fields, but it is not practical for the large fields that arise in real reverse engineering problems.

A natural approach for multiplication in $GF(2^m)$ is to multiply two elements in the field as polynomial multiplication modulo a m-degree irreducible polynomial over $GF(2)$. This operation is accomplished by simply using left-shifts and exclusive or's. In this simple procedure (also known as the direct or classical method) the memory space is not a limitation, but in a basic CPU based implementation the field size is limited by the architecture word-length.

One approach to addressing large finite fields is to use composite fields $GF((2^r)^s)$, combining lookup tables with direct multiplication. The acceleration of finite fields multiplication using FPGAs is determined mainly by access time between FPGAs and memory, since a composite field multiplication requires multiple accesses to lookup tables stored in memory.

In this work, we present an FPGA-based implementation of a multiplication algorithm over $GF(2^m)$ that exploits the symmetries in the Mastrovito matrix [11]. The proposed approach performs efficiently for large fields and has potential for efficient solutions of the reverse engineering problem for large genetic networks, as well as other finite fields applications such as cryptography and Reed-Solomon decoders.

3 Finite Field Multiplication

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 FPGAs implementations. We take advantage of reconfigurable hardware resources with the aim of accelerating computations considerably.

Multiplication is the dominant operation in the interpolation phase of the solution of the reverse engineering problem for genetic networks. 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 a 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 [14]).

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 $\Delta = m - n$ [5].

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 over the matrix M, obtaining an already reduced $m \times m$ dimensional matrix Z, such that $C = ZB$. Z is called the Mastrovito matrix [11].

4 A New FPGA-Based Approach

A common approach to the design of multipliers that is 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 [5]. 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 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, the current row is obtained by right-shifting the previous row and filling the vacated position on the left with a_i. Algorithm 1 shows this process.

Algorithm 1

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

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

5 Experimental Results

This research is focused on accelerating finite field arithmetic using FPGAs for efficient solution to the reverse engineering problem in a 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.

We have done an initial test to determine the acceleration gained by using FPGAs versus a high performance processor. A performance evaluation of FPGA and CPU implementation for the direct multiplier in $GF(2^{63})$ was made on the Cray XD1 platform; the field size was chosen to fit the 64-bit word-length of the target CPU architecture. A performance comparison between these multipliers and our approach is presented in Table 1. Times are measured for stand alone designs, in order to avoid the high overhead times arising from communication between the FPGA and the processor. Notice that the direct multiplier implementation on CPU is faster than the FPGA version of this method, a fact that is attributable to differences in clock speed: The clock rate for the Virtex-2P FPGA is about one-tenth that of on Opteron processor. However, when implemented on the same FPGA, our approach is about 65 % faster than the direct multiplier method.

Table 1. Multipliers comparison for the field $GF(2^{63})$ on Cray XD1: FPGA Virtex-2P xc2vp50-7, CPU 2.2 GHz AMD-Opteron

Multiplier	Time	Clock-period (Frequency)
Our approach on FPGA	0.62 μs	9.838 ns 101.6 MHz
Direct multiplier on FPGA	1.02 μs	16.168 ns 61.9 MHz
Direct multiplier on CPU	0.78 μs	

In Table 2 we compare our approach with other efficient multipliers reported in [3,4]. 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. Finite fields elements are represented as bit-arrays in our implementation. These arrays are a part of the entire logic design, which uses a small number of slices. For instance, the multiplier for $GF(2^{239})$ uses 1.53 % of slices in the Cray XD1 FPGA (see Table 3). Therefore there are many slices available for implementing larger finite field multipliers.

The times in Table 2 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 2. Multipliers comparison

Field	Target FPGA	Implementation	Time (μs)	Space (slices)
$GF(2^{210})$	Xilinx Virtex xcv-300-6T	Reference [7]	12.30	343
		This work	2.21	334
$GF(2^{233})$	Xilinx xc2v-6000-4	Reference [4]	2.58	not reported
		This work	2.42	415
$GF(2^{239})$	Xilinx Virtex xcv-300-6	Reference [3]	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 3). According to our results, there are significant opportunities for speeding up reverse engineering for large genetic networks on the Cray XD1 using reasonably the FPGA's physical space, however the communication time between CPU and

Table 3. Multipliers comparison 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%

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 comunication problems with the Cray XD1.

6 Conclusions and Future Work

Finite field multiplication is the dominant operation in the interpolation phase of reverse engineering genetic networks. Traditionally CPU-based implementations of large finite field multiplication have implied challenging efforts in order to deal with common limitations such as architecture word-size, and storage space. Our approach overcomes these traditional obstacles and at the same time contributes to improved performance, achieving better times than other efficient FPGAs finite field multipliers.

Although our approach has shown to be efficient for the finite fields reported in the previous section, it promises more efficient results for multipliers on larger fields. In order to efficiently solve the reverse engineering problem for large genetic networks, our FPGA implementation could be used as a co-processor for accelerating a CPU-based interpolation algorithm, provided that we can resolve the problem of high communication costs. An alternative would be to shift more of the computational burden to FPGAs by embedding our multiplier in an FPGA-based interpolation algorithm.

Future work includes extending our implementation of multiplication to finite fields generated by irreducible pentanomials. We also would like to extend the field size in order to deal with genetic networks with $m \geq 500$ genes. The multipoint interpolation for solving reverse engineering problem for very large genetic networks using high performance reconfigurable computing 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. A potential solution is to take advantage of FPGA Transfer Region of Memory using the I/O subsystem developed by OSC [2].

Finally, we could extend our ideas for doing multiplication over finite fields $GF(2^m)$ to doing the algebra of polynomials over $GF(2^m)$. This would enable

us to carry out the whole interpolation algorithm needed for the reverse engineering problem in FPGAs, thus allowing better optimization of the ration of computation time to communication time.

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. 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.
4. 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.
5. A. Halbutogullari, Ç. Koç, "Mastrovito Multiplier for General Irreducible Polynomials", IEEE Transactions on Computers, volume 49, number 5, pp. 503-518, 2000
6. T.E. Ideker, V. Thorsson, and R.M. Karp, "Discovery of regulatory interactions through perturbation: Inference and experimental design," Pacific Symposium on Biocomputing, No. 5 pp 302-313, 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. Laubenbacher and B. Stigler, "Dynamic networks," Adv. in Appl. Math Vol.26, pp. 237-251, 2001.
9. R. Laubenbacher, J. Shah, and B. Stigler, "A Computational Algebra Approach to the Identification of Gene Regulatory Networks", Proc. Third International Congress on Systems Biology, Stockholm, 2002.
10. Lee, T.I., Rinaldi, N.J., Robert, F., Odom, D.T., Bar-Joseph, Z., Gerber, G.K., Hannett, N.M., Harbison, C.R., Thompson, C.M., Simon I., Zeitlinger J., Jennings, E.G., Murray, H.L., Gordon, D.B., Ren, B., Wyrick, J.J., Tagne, J., Volkert T.L., Fraenkel, E., Gifford D.K., and Young, R.A. Transcriptional Regulatory Networks in Saccharomyces cerevisiae. Science 298: 799-804 (2002).
11. E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields", PhD thesis, Dept. of Electrical Eng., Linkvping Univ., Linkvping, Sweden, 1991.
12. O. Moreno, D. Bollman, and M. Aviñó,"Finite dynamical systems, linear automata, and finite fields", 2002 WSEAS Int. Conf. on System Science, Applied Mathematics & Computer Science and Power Engineering Systems, pp 1481-1483.
13. H. Ortiz, "Analysis of Gene Regulatory Networks Using Finite-Field models", PhD. Thesis Proposal, University of Puerto Rico, 2005.
14. B. Sunar and Ç. K. Koç "Mastrovito Multiplier for All Trinomials", IEEE Transactions on Computers", volume 48, number 5, pp. 522-527, May 1999.