High speed world level finite field multipliers in F2m

Ashkan Hosseinzadeh Namin
University of Windsor

High Speed World Level Finite Field Multipliers in $\mathbb{F}_{2^m}$

by

Ashkan Hosseinzadeh Namin

A Thesis
Submitted to the Faculty of Graduate Studies through
the Department of Electrical and Computer Engineering in Partial Fulfillment
of the Requirements for the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada
2009

Your file Votre référence
ISBN: 978-0-494-57643-4
Our file Notre référence
ISBN: 978-0-494-57643-4

Declaration of Co-Authorship

I hereby declare that this thesis incorporates material that is the result of joint research as follows:

This thesis incorporates the outcome of joint research undertaken in collaboration with Karl Leboeuf under the supervision of Dr. Roberto Muscedere. The collaboration contributions are outlined in Chapter 8. The personal contributions, design work and development performed by the author are the focus of this chapter.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contributions of other researchers to my thesis, and have obtained written permission from the co-authors to include the above materials in my thesis.

I certify that with the above qualification, this thesis, and the research to which it refers is the product of my own work.

Abstract

Finite fields have important applications in number theory, algebraic geometry, Galois theory, cryptography, and coding theory. Recently, the use of finite field arithmetic in the area of cryptography has increasingly gained importance. Elliptic curve and El-Gamal cryptosystems are two important examples of public key cryptosystems widely used today based on finite field arithmetic. Research in this area is moving toward finding new architectures to implement the arithmetic operations more efficiently.

Two types of finite fields are commonly used in practice, prime field GF(p) and the binary extension field GF(2^m). The binary extension fields are attractive for high speed cryptography applications since they are suitable for hardware implementations. Hardware implementation of finite field multipliers can usually be categorized into three categories: bit-serial, bit-parallel, and word-level architectures. The word-level multipliers provide architectural flexibility and trade-off between the performance and limitations of VLSI implementation and I/O ports, thus it is of more practical significance.

In this work, different word level architectures for multiplication using binary field are proposed. It has been shown that the proposed architectures are more efficient compared to similar proposals considering area/delay complexities as a measure of performance. Practical size multipliers for cryptography applications have been realized in hardware using

FPGA or standard CMOS technology. to similar proposals considering area/delay complexities as a measure of performance. Practical size multipliers for cryptography applications have been realized in hardware using FPGA or standard CMOS technology. Also different VLSI implementations for multipliers were explored which resulted in more efficient implementations for some of the regular architectures. The new implementations use a simple module designed in domino logic as the main building block for the multiplier. Significant speed improvements was achieved designing practical size multipliers using the proposed methodology.

To my family, with love ...

Acknowledgments

There are several people who deserve my sincere thanks for their generous contributions to this project. I would first like to express my sincere gratitude and appreciation to Dr. Huapeng Wu and Dr. Majid Ahmadi, my supervisors for their invaluable guidance and constant support throughout the course of this thesis work.

In addition to my advisors, I would like to thank the rest of my thesis committee: Dr. Roberto Muscedere and Dr. Esam Abdel-Raheem from the electrical and computer engineering department, Dr. Arunita Jaekel from the school of computer science and Dr. Gerald E. Sobelman from the University of Minnesota for their participation in my seminars, reviewing my thesis, and their constructive comments.

I also like to thank Dr. Roberto Muscedere and Till Kuendiger for their assistants regarding the VLSI CAD tools and facilities used during the course of the project. I am thankful to my research colleagues in the RCIM lab of the University of Windsor and my friends whom have supported and believed in me: Golnar Khodabandehloo, Mitra Mirhassani, Mahzad Azarmehr, Andrew Tam, Kevin Biswas and Matthew Meloche. A Special thanks goes to Karl Leboeuf, for proofreading this thesis.

Finally, my deepest gratitude goes to my family for their unconditional love, support and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

AEDS	AND-Efficient Digit-Serial.
ASIC	Application-Specific Integrated Circuit.
BPWS	Bit-Parallel Word-Serial.
BSWP	Bit-Seriall Word-Parallel.
CAD	Computer Aided Design.
CMOS	Complementary Metal-Oxide-Semiconductor.
ECC	Elliptic Curve Cryptograhy.
ECDSS	Elliptic Curve Digital Signature Standard.
FPGA	Field-Programmable Gate Array.
GF	Galois Field.
IC	Integrated Circuit.
IEEE	Institute of Electrical and Electronics Engineers.
ISO	International Organization for Standardization.
LUT	Look-Up-Table.
NB	Normal Basis.
NIST	National Institute of Standards and Technology.
ONB	Optimal Normal Basis.
RB	Redundant Basis.
RNB	Reordered Normal Basis.
SMPO	Sequential Multiplier with Parallel Output.
VHDL	Very high speed integrated circuit (VHSIC) Hardware Description Language.
VLSI	Very-Large-Scale Integration.
XEDS	XOR-Efficient Digit-Serial.

Chapter 1

Introduction

Finite field is a set of finite elements where one can add, subtract, multiply, and divide such that properties of associativity, distributivity, and commutativity are satisfied [25]. Finite fields have important applications in error control coding and cryptography[29].

Two different types of finite field are commonly used in practice: prime field $\mathbb{F}_p$, and the binary field $\mathbb{F}_{2^m}$. Binary field is an extension of the prime field, $\mathbb{F}_2$, which contains 2^m elements. Binary fields are attractive for high speed cryptography applications since they are suitable for hardware implementation [18]. For applications to elliptic curve cryptography, binary field sizes are required to be at least 160 bits respectively [18].

In $\mathbb{F}_{2^m}$, addition is nothing but exclusive-oring of two binary vectors. Multiplication is more complicated, while division or inversion can be broken down into a series of consecutive multiplication operations [11],[41]. In practice, the finite field multiplier becomes the key arithmetic unit for any system based on finite field computations.

Efficiency of finite field multiplication depends on the choice of the basis to represent field elements. Bases that have been used for realizing finite field multipliers include polynomial basis, normal basis (NB), dual bases, triangular basis, redundant representation or redundant basis, and their variations (*i.e.,* shifted polynomial basis) [18, 33, 16, 10, 7, 38, 44, 17, 8].

In this work, we are mainly interested in normal basis and redundant representation, since squaring operation can be achieved by reordering the element coefficients which is free in hardware. Free squaring operation can be used to speed up the exponentiation operation by repeated squaring and multiplication [14].

In normal basis, the complexity of multiplication is measured with the multiplication matrix [30]. For a binary extension field, the multiplication matrix entries are either zero or one, and the number of ones inside the multiplication matrix is referred to as normal basis complexity. The normal basis in GF(2^m) for which the complexity achieves its minimum, $2m-1$, is referred to as the optimal normal basis (ONB). Two types of optimal normal bases have been found which are referred to as type I and type II optimal normal basis[30]. Reordered normal basis is referred to as a certain permutation of a type II optimal normal basis [12], [44].

Redundant representation is especially interesting because it not only offers almost free squaring as normal basis does, but also eliminates modular operation for multiplication. The main idea for multiplication using redundant representation is to embed a field in a larger ring and perform the multiplication there [44]. The ring used here has a simple structure and is referred to as a cyclotomic ring, such that the modular operation can be saved in a multiplication operation. Since embedding a field is not unique, each field element in the ring can be presented in more than one way, so the representation contains a certain amount of redundancy.

The main drawback for the redundant representation is that it uses more bits to represent a field element, where the number of representation bits depends on the size of the cyclotomic ring. For the class of fields $\mathbb{F}_{2^m}$ such that there exist a type I optimal normal basis (ONB), the number of bits required for a redundant representation of a field element is $m + 1$. Also, for the class of fields $\mathbb{F}_{2^m}$ such that there exist a type II optimal normal basis (ONB), the number of bits required for a redundant representation of a field element is $2m + 1$.

Hardware implementation of finite field multipliers can usually be divided into three categories. In the first category there are bit-level or bit-serial multipliers [22],[1],[15],[11]. A bit-level multiplier takes m clock cycles to finish one multiplication in a binary field of size m. The multipliers in this class are considered to have low power consumption, occupy a small area of silicon, and operate slowly for large field sizes. The second category are bit-parallel or full-parallel multipliers [35],[24],[20],[43]. A full parallel multiplier takes one clock cycle to finish one field multiplication. These multipliers are not usually economical for implementation since they require large silicon area and high bandwidth for input and output ports.

The third category are word-level or digit-level finite field multipliers, which are the most commonly implemented in practice [12],[44],[22],[32],[31],[36],[37]. A word-level multiplier takes w clock cycles, $1 \leqslant w \leqslant m$, to finish one multiplication operation in $\mathbb{F}_{2^m}$. The value of w can be selected by designer to set the trade off between area and speed according to the application. Decreasing the value of w will result in faster and larger multipliers while increasing w will make smaller and slower multipliers. Note that bit-level and full parallel multipliers can be viewed as special cases of word-level multipliers for $w = m$ and $w = 1$ respectively.

1.1 Summary of Contributions

In this work, different word-level architectures for multiplication using binary field are proposed. It has been shown that the proposed architectures are more efficient compared to similar proposals considering area/delay complexities as a measure of performance. Practical size multipliers for cryptography applications have been realized in hardware using

FPGA or standard CMOS technology. Also, different VLSI implementations for multipliers were explored, which resulted in more efficient implementations for some of the regular architectures. The new implementations use a simple module designed in domino logic as the main building block for the multiplier. Significant improvements were achieved designing practical size multipliers using the proposed methodology.

1.2 Outline of the Thesis

The rest of this thesis is organized as follows. Chapter 2, is a brief review of finite filed theory. After covering basic definitions and elementary properties such as group, ring and field, bases for finite fields are presented. Normal basis and redundant basis representation with their arithmetic operations are discussed in detail. Type I and II optimal normal basis, which are two important classes of normal basis, and their relationship with redundant representation are also discussed in this chapter.

Chapter 3 discusses two new high speed bit-serial word-parallel, or comb style finite field multipliers. The first proposal utilizes redundant representation for any binary field, and the other uses a reordered normal basis for the binary field where there exists a type II optimal normal basis. The proposed redundant representation architecture has a smaller critical path delay compared to the previous methods, while its complexity remains approximately the same. The proposed reordered normal basis multiplier has a significantly smaller critical path delay compared to the previous methods using the same basis or normal basis. FPGA implementation results of the proposed multipliers are compared to those of the previous methods using the same basis, confirming that the proposed multipliers allow for a much higher clock rate.

Chapter 4, presents a novel serial-in parallel-out finite field multiplier using redundant representation. It is shown that the proposed architecture has either significantly lower complexity and comparable critical path delay, or significantly smaller critical path delay and comparable complexity, in comparison to the previously proposed architectures using the same representation. For the class of fields such that there exists a type I optimal normal basis, the proposed multiplier compares favorably to the normal basis multipliers. A digit-level version for the new multiplier is also presented in this chapter.

In Chapter 5, a high speed word-level finite field multiplier in $\mathbb{F}_{2^m}$ using redundant representation is proposed. For the class of fields such that there exists a type I optimal normal basis, the new architecture has significantly higher speed compared to previously proposed word-level architectures using either normal basis or redundant representation at the expense of moderately higher area complexity. One of the unique features of the proposed word-level multiplier is that the critical path delay is not a function of the field size, nor the word size. It is also shown that the new multiplier out-performs all other multipliers in the comparison when considering the product of area and delay as a measure of performance. VLSI implementation of the proposed multiplier in a $0.18\mu m$ CMOS process is also presented as a module for an elliptic curve processor.

In Chapter 6, two high speed word-level finite field multipliers using reordered normal basis are proposed, where reordered normal basis is referred to as a certain permutation of type II optimal normal basis. Complexity comparison shows that the proposed architectures are faster than all the previously presented architectures in the open literature using either a type II optimal normal basis or a reordered normal basis at the expense of moderately higher complexity. One unique feature of the new word-level architectures is that the critical path delay is not a function of the word size or the field size. This enables the proposed multipliers to operate at very high clock rate regardless of the word or field size. Such high speed word-level multipliers are expected to have applications in public key cryptography, i.e. elliptic curve cryptosystems.

Chapter 7 presents a high speed VLSI implementation of a 233-bit Serial-In Parallel-Out finite field multiplier. The proposed design performs multiplication using a reordered normal basis; a permutation of a type II optimal normal basis. The multiplier was implemented in a $.18\,\mu m$ TSMC CMOS technology using multiples of a domino logic block. The domino logic design was simulated, and functioned correctly up to a clock rate of $1.587\,GHz$, yielding a 99% speed improvement over the static CMOS' simulation results, while the area was reduced by 49%. This multiplier's size of 233 bits is currently recommended by the National Institute of Standards and Technology (NIST) in their Elliptic Curve Digital Signature Standard (ECDSS), and is used in practice for binary field multiplication in Elliptic Curve Cryptosystems.

Finally some concluding remarks and future work are presented in Chapter 8.

Chapter 2

Mathematical Preliminaries

This chapter briefly reviews the mathematical background on finite fields. It starts with reviewing basic definitions such as group, ring, and field, and then covers more advanced topics such as bases and arithmetic operations. Normal basis and redundant representation with their arithmetic operations are discussed in detail. The relationship between different classes of normal basis and redundant representation is also discussed in this chapter. For a more detailed review of finite fields and their applications readers are referred to [25, 29, 26]

2.1 Groups, Rings and Fields

Definition 2.1.1. [25] A *group* $(G, *)$ is a set G together with a binary operation $*$ on G such that the following three properties hold:

1. The binary operator $*$ is associative; that is, for any $a, b, c \in G$,
   $a * (b * c) = (a * b) * c$

2. There is an identity (or unity) element e in G such that for all $a \in G$,
   $a * e = e * a = a$.

3. For each $a \in G$, there exist an inverse element a^{-1} in G such that
   $a * a^{-1} = a^{-1} * a = e$.

If for all $a, b \in G$, $a * b = b * a$, then G is referred to as an *abelian* or *commutative group.* A group with finite number of elements is referred to as a *finite group.*

Definition 2.1.2. [25] A *ring* $(r, +, *)$ is a set R together with two binary operations, denoted by $+$ and $*$, such that the following three properties hold:

1. R is an abelian group with respect to $+$.

2. The binary operator $*$ is associative, which means for all $a, b, c \in R$
   $(a * b) * c = a * (b * c)$.

3. The distribution law holds, which means for all $a, b, c \in R$
   $a * (b + c) = a * b + a * c$ and $(b + c) * a = b * a + c * a$.

The identity element of the abelian group R with respect to $+$ is called the *zero element*, while the identity element with respect to $*$ (if it exist) is called the *identity element.* A ring is called *commutative* if the binary operator $*$ is commutative.

Definition 2.1.3. [25] A *field* $(f, +, *)$ is a set F together with two binary operations, denoted by $+$ and $*$, such that the following two properties hold:

1. F is a commutative ring under $+$ and $*$.

2. Nonzero elements of F from a group with the binary operation $*$.

A field with a finite number of elements is referred to as a *finite field.* The order of a finite field is the number of elements in the field. There exists a finite field F of order q if and only if q is a prime power, that is $q = p^m$ where p is a prime number referred to as a the *characteristic* of F and m is a positive integer [18]. For any prime power q, there is essentially only one finite field of order q. This means that any two finite fields of order q are structurally the same, except that the labeling used to represent the field elements may be different. We say that any two finite fields of order q are *isomorphic*, and denote such a field by $\mathbb{F}_{q^m}$ or $GF(q^m)$ (GF stands for Galois Field, in honor of Evariste Galois, a French mathematician who is known for his work on the theory of equations and abelian integrals).

2.2 Binary Field and Bases

For a finite field F with order of $q = p^m$, if $m = 1$ then the field is referred to as a prime field. If $m \geq 2$, then the finite field is referred to as an extension field. Finite fields of order $q = 2^m$ are called *binary fields* or *characteristic-two finite fields.*

An important factor that has an important effect on finite field arithmetic efficiency is the basis used to represent the field elements. Common bases used in practice are polynomial basis (PB) and normal basis (NB) [25],[33]. Polynomial basis is probably the most popular basis which has been widely used for hardware and software implementations [18]. Normal basis, on the other hand, is advantageous for hardware implementation since the squaring operation can be implemented at no cost. Free squaring operations can be used to speed up the exponentiation operation by repeated squaring and multiplication [14],[2].

Recently, a method of redundant representation of field elements has attracted attention [42, 7, 38, 44]. The idea here was to use the minimal cyclotomic ring in which the current field can be embedded in, and perform the field arithmetic operations in the ring. Advantages of using redundant representation not only include the free squaring operation offered by this method but also its 'basis' elements form a cyclic group, and thus the modulo reduction step can be avoided carrying out the field multiplication operation.

2.2.1 Normal Basis and Its Arithmetic in $\mathbb{F}_{2^m}$

2.2.1.1 Normal Basis Representation

Theorem 2.2.1. [25] Let $P(x)$ be a degree m irreducible polynomial over $\mathbb{F}_{2^m}$ whose m roots $\{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ are linearly independent in $\mathbb{F}_{2^m}$. Then these m roots form a basis in $\mathbb{F}_{2^m}$ which is referred to as *normal basis.*

It is well known that there always exists a normal basis for the finite field $\mathbb{F}_{2^m}$ for all positive values of m [25]. Assume that $\beta \in \mathbb{F}_{2^m}$ is an element such that $I = \{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^{m-1}}\}$ is a normal basis, then element $A \in \mathbb{F}_{2^m}$ can be represented as:

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + \cdots + a_{m-1}\beta^{2^{m-1}}.$$

The main advantage of normal basis representation is that, element A can be squared by a simple right circular shift on its coordinates, $A^2 = \sum_{i=0}^{m-1} a_{(i+1)}\beta^{2^i}$, where $(i+1)$ denotes that $i+1$ is to be reduced modulo m. This property for normal basis comes from the fact that $\beta^{2^m} = \beta$, and is used to speed up exponentiation by use of the square and multiply algorithm [18].

2.2.1.2 Normal basis multiplication

Let field elements $A, B \in \mathbb{F}_{2^m}$ be represented with respect to (w.r.t.) the normal basis $I = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ as $A = \sum_{i=0}^{m-1} a_i\beta^{2^i}$ and $B = \sum_{j=0}^{m-1} b_j\beta^{2^j}$, respectively. Then the product of A and B can be given by

$$C = A \cdot B = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \beta^{2^i}\beta^{2^j} \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j (\beta\beta^{2^{(j-i)}})^{2^i}. \tag{2.1}$$

Define $t_{i,k} \in \mathbb{F}_2$ to be the coefficient of β^{2^k} in the expansion of the product $\beta\beta^{2^i}$ when represented w.r.t. I [30],

$$\beta\beta^{2^i} = \sum_{k=0}^{m-1} t_{i,k}\beta^{2^k}. \tag{2.2}$$

Then it follows from (2.2) that

$$(\beta\beta^{2^{(j-i)}})^{2^i} = \big(\sum_{k=0}^{m-1} t_{(j-i),k}\beta^{2^k}\big)^{2^i} = \sum_{k=0}^{m-1} t_{(j-i),k}\beta^{2^{k+i}}i = \sum_{k=0}^{m-1} t_{(j-i),(k-i)}\beta^{2^k}. \tag{2.3}$$

The last step in (2.3) follows from the proper substitution on the subscript k. Substituting $\beta\beta^{2^{(j-i)}}$ in (2.1) using (2.3)

$$C = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \big(\sum_{k=0}^{m-1} t_{(j-i),(k-i)}\beta^{2^k}\big) = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\sum_{k=0}^{m-1} a_i b_j t_{(j-i),(k-i)}\beta^{2^k}. \tag{2.4}$$

Then the coefficients of the product C w.r.t. the NB I can be given by

$$C = \sum_{k=0}^{m-1} c_k\beta^{2^k}, \text{ where } c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j t_{(j-i),(k-i)}. \tag{2.5}$$

Also note that from eqn (2.5) and after proper substitution on i and j, we can compute c_{k+1} with

$$c_{k+1} = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j t_{(j-i),(k-i+1)} = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_{(i+1)} b_{(j+1)} t_{(j-i),(k-i)}. \tag{2.6}$$

Eqn (2.6) shows that c_{i+1} can be computed by applying the same formula used to compute c_i, if the coefficient vectors of A and B are cyclically shifted by one.

To obtain the values of $t_{(j-i),(k-i)}$, a matrix $T = [t_{l,n}]$ is created where row l corresponds to the coefficients in the expansion of the product $\beta\beta^{2^l}$ and the column n corresponds to the coefficients of β^{2^n} in the expansion of the products $\beta\beta^{2^l}$, $l = 0, 1, \dots, m-1$.

Matrix T is referred to as the *multiplication table* for the normal basis in this thesis. Note that in [30] the same multiplication table was denoted by T_0 while matrices T_k, $k = 1, 2, \dots, m-1$, were defined for the expansion of $\beta^{2^k}\beta^{2^l}$. It also should be noted that other matrices $T_i, i = 1, 2, ..., m-1$ in [30], can be generated from T_0 and circular row/column shifting. In this thesis we use only one multiplication matrix T when deriving the formula for computing c_i, $i = 0, 1, \dots, m-1$.

Let the number of nonzero entries in T be denoted by C_N. It can be seen from equations (2.5) and (2.6) that the product coefficient c_k can be computed by summing up exactly C_N

terms. Thus, the generation of each c_k requires C_N multiplication operations in $\mathbb{F}_2$ and $C_N - 1$ addition operations in $\mathbb{F}_2$.

C_N is referred to as the complexity of a normal basis and it has been shown that $C_N \geq 2m - 1$ [30]. When $C_N = 2m - 1$ for a NB, it is called an optimal normal basis (ONB). Since computations in the optimal normal basis are minimized, these bases are of high importance for cryptographic applications. A type II ONB corresponds to the case where no row or column in T contains more than two nonzero entries.

Example 1 Consider the finite field $\mathbb{F}_{2^5}$. The root β of the irreducible polynomial $P(x) = x^5+x^4+x^3+x+1$ generates a normal basis $I = \{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}, \beta^{2^4}\}$. The multiplication table T can be found as

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

The closed form solution for multiplication can be easily found from (2.5) as follows:

$$\begin{aligned} c_i &= b_i a_{i+3} + b_{i+1}(a_{i+2} + a_{i+4}) + b_{i+2}(a_{i+1} + a_{i+4}) + \\ &\quad b_{i+3}(a_i + a_{i+4}) + b_{i+4}(a_{i+1} + a_{i+2} + a_{i+3} + a_{i+4}). \end{aligned} \tag{2.7}$$

2.2.2 Redundant Basis and its Arithmetic in $\mathbb{F}_{2^m}$

2.2.2.1 Redundant Representation

Let β be a primitive n^{th} root of unity in some extension field of $\mathbb{F}_2$ $(\beta^n = 1)$. The splitting field of β is called the n^{th} cyclotomic field and denoted by $\mathbb{F}_2^{(n)}$. Elements in $\mathbb{F}_2^{(n)}$ can be represented in the form

$$A = a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1},\ a_i \in \mathbb{F}_2,\ i = 0, 1, \cdots, n-1. \tag{2.8}$$

Let $\mathbb{F}_{2^m}$ be a field that can be embedded in $\mathbb{F}_2^{(n)}$. The following theorem characterizes the relationship between m and n .

Theorem 2.2.2. [25] Let n be an odd positive integer. Then, $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(n)}$ if and only if m divides the multiplicative order of 2 mod n.

For a given $\mathbb{F}_{2^m}$ we are particularly interested in $\mathbb{F}_2^{(n)}$ with the minimal value of n such that $\mathbb{F}_{2^m}$ can be embedded in $\mathbb{F}_2^{(n)}$. Obviously, field element $A \in \mathbb{F}_{2^m}$ can also be represented with (2.8). Note that $1 + \beta + \beta^2 + \cdots + \beta^{n-1} = 0$ and the representation of A is not unique. For example, the two n-tuples $(a_0, a_1, \cdots, a_{n-1})$ and $(a_0 + 1, a_1 + 1, \cdots, a_{n-1} + 1)$ represent the same element A. By slightly abusing the terminology, the set $[1, \beta, \beta^2, \cdots, \beta^{n-1}]$ is denoted as *redundant basis* (RB) for $\mathbb{F}_{2^m}$ [44]. Also note that the elements of a RB form a cyclic group of order n and

$$\beta \cdot \beta^i = \begin{cases} \beta^{i+1} & i \neq n-1, \\ 1 & i = n-1. \end{cases}$$

2.2.2.2 Redundant Basis Multiplication

Let field elements $A, B \in \mathbb{F}_{2^m}$ be represented with respect to the RB $I_1 = [1, \beta, \beta^2, \ldots, \beta^{n-1}]$ as

$$A = \sum_{i=0}^{n-1} a_i \beta^i \text{ and } B = \sum_{i=0}^{n-1} b_i \beta^i,$$

respectively, where $a_i, b_i \in \mathbb{F}_2, i = 0, 1, \cdots, n-1$. Note that $n \geqslant m+1$ and $\beta^n = 1$. Then it follows $\beta^i \cdot B = \sum_{j=0}^{n-1} b_{(j-i)} \beta^j$, where $(j-i)$ denotes that $j-i$ is to be reduced modulo n. The product of field elements A and B can be given by [44]

$$A \cdot B = \sum_{i=0}^{n-1} a_i (\beta^i \cdot B) = \sum_{i=0}^{n-1} a_i (\sum_{j=0}^{n-1} b_j \beta^{i+j}).$$

$$= \sum_{i=0}^{n-1} a_i (\sum_{j=0}^{n-1} b_{(j-i)} \beta^j) = \sum_{j=0}^{n-1} (\sum_{i=0}^{n-1} a_i b_{(j-i)}) \beta^j.$$

If we define $A \cdot B = C \triangleq \sum_{j=0}^{n-1} c_j \beta^j$, then c_j can be given by

$$c_j = \sum_{i=0}^{n-1} a_i b_{(j-i)}, \quad j = 0, 1, \ldots, n-1. \tag{2.9}$$

2.2.2.3 Redundant Basis and Normal Basis

As mentioned before, when the complexity of normal basis multiplication is minimized the normal basis is referred to as optimal normal basis. Optimal normal basis representations are classified as Type I or Type II. Since computations in the optimal normal basis are minimized, these bases are of high importance for cryptographic applications. For the class of fields where there exist a type I ONB, the size of the Redundant Basis representation is almost the same as that of the NB as shown in 2.2.3. Also for the class of fields where there exist a type II ONB, the size of the Redundant Basis representation is almost twice of that of the NB as shown in 2.2.4. Note that the complexities of redundant basis multipliers for the class of fields that there exist a type II ONB can be greatly reduced by applying a symmetry property in redundant representation which is shown in 2.2.5.

Remark 2.2.3. [44] If there is a type I optimal normal basis in $\mathbb{F}_{2^m}$, then $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(m+1)}$, so there is a redundant basis of size $m+1$ for $\mathbb{F}_{2^m}$.

Remark 2.2.4. [44] If there is a type II optimal normal basis in $\mathbb{F}_{2^m}$, then $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(2m+1)}$, so there is a redundant basis of size $2m+1$ for $\mathbb{F}_{2^m}$.

Remark 2.2.5. Assume that there exists a Gauss period of type (m, k) over $\mathbb{F}_2$. Then $I = [1, \beta, \beta^2, \ldots, \beta^{n-1}]$ is a redundant representation basis for $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, where $n = mk+1$. Let $A \in \mathbb{F}_{2^m}$ and $A = (a_0, a_1, \ldots, a_{n-1})$ with respect to I. Assume $k \geqslant 2$ is even, then

$$a_k = a_{n-k}\ , \ \text{for} \quad k = 1, 2, \ldots, n-1. \tag{2.10}$$

Proof: This is a direct result from Lemma 2 in [44] by noting that the redundant basis I and the basis I_4 used in [44] satisfy $I = 1 \cup I_4$.

Chapter 3

Comb Architectures for Finite Field Multiplication in $\mathbb{F}_2^m$

3.1 Introduction

For applications to elliptic curve and ElGamal public key cryptography, binary field sizes are required to be at least 160 and 1000 bits respectively [18]. A full bit-parallel finite field multiplier in these fields could be difficult to implement using current VLSI technology and also inefficient when considering that the width of the system data bus is usually much smaller than the size of the field. Whereas a bit-serial finite field multiplier in $\mathbb{F}_{2^m}$ usually requires m clock cycles to perform one operation which is too slow, hybrid architectures offer moderate complexity and relatively high speed.

There are at least two types of hybrid architectures: bit-parallel word-serial and bit-serial word-parallel (or comb style). One important difference between the two types of architectures is that the throughput [1] of a bit-parallel word-serial architecture is proportional to the word size whereas the throughput of a comb style architecture is proportional to the reciprocal of the word size. This difference makes a comb style architecture attractive where both high throughput and small word size are required. For example, a comb style architecture would sustain a higher throughput compared to a bit-parallel word-serial one, if word size is chosen to be smaller than the square root of the field size m.

One important factor that affects the finite field computation efficiency is the choice of the basis. A few types of bases have been utilized for construction of finite field multipliers, which include polynomial basis [27], normal basis [22], dual basis [3], triangular basis [19] and redundant representation or redundant basis [7, 44, 23]. The advantage of redundant basis is that all the basis elements form a cyclic group, so that computation of modulo reduction can be saved in multiplication operation.

The idea of using redundant representation was first introduced in [7], where arithmetic of $\mathbb{F}_{2^m}$ is performed in the ring $\mathbb{F}_2[x]/(x^n-1)$. In [7], n is chosen as the minimal value such that an irreducible polynomial of degree m is a factor of x^n-1. This representation is called polynomial ring basis representation. It was later found in [44] that the value of n can be reduced further while having $\mathbb{F}_2^m$ embedded in $\mathbb{F}_2[x]/(x^n-1)$. It has been proved in [44] that the value for n is optimal when $\mathbb{F}_2^{(n)}$ is the cyclotomic field of $\mathbb{F}_2^m$. When a type II optimal normal basis exists in $\mathbb{F}_2^m$, it is found in [12, 44] that a reordered normal basis can be derived from the redundant basis such that this reordered normal basis not only offers free squaring operation but also avoids modulo reduction step in a multiplication operation.

In this chapter, we propose two new bit-serial word-parallel or comb style multipliers, one using redundant representation and the other using reordered normal basis. It is shown that the proposed redundant representation multiplier has smaller critical path delay and thus can operate much faster compared to the previous similar methods. The proposed reordered normal basis multiplier also has significantly smaller critical path delay compared

[1]Here we refer to throughput as the number of operations per one clock cycle.

to most of recent similar multipliers using reorder normal basis or normal basis. Note by choosing the word size equal to one or the field size m the proposed multiplier would have a bit-serial or a full bit-parallel architecture. We have also implemented the proposed multipliers using FPGA and the results show that the proposed multipliers are faster than the ones presented in [44] by 66 and 81 percent respectively.

The organization of the rest of this chapter is as follows: Section 2 is a brief review of redundant basis, reordered normal basis and their multiplication operations. New hybrid multipliers using redundant basis and reordered normal basis are proposed in Sections 3 and 4, respectively. The complexities of the proposed multipliers are also compared to other previous articles in these sections. FPGA implementations of different word size multipliers are presented in Section 5, and a few concluding remarks are given in Section 6.

3.2 Preliminaries on Finite Field Bases and Arithmetic in $\mathbb{F}_{2^m}$

3.2.1 Redundant Representation and Multiplication

Let $x^n - 1$ be a polynomial defined over $\mathbb{F}_2$. Then the splitting field of $x^n - 1$ is called nth *cyclotomic field*, denoted by $\mathbb{F}_2^{(n)}$. Let $\mathbb{F}_{2^m}$ be a field which can be embedded in $\mathbb{F}_2^{(n)}$. Then the following theorem characterizes the relationship between n and m.

Theorem 3.2.1. Let n be an odd positive integer. Then, $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(n)}$ if and only if m divides the multiplicative order of 2 mod n.

Proof: This is a special case of the theorem in [44]. □

We are particularly interested in $\mathbb{F}_2^{(n)}$ with the minimal value of n which $\mathbb{F}_{2^m}$ can be embedded in. Let β belong to some extension field of $\mathbb{F}_2$ and be a primitive nth root of unity. Then $\mathbb{F}_2^{(n)}$ can be generated by β and elements of $\mathbb{F}_2^{(n)}$ can be represented by

$$A = a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1},\ a_i \in \mathbb{F}_2,\ i = 0, 1, \cdots, n-1. \qquad (3.1)$$

Such representation of A is not unique since $1+\beta+\beta^2+\cdots+\beta^{n-1}=0$. we call (3.1) the *redundant representation* of A, or by slight abuse of the term 'basis', we call the set $[1,\beta,\beta^2,\ldots,\beta^{n-1}]$ a *redundant basis* for any subfield of $\mathbb{F}_2^{(n)}$ [44].

One advantage of using redundant basis for finite field arithmetic is that the modulo reduction step can be avoided for multiplication operation. This is due to the fact that the redundant basis elements form a cyclic group of order n and

$$\beta\cdot\beta^i=\begin{cases}\beta^{i+1} & i\neq n-1,\\ 1 & i=n-1.\end{cases}$$

Consider the redundant basis for $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$:

$$I_1=[1,\beta,\beta^2,\ldots,\beta^{n-1}].$$

Let $A=(a_0,a_1,\ldots,a_{n-1}), B=(b_0,b_1,\ldots,b_{n-1})\in\mathbb{F}_{2^m}$ be represented with respect to (w.r.t.) I_1, where $a_i,b_i\in\mathbb{F}_2$. The multiplication operation can be given by [44]:

$$A\cdot B=C\triangleq\sum_{j=0}^{n-1}c_j\beta^j,$$

where

$$c_j=\sum_{i=0}^{n-1}a_ib_{(j-i)},\quad j=0,1,\ldots,n-1. \tag{3.2}$$

Note that $(j-i)$ denotes that $j-i$ is to be reduced modulo n.

3.2.2 Reordered Normal Basis Representation and Multiplication

Theorem 3.2.2. [12] Let β be a primitive $(2m+1)^{\text{st}}$ root of unity in $\mathbb{F}_{2^m}$ and $\gamma=\beta+\frac{1}{\beta}$ generates a type II optimal normal basis. Then $\{\gamma_i, i=1,2,\ldots,m\}$ with $\gamma_i=\beta^i+\frac{1}{\beta^i}=\beta^i+\beta^{2m+1-i}$, $i=1,2,\ldots,m$, is also a basis in $\mathbb{F}_{2^m}$.

It has been shown that the basis $\{\gamma_i, i=1,2,\ldots,m\}$ is a permutation of the normal basis $\{\gamma^{2^i}, i=0,1,\ldots,m-1\}$ [44]. We denote the basis $I_2=[\gamma_1,\gamma_2,\ldots,\gamma_m]$ as the reordered normal basis following [44].

Reordered normal basis not only offers free squaring but also can avoid modulo reduction step in a multiplication operation. Multiplication using the reordered normal basis can be described as follows. Define

$$s(i) \triangleq \begin{cases} i \bmod 2m+1, & \text{if } 0 \leqslant i \bmod 2m+1 \leqslant m, \\ 2m+1-i \bmod 2m+1, & \text{otherwise.} \end{cases}$$

Let $A = (a_1, \ldots, a_m), B = (b_1, \ldots, b_m) \in \mathbb{F}_{2^m}$ w.r.t. I_2, and $b_0 = 0$, then the product coefficients are given by [12, 44]

$$c_j = \sum_{i=1}^{m} a_i(b_{s(j+i)} + b_{s(j-i)}), \quad j = 1, 2, \ldots, m. \tag{3.3}$$

where c_j is defined by

$$A \cdot B = C = \sum_{j=1}^{m} c_j \gamma_j.$$

3.3 Proposed Hybrid Multiplier Using Redundant Representation

3.3.1 Bit-serial word-parallel multiplication algorithm

From (3.2) it can be seen that one product bit c_j is a sum of n terms where each term is a partial product bit $a_i b_{(j-i)}$. Let w denote the word size. Write the subscript of a_i in (3.2) as $i = kw + \ell$, $k = 0, 1, \ldots, \lceil n/w \rceil - 1$ and $\ell = 0, 1, \ldots, w-1$. Replace i in (3.2) with $kw + \ell$:

$$c_j = \sum_{\ell=0}^{w-1} \sum_{k=0}^{\lceil n/w \rceil - 1} a_{kw+\ell} b_{(j-kw-\ell)}, \quad j = 0, 1, \ldots, n-1. \tag{3.4}$$

Let

$$c_j^{(\ell)} = \sum_{k=0}^{\lceil n/w \rceil - 1} a_{kw+\ell} b_{(j-kw-\ell)}, \quad \ell = 0, 1, \ldots, w-1. \tag{3.5}$$

Then (3.4) becomes

$$c_j = \sum_{\ell=0}^{w-1} c_j^{(\ell)}, \; j = 0, 1, \ldots, n-1. \tag{3.6}$$

$$\begin{array}{cccccc}
A_{\lceil n/w\rceil-1} & = & a_{\lceil n/w\rceil w-1} & a_{\lceil n/w\rceil w-2} & \cdots & a_{(\lceil n/w\rceil-1)w} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
A_1 & = & a_{2w-1} & a_{2w-2} & \cdots & a_w \\
A_0 & = & a_{w-1} & a_{w-2} & \cdots & a_0 \\
 & & \downarrow & \downarrow & \cdots & \downarrow \\
 & & A[w-1] & A[w-2] & \cdots & A[0]
\end{array}$$

Figure 3.1: Words vs. comb style inputs

Clearly, if $c_j^{(\ell)}, \ell = 0, 1, \ldots, w-1, j = 0, 1, \ldots, n-1$, can be computed and implemented in one clock cycle, then it takes only w clock cycles to obtain c_j for $j = 0, 1, \ldots, n-1$.

Clearly a multiplication carried out with (3.5) and (3.6) requires a comb style input A while input B has a much more complex format. Input operand A can be represented with $\lceil n/w\rceil$ words

$$\begin{aligned}
A &= \underbrace{a_{\lceil n/w\rceil w-1}a_{\lceil n/w\rceil w-2}\cdots a_{(\lceil n/w\rceil-1)w}} \quad \cdots \quad \underbrace{a_{w-1}a_{w-2}\cdots a_0} \\
&= \qquad\qquad A_{\lceil n/w\rceil-1} \qquad\qquad\quad \cdots \qquad\quad A_0,
\end{aligned}$$

where each word $A_i = a_{iw+w-1}a_{iw+w-2}\cdots a_{iw}$ contains w bits. Note if w can not divide n then the most significant word $A_{\lceil n/w\rceil-1}$ of A would contains some zero bits at its most significant bit positions. Let the comb inputs from A be denoted by $A[w-1], A[w-2], \ldots, A[0]$. Fig. 3.1 shows how the comb inputs can be obtained from the words of the input operand A. It can be seen that

$$A[\ell] = a_{(\lceil n/w\rceil-1)w+\ell}a_{(\lceil n/w\rceil-2)w+\ell}\cdots a_\ell, \quad \text{for } \ell = 0, 1, \ldots, w-1.$$

Define $B^{(j)}[\ell]$ such that

$$B^{(j)}[\ell] = b_{(j-(\lceil n/w\rceil-1)w-\ell)}b_{(j-(\lceil n/w\rceil-2)w-\ell)}\cdots b_{(j-\ell)}.$$

Note that $B^{(j+1)}[\ell]$ is a leftward circular shift of $B^{(j)}[\ell]$ by one bit.

Let the inner-product between two n-tuple vectors X and Y be given by

$$X \odot Y = \sum_{i=0}^{n-1} x_i y_i,$$

where $X = (x_0 x_1 \cdots x_{n-1})$, $Y = (y_0 y_1 \cdots y_{n-1})$ and $x_i, y_i \in \mathbb{F}_2$. Then it follows from (3.4),

$$c_j = \sum_{l=0}^{w-1} A[\ell] \odot B^{(j)}[\ell], \quad j = 0, 1, \ldots, n-1.$$

The algorithm for comb style multiplication using redundant representation is given below.

Algorithm 1. Comb style multiplication using redundant basis

```
Input:    $A[\ell], B^{(j)}[\ell]$
Output:   $c_j$
1.    For $j = 0$ to $n-1$
2.        $c_j = 0$;
3.        For $\ell = 0$ to $w-1$
4.            $c_j = c_j + A[\ell] \odot B^{(j)}[\ell]$;
```

3.3.2 Comb style multiplication architecture

A comb style multiplication architecture can be developed from Algorithm 1. Operand A is available at input in a comb style, *i.e.*, $A[\ell]$ is available at ℓth cycle, for $\ell = 0, 1, \ldots, w-1$. Operand B is stored in a circular shift register from which $B^{(j)}[\ell], j = 0, 1, \ldots, n-1$ can be read from in cycle ℓ. A combinatorial circuit module consisting of $\lceil n/w \rceil$ AND gates and $\lceil n/w \rceil$ XOR gates is designed to generate the inner-product $A[\ell] \odot B^{(j)}[\ell]$ in one clock cycle.

In the first clock cycle the input bits are $A[0] = a_{(\lceil n/w \rceil - 1)w} a_{(\lceil n/w \rceil - 2)w} \ldots a_0$ and by the end of cycle the contents of registers c_j are $A[0] \odot B^{(j)}[0]$. In the second clock cycle the input bits are $A[1]$ and by the end of cycle the contents of c_j are $A[0] \odot B^{(j)}[0] + A[1] \odot B^{(j)}[1]$. In the wth clock cycle the input bits are $A[w-1]$. Note that an input bit whose index exceeds $n-1$ should be replaced by a zero bit. By the end of the w^{th} cycle the contents of registers c_j are

$$c_j = \sum_{\ell=0}^{w-1} A[\ell] \odot B^{(j)}[\ell], \quad \text{for } j = 0, 1, \ldots, n-1.$$

Clearly, w clock cycles are required to finish one multiplication operation.

The output bits c_j is produced with an accumulation circuits after w cycles. A comb style redundant basis multiplier for $\mathbb{F}_{2^4}$ is shown in Fig. 3.2, where $n = 5$ and we choose $w = 2$. The multiplier architecture discussed so far is a least-significant-bit (LSB) first version, where the multiplier takes operand A in the order of $A[0], A[1], \ldots, A[w-1]$. A most-significant-bit (MSB) first version of comb style multiplication algorithm can also be developed by changing line 3. in Algorithm 1 to

```
3.        For $\ell = w - 1$ to $0$ step $-1$
```

A MSB first version of the comb style redundant basis multiplier for $\mathbb{F}_{2^4}$ is shown in Fig. 3.3.

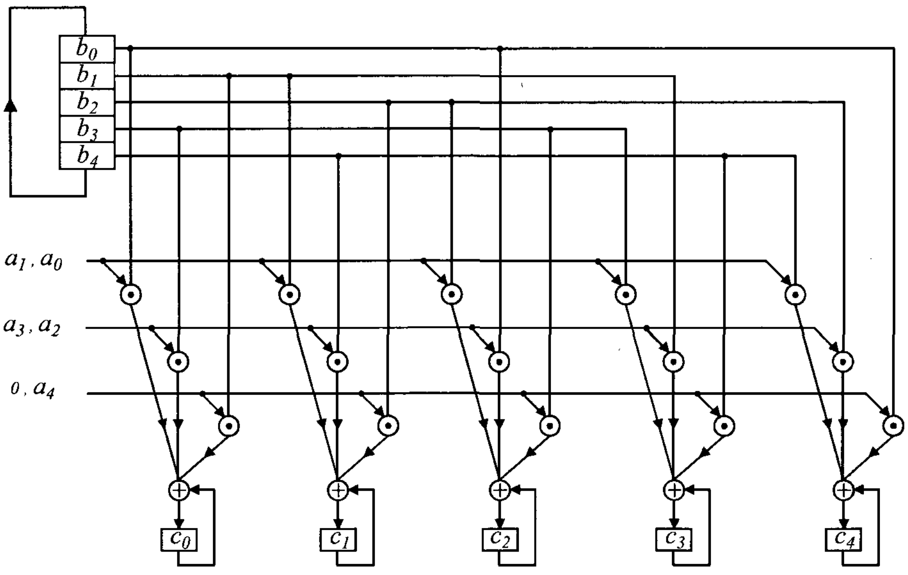


Figure 3.2: Proposed comb style redundant basis multiplier for $\mathbb{F}_{2^4}$ (LSB first)

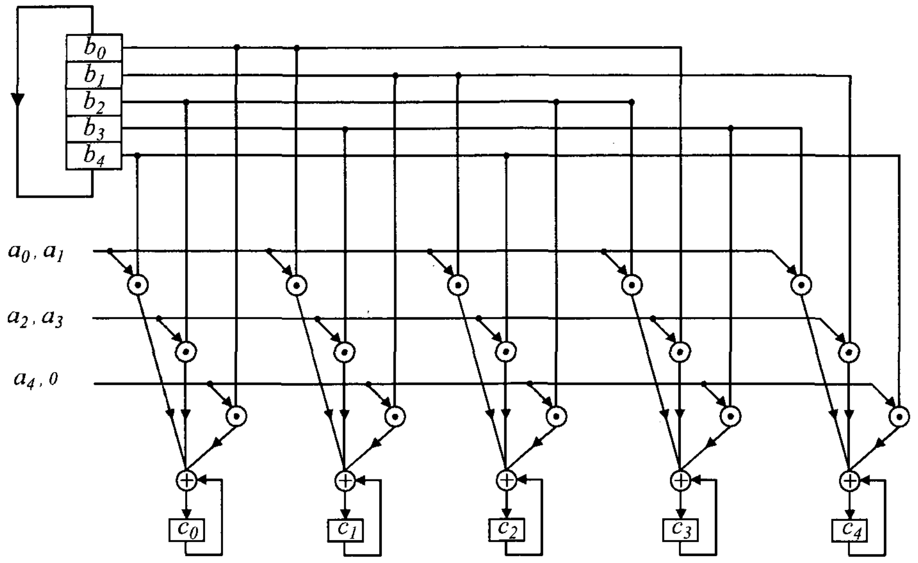


Figure 3.3: Proposed comb style redundant basis multiplier for $\mathbb{F}_{2^4}$ (MSB first)

3.3.3 Architecture complexities and comparison

Architectural complexity and gate counts for the proposed multiplier along with similar previous proposals are shown in Table 3.1. In the table the delay of a two-input AND gate is denoted by T_A and the delay for an n-input XOR gate is approximated by $\lceil \log_2 n \rceil T_X$. The first row of the table shows the complexity result of the hybrid multiplier previously proposed in [44], where w denotes the number of parallel modules. If we set the value of w for the hybrid multiplier in [44] to be approximately equal to $\lceil m/w \rceil$ in the proposed multiplier, then the complexities and the number of clock cycles for the two multipliers are about the same except the critical path delay, where the proposed multiplier has significantly less critical path delay than the one in [44].

Multipliers	Basis	# AND	# XOR	# Reg(bits)	# Cycles	Critical Path Delay
Hybrid [44]	redundant	nw	$(n-1)w$	$2n$	$\lceil n/w \rceil$	$T_A + \lceil \log_2 n \rceil T_X$
Proposed	redundant	$n\lceil n/w \rceil$	$n\lceil n/w \rceil$	$2n$	w	$T_A + \lceil \log_2 \lceil n/w \rceil \rceil T_X$

Table 3.1: Complexities comparison between hybrid redundant basis multipliers

Note that the comb style input does not cause any problem. Assume that a point addition/doubling is the basic operation for an elliptic curve system and a finite field exponentiation is the basic operation for an ElGamal system. There are dozens of finite field multiplications, squarings and/or additions required to complete one basic operation [18]. Consider a system where finite field addition and squaring operations are performed by bit-parallel structures and multiplication operations are performed by the proposed multiplier. Since all the sum or product bits are available at the same time, it does not make any difference in what order the input coefficients are fed into the multiplier.

3.4 Proposed Hybrid Multiplier Using Reordered Normal Basis

3.4.1 Bit-serial word-parallel multiplication algorithm

It can be seen from (3.3) that c_j is a sum of m terms where each term is a partial product bit $a_i b_{(j-i)}$. Let w denote the word size. Write the subscript of a_i in (3.3) as $i = kw + \ell$, $k = 0, 1, \ldots, \lceil m/w \rceil - 1$ and $\ell = 1, 2, \ldots, w$. Replace i in (3.3) with $kw + \ell$:

$$c_j = \sum_{\ell=1}^{w} \sum_{k=0}^{\lceil m/w \rceil - 1} a_{kw+\ell}\left(b_{s(j+kw+\ell)} + b_{s(j-kw-\ell)}\right), \tag{3.7}$$

for $j = 1, \ldots, m$. Let

$$c_j^{(\ell)} = \sum_{k=0}^{\lceil m/w \rceil - 1} a_{kw+\ell}\left(b_{s(j+kw+\ell)} + b_{s(j-kw-\ell)}\right),$$

for $\ell = 1, 2, \ldots, w$. If $c_j^{(\ell)}$ can be implemented and computed in one clock cycle, then it takes only w clock cycles to obtain $c_j = \sum_{\ell=1}^{w} c_j^{(\ell)}$.

Let $A[\ell], B_-^{(j)}[\ell]$ and $B_+^{(j)}[\ell]$ be defined respectively by

$$\begin{aligned} A[\ell] &= a_{(\lceil n/w \rceil -1)w+\ell} a_{(\lceil n/w \rceil -2)w+\ell} \cdots a_\ell, \\ B_-^{(j)}[\ell] &= b_{(j-(\lceil m/w \rceil -1)w-\ell)} b_{(j-(\lceil m/w \rceil -2)w-\ell)} \cdots b_{(j-\ell)}, \text{ and} \\ B_+^{(j)}[\ell] &= b_{(j+(\lceil m/w \rceil -1)w+\ell)} b_{(j+(\lceil m/w \rceil -2)w+\ell)} \cdots b_{(j+\ell)}, \text{ for } \ell = 1, 2, \ldots, w. \end{aligned}$$

Then it follows from (3.7),

$$c_j = \sum_{\ell=1}^{w} A[\ell] \odot (B_-^{(j)}[\ell] + B_+^{(j)}[\ell]), \quad j = 1, 2, \ldots, m. \tag{3.8}$$

The algorithm (LSB first) for comb style multiplication using reordered normal basis is given below. A MSB version of this algorithm is also available by changing line 3.

Algorithm 2. Comb style multiplication using reordered normal basis (LSB first)

```
Input:   $A[\ell], B_-^{(j)}[\ell], B_+^{(j)}[\ell])$
Output:  $c_j$
1.   For $j = 1$ to $m$
2.       $c_j = 0$;
3.       For $\ell = 1$ to $w$
4.           $c_j = c_j + A[\ell] \odot (B_-^{(j)}[\ell] + B_+^{(j)}[\ell])$;
```

3.4.2 Bit-serial word-parallel multiplier architecture

Similar to the redundant basis multiplier proposed in the previous section, a comb style multiplication architecture can be developed from Algorithm 2. Operand A is available at input in comb style, *i.e.*, $A[\ell]$ available at ℓ^{th} cycle, $\ell = 0, 1, \ldots, w-1$. Note that the circular shift register to store B is $(2m-1)$ bits which is almost double the size of operand B. The initial contents of this $(2m-1)$-bit shift register have to be carefully arranged so that each of the following clock cycle both $B_-^{(j)}[\ell]$ and $B_+^{(j)}[\ell], j = 0, 1, \ldots, n-1$, can be read from the register. A combinatorial circuits network is used to generate the inner-product $A[\ell] \odot (B_-^{(j)}[\ell] + B_+^{(j)}[\ell])$ in each clock cycle.

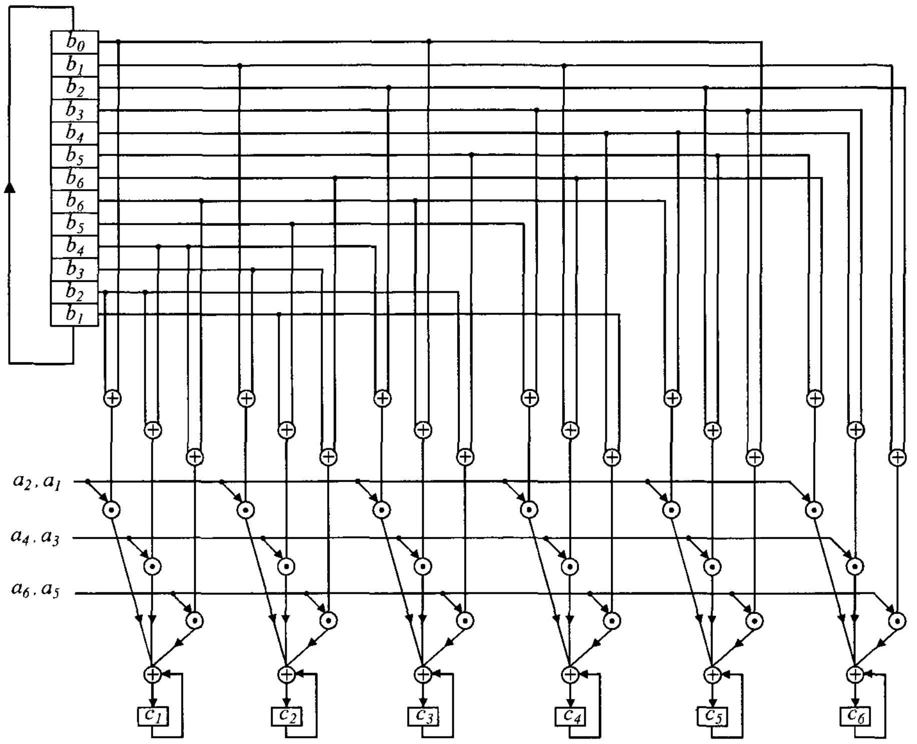


Figure 3.4: Proposed comb style multiplier in $\mathbb{F}_{2^6}$ using reordered normal basis (LSB first)

For the LSB first version of the multiplier architecture, by the end of the first clock cycle the contents of registers c_j are $A[0] \odot (B_-^{(j)}[0] + B_+^{(j)}[0])$. By the end of the second cycle the contents of c_j are $A[0] \odot (B_-^{(j)}[0] + B_+^{(j)}[0]) + A[1] \odot (B_-^{(j)}[1] + B_+^{(j)}[1])$. By the end of the w^{th} clock cycle the contents of registers c_j are

$$c_j = \sum_{\ell=0}^{w-1} A[\ell] \odot (B_-^{(j)}[\ell] + B_+^{(j)}[\ell]), \quad \text{for } j = 0, 1, \ldots, n-1.$$

An MSB version of the multiplier can also be easily developed similar to the one presented in Section 3.

Fig. 3.4 shows such a multiplier architecture for $\mathbb{F}_2^6$ when $w = 2$. Note that $b_0 = 0$. Output registers are initialized as zero and after $\lceil m/w \rceil = 3$ clock cycles they should contain the product bits $c_1, c_2, \ldots, c_6$.

3.4.3 Architecture complexities and comparison

The architecture complexities of the proposed hybrid multiplier using reordered normal basis are shown in Table 3.2, where optimal normal basis type II is denoted by ONB II and reordered normal basis is denoted by RNB. Complexities for some previously proposed hybrid multipliers using reordered normal basis and normal basis are also listed in the table. Note that most of the previous proposals are bit-parallel word-serial (BPWS) or digit-level architectures whereas the proposed architecture is bit-serial word-parallel (BSWP). Also note that we use d to denote the word (digit) size for BPWS multipliers and w to denote the word size for the proposed BSWP architecture.

In order to make comparisons of different styles of architectures on a fair background, we assume that all the architectures take the same number of clock cycles to complete one multiplication operation. It means the value of w for the proposed multiplier is equal to $\lceil m/d \rceil$ for all the previous proposals shown in table.

Multipliers	Basis	Style	# AND	# XOR	# Cycl	Critical Path Delay
MO [41]	ONB II	BPWS	$(2m-1)d$	$(2m-2)d$	$\lceil m/d \rceil$	$T_A + \lceil \log_2(2m-1) \rceil T_X$
AEDS [36]	ONB II	BPWS	$(m - d/2 + 1/2)d$	$(3m - d - 2)d$	$\lceil m/d \rceil$	$T_A + (1 + \lceil \log_2(m) \rceil)T_X$
XEDS [36]	ONB II	BPWS	$(2m - n)d$	$(2m - d/2 - 3/2)d$	$\lceil m/d \rceil$	$T_A + (1 + \lceil \log_2(m) \rceil)T_X$
w-SMPO I [37]	ONB II	BPWS	$m + (\lfloor m/2 \rfloor + 1)d$	$(2m-1)d$	$\lceil m/d \rceil$	$2T_A + (3 + \lceil \log_2(d-1) \rceil)T_X$
w-SMPO II [37]	ONB II	BPWS	$m + md$	$(m + \lfloor m/2 \rfloor)d$	$\lceil m/d \rceil$	$2T_A + (3 + \lceil \log_2(d-1) \rceil)T_X$
Hybrid [44]	RNB	BPWS	md	$(2m-1)d$	$\lceil m/d \rceil$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$
Proposed here	RNB	BSWP	$m\lceil m/w \rceil$	$2m\lceil m/w \rceil$	w	$T_A + (1 + \lceil \log_2(\lceil m/w \rceil + 1) \rceil)T_X$

Table 3.2: Complexity comparison of hybrid multipliers using reordered normal basis or normal basis

Fact 3.4.1. Assume that $m \geq 4$. Also assume that all multipliers shown in Table 3.2 are neither bit-serial nor full bit-parallel architectures and they take the same number of clock cycles to complete one multiplication. Then the proposed BSWP multiplier has the smallest critical path delay among all the multipliers listed in Table 3.2.

Proof: Since $w \geq 2$ the following inequality holds, $\lceil m/w \rceil \leq \lceil m/2 \rceil \leq (m+1)/2$.

Also because $m \geq 4$, it follows $2m - 1 = m + m - 1 \geq m + 3$. Then,

$$\begin{aligned} T_A + (1 + \lceil \log_2(\lceil m/w \rceil + 1) \rceil) T_X &\leq T_A + (1 + \lceil \log_2 \frac{m+3}{2} \rceil) T_X \\ &\leq T_A + (1 + \lceil \log_2 \frac{2m-1}{2} \rceil) T_X \\ &= T_A + \lceil \log_2 (2m-1) \rceil T_X \qquad (3.9) \end{aligned}$$

It is assumed that $w = \lceil m/d \rceil$, so $w \geq m/d$. Then we have $d \geq m/w$ or $d \geq \lceil m/w \rceil (\because$ d is an intger). Also we assumed that $d \geq 2$, which follows

$$4d = d + 3d \geq \lceil m/w \rceil + 6 > \lceil m/w \rceil + 5,$$

or

$$4(d-1) > \lceil m/w \rceil + 1.$$

It follows

$$\begin{aligned} 2T_A + (3 + \lceil \log_2(d-1) \rceil) T_X &> T_A + (1 + \lceil \log_2 4(d-1) \rceil) T_X \\ &> T_A + (1 + \lceil \log_2(\lceil m/w \rceil + 1) \rceil) T_X \qquad (3.10) \end{aligned}$$

Also note from (3.9) that

$$\begin{aligned} T_A + (1 + \lceil \log_2 m \rceil) T_X &= T_A + \lceil \log_2 2m \rceil T_X \\ &\geq T_A + \lceil \log_2 (2m-1) \rceil T_X \\ &\geq T_A + (1 + \lceil \log_2(\lceil m/w \rceil + 1) \rceil) T_X \qquad (3.11) \end{aligned}$$

Summarizing from (3.9), (3.10) and (3.11), we can conclude that the proposed BSWP multiplier has the smallest critical path delay among all the multipliers listed in Table 3.2. □

The number of AND gates for the proposed multiplier is also the smallest except for AEDS [36] which uses much more XOR gates. The number of XOR gates for the proposed multiplier is moderate, smaller than that of AEDS [36] and w-SMPO II [37] but higher than that of XEDS [36].

3.5 FPGA Implementations

The proposed multipliers have been implemented in FPGA. For the comparison purpose, some of the previous proposals are also implemented using the same FPGA technology. Considering the large number of gates involved in the proposed multipliers for a binary field of size with practical significance, we used Altera Stratix FPGA family to implement the multipliers.

FPGA implementation results of comb style redundant basis multiplier for $n = 211$ is shown in Table 3.3, along with the results for the hybrid redundant basis multiplier in [44]. We deliberately choose different values of w for the two multipliers such that w for the proposed multiplier is equal to $\lceil n/w \rceil$ for the multiplier in [44]. This setting allows the two multipliers to take the same number of clock cycles to complete one multiplication operation. In fact, we set $w = 16$ for the proposed multiplier and $w = 14$ or $n/w = 16$ for the multiplier in [44]. It can be seen from Table 3.3 that the proposed multiplier uses slightly more logic elements but has a much lower critical path delay than the multiplier in [44].

We have implemented the comb style reordered normal basis multiplier for $\mathbb{F}_{2^{209}}$ and $w = 16$. For comparison purpose, the hybrid reordered normal basis multiplier in [44] has also been implemented with FPGA, where w is chosen as 14 such that $\lceil n/w \rceil = 16$. The implementation results are shown in Table 3.4. It can be seen that the proposed multiplier not only uses sightly fewer logic elements but also has a much lower critical path delay. It can be seen from Tables 3.3 and 3.4 that the speed improvements are 66% and 81%, respectively.

Multiplier	Field Size (n)	# Logic Elements	Critical Path Delay	# Clock Cycles
PISO [44]	211	2185	$6.25ns$	16
Proposed	211	2321	$3.77ns$	16

Table 3.3: Comparison of FPGA implementations of hybrid redundant basis multipliers

Multiplier	Field Size	# Logic Elements	Critical Path Delay	# Clock Cycles
PISO [44]	209	3625	$7.96ns$	16
Proposed	209	3558	$4.40ns$	16

Table 3.4: FPGA implementation results for hybrid reordered normal basis multipliers

3.6 Conclusions

Two new bit-serial word-parallel or comb style finite field multipliers, one using redundant representation and the other using reordered normal basis, have been proposed. The hybrid architecture gives the designer the ability to set the trade off between area and speed. Architectural complexities of the proposed multipliers compare favorably to the previously proposed architectures of similar type. The two proposed multipliers have also been implemented in FPGA. The hardware implementation results show that the proposed multipliers have much lower critical path delays thus allowing much faster operating clock rates. The proposed architectures are suitable for high speed cryptographic applications such as elliptic curve cryptography.

Chapter 4

A New Finite Field Multiplier Using Redundant Representation

4.1 Introduction

In [44], a redundant representation was derived from the minimal cyclotomic ring. The idea here was to use the minimal cyclotomic ring in which the current field can be embedded in, and the field arithmetic operations are performed in the ring. Advantages of using redundant representation include that this method not only offers free squaring operation as a normal basis does but also its 'basis' elements form a cyclic group and thus modulo reduction step can be avoided in field multiplication operation. Two different types of bit-serial multipliers using redundant representation have been proposed. One is parallel-in serial-out (PISO) and the other is serial-in parallel-out (SIPO) [44]. In this chapter, a novel SIPO multiplier architecture is proposed. Compared to the previous SIPO architecture, the proposed multiplier has significantly lower complexity while having the same critical path delay. Compared to the previous PISO multiplier, the proposed architecture has significantly smaller critical time delay while using same number of gates and registers. For a class of fields that there exists a type I optimal normal basis (ONB), it is also shown that the proposed multiplier has lower complexity and smaller critical path delay than the recently proposed NB multipliers.

The organization of the rest of the chapter is as follows. Redundant representation and its multiplication are reviewed in Section 2. A brief overview of previously proposed bit-serial redundant representation multipliers is also given in this section. A new algorithm and architecture for bit-serial redundant representation multiplication is proposed in Section 3. Complexity comparison is made in Section 4. In Section 5, a digit-level version of the proposed multiplier is presented. A few concluding remarks are made in Section 6.

4.2 Preliminaries

4.2.1 Redundant Representation

Let β be a primitive n^{th} root of unity in some extension field of $\mathbb{F}_2$. The splitting field of β is called the n^{th} cyclotomic field and denoted by $\mathbb{F}_2^{(n)}$. Elements in $\mathbb{F}_2^{(n)}$ can be represented in the form

$$A = a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1},\ a_i \in \mathbb{F}_2,\ i = 0, 1, \cdots, n-1. \tag{4.1}$$

Let $\mathbb{F}_{2^m}$ be a field that can be embedded in $\mathbb{F}_2^{(n)}$. The following theorem characterizes the relationship between m and n .

Theorem 4.2.1. [25] Let n be an odd positive integer. Then, $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(n)}$ if and only if m divides the multiplicative order of *2* mod n.

For a given $\mathbb{F}_{2^m}$ we are particularly interested in $\mathbb{F}_2^{(n)}$ with the minimal value of n such that $\mathbb{F}_{2^m}$ can be embedded in $\mathbb{F}_2^{(n)}$. Obviously, field element $A \in \mathbb{F}_{2^m}$ can also be represented with (4.1). Note that $1 + \beta + \beta^2 + \cdots + \beta^{n-1} = 0$ and the representation of A is not unique. For example, the two n-tuples $(a_0, a_1, \cdots, a_{n-1})$ and $(a_0 + 1, a_1 + 1, \cdots, a_{n-1} + 1)$ represent the same element A. By slightly abusing the terminology, the set $[1, \beta, \beta^2, \cdots, \beta^{n-1}]$ is denoted as *redundant basis* (RB) for $\mathbb{F}_{2^m}$ [44].

4.2.2 Redundant Basis Multiplication

Let field elements $A, B \in \mathbb{F}_{2^m}$ be represented with respect to (w.r.t.) the RB $I_1 = [1, \beta, \beta^2, \ldots, \beta^{n-1}]$ as

$$A = \sum_{i=0}^{n-1} a_i \beta^i \text{ and } B = \sum_{i=0}^{n-1} b_i \beta^i,$$

respectively, where $a_i, b_i \in \mathbb{F}_2, i = 0, 1, \cdots, n-1$. Then it follows $\beta^i \cdot B = \sum_{j=0}^{n-1} b_{(j-i)} \beta^j$, where $(j - i)$ denotes that $j - i$ is to be reduced modulo n. The product of field elements A and B can be given by [44]

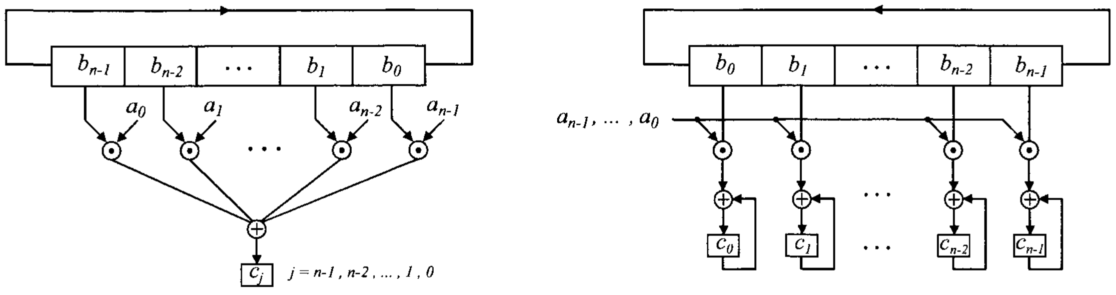
$$A \cdot B = \sum_{i=0}^{n-1} a_i (\beta^i \cdot B) = \sum_{j=0}^{n-1} (\sum_{i=0}^{n-1} a_i b_{(j-i)}) \beta^j.$$

If we define $A \cdot B = C \triangleq \sum_{j=0}^{n-1} c_j \beta^j$, then c_j can be given by

$$c_j = \sum_{i=0}^{n-1} a_i b_{(j-i)}, \quad j = 0, 1, \ldots, n-1. \tag{4.2}$$

4.2.3 An Overview of Bit-Serial RB Multipliers

At least two types of bit-serial RB multipliers have been proposed, as shown in Fig. 4.1[44]. One is SIPO type and the other is PISO type. The PISO multiplier uses fewer registers but has a longer critical path delay. A complexity comparison between these previous two multipliers and the proposed one will be made in the next section.



(a) PISO (b) SIPO

Figure 4.1: Previously proposed bit-serial RB multipliers

4.3 Proposed SIPO Multiplier

4.3.1 A New Bit-Serial RB Multiplication Algorithm

Let a_j and $b_j, j = 0, 1, \ldots, n-1$ be given as in Section 2. A new algorithm to compute RB multiplication can be shown as follows.

Algorithm 3. Bit-serial RB multiplication [MSB first]

Input: $a_j, b_j, j = 0, 1, \ldots, n-1$

Output: $c_j, j = 0, 1, \ldots, n-1$

```
1. Initialization: $c_j^{(0)} = 0$, for $j = 0, 1, \ldots, n-1$.
2. For $k = 1$ To $n$
3.         For $j = 0$ To $n-1$
4.
```

$$c_j^{(k)} = c_{(j-1)}^{(k-1)} + a_{(j)} b_{n-k}; \tag{4.3}$$

The final value $c_j^{(n)} = c_j$ *for* $j = 0, 1, \ldots, n-1$.

Proof of correctness of Algorithm 3:

It follows from (4.3),

$$\begin{aligned} c_j^{(n)} &= c_{(j-1)}^{(n-1)} + a_{(j)}b_0 \\ &= c_{(j-2)}^{(n-2)} + a_{(j-1)}b_1 + a_{(j)}b_0 \\ &= c_{(j-3)}^{(n-3)} + a_{(j-2)}b_2 + a_{(j-1)}b_1 + a_{(j)}b_0 \\ &= \cdots \\ &= c_{(j-n)}^{(0)} + a_{(j+1-n)}b_{n-1} + \cdots + a_{(j-1)}b_1 + a_{(j)}b_0 \\ &= c_{(j-n)}^{(0)} + \sum_{i=0}^{n-1} a_{(j-i)}b_i. \end{aligned} \tag{4.4}$$

Note that $c_{(j-n)}^{(0)} = 0$ from Step 1 of Algorithm 3, then we have

$$c_j^{(n)} = \sum_{i=0}^{n-1} a_{(j-i)}b_i = \sum_{i=0}^{n-1} a_i b_{(j-i)}. \tag{4.5}$$

The last step in (4.5) comes from proper substitution of subscripts. Compare (4.5) and (4.2), it follows $c_j^{(n)} = c_j$. □

4.3.2 Multiplier Architecture

An architecture to realize Algorithm 3 is proposed and shown in Fig. 4.2. Every bit of operand A should be available throughout the multiplication operation, while operand B is available in bit-serial fashion with the MSB first. The contents of the n-bit registers are initialized as zero. The registers are circularly connected and interleaved with XOR gates, where the XOR gates perform the addition operation expressed in (4.3). At each clock cycle, the registers cyclically shift and take on new values from the outputs of the XOR gates. At clock cycle k, the content of register R_j is $c_j^{(k)}$. It takes n clock cycles for the multiplier to finish one operation.

Table 4.1 shows the contents of the registers at the end of ith clock cycle, $i = 1, 2, \cdots, n$. At the end of multiplication, the registers $R_0, R_1, \cdots, R_{n-1}$, will respectively contain the

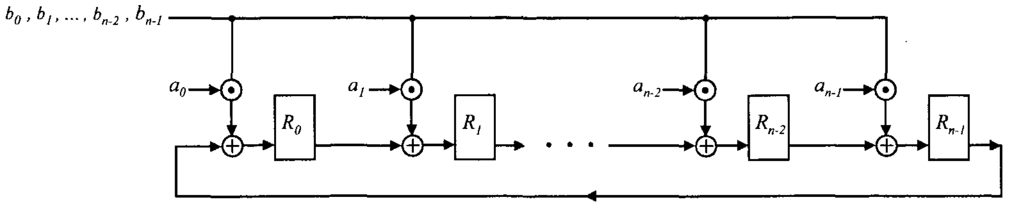


Figure 4.2: Proposed serial-in parallel-out RB multiplier (MSB first)

product coefficients $c_0, c_1, \cdots, c_{n-1}$.

Cyc. k / Reg. R_j	1	2	$\cdots$	k	$\cdots$	n
R_0	a_0b_{n-1}	$a_{n-1}b_{n-1} + a_0b_{n-2}$	$\cdots$	$a_0b_{(n-k)} + a_{n-1}b_{(n-k+1)} + \cdots + a_{(1-k)}b_{n-1}$	$\cdots$	$a_0b_0 + a_{n-1}b_1 + \cdots + a_1b_{n-1}$
R_1	a_1b_{n-1}	$a_0b_{n-1} + a_1b_{n-2}$	$\cdots$	$a_1b_{(n-k)} + a_0b_{(n-k+1)} + \cdots + a_{(2-k)}b_{n-1}$	$\cdots$	$a_1b_0 + a_0b_1 + \cdots + a_2b_{n-1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
R_{n-2}	$a_{n-2}b_{n-1}$	$a_{n-3}b_{n-1} + a_{n-2}b_{n-2}$	$\cdots$	$a_{n-2}b_{(n-k)} + a_{n-3}b_{(n-k+1)} + \cdots + a_{(n-1-k)}b_{n-1}$	$\cdots$	$a_{n-2}b_0 + a_{n-3}b_1 + \cdots + a_{n-1}b_{n-1}$
R_{n-1}	$a_{n-1}b_{n-1}$	$a_{n-2}b_{n-1} + a_{n-1}b_{n-2}$	$\cdots$	$a_{n-1}b_{(n-k)} + a_{n-2}b_{(n-k+1)} + \cdots + a_{(n-k)}b_{n-1}$	$\cdots$	$a_{n-1}b_0 + a_{n-2}b_1 + \cdots + a_0b_{n-1}$

Table 4.1: Register contents $c_j^{(k)}$ during a multiplication operation

A least-significant-bit (LSB) first version of the proposed multiplier is shown in Fig. 4.3, which has applications where the least significant bits of operand B are available before the other parts of the operand. Note that the proposed MSB first and LSB first versions have the same complexities and critical path delays.

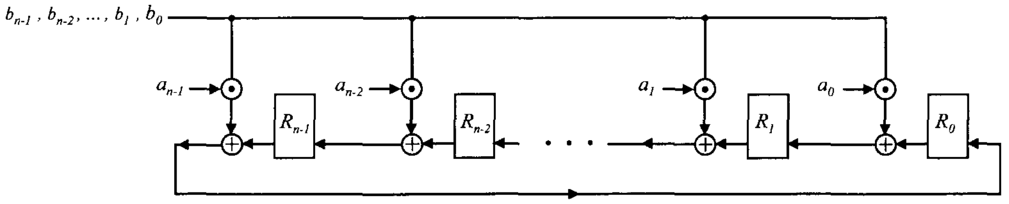


Figure 4.3: Proposed serial-in parallel-out RB multiplier (LSB first)

4.4 Complexity Comparison

4.4.1 Comparison to Other RB Multipliers

Compared to the two previous architectures [7, 44], the proposed one has a more regular structure. Moreover, the new multiplier has significantly lower complexity or smaller critical path delay compared to the previous RB multipliers, as shown in Table 4.2. The proposed architecture is shown in the row at the bottom of the table. Compared to the SIPO [44], the proposed multiplier requires only half number of registers while incurs about the same critical path delay. Compared to the PISO [44], the new architecture uses about the same number of gates and registers but has a significantly smaller critical path delay.

Architecture	# of AND	# of XOR	# of registers	Critical path delay
PISO [7, 44]	n	$n-1$	$n+1$	$T_A + \lceil \log_2 n \rceil T_X$
SIPO [44]	n	n	$2n$	$T_A + T_X$
Proposed SIPO	n	n	n	$T_A + T_X$

Table 4.2: Complexities comparison between bit-serial RB multipliers

For example, let $m = 268$ and we find that the minimal value of n is 269. Then the proposed RB multiplier requires 269 fewer flip-flops compared to the SIPO multiplier presented in [44] while they both have the same critical path delay. Compared to the PISO RB multiplier presented in [44] which has a critical path delay $T_A + \lceil \log_2 269 \rceil T_X = T_A + 9T_X$, the proposed multiplier has a much less critical path delay at $T_A + T_X$.

4.4.2 Comparison to Normal Basis Multipliers

A comparison of the proposed RB multiplier to other bit-serial NB multipliers when there exists a type I ONB is shown in Table 4.4.2. It can be seen that the proposed multiplier has the lowest complexity in terms of the number of XOR gates and registers. The proposed architecture has also the smallest critical path delay. Note that the new RB multiplier requires $m+1$ instead of m clock cycles to finish a multiplication operation.

Conversion between RB and NB is simple for certain classes of fields where both RB and NB are generated by the same Gauss Period [44]. For example, for the class of fields that there exists a type I ONB, RB elements (except the element '1') are a permutation of the NB elements. Let R and N be RB and NB for the class of fields that there exists a type I ONB, respectively. Then

$$R = \{r_i\}_{i=0}^{n-1} = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\}, \text{ and } N = \{n_j\}_{j=0}^{m-1} = \{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\},$$

where β is a n^{th} primitive root of unity or Gauss Period of type $(m, 1)$ and $n = m + 1$. It follows by noting $\beta^n = 1$

$$n_j = \beta^{2^j} = \beta^{2^j \bmod n} = \beta^{(2^j)} = r_{(2^j)}.$$

Since 2 is a generator of the multiplicative group $\mathbb{Z}_n^{\times}$, there always exists a unique $i \in \{1, 2, \ldots, n-1\}$ for a given j such that $i = (2^j)$.

Multiplier	# AND	# XOR	# Registers	Critical Path Delay	# Cyc.	Basis
Massey-Omura [22]	$2m-1$	$2m-2$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	m	NB
IMO [11]	m	$2m-2$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	m	NB
Beth-Golmann [4]	m	$2m-1$	$2m$	$T_A + 2T_X$	m	NB
Geiselmann-Gollmann [15]	m	$m + \lfloor \frac{m}{2} \rfloor$	$3m$	$T_A + 3T_X$	m	NB
Feng [9]	$2m-1$	$3m-2$	$3m-2$	$T_A + 4T_X$	m	NB
Agnew [37]	m	$2m-1$	$3m$	$T_A + 2T_X$	m	NB
b-SMPOI [36]	$\lfloor \frac{m}{2} \rfloor + 1$	$2m-1$	$3m$	$T_A + 3T_X$	m	NB
b-SMPOII [36]	m	$m + \lfloor \frac{m}{2} \rfloor$	$3m$	$T_A + 3T_X$	m	NB
Proposed	$m+1$	$m+1$	$m+1$	$T_A + T_X$	$m+1$	RB

Table 4.3: Complexities comparison between the proposed RB multiplier and bit-serial NB multipliers when there exists a type I optimal normal basis.

4.5 Proposed Digit-Level SIPO Multiplier

4.5.1 A New Digit-Level RB Multiplication Algorithm

For given j, $c_j^{(k)}$ accumulates one term $a_j b_{(n-k)}$ during clock cycle k according to (4.3) in Algorithm 3. Certain parallelism can be introduced to Algorithm 3 so that the bit-serial RB multiplication can be speed up. The proposed digit-level SIPO algorithm facilitates that $c_j^{(k)}$ accumulates w terms during clock cycle k and thus the multiplication can be completed in $\lceil n/w \rceil$ clock cycles for positive integer w and $1 \leq w \leq n$.

Algorithm 4. Digit-level RB multiplication

```
Input:    $a_j, b_j, j = 0, 1, \ldots, n-1$
          Let $w$ be an integer such that $1 \leq w \leq n$, also let $b_j = 0$ for $j = -1, ..., n - \lceil \frac{n}{w} \rceil w$.
Output:   $c_j, j = 0, 1, \ldots, n-1$

1. Initialization: $c_j^{(0)} = 0$, for $j = 0, 1, \ldots, n-1$.
2. For $k = 1$ To $\lceil n/w \rceil$
3.     For $j = 0$ To $n-1$
4.
```

$$c_{(j+\lceil n/w \rceil)}^{(k)} = c_{(j+\lceil n/w \rceil-1)}^{(k-1)} + \sum_{i=0}^{w-1} a_{(j+\lceil n/w \rceil + i\lceil n/w \rceil)} b_{n-k-i\lceil n/w \rceil}; \tag{4.6}$$

The final value $c_{(j+\lceil n/w \rceil)}^{(\lceil n/w \rceil)} = c_j$ *for* $j = 0, 1, \ldots, n-1$.

Proof of correctness of Algorithm 4:

It follows from (4.6) that

$$\begin{aligned}
c^{(k)}_{(j+\lceil n/w\rceil)}\big|_{k=\lceil n/w\rceil} &= c^{(\lceil n/w\rceil-1)}_{(j+\lceil n/w\rceil-1)} + \sum_{i=0}^{w-1} a_{(j+\lceil n/w\rceil+i\lceil n/w\rceil)} b_{n-\lceil n/w\rceil-i\lceil n/w\rceil} \\
&= c^{(\lceil n/w\rceil-2)}_{(j+\lceil n/w\rceil-2)} + \sum_{i=0}^{w-1} a_{(j+\lceil n/w\rceil+i\lceil n/w\rceil)} b_{n-\lceil n/w\rceil-i\lceil n/w\rceil} \\
&\quad + \sum_{i=0}^{w-1} a_{(j+(\lceil n/w\rceil-1)+i\lceil n/w\rceil)} b_{n-(\lceil n/w\rceil-1)-i\lceil n/w\rceil} \\
&= \cdots \\
&= c^{(0)}_{(j)} + \sum_{i=0}^{w-1} \sum_{\ell=0}^{\lceil n/w\rceil-1} a_{(j+i\lceil n/w\rceil+\ell+1)} b_{n-i\lceil n/w\rceil-\ell-1} \\
&= c^{(0)}_{j} + \sum_{i_1=0}^{n-1} a_{(j+i_1+1)} b_{n-i_1-1} &(4.7)\\
&= c^{(0)}_{j} + \sum_{i=0}^{n-1} a_{(j-i)} b_i &(4.8)\\
&= c_j\,. &(4.9)
\end{aligned}$$

Equation (4.7) uses substitution of $i_1 = i\lceil n/w\rceil + \ell$ and equation (4.8) uses substitution of $i = n - i_1 - 1$. The final step (4.9) comes from Step 1 of Algorithm 4 and equation (4.2). $\square$

4.5.2 Proposed Digit-Level RB Multiplier

Fig. 4.4 shows a hybrid SIPO architecture of RB multiplication when $w = 2$. In this case, the input operand B is divided into two parts, each of size $\lceil n/2\rceil$ bits. All the registers are initialized to zero. At the end of clock cycle k the contents of R_js are $c^{(k)}_j$. It takes $\lceil n/2\rceil$ clock cycles for the multiplier to complete one operation. At the end of $\lceil n/2\rceil$ clock cycles, the product coefficients $c_0, c_1, \ldots, c_{n-1}$ reside in Registers $R_{(\lceil n/2\rceil)}, R_{(\lceil n/2\rceil+1)}, \ldots, R_{(\lceil n/2\rceil+n-1)}$, respectively.

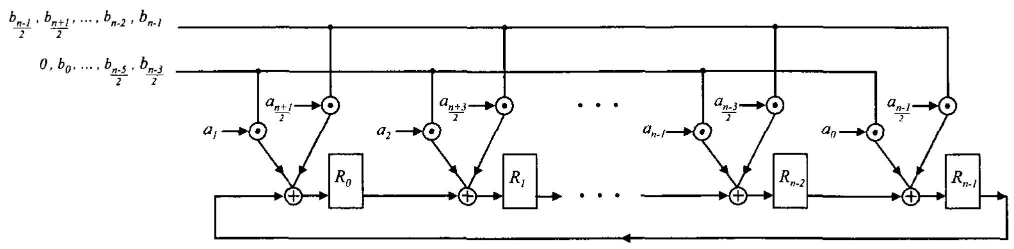


Figure 4.4: Proposed hybrid SIPO RB multiplier (LSB first)

4.5.3 Complexity Comparison to Previous RB Multipliers

Table 4.4 shows the complexity comparison for our proposed digit-level multiplier with the other existing digit-level multiplier proposed in [44], in terms of number of gates and latches as well as the critical path delay. First row of this table shows the hybrid Parallel-In Serial-Out multiplier (Hybrid PISO) containing $\lceil \frac{n}{w} \rceil$ parallel modules which was proposed in [44], and the second row presents our proposal. It is clear that our proposal has always a smaller critical path delay than the previous architecture. For small values of w the proposed architecture requires even less number of gates than the hybrid PISO, but for the large values of w the hybrid PISO has a lower gate complexity.

Multiplier	# AND	# XOR	# Reg.	Critical Path Delay	# Clock Cyc.
Hybrid PISO [44]	$n\lceil \frac{n}{w} \rceil$	$n\lceil \frac{n}{w} \rceil$	$n + \lceil \frac{n}{w} \rceil$	$T_A + (\lceil \log_2 n \rceil)T_X$	$\lceil \frac{n}{w} \rceil$
proposed	wn	$wn + w + 1$	n	$T_A + (\lceil \log_2 w + 1 \rceil)T_X$	$\lceil \frac{n}{w} \rceil$

Table 4.4: Complexities Comparison Between Digit-Level RB Multipliers

4.5.4 Complexity Comparison to Previous NB Multipliers

A complexity comparison of the proposed digit-level RB multiplier to some popular NB multipliers for a class of fields that there exists a type I ONB is given in Table 4.5. It can be seen that the proposed architecture uses the fewest registers and has the smallest critical path delay. The number of XOR and AND gates used for the new RB multiplier is also comparable to the lowest number of gates required by the NB multipliers. Note that the proposed multiplier may need one more clock cycle to complete one multiplication than the NB multipliers listed in Table 4.5 when m is a multiple of w.

Multiplier	# AND	# XOR	# Reg.	Critical Path Delay	# Cyc.	Basis
WLMO [22]	$w(2m-1)$	$w(2m-2)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	$\lceil m/w \rceil$	NB
IMO [11]	wm	$w(2m-2)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	$\lceil m/w \rceil$	NB
AEDS [36]	$(w+1)\frac{m}{2}$	$(w+1)(\frac{3}{2}m-2)+1$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	$\lceil m/w \rceil$	NB
XEDS [36]	$w(m-1)+m$	$(w+1)(m-1)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	$\lceil m/w \rceil$	NB
w-SMPOI [37]	$\frac{wm}{2}+m+w+1$	$\frac{3wm}{2}+m+w-1$	$3m$	$2T_A + (3 + \lceil \log_2(w-1) \rceil)T_X$	$\lceil m/w \rceil$	NB
w-SMPOII [37]	$wm+m+w+1$	$wm+m+w-1$	$3m$	$2T_A + (3 + \lceil \log_2(w-1) \rceil)T_X$	$\lceil m/w \rceil$	NB
proposed	$wm+w$	$wm+m+w+1$	$m+1$	$T_A + (\lceil \log_2 w + 1 \rceil)T_X$	$\lceil (m+1)/w \rceil$	RB

Table 4.5: Complexities comparison between digit-level architectures: the proposed RB multiplier versus some NB multipliers for a class of fields that there exists a type I ONB

4.6 Conclusions

A new SIPO finite field multiplier using redundant representation has been proposed. It has been shown that the multiplier compares favorably to the previous proposals in terms of complexity or critical path delay. For a class of fields that there exists a type I ONB, the proposed multiplier has significantly lower complexity compared to previously proposed NB multipliers. Digit-level version of the new multiplier has also been presented and its complexities compare favorably to other type I ONB multipliers. The new architecture accommodates inputs of MSB first and LSB first fashions. It is expected that the proposed multiplier has application in elliptic curve and ElGamal cryptography.

Chapter 5

A High Speed Word Level Multiplier in $\mathbb{F}_{2^m}$ Using Redundant Representation

5.1 Introduction

Hardware implementation of finite field multipliers usually can be categorized into three categories. The First category are bit level multipliers [22],[1],[15],[11]. A bit level multiplier takes m clock cycle to finish one multiplication in a binary field of size m. The second category are full parallel multipliers [35],[24],[20],[43]. A full parallel multiplier takes one clock cycle to finish one field multiplication.

The third category are word level or digit level finite field multipliers which are the most commonly implemented in practice [12],[44],[22],[32],[31],[36],[37]. A word level multiplier takes w clock cycles, $1 \leqslant w \leqslant m$, to finish one multiplication operation in $\mathbb{F}_{2^m}$. The value of w can be selected by designer to set the trade off between area and speed according to the application. Decreasing the value of w will result in faster and larger multipliers while increasing w will make smaller and slower multipliers. Note that bit level and full parallel multipliers can be viewed as special cases of word level multipliers for $w = m$ and $w = 1$ respectively.

In this chapter, a new word level finite field multiplier in $\mathbb{F}_{2^m}$ using redundant representation is proposed. For the class of fields that there exists a type I ONB, we show that the new architecture is much faster compared to previously proposed word level architectures using either NB or redundant representation. It is also shown that for the class of fields the new multiplier out-performs all the other multipliers when considering the product of area and delay as a measure of performance. One of the unique features of the proposed word level multiplier is that the critical path delay is not a function of the field size nor the word size. This enables the architecture to operate at very high speed even for large field sizes or large word sizes.

The organization of this chapter is as follows: Section 2 is a brief review of redundant representation and multiplication. In sections 3 and 4, a new word level algorithm and architecture for multiplication in redundant representation is proposed, respectively. The architectural complexities of the proposed multiplier are compared to other similar previous proposals in section 5. A few concluding remarks are given in section 6.

5.2 A Brief Review of Redundant Representation and Its Arithmetic in $\mathbb{F}_{2^m}$

5.2.1 Redundant Basis for $\mathbb{F}_{2^m}$

Let K be a field and $f(x) \in K[x]$ be a polynomial defined over K. Then the field that contains all the roots of $f(x)$ is called the *splitting field* of the polynomial $f(x)$. The splitting field of $x^n - 1$ is called the n^{th} cyclotomic field, denoted by $K^{(n)}$ [44]. Let β be a primitive n^{th} root of unity. Then $K^{(n)}$ is generated by β over K and elements in K can be represented in the form

$$A = a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1},\ a_i \in K.$$

Thus the set $[1, \beta, \beta^2, \ldots, \beta^{n-1}]$ acts as a basis for $K^{(n)}$. Since $1+\beta+\beta^2+\cdots+\beta^{n-1} = 0$ the representation of A is not unique. So, by sightly abusing the terminology, we call the set $[1, \beta, \beta^2, \ldots, \beta^{n-1}]$ *redundant basis* (RB) for any subfield of $K^{(n)}$.

We are particularly interested in the following case: Let K be the binary field $\mathbb{F}_2$ and $K^{(n)}$ be a cyclotomic field that $\mathbb{F}_{2^m}$ can be embedded in. The following theorem characterizes the relationship between m and n.

Theorem 5.2.1. [25] Let n be an odd positive integer. Then, $\mathbb{F}_{2^m}$ is contained in $\mathbb{F}_2^{(n)}$ if and only if m divides the multiplicative order of 2 mod n.

Remark 5.2.2. If there is a type I Optimal Normal Basis in $\mathbb{F}_2^{(m)}$, then $\mathbb{F}_2^{(m)}$ is contained in $\mathbb{F}_2^{(m+1)}$, so there is a RB of size $m+1$ for $\mathbb{F}_2^{(m)}$.

5.2.2 Redundant Basis Multiplication in $\mathbb{F}_{2^m}$

Consider the RB for $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$:

$$I = [1, \beta, \beta^2, \ldots, \beta^{n-1}].$$

Let field elements $A, B \in \mathbb{F}_{2^m}$ to be represented with respect to I :

$$A = \sum_{i=0}^{n-1} a_i\beta^i, \quad B = \sum_{j=0}^{n-1} b_j\beta^j,$$

where $a_i, b_j \in \mathbb{F}_2, i, j = 0, 1, \ldots, n-1$. Note that $\beta^n = 1$. Then multiplication of B by β^i using the RB I can be given by

$$\beta^i \cdot B = \sum_{j=0}^{n-1} b_j\beta^{i+j} = \sum_{j=0}^{n-1} b_{(j-i)}\beta^j,$$

where $(j-i)$ denotes that $j-i$ is to be reduced modulo n. Then the product of field elements A and B can be given by

$$A\cdot B=\sum_{i=0}^{n-1}a_i(\beta^i\cdot B)=\sum_{j=0}^{n-1}\Big(\sum_{i=0}^{n-1}a_ib_{(j-i)}\Big)\beta^j. \tag{5.1}$$

If we define $A\cdot B=C\triangleq\sum_{j=0}^{n-1}c_j\beta^j$, then c_j can be given by

$$c_j=\sum_{i=0}^{n-1}a_ib_{(j-i)},\quad j=0,1,\ldots,n-1. \tag{5.2}$$

5.3 Proposed Word Level Multiplication In RB

From (5.2) it can be seen that one product bit c_j is a sum of n partial product bits $a_ib_{(j-i)}$. If each partial product bit is to be calculated at one clock cycle, the multiplier will take n clock cycles to finish one multiplication operation which is the case for the bit level architecture [44].

Let w denote the word size. Then the operand A in RB can be represented in $k=\lceil n/w\rceil$ words:

$$A=\underbrace{a_0a_1\ldots a_{w-1}}_{A_0}\underbrace{a_w\ldots a_{2w-1}}_{A_1}a_{2w}\ldots\underbrace{a_{(k-1)w}\ldots a_{n-1}0\ldots0}_{A_{k-1}}.$$

Or $A=\sum_{h=0}^{k-1}A_h\beta^{hw}$, where $A_h=\sum_{\ell=0}^{w-1}a_{hw+\ell}\beta^\ell$. Note that $a_i=0$ if the subscript i is greater than $n-1$. Replace i in (5.2) with $hw+\ell$:

$$c_j=\sum_{\ell=0}^{w-1}\sum_{h=0}^{k-1}a_{hw+\ell}b_{(j-hw-\ell)},\quad j=0,1,\ldots,n-1. \tag{5.3}$$

Define new signal $d_{h,j}^{\ell}$ as follows

$$\begin{cases} d_{h,j}^{(-1)} &= 0 \text{ and}\\ d_{h,j}^{(\ell)} &= d_{h,j}^{(\ell-1)}+a_{hw+\ell}b_{(j-hw-\ell)} \text{ for } \ell=0,1,\ldots,w-1.\end{cases} \tag{5.4}$$

Then it follows from (5.4)

$$d_{h,j}^{(w-1)} = \sum_{\ell=0}^{w-1} a_{hw+\ell} b_{(j-hw-\ell)}. \tag{5.5}$$

Compare (5.3) with (5.5), it follows

$$c_j = \sum_{h=0}^{k-1} d_{h,j}^{(w-1)}. \tag{5.6}$$

An algorithmic form for word level multiplication using RB can be given as follows.

Algorithm 5. Word level RB multiplication algorithm

Input: $A = (A_0, \ldots, A_{k-1})$, $B = (B_0, \ldots, B_{k-1})$, both w.r.t. RB.

Output: $C = A \times B = (c_0, \ldots, c_{n-1})$ also w.r.t. RB

```
1. Initialization: $k = \lceil m/w \rceil$, and $d_{h,j}^{(-1)} = 0$ for $h = 0, 1, 2, \ldots, k-1$ and $j = 0, 1, \ldots, n-1$
2. For all values of $j = 0, 1, 2, \ldots, n-1$, compute
3.         For all values of $h = 0, 1, \ldots, k-1$, compute
4.                 For $\ell = 0$ To $w-1$
5.                         $d_{h,j}^{(\ell)} = d_{h,j}^{(\ell-1)} + a_{hw+\ell}[b_{(j-hw-\ell)}]$
6.                 End For
7.         $c_j = \sum_{h=0}^{k-1} d_{h,j}^{(w-1)}$
8.         End For
9. End For
```

5.4 Proposed Word Level Multiplier Architecture in RB

5.4.1 Multiplier Architecture

Based on the algorithm proposed in the last section, a new word level architecture for RB multiplication is presented and shown in Fig. 5.1. The architecture includes, from top to bottom, one n-bit circular shift register R, a permutation/expansion module with n-bit input and kn-bit output, a layer of kn AND gates, a layer of kn XOR gates, kn flip-flops, and n k-input binary tree networks of XOR gates.

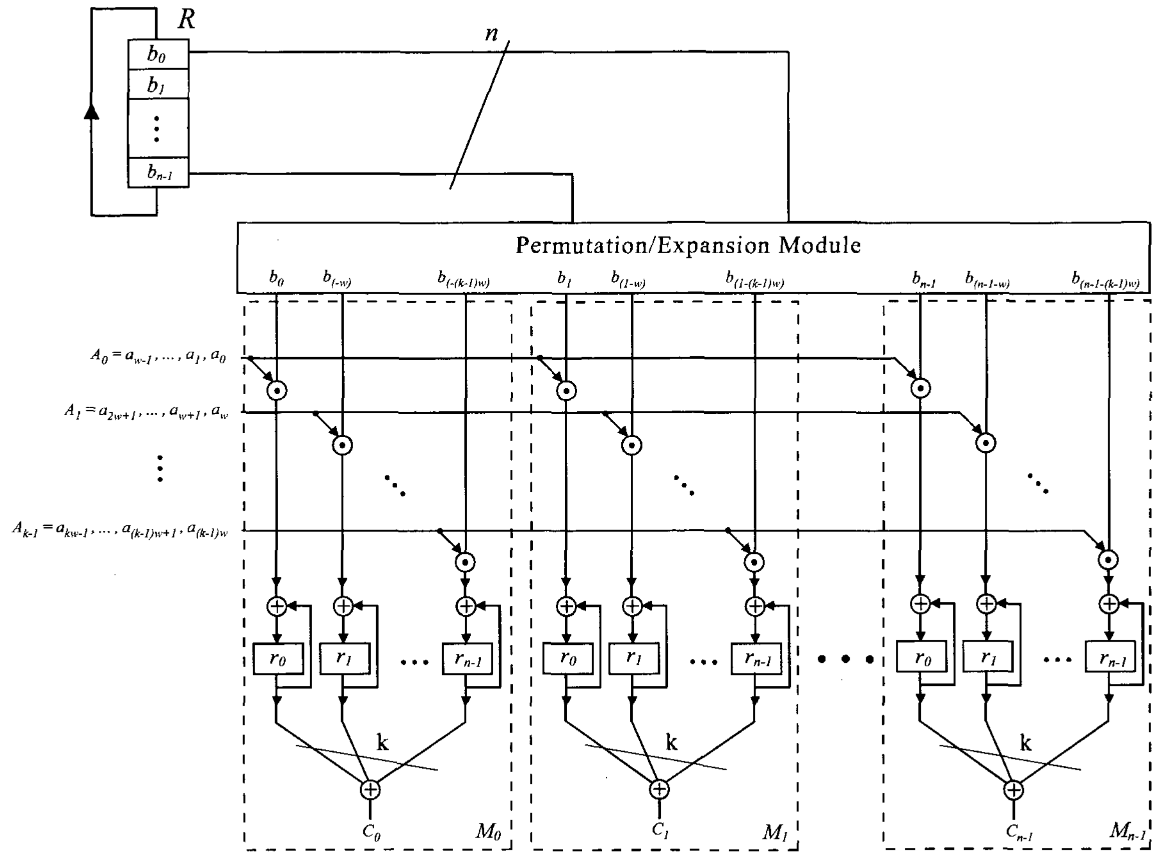

R
b_0
b_1
b_{n-1}
n
Permutation/Expansion Module
b_0 $b_{(-w)}$ $b_{(-(k-1)w)}$ b_1 $b_{(1-w)}$ $b_{(1-(k-1)w)}$ b_{n-1} $b_{(n-1-w)}$ $b_{(n-1-(k-1)w)}$
$A_0 = a_{w-1}, \dots, a_1, a_0$
$A_1 = a_{2w+1}, \dots, a_{w+1}, a_w$
$A_{k-1} = a_{kw-1}, \dots, a_{(k-1)w+1}, a_{(k-1)w}$
r_0 r_1 r_{n-1}
k
c_0 M_0
c_1 M_1
c_{n-1} M_{n-1}

Figure 5.1: Proposed high speed word level multiplier

Note that the layer of kn AND gates, the layer of kn XOR gates and the kn flip-flops form kn accumulation units in parallel, which are responsible for the accumulation operation in Step 5 of Algorithm 5. These kn accumulation units are divided into n groups with each group containing k units. The parallel structure of n groups corresponds Steps 2 and 9 and the structure of k units in each group corresponds to Steps 3 and 8 in Algorithm 5. The iteration shown in Steps 4 and 6 of the algorithm requires the architecture to take w clock cycle to complete one multiplication operation.

At first, let us look at the part of the circuit (framed by the dashed lines) that generates c_0, which is denoted by M_0. There are k accumulation units and they are numbered from left to right as $h, h = 0, 1, \ldots, k-1$. The contents of flip-flop h during clock cycle ℓ are denoted as $d_{h,0}^{(\ell)}$. Before clock cycle zero all the k flip-flops are initialized as zero, $d_{h,0}^{(-1)} = 0, h = 0, 1, \ldots, k-1$. During clock cycle ℓ, accumulation unit h performs $d_{h,0}^{(\ell)} = d_{h,0}^{(\ell-1)} + a_{hw+\ell}b_{(-hw-\ell)}$ and the content of the flip-flop is $d_{h,0}^{(\ell)}$. During clock $w-1$, flip-flop h contains $d_{h,0}^{(w-1)}$. The k-input XOR gate is used to generate the output $c_0 = \sum_{h=0}^{k-1} d_{h,0}^{(w-1)}$.

Now consider accumulation unit h at M_j. During clock cycles $0, 1, \ldots, w-1$, the accumulation unit takes inputs $a_{hw}, a_{hw+1}, \ldots, a_{hw+w-1}$ from A, and $b_{(j-hw)}, b_{(j-hw-1)}, \ldots$ $, b_{(j-hw-w+1))}$ from B, respectively. For example, during clock cycle ℓ, accumulation unit h takes inputs $a_{hw+\ell}$ and $b_{(j-hw-\ell)}$ and performs operation $d_{h,j}^{(\ell)} = d_{h,j}^{(\ell-1)} + a_{hw+\ell}b_{(j-hw-\ell)}$. From the above discussion we can observe the following:

Fact 5.4.1. For any accumulation unit, let the inputs be a_{i_1} from A and b_{j_1} from B during clock cycle ℓ, then the input is a_{i_1+1} from A and $b_{(j_1-1)}$ from B during clock cycle $\ell+1$.

Also note that at clock cycle $\ell = 0$ the input bit from A to accumulation unit h is $a_{hw}, h = 0, 1, \ldots, k-1$, which is the least significant bit of the word A_h from A. Then it can be seen from Fact 5.4.1 that inputs from operand A can be organized into k words and word h inputs to accumulation unit h in a bit serial fashion with the least significant bit first. Inputs from operand B can be organized as an n-bit circular shift register and a permutation/expansion module as shown in Fig. 5.1. The shifting direction of R should conform with Fact 5.4.1 and the permutation/expansion module can be explained as follows: during clock cycle 0, the output of the P/E module that inputs to accumulation unit h in M_j should be connect to $b_{(j-hw)}$ in R.

Let the critical path of the architecture be denoted by T_{cp}. Then T_{cp} is decided by the accumulation unit, which is $T_{cp} = T_A + T_X$, where T_A and T_X denote the time delays caused by one AND gate and one XOR gate, respectively. Note that after w clock cycles the product bits are not generated until it takes another time delay of $\lceil \log_2 k \rceil T_X$ caused by a k-input XOR gate. In order to let the architecture outputs c_j hold stable for a desired time, two options are available for the multipliers to act during the time spent for the final addition of the partial products. In the first option, the multiplier clock should be stopped after w clock cycles and the outputs can be read from the output ports after the addition delay which can be measured approximately beforehand. This style requires an extra counter to disable the input clock to the flip flops after w clock cycles. The advantage of this method is that it can save dynamic power consumption during the final addition.

The second option is to pad zero input bits for the input words for operand A after the w clock cycles. This will eliminate the need for stopping the clock signal of the multiplier and does not require extra circuitry. The output product bits can be read from the output ports after certain number of clock cycles immediately following w cycles. Assuming that the clock period is chosen to be the critical path delay $T_{cp} = T_A + T_X$, and let the number of extra clock cycles besides w be denoted by w_{ex}. Then the total number of the clock cycles required for a multiplication operation is $w + w_{ex}$, where w_{ex} can be obtained as follows

$$w_{ex} = \left\lceil \frac{\lceil \log_2 k \rceil T_X}{T_{cp}} \right\rceil = \left\lceil \frac{\lceil \log_2 k \rceil T_X}{T_A + T_X} \right\rceil. \tag{5.7}$$

Note that w_{ex} zero bits need to be appended at the end of each input word from A.

5.4.2 Architecture Complexities

The area and delay complexity of the proposed design can be easily determined from Fig. 5.1. The circular shift register R contains n flip-flops. The P/E module is nothing but a rewiring of lines. There are n modules $M_j, j = 0, 1, \ldots, n-1$, where each module contains k AND gates, k XOR gates, k flip-flops, and a k-input XOR gate that is equivalent to $k-1$ (two-input) XOR gates. In total the proposed architecture requires kn AND gates, $(2k-1)n$ XOR gates, and $(k+1)n$ flip-flops.

5.4.3 An example

An example of the proposed word level RB multiplier for $\mathbb{F}_{2^4}$ is shown in Fig. 5.2. Note $n = m + 1 = 5$ since there exists a type I ONB in $\mathbb{F}_{2^4}$. Let the two operands be given as $A = (a_4, a_3, a_2, a_1, a_0)$ and $B = (b_4, b_3, b_2, b_1, b_0)$. Choose $w = 3$ and the operand A is divided into two words $A_0 = (a_2, a_1, a_0)$ and $A_1 = (0, a_4, a_3)$.

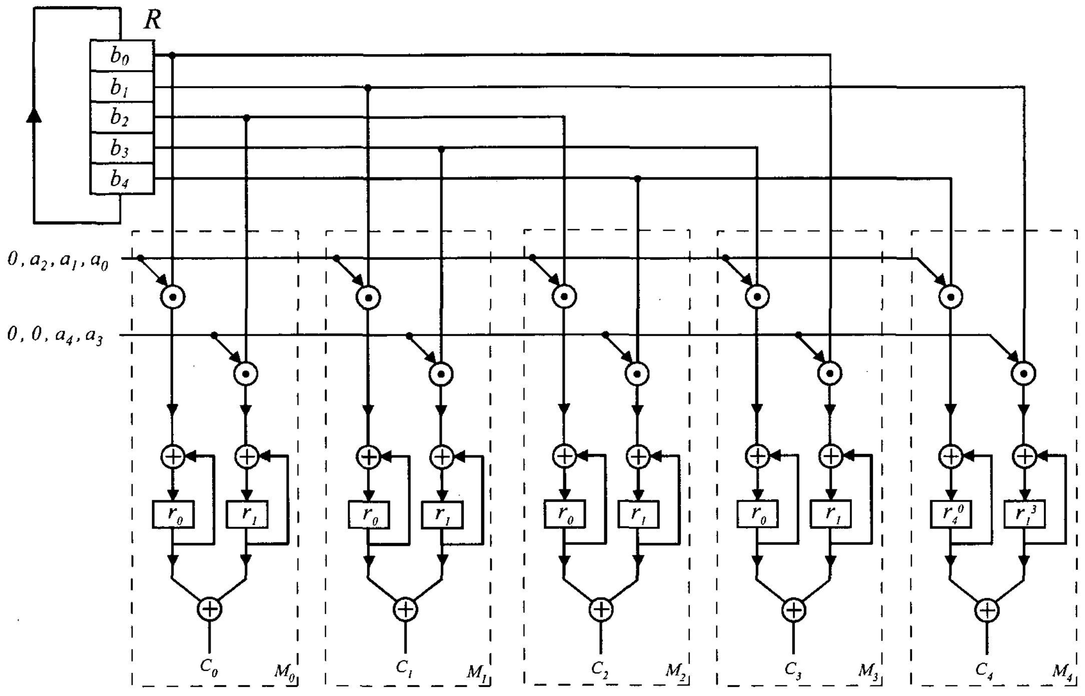


Figure 5.2: Word level ($w = 3$) multiplier in $\mathbb{F}_{2^4}$ with the padded zero bits for the input (the second option)

The critical path delay is $T_{cp} = T_A + T_X$ and it takes $w + w_{ex} = 3 + w_{ex}$ clock cycles to complete one multiplication, where w_{ex} can be obtained from (5.7) by noting $k = 2$

$$w_{ex} = \lceil T_X / T_{cp} \rceil = \lceil T_X / (T_A + T_X) \rceil = 1.$$

Note that $w_{ex} = 1$ zero bit should be appended at the end of each input word A_0 and A_1.

The proposed architecture contains $n = 5$ module M and each module M contains $k =$

Table 5.1: Contents of the flip-flops in the proposed multiplier in $\mathbb{F}_{2^4}$

Cyc	Content $d_{h,j}^{(\ell)}$ of flip-flop h in module M_j									
ℓ	$d_{0,0}^{(\ell)}$	$d_{1,0}^{(\ell)}$	$d_{0,1}^{(\ell)}$	$d_{1,1}^{(\ell)}$	$d_{0,2}^{(\ell)}$	$d_{1,2}^{(\ell)}$	$d_{0,3}^{(\ell)}$	$d_{1,3}^{(\ell)}$	$d_{0,4}^{(\ell)}$	$d_{1,4}^{(\ell)}$
-1	0	0	0	0	0	0	0	0	0	0
0	a_0b_0	a_3b_2	a_0b_1	a_3b_3	a_0b_2	a_3b_4	a_0b_3	a_3b_0	a_0b_4	a_3b_1
1	a_0b_0 $+a_1b_4$	a_3b_2 $+a_4b_1$	a_0b_1 $+a_1b_0$	a_3b_3 $+a_4b_2$	a_0b_2 $+a_1b_1$	a_3b_4 $+a_4b_3$	a_0b_3 $+a_1b_2$	a_3b_0 $+a_4b_4$	a_0b_4 $+a_1b_3$	a_3b_1 $+a_4b_0$
2	a_0b_0 $+a_1b_4$ $+a_2b_4$	a_3b_2 $+a_4b_1$	a_0b_1 $+a_1b_0$ $+a_2b_4$	a_3b_3 $+a_4b_2$	a_0b_2 $+a_1b_1$ $+a_2b_0$	a_3b_4 $+a_4b_3$	a_0b_3 $+a_1b_2$ $+a_2b_1$	a_3b_0 $+a_4b_4$	a_0b_4 $+a_1b_3$ $+a_2b_2$	a_3b_1 $+a_4b_0$
3	a_0b_0 $+a_1b_4$ $+a_2b_4$	a_3b_2 $+a_4b_1$	a_0b_1 $+a_1b_0$ $+a_2b_4$	a_3b_3 $+a_4b_2$	a_0b_2 $+a_1b_1$ $+a_2b_0$	a_3b_4 $+a_4b_3$	a_0b_3 $+a_1b_2$ $+a_2b_1$	a_3b_0 $+a_4b_4$	a_0b_4 $+a_1b_3$ $+a_2b_2$	a_3b_1 $+a_4b_0$

$\lceil n/w \rceil = 2$ accumulation units. Let the content of the flip-flop r_h in M_j during clock cycle ℓ be $d_{h,j}^{(\ell)}$. During clock cycle -1, all the flip-flops are initialized as zero. During clock cycle ℓ, it follows from (5.4) that the flip-flop r_h in M_j contains $d_{h,j}^{(\ell)} = d_{h,j}^{(\ell-1)} + a_{hw+\ell}b_{(j-hw-\ell)}$. Table 5.1 shows $d_{h,j}^{(\ell)}$ for $\ell = 0, 1, 2, 3$. Note that the flip-flop contents do not change during the last cycle due to the fact that the input from A are zero bits.

If cycle $\ell = 0$ is counted as the first clock cycle, then the final product can be read out during cycle $\ell = 3$. During clock cycle $\ell = 2$, the flip-flops contain $d_{h,j}^{(w-1)} = d_{h,j}^{(2)}$. As soon as the contents of the flip-flops are updated as $d_{h,j}^{(2)}$, the XOR network at the output end performs summation operation to obtain the product bits following (5.6)

$$\begin{aligned} c_0 &= d_{0,0}^{(2)} + d_{1,0}^{(2)}, \\ c_1 &= d_{0,1}^{(2)} + d_{1,1}^{(2)}, \\ c_2 &= d_{0,2}^{(2)} + d_{1,2}^{(2)}, \\ c_3 &= d_{0,3}^{(2)} + d_{1,3}^{(2)}, \\ c_4 &= d_{0,4}^{(2)} + d_{1,4}^{(2)}. \end{aligned} \tag{5.8}$$

The time delay caused by (5.8) is T_X and less than one clock cycle, so the total multiplication delay for the multiplier is four clock cycles. The multiplier requires $kn = 10$ AND gates and $(2k-1)n = 15$ XOR gates, and $(k+1)n = 15$ bit flip-flops. Note that the contents of flip-flops during cycle 2 are shown in Table 5.1, the final output product bits can be given following (5.8),

$$\begin{aligned}
c_0 &= a_0b_0 + a_1b_4 + a_2b_3 + a_3b_2 + a_4b_1,\\
c_1 &= a_0b_1 + a_1b_0 + a_2b_4 + a_3b_3 + a_4b_2,\\
c_2 &= a_0b_2 + a_1b_1 + a_2b_0 + a_3b_4 + a_4b_3,\\
c_3 &= a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + a_4b_4,\\
c_4 &= a_0b_4 + a_1b_3 + a_2b_2 + a_3b_1 + a_4b_0.
\end{aligned}$$

5.4.4 Word Level Architecture with MSB First

A most significant bit (MSB) first version of the multiplier architecture is also presented and shown in Fig. 5.3, where the MSB of each word from the operand A inputs to the system first. The k words each of w bits from A can be obtained as follows.

$$A = \underbrace{0\ldots 0a_0\ldots a_{n-(k-1)w-1}}_{A_0}\cdots a_{n-2w-1}\underbrace{a_{n-2w}\cdots a_{n-w-1}}_{A_{k-2}}\underbrace{a_{n-w}\cdots a_{n-1}}_{A_{k-1}}$$

Note that the shift direction of the circular shift register R is different from that in Fig. 5.1. The MSB version of the architecture has the same complexities and time delay as the LSB version (Fig. 5.1).

5.5 Complexity Comparison

5.5.1 Comparison to Other Word Level RB Multipliers

Complexity of the proposed multiplier and similar word level redundant basis multipliers compared in table 5.2. It can be seen from the table that the proposed design has the

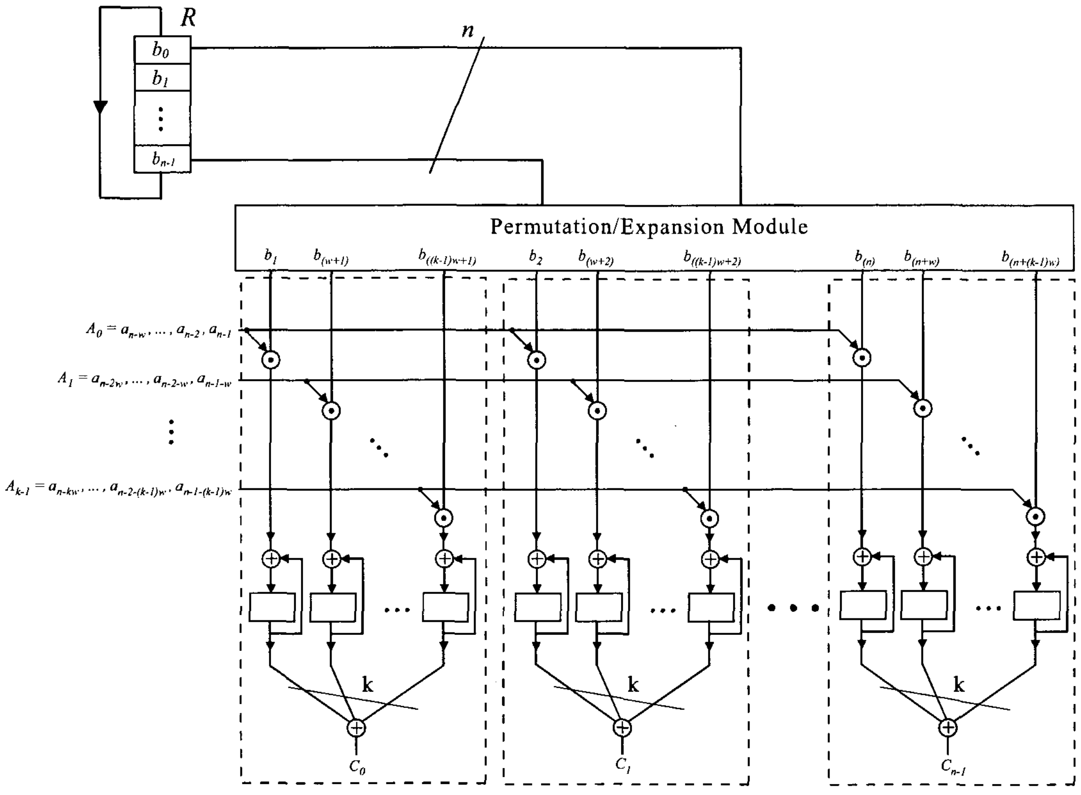

R
b_0
b_1
b_{n-1}
n
Permutation/Expansion Module
b_1
$b_{(w+1)}$
$b_{((k-1)w+1)}$
b_2
$b_{(w+2)}$
$b_{((k-1)w+2)}$
$b_{(n)}$
$b_{(n+w)}$
$b_{(n+(k-1)w)}$
$A_0 = a_{n-w}, \dots, a_{n-2}, a_{n-1}$
$A_1 = a_{n-2w}, \dots, a_{n-2-w}, a_{n-1-w}$
$A_{k-1} = a_{n-kw}, \dots, a_{n-2-(k-1)w}, a_{n-1-(k-1)w}$
k
C_0
C_1
C_{n-1}

Figure 5.3: Proposed high speed word level multiplier (MSB First)

smallest critical path delay and multiplication delay between all similar proposals. As can be seen from the table, the critical path delay of the proposed architecture is not a function of the word size or the filed size and is always equal to $T_A + T_X$.

Table 5.2: Complexity comparison for word level redundant basis multipliers

Multiplier	# AND	# XOR	# Reg	Critical Path Delay (T_{cp})	Multiplication Delay
PISO [44]	kn	$k(n-1)$	n	$T_A + \lceil \log_2 n \rceil T_X$	wT_{cp}
Comb [31]	kn	kn	$2n$	$T_A + \lceil \log_2 (k+1) \rceil T_X$	wT_{cp}
ASH [32]	kn	$kn+n$	n	$T_A + \lceil \log_2 (k+1) \rceil T_X$	wT_{cp}
Proposed RB	kn	$(2k-1)n$	$(k+1)n$	$T_A + T_X$	$wT_{cp} + \lceil \log_2 k \rceil T_X$

5.5.2 Comparison to Other Word Level NB and RB Multipliers When There Exists A Type I ONB

Type I ONB is probably the most popular and the most efficient class of NB that is used for realization of NB multipliers in the literature. For this class of fields, the size of the RB is almost the same as that of the NB as shown in Remark 1 in Section 2. Complexity of the proposed multiplier and similar proposed architectures for a class of field that there exist a type I ONB are shown in Table 5.3. The complexities for area and delay are the result of substitution of n with $m + 1$, according to remark 1.

Table 5.3: Are-Delay Complexity comparison for different architectures where there exist a type I ONB

Multiplier	Basis	# AND	# XOR	# Reg	Critical Path Delay (T_{cp})	Multiplication Delay
WLMO [22]	ONB I	$k(2m-1)$	$k(2m-2)$	$2m$	$T_A + (1 + \lceil\log_2 m\rceil)T_X$	wT_{cp}
IMO [11]	ONB I	km	$k(2m-2)$	$2m$	$T_A + (1 + \lceil\log_2 m\rceil)T_X$	wT_{cp}
AEDS [36]	ONB I	$(k+1)\frac{m}{2}$	$(k+1)(\frac{3}{2}m - 2) + 1$	$2m$	$T_A + (1 + \lceil\log_2 m\rceil)T_X$	wT_{cp}
XEDS [36]	ONB I	$k(m-1) + m$	$(k+1)(m-1)$	$2m$	$T_A + (1 + \lceil\log_2 m\rceil)T_X$	wT_{cp}
w-SMPOI [37]	ONB I	$k\frac{m}{2} + m + k + 1$	$3k\frac{m}{2} + k + m - 1$	$3m$	$2T_A + (3 + \lceil\log_2(k-1)\rceil)T_X$	wT_{cp}
w-SMPOII [37]	ONB I	$km + m + k + 1$	$km + k + m - 1$	$3m$	$2T_A + (3 + \lceil\log_2(k-1)\rceil)T_X$	wT_{cp}
PISO [44]	RB	$km + k$	km	$m + 1$	$T_A + \lceil\log_2(m+1)\rceil T_X$	wT_{cp}
Comb [31]	RB	$km + k$	$km + k$	$2m + 2$	$T_A + \lceil\log_2(k+1)\rceil T_X$	wT_{cp}
ASH [32]	RB	$km + k$	$km + k + m + 1$	$m + 1$	$T_A + \lceil\log_2(k+1)\rceil T_X$	wT_{cp}
Proposed	RB	$km + k$	$(2k-1)(m+1)$	$(k+1)(m+1)$	$T_A + T_X$	$wT_{cp} + \lceil\log_2 k\rceil T_X$

For the purpose of illustration we have tabulated the area-delay complexity for the proposed architectures with the previously proposed multipliers in Table 5.4. The field size is chosen as $m = 268$ where there exists a type I ONB. Number of parallel modules is selected to be $k = 8, 16, 32$ which represent practical-size multipliers for VLSI implementations.

The following assumptions are made in Table 5.4: The VLSI area of an XOR gate and a flip-flop are assumed to be twice and three times of the area of an AND gate respectively [1]. It is also assumed that the area of an AND gate is 1. The row Area Cost in Table 5.4 represents the sum of the number of AND gates, twice the number of XOR gates and three

[1]In a typical CMOS VLSI realization, an AND gate can be implemented with 6 transistors, while an XOR gate and one flip-flop can be implemented with 12 and 16 transistors, respectively [40].

Table 5.4: Complexity comparison of word level type I optimal NB or RB multipliers in $\mathbb{F}_{2^{268}}$ for different values of w, k.

Multiplier	Basis	w, k	#AND (N_A)	#XOR (N_X)	#Register (N_R)	Area Cost ($N_A + 2N_X + 3N_R$)	Delay Cost ($T_A = 1, T_X = 2$)	Area × Delay
WLMO [22]	ONB I	34, 8	4280	4272	536	14432	$34T_A + 340T_X = 714$	10304448
IMO [11]	ONB I		2144	4272	536	12296	$34T_A + 340T_X = 714$	8779344
AEDS [36]	ONB I		1206	3601	536	10016	$34T_A + 340T_X = 714$	7151424
XEDS [36]	ONB I		2404	2403	536	8818	$34T_A + 40T_X = 714$	6296052
w-SMPOI [37]	ONB I		1349	3491	804	10743	$68T_A + 204T_X = 476$	5113668
w-SMPOII [37]	ONB I		2421	2419	804	9671	$68T_A + 204T_X = 476$	4603396
PISO [44]	RB		2152	2144	269	7247	$34T_A + 306T_X = 646$	4681562
Comb [31]	RB		2152	2152	538	8070	$34T_A + 136T_X = 306$	2469420
ASH [32]	RB		2152	2421	269	7801	$34T_A + 136T_X = 306$	2387106
Proposed	RB		2152	4035	2421	17485	$34T_A + 37T_X = 108$	1888380
WLMO [22]	ONB I	17, 16	8560	8544	536	27256	$17T_A + 170T_X = 357$	9730392
IMO [11]	ONB I		4288	8544	536	22984	$17T_A + 170T_X = 357$	8205288
AEDS [36]	ONB I		2278	6801	536	17488	$17T_A + 170T_X = 357$	6243216
XEDS [36]	ONB I		4540	4539	536	15226	$17T_A + 170T_X = 357$	5435682
w-SMPOI [37]	ONB I		2429	6715	804	18271	$34T_A + 119T_X = 272$	4969712
w-SMPOII [37]	ONB I		4573	4571	804	16127	$34T_A + 119T_X = 272$	4386544
PISO [44]	RB		4304	4288	269	13687	$17T_A + 153T_X = 323$	4420901
Comb [31]	RB		4304	4304	538	14526	$17T_A + 85T_X = 187$	2716362
ASH [32]	RB		4304	4573	269	14257	$17T_A + 85T_X = 187$	2666059
Proposed	RB		4304	8339	4573	34701	$17T_A + 21T_X = 59$	2047359
WLMO [22]	ONB I	9, 32	17120	17088	536	52904	$9T_A + 90T_X = 189$	9998856
IMO [11]	ONB I		8576	17088	536	44360	$9T_A + 90T_X = 189$	8384040
AEDS [36]	ONB I		4422	13201	536	32432	$9T_A + 90T_X = 189$	6129648
XEDS [36]	ONB I		8812	8811	536	28042	$9T_A + 90T_X = 189$	5299938
w-SMPOI [37]	ONB I		4589	13163	804	33327	$18T_A + 72T_X = 162$	5398974
w-SMPOII [37]	ONB I		8877	8875	804	29039	$18T_A + 72T_X = 162$	4704318
PISO [44]	RB		8608	8576	269	26567	$9T_A + 81T_X = 171$	4542957
Comb [32]	RB		8608	8608	538	27169	$9T_A + 54T_X = 117$	3210246
ASH [32]	RB		8608	8877	269	27169	$9T_A + 54T_X = 117$	3178773
Proposed	RB		8608	16947	8877	69133	$9T_A + 14T_X = 37$	2557921

times the number of registers. The delay for an XOR gate is assumed to be twice of that for an AND gate [5]. If the delay of an AND is assumed to be 1, the row Delay Cost in Table 5.4 represents the multiplication delay in terms of times of the delay of an AND gate.

Compared to the other architectures the proposed architecture has much smaller delay cost at expense of modestly higher area cost. Note that both area and delay are the objectives to minimize for a word level multiplier, and decreasing one is usually at the expense of increasing the other. In Table 5.4 we also used area-delay product to show the balance between the changes of area cost and delay cost, which is given at the last column in the table.

Table 5.5: Normalized Complexity Comparison of Different Word Size Multipliers in $\mathbb{F}_{2^{268}}$

Multiplier	$w = 34, k = 8$			$w = 17, k = 16$			$w = 9, k = 32$		
	Relative Area	Relative Delay	Relative Area $\times$ Delay	Relative Area	Relative Delay	Relative Area $\times$ Delay	Relative Area	Relative Delay	Relative Area $\times$ Delay
WLMO [22]	82%	661%	545%	78%	605%	475%	76%	510%	390%
IMO [11]	70%	661%	464%	66%	605%	400%	64%	510%	327%
AEDS [36]	57%	661%	378%	50%	605%	304%	46%	510%	239%
XEDS [36]	50%	661%	333%	43%	605%	265%	50%	510%	207%
w-SMPOI [37]	61%	440%	270%	52%	461%	242%	48%	437%	211%
w-SMPOII [37]	55%	440%	243%	46%	461%	214%	42%	437%	189%
PISO [44]	41%	598%	247%	39%	547%	215%	38%	462%	177%
Comb [31]	46%	283%	130%	41%	316%	132%	39%	316%	125%
ASH [32]	44%	283%	126%	41%	316%	130%	39%	316%	124%
Proposed	100%	100%	100%	100%	100%	100%	100%	100%	100%

It can be seen that for all three word sizes, the area-delay-product for the new multiplier is much smaller than all the previously proposed architectures listed in the table.

If the area-delay-product for the proposed multiplier is one, then the relative values of the area-delay-product for some previously proposed architectures are listed in Table 5.5. IT can be seen from the table that the area-delay-product for the new multiplier is at most 80% of that of the any other ones for given values of w and k in the table.

5.6 Conclusions

A high speed word level finite field multiplier using RB has been proposed. The proposed architecture is significantly faster compared to previously proposed architectures at the expense of moderately higher area complexity. For the class of fields that there exists a type I ONB, the proposed multiplier performs much faster than other word level NB multipliers available in open literature. It was shown that the new multiplier excels all the other multipliers in comparison when considering the product of area and delay as a measure of performance.

Chapter 6

High Speed Word Level Multipliers in $GF(2^m)$ Using Reordered Normal Basis

6.1 Introduction

An important factor that has great effect on finite field arithmetic efficiency is the basis used to represent the field elements. Common bases used in practice are polynomial basis (PB) and normal basis (NB) [25],[33]. Polynomial basis is probably the most popular basis which has been widely used for hardware and software implementations [18]. Normal basis on the other hand is advantageous for hardware implementation since squaring operation can be implemented at no cost. Free squaring operation can be used to speed up the exponentiation operation by repeated squaring and multiplication [14],[2].

Since addition operation can be implemented simply by exclusive or and inversion operation with repetitive squaring and multiplication, multiplication operation is considered to be the main operation for systems using normal basis. In normal basis, complexity of multiplication is measured with the multiplication matrix [30]. For a binary extension field the multiplication matrix entries are either zero or one and the number of ones inside the multiplication matrix is referred to as normal basis complexity. The normal basis in GF(2^m) for which the complexity achieves its minimum , $2m - 1$, is referred to as the optimal normal basis (ONB). Two types of optimal normal bases have been found which are referred to as type I and type II optimal normal basis[30]. Reordered normal basis is refereed to as a certain permutation of a type II optimal normal basis [12], [44].

In this work two new word-level finite field multipliers using a reordered normal basis are presented. It is shown that the proposed architectures are faster than all the previously presented architectures in the open literature using either a type II optimal normal basis or a reordered normal basis at the expense of moderately higher complexity. One unique feature of the new word-level architectures is that the critical path delay is independent of the word size or the field size. This enables the proposed multipliers to operate at very high clock rate regardless of the word size or the field size.

The organization of this chapter is as follows. Reordered normal basis and multiplication using this basis are briefly reviewed in Section 2. In Sections 3 and 4, two new word-level multipliers using reordered normal basis are respectively proposed. Architectural complexity comparison of proposed architectures with similar proposals are presented in Section 5. Finally some concluding remarks are given in Section 6.

6.2 A Brief Review of Reordered Normal Basis and Its Arithmetic in GF(2^m)

6.2.1 Reordered Normal Basis

The idea of reordered normal basis was first proposed by Gao and Vanstone in [12] and was later used to design several multiplier architectures in [44].

Theorem 6.2.1. [12] Let β be a primitive $(2m+1)^{\text{st}}$ root of unity in $GF(2^m)$. Then $\gamma = \beta + \beta^{-1}$ generates a type II optimal normal basis. The ordered set $\{\gamma_i, i = 1, 2, \ldots, m\}$ with $\gamma_i = \beta^i + \beta^{-i}$, also forms a basis in $GF(2^m)$.

It has been shown that the basis $[\gamma_1, \gamma_2, \ldots, \gamma_m]$[1] is a permutation of the normal basis $[\gamma^{2^0}, \gamma^{2^1}, \ldots, \gamma^{2^{m-1}}]$ [12], and it is referred to as the reordered normal basis following [44].

Define function $s(i)$ which maps the set of integers to the set $\{0, 1, \ldots, m\}$ as follows [12, 44]:

$$s(i) \triangleq \begin{cases} i \bmod 2m+1, & \text{if } 0 \leqslant i \bmod 2m+1 \leqslant m, \\ 2m+1-i \bmod 2m+1, & \text{if } m < i \bmod 2m+1 \leqslant 2m. \end{cases} \tag{6.1}$$

The following lemmas will be useful either to facilitate reordered normal basis arithmetic or to derive a reordered normal basis from a given type II ONB. Note that all the results in the lemmas have been already stated in [12].

Lemma 6.2.2. Let β and γ be defined as in Theorem 6.2.1. Then $\gamma_i = \gamma_{s(i)}$ for any integer i.

Proof: Consider the following two cases and note that β is a primitive $(2m+1)^{\text{st}}$ root of unity in $GF(2^m)$.

- Case 1: If $0 \leqslant i \bmod 2m+1 \leqslant m$, then

$$\gamma_i = \beta^i + \beta^{-i} = \beta^{i \bmod 2m+1} + \beta^{-(i \bmod 2m+1)} = \beta^{s(i)} + \beta^{-s(i)} = \gamma_{s(i)}.$$

- Case 2: If $m < i \bmod 2m+1 \leqslant 2m$, then

$$\begin{aligned} \gamma_i &= \beta^i + \beta^{-i} \\ &= \beta^{i \bmod 2m+1} + \beta^{-i \bmod 2m+1} \\ &= \beta^{-(2m+1-i \bmod 2m+1)} + \beta^{2m+1-i \bmod 2m+1} \\ &= \beta^{-s(i)} + \beta^{s(i)} = \gamma_{s(i)}. \end{aligned}$$

[1]We use $[\cdots]$ to denote an ordered set.

□

Lemma 6.2.3. Given a type II optimal normal basis $I' = [\gamma^{2^0}, \gamma^{2^1}, \ldots, \gamma^{2^{m-1}}]$. Then the reordered normal basis $I = [\gamma_1, \gamma_2, \ldots, \gamma_m]$ is a permutation of the basis elements of I', and the permutation function is decided by

$$\gamma^{2^i} = \gamma_{s(2^i)},\ i = 0, 1, \ldots, m-1.$$

Proof: It follows Lemma 6.2.2 that

$$\gamma^{2^i} = \beta^{2^i} + \beta^{-2^i} = \gamma_{2^i} = \gamma_{s(2^i)} \text{ for } i = 0, 1, \ldots, m-1.$$

□

Lemma 6.2.4. For all $i, j = 1, 2, \ldots, m$, we have

$$\gamma_i\gamma_j = \gamma_{s(i+j)} + \gamma_{s(i-j)}.$$

Proof:

$$\begin{aligned} \gamma_i\gamma_j &= (\beta^i + \beta^{-i})(\beta^j + \beta^{-j}) \\ &= \beta^{i+j} + \beta^{-(i+j)} + \beta^{i-j} + \beta^{-(i-j)} \\ &= \gamma_{i+j} + \gamma_{i-j} \\ &= \gamma_{s(i+j)} + \gamma_{s(i-j)}. \end{aligned}$$

Note that the last step comes from Lemma 6.2.2. □

6.2.2 Reordered Normal Basis Multiplication

Assume that A and B are two arbitrary elements in $GF(2^m)$ represented with respect to (w.r.t) reordered normal basis $I = [\gamma_1, \gamma_2, \ldots, \gamma_m]$,

$$A = \sum_{i=1}^{m} a_i\gamma_i \text{ and } B = \sum_{i=1}^{m} b_i\gamma_i.$$

Their product C is represented with respect to the same basis $C = \sum_{i=1}^{m} c_i\gamma_i$. Assuming $\gamma_0 = 0$ and $b_0 = 0$, then following Lemma 6.2.4 $\gamma_j B, j = 1, 2, \ldots, m$, can be given by [12]

$$\begin{aligned} \gamma_j B &= \gamma_j \sum_{i=1}^{m} b_i \gamma_i \\ &= \sum_{i=1}^{m} b_i [\gamma_{s(i+j)} + \gamma_{s(i-j)}] \\ &= \sum_{i=1}^{m} [b_{s(i+j)} + b_{s(i-j)}]\gamma_i. \end{aligned} \tag{6.2}$$

The last step in (6.2) comes from proper substitution of the subscripts. The product C can be obtained as follows.

$$\begin{aligned} C &= A \cdot B \\ &= \sum_{j=1}^{m} a_j \gamma_j B \\ &= \sum_{j=1}^{m} a_j \sum_{i=1}^{m} [b_{s(i+j)} + b_{s(i-j)}]\gamma_i \\ &= \sum_{i=1}^{m} \Big(\sum_{j=1}^{m} a_j [b_{s(i+j)} + b_{s(i-j)}] \Big) \gamma_i, \end{aligned}$$

Then we have [44]

$$c_i = \sum_{j=1}^{m} a_j [b_{s(i+j)} + b_{s(j-i)}], \quad i = 1, 2, \ldots, m. \tag{6.3}$$

6.3 Proposed High Speed Word Level Multiplier Type One Using Reordered Normal Basis

6.3.1 Word-level multiplication algorithm using reordered normal basis

Let w denote the word size and $k = \lceil m/w \rceil$ be the number of words required for representing a field element in $GF(2^m)$. Write the subscript j of a_j in (6.3) as $j = gw + \ell$ for

$g = 0, 1, \ldots, k-1$ and $\ell = 1, 2, \ldots, w$, and replace j in (6.3) with $gw + \ell$:

$$c_i = \sum_{g=0}^{k-1} \sum_{\ell=1}^{w} a_{gw+\ell}\big[b_{s(i+gw+\ell)} + b_{s(i-gw-\ell)}\big]. \tag{6.4}$$

Note that the coordinates for the operand A and B will be zero if their subscript exceeds m. Define signal $d_{i,g}^{(\ell)}, i = 1, 2, \ldots, m$ and $g = 0, 1, \ldots, k-1$, as

$$\begin{cases} d_{i,g}^{(0)} &= 0 \text{ and} \\ d_{i,g}^{(\ell)} &= d_{i,g}^{(\ell-1)} + a_{gw+\ell}\big[b_{s(i+gw+\ell)} + b_{s(i-gw-\ell)}\big] \text{ for } \ell = 1, 2, \ldots, w. \end{cases} \tag{6.5}$$

Then it follows from (6.5)

$$d_{i,g}^{(w)} = \sum_{\ell=1}^{w} a_{gw+\ell}\big[b_{s(i+gw+\ell)} + b_{s(i-gw-\ell)}\big]. \tag{6.6}$$

Compare (6.4) with (6.6) it follows

$$c_i = \sum_{g=0}^{k-1} d_{i,g}^{(w)}.$$

An algorithm for word-level multiplication using reordered normal basis can be given as follows.

Algorithm 6. Word-level reordered normal basis (RNB) multiplication algorithm I

Input: $A = (a_1, \ldots, a_m)$, $B = (b_1, \ldots, b_m)$, both w.r.t. RNB,
and the word size $w, 1 \leq w \leq m$

Output: $C = A \times B = (c_1, \ldots, c_m)$ also w.r.t. RNB

```
1. Initialization: $k = \lceil m/w \rceil$, and $d_{i,g}^{(0)} = 0$ for $i = 1, 2, \ldots, m$ and $g = 0, 1, \ldots, k-1$
2. For all values of $i = 1, 2, \ldots, m$, compute
3.         For all values of $g = 0, 1, \ldots, k-1$, compute
4.                 For $\ell = 1$ To $w$
5.                         $d_{i,g}^{(\ell)} = d_{i,g}^{(\ell-1)} + a_{gw+\ell}[b_{s(i+gw+\ell)} + b_{s(i-gw-\ell)}]$
6.                 End For
7.         $c_i = \sum_{g=0}^{k-1} d_{i,g}^{(w)}$
8.         End For
9. End For
```

6.3.2 Multiplier Architecture

A high speed word-level reordered normal basis multiplier can be built based on Algorithm 6, which is referred to as word-level reordered normal basis type I (WL-RNB I) and is shown in Fig. 6.1.

From the top to the bottom, the architecture contains a $(2m+1)$-bit circular shift register, the Expansion/Permutation module, one layer of XOR gates, one layer of AND gates, one layer of accumulation units, and one layer of XOR gate networks.

Step 5 of Algorithm 6 can be implemented using one XOR gate, one AND gate and one accumulation unit as shown in the block of dashed lines in Fig. 6.1. The accumulation unit requires one XOR gate and one flip-flop. Steps 2 and 3 of Algorithm 6 require totally $m \times k$ such accumulation units in m groups with each group containing k units. Step 7 of Algorithm 6 shows that each $c_i, i = 1, 2, \ldots, m$, is a sum of k terms which are the outputs of the k accumulation units after w clock cycles. A k-input XOR gate or a binary tree of $k-1$ two-input XOR gates is used to produce the final output c_i as shown at the bottom in Fig. 6.1.

The input operand A is required to be fed into the multiplier in a comb style. Let A be divided into k words, with each word of w bits. Then, in the first clock cycle the inputs

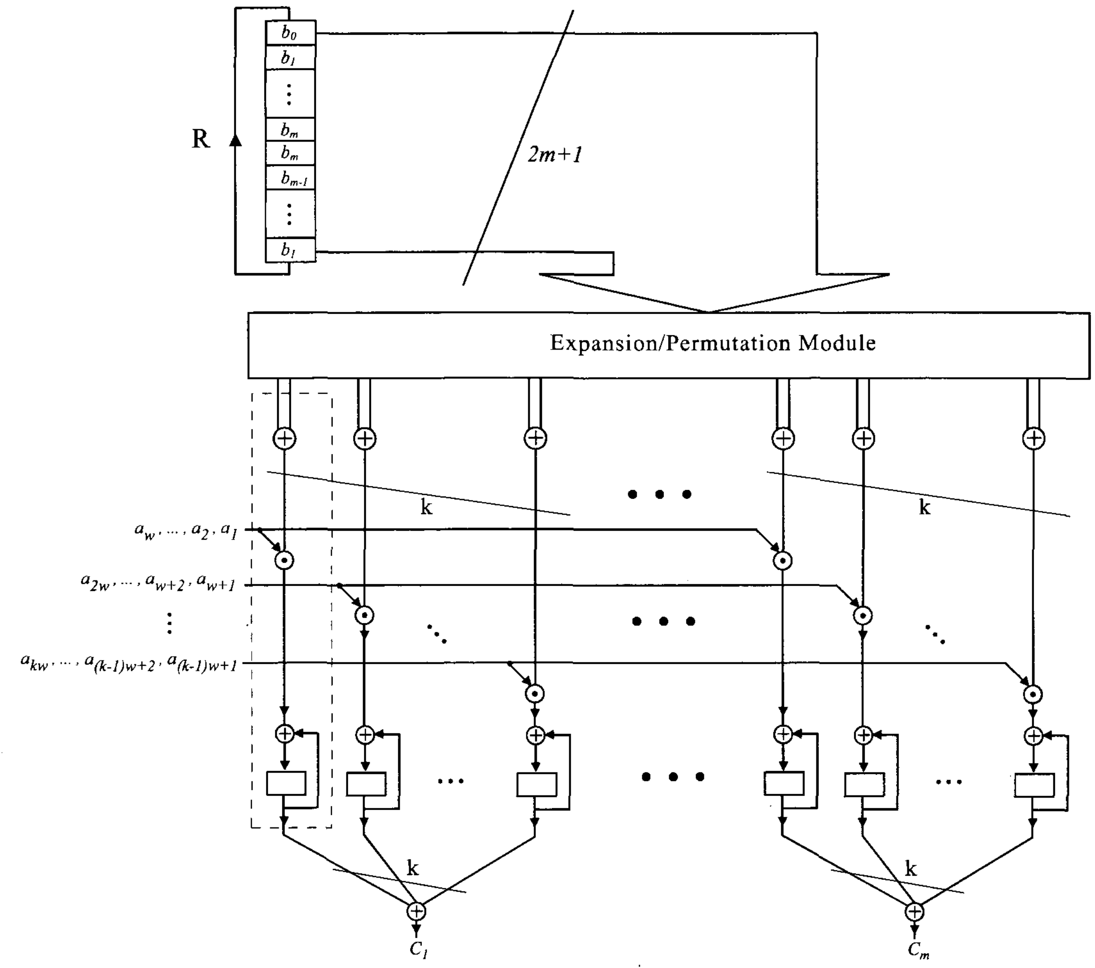

R
b_0
b_1
b_m
b_m
b_{m-1}
b_1
2m+1
Expansion/Permutation Module
k
k
$a_w, \ldots, a_2, a_1$
$a_{2w}, \ldots, a_{w+2}, a_{w+1}$
$a_{kw}, \ldots, a_{(k-1)w+2}, a_{(k-1)w+1}$
k
k
C_1
C_m

Figure 6.1: Proposed word-level high speed multiplier using reordered normal basis

are the first bit of every word, $a_1, a_{w+1}, a_{2w+1}, \ldots, a_{(k-1)w+1}$. For the second clock cycle the inputs are $a_2, a_{w+2}, a_{2w+2}, \ldots, a_{(k-1)w+2}$. Finally in the w^{th} clock cycle the inputs are $a_w, a_{2w}, a_{3w}, \ldots, a_{kw}$. Note that if the subscript of an input bit exceeds m then it is replaced by a zero bit.

The input operand B is stored in a $(2m+1)$-bit circular shift register R, and from there the input bits are fed into Permutation/Extension module. Suppose that the $(2m+1)$-bit circular shift register R is initially loaded as, from the top to bottom, $b_0, b_1, b_2, \ldots, b_m, b_m, b_{m-1}$,

$b_{m-2}, \ldots, b_1$ as shown in Fig. 6.1. Note from (6.1) that $s(i) = s(2m+1-i)$ for $i = 0, 1, 2, \ldots, 2m$, or

$$\begin{array}{lclcl}
b_0 & = & b_{s(0)} & = & b_{s(2m+1)} \\
b_1 & = & b_{s(1)} & = & b_{s(2m)} \\
 & \vdots & & \vdots & \\
b_m & = & b_{s(m)} & = & b_{s(m+1)} \\
b_m & = & b_{s(m+1)} & = & b_{s(m)} \\
b_{m-1} & = & b_{s(m+2)} & = & b_{s(m-1)} \\
 & \vdots & & \vdots & \\
b_1 & = & b_{s(2m)} & = & b_{s(1)}
\end{array}$$

Then register R can be viewed as *two* virtual $(2m+1)$-bit registers R_1 and R_2 as shown in Fig. 6.2(b): From the top to the bottom, one (R_1) contains $b_{s(0)}, b_{s(1)}, b_{s(2)}, \ldots, b_{s(2m)}$, and the other ($R_2$) contains $b_{s(2m+1)}, b_{s(2m)}, b_{s(2m-1)}, \ldots, b_{s(1)}$.

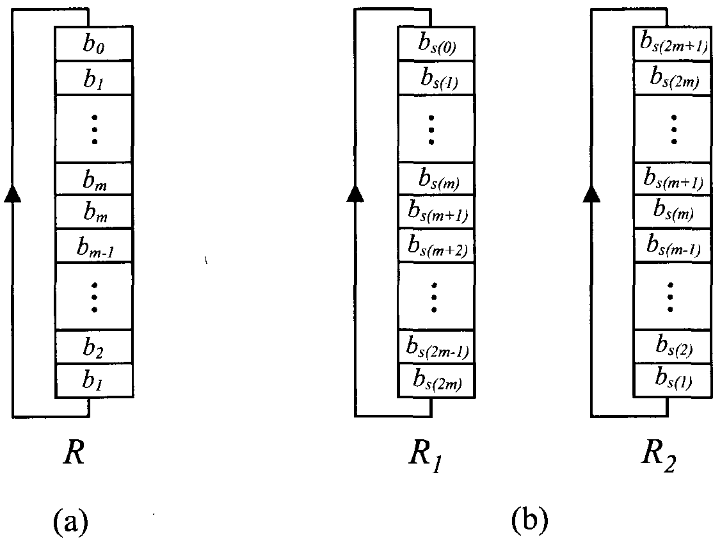


Figure 6.2: Viewing the $(2m+1)$-bit circular shift register R as two virtual $(2m+1)$-bit circular shift registers R_1 and R_2

The following two lemmas are useful for description of the Permutation/Expansion module.

Lemma 6.3.1. If the content of the i^{th} bit (the bit at top as bit zero) of R_1 is given as $r_{1,i}^{(\ell)} = b_{s(h)}$ at the clock cycle ℓ for some integer h, then its contents will be $r_{1,i}^{(\ell+1)} = b_{s(h-1)}$ at the next clock cycle.

Proof: The contents of R_1 starting from the top bit at clock cycle ℓ can be any $(2m+1)$ consecutive bits in the sequence

$$b_{s(0)}, b_{s(1)}, \dots, b_{s(2m)}, b_{s(0)}, b_{s(1)}, \dots, b_{s(2m)}, b_{s(0)}, b_{s(1)}, \dots, b_{s(2m)}, \dots.$$

So it is obvious $r_{1,i}^{(\ell+1)} = b_{s(h-1)}$ if $h > 0$. If $h = 0$ then $r_{1,i}^{(\ell+1)} = b_{s(2m)}$. The lemma follows by noting that $b_{s(2m)} = b_{s(-1)}$ from (6.1). □

Lemma 6.3.2. If the content of the i^{th} bit of R_2 is given as $r_{2,i}^{(\ell)} = b_{s(h)}$ at the clock cycle ℓ for some integer h, then its contents will be $r_{2,i}^{(\ell+1)} = b_{s(h+1)}$ at the next clock cycle.

Proof: The contents of R_2 starting from the top bit at clock cycle ℓ can be any $(2m+1)$ consecutive bits in the sequence

$$b_{s(2m+1)}, b_{s(2m)}, \dots, b_{s(1)}, b_{s(2m+1)}, b_{s(2m)}, \dots, b_{s(1)}, b_{s(2m+1)}, b_{s(2m)}, \dots, b_{s(1)}, \dots.$$

So it is obvious $r_{2,i}^{(\ell+1)} = b_{s(h+1)}$ if $h \neq 2m+1$. If $h = 2m+1$ then $r_{2,i}^{(\ell+1)} = b_{s(1)}$. The lemma follows by noting that $b_{s(1)} = b_{s(2m+2)}$ from (6.1). □

For given i and g in $d_{i,g}^{(\ell)}$ as in (6.5), every clock cycle the variable in the function $s(\cdot)$ increases by one in $b_{s(i+gw+\ell)}$, and decreases by one in $b_{s(i-gw-\ell)}$. So following Lemmas 6.3.1 and 6.3.2 the input $b_{s(i+gw+\ell)}$ is connected to bit $(i+gw+\ell)$ of R_2 while the input $b_{s(i-gw-\ell)}$ is connected to bit $(i-gw-\ell)$ of R_1. Then it is easy to see that Expansion/Permutation is just a reordering and copying module which does not contain any gates. This module accepts $2m+1$ inputs from the circular shift register and provides $2km$ outputs for the layer of XOR gates.

The critical path for the proposed multiplier contains one AND gate and two XOR gates as shown in the block formed by dashed lines in Fig. 6.1. Let T_A and T_X respectively denote the delay of a two-input AND gate and a two-input XOR gate, Then critical path delay is $T_{cp} = T_A + 2T_X$. Note that the critical path delay depends on neither the field size m nor the word size w. It is worth to point out that the binary tree of $k-1$ XOR gates is not part of the critical path since the k summation for the c_i outputs has to be calculated only once at the end of the multiplication. Consequently, the product bits are not immediately available following w clock cycles. Instead a time delay of amount about equal to $T_{ex} = \lceil \log_2 k \rceil T_X$ has to be spent before the product bits are generated at the output ends.

Two options are available for the multipliers to act during the time spent for the final addition of the partial products. In the first option, the multiplier clock should be stopped after w clock cycles and the outputs can be read from the output ports after the addition delay which can be measured approximately beforehand. This style requires an extra counter but can save dynamic power consumption during the final addition.

The second option is to enter zero input bits for input A once the w clock cycles are over. This will eliminate the need for stopping the clock signal of the multiplier and does not require extra circuitry. The output product bits can be read from the output ports after certain number of clock cycles immediately following w cycles. The exact number clock cycles required for a multiplication operation can be computed beforehand, which is (assuming that the clock period is chosen to be the critical path delay T.)

$$w + \left\lceil \frac{T_{ex}}{T_{cp}} \right\rceil = w + \left\lceil \frac{\lceil \log_2 k \rceil T_X}{T_A + 2T_X} \right\rceil.$$

For example, when $m = 233$ and $w = 32$, $k = \lceil m/w \rceil = 8$. Then the product bits are available after a time delay of $T_{ex} = \lceil \log_2 k \rceil T_X = 3T_X$ following $w = 32$ clock cycles. Since the extra time delay $T_{ex} = 4T_X$ is less than twice of the critical time delay $T_{cp} = T_A + 2T_X$ (assuming that $T_A \sim T_X/2$ and the system clock period is chosen to be the critical time delay)[2] , in practical implementation the product bits can be read out after $w + 2 = 34$ clock cycles while two zero bits have to be appended to every input word for

[2]In a typical CMOS VLSI realization, the delay of an AND gate is about half of that of an XOR gate [5].

the operand A. In the rest of the chapter, we assume that the second option is adopted for the multiplier.

6.3.3 Architecture complexities

The area and delay complexity of the proposed design can be determined from Fig. 6.1. From the top to the bottom, the complexity of each part of multiplier can be obtained as follows. The circular shift register R contains $2m + 1$ flip-flops. Expansion/Permutation module does not contain any gates or flip-flops. There are respectively mk gates in the layers of XOR gates and AND gates. The number of accumulation units is also mk, which each contains one flip-flops and one XOR gate. The m binary trees of XOR gates at the bottom consists of in total $m(k-1)$ XOR gates. The complexities can be summarized as follows:

# AND	# XOR	# Registers
km	$(3k-1)m$	$(k+2)m+1$

6.3.4 An example

A proposed word-level multiplier in $GF(2^3)$ using reordered normal basis is shown in Fig. 6.3. The word size is chosen to be $w = 2$ and then $k = \lceil m/w \rceil = 2$. The critical path delay is $T_{cp} = T_A + 2T_X$. It takes $w + 1 = 3$ clock cycles to complete one multiplication, since one extra cycle is needed due to the time delay caused by the XOR gate at the output ends. Correspondingly, a zero bit has to be appended to the end of each of the two input words for the operand A so that the contents of the flip-flops in the accumulation units remain unchanged during the last clock cycle.

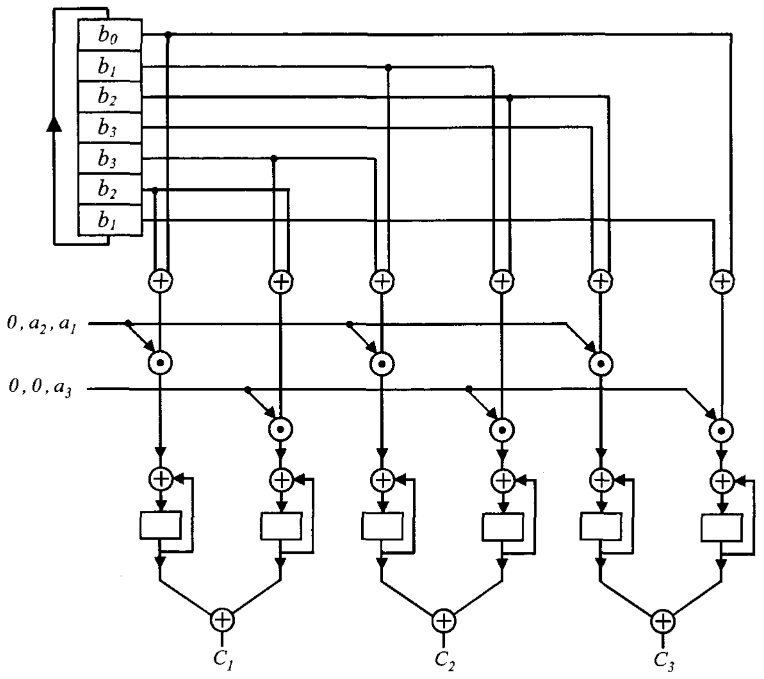


Figure 6.3: Architecture of WL-RNB I in $GF(2^3)$ with $w = k = 2$

6.4 Proposed High Speed Word Level Multiplier Type Two Using Reordered Normal Basis

6.4.1 Word-level multiplication algorithm using reordered normal basis

Let w, k, g, and ℓ be defined as in Subsection III.A. It follows from (6.4)

$$\begin{aligned} c_i &= \sum_{g=0}^{k-1}\sum_{\ell=1}^{w} a_{gw+\ell}[b_{s(i+gw+\ell)} + b_{s(i-gw-\ell)}] \\ &= \sum_{g=0}^{k-1}\left[\sum_{\ell=1}^{w} a_{gw+\ell}b_{s(i+gw+\ell)} + \sum_{\ell=1}^{w} a_{gw+\ell}b_{s(i-gw-\ell)}\right]. \end{aligned} \tag{6.7}$$

Define signals $d_{i,g}^{(\ell)}$ and $e_{i,g}^{(\ell)}, i = 1, 2, \ldots, m, g = 0, 1, \ldots, k-1$, for $\ell = 1, 2, \ldots, w$, as follows

$$\begin{cases} d_{i,g}^{(0)} &= \ = 0 \\ d_{i,g}^{(\ell)} &= d_{i,g}^{(\ell-1)} + a_{gw+\ell} b_{s(i+gw+\ell)} \\ e_{i,g}^{(0)} &= 0 \\ e_{i,g}^{(\ell)} &= e_{i,g}^{(\ell-1)} + a_{gw+\ell} b_{s(i-gw-\ell)} \end{cases} \tag{6.8}$$

Then it follows from (6.8)

$$\begin{cases} d_{i,g}^{(w)} &= \sum_{\ell=1}^{w} a_{gw+\ell} b_{s(i+gw+\ell)}, \\ e_{i,g}^{(w)} &= \sum_{\ell=1}^{w} a_{gw+\ell} b_{s(i-gw-\ell)}. \end{cases} \tag{6.9}$$

Compare (6.7) with (6.9) it follows

$$c_i = \sum_{g=0}^{k-1} [d_{i,g}^{(w)} + e_{i,g}^{(w)}] = \sum_{g=0}^{k-1} d_{i,g}^{(w)} + \sum_{g=0}^{k-1} e_{i,g}^{(w)}.$$

An algorithm form for a high speed word-level multiplication is shown as follows.

Algorithm 7. Word-level reordered normal basis (RNB) multiplication algorithm II

Input: $A = (a_1, \ldots, a_m)$, $B = (b_1, \ldots, b_m)$, both w.r.t. RNB,

and the word size $w, 1 \le w \le m$

Output: $C = A \times B = (c_1, \ldots, c_m)$ also w.r.t. RNB

```
1.  Initialization: $k = \lceil m/w \rceil$, and $d_{i,g}^{(0)} = 0, e_{i,g}^{(0)} = 0$
    for $i = 1, 2, \ldots, m$ and $g = 0, 1, \ldots, k-1$
2.  For all values of $i = 1, 2, \ldots, m$, compute
3.          For all values of $g = 0, 1, \ldots, k-1$, compute
4.                  For $\ell = 1$ To $w$ By Step One
5.                          $d_{i,g}^{(\ell)} = d_{i,g}^{(\ell-1)} + a_{gw+\ell} b_{s(i+gw+\ell)}$
6.                          $e_{i,g}^{(\ell)} = e_{i,g}^{(\ell-1)} + a_{gw+\ell} b_{s(i-gw-\ell)}$
7.                  End For
8.          $c_i = \sum_{g=0}^{k-1} d_{i,g}^{(w)} + \sum_{g=0}^{k-1} e_{i,g}^{(w)}$
9.          End For
10. End For
```

6.4.2 Multiplier architecture

A word-level multiplier following Algorithm 7 using RNB, which is referred to as WL-RNB II, is shown in Fig. 6.4. It can be seen that the architecture is similar to that of WL-RNB I. The $(2m+1)$-bit circular shift register and the Expansion/Permutation module are the same for the two architectures.

The main difference between the two multiplier architectures lies within the blocks of dashed lines respectively shown in Fig. 6.1 and Fig. 6.4. In the dashed block of WLM-RNB I shown in Fig. 6.1, the two bits of operand B are first added together and then the sum multiplies one bit of A to produce a partial product bit. The partial product bit is then fed into the accumulation unit. After w clock cycles the content of the accumulation unit along with those of the other $k-1$ accumulation units is fed into the XOR network to generate one product bit c_i.

The dashes block of WL-RNB II has the same inputs of two bits of B as that of WL-RNB I. Each input bit is first multiplies one bit of A to produce a partial product bit. The partial product bit is then fed into the accumulation unit. After w clock cycles the content

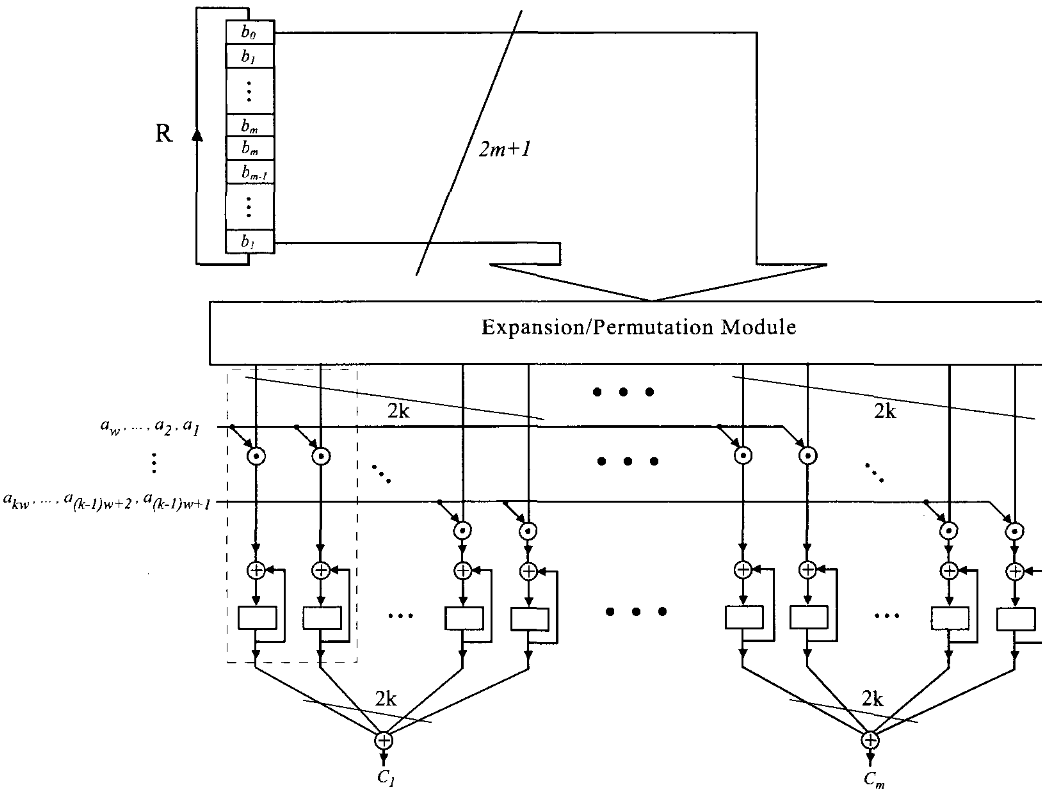


Figure 6.4: Proposed word-level high speed multiplier using reordered normal basis

of the accumulation unit along with those of the other $2k - 1$ accumulation units is fed into the XOR network to generate one product bit c_i. Note that the critical path for WL-RNB II contains only one AND and one XOR gate, which is shorter than that of WL-RNB I by one XOR gate. The final XOR gate network for generating c_i has $2k$ inputs, twice of the XOR network in WL-RNB I. So the final product bits are available after an extra time delay of about $\lceil \log_2(2k) \rceil T_X$ immediately follow w clock cycles.

6.4.3 Architecture complexity

The area and delay complexity of the proposed design can be determined from Fig. 6.4. Registers are presented in two parts of the design, first part is the circular shift register,

R, which contains $2m + 1$ flip-flops. The second part is the one bit flip-flops that hold the partial sum of the output product bits, which their number is equal to $2km$ since each output bit uses $2k$ flip-flops. The total number of two-input AND gates is equal to $2km$ since each output product bit uses $2k$ two-input AND gates. XOR gates exist in two parts of the architecture. The first part is XOR gates that follow the AND gates. For each output product the number of these two-input XOR gates is equal to $2k$ which is the same as the number of AND gates. For m output products, totally $2km$ of these two-input XOR gates exist. The second part is the $2k$-input XOR gates which exist for each of the output product bits. The equivalent number of two-input XOR gates to implement these $2k$-input XOR gates is equal to $2(k - 1)m$. Consequently The equivalent number of two-input XOR gates in the architecture is equal to $(4k - 1)m$. So the complexities can be summarized as follows:

# AND	# XOR	# Registers
$2km$	$4km - m$	$2km + 2m + 1$

The critical path delay is $T_{cp} = T_A + T_X$. Similar to WL-RNB I, the product bits are not immediately available following w clock cycles. Instead a time delay of amount about equal to $T_{ex} = \lceil \log_2 2k \rceil T_X$ has to be spent before the product bits are generated at the output ends. If the clock period is chosen to be equal to the critical path delay, the total number of clock cycles for completing a multiplication is

$$w + \left\lceil \frac{\lceil \log_2 2k \rceil T_X}{T_A + T_X} \right\rceil.$$

Take again the example of $m = 233$ and $w = 32$, $k = \lceil m/w \rceil = 8$. Then the product bits are available after a time delay of $\lceil \log_2 2k \rceil T_X = 4T_X$ following $w = 32$ clock cycles. Since the extra time delay $T_{ex} = 4T_X$ is about three times of the critical time delay $T_{cp} = T_X + T_A$, in practical implementation the product bits can be read out after $w + 3 = 35$ clock cycles while three zero bits have to be appended to the end of each input word from A to keep the contents of the flip-flops of the accumulation units unchanged during the last three clock cycles.

6.4.4 An example

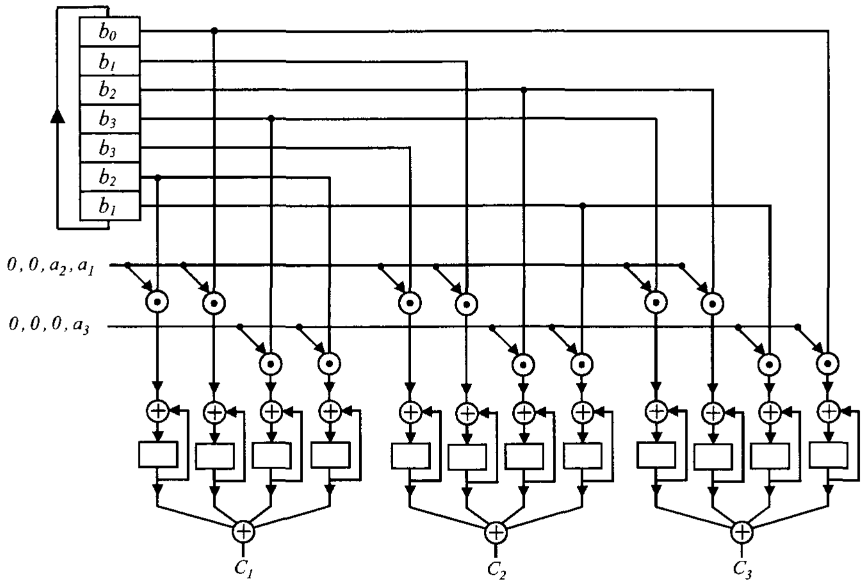


Figure 6.5: Architecture of WLM-RNB-II in $GF(2^3)$ with $w = k = 2$

A diagram of WL-RNB-II in $GF(2^3)$ is shown in Fig. 6.5. The word size is chosen to be $w = 2$ and then $k = \lceil m/w \rceil = 2$. The critical path delay is $T_{cp} = T_A + T_X$. Since a binary tree of 4 XOR gates is used at the output ends to generate the final product bits c_i which has a time delay of $2T_X$, two extra clock cycles following w clock cycles are needed before the product bits are available at the output ends. Correspondingly, two zero bits have to be appended to each of the two input words for the operand A so that the contents of the flip-flops in the accumulation units remain unchanged during the last two clock cycles.

6.5 Comparisons

Complexity comparison between the proposed multipliers and some other multipliers in the literature is made and shown in Table 6.1. Since a reordered normal basis is a permutation of a type II ONB, it should be interesting to have complexity comparison of the proposed reordered normal basis multipliers to some popular NB multipliers for the class of fields that there exists a type II ONB. These architectures are shown in the top six rows in the table. The table also includes two previously proposed word-level reordered normal basis multipliers as shown in rows seven and eight.

The first row of Table 6.1 represents the word-level Massey-Omura (WLMO) multiplier which uses k identical bit-level Massey-Omura multipliers [22], while the second row shows the improved Massey-Omura multiplier (IMO) [11]. The AND-efficient digit-serial (AEDS) and XOR-efficient digit-serial (XEDS) multipliers proposed in [36] are shown at the third and fourth rows of the table. Fifth and sixth rows of the table represent respectively the word-level sequential multipliers with parallel output type I and type II, which were recently reported in [37].

The seventh row of the table shows the Hybrid PISO architecture proposed in [44] with k levels of pipelining and the eighth row gives the Comb Style architecture recently proposed in [31]. The last two rows of the table present the proposed architectures WL-RNB I and WL-RNB II.

As can be seen the critical path delays (T_{cp}) of the proposed architectures are independent of the field size m or the word size w, and much smaller than any other previously proposed word-level architectures shown in the table. Note that the gain in the clock speed is at the expense of using significantly more flip-flops for both WL-RNB I and WL-RNB II and more VLSI gates for WL-RNB II.

The last column (Multiplication Delay) of Table 6.1 represents the time it takes to complete one multiplication operation for a multiplier. Note that proposed multipliers need some extra time ($\lceil \log_2 k \rceil T_X$ or $\lceil \log_2 2k \rceil T_X$) besides w clock cycles. If the system clock period is chosen as the critical path delay, to complete one multiplication operation WL-RNB I would take $w + \lceil \frac{\lceil \log_2 k \rceil T_X}{T_A + 2T_X} \rceil$ clock cycles while WL-RNB II would require $w + \lceil \frac{\lceil \log_2 2k \rceil T_X}{T_A + T_X} \rceil$ clock cycles.

For the purpose of illustration we have tabulated the area-delay complexity for the pro-

Table 6.1: Complexities comparison

Multiplier	Basis	# AND	# XOR	# Reg	Critical Path Delay (T_{cp})	Multiplication Delay
WLMO [22]	ONB II	$k(2m-1)$	$k(2m-2)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	wT_{cp}
IMO [11]	ONB II	km	$k(2m-2)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	wT_{cp}
AEDS [36]	ONB II	$k(m - \frac{1}{2}k + \frac{1}{2})$	$k(3m-k-2)$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	wT_{cp}
XEDS [36]	ONB II	$k(2m-k)$	$k(2m - \frac{1}{2}k - \frac{3}{2})$	$2m$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	wT_{cp}
w-SMPOI [37]	ONB II	$k(\lfloor \frac{m}{2} \rfloor + 1) + m$	$k(2m-1)$	$3m$	$2T_A + (3 + \lceil \log_2 (k-1) \rceil) T_X$	wT_{cp}
w-SMPOII [37]	ONB II	$km + m$	$k(m + \lfloor \frac{m}{2} \rfloor)$	$3m$	$2T_A + (3 + \lceil \log_2 (k-1) \rceil) T_X$	wT_{cp}
PISO [44]	RNB	km	$k(2m-1)$	$2m+1$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	wT_{cp}
Comb Style [31]	RNB	km	$2km$	$3m+1$	$T_A + (1 + \lceil \log_2 (k+1) \rceil) T_X$	wT_{cp}
WL-RNB I	RNB	km	$2km$	$(k+2)m+1$	$T_A + 2T_X$	$wT_{cp} + \lceil \log_2 k \rceil T_X$
WL-RNB II	RNB	$2km$	$(4k-1)m$	$2(k+1)m+1$	$T_A + T_X$	$wT_{cp} + \lceil \log_2 2k \rceil T_X$

posed architectures with the previously proposed multipliers in Table 6.2. The field size is chosen as $m = 233$ because the field $GF(2^{233})$ is both one of the few recommendations by National Institute of Standards and Technology (NIST) [34] and a field where there exists a type II ONB or a reordered normal basis. Word sizes of $16, 32$ and 64 bits are adopted which represent some typical bus width for a general processor or an embedded computer system.

The following assumptions are made in Table 6.2: The VLSI area of an XOR gate and a flip-flop are assumed to be twice and three times of the area of an AND gate respectively[3]. It is also assumed that the area of an AND gate is 1. The row Area Cost in Table 6.2 represents the sum of the number of AND gates, twice the number of XOR gates and three times the number of registers. The delay for an XOR gate is assumed to be twice of that for an AND gate [5]. If the delay of an AND is assumed to be 1, the row Delay Cost in Table 6.2 represents the multiplication delay in terms of times of the delay of an AND gate.

Compared to the other architectures the proposed architectures have much smaller delay cost at expense of modestly higher area cost. Note that both area and delay are the objectives to minimize for a word-level multiplier, and decreasing one is usually at the expense of increasing the other. In Table 6.2 we also used area-delay product to show the balance between the changes of area cost and delay cost, which is given at the last column

[3]In a typical CMOS VLSI realization, an AND gate can be implemented with 6 transistors, while an XOR gate and one flip-flop with set/reset inputs can be implemented with 12 and 16 transistors, respectively [40].

Table 6.2: Complexity comparison of word-level type II optimal normal basis or reordered normal basis multipliers in GF(2^{233}) for different values of w, k.

Multiplier	Basis	w, k	#AND (N_A)	#XOR (N_X)	#Register (N_R)	Area Cost ($N_A + 2N_X + 3N_R$)	Delay Cost ($T_A = 1, T_X = 2$)	Area × Delay
WLMO [22]e	ONB II		6975	6960	466	22293	$16T_A + 144T_X = 304$	6777072
IMO [11]	ONB II		3495	6960	466	18813	$16T_A + 144T_X = 304$	5719152
AEDS [36]	ONB II		3390	10230	466	25248	$16T_A + 144T_X = 304$	7675392
XEDS [36]	ONB II		6765	6855	466	21873	$16T_A + 144T_X = 304$	6649392
w-SMPOI [37]	ONB II	16, 15	1988	6975	699	18035	$32T_A + 112T_X = 256$	4616960
w-SMPOII [37]	ONB II		3728	5235	699	16295	$32T_A + 112T_X = 256$	4171520
PISO [44]	RNB		3495	6975	467	18846	$16T_A + 144T_X = 304$	5729184
Comb Style [31]	RNB		3495	6990	700	19575	$16T_A + 80T_X = 176$	3445200
WL-RNB I	RNB		3495	6990	3962	29361	$16T_A + 36T_X = 88$	2583768
WL-RNB II	RNB		3990	13747	7457	53855	$16T_A + 21T_X = 58$	3123590
WLMO [22]	ONB II		3720	3712	466	12542	$32T_A + 288T_X = 608$	7625536
IMO [11]	ONB II		1864	3712	466	10686	$32T_A + 288T_X = 608$	6497088
AEDS [36]	ONB II		1836	5512	466	14258	$32T_A + 288T_X = 608$	8668864
XEDS [36]	ONB II		3664	3684	466	12430	$32T_A + 288T_X = 608$	7557440
w-SMPOI [37]	ONB II	32, 8	1169	3720	699	10706	$64T_A + 192T_X = 448$	4796288
w-SMPOII [37]	ONB II		2097	2792	699	9778	$64T_A + 192T_X = 448$	4380544
PISO [44]	RNB		1864	3720	467	10705	$32T_A + 288T_X = 608$	6508640
Comb Style [31]	RNB		1864	3728	700	11420	$32T_A + 160T_X = 352$	4019840
WL-RNB I	RNB		1864	3728	2331	16313	$32T_A + 67T_X = 166$	2707958
WL-RNB II	RNB		3728	7223	4159	30651	$32T_A + 36T_X = 104$	3187704
WLMO [22]	ONB II		1860	1856	466	6970	$64T_A + 576T_X = 1216$	8475520
IMO [11]	ONB II		932	1856	466	6042	$64T_A + 576T_X = 1216$	7347072
AEDS [36]	ONB II		926	2772	466	7868	$64T_A + 576T_X = 1216$	9567488
XEDS [36]	ONB II		1848	1850	466	6946	$64T_A + 576T_X = 1216$	8446336
w-SMPOI [37]	ONB II	64, 4	701	1860	699	6518	$128T_A + 320T_X = 768$	5005824
w-SMPOII [37]	ONB II		1165	1396	699	6054	$128T_A + 320T_X = 768$	4649472
PISO [44]	RNB		932	1860	467	6053	$64T_A + 567T_X = 1198$	7251494
Comb Style [31]	RNB		932	1864	700	6460	$64T_A + 256T_X = 576$	3720960
WL-RNB I	RNB		932	1864	1399	8857	$64T_A + 130T_X = 324$	2869668
WL-RNB II	RNB		1864	3495	2331	15847	$64T_A + 67T_X = 198$	3137706

in the table. It can be seen that for all three word sizes, the area-delay-product for both the new multipliers is much smaller than all the previously proposed architectures listed in the table. In fact, the area-delay-product for WL-RNB I is only $75\%, 67\%$, and 77% of the previously proposed multiplier with smallest area-delay-product, Comb Style [31], for word sizes $16, 32$, and 64, respectively. WL-RNB II has a slightly higher area-delay-product than WL-RNB I, but the former has the highest speed among all the multipliers listed in the table.

6.6 Conclusions

Two high speed word-level finite field multipliers in $GF(2^m)$ using reordered normal basis are presented. Architectural complexity comparison and numerical examples show that the new architectures are faster compared to other similar proposals using either NB or RNB for the same class of fields. One unique feature of the proposed architectures is that the critical path delay of the multiplier is not a function of the field size or the word size which is the case for the previously proposed word-level multipliers. The high speed nature of the multipliers makes them suitable for high speed public-key cryptography applications where fast finite field multipliers in large fields are needed.

Chapter 7

High Speed VLSI Implementation of a Multiplier Using Redundant Representation

7.1 Introduction

Finite field arithmetic has important applications in number theory, algebraic geometry, and cryptography, particularly in public key cryptography [25, 6]. Elliptic curve and El-Gamal cryptosystems are two important examples of public key cryptosystems based completely on finite field arithmetic [28, 6]. Two types of finite fields are commonly used in practice, prime field $\mathbb{F}_p$ and the binary field $\mathbb{F}_{2^m}$. Binary field is an extension of the prime field, $\mathbb{F}_2$, which contains 2^m elements. Binary fields are attractive for high speed cryptography applications since they are suitable for hardware implementation [28, 18].

The efficiency of finite field multiplication depends on the choice of the basis to represent field elements. Few bases have been proposed in literature, including the polynomial basis, normal basis (NB), dual bases, triangular basis, and redundant representation, or redundant basis [18, 44, 37, 16].

Redundant representation is especially interesting because it not only offers an almost free squaring operation, but also eliminates modular operation for multiplication. The main concept for multiplication using redundant representation is to embed a field in a larger ring and perform the multiplication in the ring [44]. Since embedding a field is not unique, each field element in the ring can be represented in more than one way, such that the representation contains certain amount of redundancy.

The main drawback for the redundant representation is that it uses more bits to represent a field element, where the number of representation bits depends on the size of the cyclotomic ring. If a type I optimal normal basis (ONB) exists in $\mathbb{F}_{2^m}$, the number of bits required for a redundant representation of a field element is $m + 1$.

In this chapter, a new VLSI implementation for a finite field multiplier using redundant basis is proposed. Simulation of the final post place-and-route layout shows that the multiplier can be clocked up to $1.82\,GHz$, which is 143% faster than the static CMOS implementation of the same design with a Virtual Silicon standard cell logic library. Also, the proposed design occupies nearly 16% less silicon area compared to the static CMOS implementation.

Improvements for the new VLSI implementation comes from the fact that the selected multiplier has a very regular architecture and can be implemented completely with multiples of one simple building block. This block, x-module, is made out of one XOR gate, one AND gate, and a flip-flop. The AND-XOR function is achieved with a domino logic cell, and the flip-flop is selected to possess a negative edge triggered, zero hold time, D-flipflop; properties which are required to properly latch the output data of the domino logic cell.

In our VLSI implementation, we have selected an input size of 197 bits, which is in the practical range for cryptography applications [18, 6]. The proposed multiplier design is intended to be used inside an elliptic curve processor, consequently, it was designed as a macro module without any I/O pads.

The organization of the rest of the chapter is as follows: Section 2 is a brief review of redundant basis representation, multiplication and multiplier architectures. In section 3, the design and implementation of the x-module is discussed. Section 4 presents the implementation of a large size multiplier using the x-module as the main building block. Section 5 examines the static CMOS implementation of the multiplier. In section 6, comparisons between different VLSI implementations are presented. A few concluding remarks are given in section 7.

7.2 A Brief Review of Redundant Basis and its Arithmetic in $\mathbb{F}_{2^m}$

7.2.1 Redundant Basis for $\mathbb{F}_{2^m}$

Let K be a field, and $f(x) \in K[x]$ be a polynomial defined over K. Then the field that contains all the roots of $f(x)$ is called the *splitting field* of the polynomial $f(x)$. The splitting field of $x^n - 1$ is called the n^{th} cyclotomic field, denoted by $K^{(n)}$. Let β be a primitive n^{th} root of unity. Then $K^{(n)}$ is generated by β over K and elements in K can be represented in the form

$$A = a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1},\ a_i \in K.$$

Thus the set $[1, \beta, \beta^2, \ldots, \beta^{n-1}]$ can be viewed as a basis for $K^{(n)}$ [13, 14]. Since $1 + \beta + \beta^2 + \cdots + \beta^{n-1} = 0$, the representation of A is not unique. For example, the two n-tuples $(a_0, a_1, \ldots, a_{n-1})$ and $(1 - a_0, 1 - a_1, \ldots, 1 - a_{n-1})$ represent the same element A. So, the basis $[1, \beta, \beta^2, \ldots, \beta^{n-1}]$ is called *redundant basis* for any subfield of $K^{(n)}$. Note that the elements in the redundant basis form a cyclic group of order n and

$$\beta \cdot \beta^i = \begin{cases} \beta^{i+1} & i \neq n-1, \\ 1 & i = n-1. \end{cases}$$

7.2.2 Redundant Basis Multiplication in $\mathbb{F}_{2^m}$

Consider the redundant basis in $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$:

$$I = [1, \beta, \beta^2, \ldots, \beta^{n-1}].$$

Let field elements $A, B \in \mathbb{F}_{2^m}$ to be represented with respect to I :

$$\begin{aligned} A &= a_0 + a_1\beta + a_2\beta^2 + \cdots + a_{n-1}\beta^{n-1}, \\ B &= b_0 + b_1\beta + b_2\beta^2 + \cdots + b_{n-1}\beta^{n-1}, \end{aligned}$$

where $a_i, b_i \in \mathbb{F}_2, i = 0, 1, \ldots, n-1$. Note that $n \geq m+1$ and the sets of coefficients $\{a_i\}$ and $\{b_i\}$ are not unique. Also note that $\beta^n = 1$. Then multiplication operation using the redundant basis I can be given by

$$\begin{aligned} \beta^i \cdot B &= b_0\beta^i + b_1\beta^{i+1} + \cdots + b_{n-i} + \cdots + b_{n-1}\beta^{i-1} \\ &= b_{n-i} + b_{n-i+1}\beta + \cdots + b_0\beta^i + \cdots + b_{n-i-1}\beta^{n-1} \\ &= \sum_{j=0}^{n-1} b_{(j-i)}\beta^j, \end{aligned}$$

where $(j-i)$ denotes that $j-i$ is to be reduced modulo n. Then the product of field elements A and B can be given by

$$\begin{aligned} A \cdot B &= \sum_{i=0}^{n-1} a_i(\beta^i \cdot B) \\ &= \sum_{i=0}^{n-1} a_i \sum_{j=0}^{n-1} b_{(j-i)}\beta^j \\ &= \sum_{j=0}^{n-1} \Big(\sum_{i=0}^{n-1} a_i b_{(j-i)} \Big) \beta^j. \end{aligned} \tag{7.1}$$

If we define $A \cdot B = C \triangleq \sum_{j=0}^{n-1} c_j\beta^j$, then c_j can be given by

$$c_j = \sum_{i=0}^{n-1} a_i b_{(j-i)}, \quad j = 0, 1, \ldots, n-1. \tag{7.2}$$

7.2.3 Redundant Basis and Normal Basis

Normal basis is the most popular basis for hardware implementations of finite field arithmetic since the squaring operation in normal basis is simply a cyclic shift of the coordinates of the elements [25]. The complexity of multiplication under normal basis is minimized for two subclasses of normal basis referred to as optimal normal basis type I and II, which is the main reason that they are often used in practice to implement cryptosystems [30],[33].

For the class of fields where there exists a type I optimal normal basis, redundant basis elements (except the element '1') are a permutation of the optimal normal basis elements [44]. Due to this property, an $(m+1)$-bit redundant basis multiplier can be employed as an m-bit optimal normal basis type I multiplier. This property was used to select the size of the multiplier, so that it can perform operations over optimal normal basis type I.

7.2.4 Multiplier Architecture in Redundant Basis

Different architectures have been proposed for multiplication in redundant basis [44],[32], all of which realize (7.2). In this work, we are mainly interested in an architecture recently proposed in [32], which is shown in Fig. 7.1.

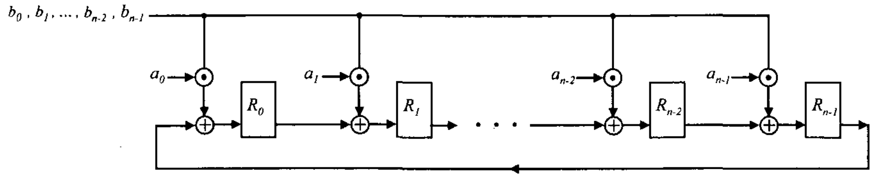


Figure 7.1: High Speed Serial Multiplier in Redundant Basis

In their architecture, all bits of the operand A should be held constant throughout the multiplication operation, while operand B is available in bit-serial fashion. The contents of the n flip-flops should be initialized to zero, while the n output bits of the multiplier can be read from the flip-flops after n clock cycles.

The main advantage of this architecture is its small critical path delay, which is equal to the delay of one XOR gate in addition to the delay of one AND gate. The high regularity of the architecture's structure is another major advantage. It has been proven that the architecture shown in Fig.7.1 has the smallest critical path delay, and the smallest area compared to all other similar multipliers [32].

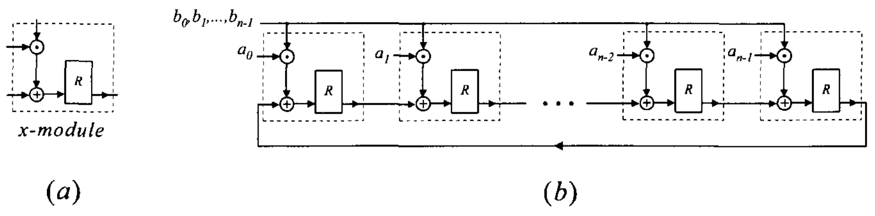


Figure 7.2: (a) x-module Block Detail (b) Multiplier Composed of x-module Blocks

The aforementioned multiplier can be modeled as a series of connected blocks composed of one XOR gate, one AND gate and a flip-flop. We refer to this block as x-module.

The detailed view of the x-module block, and the multiplier composed of a series of x-module blocks is shown in Fig. 7.2.

7.3 Design of the Multiplier Main Building Block (x-module)

Reducing the delay of the AND-XOR function in the x-module will result in an increased multiplication speed. One way to do so is to implement the critical path made out of one AND gate, and one XOR gate in domino logic, [40]. However, careful considerations must be taken into account when designing such circuits [39].

The schematic of the implemented domino AND-XOR function is shown in Fig. 7.3. As shown, the design is quite simple, consisting of 18 transistors. Transistor P1 acts as the pull-up network, charging the node Q during the precharge state. Transistors N2-8 form the pull down network responsible for discharging node Q when the appropriate combinations of inputs exist. N2-8 connect to the evaluate transistor, N9, which opens a path to ground during the evaluate phase. Transistor P3 is the keeper, reducing the charge leakage effect at node Q.

Transistors P0 and N0 create the output NOT stage, providing the output current drawn from the module. Three NOT gates also exist in the module (which are not shown in figure), which generate the complements of the module inputs.

The final layout for the domino logic block is shown in Fig 7.4. The height for the layout was set to be the same as the height of the standard cell technology from which the D-flipflop was selected: $16.50\,\mu m$. The total area for the x-module layout was measured to be equal to $108.24\,\mu m^2$.

The flipflop used in the design of the x-module was selected from the Virtual Silicon CMOS library. Care was taken to select an appropriate flip-flop to interface with our domino-logic cell. A negative-edge triggered flip-flop was needed to maximize the time available for the domino cell to evaluate. Furthermore, it was required to have a hold-time

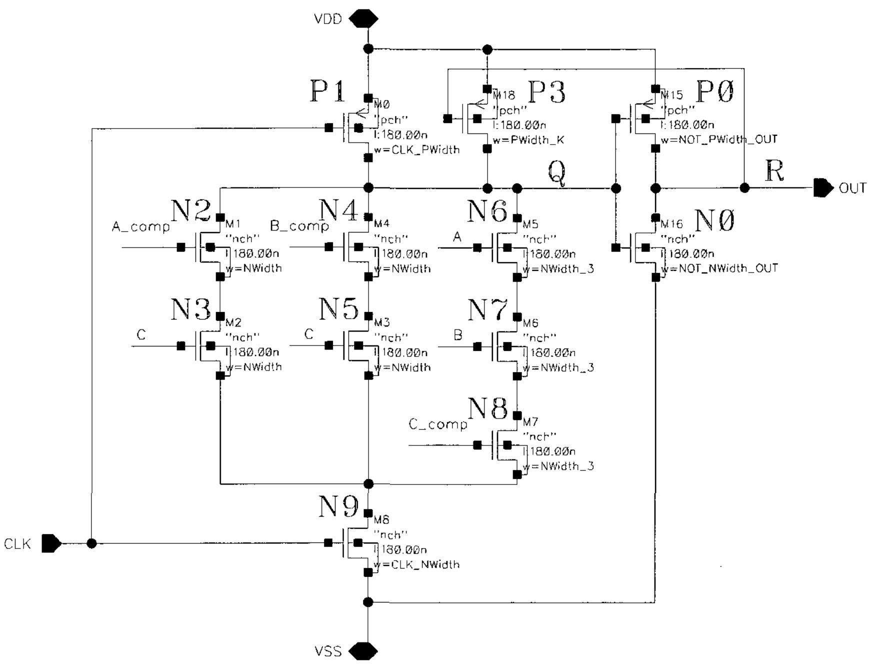


Figure 7.3: AND - XOR Function in Domino Logic

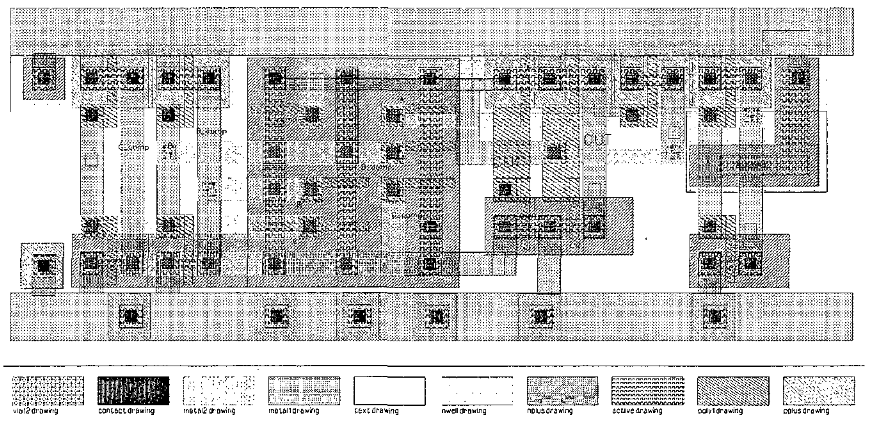


Figure 7.4: Layout for the AND-XOR Function in Domino Logic

less than or equal to zero, as the domino cell's output becomes invalid immediately after the falling edge of the clock.

7.4 Design of a Practical Size Multiplier Using the x-module

For our multiplier, we have selected an input size of 197 bits for two main reasons. First, the size is in the practical range for cryptography applications [18]. Second, there exists an optimal normal basis type I for the field size of 196, for which our multiplier would be applicable [33].

The design has been carried out with Virtuoso Layout Editor and the Cadence Schematic Composer. The design process began by replicating 196 x-module blocks, and connecting them serially. 14 blocks were used in each row, which set the total number of rows needed to 14. One extra x-module block was placed along the side bringing the total to 197.

Also, 14 buffers were selected from the standard CMOS library and carefully connected to generate the clock tree for the multiplier. Input B, was also connected to tree structures of appropriately sized buffers. Other inputs were also correctly buffered to enable high performance while complying with loading requirements. The final layout for the proposed design is shown in Fig. 7.5, its size was measured to be $481.97\,\mu m \times 125.49\,\mu m$.

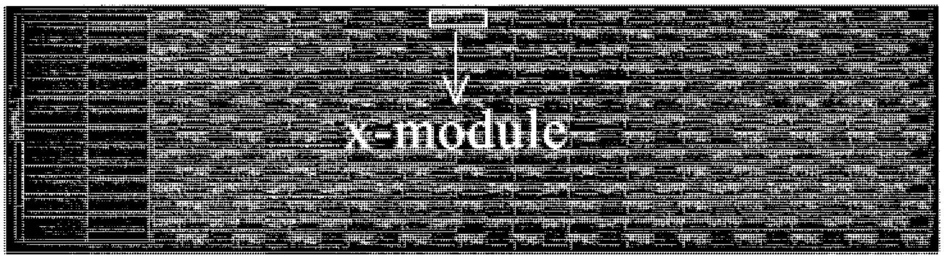


Figure 7.5: Proposed 197 Bit Multiplier Layout

The final layout of the multiplier including all parasitic capacitances was simulated with Cadence's Analog Environment using Spectre. The circuit performed correctly up to a clock rate of $1.82\,GHz$. Simulation waveforms for the clock frequency of $1.72\,GHz$ are shown in Fig. 7.6. In this figure, the first two rows represent the buffered inputs a, and b to the x-module, and the third row represents the third input, c, which is connected to the previous x-module's output. The fourth and fifths rows are the voltage at nodes Q and P of the x-module (shown in Fig. 7.3), while the sixth row indicates the output of the D-flipflop, which is the output of the x-module. Finally, the last row shows the clock output of the clock tree as it enters the x-module.

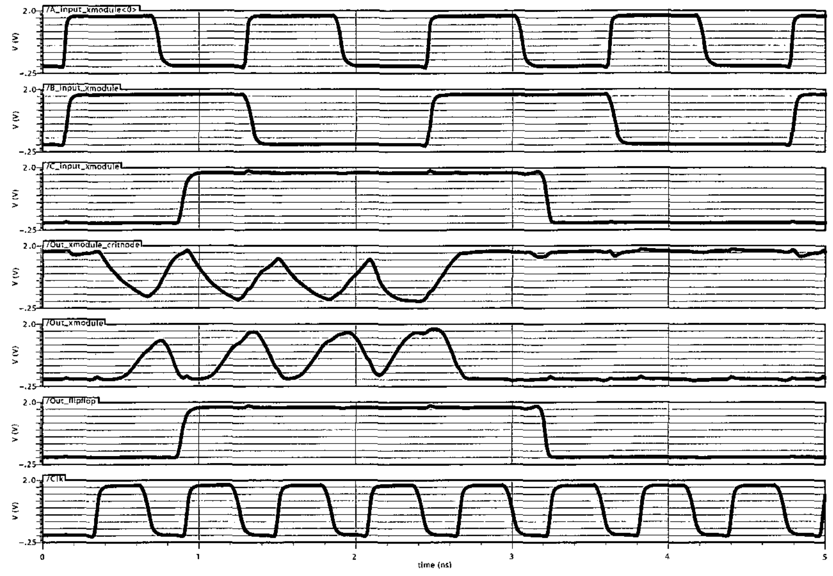


Figure 7.6: Post Place-and-Route Simulation Result of the Proposed 197 Bit Multiplier, from top to bottom: input A, input B, input C, node Q, node R, x-module out, clock

7.5 Design of Practical Size Multiplier Using static CMOS

We began the static CMOS design process by writing the parametrized *C* code to generate the VHDL code describing the multiplier hardware. Using this method, different size multipliers are easily generated by changing parameters in the *C* code. The generated VHDL code was synthesized afterwards to a gate-level netlist using Synopsys' Design Compiler. Next, the gate-level netlist was used for partitioning, placement and routing the multiplier module using Cadence Encounter; the clock tree was also generated using Encounter's Clock Synthesizer.

The post Place-and-Route area of the multiplier was 72083.85 μm^2 which could be clocked up to 748 MHz. The total number of standard cells used was 987, while achieving a maximum gate density of 80%. The final layout for the static CMOS multiplier in shown in Fig. 7.6.

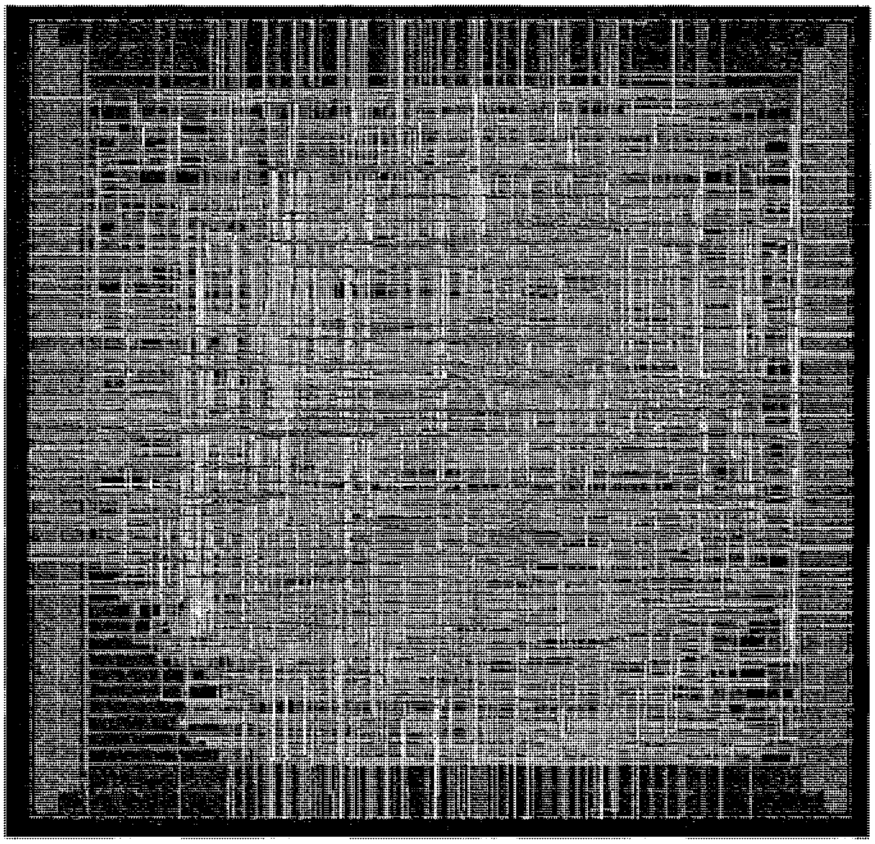


Figure 7.7: Static CMOS 197 Bit Multiplier Layout

7.6 Different VLSI implementation Comparisons

Comparison between the two VLSI implementations are shown in Tab. 7.1. The first row of the table presents the static CMOS implementation from section 7.5, and the second row represents the proposed design using our x-module. As shown, the clock rate increase is 143%, while the area reduction is 16% for the proposed design compared to static CMOS.

Table 7.1: Complexity Comparison between Two VLSI implementations for a 197 Bit Multiplier

Architecture	Area	Clock Frequency
Static CMOS	$72083.85\ \mu m^2$	$748\ MHz$
Proposed design	$60482.41\ \mu m^2$	$1.82\ GHz$

7.7 Conclusions

A new VLSI implementation for a 197 bit finite field multiplier was presented. The proposed design employs a main building block designed in domino logic. The speed improvement was measured to be 143% in comparison to static CMOS implementation, while area reduction was 16%. The post place-and-route design was successfully simulated up to a clock rate of $1.82\,GHz$. Our proposed design has applications in public key cryptography, especially elliptic curve cryptosystems where high speed, large size multipliers are needed.

Chapter 8

High Speed Implementation of a SIPO Multiplier Using Reordered Normal Basis

8.1 Introduction

Polynomial basis is the most widely used basis for hardware and software implementation. Normal basis, on the other hand, is more suitable for hardware implementation due to the simplicity of the squaring operation. In normal basis, the squaring operation is achieved via circular-shifting the element coefficients, which can be implemented in hardware at a very small cost. This leads to a similarly low cost / high speed squaring operation that can be used to accelerate the exponentiation operation by repeated squaring and multiplication via the Fermat theory [14], [21].

There exists two sub-classes of normal basis for which the complexity of the multiplication is minimized. These two classes are referred to as optimal normal basis (ONB) type I and II, which are particularly attractive for cryptography applications [30]. Many different architectures have been proposed for multiplication using these two classes of finite fields such as in [37, 15, 9, 36]. Reordered normal basis is referred to as a certain permutation of a type II optimal normal basis [44, 12].

Not all field sizes are suitable for cryptography applications. According to different standards, different field sizes have been selected based on their suitability for cryptography applications. In this work we have selected the field size of 233, which is recommended in the NIST standard for elliptic curve digital signature standard [34]. There exist a type II optimal normal basis according to the IEEE standard for public key cryptography for this field size as well [33].

In this work, a new VLSI implementation for a Serial-In Parallel-Out finite field multiplier using a reordered normal basis is presented. It is shown that the new implementation operates at a much higher speed than a static CMOS implementation of the same architecture, while significantly reducing the area. This performance advantage is the result of implementing the design as a series connection of a simple block designed and optimized in domino logic, which has a smaller delay/area compared to the equivalent static CMOS realization.

The organization of this chapter is as follows: Reordered normal basis and multiplication using this basis are briefly reviewed in Section 2. In section 3, the design and implementation of the xax-module which is the main building block of the multiplier, is discussed. Section 4 presents the implementation of a 233-bit multiplier using the xax-module as the main building block. Section 5 examines the static CMOS implementation of the same multiplier. In section 6, comparisons between different VLSI implementations are presented. A few concluding remarks are given in section 7.

8.2 A Brief Review of Reordered Normal Basis and Its Arithmetic in $\mathbb{F}_{2^m}$

8.2.1 Reordered Normal Basis Definition

Theorem 8.2.1. [12] Let β be a primitive $(2m+1)^{\text{st}}$ root of unity in $\mathbb{F}_{2^m}$ ($\beta^{2m+1} = 1$) and $\gamma = \beta + \beta^{-1}$ generates a type II optimal normal basis. Then $\{\gamma_i, i = 1, 2, \ldots, m\}$ with $\gamma_i = \beta^i + \beta^{-i} = \beta^i + \beta^{2m+1-i}$, $i = 1, 2, \ldots, m$, is also a basis in $\mathbb{F}_{2^m}$.

It has been shown that the basis $\{\gamma_i, i = 1, 2, \ldots, m\}$ is actually a permutation of the normal basis $\{\gamma^{2^i}, i = 0, 1, \ldots, m-1\}$ [44]. We denote the basis $I_2 = [\gamma_1, \gamma_2, \ldots, \gamma_m]$ as the reordered normal basis following [44]. Note that reordered normal basis not only offers free squaring but also can avoid modulo reduction step in a multiplication operation.

8.2.2 Reordered Normal Basis Multiplication

Assume that A and B are two arbitrary elements in $\mathbb{F}_{2^m}$ represented with respect to reordered normal basis $I = [\gamma_1, \gamma_2, \ldots, \gamma_m]$ and $C = A.B$,

$$A = \sum_{i=1}^{m} a_i \gamma_i \quad , \quad B = \sum_{i=1}^{m} b_i \gamma_i \quad , \quad C = \sum_{i=1}^{m} c_i \gamma_i .$$

To facilitate multiplication, function $s(i)$ has been defined, mapping set of integers to the set $\{0, 1, \ldots, 2m+1\}$ [44].

$$s(i) \triangleq \begin{cases} i \bmod 2m+1, & \text{if } 0 \leqslant i \bmod 2m+1 \leqslant m, \\ 2m+1 - i \bmod 2m+1, & \text{otherwise.} \end{cases}$$

Next compute $\gamma_j A$ where $1 \leqslant j \leqslant m$,

$$\gamma_j A = \gamma_j \sum_{i=1}^{m} a_i \gamma_i = \sum_{i=1}^{m} a_i [\gamma_{s(i+j)} + \gamma_{s(i-j)}]. \tag{8.1}$$

And also note that [44],

$$\sum_{i=1}^{m} a_i \cdot [\gamma_{s(i+j)} + \gamma_{s(i-j)}] = \sum_{i=1}^{m} [a_{s(i+j)} + a_{s(i-j)}] \cdot \gamma_i.$$

Then

$$\begin{aligned} C &= A \sum_{j=1}^{m} b_j \gamma_j \\ &= \sum_{j=1}^{m} b_j (\gamma_j . A) \\ &= \sum_{j=1}^{m} b_j \sum_{i=1}^{m} a_i [\gamma_{s(i+j)} + \gamma_{s(i-j)}] \\ &= \sum_{j=1}^{m} b_j \sum_{i=1}^{m} [a_{s(i+j)} + a_{s(i-j)}] \gamma_i \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} b_j [a_{s(i+j)} + a_{s(i-j)}] \gamma_i \end{aligned} \tag{8.2}$$

Note that it was assumed that $a_0 = 0$ [44]. The value for c_i can be calculated as follows:

$$\begin{aligned} c_i &= \sum_{j=1}^{m} b_j [a_{s(i+j)} + a_{s(j-i)}], \\ &= \sum_{j=1}^{m} a_j [b_{s(i+j)} + b_{s(j-i)}], \quad i = 1, 2, \ldots, m. \end{aligned} \tag{8.3}$$

8.2.3 A Review of Existing Architectures for ONB Type II Multiplication

As mentioned in the previous section, ONB type II and reordered normal basis representation of an element are simple permutations of each other. Therefore, reordered normal basis multipliers can be used as optimal normal basis type II multipliers and vice versa. Many different architectures have been proposed for multiplication in normal basis. The complexity of different multipliers available in open literature are listed in table 8.1.

Multiplier	# AND	# XOR	# flip-flops	# Clock Cycles	Critical Path Delay	Basis
MO [22]	$2m-1$	$2m-2$	$2m$	m	$T_A + (\lceil \log_2(2m-1) \rceil)T_X$	Normal
IMO [11]	m	$2m-2$	$2m$	m	$T_A + (1 + \lceil \log_2 m \rceil)T_X$	Normal
GG [15]	m	$\frac{3m-1}{2}$	$3m$	m	$T_A + 3T_X$	Normal
Feng [9]	$2m-1$	$3m-2$	$3m-2$	m	$T_A + 4T_X$	Normal
Agnew [1]	m	$2m-1$	$3m$	m	$T_A + 2T_X$	Normal
XEDS [36]	$2m-1$	$2m-2$	$2m$	m	$T_A + (\lceil \log_2(2m-1) \rceil)T_X$	Normal
AEDS [36]	m	$3m-3$	$2m$	m	$T_A + (\lceil \log_2(2m-1) \rceil)T_X$	Normal
w-SMPOI [37]	$\lfloor \frac{m}{2} \rfloor + 1$	$3m$	$3m$	m	$T_A + 3T_X$	Normal
w-SMPOII [37]	m	$m + \lfloor \frac{m}{2} \rfloor$	$3m$	m	$T_A + 3T_X$	Normal
PISO [44]	m	$2m-1$	$2m+1$	m	$T_A + (1 + \log_2 m)T_X$	Reordered
SIPO [44]	m	$2m$	$3m+1$	m	$T_A + 2T_X$	Reordered

Table 8.1: Complexities Comparison Between Type II ONB / Reordered Normal Basis Multipliers

In this table the first row represents the famous Massey-Omura normal basis multiplier, and the second row represents the improved version proposed by Gao and Sobelman. The third row shows the architecture proposed by Geisellman and Gollman, which exploits the symmetry property of the normal basis. The fourth and the fifth rows list the architectures proposed by Feng and Agnew respectively. The next two rows represent Reyhani's architecture XOR Efficient Digit Serial and AND Efficient Digit Serial multipliers, while the next two rows show the Sequential Multipliers with Parallel Output type I and II, also proposed by Reyhani. The last two rows list the Serial-In Parallel-Out and Parallel-In Serial-Out reordered normal basis multipliers recently proposed by Wu. As can be seen from the table, the Agnew architecture (fifth row) and Serial-In Parallel-Out architecture (last row) have the smallest critical path delay compared to other architectures.

In this work, we are mainly interested in the SIPO architecture proposed by in [44]. Our interest is mainly due to two specific properties of the architecture. First, the critical path delay is the minimum among all other architectures, except the Agnew architecture, which has similar complexity. Second, the architecture has a very regular structure which greatly simplifies the VLSI implementation.

8.3 Design of A Practical Size Multiplier Using xax-module

8.3.1 Multiplier Size Selection

According to [6], to provide a sufficient level of security, the field size is required to be at least 160 bits for an elliptic curve cryptosystem. Some fields have been recommended by different standards for use, while others were banned. In this work, we have selected the field size of 233 for three reasons. First, it is in the range suitable for Elliptic Curve Cryptography. Second, there exists a type II ONB representation, meaning that the reordered normal basis also exists [33]. Finally, the field size is recommended by the National Institute of Standards and technology (NIST) as their Digital Signature Standard (DSS) in the Elliptic Curve Digital Signature algorithm (ECDSA). A few other field sizes were also recommended by the standard, but the field size of 233 is the only one such that there exist a type II ONB.

8.3.2 Selected Multiplier Architecture

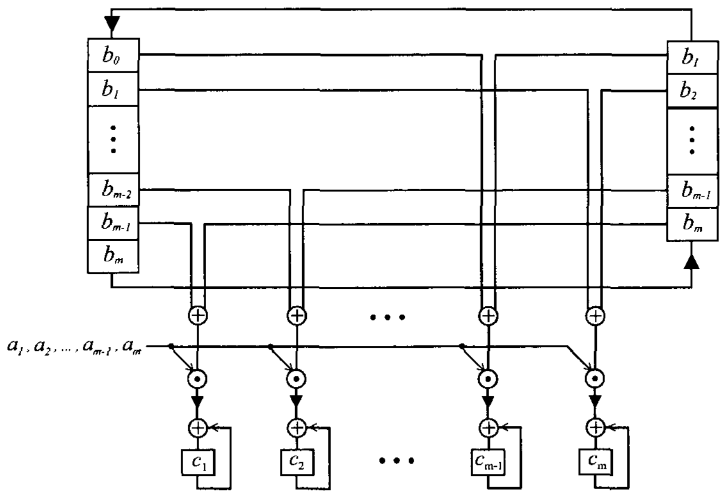


Figure 8.1: Serial-In Parallel-Out Reordered Normal Basis Multiplier

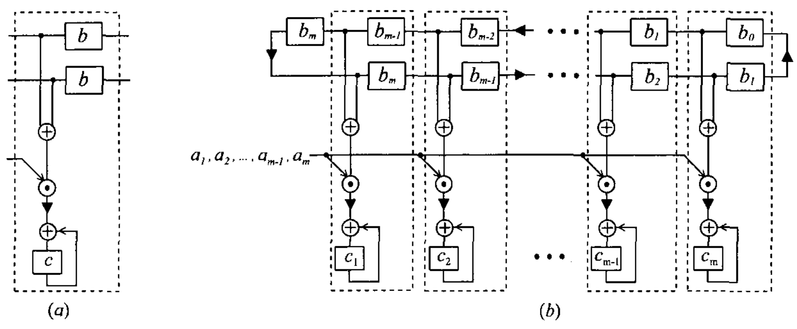


Figure 8.2: xax-module and the SIPO Multiplier Composed of xax-module

The SIPO multiplier proposed in [44] is shown in Fig. 8.1. The architecture has been redrawn in Fig. 8.2 to show its regularity, and to showcase the fact that it can implemented as a serial connection of a single module; it is shown in the figure inside the box. This module, which is made out of three flip-flops, two XOR gates and an AND gate is referred to as an xax-module. The two XOR gates, in addition to the AND gate, create the critical path of the multiplier. One way to reduce the critical path delay is to implement the XOR-AND-XOR function using domino logic [40]. However, careful consideration should be taken into account when designing such circuity [39]. The 233-bit multiplier can be made by replicating the xax-module once for every bit of the multiplier, and serially connecting them together.

8.3.3 Design and Implementation of the xax-module

The schematic shown in Fig 8.3 implements the XOR-AND-XOR function in domino logic; it is responsible for implementing the function $((b1 \oplus b2) \cdot a) \oplus c)$. We have reduced the number of transistors in the architecture, while maintaining the same functionality, by using the schematic shown in Fig. 8.4 for our implementation. In this figure, the design is quite simple, consisting of 17 transistors (as opposed to 21 in the original design).

In Fig. 8.4 transistor P1 acts as the pull-up network, charging the node Q during the precharge state. Transistors N2-13 form the pull down network responsible for discharging node Q when the appropriate combinations of inputs exist. N2-13 connect to the evaluate transistor, N1, which opens a path to ground during the evaluate phase. Transistor P2 is the keeper, reducing the charge leakage effect at node Q.

Transistors P0 and N0 create the output NOT stage, providing the output current drawn from the module. Four NOT gates also exist in the module (not shown in the figure), which generate the complements of the xax-module's inputs.

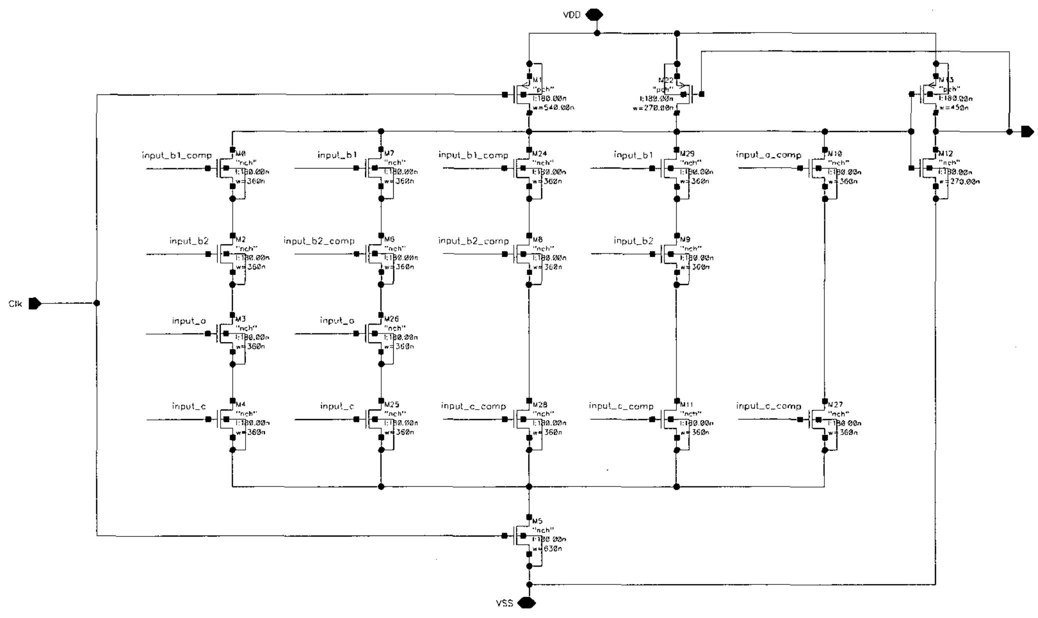


Figure 8.3: XOR-AND-XOR Function Implementation in Domino Logic

The final layout for the domino logic block is shown in Fig 8.5. Note that this figure is rotated 90 degrees to the left for better readability. The three large vertical stripes, from left to right, are VDD, VSS, and VDD wires. The section on the left are the four NOT gates used to create the complements of the inputs, while the section on the right implements the schematic shown in Fig. 8.4. The height for the layout was set to be three times the height of a standard-cell: 19.962 μm, since three D flip-flops were to be connected to each domino cell. The total area for the xax-module, including the area of three D flip-flops, was measured to be 449.18 μm^2. The area of the domino-cell on its own is 198.86 μm^2.

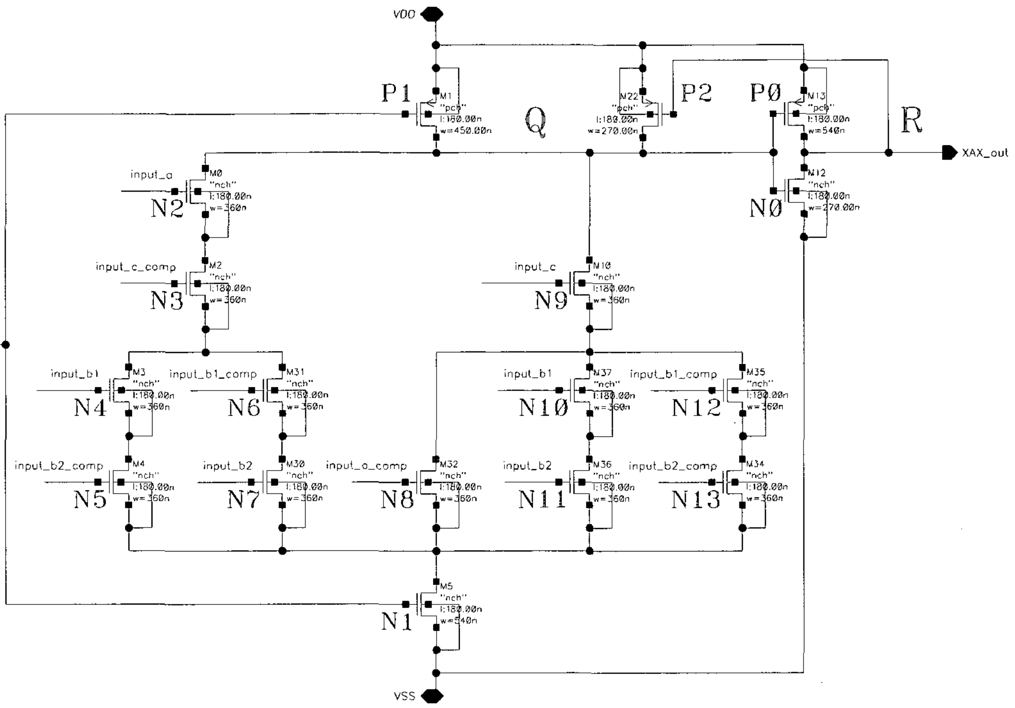


Figure 8.4: A New XOR-AND-XOR Function Implementation in Domino Logic

The flipflop used in the design of the xax-module was selected from the Virtual Silicon CMOS library. Care was taken to select an appropriate flip-flop to interface with our domino-logic cell. A negative-edge triggered flip-flop was needed to maximize the time available for the domino cell to evaluate. Furthermore, it was required to have a hold-time less than or equal to zero, as the domino cell's output becomes invalid immediately after the falling edge of the clock.

8.3.4 Design and Implementation of the 233-bit Multiplier Using the xax-module

The block diagram design of the 233-bit multiplier is shown in Fig. 8.6. As can be seen, the main part of multiplier architecture can be implemented by connecting the xax-modules

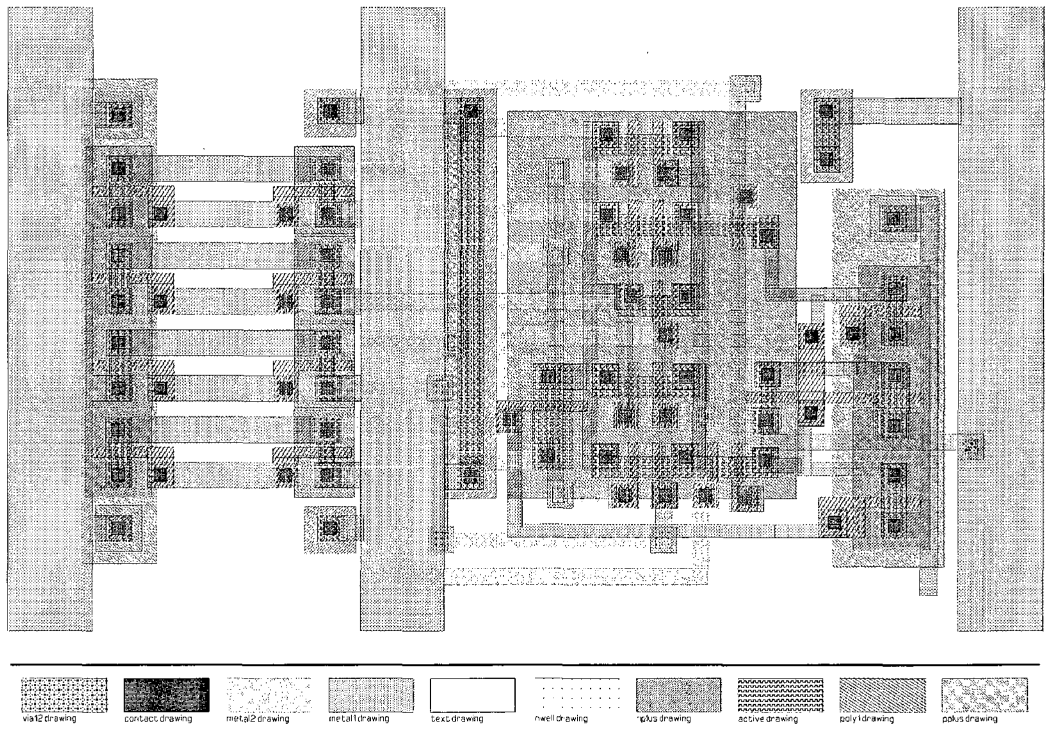


Figure 8.5: Layout for the XOR-AND-XOR Function in Domino Logic

serially. We have used 13 xax-modules in each row, for a total of 18 rows to create the complete multiplier. This row/column distribution was chosen to give the design an even aspect ratio, which is typically desirable when floorplanning. There exists one additional flip-flop, which holds $b0$ in Fig. 8.6, that needs to be added to the design.

We have added one extra xax-module, referred to as the load-module in Fig. 8.6, which is used to load the coefficients of input b serially into the multiplier when the load signal is enabled. In order to achieve this, we have connected the a input of the load-module to the load enable signal, and the c input to the external input, Ext_int. Input $b1$ of the load-module was connected to the output of the previous stage and, input $b2$ was shorted to ground. Since the xax-module implements the function $((b1 \oplus b2) \cdot a) \oplus c)$, the output of the load-module would be $((Int\_input \cdot load) \oplus Ext\_input)$. This can then be used to

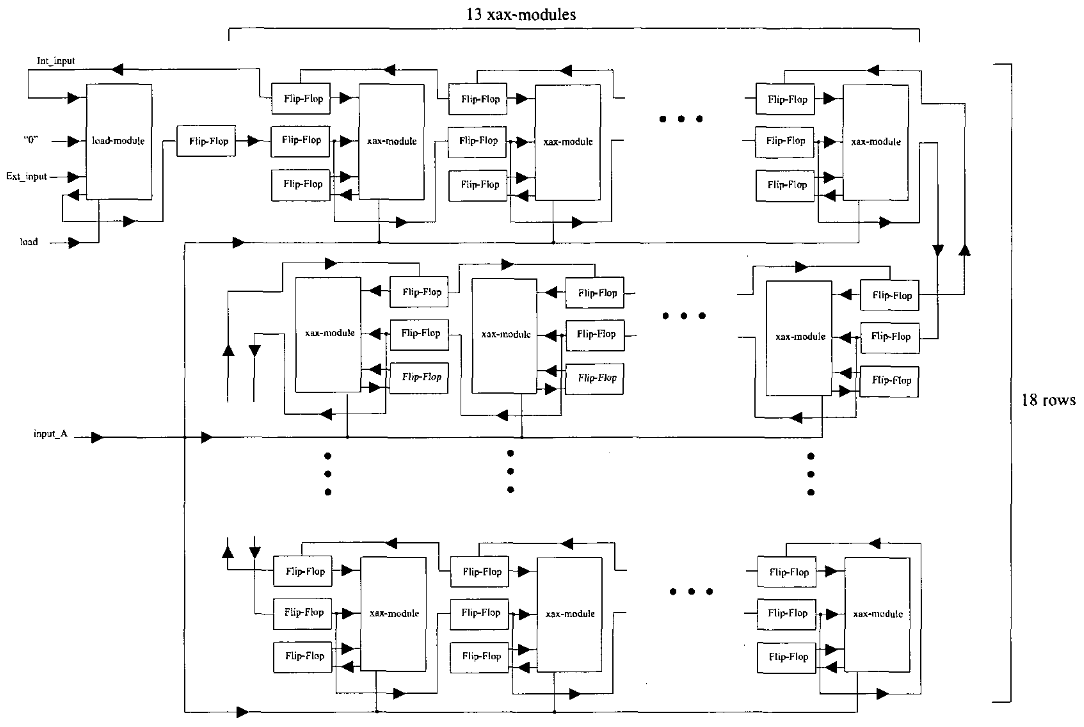


Figure 8.6: Block Diagram of the 233-bit Multiplier

load the coefficients of operand B into the circular shift register. Table 8.2 tabulates the two combinations that can be used to load the data into the multiplier. When the load signal is "0", the output of the xax-module would be equal to the Extinput, and when load is "1" and Ext_input is "1", the shift register acts as the circular shift register.

Table 8.2: load-module Input/Output Characteristics

Load	Ext_input	Int_input	Output
0	Ext_input	X	Ext_input
1	0	Int_input	Int_input

A tree-structure of similarly-sized buffers was used to generate the clock tree for the multiplier's clock signal. The same was done for the input a, since it is a high fan out net that connects to every xax-module in the design. The full layout of the multiplier is shown

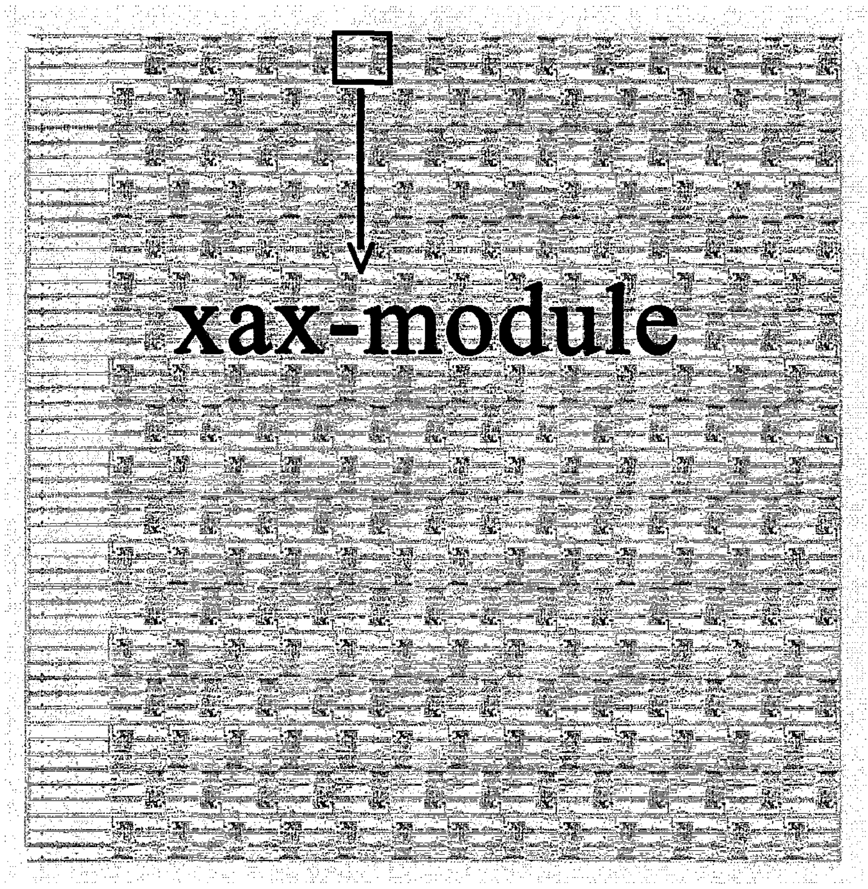


Figure 8.7: 233-bit Proposed Multiplier Layout

in Fig. 8.7.

The final layout of the multiplier including all parasitic capacitances was simulated in Cadence's Analog Environment using Spectre. The circuit performed correctly up to a clock rate of $1.587\,GHz$. Simulation voltage waveforms for the clock frequency of $1.54\,GHz$ are shown in Fig. 8.5.

In this figure, the first row is the buffered input a. Rows two, three, and four show the signals $b1$, $b2$, and c as they exit the xax-module's flip-flops and enter the XOR-AND-XOR function's inputs. Row five is the output of the xax-module (node R), while row six is the voltage at node Q in Fig. 8.4. Finally, row seven shows the multiplier's clock waveform as it exits the buffer-tree and enters the xax-modules. All 16 possible input combinations of a, $b1$, $b2$, and c, were tested and verified to give the correct output when determining the maximum operating frequency.

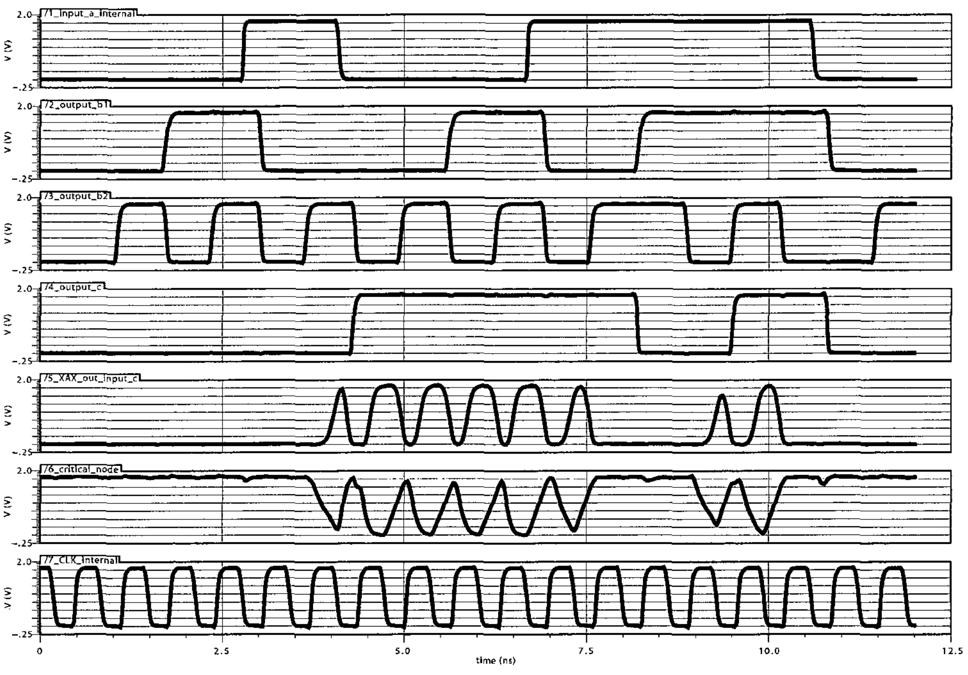


Figure 8.8: Post Place-and-Route Simulation Result of the Proposed 233-bit Multiplier, from top to bottom: input a, input $b1$, input $b2$, input c, node R, node Q, clock

8.4 Design of the 233-bit Multiplier Using Static CMOS

The static CMOS multiplier implements the same functionality as the domino logic design. Similar to the domino-logic design, the static CMOS version also incorporates a load-module (implemented in static CMOS) to serially load an external input into the multiplier.

We began the static CMOS design process by writing the parametrized C code to generate the VHDL code describing the multiplier in hardware. Using this method, different size multipliers are easily generated by changing parameters in the C code. The VHDL code was simulated using Cadence's NCSim to confirm that the architectural code was functioning correctly. Afterwards, the VHDL code was synthesized to a gate-level netlist using Synopsys' Design Compiler. Compilation parameters were always chosen to maximize the operating frequency of our design; the critical path delay at this stage was $1.11\,ns$.

Next, the generated gate-level netlist was simulated again using Cadence's NCSim to confirm that the functionality did not change during the synthesis stage. Then the verified gate-level netlist was used for partitioning, placement, and routing using Cadence's Encounter; the clock tree was also generated using Encounter's Clock Synthesizer. The worst negative slack calculated by Encounter after the place-and-route steps was reported to be $0.145\,ns$, bringing the total critical path delay to $1.255\,ns$.

The post place-and-route area of the multiplier was $216737.136\,\mu m^2$ which could be clocked up to $796\,MHz$. The total number of standard cells used was 1965, while achieving a maximum gate density of 80%. The final layout for the static CMOS multiplier is shown in Fig. 8.5.

8.5 A Comparison of Different VLSI Implementations

Comparison between the two VLSI implementations are shown in table 8.3. The first row of the table presents the static CMOS implementation from section 8.4, and the second row represents the proposed design using our xax-module. As shown, the clock rate increase is 99%, while the area reduction is 49% for the proposed design compared to static CMOS.

If we define area times delay as a performance metric, we can conclude that our proposed design is almost four times more efficient than the static CMOS design.

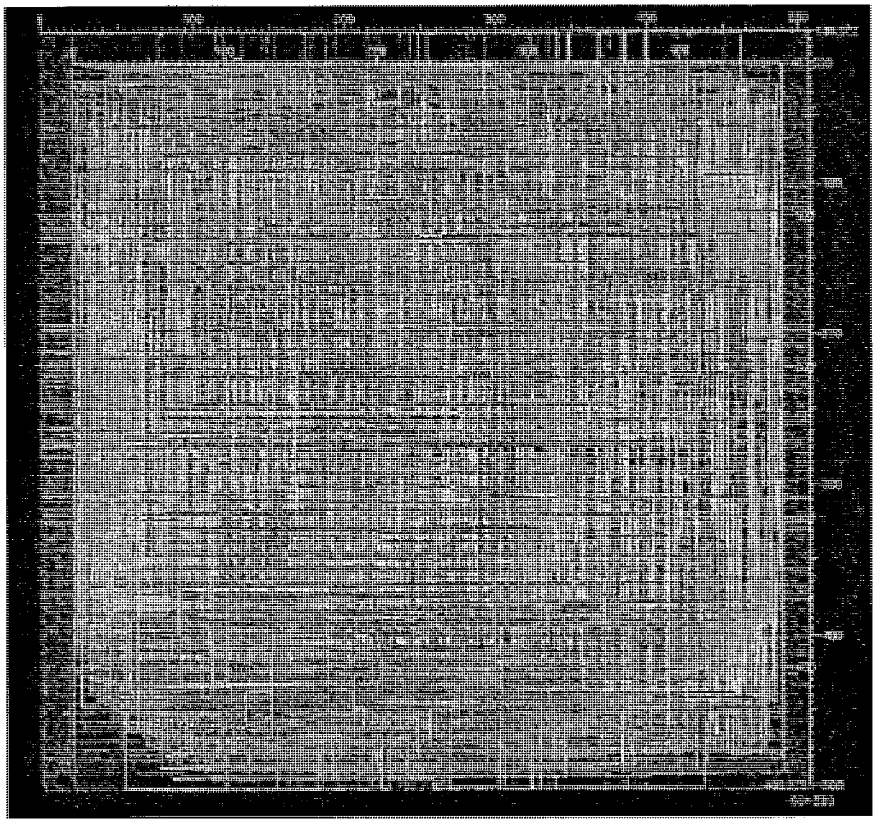


Figure 8.9: Static CMOS 233-bit Multiplier Layout

Table 8.3: Complexity Comparison Between Two VLSI Implementations for a 233-bit Multiplier

Architecture	Area	Clock Frequency
Static CMOS	216737.136 μm^2	796 MHz
Proposed design	109644.819 μm^2	1.587 GHz

8.6 Conclusions

A new VLSI implementation for a 233-bit finite field multiplier was presented. The proposed design employs a main building block designed in domino logic. The speed improvement was measured to be 99% in comparison to static CMOS implementation, while area reduction was 49%. The final design was successfully simulated up to a clock rate of $1.587\,GHz$. Our design's field size is currently recommended by the NIST standard in their Elliptic Curve Digital Signature Standard, rendering it a desirable building block in elliptic curve cryptosystem designs.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

Finite field is a set of finite elements where one can add, subtract, multiply, and divide such that properties of associativity, distributivity, and commutativity are satisfied. Finite fields have important applications in error control coding and cryptography. Two different types of finite field are commonly used in practice: prime field $\mathbb{F}_p$, and the binary field $\mathbb{F}_{2^m}$. Binary field is an extension of the prime field, $\mathbb{F}_2$, which contains 2^m elements. Binary fields are attractive for high speed cryptography applications since they are suitable for hardware implementation. In $\mathbb{F}_{2^m}$, addition is nothing but the exclusive-oring of two binary vectors. Multiplication is more complicated, while division or inversion can be broken down into a series of consecutive multiplication operations. In practice, the finite field multiplier becomes the key arithmetic unit for any system based on finite field computations.

Efficiency of finite field multiplication depends on the choice of the basis to represent field elements. Bases that have been used for realizing finite field multipliers include polynomial basis, normal basis (NB), dual bases, triangular basis, redundant representation or redundant basis, and their variations (*i.e.*, shifted polynomial basis).

In normal basis, the complexity of multiplication is measured with the multiplication matrix. For a binary extension field, the multiplication matrix entries are either zero or one, and the number of ones inside the multiplication matrix is referred to as normal basis complexity. The normal basis in GF(2^m) for which the complexity achieves its minimum, $2m - 1$, is referred to as the optimal normal basis (ONB). Two types of optimal normal bases have been found which are referred to as type I and type II optimal normal basis.

Hardware implementation of finite field multipliers can usually be divided into three categories. In the first category there are bit-level or bit-serial multipliers. A bit-level multiplier takes m clock cycles to finish one multiplication in a binary field of size m. The multipliers in this class are considered to have low power consumption, occupy a small area of silicon, and operate slowly for large field sizes. The second category are bit-parallel, or full-parallel multipliers. A full parallel multiplier takes one clock cycle to finish one field multiplication. These multipliers are not usually economical for implementation since they require a large silicon area and high bandwidth for input and output ports. The third category are word-level or digit-level finite field multipliers, which are the most commonly implemented in practice. A word-level multiplier takes w clock cycles, $1 \leqslant w \leqslant m$, to finish one multiplication operation in $\mathbb{F}_{2^m}$. The value of w can be selected by designer to set the trade off between area and speed according to the application. Decreasing the value of w will result in faster and larger multipliers while increasing w will make smaller and slower multipliers. Note that bit-level and full parallel multipliers can be viewed as special cases of word-level multipliers for $w = m$ and $w = 1$ respectively.

In this thesis, a number of high speed word-level architectures for finite field multiplication have been proposed. Most of the proposed architectures have been implemented in hardware, using FPGA or standard CMOS platforms. It has been shown that proposed word-level architectures are more efficient compared to optimal normal basis type I or type

II architectures for the classes of field in which they exist.

Also, different VLSI implementations for finite field multipliers were explored, which resulted in more efficient implementations for some of the regular architectures. The new implementations use a simple module designed in domino logic as the main building block for the multiplier. Significant improvements were achieved while designing practical sized multipliers using the proposed methodology.

9.2 Future Work

More research can be conducted in finding more efficient algorithms and architectures for finite field multiplication in optimal normal basis type I or type II classes of fields. Special attention should be paid to full parallel architectures for two primary reasons. First, advances in VLSI design now allow for large parallel systems to be realized as a chip. Secondly, the need for faster multipliers to further increase the encryption and decryption processes is of greater importance as more data must be encrypted. Note that the main challenges in designing such multipliers would be in managing the power and I/O requirements, which are different from word-level architectures.

Further research efforts should be devoted to other classes of normal basis, including the Gaussian normal basis types III, and IV. These classes of fields are considered to be the most efficient, after optimal normal basis types I and II, for cryptography applications. Some examples of this are the binary field sizes of 163 and 409 which are recommended by National Institute of Standards and Technology for elliptic curve cryptography.

Another research area worth exploring is low power VLSI design and implementation of the bit-level multiplier architectures. The low power and small area requirements of such designs makes them attractive candidates for resource-constrained applications such as smart cards, cellular phones, and personal digital assistants.

References

[1] G.B. Agnew, R.C. Mullin, I.M. Onyszchuck, and S.A Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3:63–79, 1991.

[2] G.B. Agnew, R.C. Mullin, and S. Vanstone. Fast exponentiation in F_{2^n}. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 251–255, New York, NY, 1988. Springer.

[3] E.R. Berlekamp. Bit-serial reed-solomon encoders. *IEEE Trans. on Information Technology*, 28(6):869–874, November 1982.

[4] T. Beth and D. Gollman. Algorithm engineering for public key algorithms. *IEEE Journal on Selected Areas in Communication*, 7(4):458–465, May 1989.

[5] Taiwan Semiconductor Manufacturing Company. $0.18\mu m$ TSMC cmos technology standard cell library, September 1999.

[6] Certicom Corporation. Current public-key cryptographic systems, 2000. White Paper.

[7] G. Drolet. A new representation of elements of finite fields GF(2^m) yielding small complexity arithmetic circuits. *IEEE Trans. on Computers*, 47(9):938–946, September 1998.

[8] H. Fan and M.A. Hasan. Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$. *IEEE Transactions on Circuits and Systems I*, 53:2606–2615, 2006.

[9] M. Feng. A VLSI architecture for fast inversion in $GF(2^m)$. *IEEE Trans. Computers*, 38(10):1383–1386, October 1989.

[10] R. Furness, S.T. Fenn, and M.Benaissa. Multiplication using the triangular basis representation over $GF(2^m)$. In *Global Telecommunications Conference, 1996. GLOBECOM '96*, volume 2, pages 788–792, November 1996.

[11] L. Gao and G.E. Sobelman. Improved vlsi designs for multiplication and inversion in $GF(2^m)$ over normal bases. In *Proceedings of the 13th Annual IEEE ASIC/SOC Conference*, pages 97–101, September 2000.

[12] S. Gao and S. Vanstone. On orders of optimal normal basis generators. *Mathematics of Computation*, 64(2):1227–1233, July 1995.

[13] S. Gao, J. von zur Gathen, and D. Panario. Gauss periods and fast exponentiation in finite fields. *Lecture Notes in Computer Science*, 911:311–322, 1995.

[14] S. Gao, J. von zur Gathen, D. Panario, and V. Shoup. Algorithms for exponentiation in finite fields. *Journal of Symbolic Computation*, 29:879–889, 2000.

[15] W. Geiselmann and D. Gollmann. Symmetry and duality in normal basis multiplication. In *Proceedings of the Applied Algebra, Algebraic Algorithms, and Error Correcting Codes Symposium (AAECC-6)*, pages 230–238, July 1988.

[16] W. Geiselmann and D. Gollmann. Self-dual bases in F_{q^n}. *Designs, Codes and Cryptography*, 3:333–345, 1993.

[17] W. Geiselmann and R. Steinwandt. Redundant representation of GF(q^n) for designing arithmetic circuits. *IEEE Trans. on Computers*, 52(7):1848–1853, July 2003.

[18] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, New York, NY, 2003.

[19] M.A. Hasan and V.K. Bhargava. Low complexity architecture for exponentiation in gf(2^m). *Electronic Letters*, 28(21):1984–1986, October 1992.

[20] M.A. Hasan, M.z. Wang, and V.K. Bhargava. A modified massey-omura parallel multiplier for a class of finite fields. *IEEE Trans. on Computers*, 42(10):1278–1280, October 1993.

[21] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *nformation and Computation*, 78:171–177, September 1988.

[22] J.L.Massey and J.K.Omura. U.S. Pat. 4587627: Computational method and epparatus for finite field arithmetic, September 1986.

[23] R. Katti. Low complexity multiplication in a finite field using ring representation. *IEEE trans. on Computers*, 52(4):418–427, April 2003.

[24] C.K. Koc and B. Sunar. Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. *IEEE Trans. on Computers*, 47(3):353–356, March 1998.

[25] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, Cambridge, England, 1997.

[26] R. Lidl and H. Niederreiter. *Finite Fields (Encyclopedia of Mathematics and its Applications).* Cambridge University Press, Cambridge, England, 2008.

[27] E.D. Mastrovito. *Architectures for Computations in Galois Fields.* PhD thesis, Linköping University, Linköping, Sweden, 1991.

[28] A. Menezes, P. v. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC-Press, 1996.

[29] A.J. Menezes, I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, and T. Yaghoobian. *Applications of Finite Fields.* The Springer International Series in Engineering and Computer Science, New York, NY, 1993.

[30] R.C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R.M. Wilson. Optimal normal bases in GF(p^n). *Discrete Applied Mathematics*, 22:149–161, February 1989.

[31] A.H. Namin, H. Wu, and M. Ahmadi. Comb architectures for finite field multiplication in F_{2^m}. *IEEE Trans. on Computers*, 56:909–916, July 2007.

[32] A.H. Namin, H. Wu, and M. Ahmadi. A new finite field multiplier using redundant representation. *IEEE Trans. on Computers*, 57:716–720, May 2008.

[33] Institute of Electrical and Electronics Engineers. IEEE standard specifications for public-key cryptography, August 2000.

[34] National Institute of Standard and Technology. Digital signature standards, Jan 2000.

[35] A. Reyhani-Masoleh and M.A. Hasan. A new construction of massey-omura parallel multiplier over $GF(2^m)$. *IEEE Trans. on Computers*, 51(5):511–520, May 2002.

[36] A. Reyhani-Masoleh and M.A. Hasan. Efficient digit-serial normal basis multipliers over GF(2^m). *IEEE Trans. on Computers, special issue on cvryptographic hardware and embedded systems*, 52(4):428–439, April 2003.

[37] A. Reyhani-Masoleh and M.A. Hasan. Low complexity word-level sequential normal basis multipliers. *IEEE Trans. on Computers*, 54(2):98–110, February 2005.

[38] J.H. Silverman. Fast multiplication in finite fields GF(2^n). *Lecture Nores in Computer Science, Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, 1717:122–134, Aug 1999.

[39] P. Srivastava, A. Pua, and L. Welch. Issues in the design of domino logic circuits. In *Proceedings of the 8th Great Lakes Symposium on VLSI*, pages 108–112, 1998.

[40] J.P. Uyemura. *CMOS Logic Circuit Design.* Springer, New York, NY, 1999.

[41] C.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 Trans. on Computers*, C-34(8):709–717, August 1985.

[42] J.K. Wolf. Efficient circuits for multiplying in GF(2^m) for certain values of m. *Discrete Mathematics*, 106-107:497–502, September 1992.

[43] H. Wu and M.A. Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Trans. on Computers*, 47(8):883–887, August 1998.

[44] H. Wu, M.A. Hasan, I.F. Blake, and S. Gao. Finite field multiplier using redundant representation. *IEEE Trans. on Computers*, 51(11):1306–1316, November 2002.

VITA AUCTORIS

Ashkan Hosseinzadeh Namin was born in Tehran, IRAN, on September 21, 1979. He received his BSc degree in Electrical Engineering from Isfahan University of Technology, Isfahan, Iran, in 2002, and the MSc degree in Electronics from Sharif University of Technology, Tehran, Iran in 2004. Since September 2004, he has been pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, On, Canada. His research interests include digital and analog integrated circuits, architectures in finite fields, and hardware implementation of cryptosystems.