Kingdom of Saudi Arabia

Ministry of Education

Umm Al-Qura University

Colleage of Computer and Information Systems

Department of Computer Science and Engineering

Design and Implementation of a Compact *GF(2^m)* Optimal Normal Basis Field Arithmetic Unit

MASTERS DEGREE

BY

HUSAM IBRAHIM RASHAD ALDOOBI

43580349

UMM ALQURA UNIVERSITY

SUPERVISED BY

Prof. Turki Al-Somani

Academic Year 1441/2020

Design and Implementation of a Compact $GF(2^m)$ Optimal Normal Basis Field Arithmetic Unit

Signature of Author …….

Committee Members	**Signature and Date**
Prof. Turki Al-Somani	……..
Prof. Adnan Qutub	..……
Dr. Fahad Al-Dosari	……..

Date of Degree: Spring 2020

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Prof. Turki Al-Somani and I am deeply indebted to him. The door of his office was always open whenever I ran into a trouble spot or had a question about my research or writing. He was very patient with me and always encourages me during the whole process. I would like also to acknowledge the continuous support of the previouse Dean of College of Computer and Information Systems (CIS) and the current Dean of Information Technology (IT) at Umm Al-Qura University (Dr. Fahad Al-Dosari), the Dean of CIS (Dr. Majid Al-Qithami), the Vice Dean of CIS for Research and Graduate Studies (Dr. Waleed Al-Asmari), the Chairman of the Department of Computer Engineering (Dr. Mohammed Sinky), and all faculty members and staff in the CIS.

DEDICATION

This thesis is dedicated to my mother who passed away couple of months ago leaving a huge hole in my heart; may Allah bless her soul. This work is also dedicated to my wife, who has been a constant source of support and encouragement during the challenges of graduate school and life. This work is also dedicated to my father and my two children.

ABSTRACT

This thesis proposes the design of an area-efficient compact optimal normal basis field arithmetic unit (FAU) utilizing the common parts between the Massy-Omura multiplier and the Itoh - Tsugii inverter. The field arithmetic operations include addition, multiplication, and inversion. Addition can be easily implemented as an XOR of the corresponding vectors. Multiplication typically requires more computational time than addition, and it has more circuit complexity. Multiplicative inversion can be conducted by repeatedly applying the multiplication squaring algorithm. The design showed decreased hardware complexity and a decrease in the number of inputs compared to the standard approach, which makes the design very attractive when implementing elliptic curve cryptosystems in resource-constrained devices such as, smart cards, radio-frequency identification (RFID), and wireless sensor networks. The design was initially run on 173-bit input; it was then adjusted to run on 233, 350, and 515-bit inputs. The proposed design was coded using VHDL on Xilinx's ISE design suit 14.5 and simulated on an Artix7 XC7A200T field-programmable gate array (FPGA).

ملخص الرسالة

الاسم كاملا: حسام إبراهيم رشاد الدوبي

عنوان الرسالة: تصميم وبناء وحدة حساب مدمجة عادية الأسس

التخصص: علوم وهندسة الحاسب الآلي

تاريخ الدرجة العلمية: 1441 - 2019

تقترح هذه الأطروحة تصميم قاعدة حسابية مدمجة أولية عادية الأسس باستخدام الأجزاء المشتركة بين مضاعف (ماسي اومورا) وخوارزمية المعكوس (ايتو-تسوجي). وتشمل العمليات الحسابية التي سينفذها التصميم كل من: الجمع ، الضرب ، والمعكوس. عملية الجمع يمكن تنفيذها بسهولة باعتبارها بوابة منطقيةXOR. بينما يتطلب الضرب عادةً وقتًا حسابيًا أكثر من الجمع ولديه المزيد من التعقيد في دائرته الكهربائية. يمكن إستخراج المعكوس من خلال تطبيق خوارزمية التربيعية الضربية بشكل متكرر. تم إنشاء التصميم المقترح بواسطة لغة البرمجة VHDL باستخدام برنامج التصميم ISE من Xilinx 14.5 ومحاكاتها على قطعة Artix7 XC7A200T FPGA. تم تشغيل التصميم في البداية على مدخلات ذات 173 بت ، ثم تم ضبطه ليتم تشغيله على مدخلات 233 ، 350 ، 515 بت. أظهرت النتائج انخفاضًا في تعقيد المكونات للتصميم المقترح في جميع عدد المدخلات مقارنةً بالتصميم القياسي. مما يجعل التصميم جذابًا للغاية للأجهزة المحدودة الموارد ، مثل البطاقات الذكية وشبكات RFIDوأجهزة الاستشعار اللاسلكية ، عند تطبيق التشفير باستخدام المنحنيات الاهليجية.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1:

INTRODUCTION

1-1. Motivation

Efficient hardware implementation of elliptic curve cryptosystems (ECCs) in resource-constrained devices is important in many applications on small devices such as smart cards, radio-frequenciy identification (RFID), and wireless sensor networks. Computations in finite fields combined with low-hardware-complexity architectures are important in many areas, including coding theory, computer algebra systems, and public-key cryptosystems (e.g., ECCs). Although all finite fields of the same cardinality are isomorphic, their arithmetic efficiency depends greatly on the basis used for field element representation. The most commonly used are polynomial basis (PB) and normal basis (NB).

Polynomial basis: **PB** may also refer to a basis of the extension of the form {1, α,……., α^{m-1} } where α is the root of a primitive polynomial of degree m equal of the degree of the extension.

Normal basis: NB GF(2^m) is a basis of the form $\left(\beta.\beta^2.\beta^4.\beta^8.\ldots\ldots\beta^{2^{(m-1)}}\right)$, where $\beta \epsilon GF(2^m)$.

Arithmetic over NB finite fields GF(2^m) has recently been used in many significant applications, including error-correcting codes, cryptography, digital signal processing, switching theory and pseudorandom number generation. Addition, multiplication, exponentiation, and inversion are the most important computations in finite field arithmetic. Therefore, fast multiplication algorithms with low circuit complexity are much desired. Because such computations cannot be performed in real time on general-purpose computers, hardware-efficient architectures for multiplication in GF(2^m) are highly desirable.

1-2. Main Contribution

The contributions in this thesis are as follows:

- We propose a new compact optimal normal basis field arithmetic unit (FAU).
- We reduce the area cost in terms of NAND gates compared to a standard FAU.
- We reduce the area cost in terms of NAND gates compared to the research done in 3.1.
- We reduce the number of slice regiters and slice lookup tables compared to a standard FAU.
- We model the proposed design using VHDL and Implement it on Xilinx Artix7 XC7A200T FPGA over GF(2^{173}, 2^{233}, 2^{350}, 2^{515}),

1-3. Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 introduces the required background on the field of arithmetic operations and optimal normal basis. Chapter 3 presents the literature review. Chapter 4 describes the design methodology of the FAU proposed in this thesis. Chapter 5 shows the results of implementing the design and comparing it to the standard FAU. Chapter 6 concludes the thesis and discusses future work.

1-4. Chapter Summary

In this Chapter, we mentioned the motivation for the thesis and what contributaions were made. Also the whole structure of the thesis was explained. In the next Chapter we mention some background information regarding field arithmetic and optimal normal basis.

Chapter 2:

BACKGROUND INFORMATION

2-1. Finite Field Arithmetic

A finite field in abstract algebra [1], contains only a finite number of elements. Finite fields are important in cryptography, algebraic geometry, , number theory, coding theory, and Galois Theory. A set of elements G with any binary operation■ is called a *group* , it has the following properties:

1. Closure: $\forall\, a \blacksquare b \ \in G. a \ \blacksquare \ b \ \in G$.
2. Associativity: $\forall\, a.b.c \ \in G. (a \ \blacksquare \ b) \ \blacksquare \ c = a \ \blacksquare \ (b \ \blacksquare \ c)$.
3. Identity: The group contains an identity element $e \ \in G$ such that
4. $\forall\, a \in G \,. e \ \blacksquare \ a = a \ \blacksquare \ e$.
5. Inverse: Every element $a \in G$ has an inverse $a^{-1} \in G$ such that $a \blacksquare a^{-1} = a^{-1} \blacksquare a = e$.

Abelian groups are groups with a commutative group operation: i.e,

$$a \blacksquare b = b \blacksquare a \ \forall\, a.b \in G$$

Cyclic groups are groups that have a generator element. An element $\in G$, is a generator of the group if each element $a \in G$ can be generated by repeated application of the group operation on g. Thus, $\forall\, a \in G$,

$$a = \underbrace{g \blacksquare g \blacksquare g \ldots . \blacksquare g}_{itimes}$$

Groups with the "+" group operator are called *additive groups* and are specified as

$$ig = \underbrace{g + g + g + \cdots + g}_{itimes}$$

Similarly, groups with the "*" group operator are called *multiplicative groups* and specified as

$$g^i = \underbrace{g * g * g * g \ldots * g}_{itimes}$$

The number of elements in a group is represented by the symbol |*G*| and is called the order of the group *G*. A set of elements *F* is called a *field*, it has two binary operations, represented here as multiplication (*) and addition (+), and have the following properties:

1. With respect to the "+" operation, F is an abelian group.
2. An abelian group is formed by the elements in the set F* under the "*" operation.

   All the elements in F forms the set F*, except the additive identity.
3. The two binary operations apply the distribution law as follows:

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

The symbol GF(q) represents the finite fields or "Galois field", named after Evariste Galois, . For any positive integer *m* and prime *p*, there always exists a Galois field of order $q = p^m$. The characteristic of the finite field GF(p^m) is the prime *p*.

2-2. Arithmetic Logic Unit

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bit-wise operations on integer binary numbers. An ALU is a fundamental building block of many types of computing circuits, including the graphics processing units (GPUs) and the central processing unit (CPUs) of computers. A number of basic arithmetic and bit-wise logic functions are commonly supported by ALUs: Addition, Subtraction, AND,XOR, and Cyclic shifting. To the best of our knowledge, there are no ALUs that can perform inversion, this thesis propose a way to implement it.

2-3. GF(2^m) Arithmetic

Binary fields are finite fields of order 2^m, also called characteristic-two finite fields,[2]. They are particularly efficient for hardware implementation. The elements of GF(2^m) have coefficients of either 0 or 1, and are called binary polynomials. The degree of each polynomial is less or equal to $m - 1$ since there are 2^m polynomials in the field. Therefore, the elements can be represented as m-bit strings. Each bit in the bit string corresponds to the coefficient in the polynomial at the same position. For example, $GF(2^3)$ contains 8 elements $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$. The term $x+1$ is actually $0x^2+1x+1$, so it can be represented as a bit string 011. Similarly, $x^2+x = 1x^2+1x+0$, so it can be represented as 110. Arithmetic efficiency depends greatly on the basis of field element representation. Elements of the field are represented in terms of a basis. Most implementations use either a PB or a NB. NB is more suitable for hardware implementations than PB because NB operations mainly comprise rotation, shifting and exclusive-OR operations, which can be efficiently implemented in hardware.

2-4. Optimal Normal Basis

An NB GF (2^m) is a basis of the form $\left(\beta.\beta^2.\beta^4.\beta^8.\ldots\ldots\beta^{2^{(m-1)}}\right)$, where $\beta \epsilon$ GF(2^m).In an NB, an element $A \in$ GF(2^m) can be uniquely represented in the form

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$$

where $a_i \epsilon \{0.1\}$.

GF(2^m) operations [3] using NB are performed as follows:

i. **Addition & Subtraction** are performed by a simple bit-wise exclusive-OR (XOR) operation.

In modulo 2 arithmetics, 0+0 ≡ 0 mod 2, 1+0 ≡ 1 mod 2 , and 1+1 ≡ 0 mod 2, which coincide with bit-XOR, i.e., 0⊕0=0, 1⊕0=1, and 1⊕1=0, respectively. Therefore, addition is simply bit-by-bit XOR for binary polynomials.
Also, in modulo 2 arithmetics, -1 ≡ 1 mod 2, and so the result of the subtraction of elements is the same as addition, For example:

- Addition: $(x^2+x+1) + (x+1) = x^2+2x+2$. Because 2 ≡ 0 mod 2 the final result is x^2. It can also be computed as 111⊕011=100, where. 100 is the bit string representation of x^2.
- Subtraction: $(x^2+x+1) - (x+1) = x^2$

ii. Squaring is simply a rotate left operation. Thus, if $A = (a_{m-1}, a_{m-2}, \ldots a_1, a_0)$, then $A^2 = (a_{m-2}, a_{m-3}, \ldots a_0, a_{m-1})$

iii. Multiplication: ∀A. B ϵ GF (2^m). where

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} \text{ and } B = \sum_{i=0}^{m-1} b_i \beta^{2^i}$$

The product C=A*B is given by

$$C = A * B = \sum_{i=0}^{m-1} c_i \beta^{2^i}$$

Then, multiplication is defined in terms of a multiplication table $\lambda_{ij} \in \{0.1\}$

$$C_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \lambda_{ij} a_{i+k} b_{j+k}$$

the complexity of the multiplication process is defined by the number of non-zero elements in the λ matrix and accordingly the complexity of the hardware implementation. This value is defined as C_N and it is equal to 2_{m-1} for optimal normal basis **(ONB)** [4]. An **ONB** is a normal basis with the minimum number of non-zero elements in the λ_{ij} matrix. Such a basis typically leads to efficient hardware

implementations becaause operations mainly comprise rotation, shifting, and exclusive-OR operations.

iv. **Inversion:** The inverse of $a \in GF(2^m)$, denoted as a^{-1}, is defined as follows:

$$aa^{-1} = 1 \; mod \; 2^m$$

Most inversion algorithms used are derived from Fermat's Little Theorem:

$$a^{-1} = a^{2^m-2} = (\, a^{2^{m-1}-1})^2 \;\; \text{For all } a \neq 0 \text{ in GF}(2^m).$$

2-5. Types of Optimal Normal Bases

The derivation of values of the λ matrix element is dependent on the field size *m*. There are two types of ONBs , Type I and Type II [4]. An ONB Type I exists in a given field GF(2^m) if

- m+1 is a prime
- 2 is a primitive in GF(m+1)

An ONB Type II exists in GF(2^m) if

- 2m+1 is prime
- Either 2 is a primitive in GF(2m+1) or 2m+1= 3 (mod 4) and 2 generates the quadratic residues in GF(2m+1)

An ONB exists in GF(2^m) for 23% of all possible values of *m* [4]. The $\lambda^{(k)}$ matrix can be constructed by a k-fold cyclic shift to $\lambda^{(0)}$ as follows:

$\lambda_{ij}^{(k)} = \lambda_{i-k.j-k}^{(0)}$ for all $0 \leq i.j.k \leq m-1$

The $\lambda^{(0)}$ matrix is derived differently for the two types of ONBs. For the Type I ONB, $\lambda_{ij}^{(0)} = 1$ iff i and j satisfy one of the following two congruencies [5]:

- $2^i + 2 \equiv 1 \; mod \; (m+1)$
- $2^i + 2^j \equiv 0 \; mod \; (m+1)$

For Type II ONB $\lambda_{ij}^{(k)} = 1$ if *i* and *j* satisfy one of the following four congruencies [5]:

- $2^i + 2^j \equiv 2^k \, mod(2m+1)$
- $2^i + 2^j \equiv -2^k \, mod \, (2m+1)$
- $2^i - 2^j \equiv 2^k \, mod \, (2m+1)$
- $2^i - 2^j \equiv -2^k \, mod \, (2m+1)$

Therefor, $\lambda_{ij}^{(0)} = 0$ if *i* and *j* satisfy one of the following four congruencies:

$$2^i \pm 2^j \equiv \pm 1 \, mod \, (2m+1)$$

2-6. Elliptic Curve Cryptography

Elliptic Curve Cryptography [6] uses a group of points for cryptographic schemes with coefficient sizes of 160-256 bits, significantly reducing the computational effort. The inability to compute the multiplicand given the original and product points and the ability to compute a point multiplication determines the security of elliptic curve cryptography. The primary benefit promised by elliptic curve cryptography is a smaller key size [7], thus reducing storage and transmission requirements, which makes it popular for use in embedded systems and resource-constrained devices. The size of the elliptic curve determines the difficulty of the problem. Operations used in ECCs:

- Modular addition and subtraction
- Modular multiplication
- Modular inversion

2-7. Chapter Summary

This background chapter introduced some important concepts in finite field GF(2^m) arithmetic operations such as addition, multiplication, squaring and inversion. It also explained the concept of an ONB which has a minimum possible number of non-zero elements in the λ_{ij} matrix that defines its type. There are two types of ONBs, Type I and Type II. Each type exists in a given field GF(2^m) if one of several conditions that justify $\lambda_{ij}^{(0)} = 1$ is applied. This chapter also introduced the ECC concept and how its cryptography uses a group of points for cryptographic schemes with coefficient sizes of over 160 bits. This last point is very important as it shaped the number of bits selected to run the design, as described in Chapter 5.

Chapter 3:

LITERATURE REVIEW

In this chapter, we survey the research on the multiplication and inversion operations of normal bases GF(2^m) and highlight the Massey-Omura multiplier and the Itoh-Tsuji inversion algorithm, which are used in this thesis.

3-1. Pipelined Multiplicative Inverse Architecture for Advanced Encryption System Cryptography

Abd-El-Barr and Khattab [8] introduced architecture for performing a recursive pipeline algorithm to optimize the performance of multiplicative inverse operations in the Galois Field GF(2^8), which is used in performing S-Box byte-substitution in advanced encryption system (AES) cryptosystems**.** The S-Box performs a non-linear transformation on the data by replacing each individual byte with a different byte. The main purpose of the byte substitution is to bring confusion to the data to be encrypted. By determining the multiplicative inverse of a given state in finite field GF(2^8), the replacement bytes can be obtained**.** Abd-El-Barr and Khattab's improvement was to efficiently utilize the resources available. Their main observation was that because some gates could be triggered concurrently, improved circuitry should follow a pipelined approach. In a pipelined architecture, it is important to emphasize the order of the operations, and hence pipelined stages will be explicitly shown. Figure .1 shows the pipeline architecture. Here, the subscript 4 indicates a data size of 4 bits because; the operations are implemented in the field GF$(2^4)^2$.

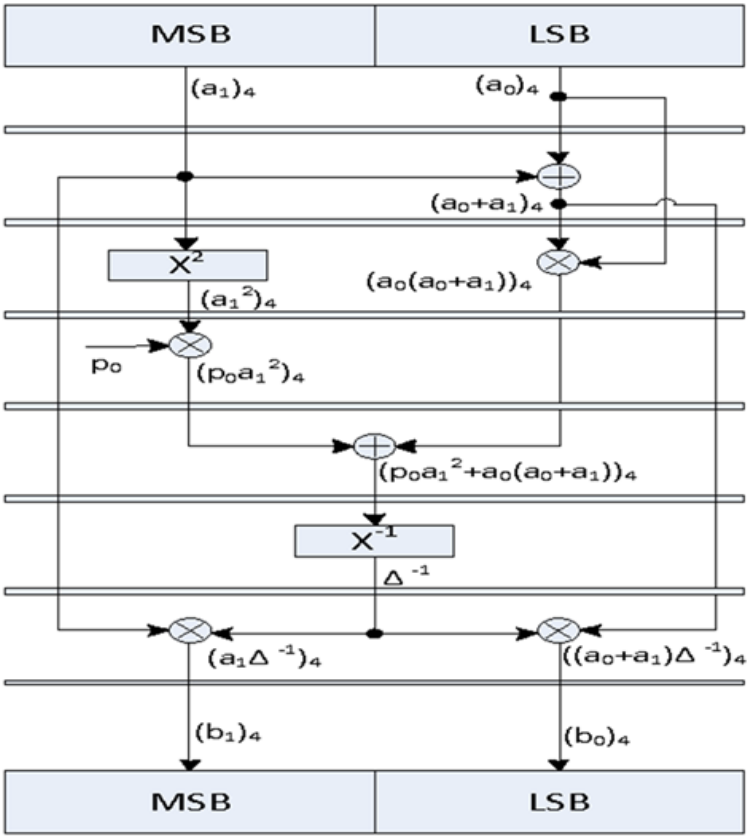

Figure 1 . Multiplicative inverse gate implementation over GF$(2^4)^2$ using a pipeline.

After making some calculations, the authors concluded that the total cost for the architecture in terms of gate cost was 48(•) and 152(*x*). where (•) represented AND gates and (*x*) represented XOR gates. In order to convert this area cost into pure two-way NAND gates, an AND gate took two NAND gates and an XOR gate took four NAND gates. This way, the total area in NAND gate units was 48(2)+152(4) = 704. The authers concluded that their proposed pipeline approach decreased the time delay at the expense of a bit more area.

3-2. Low-complexity Hardware Architecture of Gaussian Normal Basis Multiplication over GF(2^m) for ECCs

Rashidi, Sayedi, and Farashahi [9] presented an efficient high-speed architecture of a Gaussian normal basis (GNB) multiplier over a binary finite field GF(2^m) The structure was constructed by using some regular modules for computation of exponentiation by powers of 2 and low-cost blocks for multiplication by normal elements of the binary field. For the powers of 2 exponents, the modules were implemented by some simple cyclic shifts in the NB representation.

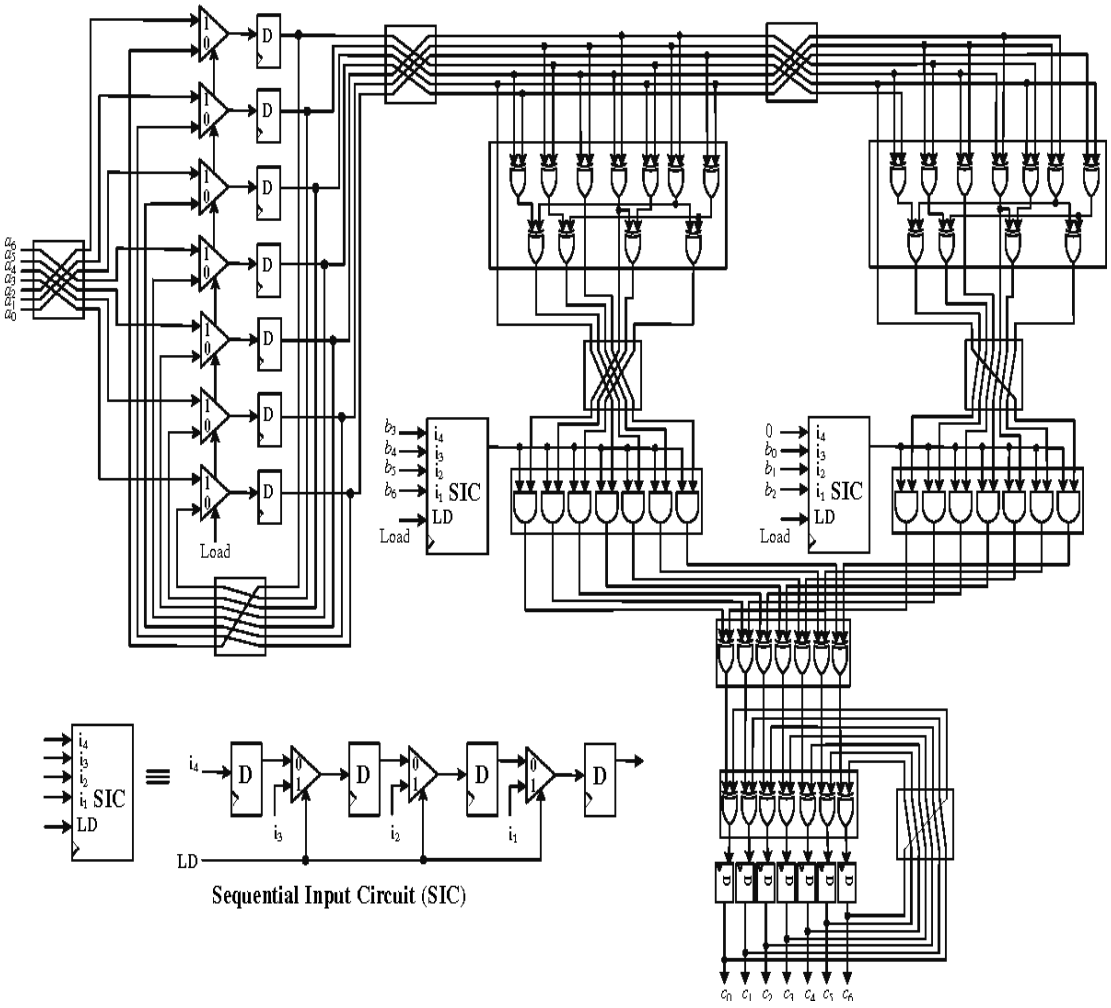

Figure 2. Proposed structure of the digit-serial GNB multiplier over GF(2^7). with w = 4 and d = 2

For the case of word = 4 and digit = 2, the word representations of *B* are

$$B_1 = b_6\beta^{2^6} + b_5\beta^{2^5} + b_4\beta^{2^4} + b_3\beta^{2^3}$$

$$B_2 = b_2\beta^{2^2} + b_1\beta^{2^1} + b_0\beta$$

And the multiplication result is $C = (((C_1^2 + C_2)^2 + C_3)^2 + C_4)$, where $C_1 - C_4$ are

$$C_1 = ((A^{2^{-3}})^{2^3}b_6 + (((A^{2^{-3}})^{2^{-3}})^{2^4}\beta)^{2^{-1}}b_2$$

$$C_2 = (((A^{2^{-3}})^2)^{2^{3-}}\beta)^{2^3}b_5 + ((((A^{2^{-3}})^{2^2})^{2^{-3}})^{2^4}\beta)^{2^{-1}}b_1$$

$$C_3 = (((A^{2^{-3}})^{2^2})^{2^{-3}}\beta)^{2^3}b_4 + ((((A^{2^{-3}})^{2^2})^{2^{-3}})^{2^4}\beta)^{2^{-1}}b_0$$

$$C_4 = (((A^{2^{-3}})^{2^3})^{2^{-3}}\beta)^{2^3}b_3 + ((((A^{2^{-3}})^{2^3})^{2^{-3}})^{2^4}\beta)^{2^{-1}}b_{-1}$$

The proposed digit-serial GNB method had *(m− 1)(T − 1) d + dm* number of XOR gates and *dm* number of AND gates. After implementing the design in FPGAVirtex-4 XC4VLX100 for GF (2^{233}) the authors obtained 1458 slice registers and 2811 lookup tables (LUTs). After comparing their work with other digit-serial structures, the authers concluded that the proposed work had suitable timing characteristics and hardware utilization results.

3-3. Massey - Omura Multiplier

In finite field arithmetic, multiplication is more complicated than squaring and addition operations. For efficient finite field computations, an efficient multiplier is highly needed. Finite field multipliers using normal bases can be classified into two main categories: (1) conversion-based multipliers, and (2) λ-matrix-based multipliers. A Massey- Omura multiplier [10] is a λ-matrix-based multiplier that computes *c = a x b* based on a matrix-vector product where the constant matrix λ is composed of only GF(2) elements. The notation $c[i]$ is the *i*-th bit of *c* (starting with least significant bits (LSBs)) Massey and Omura proposed an efficient NB bit-serial multiplier over GF(2^m) that required only two *m*-bit cyclic shift registers and combinational logic (which consists of a set of AND XOR logic gates). The space complexity of the Massey-Omura

multiplier is (2*m* - 1) AND gates + (2*m* - 2) XOR gates, depending on the number of non-zero elements in the λ- matrix.

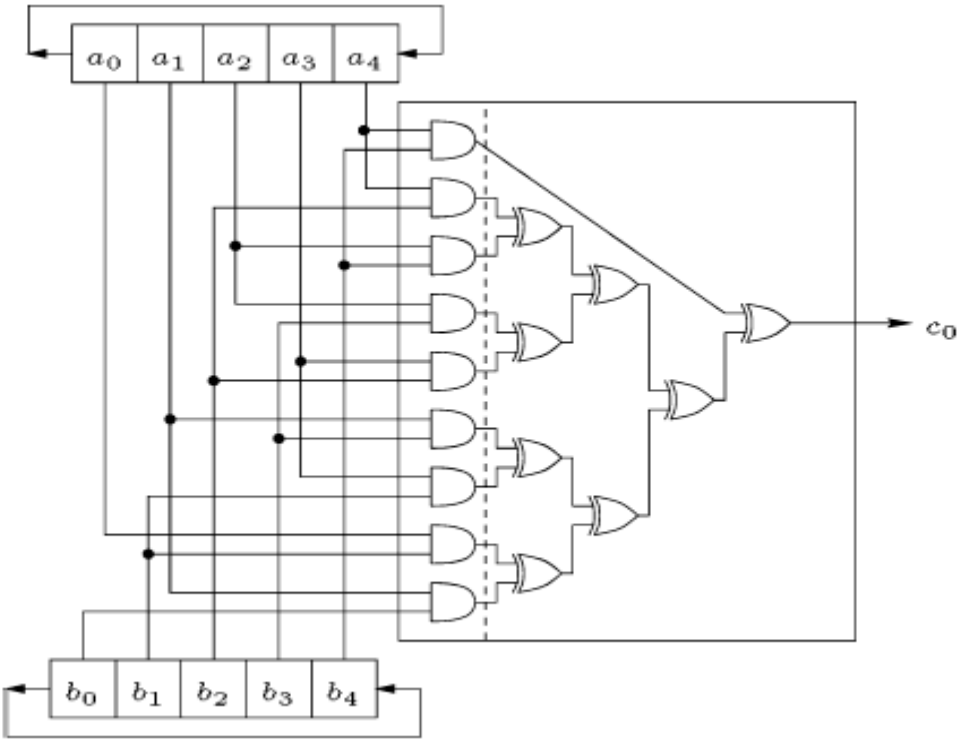

Figure 3. 5-bit Massy-Omura Multiplier Circuitry

One advantage of the Massey-Omura multiplier is that it can be used with both types of optimal normal bases (Type I and Type II). Another advantage is that it is a bit-serial multiplier, and hence the same circuitry used to generate *c0* can be used to generate *c[i] (i = 1; 2; :::;m - 1).*

Algorithm 1: Massey-Omura multiplication

Operands: *a*,*b* in GF(2^m) represented in NB

Result: $c = a \times b$

1. $c \leftarrow 0$
2. **for** *i* **from** 0 **to** $m - 1$ **do**
3. $c[0] \leftarrow a \times \lambda \times b$
4. $a \leftarrow \mathrm{ROL}(a, 1)$; $b \leftarrow \mathrm{ROL}(b, 1)$; $c \leftarrow \mathrm{ROL}(c, 1)$
5. **return** *c*

3-4. Itoh-Tsuji Inversion Algorithm

Inversion using NB consists of multiplications and cyclic shifts. The number of multiplications is the major parameter for efficient inversion because cyclic shifts require almost trivial time. Inversion algorithms can be classified into three main categories :(1) standard, (2) exponent grouping (3) exponent decomposing inversion algorithms. Beacuse the number of multiplications is the main parameter in determining the computation time of the inversion operation, several algorithms have attempted to improve the inversion speed by decomposing the exponent to reduce the required number of multiplications and replacing it with squaring operations, which are much simpler compared to multiplications. Itoh and Tsuji [11], proposed a GF(2^m) inversion algorithm derived from Fermat's Little Theorem using normal bases. The basic idea was to decompose the exponent *m* -1 as follows:

$$a^{-1} = a^{2^{m-2}} = (a^{2^{m-1}-1})^2.$$

The exponent 2^{m-1} is further decomposed as follows:

1. If *m* is odd, then

   $$(2^{m-1} - 1) = \left(2^{\frac{m-1}{2}} - 1\right)\left(2^{\frac{m-1}{2}} + 1\right).$$

   and

   $$a^{2^{m-1}} = (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}+1}$$

2. If *m* is even, then

   $$(2^{m-1} - 1) = 2(2^{m-2} - 1) + 1 = 2\left(2^{\frac{m-2}{2}} - 1\right)\left(2^{\frac{m-2}{2}} + 1\right) + 1.$$

   and

   $$a^{2^{m-1}} = 2^{2\left(2^{\frac{m-2}{2}}-1\right)\left(2^{\frac{m-2}{2}}+1\right)+1}$$

The proposed algorithm by Itoh and Tsuji is shown below [3]. It requires three *m*- bit cyclic shift registers, one barrel shifter, two down counters and one multiplier. It requires $log_2(m-1) + v(m-1) - 1$ multiplications, where *v*(*x*) is the number of 1s in the binary representation of *x*.

Algorithm 2: Itoh-Tsujii inversion

Inputs: a

Output: $l = a^{-1}$

1. Set $s \leftarrow [log_2(m-1)] - 1$
2. Set $p \leftarrow a$
3. For $i = s\ down\ to\ 0$ do
   - 3.1 Set r $\leftarrow$ shift $m-1$ to right by s bit(s)
   - 3.2 Set $q \leftarrow p$
   - 3.3 Rotate q to left by $[r/2]$ bit (s)
   - 3.4 Set $t \leftarrow p \times q$
   - 3.5 If last bit of $r = 1$
     - 3.5.1 rotate t to left by 1 bit.
     - 3.5.2 $p \leftarrow t \times a$
   - 3.6 Else
     - 3.6.1 $p \leftarrow t$
   - 3.7 $s \leftarrow s - 1$
4. Rotate p to left by 1 bit
5. Set $l \leftarrow p$
6. Return l

3-5. Fast Inversion in GF(2^m) with NB Using Hybrid-Double Multipliers

Azarderakhsh, Jarvinen, and Dimitrov [12] presented techniques to exploit recently proposed hybrid-double multipliers for fast inversions in binary fields GF(2^m) with normal bases. A hybrid-double multiplier computes a double multiplication, the product of three elements in GF(2^m), with a latency comparable to the latency of single multiplication of two elements.

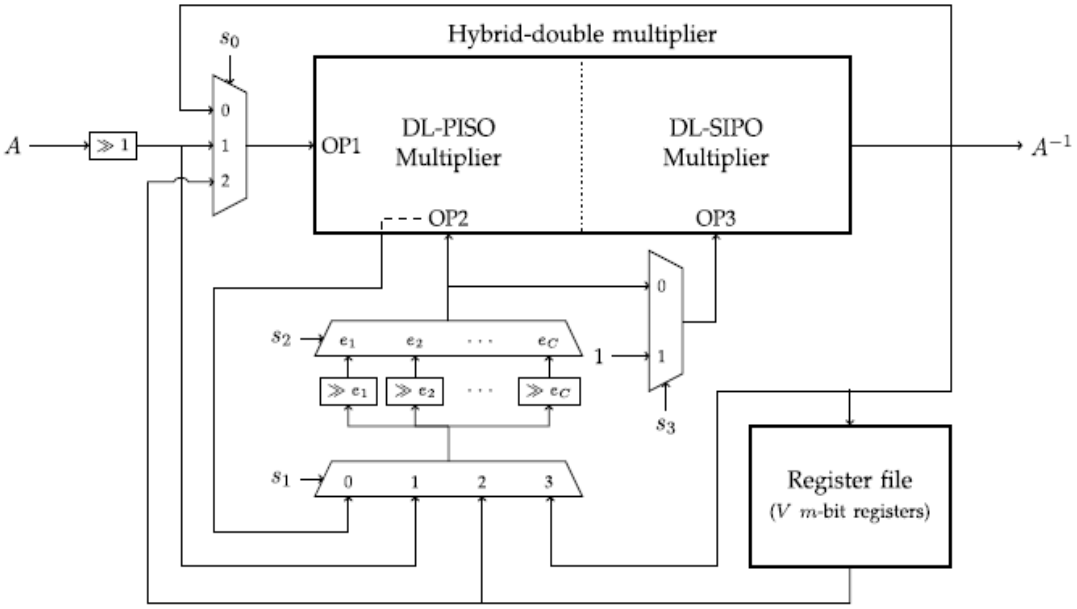

Figure 4. The proposed inverter architecture using a hybrid-double multiplier.

The authers concluded that faster inversion times were achieved at the expense of larger area requirements. The proposed scheme is applicable primarily for high-performance cryptographic applications where the aim is to maximize the speed of inversions. However for small devices with limited resources, the IT inverter is recommended [11].

3-6. Small FPGA-based Multiplication-Inversion Unit for NB Representation in GF(2^m)

Métairie, Tisserand, and Casseau [13] proposed a small FPGA- based multiplication-inversion unit that uses permuted normal basis (PNB) representation, Massey-Omura multiplication, and Itoh-Tsujii inversion algorithms. They produced the output bits of the multiplication serially, like the original Massy-Omura, but with two bits at each clock cycle.

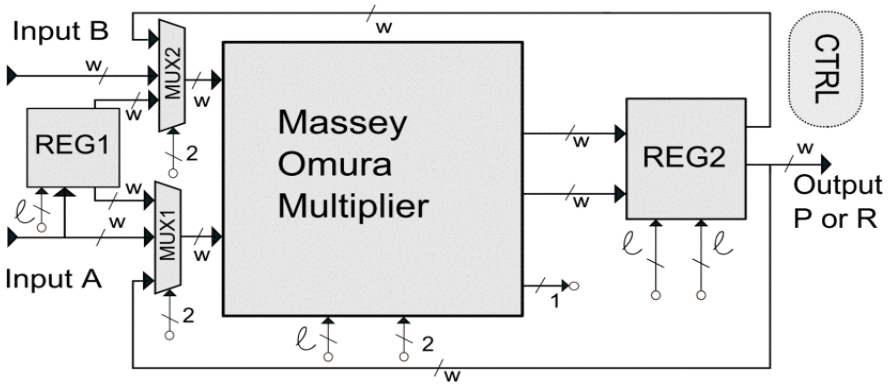

Figure 5. Architecture of the multiplication-inversion unit (MIU).

Upon further studying their work and the results they provided, we found that their multiplication-inversion unit leads to about 20% theoretical speed-up over previous works at the cost of area efficiency.

3-7. Chapter Summary

In this chapter we surveyed different types multiplication and inversion algorithms and found that the Massey-Omura multiplier and the Itoh-Tsuji inverter were the best for resource-constrained devices. In the next chapter, we introduce and discuss our design.

Chapter 4:

MYTHODOLOGY

The main idea for this thesis came from a key observation from the Itoh-Tsuji inverter [11], which was that the inverter's three cyclic shift registers could be used for both the multiplier and the inverter by proper scheduling. These cyclic shift registers could also be used to perform concurrent squaring. Accordingly, a compact GF(2^m) NB field arithmetic unit (FAU) is presented in this section.

4-1. The Proposed Design

The main idea of the proposed FAU is to utilize the common parts of the Massey-Omura multiplier [10] and the Itoh-Tsujii inverter [11]. The remaining field arithmetic operations are also included, where these are:

- Addition
- Squaring
- Square root
- AND
- Multiplication
- Inversion

The standard approach of the design performs these operations individually, each with its input and output registers. The proposed design will combine these operations into a compact FAU to benefit from shared registers. The ultimate goal was to design an FAU that can perform the basic operation required of a processor that implements ECC systems for resource-constrained devices.

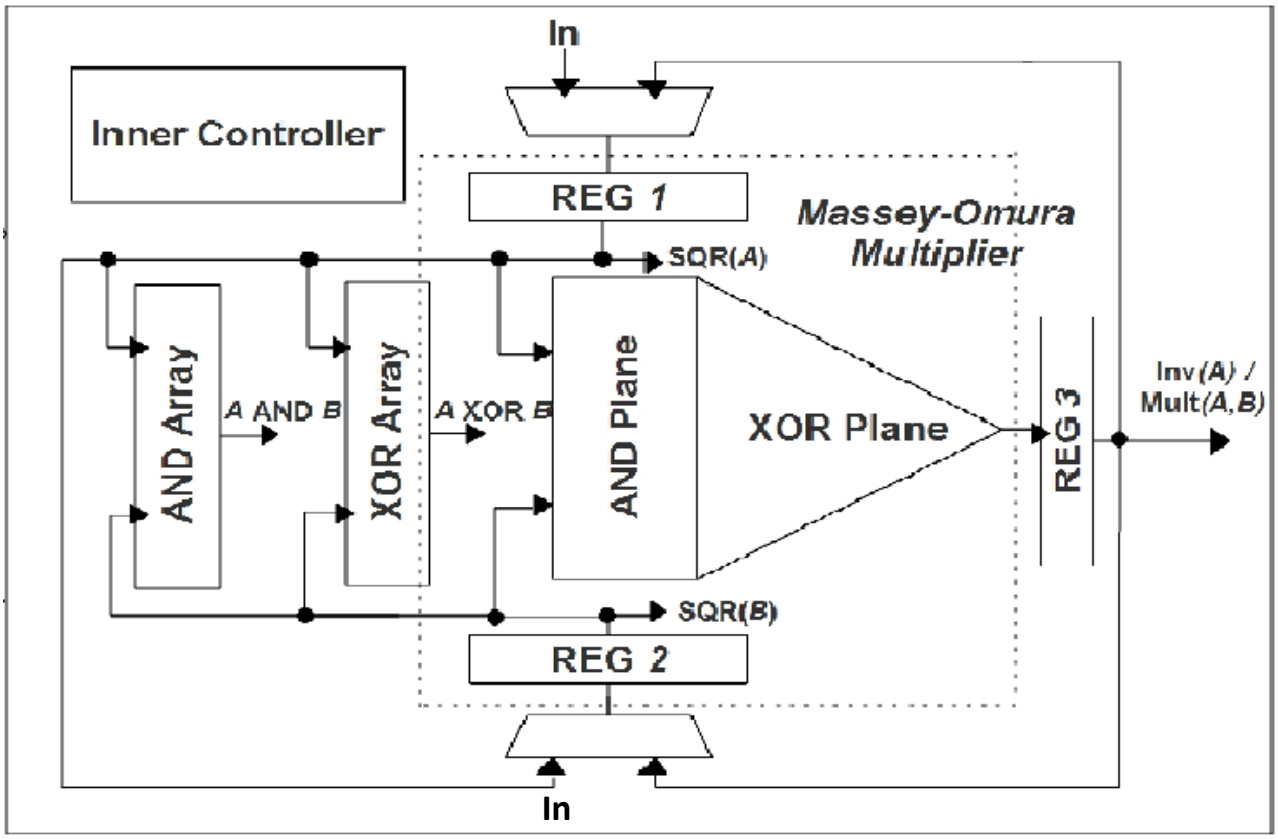

Figure 6. The Proposed FAU design

The proposed FAU consists of three cyclic shift registers, three multiplexers (MUXs), two down counters, one barrel shifter, *m*-AND gates for the AND operation, *m*-XOR gates for the XOR operation, an AND plane, which is the *2m* - 2 AND gates for the Massey-Omura multiplier and an XOR tree, which is the *2m* - 1 XOR gates for the Massey-Omura multiplier.

4-2. Operations of the Proposed FAU

Here we have the inputs: *A, B.*

Outputs: *(A AND B)/(A XOR B)/A.B/*A^{-1}/A^2 / $B^2/\sqrt{A}\sqrt{B}$

1. *AND/XOR operation:* The inner controller loads both A&B into REG1 & REG2. Accordingly, we get the output A AND/XOR B.

2. *Squaring/Squareroot:* The inner controller loads both A&B into REG1 & REG2. Accordingly, with cyclic shift to left/right we get the outputs A^2B^2/ $\sqrt{A}\sqrt{B}$ concurrently.

3. *Multiplication:* The inner controller loads both A&B into REG1 & REG2. The same procedure for the Massey-Omura multiplier is followed to get the output AB and the output is saved in REG3.

4. *Inversion:* The Itoh-Tsujii algorithm is implemented in the proposed FAU.

Algorithm 3: Itoh-Tsuji inversion in the proposed FAU

Inputs: a

Output: $l = a^{-1}$

1. Set $s \leftarrow [log_2(m-1)] - 1$
2. Set $REG1 \leftarrow a$
3. For $i = s\ down\ to$ 0 do
    - 3.1 Set r ← shift $m-1$ to right by s bit(s)
    - 3.2 Set $REG2 \leftarrow REG1$
    - 3.3 Rotate *REG2* to left by $[r/2]$ bit (s)
    - 3.4 Set $REG3 \leftarrow REG1 \times REG2$
    - 3.5 If last bit of $r = 1$
        - 3.5.1 Rotate *REG3* to left by 1 bit.
        - 3.5.2 $REG2 \leftarrow REG3$
        - 3.5.3 $REG3 \leftarrow REG2 \times REG1$
        - 3.5.4 $REG1 \leftarrow REG3$
    - 3.6 Else
        - 3.6.1 $REG1 \leftarrow REG3$
    - 3.7 $s \leftarrow s - 1$
4. Rotate $REG1$ to left by 1 bit
5. Set $l \leftarrow REG1$
6. Return l

The inner controller loads input a into REG 1 and logarithm $log_2(m-1)$ is calculated and stored in variable *s* to determine the number of iterations the algorithm will take to produce the inverse (i.e., the number of multiplications and squarings). In the first iteration, the binary representation of the number of bits *m-1* is shifted to the right *s* times, converted into decimal form, and stored in variable *r*. The value in REG 1 is then loaded into REG 2 and rotated to the left (square operation) r/2 times. Next, REG 1 and REG 2 are multiplied (Massey-Omura multiplication), and the result is stored in REG 3. Then, the least significant bit in the binary representation of *r* is then checked; if it is 0 the result in REG 3 is loaded into REG 1 and the next iteration begins. However if it is 1, the result stored in REG 3 is squared (i.e., rotated to the left by 1) and loaded into REG 2, REG 1 and REG 2 are multiplied, the result is stored in REG 3 and loaded into REG 1, and the next iteration begins. After the conclusion of all iterations, the final value in REG 1 is squared (i.e., rotated to the left by 1) and returned as the inverse *l* of the input a.

4-3. Chapter Summary:

In this chapter, we described the proposed design concept, which is to utilize the common parts of the Massey-Omura multiplier and the Itoh-Tsujii inverter. We also established in detail that it performs addition, squaring, square root, ANDing, multiplication, and inversion operations. In the next chapter, we present the results obtained after implementing the design in VHDL, and then we compare the results with those of the standard approach for multiple number of bits.

Chapter 5:

IMPLEMENTATION RESULTS

5-1. Evaluating the Proposed Design

To evaluate the proposed FAU, we compared it with a standard FAU that performs the operations mentioned in 4.1 separately. The standard FAU consists of the following:

- m AND gates for the AND operation and $2m$ - 1 AND gates for the Massey-Omura multiplier. (total = $m + 2m$-1 = $3m$-1 AND gates)
- m XOR gates for the XOR operation and $2m$ - 2 XOR gates for the Massey-Omura multiplier. (total = $m + 2m - 2 = 3m$ – 2 XOR gates)
- $3m$-bit cyclic shift registers for the Massey-Omura multiplier and $3m$- bit cyclic shift registers for the Itoh-Tsuji inverter. (total = $3m + 3m = 6m$-bit registers)
- $2m$-bit cyclic shift registers for two concurrent squaring operations. (total = $2m$-bit registers)
- Two m 2-to-1 MUXs for the Itoh-Tsuji inverter. (total = $2m$ MUXs)

The proposed FAU consists of the following:

- m AND gates for the AND operation and $2m$ - 1 AND gates for the Massey-Omura multiplier. . (total = $m + 2m$-1 = $3m$-1)
- m XOR gates for the XOR operation and $2m$ - 2 XOR gates for the Massey-Omura multiplier. (total = $m + 2m - 2 = 3m - 2$)
- $3m$-bit cyclic shift registers for both the Massey-Omura multiplier and the Itoh-Tsuji inverter. (total = $3m$-bit registers)
- $2m$-bit cyclic shift registers for two concurrent squaring operations. (total = $2m$-bit registers)
- Two m 2-to-1 MUXs for the Itoh-Tsuji inverter. (total = $2m$ MUXs)

We convert all combination logic/gate in both designs to their NAND equivalents.

Table. 1 Area cost of proposed design in terms of NAND

Clogic/Gates	NANDs	Standard	NANDs	Proposed	NANDs
AND	*2*	*3m - 1*	*6m – 2*	*3m - 1*	*6m - 2*
XOR	*4*	*3m - 2*	*12m – 8*	*3m - 2*	*12m – 8*
m-bit Reg	*4m*	*8*	*32m*	*3*	*12m*
m 2-to-1 MUX	*3m*	*2*	*6m*	*2*	*6m*
		Total =	*56m – 10*	*Total =*	*36m - 10*

Table 1 summarizes the hardware requirements of the proposed FAU and the standard FAU. It also shows the equivalent NAND gate cost of both designs. The first two columns of Table 1 show the combination logic/gates and the required number of NANDs to implement them, respectively. The third and fourth columns show the required number of these combinational logic/gates for the standard FAU and their equivalent NANDs, respectively. Similarly, the fifth and sixth columns show the required number of these combinational logic/gates for the proposed FAU and their equivalent NANDs, respectively. Furthermore, the fourth and the sixth columns show the results of multiplying the second column by the third and fifth columns, respectively. Finally, the last row shows the total number of NANDs for the standard and the proposed FAUs.

The results of Table 1 show that the proposed FAU saves 35% [1- (36*m* -10)/ (56*m* -10) *100 = 35] of the total number of NANDs as compared to the standard FAU. For resource-constrained devices such as smart cards [14], RFID [15] and wireless sensor networks [16], our compact FAU is a very attractive when implementing ECCs. In Table 2 we compare our results with those of the architecture mentioned in Section 3.1 for $m = 8$.

Table. 2 Comparing area cost of the pipelined multiplicative Inverse with our proposed design in terms of NAND

Clogic/Gates	NANDs	Pipelined	NANDs	Proposed	NANDs
AND	*2*	*48*	*96*	*23*	*46*
XOR	*4*	*152*	*608*	*22*	*88*
m-bit Reg	*4m*	-	-	*3*	*96*
m 2-to-1 MUX	*3m*	-	-	*2*	*48*
		Total =	*704*	*Total =*	*278*

In Table 2. We compare the area cost in NAND gate units that was presented in the Pipelined Multiplicative Inverse Architecture in Section 3.1 with our Proposed FAU. The Pipelined Architecture had 48(2) + 152(4) = 704 NAND for m = 8. In our proposed design we gave m the value 8 and calculated the area cost 36(8) -10 = 278, or 60% less area.

5-2. Implementing the Design in VHDL

The two main points in comparing the proposed FAU with the standard FAU is to compare the number of slice registers and the number of slice LUTs.We implemented the design for 173 bits using VHDL Coding and Simulation on Xilinx Artix7 XC7A200T FPGA . Table 3 shows the data after synthesizing the design.

Table 3. Synthesized results of 173-bit input

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1076	269200	0%
Number of Slice LUTs	7404	134600	5%
Number of fully used LUT-FF pairs	1070	7410	14%
Number of bonded IOBs	526	285	184%
Number of BUFG/BUFGCTRL/BUFHCEs	1	152	0%

The third row in table 3 shows that the proposed design have 1076 slice registers with almost 0% utilization of available logic.The forth row shows that the proposed design have 7404 LUTs with approximately 5% utilization of available logic.

One point to clear in the sixth row, it shows the number of bounded input/output with utilization exceeding the available logic. This will not give an error when running the simulation and the results are correct. But when implementing the design on a physical chip some procedures must be taken to insure the utilization doesn't exceed the available logic. One suggestion is to add a buffer to pass 50 bits of the input at a time, another suggestion is to choose a chip with a large number of I/O pins. In the next section the number of slice registers and LUTs for the proposed design is compared to the standard design.

5-3. Comparing the Slice Registers and LUTs with the Standard Design.

Table 4. Comparison of slice registers in LUTs of 173-bit design

Standard design	Number of slice registers	Number of slice LUTs	Clock cycles	Minimum time (ns)
ANDing	173	173	1	1.060
XORing	173	173	1	1.060
Sqr /Sqrt	0	0	1	0.400
Multi	519	699	173	1354.936
Inv	1366	8884	2170	18139.03
Total	2231	9929	-	-
Proposed design	1076 (48%)	7404 (25 %)	Multiplication 179 Inversion 2516	1290.59 18140.36

Table 4 summarize the comparison between the proposed FAU and the standard FAU for 173-bit. The first column shows the operations performed in the design. The second column shows the number of slice registers for each operation and design. The third column shows the number of slice LUTs for each operation and design. The fourth column shows the number of clock cycles for each design. The fifth column shows the minimum time required to perform the operations. The total number of slice registers for the standard design is 2231 and LUTs is 9929, whereas the slice registers for the proposed design are 1076 and LUTs are 7404. By comparing the two results, it is apparent that the proposed design reduces the number of slice registers by 48% and the number of LUTs by 25%. This makes the proposed FAU very attractive and suitable for resource-constrained devices.

5-4. Running the Design on Different Number of Bits

To further evaluate and analyze the proposed design, we ran the code for 233, 350, and 515 bits and compared the number of slice registers and slice lookup tables (LUTs) between the standard FAU and the proposed FAU, as we did with 173 bits . The same code was used on all number of bits, the only difference is being that the AND XOR circuitry for the Massey-Omura multiplier changed for each number of bits. The circuitry was obtained from the λ matrix described in Section 2-4. The results are given in Table 5.

Table 5. Comparing the proposed and standard design for different numbers of bits

No. of Bits	Standard	Proposed	Standard	Proposed
	Slice registers		Slice LUTs	
173 bit	2231	1076	9929	7404
233 bit	3087	1436	14682	10945
350 bit	4679	2491	29901	17088
515 bit	7353	3650	106364	85442

The first column of Table 5 shows the different number of bits used to implement the design. The second column shows the number of slice registers of the standard design for the different number of bits, while the third column shows the number of slice registers of the proposed design. The fourth column shows the number of slice LUTs of the standard design for the different number of bits, while the fifth column shows the number of slice LUTs of the proposed design. For the 233-bit input, the proposed design reduced the number of slice registers by 46% whereas the number of LUTs was reduced by 25%. For the 350-bit input, the proposed design reduced the number of slice registers by 53% whereas the number of LUTs was reduced by 42%. For the 515-bit input the proposed design reduced the number of slice registers by 49% whereas the number of LUTs was reduced by 20%.

5-5. Analyzing the results

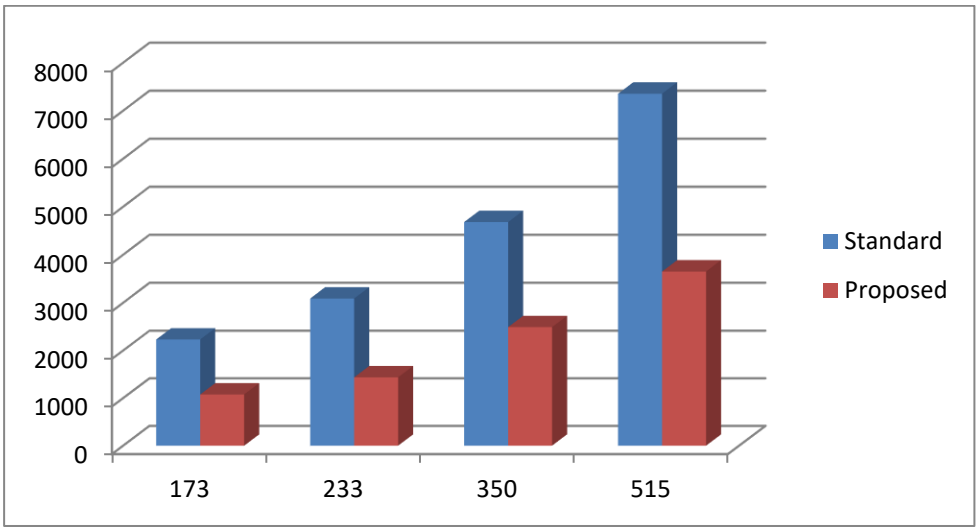

Figure 7. Comparing slice registers of the standard and proposed designs

Figure 7 shows that as the number of bits increased in the proposed design of reduction of slice registers is nearly half (50%) of the standard design . This is because the proposed design removes the additional slice registers needed to perform individual addition, ANDing, multiplication and inversion in the standard design.

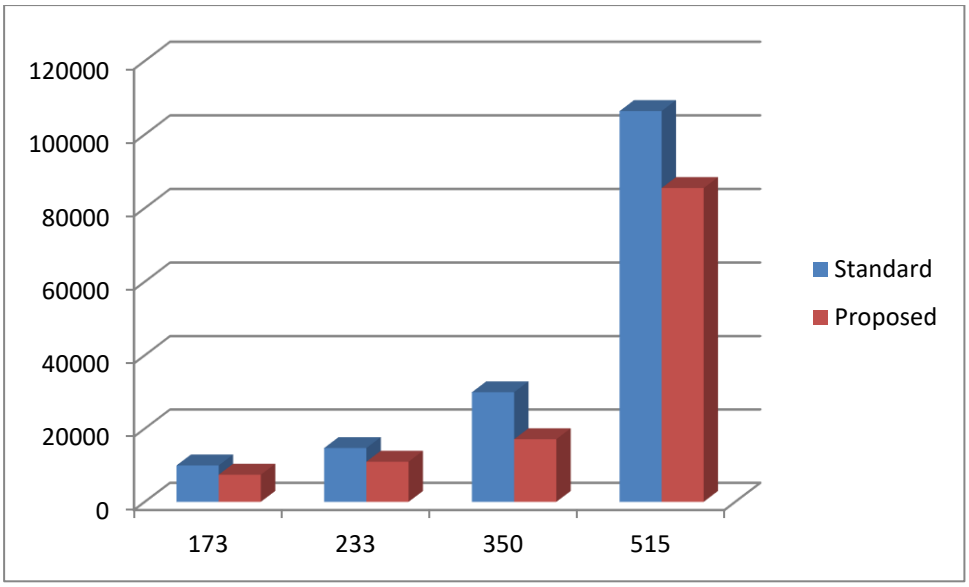

Figure 8. Comparing LUTs of the standard and proposed designs

Figure 8 compares the lookup tables (LUTs) used in the standard design and in our proposed design. To further analyze and understand these results, we need to know that in our proposed FAU the inversion operation determines the area cost of our design as it performs many multiplications. We must also know that the number of

multiplications is the primary factor that determines the computation of the inversion operation, and that the Itoh-Tsujii inversion algorithm decomposes the exponent to reduce multiplications and replace them with squaring operations, depending on the number of 1's in the binary representation of the number of bits (*m-1*).

These factors combine to determine the number of LUTs for each number of bits. For example, when we calculate the number of squaring and multiplications in the inversion operation of the 350-bit design, we have $log_2(350-1) = 8$ and $349 = (101011101)$. Therefor, when we run the Itoh-Tsujii algorithm we have:

$s = 7$,

for $i = 7\ down\ to\ 0$ (the inversion algorithm will take 8 iterations)

First iteration:

$r \leftarrow$ (shift 349 to right by 7 bits) = (101110110) = 374

Rotate *REG2* to left by $[r/2]$ bits = 374/2 = 187 (squaring)

Second iteration:

$r \leftarrow$ (shift 349 to right by 6 bits) = (011101101) = 237

Rotate *REG2* to left by $[r/2]$ bits = 237/2 = 118 (squaring)

Third iteration:

$r \leftarrow$ (shift 349 to right by 5 bits) = (111011010) = 474

Rotate *REG2* to left by $[r/2]$ bits = 474/2 = 237 (squaring)

Fourth iteration:

$r \leftarrow$ (shift 349 to right by 4 bits) = (110110101) = 437

Rotate *REG2* to left by $[r/2]$ bits = 437/2 = 218 (squaring)

Fifth iteration:

$r \leftarrow$ (shift 349 to right by 3 bits) = (101101011) = 374

Rotate *REG2* to left by $[r/2]$ bits = 374/2 = 187 (squaring)

Sixth iteration:

$r \leftarrow$ (shift 349 to right by 2 bits) = (011010111) = 215

Rotate *REG2* to left by $[r/2]$ bits = 215/2 = 107 (squaring)

Seventh iteration:

$r \leftarrow$ (shift 349 to right by 1 bit) = (110101110) = 430

Rotate *REG2* to left by $[r/2]$ bits = 430/2 = 215 (squaring)

Final iteration:

$r \leftarrow$ (shift 349 to right by 0 bits) = (101011101) = 349

Rotate *REG2* to left by $[r/2]$ bits = 349/2 = 174 (squaring)

Sqauring = 174 +215 +107 +187 +218 +237 +118 +187 +5 (the number of 1's in the first 7 bits of 349) + 1 (after all iterations are finished) = 1449. Multiplications = 8 (number of iterations) +5 (number of 1's in the first 7 bits of 349) = 13

For the number of squaring and multiplications in the inversion operation of the 515-bit design, we have $log_2(515-1) = 9$ and 514 = (1000000010).

s = 8,

for $i = 8\ down\ to\ 0$ (the inversion algorithm will take 9 iterations)

First iteration:

$r \leftarrow$ (shift 514 to right by 8 bits) = (0000001010) = 10

Rotate *REG2* to left by $[r/2]$ bits = 10/2 = 5 (squaring)

Second iteration

$r \leftarrow$ (shift 514 to right by 7 bits) = (0000010100) = 20

Rotate *REG2* to left by $[r/2]$ bits = 20/2 = 10 (squaring)

Third iteration:

$r \leftarrow$ (shift 514 to right by 6 bits) = (0000101000) = 40

Rotate *REG2* to left by $[r/2]$ bits = 40/2 = 20 (squaring)

Forth iteration:

$r \leftarrow$ (shift 514 to right by 5 bits) = (0001010000) = 80

Rotate *REG2* to left by $[r/2]$ bits = 80/2 = 40 (squaring)

Fifth iteration:

$r \leftarrow$ (shift 514 to right by 4 bits) = (0010100000) = 160

Rotate *REG2* to left by $[r/2]$ bits = 160/2 = 80 (squaring)

Sixth iteration:

$r \leftarrow$ (shift 514 to right by 3 bits) = (0101000000) = 320

Rotate *REG2* to left by $[r/2]$ bits = 320/2 = 160 (squaring)

Seventh iteration:

$r \leftarrow$ (shift 514 to right by 2 bits) = (1010000000) = 640

Rotate *REG2* to left by $[r/2]$ bits = 640/2 = 320 (squaring)

Eighth iteration:

$r \leftarrow$ (shift 514 to right by 1 bit) = (0100000001) = 257

Rotate *REG2* to left by $[r/2]$ bits = 257/2 = 128 (squaring)

Final iteration:

$r \leftarrow$ (shift 514 to right by 0 bits) = (1000000010) = 514

Rotate *REG2* to left by $[r/2]$ bits = 514/2 = 257 (squaring)

Squaring = 5 + 10 + 20 + 40 + 80 + 160 + 320 +128 + 257 + 1 (number of 1's in the first 9 bits of 514) + 1 (after all iterations are finished) = 1022

Multiplications = 9 (number of iterations) + 1 (number of 1's in the first 9 bits of 514) = 10.

Comparing the 350-bit design and the 515-bit design reveals that although the 350-bit design has more multiplications than the 515-bit design, it has fewer LUTs because it performes more exponential decomposing and its input has fewer bits. As for the 173-bit and 233-bit designs: $log_2(172) = 7$, $log_2(232) = 7$, 172 = (10101100), 233=(11101000). As these have the same logarithm and the same number of 1's in their binary representation , they have the same number of multiplications (7 iterations + 3 number of 1's in the first 7 bits) = 10. Therefore, the 233-bit design has more LUTs because its inputs have more bits.

We conclude that our proposed design yields the best performance when there is more multipications and squaring, because our design reduced the number of LUTs in the 350-bit design by 42%.

5-6. Chapter Summary

In this chapter, we calculated the area cost in terms of NAND gates of the proposed FAU design compared to the standard FAU design and found it saves 35% of the area cost. We also implemented our design and tested it for 173, 233, 350, and 515 bits and compared the number of slice registers and LUTs, and we found that our design reduces the registers by about half compared it to the standard design, which make it desirable for resources-constrained devices. The results were validated in model sim.

Chapter 6:

CONCLUSION

In this thesis, a design was proposed that uses the common parts of the Massey-Omura multiplier and the Itoh-Tsuji inverter to implement a compact GF(2^m) normal basis field arithmetic unit. The proposed design is very attractive for resource-constrained devices when implementing ECCs. The proposed FAU saves 35% of the total number of NANDs. This result was compared to the work done in [8] and our results were 60% better for m=8. It also saved 48-50% of total slice registers as compared to the standard design. The implementation of the design was done by VHDL coding and simulation with an Artix7 XC7A200T FPGA.

Future Work

Although the methodologies and results were quite good, there are many ways to improve upon this work. Designing the FAU chip from high-level VHDL code was a good learning experience for real-world applications, as full-custom chip design is rare. All the utilization data were synthesized from the VHDL description. Another way to improve work on the design would be to try different multiplication algorithms, like the bit-parallel version of the Massey-Omura multiplier [17,18] to see what difference a parallel approach can have on the design in terms of area.

REFERENCES

1. A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A.Vanstone, and T. Yaghoobian, 1993. “Applications of Finite Fields”, (Kluwer Academic Publishers, Boston, MA).

2. R, Lidl, and H. Niederreiter, 1994. “Introduction to finite fields and their applications”, (Cambridge University Press, Cambridge, UK, revised edition).

3. T. F. Al-Somani, and A. Amin, 2006. “Hardware Implementations of GF 2^m arithmetic using normal basis” J. Appl. Sci. 6(6) 1362-1372.

4. R. C. Mullin, I. M. Onyszchuk, S. A.Vanstone, and R. M. Wilson, 1988. “Optimal normal bases in GF(p^m)”, Discrete Appl. Math., 22(2), 149-161.

5. M. Rosing, 1999. “Implementing Elliptic Curve Cryptography” (Manning Publication Co., Shelter Island, NY).

6. N. Koblitz, 1987. “Elliptic curve cryptosystems”, Math. Comput. 48, 203-209.

7. A. J. Menezes, 1993 “Elliptic Curve Public Key Cryptosystems”, (Kluwer Academic Publishers, Dordrecht, The Netherlands).

8. M. Abd-El-Barr, and A. Khattab, 2014. “An efficient pipelined multiplicative inverse Architecture for the AES cryptosystem” Int. J. Inf. Electr. Eng. 4(2), 81.

9. B. Rashidi, M. Sayedi, and R. Farashhani, 2017. “Efficient and low-complexity hardware architecture of Gaussian normal basis multiplication over GF (2^m) for elliptic curve cryptosystems” IET Circ. Devices Syst.,11(2), 103 –112.

10. J. L. Massey, and J. K. Omura, 1986. “Computational method and apparatus for finite field arithmetic” U.S. Patent Number 4,587,627.

11. T. Itoh, and S. Tsujii, 1988, “A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis”, Info. Comput. 78, 171-177.

12. R. Azarderakhsh, K. Jarvinen, and V. Dimitrov, 2014. “Fast inversion in GF (2^m) with normal basis using hybrid-double multipliers” IEEE Trans. Comput. 63(4), 1041-1047.

13. J. Métairie, A. Tisserand, E. Casseau, 2015. “Small FPGA-based multiplication-inversion unit for normal basis representation in GF(2^m)” IEEE Computer Society Annual Symposium on VLSI, 440-445.

14. Secure Technology Alliance. 2017. “About Smart Cards: Introduction: Primer” 7 August 2019.

15. RFID Security. 2008. The Government of the Hong Kong Special Administrative Region document, "https://www.infosec.gov.hk/english/technical/files/rfid.pdf" 7 August 2019.

16. C. Daniel, F. Solenir, and Gledson, O. 2017. "Cryptography in wireless multimedia sensor networks: A survey and research directions" Cryptogr. J. 1(4). doi:10.3390/cryptography1010004

17. A. Reyhani-Masoleh, and M.A. Hasan, 2002. "A new construction of Massey-Omura parallel multiplier over GF (2^m)" IEEE Trans. Comput. 51, 511-520.

18. C.K. Koc, and B. Sunar, 1998 "Low complexity bit parallel canonical and normal basis multiplier for a class of finite fields" IEEE Trans. Comput. 47, 353-356.

19. C. Wang, T. Truong, H. Shao, L. Deutsch, J. K. Omura, and I. Reed, 1985 "VLSI architectures for computing multiplications and inverses in GF(2^m)", IEEE Trans. Comput. 34(8), 709-716.

20. T. F. Al-Somani, "Design and analysis of efficient and secure elliptic curve cryptoprocessors" PhD Dissertation, (King Fahd University of Petroleum and Minerals, Saudi Arabia).