ACCELERATION OF FINITE FIELD ARITHMETIC WITH AN APPLICATION TO REVERSE ENGINEERING GENETIC NETWORKS

by

Edgar Ferrer Moreno

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
in
COMPUTING AND INFORMATION SCIENCES AND ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2008

Approved by:

_____   _____
Omar Colón, PhD   Date
Member, Graduate Committee

_____   _____
Oscar Moreno, PhD   Date
Member, Graduate Committee

_____   _____
Nayda Santiago, PhD   Date
Member, Graduate Committee

_____   _____
Dorothy Bollman, PhD   Date
President, Graduate Committee

_____   _____
Edusmildo Orozco, PhD   Date
Representative of Graduate Studies

_____   _____
Néstor Rodríguez, PhD   Date
Chairperson of the Department

Abstract

Finite field arithmetic plays an important role in a wide range of applications. This research is originally motivated by an application of computational biology where genetic networks are modeled by means of finite fields. Nonetheless, this work has application in various research fields including digital signal processing, error correcting codes, Reed-Solomon encoders/decoders, elliptic curve cryptosystems, or computational and algorithmic aspects of commutative algebra. We present a set of efficient algorithms for finite field arithmetic over $GF(2^m)$, which are implemented on a High Performance Reconfigurable Computing platform. In this way, we deliver new and efficient designs on Field Programmable Gate Arrays (FPGA) for accelerating finite field arithmetic. Among the arithmetic operations, the most frequently used and time consuming operation is multiplication. We have designed a fast and space-saving multiplier, which has been used for creating other efficient architectures for inversion and exponentiation which have in turn been used for developing a new and efficient architecture for finite field interpolation. Here, the bit-level representation of the elements in $GF(2^m)$ and some special structures in the formulation of multiplication and inversion algorithms, have been exploited in order to use efficiently the FPGAs resources. Furthermore, we have also proposed a novel approach for multiplication over finite fields $GF(p^m)$, with $p \neq 2$, where the computational complexity is reduced from $O(n^2)$ to $O(n \log n)$.

Resumen

La aritmética sobre campos finitos desempeña un importante papel en diversas aplicaciones. Esta investigación ha estado motivada originalmente por una aplicación de Biología Computacional donde se modelan redes genéticas a través de campos finitos. Sin embargo, este trabajo tiene también aplicación en varios campos de investigación, tales como el procesamiento digital de señales, códigos de corrección de errores, codificación y decodificación de códigos de Reed-Solomon, criptografía de curva elíptica, o aspectos computacionales y algorítmicos de algebra conmutativa. Presentamos en este trabajo, una colección de algoritmos eficientes para hacer aritmética sobre cuerpos finitos del tipo $GF(2^m)$, los cuales son implantados sobre una plataforma de computación reconfigurable de alto rendimiento. De esta manera, proveemos diseños nuevos y eficientes para acelerar aritmética de campos finitos sobre FPGAs (del inglés Field Programmable Gate Arrays). Entre las operaciones aritméticas, la multiplicación es la operación que se usa con mayor frecuencia y es además la más lenta. Hemos diseñado por lo tanto, un multiplicador rápido y económico en espacio, el cual ha sido usado para crear otras arquitecturas eficientes para el cálculo de inversas y de exponenciación, las mismas han sido usadas para a su vez desarrollar una arquitectura eficiente para interpolación sobre cuerpos finitos. En este caso, la representación de los elementos del cuerpo $GF(2^m)$ a nivel de bits y algunas estructuras especiales en la formulación de los algoritmos de inversa y multiplicación ha sido aprovechada con el fin de usar eficientemente los recursos del FPGA. También hemos propuesto un método nuevo para multiplicación sobre cuerpos finitos del tipo $GF(p^m)$, con $p \neq 2$, donde la complejidad computacional es reducida de $O(n^2)$ a $O(n \log n)$.

To Merle and Hugo

Aknowledgments

This work has been accomplished thanks to the help of great leaders who contributed their time, ideas and resources to this project. First and foremost, I would like to express special gratitude to my advisor Dr. Dorothy Bollman, who helped me with her knowledge and great ideas. I thank as well the members of my graduate committee and the representative of Graduate Studies for their valuable suggestions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> Computer Science is a science of
> abstraction -creating the right
> model for a problem and
> devising the appropriate
> mechanizable techniques to solve
> it-.
>
> Alfred Aho & Jeffrey Ullman

The use of computational tools for accelerating engineering and scientific computing applications has been a fundamental theme in computer science research. High Performance Computing (HPC) has been used successfully for the acceleration of demanding computational applications, but the computational requirements are growing as researchers (from a broad range of science and engineering disciplines) are formulating more sophisticated problems. This pressure has influenced an interest in pushing the limits of current technologies, and in exploring new technologies for HPC. This dissertation is motivated by the potentiality of accelerating a biological application by means of a new way of high performance computing, namely High Performance Reconfigurable Computing (HPRC), where speedup is achieved by exploiting the synergism between hardware and software execution [32].

1.1 Motivation

Finite field arithmetic has a wide range of applications in various fields of science and engineering, including digital signal processing, cryptography, error-correcting codes and, more recently, in modeling genetic networks as finite dynamical systems.

The dynamical system concept is a mathematical formalization for any fixed "rule" which describes the time dependence of a point's position in its environment space [70]. Finite dynamical systems are dynamical systems on finite sets. The theory of finite dynamical systems has been used successfully in modeling gene regulatory networks [12, 80, 81, 98] by means of finite fields.

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

In this work we consider the reverse engineering problem in the context of univariate finite fields models [12, 14, 15]. In this framework, which is based on the theory of finite dynamical systems, solutions of the reverse engineering problem relies on intensive arithmetic computations over finite fields. Addition and multiplication are the two basic operations. But even though addition is easily realized at very low computational cost, multiplication is costly in terms of computation time and circuit complexity. Moreover, other arithmetic operations on finite fields used for reverse engineering such as inversion and exponentiation are performed by repeated multipli-

cations. The research elaborated in this dissertation provides a means for effectively accelerating the finite field arithmetic involved not only in reverse engineering large genetic networks, but in the whole range of applications of finite fields.

1.2 Problem Definition

The goal of this work is to provide a set of fast algorithms and then implementations for performing finite field arithmetic, including not only the usual operations of addition, subtraction, multiplication, and division, but also interpolation.

This work was motivated by the reverse engineering problem for genetic networks (a more detailed description of which is given in Section 3.2) which can be loosely stated in the context of the univariate finite field model as follows:

Given a time series of gene expression measurements that have been discretized to a prime number p of expression levels, $s_1, s_2, \ldots, s_n$ where each s_i represents the "state" of say m genes and a set of conditions χ, find a function f defined on the finite field $GF(p^m)$ such that $f(s_i) = s_{i+1}$, for all $i = 1, 2, \ldots, n-1$ and f satisfies the conditions in χ. The set of all functions f satisfying $f(s_i) = s_{i+1}$, $i = 1, 2, \ldots, n-1$, is given by

$$f(x) = P(x) + g(x)$$

where P is a polynomial determined by interpolating at the given points of the time series and $g(x)$ belongs to the ideal of polynomials that vanish on the s_i.

In order to perform interpolation for large genetic networks, it is essential to develop the capacity for performing very fast and efficient arithmetic over finite fields.

Researchers have employed various strategies to accelerate finite field calculations using both uniprocessor [55] and parallel computing based solutions [14]. However, there exist factors that limit the performance and scaling of finite field arithmetic algorithms such as limit in memory space, load balancing or simply Amdahl's law, which provides an upper bound on the speedup achievable by applying a certain number of processors to solve a problem in parallel. According to this argument [4], the speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program.[1] For example, if 95% of a program can be parallelized, the theoretical maximum speed-up using parallel computing would be twenty times, regardless the number of processors used.

Recently the potentialities of FPGAs have been taken into consideration for improving the performance of high-performance computing (HPC) applications as an alternative to massively parallel computing. High performance reconfigurable computers, based on the use of high-performance processors and FPGAs for accelerating HPC applications, are gaining interest in different research areas [48]. There has been significant research to support the potential performance gains available through the use of reconfigurable hardware for certain classes of computationally-intensive tasks. Nonetheless, despite well-known advantages of HPRC [16], using this technology could present significant challenges that need to be resolved [56]. Therefore, the suitability of a problem for HPRC based solution should be judiciously studied.

The problem of accelerating the interpolation phase of reverse engineering for large

---

[1]Amdahl's Law: If $1 - P$ is the fraction of a calculation that is serial and P the fraction that can be parallelized, then the greatest speedup that can be achieved using N processors is: $\frac{1}{(1-P)+\frac{P}{N}}$. In the limit, as N tends to infinity, the maximum speedup tends to $\frac{1}{(1-P)}$.

genetic networks could expend many computational resources and development effort. However, given that elements in binary extension finite fields can be represented as bit sequences on hardware platforms, a solution based on high performance reconfigurable computing seems to be a practical approach for effectively accelerating computations of finite field arithmetic involved in reverse engineering large genetic networks that are modeled by finite fields of characteristic 2.

Some important issues concerning an efficient HPRC-based solution of our intended application have to be overcome. Problems such as CPU-FPGA interfaces, selecting optimal algorithms suitable for FPGAs, using appropriate structures for the designs, limited resources, administrating wisely the time/space tradeoff, must be addressed in order to develop an application delivering substantial performance improvement with a reasonable use of computational resources.

1.3 Research Objectives

The main goal of this research is to provide computational means to achieve efficient designs for fast finite field arithmetic that are needed in applications such as digital signal processing, coding theory, cryptography, and especially reverse engineering genetic networks.

Optimal and appropriate algorithms must be selected in order to solve problems that include mainly finite field multiplication, inversion, and interpolation.

We deal mainly with arithmetic in fields $GF(2^m)$. Such a field models a genetic network in which each of the m genes has just two states, either *on* or *off*. Motivated by this application, we wish to develop algorithms that can be readily implemented

for a large number of genes m.

It is necessary to study the state of the art of methods employed for solving the aforementioned problems, and optimize procedures for implementing efficient solutions according to the requirements of the intended application.

In order to optimally use the available resources and simultaneously achieve significant speedups, it is crucial to evaluate some drawbacks that need to be solved. In this sense, the following must be considered: the impact of data flow and data representation on the architectures performance, the overhead associated with communications between CPU and FPGA, the area constraints, and the problem size.

1.4 Contribution

A novel approach for finite field multiplication over odd-prime extension fields has been introduced. A fast and space-saving design for a finite field multiplication over $GF(2^m)$ was also introduced. This multiplier became essential for the design of an efficient architecture for finite field inversion toward the ultimate and more challenging problem of developing a new and efficient architecture for finite field interpolation.

This research provides novel and efficient computational methods for accelerating finite field based algorithms employed for the interpolation phase of the solution of the reverse engineering problem for genetic networks. This computational biology application could be practical for biologists who need to have a better understanding on complex biological phenomena, such as, prediction of effects of new drugs, or disease mechanisms.

Although the ideas developed are intended for the aforementioned computational

biology application, these notions could be employed as well to other applications where models based on finite fields are involved. In this sense, this work provides a contribution into the computational and algorithmic aspects of commutative algebra. Furthermore, high-performance finite field arithmetic is useful for solving problems in digital signal processing, error correcting codes, Reed-Solomon encoders/decoders, elliptic curve cryptosystems, and learning algorithms.

We have developed a novel high-speed and space-saving design for finite field multiplication in $GF(2^m)$. This simple and fast architecture is useful for implementing an efficient FPGA-based approach for inversion which is used for developing finite field interpolation. The former became the principal achievement in this work, given that the major research effort was oriented to overcome some implied issues concerning finite field interpolation. As a result, a new efficient architecture for univariate polynomial interpolation over binary finite fields has been obtained. The proposed interpolator reaches substantial acceleration factors. Thus, it promises to be useful for an efficient solution of the reverse engineering problem for Boolean genetic networks, in the same way as it can contribute to solve other problems requiring efficient interpolation over large finite fields $GF(2^m)$. To the best of our knowledge, this is the first work concerning an entire design of finite field polynomial interpolation for FPGAs.

We have also developed a new multiplication for certain fields of characteristic $p \neq 2$. This algorithm, based on convolution, reduces the complexity of multiplication in $GF(p^m)$ from $O(m^2)$ to $O(m \log m)$.

The algorithms and architectures proposed have proven to perform efficiently in

an HPRC environment. We hope that this will contribute to the development of the incipient research area that is HPRC. But this contribution is not only limited to the achieved results, but to how the results were achieved. Moreover, the methods and techniques employed in this work could be used with future reconfigurable computing technologies.

1.5 Dissertation Outline

The remainder of this dissertation is organized as follows: Chapter 2 summarizes the fundamental theory relevant to the materials presented in this dissertation. The first section of the chapter introduces some basic definitions concerning high performance reconfigurable computing followed by a description of the hardware and software tools employed in the development of this research. The chapter closes with an overview of finite field theory, and finite field arithmetic techniques.

Chapter 3 describes reverse engineering in the context of univariate finite field model. Some theory about reverse engineering and a description of the model used are presented.

The design of a new finite field multiplier over $GF(2^m)$ is presented in Chapter 4. Experimental results are shown. In addition, the use of finite field multiplication for computing finite field inversion is considered.

Finite field polynomial interpolation on FPGAs is studied in Chapter 5, the complete design of the proposed architecture is described, and some numerical experiments are presented.

Finite field multiplication over $GF(p^m)$, with $p \neq 2$ via *number theoretic transform*

is addressed in Chapter 6.

Some ethical issues concerning the present research are mentioned in Chapter 7. Finally, Chapter 8 gives some concluding remarks, and the chapter closes with recommendation for future work.

Chapter 2

Background material

This Chapter summarizes the fundamental mathematical and computational theory which is relevant to the materials presented in this dissertation.

2.1   High Performance Reconfigurable Computing (HPRC)

In this work we consider the opportunities of adapting a current high-performance computing application to efficiently operate on a reconfigurable platform using a High Performance Reconfigurable Computing (HPRC) paradigm. Some fundamental terms and a brief description of the reconfigurable platform used in the present research are described in this section.

2.1.1 HPRC Terminology

This section introduces some HPRC terms which are referred to in subsequent sections. These and other terms commonly used in HPRC can be found in [11, 48, 101].

Reconfigurable Computing is a computing paradigm employing FPGAs or reconfigurable devices for processing data. A different bitstream can be loaded during the execution of a program or to run a different program on the fly.

A **Field Programmable Gate Array (FPGA)** is a regularly tiled two-dimensional array of logic blocks. The logic blocks communicate through a programmable interconnection network that includes both nearest neighbor as well as hierarchical and long path wires. An algorithm design produces a bit pattern that connects the logic blocks in an FPGA in order to implement that algorithm in hardware.

A **Reconfigurable Device** may be an FPGA, or any other device whose functionality can be changed during execution. If in a hardware architecture both functionalities of processing elements and interconnections between them can be modified after manufacture time then it is a reconfigurable device or architecture.

High Performance Reconfigurable Computing is defined as the study of computation using reconfigurable devices and high-performance computers.

Speedup is a measure of how much faster a given program runs when executed onto a reconfigurable device as compared to serial execution on a single

processor. The speedup ratio is determined as by $S = \frac{\text{runtime on CPU}}{\text{runtime on FPGA}}$.

Bitstream is the file that configures the FPGA (often has a .bit or .bin extension). The Bitstream gets loaded into an FPGA when ready for execution. It is obtained after synthesis, mapping, place and route phases of the implementation process.

Configuration should refer to the bitstream currently loaded on an FPGA.

Reconfiguration also named programming, or re-programming, is the action of loading a circuit design onto an FPGA.

Synthesis is the process of creating a netlist from a circuit description, usually described by an HDL (Hardware Description Language).

Place and Route is the process of converting a netlist into physically mapped and placed components on the FPGA, ending in the creation of a bitstream. Essentially tries to fit the circuit design onto the FPGA surface as well as possible.

Local Memory is a memory directly connected to an FPGA but that is not inside the FPGA chip itself. Often named as DRAM, SRAM, QDR, DDR SRAMs, or ZBT RAM.

Host Memory is a memory accessible by the whole computer. It should refer to memory on the microprocessor motherboard and it is not necessarily directly accessible through the FPGA.

Hardware Emulation/Simulation is the process of mimicking the behavior of a circuit or FPGA configuration on a CPU based system.

2.1.2   Hardware and Software Tools

The implementations and results presented in this work have been developed on the Cray XD1 system [25]. This is a modular high performance computing system with the base unit consisting of a chassis with up to six nodes which are technically known as compute blades. In our Cray XD1, each compute blade contains two AMD Opteron 275 2.2 GHz dual core processors with 8 GBs of memory per processor. Each compute blade includes also a RapidArray processor which provides two 2 GB/s RapidArray links to the switch fabric. An application acceleration system is also included in a compute blade.
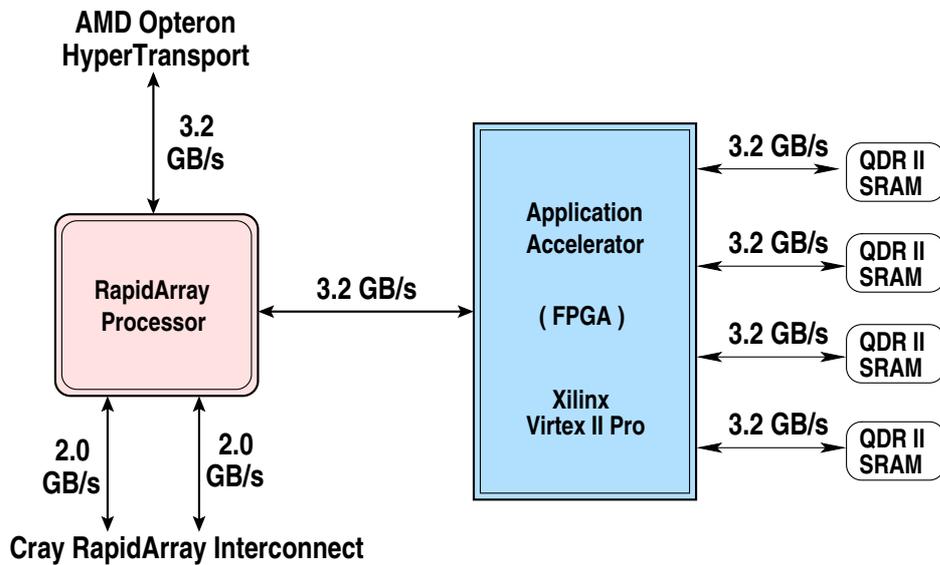
Figure 2.1: The Cray XD1 processor module.

The application acceleration system is an FPGA-based reconfigurable computing module that provides an FPGA complemented with a RapidArray Transport core providing a programmable clock source, and a 3.2 GB/s link to the AMD Opteron processor (see Figure 2.1). Four banks of 8 MB Quad Data Rate II (QDR) SRAM local memories are included as well in the application acceleration module. The FPGAs units are Xilinx Virtex II Pro 50.

Each node runs a Cray modified version of SuSE Linux (kernel 2.6.5). The Cray XD1 is supplied with standard primitives [26] (RapidArray Communications Libraries) for FPGA setup and CPU-FPGA interactions. The FPGA developments presented in this work were done by using the tools included in the Xilinx ISE Foundation 9.1i development toolset [140]. Simulations have been done using the ModelSim simulator of Mentor Graphics [95]. In this work all codes are synthesized from VHDL language.

2.2   Finite Fields

A brief overview about fundamentals of finite fields is given in this section. A comprehensive review of finite fields with important definitions and properties with proofs can be found in [85].

Informally, a field is a set of elements in which it is possible to add, subtract, multiply and divide, such that the commutative, associative and distributive properties are satisfied. The fields with a finite number of elements are called finite fields. This is stated more formally in the following definition.

Definition 1 *A finite field $\{F, +, \cdot\}$ consists of a finite set F, and two operations $+$ and $\cdot$ that satisfy the following properties:*

1. *$\forall a, b \in F, a + b \in F, a \cdot b \in F$*

2. *$\forall a, b \in F, a + b = b + a, a \cdot b = b \cdot a$*

3. *$\forall a, b, c \in F, a + (b + c) = (a + b) + c, (a \cdot b) \cdot c = a \cdot (b \cdot c)$*

4. *$\forall a, b, c \in F, a \cdot (b + c) = (a \cdot b) + (a \cdot c)$*

5. *$\exists 0, 1 \in F, a + 0 = 0 + a = a, a \cdot 1 = 1 \cdot a = a$*

6. *$\forall a \in F, \exists (-a) \in F$ such that $a + (-a) = (-a) + a = 0$*

   $\forall a \neq 0 \in F, \exists a^{-1} \in F$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$

Finite fields are also referred to as *Galois fields*. A finite field with q elements is denoted by $GF(q)$. The number of elements in a field can be either prime or a power of prime. From now p will denote a prime. $GF(p^m)$ is the field of p^m elements, it is also called an extension field of $GF(p)$ and p is called the characteristic. It can be shown that for any element α in a field of characteristic p, $p\alpha = \alpha + \alpha + \cdots + \alpha$ (p times) is equal to zero.

Some additional definitions and properties of finite fields needed for understanding the material presented in this dissertation are introduced below.

Definition 2 *The order of a finite field is the number of elements in the field.*

Definition 3 *Let α be a nonzero element of $GF(p^m)$, the order of α is the smallest positive integer, $ord(\alpha)$, such that $\alpha^{ord(\alpha)}$ is the identity element of $GF(p^m)$.*

For $\alpha \neq 0$ in $GF(p^m)$, $ord(\alpha)$ always divides $p^m - 1$. Hence $\alpha^{p^{m-1}}$ is always the identity element of $GF(p^m)$.

Definition 4 *When $ord(\alpha) = p^m - 1$, α is called a primitive element of $GF(p^m)$.*

Definition 5 *A polynomial, whose coefficients are elements of $GF(p^m)$, is said to be a polynomial over $GF(p^m)$.*

Definition 6 *A polynomial over $GF(p^m)$ is irreducible if it cannot be factored into non-trivial polynomials over the same field.*

Every irreducible polynomial of degree m over $GF(p)$ defines an unique extension field $GF(p^m)$, and for every power of a prime p^m, there is exactly one (up to isomorphism) field $GF(p^m)$.

Definition 7 *A primitive polynomial is a polynomial $F(X)$ with coefficients in $GF(p)$ which has a root α in $GF(p^m)$ such that $\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{p^m-2}\}$ is the entire extension field $GF(p^m)$, and moreover, $F(X)$ is the smallest degree polynomial having α as root.*

2.3 Finite Fields Representation

The representation of the field elements distinguishes the particular features in the finite field arithmetic. The most common representations are the powers representation, dual basis, normal basis, and standard basis [59].

Let α be a primitive element of $GF(p^m)$. In the **powers representation**, the set of elements of $GF(p^m)$ can then be represented as:

$$\{0, 1, \alpha, \alpha^2 \ldots, \alpha^{p^{m-2}}\}$$

In a **normal basis** representation, each of the basis elements is related to any one of them by applying the p-th power mapping repeatedly, where p is the characteristic of the field, that is to say:

Let $GF(p^m)$ be a field with p^m elements, and β an element of it such that the m elements

$$\{\beta, \beta^p, \beta^{p^2}, \ldots, \beta^{p^{m-1}}\}$$

are linearly independent.

The first normal basis multiplication algorithm was reported by Massey and Omura [90] and its first implementation was reported by Wang *et al* [138]. To date, numerous implementations based on the Massey-Omura multiplier have been reported [54, 111, 112].

The **dual basis** is not a concrete basis like the polynomial basis or the normal basis; it rather provides a way of using a second basis for computations. Using a dual basis can provide a way to easily communicate between devices that use different bases, rather than having to explicitly convert between bases using the change of bases formulas. The original dual basis representation for finite field multiplication is due to Berlekamp [10]. Later on, this algorithm was modified, generalized, and implemented in hardware by Hsu *et al* [58]. Other dual basis implementations based on Berlekamp's algorithm have been reported [47, 114].

The **standard basis** is a natural representation of finite field elements as polynomials over a ground field, which is also known as polynomial representation. It is defined as follows:

Let $\alpha \in GF(p^m)$ be the root of an irreducible polynomial of degree m over $GF(p)$. The standard or polynomial basis of $GF(p^m)$ is then

$$\{0, 1, \alpha, \ldots, \alpha^{m-1}\}$$

Thus, in this representation each element of $GF(p^m)$ is expressed as a polynomial $c_0 + c_1\alpha + c_2\alpha^2 + c_{m-1}\alpha^{m-1}$ over $GF(p)$.

Because of its simplicity, the standard basis representation has been widely used. The earliest standard basis multiplier was proposed by Bartee *et al.* [9]. A first high performance standard basis multiplier for VLSI was reported by Scott *et al.* [121]. Some recent implementations are reported in [113]. Since the present research has been developed concerning standard basis, in the remainder of this Chapter we will consider uniquely the previous work related with finite fields represented in standard basis.

In this research, the finite fields of fundamental interest are the *extension fields* of $GF(2)$, denoted by $GF(2^m)$. The simplest example of a finite field is the binary field $GF(2) = \{0, 1\}$, the operations in this field are addition and multiplication modulo 2.

From Table 2.1 it is easily verified that the 6 properties of Definition 1 hold, and therefore $GF(2)$ is a finite field.

Table 2.1: Operations for the finite field $GF(2)$

a	b	$a+b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

We can create larger fields by extending $GF(2)$ to an m-dimensional vector space leading to finite fields of size 2^m. Then we have the field $GF(2^m)$ where each element could be seen as a binary m-tuple.

As an example of an extension finite field, consider the field GF(8). We can use three alternate and equivalent representations to represent each element in the field:

1. In the *powers representation* all non-zero elements in GF(8) may be represented as powers of a primitive field element α (see details in [85]), then each non-zero element is of the form α^n for $n = 0, 1, \ldots, 6$

2. In the *polynomial representation* each element in the field $GF(8) = GF(2^3)$ is represented as polynomials with degree less than 3 whose coefficients belong to $GF(2)$. The polynomials are defined according to the irreducible polynomial that generates the field.

3. In the *m-tuple representation* each element in the field $GF(8) = GF(2^3)$ can be represented as an 3-dimensional binary vector, i.e, a binary 3-tuple. Each vector is determined by the coefficients of the respective polynomial representation.

We can take advantage of the *powers* representation in a mathematical framework while the *m-tuple* representation is convenient to deal with digital hardware. The

Table 2.2: Alternative representations in the field $GF(2^3)$

Powers Representation	Polynomial Representation	3-tuple Representation
0	0	000
α^0	1	001
α^1	α	010
α^2	α^2	100
α^3	$\alpha + 1$	011
α^4	$\alpha^2 + \alpha$	110
α^5	$\alpha^2 + \alpha + 1$	111
α^6	$\alpha^2 + 1$	101

use of both representations for fast finite field arithmetic is addressed in the next subsections.

2.4 Related Work on Finite Field Arithmetic

In this section we review some recent works concerning arithmetic in binary extension fields $GF(2^m)$ with standard basis representation.

2.4.1 Multiplication over Finite Fields

The finite field multiplication plays a predominant role in accelerating reverse engineering of genetic networks and other known finite field applications. In consequence, it has been necessary to expend important efforts in designing efficient multipliers. It is well known that arithmetic in $GF(2^m)$ has been attractive for implementing in hardware, hence the binary finite field arithmetic using FPGAs has gained significant attention in recent years. In this manner, different FPGA based approaches have been proposed in recent years. An early survey of finite field multiplier designs and

their performance characterization on FPGAs is presented in [1].

A much studied method for finite field multiplication is the Massey-Omura Multiplier. This approach has been improved from its original [89], by removing redundancies [111]. This method is essentially effective for normal bases, however, it has been used on applications intended for standard basis. Savas *et al*, used this multiplier combined with a process for normal/standard basis conversion [119]. Other improvements of this method for FPGA platform have been reported [1].

A multiplication method generally used in cryptosystems is the so-called Montgomery multiplication. Montgomery multiplication was first proposed for efficient integer modular multiplication [97]. Later on, it was extended to finite field multiplication in $GF(2^m)$ by Koç *et al.* [76]. They describe this multiplication method as follows:

Let $f(x)$ be an irreducible polynomial that defines the field $GF(2^m)$ and $r(x)$ be a fixed element in $GF(2^m)$ such that $\gcd(f(x), r(x)) = 1$. Then, the extended Euclidean algorithm can be used to determine $\tilde{f}(x)$ and $\tilde{r}(x)$ that satisfy

$$r(x)\tilde{r}(x) + f(x)\tilde{f}(x) = 1 \tag{2.1}$$

clearly $\tilde{r}(x) = r^{-1}(x)$ is the inverse of $r(x)$. Given two fields elements $a(x), b(x) \in GF(2^m)$, the Montgomery multiplication is given by

$$c(x) = a(x)b(x)r^{-1}(x) \bmod f(x) \tag{2.2}$$

The efficiency of this multiplier is dependent on the chosen fixed field element

$r(x)$. Efficient architectures for certain class of fields $GF(2^m)$ have been implemented on FPGAs [93].

Essentially, a finite field multiplication in standard basis consists of a polynomial multiplication followed by a modular reduction. Some authors have realized that the performance of a multiplier can be improved by reducing computation in these two steps. Some proposed solutions include combining all the computations inito one step, computing both steps at the same time, or precomputing the first step. Next we will refer to some approaches concerning these issues.

Systolic array architectures have been considered in the design of multipliers over $GF(2^m)$, this paradigm has been useful for speeding up computations by exploiting bit-level parallelism and pipelining. Some systolic architectures for fast finite field multiplication have been presented [29, 82]. A high-throughput hardware-efficient semi-systolic linear array for a serial-parallel implementation of finite field multiplier over $GF(2^m)$ is presented in [45], where the polynomial multiplication step is computed into a serial design while the reduction step of multiplication is performed by a bidirectional modulo reduction technique.

An efficient multiplication scheme for a standard basis multiplier has been developed by Mastrovito in [91]. In this approach the multiplication $C = A \cdot B$ is performed by means of a matrix-vector product $\vec{c} = Z\vec{b}$, where $\vec{c}$ and $\vec{b}$ are the components vectors of C and B and Z is the Mastrovito matrix whose elements are obtained by XOR operations over some of the components of A. With the construction of Z the reduction step is precomputed and polynomial multiplication step is performed by the Matrix-vector product. The Mastrovito based multiplier has been broadly used

due to its capabilities for reducing the time and space complexity when finite fields generated by some classes of irreducible polynomials are used. Given that the amount of operations in the multiplier is determined by the irreducible polynomial which defines the field, some authors have proposed architectures based on certain irreducible polynomials [6, 132]. Other variant of multipliers based on Mastrovito matrix have been reported in [52, 108, 125].

In Chapter 4 we will present a novel design based on the Mastrovito matrix [39], this multiplier has been compared with other standard basis multiplier over $GF(2^m)$, some of which are mentioned below.

In [44], an efficient multiplier architecture of the type serial/parallel is presented where the modular reduction is carried out 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 [75], 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 [49], 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.

2.4.2 Division over Finite Fields

Finite field division, which implies computation of inversion, is the most complex finite field arithmetic operation and various algorithms and architectures have been proposed based on different approaches [31], such as the extended Euclidean algorithm or one of its derivatives, the extended binary gcd [100] (also known as extended Stein's algorithm), or Fermat's little theorem.

New formulations of the extended Euclidean algorithm have led to design architectures for finite field inversion. For instance, in [141] Yan *et al.* propose a version of the extended Euclidean algorithm to deal with a new two-dimensional systolic architectures for inversion in $GF(2^m)$. Another new architecture based on the extended Euclidean algorithm uses a distributed control mechanism which results in the architecture having the same circuitry regardless of the value of m, this architecture provides good scalability properties.

Stein's algorithm has been used in an application to cryptosystems in [73]. A variation of this architecture is presented in [74] for $GF(2^{163})$ and $GF(2^{239})$. These kinds of algorithms are usually considered to be slow [41], because a great number of degree comparisons is required at each step, increasing in this way the area-time complexity. However, in [96] the authors claim to overcome the traditional obstacles by replacing the comparisons by a much more simple counter and taking advantage of binary representations on FPGAs. This idea was exploited also by Wu *et al.* in [139], where two very similar serial binary shift-right algorithms are presented. The authors show that these modifications lead to an even better area-time complexity.

Another well studied finite field method for inversion is due to Itoh and Tsujii.

This algorithm is based on Fermat's Little Theorem and uses a clever re-arrangement of the finite field operations to compute binary exponentiations through addition chains. The algorithm was originally conceived for inversion over normal basis representation [65] in $GF(2^m)$. However, since the first publication some generalizations have been reported. In [50], Guajardo *et al.* have formulated a design generalizing the algorithm to any field of the type $GF(p^m)$, showing that the method can be used in standard basis too. Other improvements of Ito-Tsujii algorithm are reported in [117, 134, 142]. Recently Rodriguez *et al.* [115, 116] have proposed parallel architectures of the standard Itoh-Tsujii algorithm, which deliver good performance on FPGAs.

We have developed an FPGA based implementation of the standard Itoh-Tsujii algorithm [50, 117]. In Chapter 5, we will provide more details concerning this architecture for finite field inversion as a component of Newton's algorithm for interpolation over finite fields.

2.4.3 Interpolation over Finite Fields

The interpolation process always implies intensive arithmetic. It is a given that as the finite field arithmetic involved in interpolation develops into abundant and complex operations, interpolation over finite fields becomes a challenging process. In recent years, some researchers have considered the suitability of interpolation over finite fields for certain applications such as decoding error correcting codes [105], testing and fault detection [28], and in learning algorithms [120].

More recent developments include the work of Zilic and Vranesic [145]. The afore-

mentioned research, presents a multivariate interpolation algorithm over arbitrary fields which is suited for small finite fields. This algorithm uses tools of linear algebra, including the new properties of the generalized multivariate Vandermonde matrix.

The use of finite field interpolation in a public key cryptography application was evaluated in [79] for tackling the problem of the discrete logarithm. The authors studied the Aitken and Neville interpolation methods on discrete exponential functions over finite fields; they concluded that the computational cost of finding a polynomial that interpolates the discrete logarithm by either method is high. However, the approach could be applied to low degree polynomials.

A parallel approach for univariate polynomial interpolation over finite field is proposed by Bollman *et al.* in [14]. They obtain the interpolation polynomial through Lipson's algorithm which is based on the Chinese remainder theorem. Using the divide-and-conquer idea, Lipson's algorithm builds a solution in a tree-like fashion. This feature is exploited for the parallelization of the algorithm.

The methods and techniques presented in this review have been conceived for software based solutions. As far as we know, up until now no other finite field interpolation method has been entirely developed for hardware devices or FPGAs.

We have developed an FPGA based implementation for univariate polynomial interpolation over $GF(2^m)$ [40]. In Chapter 5, we will provide more details concerning this novel architecture.

Chapter 3

Reverse Engineering Genetic Networks

> The machine does not isolate
> man from the great problems of
> nature but plunges him more
> deeply into them.
>
> Antoine de Saint-Exupéry

The results of our research on fast finite field arithmetic, given in the succeeding chapters, have a wide range of applications. However, as mentioned previously, our work has been motivated by the reverse engineering problem for genetic networks. Thus, before discussing our specific results in fast finite field arithmetic, in this chapter we give an overview of the reverse engineering problem and our approach to the problem through the use of univariate finite field model.

3.1   On the Use of Mathematical Models

For decades biologists have claimed the need to formalize the process of modeling and analyzing biological systems. Various mathematical models have been proposed, from those described by systems of differential equations [143] to those descriptive models based on a formal language [46]. However various of these models have been

debated in the biologist community because a rigorous mathematical knowledge is required or on the contrary, those models turn out to be oversimplified [92]. Many other models have been developed in the past decades, still in recent years researchers have employed significant effort for the development of new models adapted to the need of new technologies.

Recent technological advances in the life sciences have contributed to the increase of the amount of experimental data, such as whole genome sequences or structures of proteins in living organisms. With this abundance of information has come the ability to gain knowledge about the underlying system. In response to these modern exigencies, various methods for discovering interactions in biological systems have been proposed. In what follows, we describe a number of different reverse engineering approaches for modeling genetic networks, comprising continuous and discrete methods.

3.1.1 Machine Learning Methods

Machine learning techniques such as genetic algorithms, neural networks, and fuzzy logic have been broadly applied to reverse engineering genetic networks. **Genetic algorithms** have been employed for parameter estimation in genetic networks models from both artificial and experimental microarray data [110, 137], genetic algorithms were also used in [62] to construct genetic networks from time-series gene expression data. Other methods that use genetic algorithms have been developed with different modeling frameworks of genetic regulatory networks, see for example [5, 133].

Neural networks have been employed for clustering of gene expression data [57], while in [60] the relationship between clusters is determined by using artificial neural networks. Kasabov in [69] employed neuro-fuzzy style neural networks, knowledge-based neural networks, for the classification of clusters and reverse engineering of Genetic networks. Sokhansanj *et al.* [124] have introduced a linear fuzzy gene network model that represents a set of fuzzy 'if-then' rules for genetic regulatory networks. In cite [20], other applications of machine learning techniques for reverse engineering regulatory networks are reviewed.

3.1.2 Bayesian Networks

Bayesian methods make use of the Bayes' rule to reverse engineer genetic networks by inferring the causal relationship between two network nodes based on conditional probability distributions. Friedman *et al.* in [42] proposed Bayesian networks to infer causal dependencies between genes in gene regulatory networks. An extension of this work was proposed in [107] by Pe'er *et al.* to reverse-engineer significant subnetworks of interacting genes such that detailed regulation types (activation or inhibition) can be inferred from the input data of perturbation experiments such as gene deletion or over-expression.

The concept of a dynamic Bayesian network was introduced by Hartemink *et al.* in [53] to deal with time-dependent data. Basically, these are simple extensions of the static Bayesian methods using time-series input data. Dynamic Bayesian methods also focus on the probabilistic causal relationship between two network nodes and assume that these relationships do not change over time like the static Bayesian

methods. Recently, Zou and Conzen [146] have investigated a new dynamic Bayesian algorithm for predicting the gene regulatory networks from time course expression data, identifying events that take place over a given period of time, and estimating the so-called transcriptional time-lag between genes. The authors claim their approach significantly improves accuracy and reduces computational time compared with existing dynamic Bayesian networks approaches. More about Bayesian networks models for reverse engineering genetic networks can be studied in [20, 66].

3.1.3 Ordinary Differential Equations

Systems of ordinary differential equations (ODE) have been successfully applied for modeling biological systems. In general, an n-nodes gene regulatory network can be represented by a system of ordinary differential equations

$$\frac{dx_1(t)}{dt} = f_1(x_1(t), \ldots, x_n(t))$$
$$\vdots$$
$$\frac{dx_n(t)}{dt} = f_n(x_1(t), \ldots, x_n(t))$$

where $x(t) = (x_1(t), \ldots, x_n(t))$ is a vector of nodes x_i ($1 \leq i \leq n$) representing gene expression levels at time t and $f = (f_1, \ldots, f_n)$ is a vector valued function from the real n-dimensional space $I\!\!R^n$ into $I\!\!R^n$.

Various approaches of reverse engineering based on ODE models have been reported in the literature. Yeung *et al.* described in [143] a method to reverse-engineer genetic networks with linear ODEs where a set of solution is determined by singular value decomposition (SVD). In [135], it is presented an alternate algorithm, where

new linear algebra techniques are used to choose one solution from the set of solutions obtained by SVD. In [17] Chen *et al.* used linear ODEs in order to formulate four models for gene and protein expression data. They describe two algorithms for constructing such models from data. Other ODE-based approaches for reverse engineering genetic networks can be studied in [20, 66, 126].

3.1.4 Boolean Networks

A Boolean network is defined by $G(V, F)$, where $V = \{v_1, \ldots v_n\}$ represents a set of nodes corresponding to genes, and $F = \{f_1, \ldots f_n\}$ is a set of Boolean functions assigned to each node. The state of a node is completely determined by the values of other nodes at a determined time, the transitions between states are determined by Boolean functions.

A Boolean function is a function involving Boolean variables and the operations $\wedge$, $\vee$, $\neg$, which are defined in Table 3.1.

Table 3.1: Boolean Operations.

x	y	$x \wedge y$	$x \vee y$	$\neg x$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Boolean methods for reverse engineering are used to infer gene regulatory networks by applying Boolean logic to the discretized gene states which indicate gene expression levels. The states values are 0 and 1, where 0 mean an *off* (unexpressed or inactive) state, while 1 mean an *on* (expressed or active) state of genes.

Stuart Kauffman was amongst the first biologists to use the idea of Boolean networks to model gene regulatory networks as logical switch networks [71, 72]. In the last decades this approach has been well studied. Recent studies in this field include the contribution of Liang *et al.* [84] who proposed an information-theoretic algorithm for constructing Boolean networks from Boolean time series data. The algorithm constructs both the global function as well as the graph describing the system, such that the system can take the maximal amount of information, as defined by the Shannon's entropy [1]. Although this study is limited to synchronous Boolean networks, the algorithm is generalized to include multi-state models.

In [3], Akutsu *et al.* present an reverse engineering approach based on a Boolean network model without time delay (asynchronous), for identifying a genetic network by multiple gene disruptions and overexpressions. They calculated upper and lower bounds on the number of experiments that would be required if the network were Boolean.

Ideker *et al.* formalized a model for reverse engineering through an inference method called predictor. The predictor method is used to provide candidate networks as a Boolean network model that are consistent with expression data by employing combinatorial optimization techniques [63, 64].

In this review, it is worth mentioning a Boolean network approach that incorporates stochastic features of gene regulation. Probabilistic Boolean networks have been introduced in [122]. They are probabilistic extensions of Boolean methods, these net-

---

[1] In information theory, the Shannon entropy or information entropy is a measure that quantifies the information contained in a message, usually in bits or bits/symbol. It is the minimum number of bits (message length) needed to encode a string of symbols, based on the frequency of the symbols.

works consider many Boolean functions $f_{i_1}, f_{i_2}, \ldots f_{i_k}$, of each node x_i and the probabilities with which each Boolean function f_{i_j} is chosen to predict the state of x_i. Some recent studies concerning probabilistic Boolean networks include [30], [87], [144]. A comparison of probabilistic Boolean network versus dynamic Bayesian network approaches for reverse engineering is presented in [83].

There exist many more reverse-engineering methods based on Boolean models. This review does not claim to be comprehensive, but it provides a context for new methods inspired by Boolean models. For more thorough reviews the reader can consult [66].

3.1.5 Finite Field Models

One of the disadvantages of the Boolean network modeling framework is the limited range of gene expression levels, given that Boolean variables can only represent all or no effects [103]. The need to discretize gene expression data into an *on/off* scheme causes loss of information. In response to this deficiency, researchers have proposed to generalize the Boolean genetic networks to finite field genetic networks.

Laubenbacher *et al.* [81] have proposed a multivariate model in which each of m genes is described by a function $f_i : GF(p)^m \to GF(p)$. Moreno *et al.* [98] have proposed a univariate model in which the dynamics of the complete network of m genes is described by a single function $f : GF(p^m) \to GF(p^m)$. Now, each of the Boolean operations, $\wedge$, $\vee$, and $\neg$ can be expressed in terms of mod 2 operations, i.e.,

$$x \wedge y \;=\; xy$$

$$x \vee y \;=\; x + y + xy$$

$$\neg x \;=\; 1 + x$$

and so each of the above Boolean models can be considered as finite field model where

$p = 2$.

Bollman *et al.* show that the univariate and multivariate models are equivalent

and that one can be converted to the other by means of a discrete Fourier transform.

Laubenbacher *et al.* [81] provide a computer algebra solution to the reverse engi-

neering problem for the multivariate model which can be described as follows:

Given a sequence of n "states" $s_1, s_2, \ldots, s_n \in GF(p)^m$, find all functions

$f_i : GF(p)^m \to GF(p)$ such that f_i maps each s_j to the i-th coordinate of s_{j+1}, and

from each such set choose a function that is not identically equal to zero at all s_j.

Alternative models of gene expression in genetic networks based on finite fields

are addressed by Ortiz-Zuazaga *et al.* in [104]. They have developed heuristic proce-

dures that select genes based on coarse-grained reproducible changes. The selection

procedure clusters genes into discrete groups suitable for reverse engineering. Ortiz-

Zuazaga also proposes in [102] a probabilistic finite field genetic networks model which

is an extension of the probabilistic Boolean networks. This probabilistic model com-

bines the benefit of probabilistic Boolean networks with finite field genetic networks.

In a sense, this approach is useful for overcoming limited ranges of gene expression

while deals with the uncertainty in expression measurements. In this context, a par-

ticular model of interest is the so-called ternary model where the values are expressed

in term of elements in the field $GF(3)$, so it is possible to capture the biological intuition of genes being expressed, repressed or unchanged.

Continuous methods have been studied because biological systems have been understood in terms of continuous events, where the system moves continuously from one state to another. However, discrete methods are usually involved in discovering regulatory interactions in biological systems as well, even when raw continuous data is analyzed [128]. For instance, dynamic models constructed from reverse engineering methods must fit discrete instances of a continuous process [129]. Usually discrete methods include a discretization step.

A Finite Dynamical System (FDS) constitutes a very natural discrete model for regulatory process, such as genetic networks [12]. In the present research we focus on a discrete method which represents gene interaction in a biological system through graphs associated with functions. By using this FDS-based model, it is feasible to express the considerable quantity of data in a computationally tractable environment.

The model mentioned above is characterized by systems of discrete-value functions. Vertices on the graph correspond to states in a biological system, which take on discrete quantities or levels, and the edges depict interactions between biochemical states affecting their levels. The number of discrete quantities or levels to be considered is two, to signify presence/absence or activity/inactivity of the genes in the system. The aforementioned is equivalent to the Boolean network model (reviewed in Section 3.1.4). This approach results into two related models. Namely, the univariate model which was developed by Moreno and colleagues [98, 99], and the Multivariate Model developed by Laubenbacher *et al.* [81]. The multivariate model gives local in-

formation at each gene, whereas the univariate model gives global information about the network. However, one model can be converted to the other by means of discrete Fourier transform (see [12])

In [102] Ortiz introduces an alternative model where the scope of the states is expanded by adding a third level, this is known as the ternary finite field model.

The application presented in this work tends toward the univariate version of the binary finite field model.

3.2 Reverse engineering genetic networks

3.2.1 Preliminaries

In general the term reverse-engineering can be defined as follows:

Definition 8 *Reverse engineering is the general process of analyzing a subject system to identify its components and their interrelationships, and create representations of the system in another form.*

In this work, we consider reverse-engineering in terms of the following definition:

Definition 9 *The reverse-engineering problem for genetic regulatory networks is the problem of determining the network that describes functional relations between genes, given a set of experimental data (gene expression data).*

Gene regulatory networks (GRN) represent the set of all interactions among genes. We are interested in tackling the problem of reverse engineering genetic regulatory networks from time-series gene-expression through the finite field univariable model.

A GRN with m genes can be represented by a finite dynamical system.

Definition 10 *A Finite Dynamical System (FDS) is an ordered pair (X, f) where X is a finite set and f is a function $f : X \to X$.*

Definition 11 *The state diagram of a FDS (X, f) is the digraph whose nodes are members of X and whose edges are the set of all $(x, f(x))$, where $x \in X$*

In an n-dimensional FDS state diagram, each node represents the states of the n genes at a determined time of the time-series, while the edges represent transitions between states.

A network of m genes in the multivariate finite field model is represented by the FDS $(GF(p)^m, f)$. The state of each gene i is represented by an $a_i \in GF(p)$ and the next state of gene i is given by the value of a function $f_i(a_1, a2, \ldots, a_m) \in GF(p)$. Given a state $(a_1, \ldots, a_m)$ of the network, the next state is thus given by the function

$$f(a_1, \ldots, a_m) = f_1(a_1, \ldots, a_m), f_2(a_1, \ldots, a_m), \ldots, f_m(a_1, \ldots, a_m)$$

A network in the univariate finite field model is represented by the FDS $(GF(p^m), f)$. In this case, each $\alpha \in GF(p^m)$ represents a state of the m genes and each value of f represents the next states of the m genes, given the present state. In either model there are a total of p^m possible states, but in practice we have information on only a small fraction of these.

For k data points in the time-series expression data, we know k states in $GF(p^m)$; that is $s_0, s_1, s_2, \ldots, s_{k-2}, s_{k-1}$. In this way the dynamics of the network is described

by the time series

$$f(s_0) = s_1, f(s_1) = s_2, \ldots, f(s_{k-2}) = s_{k-1}. \tag{3.1}$$

3.2.2 The Reverse Engineering Problem for the Univariate Finite Field Model

In the context of univariate finite field models, the reverse engineering problem is stated 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 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 set of all such polynomials g constitutes an ideal. 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. Efficient means for computing the interpolation step of reverse engineering for $p = 2$ are provided in Chapter 5.

3.2.3 Dealing with Large Genetic Networks

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 [102].

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, our methods for finite field arithmetic are defined on fields $GF(2^m)$. Nevertheless, one can take advantage of the isomorphism $GF((2^r)^s) \approx GF(2^{rs})$ in order to extend the model for representing networks where each gene has 2^r states.

In practice, m can be quite large. For example, [82] outlines a study of gene regulatory networks in yeast. Yeast has 6000+ genes. Their 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 and expensive arithmetic computations over finite fields. 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.

Fast finite field multiplication on $GF(2^m)$ can be achieved by using Zech logarithm tables or by directly performing bit-level operations on CPU registers. But these approaches are not practical for large finite fields (see Sections 4.1.1 and 4.1.2 for an explanation). On the other hand, composite fields can be useful for dealing with large genetic networks, but the composite field representation is not optimal for exploiting software and hardware resources for acceleration purpose (see Section 4.1.3 for details). In the following chapter we present an efficient solution for carrying out fast finite field multiplication for large fields.

Chapter 4

Finite Field Multiplication in $GF(2^m)$

> Speed has always been
> important otherwise one
> wouldn't need the computer.
>
> Seymour Cray

4.1 Fast Arithmetic in $GF(2^m)$

The most common, as well as the slowest operation in most finite field application is
multiplication. Our aim is to develop an efficient multiplier for $GF(2^m)$. To achieve
this goal, we have considered several approaches: the table lookup method, the direct
computation on hardware registers via bit-level operations, and the composite fields
method.

4.1.1 Lookup Tables

A lookup table is a data structure that associates keys with values, which is used to
replace a runtime computation with a simpler lookup operation. The speed gain can
be significant, since retrieving a value from memory is often faster than undergoing an

expensive computation. Since there is no computation involved, the look-up operation is performed in constant time $O(1)$.

Using power representation, multiplications can be implemented by first adding the exponents of the operands, followed by an integer modulo reduction. Similarly, exponentiation can be implemented easily. However the penalty of choosing the power representation results in more complicated additions. A useful approach for finite field additions in power representation is the so-called Zech Logarithms or simply *Zech log*. Details about the definition and properties of Zech's Logarithms can be found in [61].

Definition 12 Zech log*: Let α be a primitive element of a finite field, then $Z(i)$, the Zech log of an integer i may be defined such that*

$$\alpha^{Z(i)} = 1 + \alpha^i \qquad (4.1)$$

It should be noted that every nonzero element of $GF(2^m)$ has a unique representation in the form $1 + \alpha^i$, note also that for $a \leq b$, $\alpha^a + \alpha^b = \alpha^a(1 + \alpha^{b-a}) = \alpha^{a+Z(b-a)} \bmod (2^m - 1)$. Addition is thus performed by adding one exponent to the Zech log of the difference of two exponents, these Zech logs are found in a precomputed table. In this way we can perform fast multiplications and additions by table lookup, avoiding costly computations.

Example 1 *Table 4.1 is useful for performing arithmetic over $GF(2^4)$.*

In Table 4.1 powers of α are used, this representation was described in section 2.3.

*The right column corresponds to the Zech logarithms, which are defined by $z(i) = log(\alpha^i + 1)$, $z(0)$ is denoted by *****.*

Table 4.1: Lookup table for $GF(2^4)$ with Zech logarithms

i	α^i	$z(i)$
0	0001	*
1	0010	4
2	0100	8
3	1000	14
4	0011	1
5	0110	10
6	1100	13
7	1011	9
8	0101	2
9	1010	7
10	0111	5
11	1110	12
12	1111	11
13	1101	6
14	1001	3

Multiplication: $\alpha^9 \cdot \alpha^5 = \alpha^{9+5} = \alpha^{14}$

Addition: $\alpha^7 + \alpha^3 = \alpha^3(\alpha^4 + 1) = \alpha^3\alpha^{z(4)} = \alpha^3\alpha^1 = \alpha^4$

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 actual reverse engineering problems.

4.1.2 Finite Field Arithmetic via bit-level Operations

We have seen in section 2.3 that an element in $GF(2^m)$ can be represented as a sequence of m bits in $GF(2)$. 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.

Example 2 *The addition of the two elements $\alpha^4 + \alpha^2 + \alpha$ and $\alpha^4 + \alpha + 1$ over $GF(2^5)$ is defined by*

$$
\begin{array}{ccccc}
1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
\hline
0 & 0 & 1 & 0 & 1
\end{array}
$$

Notice that addition operation in GF(2) (as defined in Table 2.1) is just the XOR of two words. This is possible because there is no carry propagation. So, addition is a very fast operation whose complexity is constant. A very natural approach for multiplication in $GF(2^m)$ is to multiply two elements in the field as polynomial multiplication modulo a irreducible polynomial, using the school-book method for polynomial multiplication. This operation is accomplished using simply left-shifts and XORs. We call this simple procedure the *direct method* for multiplication [36]. The following example illustrates direct multiplication.

Example 3 *Consider the multiplication $(\alpha^3 + \alpha^2 + \alpha) \times (\alpha^4 + \alpha + 1)$ in the finite field $GF(2^5)$ defined by the irreducible polynomial $t^5 + t^2 + 1$. The direct multiplication is illustrated in Figure 4.1.*

Taking advantage of the hardware architecture, shifts and XOR operations are accomplished as bit-parallel operations on CPU registers, so this method has complexity $O(m)$. But in the basic implementation of this method, the field size is limited by the architecture word-length. Although some data structures could work for overcoming

$$
\begin{array}{l}
\alpha^3 + \alpha^2 + \alpha \longrightarrow \quad 0\ 1\ 1\ 1\ 0 \\
\alpha^4 + \alpha\ + 1 \longrightarrow \quad \underline{1\ 0\ 0\ 1\ 1} \\
\qquad\qquad\qquad\qquad\quad 0\ 1\ 1\ 1\ 0 \\
\qquad\qquad\qquad\quad 0\ 1\ 1\ 1\ 0 \\
\qquad\qquad\qquad 0\ 0\ 0\ 0\ 0 \\
\qquad\qquad\quad 0\ 0\ 0\ 0\ 0 \\
\qquad\qquad 0\ 1\ 1\ 1\ 0 \\
\qquad\qquad \underline{} \\
\qquad\quad 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\
\qquad\qquad 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\
\qquad\qquad\quad 1\ 0\ 0\ 1\ 0\ 1\ 0 \\
\qquad\qquad\qquad 1\ 0\ 0\ 1\ 0\ 1 \\
\qquad\qquad \underline{} \\
\qquad 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1 \longleftarrow \alpha^3 + 1
\end{array}
$$

Figure 4.1: Example of finite field multiplication over $GF(2^5)$

the field size limit, the overall performance of these implementations is affected by lack of plain bit-level parallelism and straightforward operations.

In order to deal with large finite fields, we can take an additional advantage by combining the direct multiplication method presented in this Section with the table lookup method presented in Section 4.1.1. In this sense, we bring into play the idea of *composite fields*.

4.1.3   Composite Fields

A special type of finite fields $GF(2^k)$ where the exponent is a composite integer $k = nm$ is commonly called a composite field. If k is a composite number $k = nm$, then it is possible to derive a different representation method by defining $GF(2^k)$ over $GF(2^n)$. This is an extension field which is not defined over the prime field but one of its subfields.

A composite field is denoted as $GF((2^n)^m)$ where $GF(2^n)$ is known as the ground field over which the composite field is defined.

Since the fields $GF(2^{nm})$ and $GF((2^n)^m)$ are isomorphic, we can choose conveniently the ground and extension field sizes in order to represent finite field operations over $GF(2^{nm})$ as operations in $GF((2^n)^m)$ an use table lookup for operations in the ground field $GF(2^n)$.

Elements in $GF((2^n)^m)$ are then of the form $f_0 + f_1\beta + f_2\beta^2 + \cdots + f_{m-1}\beta^{m-1}$ where each f_i is of the form $f_i = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{n-1}\alpha^{n-1}$, each $a_j \in GF(2)$, α is a root of an irreducible polynomial of degree n over $GF(2)$, and β is the root of an m-degree irreducible polynomial over $GF(2^n)$. It can be shown that an m-degree polynomial which is irreducible over $GF(2)$ is also irreducible over $GF(2^n)$ if $\gcd(n,m) = 1$.

By choosing judiciously the ground and extension field some computation could be shifted to the extension field, while the operations in the ground field could be performed by reasonably sized lookup tables. This technique trades additional computation for a significant decrease in storage space.

Obviously the field size in a composite field is conditioned to a composite number nm, where m and n are relatively primes. Furthermore, the choice of m and n are dependent of the available hardware resources. In this way, the amount of available memory constrains the ground field size and leads to inconsistent performance across architectures. In [123] Shu *et al.* have addressed this issue by avoiding lookup tables and combining bit-parallel operations in the ground field with serial operations in the extension field. However, this method is feasible only for composite fields gen-

erated by low Hamming weight irreducible polynomials, where composite fields can be constructed via an irreducible pentanomial of degree nm, but not an irreducible trinomial of degree nm.

It has be claimed that the property of composite fields could be used to derive more efficient multipliers [118, 131]. However, according to Shu and colleagues [123], to date there has not been enough solid empirical evidence to support this view, in particular for FPGA implementations.

The access from the FPGA to memory is expensive, regardless of whether it is a local memory attached directly to the FPGA or the host memory. Therefore, combining direct computation with lookup tables on an FPGA implementation of a composite field multiplier becomes a challenging effort. On the other hand, given that on current FPGAs it is possible to allocate data using tailored registers, we can think about straightforward implementations using registers as long as needed, avoiding excessive communications between FPGAs and memories.

4.2   The Mastrovito Method

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

$$C(\alpha) = A(\alpha) * B(\alpha) \mod f(\alpha) \tag{4.2}$$

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 $2m - 2$.

$$Q(\alpha) = A(\alpha) * B(\alpha) = \left(\sum_{i=0}^{m-1} a_i \alpha^i \right) \left(\sum_{i=0}^{m-1} b_i \alpha^i \right) \tag{4.3}$$

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 (4.3) 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 $A(\alpha)$. This is:

$$
\begin{bmatrix}
q_0 \\
q_1 \\
\vdots \\
q_{m-2} \\
q_{m-1} \\
q_m \\
q_{m+1} \\
\vdots \\
q_{2m-3} \\
q_{2m-2}
\end{bmatrix}
=
\begin{bmatrix}
a_0 & 0 & 0 & \cdots & 0 & 0 \\
a_1 & a_0 & 0 & \cdots & 0 & 0 \\
\vdots & & & \ddots & & \\
a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 & 0 \\
a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 \\
0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\
0 & 0 & a_{m-1} & \cdots & a_3 & a_2 \\
\vdots & & & \ddots & & \\
0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\
0 & 0 & 0 & \cdots & 0 & a_{m-1}
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1 \\
b_2 \\
\vdots \\
b_{m-1}
\end{bmatrix}
$$

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 $GF(2^m)$, here $g(\alpha)$ is an n-degree polynomial. The 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:

$$
N[m, n] = \left\lceil \frac{m-1}{\Delta} \right\rceil \tag{4.4}
$$

where $\Delta = m - n$ [52].

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 $N[3, 2] = 2$.

Instead of performing finite field arithmetic in two steps as described above, the

Mastrovito method formulates these two steps into a single matrix-vector product.

The modular reduction is previously computed over the matrix M obtaining an al-

ready reduced $m \times m$ dimensional matrix Z, this matrix is called the *Mastrovito*

matrix. As result, the finite field multiplication defined in (4.2) is computed by

$C = ZB$.

4.3   A New FPGA-based Approach

A common method for obtaining efficient FPGA implementations is by means of

pipelining. In the case of multiplication in $GF(2^m)$, such an approach can be ef-

fected by exploiting the symmetries of the Mastrovito matrix. In this section we are

introducing a pipelining design for computing efficiently the matrix-vector product

involved in the Mastrovito method.

A method for constructing the Mastrovito matrix is proposed in [52]. According

to this method if $GF(2^m)$ is defined by the trinomial $\alpha^m + \alpha^n + 1$ then Z is given by

$$Z = \begin{bmatrix} U \\ \\ L \end{bmatrix} \tag{4.5}$$

where U and L are Toeplitz matrices defined as follows:

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

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

$$[a_0] || \sum_{i=0}^{N-1} F[i\Delta \to] \tag{4.6}$$

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

$$G + \sum_{i=0}^{N-1} F[i\Delta \to] \tag{4.7}$$

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

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

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

and so L_0 is

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

The multiplication is computed in m 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* $m - 1$
        $c_{(i+n) \bmod m} \leftarrow S \cdot B$
        $S \leftarrow$ *right-shift(S)*
        $s_0 \leftarrow a_i$
    end for
    return(C)

The proposed multiplier is designed by exploiting the symmetries that take place in the Mastrovito matrix when the field is defined by an irreducible trinomial, but they do not exist for some field degrees. For instance, there exists 269 irreducible

trinomial $x^m + x^n + 1$ over $GF(2)$ with degrees $m < 512$, the rate is about one half (0.52). Actually, Paszkiewicz has determined in [106] that the rate for which an irreducible polynomial over $GF(2)$ and degree $m < 10000$ exists is a bit greater than 0.5.

4.4 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 about the fundamentals of FIR filters and FPGA implementations can be found in [94].

A traditional architecture of a FIR filter is shown in figure 4.2. 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.
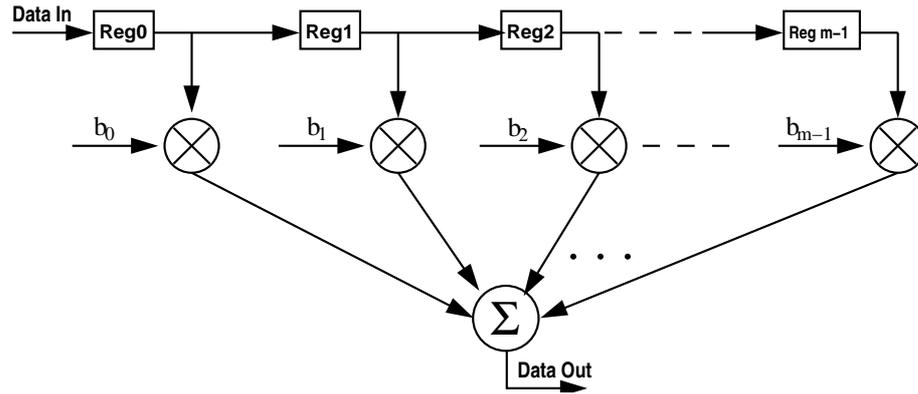
Figure 4.2: A m-Tap FIR filter traditional architecture.

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 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 4.3. 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
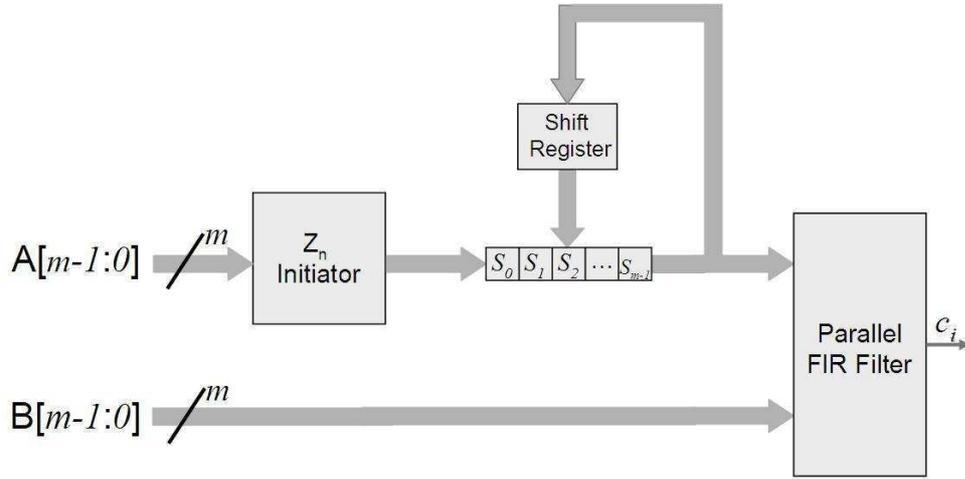
Figure 4.3: Block diagram of the proposed multiplier.

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.

4.5 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 [44], 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 [75], 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 [49], 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 5.2 we compare our approach with the mentioned polynomial basis multipliers reported in [44, 49, 75]. 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.

We have synthesized over different target devices for comparison propose, 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 5.2 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 4.2: Multipliers comparison

Field	Target FPGA	Implementation	Time (μs)	Space (slices)
$GF(2^{210})$	Xilinx Virtex xcv-300-6	Reference [75]	12.30	343
		This work	2.21	334
$GF(2^{233})$	Xilinx xc2v-6000-4	Reference [49]	2.58	not reported
		This work	2.42	415
$GF(2^{239})$	Xilinx Virtex xcv-300-6	Reference [44]	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 5.4). 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 [34] have reported similar communication problems with the Cray XD1. However, in [130] 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 4.3: 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%

4.6   Finite Field Inversion

Among the arithmetic operations needed in the interpolation phase of the reverse engineering problem, the inversion is the most time consuming operation. Finite field inversion is computed by repeated multiplications, various methods for finite field inversion were described in Section 2.4.2. The Itoh-Tsujii algorithm [65], which is based on Fermat's Little Theorem has been reported to be an efficient alternative for FPGA implementations [116].

Fermat's little theorem states that if α is a nonzero element of $GF(p)$ then $\alpha^{p-1} \equiv 1 \pmod{p}$. In terms of binary extension fields, this theorem establishes that for any nonzero element $\alpha \in GF(2^m)$, the identity $\alpha^{-1} \equiv \alpha^{2^m-2}$ holds. Thus, multiplicative inversion can be computed by:

$$\alpha^{-1} = \alpha^{2^m-2} = \alpha^{\sum_{i=1}^{m-1} 2^i} = \alpha^{2^1} \times \alpha^{2^2} \times \cdots \times \alpha^{2^{m-1}} \tag{4.8}$$

Equation 4.8 can be carried out through $m-1$ squaring and $m-2$ multiplications by using the binary exponentiation method. However, the Itoh-Tsujii method reduces

the number of multiplications to $k + HW(m-1) - 1$, and the number of exponentiations to $k + HW(m-1)$, where $k = \lfloor \log_2(m-1) \rfloor$, and $HW(m-1)$ is the Hamming weight (i.e. the number of nonzero bits) of the binary representation of $m-1$. The Itoh-Tsujii algorithm computes $\alpha^{2^m - 2}$ by using recursive rearrangements of the finite fields operation involved in equation 4.8. The algorithm is derived from the following theorem:

Theorem 1 *Let A be any arbitrary nonzero element in the field $GF(2^m)$. Let us consider*

$$m - 1 = 2^{k_1} + 2^{k_2} + \cdots + 2^k_{t-1} + 2^{k_t}$$

where $k_1 > k_2 > \cdots > k_{t-1} > k_t$. For each k in $\{k_1, k_2, \ldots, k_{t-1}, k_t\}$ consider the sum $s_k = \sum_{i=1}^{2^k} 2^i = 2 + 2^2 + 2^3 + \cdots + 2^{2^k}$. Then the multiplicative inverse of A, can be written as

$$A^{-1} = A^{2^{m-1} + \cdots + 2^2 + 2} = (A^{s_{k_t}}) \left(\cdots (A^{s_{k_3}}) \left[(A^{s_{k_2}}) (A^{s_{k_1}})^{2^{2^{k_2}}} \right]^{2^{2^{k_3}}} \cdots \right)^{2^{2^{k_t}}}$$

Proof: See sketch in [117]. A more detailed proof can be found in [50].

In order to carry out inversion over finite fields through the Itoh-Tsujii algorithm, it is essential to perform two basic operations: finite field multiplication, whose algorithm has been described in Section 4.3, and the binary exponentiation which is

implemented by repeated multiplications such as depicted in Algorithm 2. We derived from Theorem 1 the Itoh-Tsujii algorithm presented in Algorithm 3.

Algorithm 2 bin-power
   Input: X, l
   Output: $Y = X^{-1}$

   $X \leftarrow X \cdot X$
   For $i = 0$ to l
       For $j = 1$ to 2^{l-1}
           $X \leftarrow X \cdot X$
       EndFor
       $Y \leftarrow X$
   EndFor
   Return(X)

Algorithm 3 Itoh-Tsujji Algorithm
   Input: $A \in GF(2^m)$, $A \neq 0$, m
   Output: $C = A^{-1}$

   $j = 1$
   $C \leftarrow A$
   For $i = 0$ to $k_1 - 1$
       $D \leftarrow$ *bin-power(C,i)*
       If $LSB(m-1) = 1$ *{comment: LSB = Least Significant Bit }*
           $As_{k_j} = C$
           $k_j = i$
           j++
       EndIf
       right-shift(m − 1)
       $C \leftarrow C \cdot D)$
   EndFor
   For $t \leftarrow j$ downto 2
       $C \leftarrow As_{k_{t-1}} \cdot$ *bin-power(C,k_{t-1})*
   EndFor
   Return(C)

Example 4 *As an example we consider the inverse of $A \in GF(2^{11})$, $A \neq 0$. Note that $m - 1 = 10 = 2^3 + 2 = 2^{k_1} + 2^{k_2}$. Then*

$$A^{-1} = \left(A^{s^{k_2}}\right)\left(A^{s^{k_1}}\right)^{2^{2^{k_2}}} = \left(A^{2+2^2}\right)\left(A^{2+2^2+2^3+\cdots+2^8}\right)^{2^2}$$

Notice that in computing $A^{s^{k_1}}$, we also compute $A^{s^{k_2}}$. And $A^{s^{k_1}}$ can be computed by an addition chain. Then the computation of A^{-1} is as follows:

A^2

A^{2^2}

$A^2 \cdot A^{2^2} = A^{2+2^2}$

$\left(A^{2+2^2}\right)^{2^2} = A^{2^3+2^4}$

$\left(A^{2+2^2}\right) \cdot \left(A^{2^3+2^4}\right) = A^{2+2^2+2^3+2^4}$

$\left(A^{2+2^2+2^3+2^4}\right)^{2^{2^2}} = A^{2^5+2^6+2^7+2^8}$

$\left(A^{2+2^2+2^3+2^4}\right) \cdot \left(A^{2^5+2^6+2^7+2^8}\right) = A^{2+2^2+2^3+2^4+2^5+2^6+2^7+2^8}$

$\left(A^{2+2^2}\right) \cdot \left(A^{2+2^2+2^3+2^4+2^5+2^6+2^7+2^8}\right)^{2^2} = A^{2+2^2+2^3+2^4+2^5+2^6+2^7+2^8+2^9+2^{10}}$

Although this section was focused in the Itoh-Tsujii inversion algorithm over $GF(2^m)$, this algorithm can be used with the same computational complexity on any field $GF(p^m)$. In the next chapter we will study an efficient approach for finite field multiplication over $GF(p^m)$ where p is an odd prime. An efficient FPGA-based implementation of Itoh-Tsuji's algorithm over $GF(2^m)$ is presented in Section 5.3.

Chapter 5

Polynomial Interpolation over $GF(2^m)$

A supercomputer is a device for converting a CPU-bound problem into an I/O bound problem .

Ken Batcher

5.1 Finite Field Polynomial Interpolation on FPGAs

The work presented in this chapter deals with the problem of constructing a function which exactly fits a set of data points over finite fields. Interpolation over finite fields has application in various fields, such as error-correcting codes, cryptography, and learning algorithms [145], and reverse engineering genetic networks. In this work, finite field polynomial interpolation is used as a means toward solving the reverse engineering problem for genetic networks as described in Chapter 3.

Finite field polynomial interpolation for reverse engineering large genetic networks can be very computationally intense. In order to accelerate the interpolation process, we introduce an efficient architecture for an FPGA implementation. But, the suit-

ability of this problem for a HPRC based solution should be judiciously analyzed. In this way, selecting an appropriate algorithm for an FPGA implementation becomes a challenging effort. A first approach is to consider well performed algorithms for polynomial interpolation. In this sense, Lipson's algorithm promises to be a good candidate given its low computational complexity, nevertheless the suitability of this algorithm for an efficient FPGA implementation has to be validated. In the following section we will consider the optimality of Lipson's algorithm for an FPGA solution beyond the computational complexity.

5.2 Lipson's Algorithm for Interpolation

Different methods exist for finite field polynomial interpolation, some classical methods such as Lagrange or Newton's interpolation are relatively simple. They are derived from simple formulas, but their computational complexity is $O(n^2)$, where n is the number of points to be interpolated. In contrast, other methods such as Lipson's algorithm [86], which is based on the Chinese remainder theorem has complexity $O(n \log^2 n)$. Bollman *et al.* [14] have shown that a parallel version of this algorithm is suitable for speeding up finite field interpolation on shared memory systems.

Lipson's algorithm builds a solution using the divide-and-conquer idea. It develops the interpolation polynomial as a recursive sequence of polynomial multiplications and additions. In each cycle a number of polynomials are constructed. Each polynomial is obtained by multiplying and adding pairs of polynomials in a tree-like fashion (see for example Figure 5.1).

Given the following points in $GF(2^3)$:

$(\alpha^4,1),(1,\alpha),(\alpha,0),(0,\alpha^5),(\alpha^5,\alpha^3),(\alpha^3, \alpha^6),(\alpha^6,\alpha^2),(\alpha^2,\alpha^4)$,

find the polynomial P(X) that interpolates those points.

The Field $GF(2^3)$ is generated by $\alpha^3+\alpha+1$

i	α^i	*decimal*
*	000	0
0	001	1
1	010	2
2	100	4
3	011	3
4	110	6
5	111	7
6	101	5

$P(X) = \alpha^6 X^6 + X^5 + \alpha^5 X^4 + \alpha^2 X^3 + \alpha X + \alpha^5$
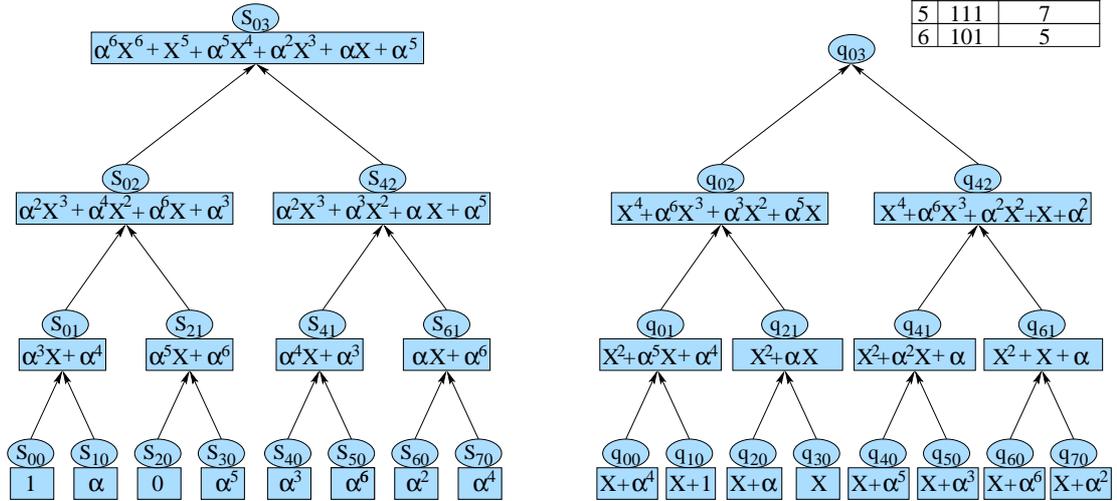
Figure 5.1: Example of interpolation using Lipson's Algorithm.

Despite its low asymptotic complexity, this approach requires a large amount of storage space. For this reason, it is not an optimal algorithm for an FPGA implementation.

It is not unusual to encounter algorithms that are optimal for CPU-based implementations that are not necessarily the best approach for FPGA-based implementations. This issue is addressed by Herbordt *et al* in [56]. In this sense, Newton's algorithm is a better alternative than Lipson's for implementing an efficient solution on FPGAs. Newton's interpolation design is based on multiply-and-accumulate (MAC) sequences, so it makes the most of the area in the FPGA. Results shown in

Table 5.1 support an assessment regarding the suitability of Lipson's and Newton's interpolation as an efficient FPGA-based solution. As can be seen from the table, using Lipson's algorithm, an Xilinx Virtex II Pro 50 FPGA becomes saturated when only eight points in $GF(2^{63})$ are interpolated.

Table 5.1: Performance comparison of Lipson's and Newton's interpolation over $GF(2^{63})$ implemented on an FPGA Xilinx Virtex II Pro 50.

Points	Interpolation Algorithm	Time ($\mu sec.$)	Space (% slices)
4	Lipson	152.3	88.9
	Newton	146.4	32.4
8	Lipson	697.6	103.1
	Newton	815.3	36.3

5.3   Univariate Newton's Interpolation

Newton's interpolation algorithms can be used over any field to obtain the coefficients c_i of a univariate polynomial $f(x) = \sum_{i=0}^{t-1} c_i x^i$ of degree $t-1$ from the values at arbitrary t points.

Let $G_{j-1}(x)$ interpolate $j-1$ points (x_k, y_k), $1 \leq k < j$, so that $G_{j-1}(x_k) = y_k$. Also let

$$D_{j-1}(x) = (x - x_1) \cdots (x - x_{j-1}) \tag{5.1}$$

Then we can compute $G_j(x)$ by the formula

$$G_j(x) = [y_j - G_{j-1}(x_j)] \frac{D_{j-1}(x)}{D_{j-1}(x_j)} + G_{j-1}(x) \tag{5.2}$$

Finite field polynomial interpolation is very computationally intense. As it happens with any other interpolation algorithm, in Newton's algorithm the most frequently used operation by far is multiplication, and furthermore, this operation tends to be the slowest. In [37] we have exploited the bit-level representation of the field elements in $GF(2^m)$ to develop a fast and space-saving architecture for finite field multiplication on FPGAs. Our finite field multiplier has been successfully used for achieving efficient computation of inverses of a finite field element, as well as finite field polynomial multiplication and evaluation. In the next subsections we will depict the design of the three aforementioned operations, which are used for implementing the overall architecture of Newton's interpolation.

5.3.1   Architecture of Newton's Interpolation

In this section, we present an efficient architecture for computing univariate polynomial interpolation. The idea is a natural extension of the formula 5.2, which operates over accumulative results. By exploiting this feature, we have produced a compact design that leads to an area-efficient architecture. This serial architecture computes an interpolation polynomial P in n steps, where n is the number of input points $\{(x_1, y_1) \dots (x_n, y_n)\}$.

The main functional units in the overall architecture are polynomial multiplication, evaluation and inversion. They work together in each cycle in order to update values of D and G toward the interpolation polynomial P. This architecture is drawn in the block diagram of figure 5.2.
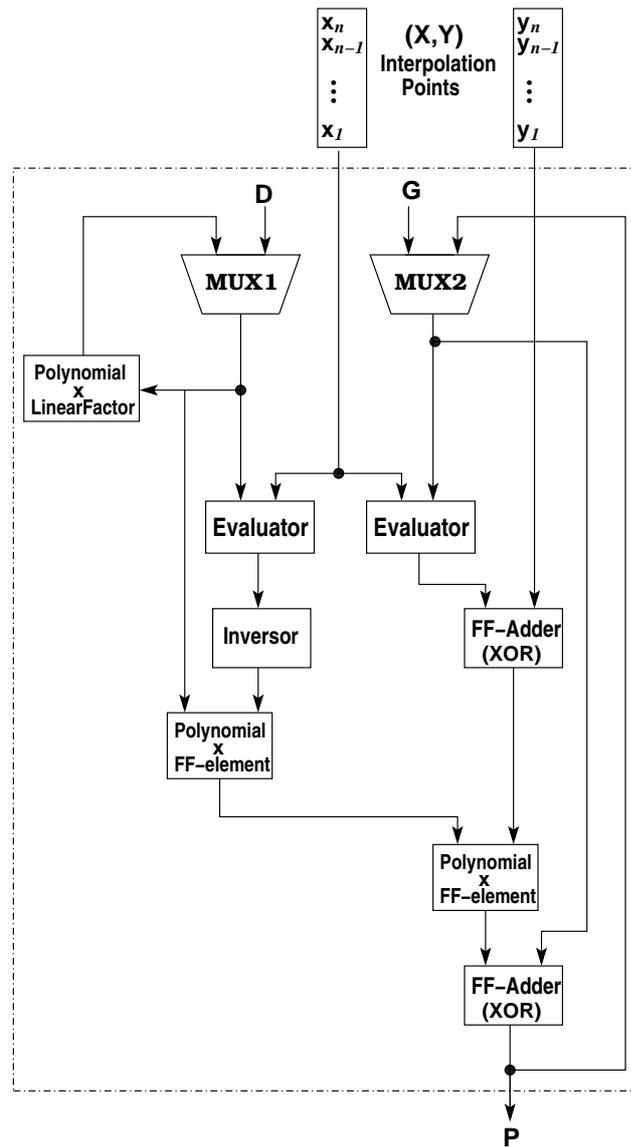
Figure 5.2: Block diagram for the general architecture of Newton's interpolation

5.3.2   Polynomial multiplication

In order to develop the Newton formula into an FPGA implementation, it is required to carry out polynomial multiplications. However only two very simple special cases of polynomial multiplication are necessary, which are:

1. The product of a polynomial with a finite field element (i.e. the product of an i-degree polynomial with a 0-degree polynomial, $0 < i < n$), which is used for computing the products involved in the formula 5.2.

2. The product of a polynomial with a linear factor (i.e. the product of an i-degree polynomial with a 1-degree monic polynomial, $0 < i < n$), which is used for computing the products involved in the formula 5.1.

5.3.3   Evaluation

In our interpolation architecture, the polynomial evaluation is addressed by using Horner's rule [23]. This is an algorithm for efficient polynomial evaluation which reduces the number of necessary multiplications by simply factoring out powers of x, giving

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 = (\cdots (a_n x + a_{n-1})x + \cdots)x + a_0 \tag{5.3}$$

The FPGA implementation of Horner's rule develops into a simple design which computes the evaluation of the finite field element X over an n-degree polynomial in n cycles, given the sequence of $n + 1$ polynomial coefficients $[c_n, c_{n-1}, \ldots, c_0]$. This logic is depicted in figure 5.3.
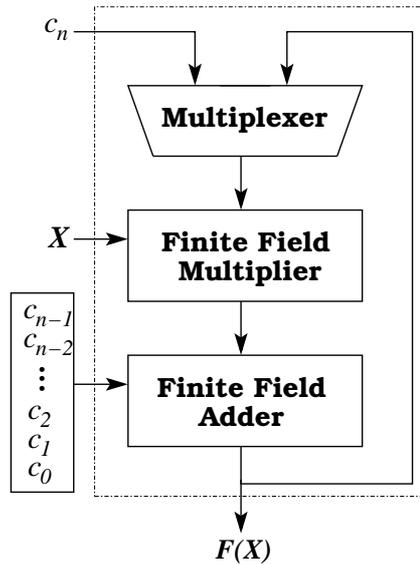
Figure 5.3: Block diagram for polynomial evaluation using Horner's algorithm

5.3.4 Inversion

In order to compute the finite field divisions involved in Newton's formula (formula 5.2), we use the Itoh-Tsujii algorithm for inversion over finite fields in the standard basis. This is an exponentiation-based inversion algorithm which reduces the complexity of computing the inverse of non-zero elements in $GF(2^m)$ by computing binary exponentiation through addition chains. Details of the construction of this inversion algorithm are shown in [50].

An efficient FPGA implementation of the Itoh-Tsujii algorithm is presented by Rodríguez *et al.* in [114]. They consider both the standard and parallel versions of the algorithm where the dataflow is influenced by Finite State Machines (FSM). However, instead of using an FSM-based design, we use the straightforward architecture described in figure 5.4, which reduces the area used, although it slightly affects the run

time of inversion. For instance, on a Virtex 2 xc2v4000-6bf957, the design proposed by Rodríguez *et al.* computes an inverse over $GF(2^{193})$ in 0.76 μ-seconds and 2345 slices, while our architecture spends 1.03 μ-seconds and 613 slides. In the context of Newton's interpolation, our inversion design is a reasonable alternative, since the number of inversions required for interpolation does not grow asymptotically. More precisely, n inversions are required for interpolating n points.

5.4 Numerical experiments

This research is focused on accelerating interpolation over fields $GF(2^m)$ using FPGAs for an efficient solution of the reverse engineering problem for genetic networks 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.

The tools included in the Xilinx ISE Foundation 9.1i development toolset have been used for the design, synthesis, implementation and verification of results. FPGA-based designs are implemented in VHDL, while CPU-based algorithms are developed using the C language, compiling on a single 2.2 GHz Opteron AMD.

The Cray XD1 is supplied with standard primitives for CPU-FPGA interaction and they were used for setting, loading and running FPGA applications for the Cray XD1's CPU. Opterons communicate with the FPGA using API (Application Programming Interface) calls across a proprietary bus called the RapidArray Transport
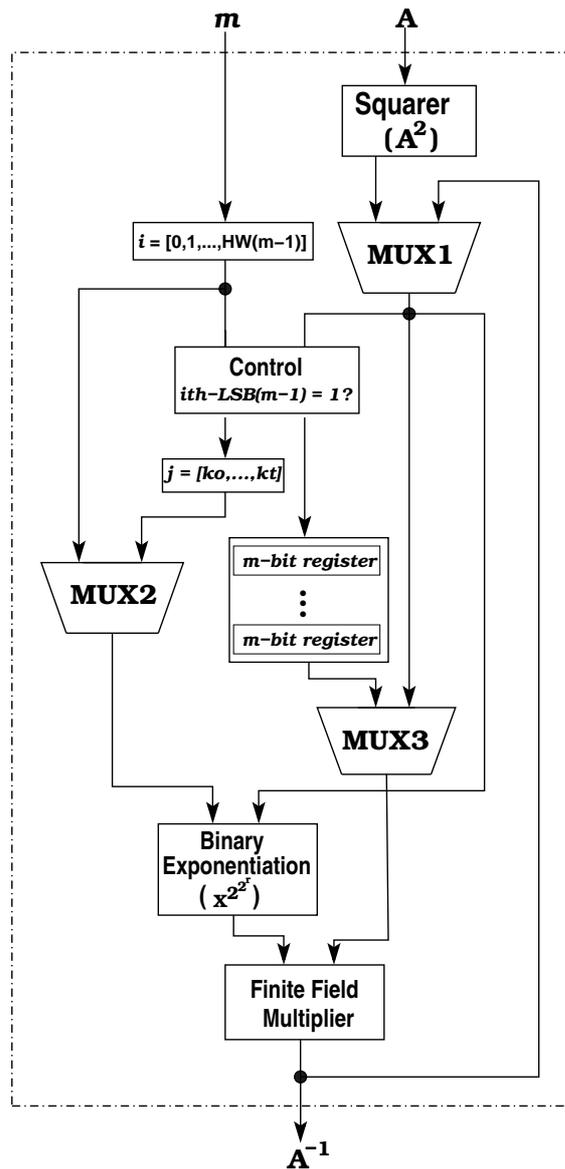
Figure 5.4: Block diagram for the $GF(2^m)$ Itoh-Tsujii inversion algorithm.

Table 5.2: Performance comparison of interpolation algorithm

Field	Points	Time (*msec.*)		Space
		CPU	FPGA	Utilization
$GF(2^{193})$	4	260.5	2.2	45.84 %
	40	3452.8	25.8	54.33 %
$GF(2^{210})$	4	443.0	4.3	47.83 %
	40	6140.6	49.9	56.69 %
$GF(2^{233})$	4	734.0	7.4	50.61 %
	40	10815.5	96.1	59.98 %
$GF(2^{239})$	4	843.8	8.0	51.35 %
	40	12508.1	107.9	60.86 %
$GF(2^{303})$	4	2702.1	21.9	59.73 %
	40	46989.5	337.9	70.79 %
$GF(2^{333})$	4	4316.8	37.1	63.95 %
	40	80650.0	605.6	75.79 %
$GF(2^{441})$	4	18915.2	179.3	80.79 %
	40	448710.0	3640.4	95.75 %
$GF(2^{471})$	4	34995.2	3336.7	85.94 %
	40	875627.0	7470.1	99.14 %

Core (RT-Core) [27].

Typically the communication model used in CPU-FPGA communication 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. In [36] we concluded that this is a costly communication model. For this reason, we have used as alternative the I/O subsystem developed by Ohio Supercomputer Center (OSC) [34]. This is an efficient general purpose I/O interface to the RTCore. Instead of pushing data into the FPGA register, this subsystem is able to allocate shared memory space for communication between the FPGA and CPU by using a dedicated memory space in the host called the FPGA transfer region (`ftrmem`) [26]. Therefore, our implementation uses a pull communication model where the FPGA pulls the input data from

the source, and finally stores the results in the destination. The overhead associated with the CPU-FPGA communication is included in the time measurements presented in the FPGA column of Table 5.2.

We have performed several measurements in order to evaluate the performance of our architecture. A summary of various comparative results are shown in Table 5.2. These results are used for calculating the acceleration gained by the Virtex II Pro over the 2.2 GHz Opteron processor. This metric is obtained from the rounded quotient:

$$Acceleration\ Factor = \frac{Runtime_{CPU}}{Runtime_{FPGA}} \tag{5.4}$$

The choice of values for our experiments was motivated by our intended application to reverse engineering genetic networks. The input data in this application are the result of time series microarray experiments. Microarrays allow researchers to simultaneously measure the expression of thousands of genes. However, the number of points in the time series of genes is much less than the number of genes due to the high cost of microarray experiments. Typically, the number points in the time series will be less than 20 and rarely more than 40 [8]. Thus, we would like to make the field size m as large as possible, but maintain a range for the number of points between, say 4 and 40.

It should be noted that problem size for interpolation over a fixed field is usually measured in terms of the number of points n. However, for finite fields $GF(2^m)$ the

Table 5.3: Acceleration factor of interpolation algorithm

Field	Irreducible Polynomial	Points	Acceleration Factor
$GF(2^{193})$	$x^{193} + x^{15} + 1$	4	118X
		40	132X
$GF(2^{210})$	$x^{210} + x^7 + 1$	4	103X
		40	122X
$GF(2^{233})$	$x^{233} + x^{74} + 1$	4	99X
		40	112X
$GF(2^{239})$	$x^{239} + x^{36} + 1$	4	105X
		40	115X
$GF(2^{303})$	$x^{303} + x + 1$	4	123X
		40	139X
$GF(2^{333})$	$x^{333} + x^2 + 1$	4	116X
		40	133X
$GF(2^{441})$	$x^{441} + x^7 + 1$	4	105X
		40	123X
$GF(2^{471})$	$x^{471} + x + 1$	4	104X
		40	117X

problem size depends on both n amd m. In our case, we hold n constant and vary m. We note that execution times increase with field size. However, the acceleration factor tends to be constant regardless of the field size. This makes it feasible to scale the size of the problem as long as the area space allows. This means in our application context, that the reverse engineering problem can be extended to larger genetic networks without sacrificing performance. Although the acceleration factor is independent of the field size, it is affected by the irreducible polynomial which defines the finite field. This is because the Mastrovito-based multiplier [37] used in this interpolation architecture defines the number of bit-level operations according to a generating trinomial $x^m + x^n + 1$. More precisely, the number of operations in the Mastrovito multiplier is in proportion to n. This can be noted in Table 5.4,

where the execution time of multiplication is slightly affected by n, consequently the acceleration factor is also affected since multiplication is the dominant operation in any polynomial interpolation algorithm.

Table 5.4: Finite field multiplication over $GF(2^{471})$.

Irreducible Polynomial	Time ($\mu sec.$)
$x^{471} + x + 1$	4.87
$x^{471} + x^{119} + 1$	5.39
$x^{471} + x^{470} + 1$	6.89

Chapter 6

Finite Field Multiplication in $GF(p^m)$ with $p \neq 2$

> Just because something doesn't
> do what you planned it to do
> doesn't mean it's useless.
>
> Thomas A. Edison

6.1   Introduction

In the preceding chapters we have developed fast finite field designs for fields $GF(2^m)$.

The question arises if similar designs could be developed for fields $GF(p^m)$ of arbitrary

prime characteristic p. Such results will be of benefit to finite field models for genetic

networks. In the characteristic 2 model, it is assumed that every gene is either is

on (active) of *off* (inactive). However, an arbitrary prime characteristic p model,

would allow for more flexibility in the discretizing microarray data. In this chapter

we present a new algorithm for multiplication in certain fields $GF(p^m)$ with $p \neq 2$.

Multiplication in the field $GF(p^m)$ can be regarded as multiplication of polynomi-

als over $GF(p)$ modulo an irreducible polynomial over $GF(p)$ that defines $GF(p^m)$.

This method requires $O(m^2)$ mod p operations. Our new algorithm, based on convo-

lution requires only $O(m \log m)$ mod p operations. Several authors [7], [18] suggest this method for the actual polynomial multiplication in $GF(p^m)$, but they ignore the second step necessary, which consists of reducing the polynomial product modulo the polynomial that defines the field.

In this chapter we show that for a certain family of finite fields $GF(p^m)$, multiplication, including the reduction step, can be expressed in terms of convolution. This result is achieved by showing that a variant of the Mostrovito matrix can be embedded in a circulant matrix and then using the fact that the product of a circulant matrix and a vector can be expressed as a convolution.

We use the following notation: A column (row) vector whose elements are a_i, $i = 0, 1, 2, \ldots, n-1$ is denoted by $[a_0, a_1, \ldots, a_{n-1}]$ ($[a_0 \ a_1 \ \ldots \ a_{n-1}]$). For any matrix M, M^T denotes the transpose of M. Thus, $[a_0, a_1, \ldots, a_{n-1}]^T = [a_0 \ a_1 \ \ldots \ a_{n-1}]$. For any two vectors $a = [a_0, a_1, \cdots, a_{n-1}]$ and $b = [b_0, b_1, \cdots, b_{n-1}]$ over a ring R, we denote the inner product $a_0 b_0 + a_1 b_1 + \cdots a_{n-1} b_{n-1}$ by $a \cdot b$. For any matrix M, we denote by M_i the $i-th$ row of M, where the rows are numbered starting at zero. Following [131], we use the notation $V[\rightarrow i]$ to represent the (row) vector V shifted right i positions and filling the vacated positions to the left with zeros.

6.2 The Mastrovito Matrix.

Although the Mastrovito matrix has been defined for fields of characteristic 2, the same idea can be applied to a field of arbitrary characteric $GF(p^m)$. If $a = a_0 + a_1\alpha + \cdots a_{m-1}\alpha^{m-1}$ and $b = b_0 + b_1\alpha + \cdots b_{m-1}\alpha^{m-1}$ are elements of $GF(p^m)$, then the vector of coefficients of the ordinary polynomial product $d_0 + d_1\alpha + \cdots d_{2m-2}$ of a and

b can be expressed as follows:

$$
\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{m-2} \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{bmatrix}
=
\begin{bmatrix}
a_0 & 0 & 0 & \cdots & 0 & 0 \\
a_1 & a_0 & 0 & \cdots & 0 & 0 \\
\vdots & & & & & \\
a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 & 0 \\
a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 \\
0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\
0 & 0 & a_{m-1} & \cdots & a_3 & a_2 \\
\vdots & & & & & \\
0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\
0 & 0 & 0 & \cdots & 0 & a_{m-1}
\end{bmatrix}
\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-1} \end{bmatrix}
$$

We denote the $2m - 1 \times m$ matrix in the above equation by M.

Lemma 1. For any field $GF(p^m)$, let $\alpha, \alpha^2, \cdots, \alpha^{m-1}$ be a polynomial basis and for each $i = 0, 1, \cdots, p^m - 1$, let $\alpha^i = c_{i,0} + c_{i,1}\alpha + \cdots c_{i,m-1}\alpha^{m-1}$. Let $a = a_0 + a_1\alpha + \cdots a_{m-1}\alpha^{m-1}$ and $b = b_0 + b_1\alpha + \cdots b_{m-1}\alpha^{m-1}$ be elements of $GF(p^m)$. Then the vector of coefficients of the product of a and b is given by Zb where

$$
Z =
\begin{bmatrix}
M_0 + c_{m,0}M_m + c_{m+1,0}M_{m+1} + \cdots + c_{2m-2,0}M_{2m-2} \\
M_1 + c_{m,1}M_m + c_{m+1,1}M_{m+1} + \cdots + c_{2m-2,1}M_{2m-2} \\
\vdots \\
M_{m-1} + c_{m.m-1}M_m + c_{m+1,m-1}M_{m+1} + \cdots + c_{2m-2,m-1}M_{2m-2}
\end{bmatrix}
$$

Proof.

$$a \cdot b = d_0 + d_1\alpha + d_2\alpha^2 + \cdots d_{2m-2}\alpha^{2m-2}$$

$$= M_0 \cdot b + M_1 \cdot b\alpha + M_2 \cdot b\alpha^2 + \cdots + M_{m-1} \cdot b\alpha^{m-1}$$

$$+ M_m \cdot b(c_{m,0} + c_{m,1}\alpha + c_{m,2}\alpha^2 + \cdots + c_{m,m-1}\alpha^{m-1})$$

$$+ M_{m+1} \cdot b(c_{m+1,0} + c_{m+1,1}\alpha + c_{m+1,2}\alpha^2 + \cdots + c_{m+1,m-1}\alpha^{m-1})$$

$$+ M_{m+2} \cdot b(c_{m+2,0} + c_{m+2,1}\alpha c_{m+2,2}\alpha^2 + \cdots + c_{m+2,m-1}\alpha^{m-1})$$

$$+ \cdots + M_{2m-2} \cdot b(c_{2m-2,0} + c_{2m-2,1}\alpha + c_{2m-2,2}\alpha^2 + \cdots + c_{2m-2,m-1}\alpha^{m-1}) = (M_0 \cdot$$

$$b + M_m \cdot bc_{m,0} + M_{m+1} \cdot bc_{m+1,0} + \cdots + M_{2m-2} \cdot bc_{2m-2,0})$$

$$+ (M_1 \cdot b + M_m \cdot bc_{m,1} + M_{m+1} \cdot bc_{m+1,1} + \cdots + M_{2m-2} \cdot bc_{2m-1,1})\alpha$$

$$+ \cdots + (M_{m-1} \cdot b + M_m \cdot bc_{m,m-1} + + \cdots + M_{m_2m-2} \cdot bc_{2m-2,m-1})\alpha^{m-1} \text{ and so the}$$

vector of coefficients of $a \cdot b$ can be expressed as

$$Z \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix}$$

6.3   A Toeplitz variant of the Mastrovito matrix.

We call the matrix Z defined in Lemma 1, the *Mastrovito* matrix of the multiplication

by a.

The matrix Z depends on the irreducible polynomial that defines $GF(p^m)$ as well

as the rows $M_i, m \leq i \leq 2m - 2$. In the rest of this paper we assume that $GF(p^m)$ is

generated by a polynomial of the form $x^m - x^n - 1$. We shall see that in this case, Z has

useful symmetries which for certain fields can be exploited to reduce the complexity

of multiplication in $GF(p^m)$. For this, we make use of the "reduction matrix" defined

in [131] for the case $p = 2$, but which holds for arbitrary p as well.

The *reduction matrix* is produced by reducing higher ordered elements $\alpha^m, \alpha^{m+1}, \cdots, \alpha^{2m-2}$

modulo $\alpha^m - \alpha^n - 1$, where α is a root of the irreducible polynomial $x^m - x^n - 1$ over

$GF(p)$ that generates the field $GF(p^m)$. It is useful to partition the powers of α in

blocks of length $m - n$. There are $q + 1$ blocks where q and r are the unique integers

for which $m - 2 = (m - n)q + r$, $0 \leq r < m - n$. We have

$\alpha^m = 1 + \alpha^n$

$\alpha^{m+1} = \alpha^{n+1} + \alpha$

$\vdots$

$\alpha^{2m-n-1} = \alpha^{m-n-1} + \alpha^{m-1}$

$\alpha^{2m-n} = \alpha^{m-n} + \alpha^m$

$\qquad = \alpha^{m-n} + 1 + \alpha^n$

$\vdots$

$\alpha^{3m-2n-1} = \alpha^{2(m-n)-1} + \alpha^{(m-n)-1} + \alpha^{m-1}$

$\vdots$

$\alpha^{(q+1)m-qn} = \alpha^{q(m-n)} + \alpha^{(q-1)(m-n)} + \cdots + \alpha^{(m-n)} + \alpha^m$

$\qquad = \alpha^{q(m-n)} + \alpha^{(q-1)(m-n)} + \cdots + \alpha^{m-n} + 1 + \alpha^n$

$\vdots$

$\alpha^{(q+1)m-qn+r} = \alpha^{q(m-n)+r} + \alpha^{(q-1)(m-n)+r} + \cdots + \alpha^{m-n+r} + \alpha^r + \alpha^{n+r}$

The reduction matrix tells us how to compute the Mostrovito matrix for the special

case where $GF(p^m)$ is defined by $x^m - x^n - 1$. The $i - th$ equation is given by

$$\alpha^{m+i} = \alpha^{(m-n)q_i+m+r_i}$$

$$= \alpha^{(q_i+1)m-nq_i+r_i}$$

$$= \alpha^{n+r_i} + \alpha^{q_i(m-n)+r_i} + \alpha^{(q_i-1)(m-n)+r_i} + \cdots + \alpha^{(m-n)+r_i} + \alpha^{r_i}$$

where q_i and r_i are the unique integers for which $i = (m-n)q_i + r_i$, $0 \leq r_i < m - n$.

Thus, starting with

$$A = \begin{bmatrix} a_0 & 0 & 0 & 0 & 0 & 0 \\ a_1 & a_0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & \\ a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 \end{bmatrix},$$

the previous equation tells us that M_{m+i} must be added to rows $n + r_i$, $q_i(m-n) + r_i$, $(q_i - 1)(m-n) + r_i, \cdots, (m-n) + r_i, r_i$ of A.

Let us write

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

where U is the $n \times m$ matrix consisting of rows $0, 1, \cdots, n - 1$ of Z and L is the $m - n \times m$ matrix consisting of rows $n, n + 1, \cdots, m - 1$ of Z. We define the *modified Mostrovito matrix* by

$$Z'_{a,p^m} = Z' = \begin{bmatrix} L \\ U \end{bmatrix}$$

Lemma 2. If $GF(p^m)$ is defined by a trinomial of the form $x^m - x^n - 1$, then Z'_{a,p^m} is Toeplitz.

Proof.

For each $i = 0, \cdots q$, define T_i to be the $m \times m$ matrix containing all zeros, except in the upper triangle which contains rows $(i+1)m - in$ through $2m - 2$ of M and let B_i be the $m - n \times m$ matrix consisting of the $m - n$ rows of M starting with row $(i+1)m - in$. Then Z can be formed by adding $X = B_0 + B_1 + \cdots + B_q$ to rows n through $m - 1$ of $Y = A + T_0 + T_1 + \cdots + T_q$. Now the sum of Toeplitz matrices is also Toeplitz and so both X and Y are Toeplitz and hence U and L are Toeplitz. In order to show that Z' is Toeplitz it remains only to show that L_{m-n-1}, i.e., the last row of L, shifted one position to the right differs from U_0, i.e., the first row of U in only the first position. We have

$$L_{m-n-1} = M_{m-1} + M_{2m-n-1} + M_{3m-2n-1} + \cdots + M_{(q+1)m-qn-1}$$

and

$$U_0 = M_0 + M_m + M_{2m-n} + M_{3m-2n} + \cdots + M_{(q+1)-qn}$$

Furthermore, $M_{(i+1)m-in-1}[\rightarrow 1] = M_{(i+1)m-in}$ for each $i = 0, 1, \cdots, q$. Furthermore, $M_0 = [a_0, 0, \cdots, 0]$ and so $L_{m-n-1}[\rightarrow 1] = U_0 + [a_0, 0, \cdots, 0]$ and hence Z' is Toeplitz.

6.4 Multiplication and Number-Theoretic Transforms

Multiplication in $GF(p^m)$ is simply polynomial multiplication modulo an irreducible polynomial over $GF(p)$. One of the ways to speed up multiplication in fields of characteristic zero is by use of the "Convolution Theorem." It is thus natural to ask if these same ideas can be applied to multiplication in a finite fields.

The *cyclic convolution* $a \otimes b$ of two vectors $a = [a_0, a_1, \cdots, a_{m-1}]$ and $b =$

$[b_0, b_1, \cdots, b_{m-1}]$ over a ring is a vector $c = [c_0, c_1, \cdots, c_{m-1}]$ where $c_i = \sum_{j=0}^{m-1} a_j b_{i-j}$

for each $i = 0, 1, \cdots, m-1$ and where $b_j = b_{m-j}$ for $j = 0, 1, \cdots, m-1$. The convo-

lution theorem for the complex case states that for vectors a and b over the fields C

of complex numbers, $a \otimes b = F^{-1}(F(a) \odot F(b))$ where F denotes the discrete Fourier

transform (DFT) and $\odot$ denotes the pointwise product of the vectors $F(a)$ and $F(b)$.

Using the convolution theorem and a fast Fourier transform (FFT) to compute F and

F^{-1}, the time to compute $a \otimes b$ can be reduced from $O(m^2)$ to $O(m log\ m)$.

The DFT for vectors of length m over C is defined in terms of a primitive $m - th$

root of unity. Such a number exists for any integer $m > 1$. However, there are

restrictions on the corresponding concept defined on finite fields (or rings). A DFT

over a finite field or finite ring, called a "number-theoretic transform" is defined, for

our purposes, as follows. Let Z_N be the ring of integers modulo an integer $N > 1$, let

$d > 1$ be an integer and let r be a primitive root of unity modulo N, i.e., $r^d = 1\ mod\ N$

and d is the least positive integer with this property. We define the *number-theoretic*

transform (NTT) of length d over Z_N to be the linear transformation $F_d = F : Z_N^d \to$

Z_n^d, represented by the matrix $F = [r^{ij}]$, $0 \leq i, j \leq d-1$. The *inverse number-theoretic*

transform of length d over Z_N is defined by $F^{-1} = d^{-1}[r^{-ij}]$, $0 \leq i, j \leq d - 1$, where

d^{-1} is the inverse of the field element $1 + 1 + \cdots 1$ (d times).

An NTT over Z_N of length d (and its inverse) exists if and only if there exists

a $d - th$ root of unity in Z_N. Furthermore, the convolution theorem holds in this

case. When N is prime, then Z_N is a field and $Z_N - \{0\}$ is a cyclic group under

multiplication. Hence when N is prime, an NTT of length d exists if and only if d

divides $N - 1$.

We have shown that the modified Mastrovito matrix is Toeplitz. A special type of Toeplitz matrix that is of interest are the "circulant" matrices. An $n \times n$ matrix is said to be *(right) circulant* if every row after the first can be obtained from the previous row by a right circular shift.

Lemma 3 Let A be an $m \times m$ Toeplitz matrix and let $r \geq 2m - 1$. Then A can be embedded in an $r \times r$ circulant matrix.

Proof. Let $[a_1 \; a_2 \; \cdots \; a_m]$ and $[a_1, b_1, b_2, \cdots, b_{m-1}]$ be the first row and the first column, respectively, of an $m \times m$ Toeplitz matrix and let C be the circulant matrix whose first row is $[a_1 \; a_2 \; \cdots \; a_m \; 0 \; \cdots \; 0 \; b_{m-1} \; b_{m-2} \; \cdots \; b_2 \; b_1]$, where there are $r + 1 - 2m$ zeros.

Lemma 4 Let C be a circulant $n \times n$ matrix and let x be a vector of length n. Then

$$Cx = c \otimes x$$

where c is the first column of C.

Proof. The proof follows from the definition of cyclic convolution.

Lemma 5 Let $[a_0, a_1, \cdots, a_{m-1}]$ and $[b_0, b_1, \cdots, b_{m-1}]$ be the vectors of coeficients of elements a and b, respectively, in $GF(p^m)$. Suppose $x^m - x^n - 1$ is irreducible over $GF(p)$ and let

$$a' = [a_n, a_{n+1}, \cdots, a_{m-1}, a_0, a_1, \cdots, a_{n-1}, 0, \cdots, 0, s_1, s_2, \cdots, s_{m-1}]$$

be a vector of length $r \geq 2m - 1$, where

$$
\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{m-1} \end{pmatrix} = \begin{pmatrix} f_{m-1} \\ f_{m-2} \\ \vdots \\ f_1 \end{pmatrix} + \begin{pmatrix} c_{m,n} & c_{m+1,n} & \cdots & c_{2m-2,n} \\ 0 & c_{m,n} & \cdots & c_{2m-3,n} \\ \vdots & & & \\ 0 & 0 & \cdots & c_{m,n} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{m-1} \end{pmatrix}
$$

where each f_j is a_{n-j} if $j \leq n$ and is zero otherwise. Let $[d_0, d_1, \cdots, d_{m-1}]$ be the vector of coefficients of the product of a and b in $GF(p^m)$ and let b be the vector of length r obtained by padding b with $r - m$ zeros. Then $[d_n, d_{n+1}, \cdots, d_{m-1}, d_0, d_1, \cdots, d_{n-1}]$ consists of the first m components of $a' \otimes b'$.

Proof. By Lemma 2, the modified Mastrovito matrix Z' is Toeplitz. By Lemma 3, Z' can be embedded in an $r \times r$ circulant matrix $\bar{Z}'$ and so the product ab can be obtained from the matrix vector product $\bar{Z}'b'$ where b' is a vector of length r by padding b with an appropriate number of zeros. Thus by Lemma 4, ab can be obtained by computing the convolution $a' \otimes b'$, where a' is the first column of $\bar{Z}'$. By the proof of Lemma 3, a' consists of the first column of Z' followed by an appropriate number of zeros, followed by last $m - 1$ elements written in reverse order of the first row of Z'. But the first row of Z' is the n-th row of Z, which is $[s_{m-1}, s_{m-2}, \cdots, s_1]$ where the s_i are given as above.

The proof of Lemma 5 gives us a method for computing products in terms of convolution in fields $GF(p^m)$ that can be defined by irreducible trinomials of the form $x^n - x^n - 1$. When the number-theoretic transform exists, this means that we can asymptocally reduce the complexity of multiplication by applying the Convolution

Theorem and using a fast method for computing the NTT. Pollard [109] has shown

that an NTT of length N over $GF(p)$ can be computed with $O(N(N_1 + N_2 + \cdots N_k)$

mod p operations where $N = N_1 N_2 \cdots N_k$. Such an NTT is efficient when N is highly

composite. A case of particular interest is given by the following

Theorem. Let p be a prime of the form $t2^k + 1$ where t is odd and let m be such that

$2^k \geq 2m - 1$. If $x^m - x^n - 1$ is irreducible over $GF(p)$ for some n, then multiplication

in $GF(p^m)$ can be performed with $O(m \log m)$ *mod* p operations.

Proof.

Since $p = t2^k + 1$, we have $p - 1 = t2^k$ and so a NTT of length 2^s exists for any

$s \leq k$. So we can submerge the modified Mastrovito matrix in a circulant matrix

of size 2^s where 2^s is the smallest power of 2 which is greater than or equal to

$2m - 1$. Then we compute the first column a' and use a fast algorithm to compute

$a' \otimes b' = F_1(F(a') \odot F(b'))$ where b' is b padded with $2^s - m + 1$ zeros and then read

off the coefficients of the product ab. However, the computation of a' requires the

computation of the matrix-vector product

$$
C \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{m-1} \end{pmatrix}
$$

where

$$
C = \begin{pmatrix}
c_{m,n} & c_{m+1,n} & \cdots & c_{2m-2,n} \\
0 & c_{m,n} & \cdots & c_{2m-3,n} \\
\vdots & & & \\
0 & 0 & \cdots & c_{m,n}
\end{pmatrix}
$$

But since C is also Toeplitz, this matrix-vector product can also be computed by the same method of submerging C in a $2^s \times 2^s$ circulant matrix and computing the convolution of the first column of the circulant with a padded form of the vector. The complexity of each of the two convolutions is $O(m \ log \ m)$ and thus multiplication can be performed in time $O(m \ log \ m)$.

Example. Let us consider multiplication in $GF(257^6)$. The above theorem applies since $257 = 2^8 + 1$ and the trinomial $x^6 - x - 1$ is irreducible over $GF(257)$. Let $[a_0, a_1, a_2, a_3, a_4, a_5]$ and $[b_0, b_1, b_2, b_3, b_4, b_5]$ be the vectors of coefficients of two elements a and b of $GF(257^6)$ and let $[c_0, c_1, c_2, c_3, c_4, c_5]$ be the vector of coefficients of the product. Then $[c_1, c_2, c_3, c_4, c_5, c_0]$ consists of the first six components of

$$
F^{-1}(F(\bar{Z}^{(1)}) \odot F(\bar{b}))
$$

where $\bar{Z}^{(1)}$ is the column vector $[a_1, a_2, a_3, a_4, a_5, a_0, 0, 0, 0, 0, 0, a_1 + a_2, a_2 + a_3, a_3 + a_4, a_4 + a_5, a_0 + a_5]$ and $\bar{b} = [b_0, b_1, b_2, b_3, b_4, b_5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ and where F denotes an NTT of length 16.

It is natural to ask how abundant are the cases that satisfy the hypotheses of the above theorem.

There are an infinite number of primes of the given form. Indeed, it follows from Dirchlet's theorem that for any fixed k, there are infinitely many t for which $t2^k + 1$ is prime. When $t = 1$, such a prime is a *Fermat prime* and in this case, it is known that k must be a power of 2. Thus, every Fermat prime is of the form $F_n = 2^{2^n} + 1$. There are only five known Fermat primes, namely, $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, and $F_4 = 65537$. There has been considerable effort expended in the search for additional Fermat primes. In this regard, it is known that the only prime factors of a number of the form $2^m + 1$ are of the form $t2^k + 1$. In 1958 Robinson [**?**] published a table of primes of the form $t2^k + 1$ and there have been a number of publications since then that are dedicated to extending Robinson's table.

The second hypothesis concerning the existence of an irreducible trinomial $x^m - x^n - 1$ over $GF(p)$ is more elusive. Determining the irreducibility of polynomials modulo a prime is a computationally intensive problem. For example, a program we wrote in Mathematica took over two weeks to determine those values of m. $2 \leq m \leq 525$, for which there exists a trinomial $x^m - x^n - 1$ that is irreducible over $GF(p)$ where $p = 12289 = 3 \cdot 2^{12} + 1$. The number of such cases was 218.

Chapter 7

Ethical Issues

> The intrusion of computers into molecular biology shifted power into the hands of those with mathematical aptitudes and the computer savvy.
>
> Michio Kaku

Ethics is a major branch of philosophy. It involves systematizing, defending, and recommending concepts of right and wrong behavior. It is significantly broader than the common conception of analyzing right and wrong, the ethical theories have been broadly studied by philosophers and, more recently, it have gained attention in different subject areas related to the sciences. In consequence, the *applied ethics* has emerged as an important branch in an attempt to apply 'theoretical' ethics to real world dilemmas in sciences.

Some controversial issues concerning the present research fall within two important applied ethical topics: *computer ethics* and *bioethics*. Aside from the ethical aspects considered and strictly held for developing this research in terms of proper scientific work, knowledge production practices, and the scientific integrity, we will examine some specific and controversial ethical issues related to the main themes involved in the present work. In the following section we will analyze the ethical issues

implied in the application problem (Reverse Engineering), the application field of this thesis (Computational Biology), and the computing paradigm used (Reconfigurable Computing).

7.1   Ethics and Reverse Engineering

As stated in Definition 8, reverse engineering is a general process for analyzing a technology specifically to ascertain how it was designed. Reverse engineering has its origins in the analysis of hardware for commercial or military advantage [19]. The purpose is to deduce design decisions from end products with little or no additional knowledge about the procedures involved in the original production.

Reverse engineering has been held as a legitimate form of discovery, but its correct use can engender suspects. Using reverse engineering involves some ethical issues in different applications.

Reverse engineering is applied for disassembly or decompilation of computer programs by reading the object or bin code of the program and translating them into source code. By analyzing input and output data, reverse engineering can determine some structures of the program and identify how it operates for presenting the information in an understandable computer language. This practice could be considered as the robbery of intellectual property. Dishonest competitors can steal a software design using testing and analysis. Once the competitor obtains the design details, they can improve upon the original product with minimal research and development costs.

Some basic ideas about reverse engineering used in this research have been widely

used for developing encoders and decoders for cryptosystems. It is possible to reverse-engineer an encryption system by a simple input-output scan attack, so secure data can be cracked.

Design of circuits in a semiconductor chip can be determined by reverse engineering. These chips are hardwired with circuits described by simple combinatorial logic functions. In a logic function, a given input will always produce a predictable output. An input-output analysis clearly allows one to reproduce the logical function that describes the circuit. By a simple input-output scan attack, it is possible to reverse engineer a design using a large number of possible inputs, and monitoring outputs to determine the internal logic functions of the semiconductor chip.

The security and integrity of protected data, and the intellectual property in hardware or software design can become vulnerable by using reverse engineering. As a consequence, researchers and developers of reverse engineering technologies must be seriously concerned about the ethical responsibility in their works.

7.2 Reconfigurable Computing and Ethics

The use of reconfigurable technologies implies some ethical issues that have to be considered by computing professionals. This technology could be used for dishonest purposes. For instance, FPGAs have been misused for cracking private data. On the other hand, this technology could be exploited for a better quality of life through integrity methods. In this sense FPGAs can be put to good use for green conservationism by substantially lowering power consumption of a supercomputing platform.

The potentialities of FPGA for accelerate reverse engineering on cryptosystem

can be used for hacking secure or confidential codes, such as passwords or banking information. Even when exhaustive key search is used, the advantage of FPGA performance can be exploited for cracking secure codes. For example, in [24] the authors designed an simple FPGA program in order to attack the IBM 4758 CCA, used in retail banking to protect the ATM infrastructure, they developed a practical scheme that extracted Data Encryption Standard keys from the system in a single 10-minute session. In [77], Kumar *et al.* use an array of low-cost FPGAs for performing an exhaustive key search of the Data Encryption Standard in less than nine days on average.

The conservation ethic is an ethic of resource use, allocation, exploitation, and protection. It is focused on maintaining the health of the natural world. The philosophy of this ethic proposes the conservation of materials and energy, as an imperative mean toward the protection of the natural world and its much needed and endangered resources.

Kris Gaj, a George Mason University researcher, headed an investigation which deserves mention because of its pertinence and relevance in how reconfigurable computing can work in favor of the ethics of conservation. Even though Gaj never mentions ethics, his work presents us with a determination to prove that, when put to good use, FPGAs performance can be useful for reducing excessive energy and resource consumption, versus the traditional high performance computers. His study [43] compares the consumption power of an SRC-6E reconfigurable computer with a clus-

ter of 100 processors 2.4 GHz Pentium IV [1], in a time span of five years [2]. The total power consumption of the reconfigurable computer over five years was 21900 kW-hour, while the cluster consumed 930833 kW-hour. Moreover, the total cost of power of the reconfigurable computer over the five year period is \$2,628, while the total cost of power of the cluster is \$111,700, which represents savings of \$109,072 [3].

7.3 Computational Biology and Ethics

A concern in bioethics has to do with misconduct in collecting and processing data and publishing results of scientific research. Bioethicists have identified as unethical practice to handle data with computer programs, and report it as experimental data. Reverse engineering of genetic regulatory networks could be used for this ethical transgression. So, from very few sampling data a genetic network can be inferred and more gene expression data can be obtained from the inferred network.

The source of sampling data for reverse engineering gene networks is also an important issue in bioethics. For example, when researchers deal with biological data, they have to consider carefully the property rights in genetic information [127].

Nonetheless, the intended application of this research is not developed or thought to be used as a threat; instead it pretends to be part of the cause for politically correct practices in molecular biology research. Thus, it contemplates taking part in a positive outcome, and more importantly, worthy benefits for the human race

---

[1]For a 100x acceleration it is assumed 100% cluster efficiency, i.e. the application can be perfectly parallelized across 100 microprocessors, scaling the application linearly.

[2]It is assumed both systems used non-stop over a five year period.

[3]Cost of power estimation of \$0.12 per kW-hour was calculated by the average commercial cost of power in Los Angeles, New York, San Francisco and Washington DC.

and biological species in general. For example, molecular biology experiments usually involve the sacrifice of animals, and typically many experiments and repetitions are required. In these cases the reverse engineering proposed in our research could help minimize considerably the number of experiments with animals.

The biological application considered in this work can be associated to the genetic manipulation controversy. The engineering of genetically modified organisms for cloning species or transgenic creation has been an important issue in bioethics. Even if these issues are concerned with biologists, the responsibility of computer scientists and engineers is significant in bioinformatics [136]. This matter is clearly described by Kaku in [68] (the quote is taken from [88]): " The intrusion of computers into molecular biology shifted power into the hands of those with mathematical aptitudes and the computer savvy".

Chapter 8

Conclusions and Future Work

8.1 Conclusions

We have introduced a novel approach for finite field multiplication over odd-prime extension fields. A fast and space-saving design for a finite field multiplication over $GF(2^m)$ was also introduced. This multiplier became essential for the design of an efficient architecture for finite field inversion toward the ultimate and more challenging problem of developing a new and efficient architecture for finite field interpolation.

The structure of our multiplication algorithm for $GF(2^m)$ allows us to enhance performance by exploiting Mastrovito matrix symmetries, while avoiding modular reductions in the multiplication process. The core operation in the algorithm loop is the inner product which is accelerated by bit-level parallelism implemented through a parallel FIR filter. This simple architecture uses a small amount of FPGA area, delivering a low area implementation which makes possible to shift more of the computational burden to FPGAs by embedding our multiplier into more complex tasks, such as inversion and interpolation.

A novel architecture for polynomial interpolation over fields $GF(2^m)$ was presented. The implementation results and the performance analysis show that FPGAs have the potential for accelerating some resource demanding problems, such as our intended application to reverse engineering large genetic networks. Significant speedups of more than a hundred times over a 2.2 GHz Opteron processor were achieved. In addition, our implementation has proved to use optimally the resources in the FPGA. In this sense, interpolation over larger fields can be tackled without sacrificing performance.

The symmetries in the Mastrovito matrix were also exploited in order to formulate a new approach for fast finite field multiplication over $GF(p^m)$, with $p \neq 2$. We showed that the computational complexity of multiplication can be reduced by using convolution properties of the Number Theoretic Transform. This approach promises efficient solutions for applications that use expensive arithmetic operation over $GF(p^m)$ with $p \neq 2$.

8.2 Future Work

Future work includes developing means for an efficient implementation of finite field multiplication via number theoretic transform for fields $GF(p^m)$ of odd prime characteristic. This multiplier could be used for the interpolation phase of reverse engineering genetic networks with p levels of gene expression.

The interpolation and other applications of intensive arithmetic over finite fields can be extended to larger problems beyond the capacity of a single FPGA. The

problem size can be extended by implemented solutions based on partitioning the

problem in pieces that can be processed via parallelism with FPGAs or by using

dynamically reconfigurable computing.

Bibliography

[1] G. Ahlquist, B. Nelson, and M. Rice, "Optimal Finite Field Multipliers for FP-GAs", Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications, Lecture Notes In Computer Science, volume 1673, pp. 51-60, Springer-Verlag, London, UK, 1999.

[2] T. Akutsu, S. Miyano, and S. Kuhara. "Identification of Genetic Networks from a Small Number of Gene Expression Patterns Under the Boolean Network Model", In Pacific Symposium on Biocomputing, Volume 4, pp. 17-28, 1999.

[3] T. Akutsu, S. Kuhara, O. Maruyama and S. Miyano, "Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model", Theoretical Computer Science, Volume 298, pp. 235-251, 2003.

[4] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", Readings in computer architecture (reprinted from Proceedings of AFIPS Conference pp. 483-485, 1967), pp. 79-81, Morgan Kaufmann Publishers, San Francisco, CA., USA 2000.

[5] C. Auliac, V. Frouin, X. Gidrol, and F. d'Alche-Buc, "Evolutionary approaches for the reverse-engineering of gene regulatory networks: a study on a biologically realistic dataset", Volume 9-91, Number 1, pp. 1-9, 2008.

[6] M. Bahramali, H. Shahhoseini, "The Best Irreducible Pentanomials For A Mastrovito GF Multiplier", Proceedings of IEEE International Conference on Computer Systems and Applications, pp. 493 - 499, 2006.

[7] S. Baktir and B. Sunar, "Achieving efficient polynomial multiplication in Fermat fields using the fast Fourier Transform", ACM Southeast Regional Conference Proceedings of the 44th annual Southeast regional conference, ACM Press, pp. 549-554, 2006.

[8] Z. Bar-Joseph, "Analyzing time series gene expression data", Bioinformatics, Volume 20, Number 16, pp. 2493-2503, November 2004.

[9] T. Bartee, D. Schneider, "Computation with finite fields", Information and Control, volume 6, pp. 79-98, Mar. 1963.

[10] E. Berlekamp, "Bit-serial Reed - Solomon encoders". IEEE Transactions on Information Theory, volume 28, number 6, pp. 869-874, 1982.

[11] C. Bobda, "Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications", Springer-Verlag, 2007.

[12] D. Bollman, O. Colon, and E. Orozco. "Fixed Points in Discrete Models for Regulatory Genetic Networks". EURASIP Journal on Bioinformatics and Systems Biology, Issue 2007, number 1, pp. 18, August 2007.

[13] D. Bollman, and E. Ferrer, "Finite Field Multiplication via Number-Theoretic Transforms", to be submitted, 2008.

[14] 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.

[15] D. Bollman, and E. Orozco, "Finite Fields Models for Genetic Networks", unpublished manuscript, 2005.

[16] D.Buell, T. El-Ghazawi, K. Gaj, V. Kindratenko, "Guest Editors' Introduction: High-Performance Reconfigurable Computing", IEEE Computer, volume 40, number 3, pp. 23-27, March 2007.

[17] T. Chen, H. He, and G. Church, Modeling gene expression with differential equations, Proceedings of the Pacific Symposium on Biocomputing (Singapore) (R. Altman, A. Dunker, L. Hunter, and T. Klein, eds.), Volume 4, World Scientific Press, pp. 29-40, 1999.

[18] Lo Sing Cheng, Ali Min, and Tet Hin Yeap, "Efficient FPGA Implementation of FFT Based Multipliers", in Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Saskatoon, Canada, pp. 1300-1303, May 2005.

[19] E. Chikofsky, H. Cross, "Reverse engineering and design recovery: a taxonomy", IEEE software, volume 7, number 1, pp. 13-17, 1990.

[20] K. Cho, S. Choo, S. Jung, J. Kim, H. Choi, and J. Kim, "Reverse engineering of gene regulatory networks", IET Systems Biology, Volume 1, Issue 3O, pp 149-163, May 2007.

[21] O. Colón-Reyes, R. Laubenbacher and B. Pareigis, "Boolean Monomial Dynamical Systems", Annals of Combinatorics, Volume 8, Number 4, pp. 425-439, January 2005.

[22] O. Colón-Reyes, A. S. Jarrah, R. Laubenbacher, and B. Sturmfels, "Monomial dynamical systems over finite fields", Journal of Complex Systems, Volume 16, Number 4, pp. 333-342, 2006.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "Introduction to Algorithms", MIT Press and McGraw-Hill, 2001.

[24] R. Clayton, and M. Bond, "Experience Using a Low-Cost FPGA Design to Crack DES Keys", 4th International Workshop on Cryptographic Hardware and Embedded Systems, Lecture Notes In Computer Science, Volume 2523, pp. 579 - 592, 2002.

[25] Cray Inc., "Cray XD1 System Overview", Release 1.2, 2005.

[26] Cray Inc., "Cray XD1 programming", Release 1.2.1, S-2433-121, 2005.

[27] Cray Inc., "Cray XD1 FPGA development", Release 1.2, S-6400-12, 2005.

[28] T. Damarla and M. Karpovsky, "Fault Detection in Combinational Networks by Reed-MullerTransforms", IEEE Transactions on Computers, Volume 38, Number 6, pp. 788-797, June 1989.

[29] A. Daneshbeh, and M. Hasan,"A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over $GF(2^m)$", IEEE Transactions on Computers, Volume 54, Issue 3, pp. 370-380, 2005.

[30] A. Datta, R. Pal, A. Choudhary, E. R. Dougherty, "Control Approaches for Probabilistic Gene Regulatory Networks", IEEE Signal Processing Magazine, Vol. 24, No. 1, pp. 54-63, 2007.

[31] J-P. Deschamps, and G. Sutter, "Hardware Implementation of Finite-Field Division", Acta Applicandae Mathematicae, Volume 93, Numbers 1-3, pp. 119-147, 2006.

[32] T. El-Ghazawi, "Is High-Performance, Reconfigurable Computing the Next Supercomputing Paradigm?", Proceedings of the ACM/IEEE SC 2006 Conference, pp. xv - xv, Nov. 2006.

[33] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, D. Buell, "The Promise of High-Performance Reconfigurable Computing", IEEE Computer, Volume 41, Number 2, pp. 69-76, February 2008.

[34] J. Fernando, D. Dalessandro, A. Devulapalli, and A. Krishnamurthy, "Enhancing FPGA Based Encryption", Ninth Workshop on High Performance Embedded Computing (HPEC). Sept. 2005.

[35] J. Fernando, D. Dalessandro, A. Devulapalli, K. Wohlever, "Accelerated FPGA Based Encryption", The 2005 Cray Users Group Conference, Albuquerque, NM, May 2005.

[36] E. Ferrer, and E. Orozco, "Fast Arithmetic in Large Finite Fields", Seminario Interuniversitario de Investigación en Ciencias Matemáticas (SIDIM-XX), Mayagüez, PR, Feb. 2005.

[37] E. Ferrer, D. Bollman, and O. Moreno, "Efficient Finite Field Arithmetic for Field Programmable Gate Arrays (FPGAs)", Richard Tapia Celebration of Diversity in Computing Conference, Albuquerque, NM, October 2005.

[38] E. Ferrer, D. Bollman, and O. Moreno, "Toward a Solution of the Reverse Engineering Problem Using FPGAs", in Lehner et al. (Eds.): Lecture Notes in Computer Sciences, Proceedings of Euro-Par 2006 Workshops, Dresden, Germany, Volume 4375, pp. 301-309, September 2006.

[39] E. Ferrer, D. Bollman, and O. Moreno. "A fast finite multiplier". In C. Diniz et al. (Eds.): Lecture Notes in Computer Sciences, Proc. International Workshop on Applied Reconfigurable Computing, Volume 4419, pp. 238246, March 2007.

[40] E. Ferrer, and D. Bollman, "A new efficient architecture for univariate polynomial interpolation over $GF(2^m)$", to appear in Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms: ERSA'2008.

[41] K. Fong, D. Hankerson, J. López, A. Menezes, "Field Inversion and Point Halving Revisited", IEEE Transactions on Computers, Volume 53, Number 8, pp. 1047-1059, 2004.

[42] N. Friedman, M. Linial, I. Nachman, and Dana Pe'er, "Using Bayesian Networks to Analyze Expression Data", Journal of Computational Biology, Volume 7, Number 3-4, pp. 601-620, Aug 2000.

[43] K. Gaj, "Performance of Reconfigurable Supercomputers", RSSI-2005 - Reconfigurable Systems Summer Institute, Urbana-Champaign, July 2005.

[44] 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.

[45] C. Lu, K. Garhiksto, and L. Buzio, "A serial/parallel semi-systolic multiplier for finite field $GF(2^m)$", in Proceedings of 17th IEEE International Symposium on Computer Arithmetic, 2005.

[46] M. Gheorghe and V. Mitrana, "A Formal Language-Based Approach in Biology", Comparative and Functional Genomics, Volume5 , Number 1, pp. 91-94, February 2004.

[47] D. Gollmann, "Equally Spaced Polynomials, Dual Bases, and Multiplication in F_{2^n}", IEEE Transactions on Computers, volume 51, number 5, pp. 588-591, 2002.

[48] M. Gokhale and P. S. Graham. "Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays", Springer-Verlag, 2005.

[49] 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, pp. 268-271, 2003.

[50] J. Guajardo, and C. Paar, "Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes", Designs, Codes and Cryptography, Volume 25, Issue 2, pp. 207-216, 2002.

[51] P. D'Haeseleer, S. Liang and R. Somogyi, "Genetic Network Inference: From Co-Expression Clustering to Reverse Engineering", Bioinformatics, Volume 16, Number 8, pp. 707-726, 2000.

[52] A. Halbutogullari, Ç. K. Koç, "Mastrovito multiplier for general irreducible polynomials", IEEE Transactions on Computers, Volume 49, Issue 5, pp. 503-518, May 2000.

[53] A. Hartemink, D. Gifford, T. Jaakkola, and R. Young, "Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks", Proceedings of the Pacific Symposium on Biocomputing (Singapore) (R. Altman, A. Dunker, L. Hunter, and T. Klein, eds.), Volume 6, World Scientific Press, pp. 422-433, 2001.

[54] M. A. Hasan, M. Z. Wang, and V. K. Bhargava, "A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields", IEEE Transactions on Computers,volume 42, number 10, pp.1278-1280, October 1993.

[55] M. A. Hasan: Look-Up Table-Based Large Finite Field Multiplication in Memory Constrained Cryptosystems. IEEE Trans. Computers 49(7): 749-758 (2000)

[56] M.C., Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, D. DiSabello, "Achieving High Performance with FPGA-Based Computing", IEEE Computer, volume 40, number 3, pp. 50-57, March 2007.

[57] J. Herrero, A. Valencia, and J. Dopazo, "A hierarchical unsupervised growing neural network for clustering gene expression patterns", Bioinformatics, Volume 17, Number 2, pp. 126-136, 2001.

[58] I. S. Hsu, I. S. Reed, T. K. Truong, K. Wang, C. S Yeh, and L. J. Deutsch, "The VLSI Implementation of a Reed-Solomon Encoder Using Berlekamp's Bit-Serial Multiplier Algorithm", IEEE Transactions on Computers, volume 33, number 10, pp. 906-911, 1984.

[59] I. S. Hsu, T. K. Truong, L. J. Deutsch, I. S. Reed, "A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or Standard Bases", IEEE Transactions Computers, volume 37, number 6, pp. 735-739, 1988.

[60] J. Huang, H. Shimizu, and S. Shioya, "Clustering gene expression pattern and extracting relationship in gene network based on artificial neural networks", Journal of Bioscience and Bioengineering, Volume 96, Number 5, pp. 421-428, 2003.

[61] K. Huber, "Some Comments on Zech's Logarithms", IEEE Transactions on Information Theory, Volume 36, Issue 4, pp. 946-950, July 1990.

[62] H. Iba, and A. Mimura, "Inference of a gene regulatory network by means of interactive evolutionary computing", Information Sciences, Volume 145, Issue 3-4, pp. 225236, 2002.

[63] T. Ideker, V. Thorsson, and R. Karp, "Discovery of Regulatory Interactions Through Perturbation: Inference and Experimental Design", In Pacific Symposium on Biocomputing volume 5, pp. 302-313, 2000.

[64] T. Ideker, V. Thorsson, J. Ranish, R. Christmas, J. Buhler, J. Eng , R. Bumgarner, D. Goodlett, R. Aebersold, and L. Hood, "Integrated genomic and proteomic analyses of a systematically perturbed metabolic network", Science, Volume 292, pp. 929934, 2001.

[65] T. Itoh, and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases", Information and Computation, Volume 78, Issue 3, pp. 171-177, 1988.

[66] H. de Jong, "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review", Journal of Computational Biology, Volume 9, Number 1, pp. 67-103, 2002.

[67] H. de Jong, "Modeling and Simulation of Genetic Regulatory Networks", Lecture Notes in Control and Information Sciences, Volume 294, pp. 111-118, Sep. 2004.

[68] M. Kaku, "Visions: How Science Will Revolutionize the 21st Century", Oxford University Press, Oxford, 1998.

[69] N. Kasabovb, "Knowledge-based neural networks for gene expression data analysis, modelling and profile discovery", Drug Discovery Today: BIOSILICO, Volume 2, Issue 6, pp. 253-261, November 2004.

[70] A. Katok and B. Hasselblatt, "Introduction to the Modern Theory of Dynamical Systems", Cambridge University Press, 1995.

[71] S. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets". Journal of Theoretical Biology, Number 22, pp. 437- 467, 1969.

[72] S. Kauffman, "Homeostasis and differentiation in random genetic control networks". Nature, Number 224, pp. 177-178, 1969.

[73] C. H. Kim, S. Kwon, C. P. Hong, and I. G. Nam, "Efficient bit-serial systolic array for division over $GF(2^m)$ (elliptic curve cryptosystem applications)", Proceedings of the 2003 International Symposium on Circuits and Systems, Volume 2, pp. II-252 - II-255, 2003.

[74] C. H. Kim, S. Kwon, J. J. Kim, C. P. Hong, "A Compact and Fast Division Architecture for a Finite Field $GF(2^m)$", Lecture Notes in Computer Science, Issue 2667, pp. 855-864, Springer-Verlag, 2003.

[75] 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.

[76] C. Koç, T. Acar, "Montgomery Multiplication in $GF(2^k)$", Designs, Codes and Cryptography, Volume 14, Issue 1, pp. 57-69, 1998.

103

[77] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer and M. Schimmler, "Breaking Ciphers with COPACOBANA A Cost-Optimized Parallel Code Breaker", Lecture Notes in Computer Science, Cryptographic Hardware and Embedded Systems, Volume 4249, pp. 101-118, 2006.

[78] H. L ahdesm aki, I. Shmulevich, and O. Yli-Harja, "On Learning Gene Regulatory Networks Under the Boolean Network Model", Machine Learning, Volume 52 , Issue 1-2, pp. 147-167, 2003.

[79] E.C. Laskari, G.C. Meletiou, and M.N. Vrahatis, "Aitken and Neville Inverse Interpolation Methods over Finite Fields", Applied Numerical Analysis and Computational Mathematics, Volume 2, Issue 1, pp. 100-107, 2005.

[80] R. Laubenbacher and B. Pareigis, "Equivalence Relations on Finite Dynamical Systems", Advances in Applied Mathematics, Volume 26, Number 3, pp. 237-251, 2001.

[81] R. Laubenbacher, and B. Stigler, "A computational algebra approach to the reverse engineering of gene regulatory networks", Journal of Theoretical Biology, Volume 229, Issue 4, pp. 523-537, 2004.

[82] C-Y. Lee, "Low-complexity bit-parallel systolic multipliers over $GF(2^m)$", Integration, the VLSI Journal, Volume 41 , Issue 1, pp. 106-112, 2008.

[83] P. Li, C. Zhang, E. J. Perkins, P. Gong, Y. Deng, "Comparison of probabilistic Boolean network and dynamic Bayesian network approaches for inferring gene regulatory networks", BMC Bioinformatics, 8(Suppl 7):S13, 9 pages, 2007.

[84] S. Liang, S. Fuhrman, and R. Somogyi, "REVEAL, a general reverse engineering algorithm for inference of genetic network architectures", Proceedings of the Pacific Symposium on Biocomputing (Singapore) (R. Altman, A. Dunker, L. Hunter, and T. Klein,eds.), vol. 3, World Scientific Press, pp. 18-29, 1998.

[85] R. Lidl and H. Niedereitter, "Introduction to Finite Fields and Their Applications," Cambridge University Press, 2nd edition, 1994.

[86] J. Lipson, "Chinese remaindering and interpolation algorithms," Proc. 2nd Symposium in Symbolic and Algebraic Manipulation," pp. 372-391, 1971.

[87] S. Marshall, L. Yu, Y. Xiao, E. R. Dougherty, "Inference of a Probabilistic Boolean Network from a Single Observed Temporal Sequence", EURASIP Journal on Bioinformatics and Systems Biology., Vol. 2007, Article ID 32454, 15 pages, May, 2007

[88] A. Marturano, and R. Chadwich, "How the role of computing is driving new genetics' public policy". Ethics and Information Technology Journal, Springer Science, Kluwer Academic Publishers, volume 6, number 1, pp. 43-53, 2004.

[89] J.L. Massey and J.K. Omura. Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission. U.S Patent No: 4,567,600 Jan, 1986

[90] J.L. Massey, and J.K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic", US Patent No. 4,587,627, to OMNET Assoc., Sunnyvale CA, Washington, D.C.: Patent and Trademark Office, 1986.

[91] E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields", PhD thesis, Dept. of Electrical Eng., Linkvping Univ., Linkvping, Sweden, 1991.

[92] R. May "Uses and abuses of mathematics in biology", in Science, Number 303, pp. 790-793, 2004. Also published in International Journal of Humanities and Peace, Vol 21, Number 1, pp. 40-43, 2005.

[93] C. McIvor, M. McLoone, J.V McCanny,"FPGA Montgomery multiplier architectures - a comparison", Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 279-282, April 2004.

[94] U. Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays", Second Edition. Springer Verlag, Berlin, 2004

[95] Mentor Graphics®, "ModelSim SE Datasheet, Advanced Simulation and Debug", http://www.mentor.com/products/fv/digital-verification/modelsim-se/upload/se.pdf , Mentor Graphics, 2008.

[96] G. Meurice, and J-J Quisquater, "Iterative Modular Division over $GF(2^m)$: Novel Algorithm and Implementations on FPGA", Second International Workshop on Reconfigurable Computing: Architectures and Applications, Lecture Notes in Computer Science, Volume 3985, pp. 370-382, 2006.

[97] P. L. Montgomery, "Modular Multiplication without Trial Division", Mathematics of Computation, Volume 44, Number 170, pp. 519-521, 1985.

[98] 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, 2002.

[99] O. Moreno, H. Ortiz-Zuazaga, C. Corrada-Bravo, M. Aviñó-Diaz, and D. Bollman. "A finite field deterministic genetic network model", unpublished manuscript, 2005. http://www.hpcf.upr.edu/~humberto/papers/FFDeterGenNet.pdf

[100] G. H. Norton. "Extending the Binary GCD Algorithm", In Algebraic Algorithms and Error-Correcting Codes,pages 363372, Springer-Verlag, Berlin, 1986.

[101] OpenFPGA, "Reconfigurable Computing Terms on the OpenFPGA", https://isl.ncsa.uiuc.edu/twiki/bin/view/OpenFPGA/ReconfigurableComputingTerms

[102] H. Ortiz-Zuazaga, "Analysis of Gene Regulation Networks Using Finite-Field Models", CISE Ph.D. Thesis Proposal, University of Puerto Rico, Mayagüez, Puerto Rico, 2006.

[103] H. Ortiz-Zuazaga, M. Aviñó-Diaz, R. Laubenbacher, O. Moreno, "Finite fields are better Booleans", poster presented at the Seventh Annual Conferenc on Computational Molecular Biology (RECOMB 2003), Germany, April 2003.

[104] H. Ortiz-Zuazaga, S. Peña de Ortiz, O. Moreno. "Error Correction and Clustering Gene Expression Data Using Majority Logic Decoding". Proceedings of The 2007 International Conference on Bioinformatics and Computational Biology (BIOCOMP'07), June 2007.

[105] F. Parvaresh, A. Vardy, "Correcting Errors Beyond the Guruswami-Sudan Radius in Polynomial Time", Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS05), pp. 285-294, 2005.

[106] A. Paszkiewicz, "Some observations concerning irreducible trinomials and pentanomials over Z_2", Tatra Mountains Mathematical Publications. Volume 32, number. 3, pp. 119-127, 2005.

[107] D. Pe'er, A. Regev, G. Elidan, N. Friedman, "Inferring Subnetworks from Perturbed Expression Profiles", Bioinformatics, Volume 17, Suppl. 1, pp. S215S224, 2001.

[108] N. Petra, D. De Caro, A.G Strollo, "A Novel Architecture for Galois Fields $GF(2^m)$ Multipliers Based on Mastrovito Scheme", IEEE Transactions on Computers, Volume 56, Issue 11, pp. 1470-1483, Nov. 2007.

[109] J.M. Pollalrd, "The Fast Fourier Transform in a Finite Field", Mathematics of Computation, Volume 15, Number 114, pp. 365-374, 1971.

[110] D. Repsilber, H. Liljenstrom, and S. Andersson, "Reverse Engineering of Regulatory Networks: Simulation Studies on a Genetic Algorithm Approach for Ranking Hypotheses", Biosystems, Number 66, pp. 31-41, 2002.

[111] A. Reyhani-Masoleh, M. A. Hasan, "A New Construction of Massey-Omura Parallel Multiplier over GF(2^m)", IEEE Transactions on Computers, volume 51, number 5, pp. 511-520, 2002.

[112] A. Reyhani-Masoleh, M. Hasan, "Low Complexity Word-Level Sequential Normal Basis Multipliers", IEEE Transactions on Computers, volume 54, number 2, pp. 98-110, 2005.

[113] A. Reyhani-Masoleh, M.A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$ ", IEEE Transactions on Computers, volume 53, number 8, pp. 945-959, 2004.

[114] F. Rodríguez-Henríguez, and Ç. K. Koç, "Parallel multipliers based on special irreducible pentanomials", IEEE Transactions on Computers, volume 52, number 12, pp. 1535-1542, 2003.

[115] F. Rodríguez-Henríquez, G. Morales-Luna, N. Saqib, and N. Cruz-Cortés, "A Parallel Version of the Itoh-Tsujii Multiplicative Inversion Algorithm", In C. Diniz et al. (Eds.): Lecture Notes in Computer Sciences, Proc. International Workshop on Applied Reconfigurable Computing, ARC-2007, Volume 4419, pp. 238246, March 2007.

[116] F. Rodríguez-Henríquez, G. Morales-Luna, N. Saqib, and N. Cruz-Cortés, "Parallel Itoh-Tsujii multiplicative inversion algorithm for a special class of trinomials", Designs, Codes and Cryptography, Volume 45, Number 1, pp. 1937, October 2007.

[117] Rodríguez-Henríquez, N. A. Saqib, and N. Cruz-Cortés, "A fast implementation of multiplicative inversion over $GF(2^m)$", International Symposium on Information Technology (ITCC 2005), Volume 1, pp. 574579, Las Vegas, Nevada, U.S.A., April 2005.

[118] E. Savas and Ç. K. Koç, "Efficient methods for composite field arithmetic", Technical Report, Oregon State University, 1999.

[119] E. Savas, A. F. Tenca, M. E. Ciftcibasi, and C. K. Koç, "Novel multiplier architectures for $GF(p)$ and $GF(2^n)$", IEE Proceedings - Computers and Digital Techniques, Volume 151, Number 2, pp. 147-160, March 2004.

[120] R.E. Schapire and L.M. Sellie, "Learning Sparse Multivariate Polynomials over a Field with Queries and Counterexamples". Proceedings Symposium of Computational Learning Theory (COLT 93), pp. 17-26, May 1993.

[121] P. A. Scott, S. E. Tarvares, and L. E. Peppard, "A fast multiplier for $GF(2^m)$", IEEE Journal on Selected Areas in Communication, volume SAC-4, Jan. 1986.

[122] I. Shmulevich, E. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean Networks: A Rule-based Uncertainty Model for Gene Regulatory Networks", Bioinformatics, Volume 18, Number 2, pp. 261-274, 2002.

[123] C. Shu, S. Kwon and K. Gaj, "FPGA Accelerated Multipliers over Binary Composite Fields Constructed via Low Hamming Weight Irreducible Polynomials", IET Computers and Digital Techniques, Volume 2, number 1, pp. 6-11, 2008.

[124] B. Sokhansanj, J. Fitch, J. Quong, and A. Quong, "Linear fuzzy gene network models obtained from microarray data by exhaustive search", BMC Bioinformatics, Volume 5:108, pp. 1-12, 2004.

[125] L. Song, and K. Parhi, "Low-complexity modified Mastrovito multipliers over finite fields $GF(2^m)$", Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, Volume 1, pp. 508-512, 1999.

[126] C. Soule, "Mathematical approaches to differentiation and gene regulation", CR Biologies, Volume 329, pp. 1320, 2006.

[127] R. Spinello, "Property Rights in Genetic Information", Ethics and Information Technology Journal, Springer Science, Kluwer Academic Publishers, volume 6, number 1, pp. 29-42, 2004.

[128] H. Steck, and T. S. Jaakkola "Predictive Discretization During Model Selection", Lecture Notes In Computer Science, Volume 3175, pp. 1-8, Springer-Verlag, 2004.

[129] B. Stigler, "An Algebraic Approach to Reverse Engineering with an Application to Biochemical Networks". Virginia Tech Digital Library and Archives. PhD Dissertation. 2005.

[130] D. Strenski, "Computational Bottlenecks and Hardware Decisions for FPGAs". FPGA and Programmable Logic Journal, Nov 14, 2006.

[131] B. Sunar, E. Savas, and Ç. K. Koç, "Constructing Composite Field Representations for Efficient Conversion", IEEE Transactions on Computers, Volume 52, Number 11, pp. 1391-1398, 2003.

[132] B. Sunar, and Ç. K. Koç, "Mastrovito Multiplier for All Trinomials", IEEE Transactions on Computers, Volume 48, Number. 5, pp. 522-527, 1999.

[133] M. Swain, T. Hunniford, J. Mandel, N. Palfreyman, W. Dubitzky,"Modeling gene-regulatory networks using evolutionary algorithms and distributed computing", IEEE International Symposium on CCGrid 2005, Volume 1, Issue , 9-12 pp. 512 - 519, 2005.

[134] N. Takagi, J. Yoshiki, and K. Tagaki, "A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis", IEEE Transactions on Computers, Volume 50, Number 5, pp. 394398, May 2001.

[135] J. Tegnér, M. Yeung, J. Hasty, and J. Collins, "Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling", Proceedings of the National Academy of Science of the United States of America Volume 100, Number 10, pp. 5944-5949, 2003.

[136] P. B. Thompson, "Is the GMO controversy relevant to computer ethics?", Ubiquity Volume 9, Issue 2, January 2008.

[137] M. Wahde, and J. Hertz, "Coarse-grained Reverse Engineering of Genetic Regulatory Networks", Biosystems, Number 55, pp. 129136, 2000.

[138] Ch.C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in GF(2^m)", IEEE Transactions on Computers, Volume 34, number 8, pp. 709-717, 1985.

[139] C.-H. Wu, C.-M. Wu, M.-D. Shieh, and Y.-T. "Hwang, High-Speed, Low-Complexity Systolic Designs of Novel Iterative Division Algorithms in $GF(2^m)$", IEEE Transactions on Computers, Volume 53, Number 3, pp. 375-380, 2004.

[140] Xilinx Inc., "Xilinx ISE 9.1i Software Manuals and Help - PDF Collection", Xilinx, Inc., 2007.

[141] Z. Yan, D. Sarwate, and Z. Liu "High-speed Systolic Architectures for Finite Field Inversion", Integration, the VLSI Journal, Volume 38, Issue 3, pp. 382398, 2005.

[142] S. Yen, "Improved normal basis inversion in $GF(2^m)$", IEE Electronic Letters, Volume 33, Number 3, pp. 196197, January 1997.

[143] M. Yeung, J. Tegnér, and J. Collins, "Reverse Engineering Gene Networks Using Singular Value Decomposition and Robust Regression", Proceedings of the National Academy of Science of the United States of America Volume 99, number 9, pp. 6163-6168, 2002.

[144] S-Q. Zhang, W.-K. Ching, M. K. Ng , T. Akutsu, "Simulation study in Probabilistic Boolean Network models for genetic regulatory networks", International Journal on Data Mining and Bioinformatics, Vol. 1, No. 3, 2007.

[145] Z. Zilic and Z. Vranesic. "A deterministic multivariate interpolation algorithm for small finite fields", IEEE Transaction on Computer, Volume 51, Number 9, pp. 11001105, September 2002.

[146] M. Zou, and S. Conzen, "A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data", Bioinformatics, Volume 21, Number 1 pp. 71-79, 2005.