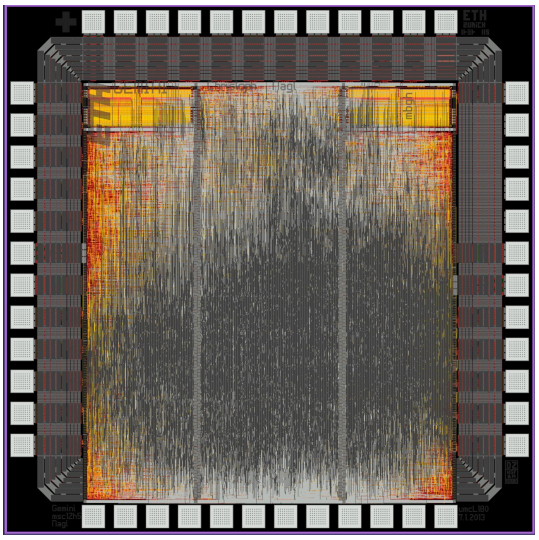
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TU Graz

Master Thesis

A Hardware Architecture for Identity-Based Encryption Using Bilinear Pairings



Christoph Nagl

cnagl@sbox.tugraz.at

May, 2013

Advisors: Dipl.-Ing. Michael Mühlberghuber, ETH Zürich
Dr. Frank K. Gürkaynak, ETH Zürich
Dr. Michael Hutter, TU Graz

Assessors: Prof. Karl-Christian Posch, TU Graz
Prof. Hubert Käslin, ETH Zürich

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, 23 May 2013
Place, Date

Signature

Acknowledgements

First and foremost, I want to thank my advisors Michael Mühlberghuber and Frank Gürkaynak from the Integrated Systems Laboratory at the Swiss Federal Institute of Technology in Zürich. Both supported me throughout the whole project and shared their knowledge and experience, providing valuable advice to accomplish this work. I am also indebted to my advisor Michael Hutter at the Institute for Applied Information Processing and Communications of Graz University of Technology. He supported me in doing this thesis abroad and spent valuable time to provide corrections to the final draft of this thesis. I also wish to thank my assessors at Graz University of Technology and at the Swiss Federal Institute of Technology, namely Prof. Karl-Christian Posch and Prof. Hubert Käslin for providing me with the opportunity to do this thesis.

Last but not least I want to thank my parents who have supported me during my whole studies and also my girlfriend Petra for her constant support—especially during the last months.

Abstract

Bilinear pairings are the fundamental operations for new cryptographic schemes commonly referred to as pairing-based cryptography. Being based on elliptic-curve cryptography, they provide an abundance of new protocols and schemes such as identity-based encryption, which extend the portfolio of cryptographic applications. However, computation of a bilinear pairing is generally complex and costly. Consequently, efficient and fast algorithms and implementations are of particular interest in order to make pairing-based cryptography applicable.

The main goal of this thesis is to build a hardware architecture implementing a bilinear pairing algorithm, and to explore applicability for low-resource environments such as embedded devices or smart cards. Smart cards are generally resource-constrained environments with a scarce area and power budget, being mainly used in interactive applications so that pairing computation time has to be acceptably low.

This work focuses on a low-area hardware architecture to compute the so-called η_T pairing based on supersingular elliptic curves on finite fields of characteristic two. The proposed architecture uses a microcoded Application Specific Integrated Processor (ASIP) with a minimal instruction set to implement the pairing algorithm. Targeting a high security level of 128 bit requires the arithmetic unit to perform finite-field operations with 1223 bit wide operands. In the given setup the finite-field multiplication is the dominating operation. Therefore, efficient multiplication was especially focused in this thesis. This work investigates Karatsuba- and digit-serial-based multiplier architectures, where the former group performs iterative versions of the Karatsuba algorithm and the latter group applies different digit sizes including the bit-serial case. In total, seven designs were subject to a CMOS 180 nm standard-cell based Application-Specific Integrated Circuit (ASIC) design flow in order to provide an evaluation based on cost figures of area, computation time, power, and energy consumption.

The obtained results demonstrate that the computation of the bilinear η_T pairing is feasible with small chip area of about 50 kGE and within several milliseconds. The given results clearly indicate that pairing-based cryptography is ready for future applications in resource-constrained environments such as smart cards and embedded devices.

Keywords: Bilinear Pairing, η_T Pairing, Binary Finite Field, 128-bit Security Level, Pairing-Based Cryptography, Identity-Based Encryption, ASIC, Hardware Architecture, Iterative Karatsuba Multiplier, Digit-Serial Multiplier

Kurzfassung

Bilineare Abbildungen sind die zentralen mathematischen Operationen in neuen kryptographischen Verfahren. Diese abbildungsbasierten Verfahren erlauben es, eine Vielzahl von Protokollen und Schemata zu realisieren, und so, neue kryptographische Anwendungsgebiete, wie beispielsweise identitätsbasierte Verschlüsselung, zu erschliessen. Da die Berechnung von bilinearen Abbildungen generell aufwändig und kostspielig ist, sind effiziente Algorithmen und Implementationen von speziellem Interesse um solche Verfahren praktisch umzusetzen.

Das wesentliche Ziel dieser Diplomarbeit ist es eine integrierte Schaltung zu entwerfen, welche eine bilineare Abbildung berechnet, und die Anwendbarkeit von bilinearen Abbildungen in ressourcenbeschränkten Umgebungen wie eingebetteten Systemen oder Chipkarten zu ermitteln. Chipkarten bieten generell nur wenig Chipfläche und ein knappes Leistungsbudget, zudem muss die Berechnungsdauer aufgrund vorwiegend interaktiver Anwendungen entsprechend kurz sein.

Diese Arbeit konzentriert sich auf eine flächenminimierte integrierte Schaltung um eine η_T Abbildung zu berechnen, welche auf supersingulären elliptischen Kurven, definiert über endlichen Binärkörpern, basiert. Die entworfene Schaltung nutzt einen anwendungsspezifischen integrierten Prozessor mit minimalem Befehlssatz um eine solche Abbildung zu berechnen. Um Anwendungen auf hoher Sicherheitsstufe zu ermöglichen, ist es notwendig, Körperoperationen mit 1223-bit Operanden durchzuführen. Auf die Implementation der modularen Multiplikation wurde spezielle Aufmerksamkeit gelegt, da diese Operation die Systemeigenschaften maßgeblich beeinflusst. Im Speziellen wurden Multiplizierarchitekturen realisiert, die auf einer mehrfachiterativen Anwendung des Karatsuba-Algorithmus bzw. digit- und bit-seriellen Multiplikation beruhen. Insgesamt wurden sieben verschiedene Architekturen entworfen und in einem CMOS 180 nm standardzellenbasierten Entwurfsprozess entwickelt, wodurch eine konkrete Evaluierung basierend auf Flächenbedarf, Berechnungsdauer, sowie Leistungs- und Energiebedarf gegeben werden kann.

Die Ergebnisse zeigen, dass eine bilineare η_T Abbildung mit einer Schaltungsgröße von zirka 50 kGE und innerhalb weniger Millisekunden umsetzbar ist. Diese Resultate weisen eindeutig darauf hin, dass abbildungsbasierte Kryptographieverfahren zukünftig in ressourcenarmen Umgebungen wie Chipkarten und eingebetteten Systemen anwendbar sind.

Stichwörter: Bilineare Abbildung, η_T Abbildung, 128 bit Sicherheitsniveau, Binärkörper, abbildungsbasierte Kryptographie, identitätsbasierte Verschlüsselung, Integrierte Schaltung, iterative Karatsubamultiplikation, digit-serielle Multiplikation

Contents

Chapter 1

Introduction

Cryptography is the science and discipline of information security. It provides techniques for secure communication by means of cryptographic services such as encryption and decryption. With the advent and widespread deployment of computer technology, digital communication has changed many aspects in every day life and introduced us to the so-called *digital age*. As our personal and economic lives become increasingly dependent on digital communication technology, we are in need of new approaches to build and maintain trust in digital communication and transmitted data. Cryptography is able to provide protection of sensitive data and authentication of remote entities we communicate with. Public-key cryptography has been available for several decades but has neither attained widespread attention nor significant application in everyday communication. Main reasons for this are that a traditional public-key based cryptosystem requires to acquire and maintain public keys of entities we want to communicate with. In traditional public-key cryptosystems both entities need to generate a private/public key pair and exchange their public keys before they can exchange sensitive information. Pairing-Based Cryptography (PBC) provides solutions to circumvent this limitation by so-called *bilinear pairings*. Using bilinear pairings, it is possible to generate a public key from a character string identifying a receiver—like an ordinary email address. Using identification strings to derive public keys is the fundamental concept of so called Identity-Based Encryption (IBE) schemes. In an identity-based cryptosystem, the identification string may be any information which is publicly known a priori to the sender. Hence, the sender does not need to obtain the public key of the receiver but can generate it independently using the identification string of the receiver. Moreover, IBE allows to encrypt messages using an identity-based public key without the requirement that a corresponding private key has been generated before. This allows to generate public keys independently of any antecedent action by the receiving entity. The ability to use simple character strings to gener-

ate public keys provides completely new ways to implement secure communication schemes especially for large and dynamic user populations.

In general, information security is based on security services such as authentication, confidentiality, data-integrity, availability, and non-repudiation.

Authentication Entities entering secure communication need to identify each other by means of authentication (*entity authentication*). Information transmitted over a secure channel may also need to be authenticated to ensure its origin (*data authentication*).

Confidentiality Sensitive information is only accessible to authorized entities. This security service is enabled by the cryptographic primitives of encryption and decryption.

Data-Integrity This service ensures that unauthorized data manipulation, or loss of parts thereof can be recognized.

Availability Sensitive information can be demanded by the user and made accessible at any time.

Non-Repudiation Unauthorized entities may not successfully deny previous commitments or actions.

To provide the aforementioned services, two types of cryptography are used, namely symmetric cryptography and asymmetric cryptography, which we want to discuss briefly in the following.

Symmetric-Key Cryptography In *symmetric-key cryptography*, the keys used for encryption and decryption are based on a secret key known to all entities taking part in the secure communication. Hence, systems of this type are also called *secret-key cryptography*. Customary, we can differ two types of algorithms in secret-key cryptography namely algorithms for stream ciphers and algorithms for block ciphers. Stream ciphers encrypt a single bit or even a single byte at a time allowing to process data streams of arbitrary length. Prominent examples for stream ciphers are RC4 and A5 being used to secure GSM networks. Block ciphers operate block-wise on plaintext and cipher text. Well-known examples of block ciphers are the DEA and the Rijndael cipher, standardized as the Data Encryption Standard (DES) and Advanced Encryption Standard (AES).

While secret-key cryptography offers high security and provides computationally efficient implementations, they do not support the security services of authentication and non-repudiation. Additionally, the key distribution of the shared secret to the participating entities is problematic as a so-called *secure channel* is

required to avoid eavesdropping adversaries to compromise the system. Deploying a symmetric key system on a large scale using a single secret key does not provide confidentiality between the participants and bears a high risk of key compromise. To avoid this, each entity pair combination needs to be attributed with a distinct secret key. Consequently, the number of required keys scales bad with the number of participants in such a system. Furthermore, the set of keys needs to be managed to provide means of changing a key in order to handle keys being compromised. As such, a secret-key system exhibits several key-distribution problems especially for large and dynamic user groups.

Asymmetric-Key Cryptography Modern cryptosystems are based on the assumption that some computational problem is reasonably hard to compute while at the same time being easy in case some secret information is known. Typical examples of computational problems, which are customarily used for cryptographic applications, are the Integer Factorization Problem (IFP), the Discrete-Logarithm Problem (DLP), and the Elliptic-Curve Discrete Logarithm Problem (ECDLP). The popularity of elliptic-curve based cryptosystems is due to the fact that their underlying mathematical problem is harder regarding the computational cost of the most sophisticated algorithms to solve it. As the underlying problem is generally believed to be harder than the other applied problems, key sizes in an ECDLP-based system can be reduced while maintaining an equivalent level of security. Small key sizes are especially interesting in terms of low memory and low bandwidth requirements, which is a considerable advantage of ECDLP-based cryptosystems.

The basic idea of using different keys for encryption and decryption is the basis of *asymmetric-key cryptography*, also called *public-key cryptography*, devised by Whitfield Diffie and Martin Hellman in 1976. A concrete realization of such a scheme was provided by Rivest, Shamir, and Adleman in 1978 by the RSA public-key encryption system based on the discrete logarithm problem. In a public key cryptosystem, two separate keys are considered for each communicating party namely a *private key* and a *public key* which represent a so-called *key pair*. As their names suggest, the private key is supposed to be exclusively known to the respective party it belongs to while the public key is made publicly available. The sender uses the public key of the receiver to encrypt a message to the receiver. In order to do so, the public key of the receiver has to be available to the sender, and the receiver must have generated a key pair. The receiver can then decrypt the message using his private key.

For this scheme to work, the two entities need to exchange their keys prior to secure communication. A solution to the problem of exchanging public keys over an insecure channel was introduced by Diffie and Hellman by the so-called *Diffie-*

Hellman key exchange protocol. However, if a malicious user is able to intercept and manipulate the communication between two parties, it is possible to perform a key-exchange sequence with both sides preventing any direct communication and transparently manipulating any future interaction. In this way, an adversary can get hold of sensitive information originating from both parties. Such a scenario is commonly called a *man in the middle attack* and can be avoided if both parties authenticate themselves to each other by entity authentication in the key exchange phase. Hence, entity authentication is mandatory to maintain a secure public-key cryptosystem. Entity authentication can be accomplished with involving a third entity usually called *trusted authority* (TA) or *certifying authority* (CA). Using so-called *digital signature schemes*, any entity may *sign* a message with its secret key, producing a so-called *signature* or *certificate*. Certificates are issued by the TA when an entity has proven its identity to the TA and its association to a public key. Usually a physical verification step is applied by the TA to determine the correctness of an entity to identity relation. This signature can then be checked by other entities by verifying certificates, as the public key of the TA, which was used to generate the certificate, is publicly available.

Involving a trusted third party, which issues certificates, allows to provide the security service of entity authentication. However, keys and their attributed certificates may get compromised and hence, keys need to get revoked and their certificates invalidated. In case of widespread deployment and dynamic user population of a cryptosystem, the issue of certificate management gets complex and difficult to maintain. Certificates are usually valid only for a certain amount of time and hence, need to get renewed. Also, the secret key of an entity may get compromised which requires to revoke certificates and generate a new one. The issue of certificate management is one major factor which complicates widespread deployment of public-key cryptosystems.

1.1 Identity-Based Encryption

We want to start the following introduction to identity-based encryption by a quote from Adi Shamir who devised the idea for such a scheme [50].

> *"An identity-based scheme resembles an ideal mail system: If you know somebody's name and address you can send him messages that only he can read (...) It makes the cryptographic aspects of the communication almost transparent to the user, and it can be used effectively even by laymen who know nothing about keys or protocols."*
>
> - Adi Shamir

In 2001, Boneh and Franklin introduced an implementation of an IBE scheme using bilinear pairings based on elliptic curves. They defined the IBE-security-model with a natural analogue of the Diffie-Hellman problem, hence referred to as the bilinear Diffie-Hellman (BDH) problem. They define a set of four algorithms to form a complete IBE system. Traditionally, these algorithms are called setup, extraction, encryption, and decryption. The setup and extraction algorithms are used to generate system parameters and calculate private keys respectively. The system parameters generated in the setup step are maintained in a trusted authority usually called the Private-Key Generator (PKG). These system parameters create the whole IBE environment and contain a master secret key which is used by the PKG to derive private keys from a user-identity string. It is vital that this key is kept secret as it allows to generate private keys (cf. key escrow). Other system parameters are made public. With these public parameters, a user can derive public keys from an identity string (e.g., an email address) used to identify entities. The derived public key is then used to encrypt a message. Hence, in contrast to a traditional public-key scheme, in an IBE scheme the public key and private key generation is decoupled. Upon reception of a message, the recipient may obtain his private key from the PKG after authenticating his identity. After successful authentication to the PKG, a private key according to the recipients identity string and the system's master key is calculated. With the private key, the recipient may then decrypt the message.

Setup Initialization of all system parameters including the master secret that is used by the PKG to generate private keys.

Encrypt Uses a public key which can be calculated by any user using the public system parameters to encrypt a message for a given identification string.

Extraction Generation of a private key as a function of the system parameters, the master secret, and an identification string.

Decrypt Encrypted information can be decrypted using a private key for a given identification string calculated by the PKG using the master secret.

In general IBE systems provide the cryptographic service of confidentiality by means of identity-based encryption. However, they do not provide the services of integrity, availability, authentication, and non-repudiation. To provide these services digital signatures based on a traditional public-key system can be used in order to build a hybrid cryptosystem which unifies the strengths and advantages from both technologies [13, 37].

Figure 1.1 illustrates a typical sequence of steps used to encrypt a message in an IBE system. To encrypt a message, the sender requires the public system

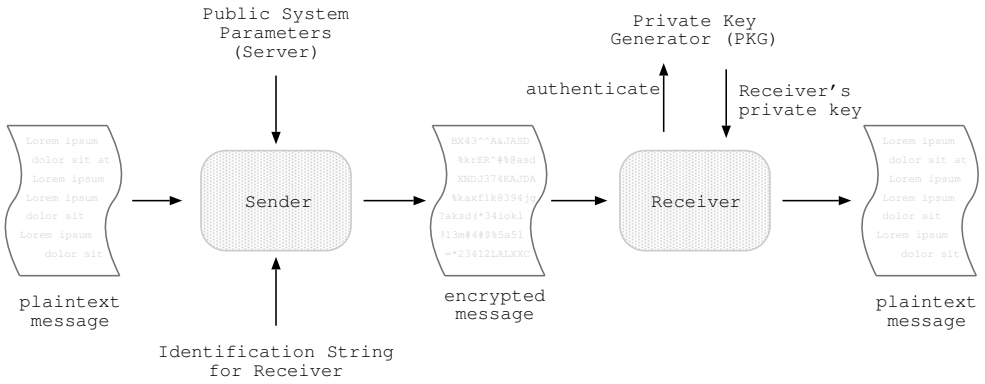


Figure 1.1: Encryption with an IBE system

parameters and an identity string which uniquely identifies the receiver. Using a so-called *bilinear pairing*, the receiver's public key can be calculated from the identity string and the public system parameters. With this public key, the message is encrypted. To decrypt the message, the receiver has to prove his identity to the PKG by authentication in order to receive his private key. It should be noted that after all private keys have been issued, the PKG instance is not required anymore and can theoretically be shut down. However, changing the system parameters allows to have short-lived keys which requires to maintain a PKG instance. Short-lived keys are especially interesting as they allow to avoid the complex issue of certificate management as given in traditional cryptosystems.

Key Escrow One notable property of IBE schemes is their inherent property of key escrow. A common fact in IBE schemes is that the private-key generator has the ability to calculate private keys for any identity within the system. Consequently, an IBE scheme does not provide the cryptographic service of non-repudiation. Also it makes the system's PKG a high value target for attackers. To avoid the problem of key escrow, several methods were proposed, such as:

- Distributed PKG
- Certificate-Less Encryption
- Certificate-Based Encryption

For a more detailed discussion of these methods, we want to refer to [37]. Conversely, key escrow may be a desirable feature in a strictly hierarchical system where an authorized third party requires to have ultimate access to keys in the system. Thus, IBE-based systems allow to realize the legitimate need for large bodies such as governments or international corporations to have access to encrypted

communications within a given system. So far, traditional encryption systems do not provide a technical solution to such use cases.

1.2 Motivation

Bilinear pairings are the key to realize a whole new set of protocols and schemes commonly referred to as pairing-based cryptography. With schemes such as identity-based encryption it is possible to build new systems and extend yet existing ones. Prominent applications of bilinear pairings include identity-based encryption, three-party key exchange, short signature schemes, broadcast encryption, and many others. Identity-based encryption allows to overcome one of the main shortcomings in traditional public-key cryptosystems by addressing the key distribution and management problem, which is of special concern in the case of large and dynamic user populations.

1.3 Contributions and Global Context

- First low-area ASIC implementation of η_T pairing at 128 bit security level in open literature
- Indicates applicability of pairing-based cryptography in resource-constrained environments (standard-cell based UMC 180 nm ASIC design flow)
- Various multiplier designs implemented offering trade-off decisions for future design integration
- Low-area design exploration of large operand multiplication using iterative Karatsuba algorithm, digit-serial, and bit-serial multipliers
- Smallest design compared to related work of 128-bit security ASIC implementations (Chapter 8)

1.4 Outlook

The fundamental concepts which are required to compute bilinear pairings are given in Chapter 2, which briefly introduces to finite fields. Chapter 3 gives a short introduction to elliptic curves with respect to elliptic-curve cryptography. Finite fields and elliptic-curve operations are the operational building blocks to implement bilinear pairings based on elliptic curves. The concept of bilinear pairings

and their basic properties as well as prominent pairing algorithms are covered in Chapter 4. This chapter also describes the pairing algorithms used for the hardware implementation in this thesis and further discusses their computational complexity. Chapter 5 treats various multiplier architectures and presents an application specific integrated processor used as a controlling block in the top-level architecture implementing the η_T pairing. The presented architectures are basically designed according to the main objective of this thesis, which is to develop an application-specific integrated circuit able to compute a bilinear pairing with low requirements on chip area. Chapter 6 covers the description of the ASIC design flow applied to obtain digital designs ready for fabrication. The results of the implemented designs are presented in Chapter 7 discussing figures of area, power, and energy consumption. We give a short overview on previous work of ASIC pairing hardware accelerators at a security level of 128 bit in Chapter 8. Final remarks and conclusions are drawn in Chapter 9. Possible topics for future work on this matter are proposed and shortly described in Section 9.2. Pairing-based cryptography is a relatively young field but yet already rich of protocols and applications based on bilinear pairings. In Section A, we want to highlight some of the most prominent applications and protocols based on bilinear pairings of today. Supplementary material is provided in the appendix.

Chapter 2

Finite Fields

This chapter gives a basic introduction to finite fields providing the basis for elliptic-curve cryptography. It especially focuses on finite fields of characteristic two—also referred to as binary finite field. The binary finite field operations presented in this chapter represent the basis in order to compute the η_T pairing as presented in Chapter 4.

In general, arithmetic in finite fields provides the foundation for many cryptographic systems used today. Operations in finite fields provide a way to construct computationally hard problems such as the discrete logarithm problem. In cryptography, such a problem needs to provide properties of a so-called *trapdoor one-way function* which can be computed efficiently while the inversion is computationally hard without knowledge of supplementary information such as a secret key. Furthermore, finite fields allow to define operations based on elliptic-curve equations where elements of a finite field represent the coordinates for the set of discrete points residing on elliptic curve—fulfilling the elliptic-curve equation. Operations based on elliptic-curve points can then be used to construct the elliptic-curve discrete logarithm problem. In the context of this work, finite field operations are used to calculate a bilinear pairing based on elliptic curves over binary finite fields called the η_T pairing. The following sections give a short introduction on finite fields, especially on binary finite fields in polynomial representation providing the basis to compute the η_T pairing.

In abstract algebra, a finite field describes an algebraic structure with a finite number of elements. A finite field is usually denoted by $\mathbb{F}_q$ or $\mathrm{GF}(q)$ for Galois field in reference to Évariste Galois—the founder of finite field theory. The order of a field is indicated by q which is the number of elements in the field. A finite field of order q exists if q is of prime power. Consequently, we can categorize finite fields by the form of q representing p^n where p is a prime number and n is a positive integer. Furthermore, p is called the characteristic of the field and n the extension

of the field. Depending on field characteristic and the field extension we have finite field categories of the following:

- $\mathbb{F}_p$ with $n = 1$ has a prime number of elements and hence is called *prime field*
- $\mathbb{F}_{p^n}$ with $n > 1$ is extended by m and hence is called *extension field*.
- $\mathbb{F}_{p^{m \cdot n}}$ with $m, n > 1$ denote a field of composite extension degree called *composite extension field*.
- $\mathbb{F}_{2^n}$ is a special case of an extension field. Having a base of 2 it is called *binary extension field*.
- $\mathbb{F}_{3^n}$ is a special case of an extension field called *ternary extension field*.

The pairing presented in this work is based on a binary finite extension field of extension degree 1223 denoted as $\mathbb{F}_{2^{1223}}$.

2.1 Properties

A field is an algebraic structure of a set $\mathbb{F}$ together with two operations, addition (denoted by $+$) and multiplication (denoted by $\cdot$) both satisfying usual arithmetic properties. These properties are given in the following:

- Associativity
  - Additive: $a + (b + c) = (a + b) + c$ for all $a, b, c \in \mathbb{F}_q$
  - Multiplicative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in \mathbb{F}_q$
- Commutativity
  - Additive: $a + b = b + a$ for all $a, b \in \mathbb{F}_q$
  - Multiplicative: $a \cdot b = b \cdot a$ for all $a, b \in \mathbb{F}_q$
- Identity element
  - Additive: e_a where $a + e_a = a$ for all $a \in \mathbb{F}_q$
  - Multiplicative: $e_m \neq e_a$ where $a \cdot e_m = a$ for all $a \in \mathbb{F}_q$
- Inversion element
  - Additive: For any $a \in \mathbb{F}_q$ there exists an additive inverse element $-a$ such that $a - a = e_a$

  - Multiplicative: For any $a \neq e_a \in \mathbb{F}_q$ there exists a multiplicative inverse element a^{-1} such that $a \cdot a^{-1} = e_m$

- Distributivity
  - $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}_q$

Extension of fields Let K and L be finite fields. If there exists a field homomorphism from K into L, the field L is called an *extension field* of K. This extension field relation is denoted as L/K [13]. With a quadratic non-residue s in the binary field, we can construct a basis $1, s$ to build an extension field. Hence, the constructed field is called a quadratic extension binary field. A representation of a quadratic extension binary field element has twice the size of an ordinary extension binary field element. Combining the quadratic non-residue s of the binary field and a quadratic non-residue t of the quadratic extension field allows to build the basis $1, s, t, st$. Using this basis, we can represent a quartic extension binary field element which has quadruple size compared to an ordinary extension binary field element. The extension field $\mathbb{F}_{q^4}$ is represented using a tower of extensions $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$ and $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2 + v + u)$ with the basis for $\mathbb{F}_{q^4}$ over $\mathbb{F}_q$ is $\{1, u, v, uv\}$. For the binary field, we can formulate a quadratic $\mathbb{F}_{2^{2m}}$ and quartic $\mathbb{F}_{2^{4m}}$ extension field on the basis of $(1, s, t, st)$ where $s^2 = s + 1$ and $t^2 = t + s$ [5]. In the remainder of this thesis, operations using the field extension are summarized as *extension field arithmetic*.

2.2 Binary Field Arithmetic

In the following, we describe the set of operations required to calculate the η_T pairing in a binary finite extension field represented in polynomial representation. The given operations provide the fundamental arithmetic for elliptic curve based calculations and higher extension field arithmetic.

A binary extension field element in $\mathbb{F}_{2^m}$ is represented by polynomials of degree less than m where the coefficients of the polynomial are in the field $\mathbb{F}_2$. Equation 2.1 denotes a general definition for binary extension field elements [34].

$$\mathbb{F}_{2^m} = a(x) | a(x) = a_{m-1} \cdot x^{m-1} + \cdots + a_1 \cdot x + a_0, a_i \in \mathbb{F}_2 \tag{2.1}$$

Considering hardware implementation, the arithmetic in binary fields has the advantage that the operation of addition can be represented by a simple exclusive-or operation so that no carry propagation issue occurs in contrast to arithmetic of prime fields. As carry propagation usually contributes to the longest path delay in an adder circuit, binary finite-field arithmetic provides fast implementations. Also

field subtraction is equivalent to the addition operation which is also an advantage. As we will see later in this section, also the squaring operation can be realized very efficiently. The fact that these operations can be implemented efficiently is important to implement a cost intensive computation as bilinear pairings. The set of operations mandatory to calculate a pairing depends on the actual pairing and the algorithmic description thereof. Early algorithms to compute the η_T pairing were based on finite field operations such as calculating a finite field square root. More recent versions of the η_T pairing were optimized so that square root calculation is not required. As the computation of square roots introduces additional hardware circuitry, the square-root free version was favored. So in order to implement the η_T pairing, we discuss the following operations:

- Addition/Subtraction,
- Multiplication,
- Squaring,
- Reduction,
- Inversion.

In so-called polynomial representation, binary field elements correspond to polynomials where the coefficients are given as binary elements. So binary polynomials have coefficients of either 0 or 1 allowing to be represented by a simple bit string. Consequently, the field $\mathbb{F}_{2^m}$ contains 2^m binary polynomials where each polynomial has a maximum degree of $m-1$ which we can represent by bit strings of length m.

Operations in higher extension degree as required for calculating the η_T pairing can be calculated in the base extension field $\mathbb{F}_{2^m}$. The algorithmic descriptions of higher extension degree elements is given in Chapter 4.

2.2.1 Addition/Subtraction

Finite field addition of elements in polynomial representation correspond to an addition of polynomials where the coefficient arithmetic is performed modulo 2. Consequently, we can add two binary field elements with a bit-independent exclusive-or operation. Regarding a hardware implementation, a bit-wise XOR operation is sufficient to obtain the modular sum of two finite field elements. Other than in the case of prime field arithmetic, there is no carry propagation and no modular reduction step involved. This makes the addition over binary fields very attractive for hardware implementation.

2.2.2 Multiplication

Multiplication in a binary finite field corresponds to a logical AND operation of the multiplier coefficients and the multiplicand. To obtain a modular result multiplication of binary field elements is performed modulo an irreducible polynomial $f(x)$. The implementation of finite field arithmetic and further description is contained in Chapter 5. For further reference on binary finite field multiplication we want to refer to [25] and [34].

2.2.3 Squaring

Squaring a binary polynomial is a linear operation which can be implemented efficiently. As the coefficients of a binary polynomial are powers of two, the squaring operation can be performed by doubling the degree of each coefficient of a polynomial. Given a binary polynomial of degree $m-1$: $a(x) = a_{m-1} \cdot x^{m-1} + a_{m-2} \cdot x^{m-2} + \cdots + a_1 \cdot x^1 + a_0$ we find the squared polynomial by

$$a(x)^2 = a_{m-1} \cdot x^{2m-2} + a_{m-2} \cdot x^{2m-4} + \cdots + a_1 \cdot x^2 + a_0. \tag{2.2}$$

So the squaring operation can be realized by expanding the coefficients of a polynomial and inserting elements with zero-valued coefficients according Equation 2.2. For a hardware implementation, squaring is basically a matter of re-wiring the signals representing the polynomial coefficients. Hence, a squaring operation can be easily performed within one clock cycle and little area overhead. A modular squaring also requires to perform a reduction step to obtain an element of the given finite field. This reduction is performed modulo the irreducible polynomial which defines the given finite field.

2.2.4 Inversion

Among the given set of finite field operations of addition, multiplication, and squaring, the inversion is the most time-consuming one. Inversion is used to obtain the multiplicative inverse of a non-zero finite field element. The multiplicative inverse is the element a^{-1} so that $a^{-1} \cdot a \equiv 1 \bmod f(x)$ where $a \in \mathbb{F}_q$. Inversion algorithms can be categorized into two groups. The first group is based on the extended Euclidean algorithm and its variants [25] and the second group are inversion algorithms based on field multiplication. Typically the following algorithms are considered for finite field inversion:

- Fermat's little theorem,
- Itoh-Tsujii's algorithm [28, 47],

- Almost Inverse algorithm [24, 49].

As low area is a major design goal, we focus on field inversion by application of Fermat's little theorem as it allows us to calculate an inversion by applying repeated multiplications and squarings. Hence, we need no additional hardware for the inversion operation effectively minimizing area consumption. In the following, we want to discuss Fermat's little theorem and its application according to the given finite field and reduction polynomial.

Fermat's Little Theorem

The main advantage in using Fermat's little theorem is that its application does not require implementation of additional hardware. In fact, it allows to re-use arithmetic of multiplication and squaring to calculate the inversion. While inversion based on Fermat's little theorem is not very fast, it provides considerable advantages concerning circuit area. Low-area designs can make use of Fermat's little theorem to trade computation time for circuit area. For an algorithmic computation where the number of inversions is low compared to other operations, it may be advantageous to save circuit area by applying Fermat's little theorem for a better overall area-time trade-off. Another reason to avoid implementing a distinct inversion circuit is that it could potentially increase the critical path delay and effectively decrease the performance of the whole circuit.

Fermat's little theorem given in Equation 2.3 provides the basic formula to calculate an inversion using field operations of multiplication and squaring.

$$a^{-1} = a^{2^m-2}, \quad a \in \mathbb{F}_q \tag{2.3}$$

There exist several ways to compute the exponentiation by $2^m - 1$ which we want to discuss in the following. Given the case that we want to calculate x^n, where n is a positive integer, we could either make $n-1$ consecutive multiplications or multiply the squared partial results. To obtain x^{16}, we would calculate $x \cdot x$, $x^2 \cdot x^2$, $x^4 \cdot x^4$, $x^8 \cdot x^8$ and hence obtain x^{16} by only 4 multiplications. We can apply this idea to general values of n using a method called *binary exponentiation* [32]. By applying repeated multiplications and squarings to the non-zero field element a, the finite field can be traversed much more efficiently to obtain the inverse. With $2^m - 2 = \sum_{i=1}^{m-1} 2^i$ we can write Fermat's little theorem as

$$a^{-1} = a^{\sum_{i=1}^{m-1} 2^i} = \prod_{i=1}^{m-1} a^{2^i} \tag{2.4}$$

to compute the inverse field element a^{-1} with $m-1$ squarings and $m-2$ multiplications. The simple square and multiply algorithm is very easy to implement

but has performance drawbacks compared to the solution in Algorithm 1. For the fixed reduction trinomial $x^{1223} + x^{255} + 1$ we can directly asses the computational cost in terms of field multiplications and squarings for the inversion algorithm. The finite field operations in the simple square and multiply algorithm are related to the Hamming weight of $2^m - 2$ where m is 1223 as given by the irreducible polynomial $f(x)$. The binary representation of $2^m - 2$ is a 1223-bit wide number where all bits, excluding the least significant bit, are set to 1. Applying the square and multiply algorithm, we substitute each 1 in the representation of $2^m - 2$ by a square and multiply operation and each 0 by a sole squaring operation. To obtain the correct calculation sequence, the first square and multiply pair is not executed. This approach gives an inversion cost of 1221 multiplications and 1222 squarings for the given field size. Algorithm 1 gives an improved way to obtain an inversion result using Fermat's little theorem which is applicable if the extension degree m is odd.

Algorithm 1 Inversion in $\mathbb{F}_{2^m}$

Input: $u \in \mathbb{F}_{2^m}, u \neq 0, m$ is odd
Output: $v = u^{-1} \in \mathbb{F}_{2^m}$.
1: $U \leftarrow u^2$
2: $V \leftarrow 1$
3: $x \leftarrow (m-1)/2$
4: **while** $x \neq 0$ **do**
5: $\quad U \leftarrow U \cdot U^{2^x}$
6: $\quad$ **if** x is even **then**
7: $\quad\quad x \leftarrow x/2$
8: $\quad$ **else**
9: $\quad\quad V \leftarrow V \cdot U$
10: $\quad\quad U \leftarrow U^2$
11: $\quad\quad x \leftarrow (x-1)/2$
12: $\quad$ **end if**
13: **end while**
Return V

$\Sigma = (15M, 1222S)$

As multiplications are costly operations we want to minimize them as far as possible as the pairing computation is by itself a costly computation demanding efficient implementation. Observing that the approach of Algorithm 1 in [25] requires considerably less multiplications allows us to reduce the computational cost for inversion in $\mathbb{F}_{2^m}$. With the improved algorithm we can perform a $\mathbb{F}_{2^m}$ inversion with just 15 multiplications and 1222 squarings.

2.2.5 Modular Reduction

To obtain a modular result, operations of multiplication and squaring are performed modulo an irreducible reduction polynomial $f(x)$. Modular reduction can be performed efficiently by a linear operation for certain types of sparse reduction polynomials such as trinomials and pentanomials. Hence, the ANSI X9.62 and X9.63 standards [2, 3] recommend using such reduction polynomials. Several fast modular-reduction algorithms tailored to such polynomials are illustrated in [25].

In the following, we want to focus on modular reduction using the irreducible polynomial as used to calculate the η_T pairing presented in Chapter 4. We will illustrate how the specific property of this reduction polynomial can be used to be implemented by a combinatorial network, allowing to reduce polynomials in a single computation cycle. The η_T pairing is based on a binary finite field $\mathbb{F}_{2^{1223}}$ and a irreducible reduction trinomial $f(x) = x^m + x^\alpha + x^\beta = x^{1223} + x^{255} + 1$. Fast reduction by trinomials is possible if the second non-zero coefficient denoted as x^α satisfies the relation of $2\alpha - 2 < m$, where m is the order of the irreducible polynomial. As the irreducible trinomial used to compute the η_T pairing fulfills this condition, a fast reduction can be defined. The steps applied to reduce by the given trinomial are illustrated in Figure 2.1 and described in the following:

$$f(x) = x^m + \underbrace{x^\alpha + x^\beta}_{r(x)}. \tag{2.5}$$

Let the finite field be constructed by a reduction polynomial of the form $f(x) = x^m + x^\alpha + 1$ and let $a(x)$ and $b(x)$ be elements of GF(2^m), where $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{i=0}^{m-1} b_i x^i$. Hence, the plain polynomial product $c(x) = a(x) \cdot b(x)$ has degree $2m - 2$ which needs to be reduced to the modular product $c'(x) = c(x) \bmod f(x)$ of degree m-1.

$$\begin{aligned}
c(x) &= c_{2m-2}x^{2m-2} + \ldots + c_1 x + c_0 \\
&= \underbrace{c_{2m-2}x^{2m-2} + \ldots + c_{m+1}x^{m+1} + c_m x^m}_{c_H(x)} + \underbrace{c_{m-1}x^{m-1} + \ldots + c_1 x + c_0}_{c_L(x)} \\
&= (c_{2m-2}x^{m-2} + \ldots + c_{m+1}x^m + c_m)x^m + c_L(x) \\
&\equiv (c_{2m-2}x^{m-2} + \ldots + c_m) \cdot r(x) + c_L(x) \bmod f(x) \\
&= (\underbrace{c_{2m-2}x^{m-2} + \ldots + c_m}_{(1)}) \cdot (x^\alpha + \underbrace{x^\beta}_{1}) + c_L(x) \\
&= (\underbrace{c_{2m-2}x^{m-2+\alpha} + \ldots + c_m x^\alpha}_{(2)}) + (\underbrace{c_{2m-2}x^{m-2} + \ldots + c_m}_{(1)}) + c_L(x)
\end{aligned} \tag{2.6}$$

In Equation 2.6, we apply Equation 2.5 once giving us two terms where $c_L(x)$ is the lower part of the product which does not need to be further reduced and

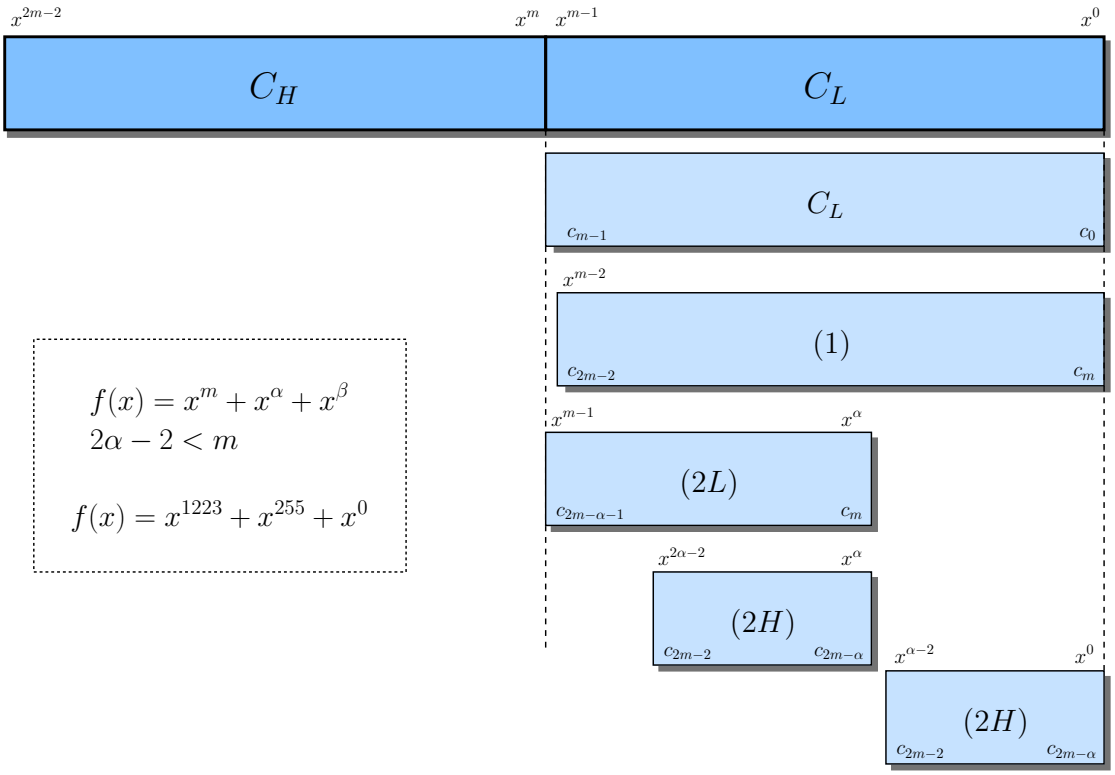


Figure 2.1: Fast reduction by trinomial $f(x) = x^{1223} + x^{255} + 1$

the second term (1) which exceeds degree x^m and thus requires further reduction. Consequently, we apply Equation 2.5 again reducing term (2) to obtain the fully reduced result.

$$\begin{aligned}(2) &= c_{2m-2}x^{m-2+\alpha} + \ldots + c_{2m-\alpha}x^m + c_{2m-\alpha-1}x^{m-1} + \ldots + c_m x^\alpha \\ &\equiv \underbrace{(c_{2m-2}x^{\alpha-2} + \ldots + c_{2m-\alpha})}_{(2H)} \cdot x^m + \underbrace{c_{2m-\alpha-1}x^{m-1} + \ldots + c_m x^\alpha}_{(2L)} \\ &= (c_{2m-2}x^{\alpha-2} + \ldots + c_{2m-\alpha}) \cdot \underbrace{(x^\alpha + 1)}_{r(x)} + (2L) \\ &= (c_{2m-2}x^{2\alpha-2} + \ldots + c_{2m-\alpha}x^\alpha) + (c_{2m-2}x^{\alpha-2} + \ldots + c_{2m-\alpha}) + (2L)\end{aligned}$$

The reduced result $c'(x)$ is obtained by summing up the terms obtained by repeated reduction by the trinomial as given in Equation 2.7. In the binary finite field, the addition operation is equivalent to an XOR operation. So in order to reduce the product, we can use a combinatorial XOR network which allows to perform reduction in a single step, i.e.,

$$c'(x) = c_L(x) \oplus (1) \oplus (2L) \cdot x^\alpha \oplus (2H) \cdot x^\alpha \oplus (2H). \tag{2.7}$$

By summing up the overlapping coordinates of the partially reduced terms, we can estimate the complexity of the reduction circuit in required XOR gates. Adding the segments requires $m + (m - \alpha) + 2\alpha$ XOR gates. Consulting the standard-cell datasheet, we see that the 2-input XOR-gate standard cell of normal driving strength requires nine cell units where the 2-input NAND standard cell of normal driving strength requires 3 cell units. Normalizing for 2-input NAND gate equivalents we obtain an estimated area of $2701 \times 3 = 8103\,\mathrm{GE}$ for the reduction circuit.

Chapter 3

Elliptic Curves

This chapter introduces elliptic curves based on finite fields as a basis to compute bilinear pairings. Special focus is given on supersingular elliptic curves which provide the basis for the pairing which has been implemented in this thesis.

Elliptic curves are the enabler technology for an efficient implementation of bilinear pairings. Elliptic-curve cryptography provides several types of curves, different point-coordinate representations, and consequently also a large set of algorithms and optimizations to perform elliptic-curve arithmetic. Bilinear pairings can be calculated efficiently using elliptic-curve theory. Especially so-called supersingular elliptic curves play an important role in the context of pairing-based cryptography. In the following, we want to briefly introduce elliptic curves and the basic arithmetic on elliptic curves providing the basis for bilinear pairings based on elliptic curves.

The introduction of elliptic curves to cryptography was proposed in 1985 by Victor Miller [40] and 1987 by Neal Koblitz [33]. Since then, elliptic-curve cryptography provided a wide area for research and found numerous applications. One main reason is that elliptic-curve cryptography offers a higher level of security per bit than other comparable asymmetric cryptographic primitives such as RSA. Hence, elliptic-curve cryptography requires considerably smaller key sizes at the same security level which in turn reduces cost figures. Being attractive for applications in low-resource environments such as smart cards, RFID tags, or even sensor-node networks, elliptic-curve cryptography is still a very active research area.

Elliptic curves are be defined over a finite set of points which are elements of a finite field defined by an irreducible polynomial. As such, an elliptic curve E is said to be defined *over* a finite field K which is denoted as E/K where K is called the *underlying field*. The set of points of a curve form an associated Abelian group which allows mathematical operations on that curve or set of points. The set of points form a cyclic subgroup G on that curve. The fundamental equation

to elliptic curves is the so-called *Weierstraß equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where the coefficients a_{1-6} are elements of K and x, y are the coordinates of a point P on that curve where $x, y \in K$. The set of points on E over K is referred to as K-rational points on E, denoted as $E(K)$ and defined by

$$E(K) = \{(x, y) \in K^2 : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} U \{\infty\}$$

where ∞ is called the *point at infinity*. In its *projective form* $[X : Y : Z]$ three coordinates are used to describe a point that satisfies the equation. By substituting X for $\frac{X}{Z}$ and Y for $\frac{Y}{Z}$ the equation can be transformed to its *affine form*. To describe an elliptic-curve point in affine coordinate representation, only two coordinate elements are required. Finite field elements satisfying this equation represent points on the elliptic curve. Depending on the finite field characteristic, the elliptic-curve equation can be simplified through a change of variables. For a finite field K of characteristic two, we obtain two simplified Weierstraß equations depending on the value of a_1. For $a_1 \neq 0$ we obtain Equation 3.1 which is said to be *non-supersingular* and for $a_1 = 0$ we obtain Equation 3.2 which is said to be *supersingular* with $a, b, c \in K$ [25].

$$E : y^2 + xy = x^3 + ax^2 + b \tag{3.1}$$

$$E : y^2 + cy = x^3 + ax + b \tag{3.2}$$

Group Law Operations on points of an elliptic curve are performed according to the group law of a curve E defined over a field K. The elementary operation is the addition of two points residing on the curve giving another point also resides on that curve. The point addition is performed according to the so-called *chord-and-tangent* rule which can be demonstrated geometrically for elliptic curves defined over real numbers $\mathbb{R}$. The set of points on the curve and the addition operation together with the element ∞ form an Abelian group where ∞ is the group's neutral element. The operations performed on this group are usually referred to as *elliptic-curve arithmetic* and represent the basis of elliptic-curve cryptosystems. Based on point addition, another operation called point doubling can be defined which also applies a version of the chord-and-tangent rule. Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ residing on an elliptic curve E/K we may write the point addition as $P + Q = R$ denoting that the result is a third point $R = (x_3, y_3)$. Doubling a point P is denoted as $2 \cdot P = R$.

Group Order Given an elliptic curve defined over a field $E/\mathbb{F}_q$, the number of points on that curve is denoted as $\#E/\mathbb{F}_q$ and referred to as the *order* of a curve

E over $\mathbb{F}_q$. Hasse's theorem provides a way to define an interval for the order of a curve.

Definition 1. *Let p be the characteristic of $\mathbb{F}_q$ and t be the trace of $E/\mathbb{F}_q$. An elliptic curve E defined over $\mathbb{F}_q$ is called* supersingular *if p divides t where t is the so-called* trace *of E over K.*

The trace t of a curve is defined through *Hasse's theorem* which states that for a curve $E/\mathbb{F}_q$ the order of that curve $\#E/\mathbb{F}_q = q+1-t$ where $|t| \leq 2\sqrt{q}$. Since the value of $2\sqrt{q}$ is small relative to q the number of points in $E(\mathbb{F}_q)$ is approximately q [25].

3.1 Supersingular Curves

Supersingular elliptic curves have played a key role in the history of bilinear pairings. Originally supersingular curves were introduced by Koblitz and Miller and later avoided in cryptographic applications with the discovery of so-called MOV attacks [38]. The reductions of the discrete-logarithm problem in [38] used the fact that supersingular curves have a small embedding degree of $k \leq 6$. These attacks apply bilinear pairings to attack the discrete logarithm problem of supersingular curves. Lately, with the advent of constructive applications of bilinear pairings, supersingular curves are used again. Prominent examples of constructive applications of pairing-based cryptography are given in Section A.

Definition 2. *An elliptic curve $E/\mathbb{F}_q$ where $q = p^m$ with p being prime and $m \in \mathbb{N}$ with $\#E(\mathbb{F}_q) = q+1-t$ is called* supersingular *if and only if the greatest common denominator of t and q is larger than 1 [18].*

Supersingular curves offer a distortion map ϕ which maps a point P to another point $\phi(P)$. Ordinary curves with embedding degree $k > 1$ do not have distortion maps. For supersingular curves a distortion map always exists. A distortion map ϕ is used to transform elements of $E(\mathbb{F}_q)$ to $E(\mathbb{F}_{q^k})$

$$\phi : E(\mathbb{F}_q[\ell] \rightarrow E(\mathbb{F}_{q^k})[\ell]$$

Definition 3. *Let n be a number prime to q. The smallest number k such that $n|q^k - 1$ is called the embedding degree with respect to n [13]. For supersingular elliptic curves k is always small.*

Definitions of supersingular elliptic curves over finite fields of $\mathbb{F}_p$, $\mathbb{F}_{2^m}$, and $\mathbb{F}_{3^m}$ with corresponding embedding degree k [6]:

$$E/\mathbb{F}_p : y^2 = x^3 + (1-b)x + b, \quad b \in 0, 1, N = p+1, k = 2;$$
$$E/\mathbb{F}_2 : y^2 + y = x^3 + x + b, \quad b \in 0, 1, N = 2^m + 1 \pm 2^{m+1/2}, k = 4;$$
$$E/\mathbb{F}_3 : y^2 = x^3 - x + b, \quad b \in -1, 1, N = 3^m + 1 \pm 3^{(m+1)/2}, k = 6;$$

Availability of distortion maps is also important as they make the pairing input elements linearly independent. With the input elements P and Q being linearly independent, we obtain a strong non-degeneracy property which means that the self pairing is not trivial. For cryptographic applications, pairings have to be non-degenerate. Many protocols require a pairing of two inputs from the same cyclic group to be non-degenerate. While distortion maps exist for supersingular curves, they do not exist for ordinary elliptic curves. In Page et al. [43], compare the efficiency of supersingular curves and ordinary elliptic curves in pairing-based cryptosystems. One drawback of ordinary curves is the inavailability of distortion maps. A distortion map transform elements of $E(\mathbb{F}_q)$ to $E(\mathbb{F}_{q^k})$ so that arithmetic can be performed efficiently in $E(\mathbb{F}_q)$ and then mapped to $E(\mathbb{F}_{q^k})$. However, for ordinary curves hash functions are used to take their values in $E(\mathbb{F}_{q^k})$ so that the hashed point is defined over $E(\mathbb{F}_{q^k})$ where arithmetic is slower than on $E(\mathbb{F}_q)$ [13]. The inavailability of distortion maps on ordinary elliptic curves can be a problem as some security proofs are based on the existence of distortion maps. So for a protocol do be provable in these terms one must stick to supersingular curves [18].

Comparing supersingular curves with Barreto-Naehrig curves, which are also applied for implementing bilinear pairings, shows that supersingular curves have simpler curve arithmetic than Barreto-Naehrig curves and are more efficient due to their efficient formulæ for point tripling if used over a ternary finite field. The field arithmetic of supersingular curves generally benefits from a small characteristic as carry propagation considerations are not necessary for small characteristics—making supersingular curves better suited for implementations in hardware. The *embedding degree* k of supersingular curves is generally small. The supersingular curves over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$ have an embedding degree of 4 and 6 respectively. The relatively low embedding degree requires to use a larger field size in the multiplicative group $\mathbb{F}_{q^k}$ to keep the security level.

Chapter 4

Bilinear Pairings

This chapter starts with an introduction to bilinear pairings, which are the key component in pairing-based cryptography, and several applications thereof. Next, we will discuss the properties of a bilinear pairing regarding a constructive cryptographic application. We continue with a presentation of pairing algorithms for identity-based encryption such as the Weil, Tate, and η_T pairing. Special focus is given on the *truncated eta* or short η_T pairing as it is an optimized version of the Tate pairing allowing efficient implementations in hardware. The description of the η_T pairing also contains algorithmic descriptions and a short computational analysis representing the basis for design considerations concerning the hardware architectures given in Chapter 5.

4.1 Introduction

Pairing-based cryptography is based on mappings between two algebraic groups. A so-called bilinear map or *bilinear pairing* allows to construct such a map between two groups. Such a map is especially interesting in cryptography as it allows to also map the computational problems attributed to these algebraic groups. So if a problem definition is mapped to another usually easier problem in another group, this effectively reduces the cryptographic strength of the former. This approach is usually referred to as *reduction* and was exploited in early cryptographic applications of bilinear pairings. If the finite-field discrete logarithm problem is computationally more feasible than its corresponding elliptic-curve discrete logarithm problem, bilinear pairings can be used to effectively reduce the security level of a cryptosystems. Bilinear maps were introduced to cryptography in 1991 by Menezes, Okamoto and Vanstone [38] and 1994 by Frey and Rück [19] to reduce the elliptic-curve discrete logarithm problem to a discrete logarithm problem in the multiplicative group of a finite field (Figure 4.1). Until then no sub-exponential

time algorithm was known for solving the elliptic-curve discrete logarithm problem. The so-called MOV-attack uses the Weil pairing to reduce the elliptic-curve discrete logarithm problem to the logarithm problem in the multiplicative group of an extension of the curve's underlying finite field. For the class of supersingular elliptic curves this reduction provides probabilistic sub-exponential time algorithms to solve the elliptic curve discrete logarithm problem. The first group is usually referred to as *gap group* because the decisional Diffie-Hellman problem [9] gets easier due to the reduction to a computationally weaker problem in the second group. While the decisional Diffie-Hellman problem of the gap group is weakened the computational Diffie-Hellman problem is not.

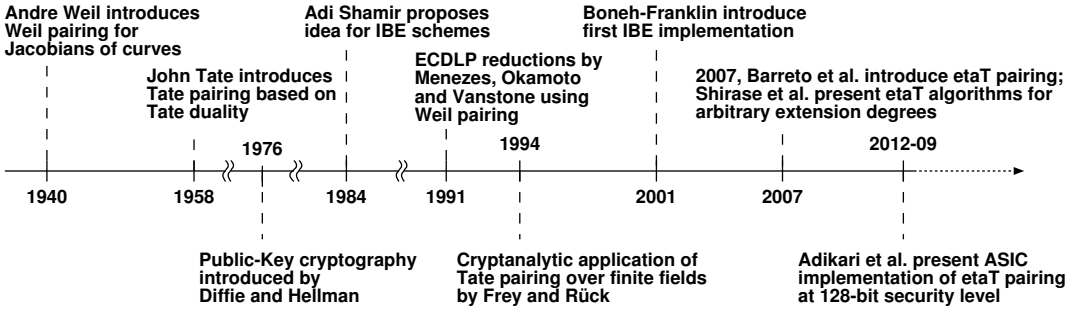


Figure 4.1: Historical timeline of bilinear pairings

A cryptographic pairing represents a bilinear map which maps two group elements to an element of a third group. Given that G_1,G_2,G_T are groups of large prime order q the map e can be denoted as

$$e : G_1 \times G_2 \rightarrow G_T. \tag{4.1}$$

The groups G_1 and G_2 are generally additive groups and G_T a multiplicative group. In the following we will denote the generators of the additive groups as P and Q respectively. In additive notation, we have a scalar multiple aP given as

$$aP = \overbrace{P + P + \ldots + P}^{\text{a times}} \tag{4.2}$$

which can be used with another scalar multiple of a generator in G_2 to be mapped to an element of the additive group G_T. Pairings are usually classified into different types. We want to briefly give the classification as introduced in [20]. If two elements of the same group are used to map to the target group G_T, the pairing is called *symmetric* or Type-I pairing. If the input groups differ $G_1 \neq G_2$, the pairing is called *asymmetric*. Asymmetric pairings are divided into pairings where there exists an efficient distortion map (cf. homomorphism) $\phi : G_2 \rightarrow G_2$ (Type II

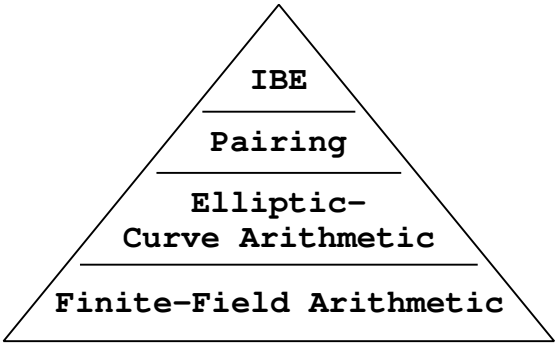


Figure 4.2: Abstraction layers for pairing-based cryptography

pairing) and where there is no such efficiently computable homomorphism between G_1 and G_2 (Type III pairing):

$$e : G_1 \times G_2 \to G_T \text{ where } G_1 = G_2. \tag{4.3}$$

In the following, we generally consider Type II pairings which can be generalized to Type I pairings if required (Equation 4.3).

Due to the bilinear property of pairings, new cryptographic schemes can be constructed which in turn extend the portfolio of cryptographic applications. Prominent applications of pairing-based cryptosystems are

- Identity Based Encryption,
- Tripartite Key Exchange, and
- Short signatures.

Identity-based encryption was envisioned in 1984 by Adi Shamir [50] and instantiated in 2001 by Boneh and Franklin [10]. Antoine Joux introduced the one-round tripartite key-exchange protocol in [29]. Boneh, Lynn, and Shacham provided the short signature schemes in [12]. Boneh, Gentry and Waters presented efficient broadcast encryption systems in [11]. More detailed descriptions of pairing-based protocols are presented in Section A. The pairing algorithms according to Weil and Tate contain fairly complex mathematics. From an abstract perspective, the pairing computation is based on finite fields and elliptic-curve operations and provides a basis for pairing-based protocols such as IBE (Figure 4.2).

Security The security level of a pairing-based cryptosystem relies on the hardness of two computational problems. First, the discrete logarithm problem in the additive group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points on E (cf. computational elliptic-curve

Diffie-Hellman problem, ECDHP). Second, the discrete logarithm problem in the multiplicative group $\mathbb{F}^*_{q^k}$ (cf. finite-field Diffie-Hellman problem) which is usually also referred to as MOV security. Both problems should be computationally infeasible for the overall system to be secure. The discrete logarithm problem on elliptic curves can be solved using the Pollard rho algorithm [55]. The discrete logarithm problem in finite fields can be solved using the index calculus attack. The running time of the Pollard rho algorithm is $O(\sqrt{r})$ with r being the size of the largest prime-order subgroup of $E(\mathbb{F}_q)$ [44]. The index calculus algorithm has a sub-exponential complexity in the field size. In order to achieve a balanced security level in both groups the extension field size q^k needs to be significantly larger than the r [18]. This ratio is expressed using the embedding degree, which often is equal to the degree k of the extension field the pairing maps to, and the parameter $\rho = \frac{log(q)}{log(r)}$. The parameter ρ measures the size of the prime-order subgroup on the curve r relative to the size of the base finite field q. Coppersmith's index calculus method solves the discrete logarithm problems in finite fields of small characteristic [14]. As it improves solving the discrete logarithm problem in fields of small characteristic the field sizes of small characteristic, fields need to be increased accordingly [18]. Due to this fact, fields of characteristic two and three require having larger field sizes than fields of larger prime characteristic to obtain the same security level. The security of a pairing is usually considered by the so-called MOV security. The MOV security of a pairing is equivalent to the bit length of the smallest finite field into which the pairing algorithm embeds to [22].

4.2 Properties

In order to be of practical value for cryptographic applications, a bilinear pairing should have the properties of bilinearity and non-degeneracy, and should provide a way for efficient (sub-polynomial) computation. In the following, we briefly discuss the former two properties.

Bilinearity The bilinear property of a pairing algorithm is of fundamental importance in order to build pairing-based cryptosystems. The pairing algorithm is a bilinear operation which maps two elements of input groups G_1 and G_2 to an element of an output group G_T, where G_1 and G_2 are additive groups and G_T is a multiplicative group. A map $e : G_1 \times G_2 \rightarrow G_T$ is called a bilinear map or *pairing*, if it satisfies the following condition:

$$e(P + Q, R) = e(P, R) \cdot e(Q, R) \quad \forall P, Q \in G_1; R \in G_2 \tag{4.4}$$

$$e(P, R + S) = e(P, R) \cdot e(P, S) \quad \forall P \in G_1; R, S \in G_2 \tag{4.5}$$

Usually, also the following notation is used in literature to define the bilinear property: Given that $\forall P, Q \in G_1$, $\forall a, b \in \mathbb{Z}_q^*$ bilinearity holds if

$$e(aP, bQ) = e(P, Q)^{ab}.$$

Non-degeneracy To be of practical use in the context of pairing cryptography, a bilinear map should not be degenerate when the inputs (P,Q) are linearly dependent. In general, a pairing is called degenerate if it maps to the identity.

Definition 4. *Let P,Q be points of order m. Then P and Q are linearly independent if there is no integer n so that $P = nQ$*

A pairing is called non-degenerate if the following condition holds:

$$e(P, Q) \neq 1 \quad \forall P \in G_1; Q, S \in G_2. \tag{4.6}$$

Computability To be of practical value, pairings need to be computable efficiently in sub-polynomial time. As bilinear pairings are generally complex and hard to calculate this is a usual requirement for pairing implementations.

Miller's Algorithm The basic algorithm to efficiently compute pairings was introduced in 1986 by Victor Miller in an unpublished manuscript [39]. Probably motivated by the intense research activity in the field of bilinear pairings, it was officially published in 2004 [41]. The so called *Miller algorithm* is an extension of the elliptic curve double-and-add operation which evaluates a pairing on an algebraic curve. Preceding algorithmic solutions to evaluate such functions were exponential in the size of the input where Miller's algorithm is linear in the size of the input [39]. Similar to the double-and-add algorithm, the Miller algorithm also requires to loop over a certain number of iterations. Attempts to improve a pairing algorithm generally focus on reducing the number of required Miller iterations. In its standard version, it applies a double-and-add iteration on the bits of the prime subgroup r using so-called *line evaluation* functions. In the case of supersingular curves, the Hamming weight of r can be chosen arbitrarily, which allows to use primes of low Hamming weight to improve the computation time. This algorithm is the most integral part in pairing calculations.

Pairings provide a map between elliptic-curve subgroups and finite field subgroups. In a cryptographic context, pairing algorithms provide a map between computational problem based on elliptic curves and computational problems based on finite fields. As the embedding degree sets the finite-field size, it directly influences the security provided by a finite field or the hardness of the associated problem. If the embedding degree is low, the finite field size is also low and the

ECDLP provides lower security as it is more feasible to solve it. Hence elliptic curves of low embedding degree are not suitable for cryptographic applications requiring a high security. However, as the Miller algorithm pursues operations on both groups a high embedding degree does not increase the security level for the whole system as the smaller group remains the limiting factor for the overall security level.

4.3 Weil Pairing

André Weil introduced the first bilinear pairing based on so-called Jacobians of curves in 1940. The Weil pairing, named after André Weil, introduced a pairing on points of finite order on elliptic curves which is bilinear, non-degenerate for asymmetric arguments, and efficiently computable. The fact that it is efficiently computable is due to Victor Miller who devised an algorithm for the evaluation of algebraic curves in [39, 41] Given an elliptic curve E/K and r being an integer prime to the characteristic of the field, we can write the Weil pairing as

$$e_r : E[r] \times E[r] \to \mu_r \subset \bar{K} \tag{4.7}$$

where the result μ_r is the group of rth roots of unity in $\bar{K}$ [53]. Where $\bar{K}$ denotes the algebraic closure of K, and $E[r]$ the group of all r-torsion points of E. Concerning cryptographic applications, the main drawback of the Weil pairing is that it is degenerate if applied to the cyclic subgroup of order ℓ [13]. So a symmetric Weil pairing $W_\ell(P, P)$, where the same cyclic subgroup of order ℓ is used in both groups, always results in 1. For more information on the Weil pairing and its efficient calculation, we want to refer to [41].

4.4 Tate Pairing

The Tate pairing goes back to John Tate who introduced so-called Tate-duality pairings. It was extended by Steven Lichtenbaum in 1969, as well as Gerhard Frey and Hans-Georg Rück in 1994 [19]. Hence the Tate pairing is also referred to as Tate-Lichtenbaum pairing. The Tate pairing describes the computation of a map e which maps elements of $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$ to an element of the multiplicative group $\mathbb{F}^*_{q^k}$ (Equation 4.8).

$$e : E(\mathbb{F}_q[r]) \times E(\mathbb{F}_{q^k})[r] \to \mu_r \subset \mathbb{F}^*_{q^k} \tag{4.8}$$

Computation of the Tate pairing is possible by application of Miller's algorithm [41] and the extensions due to [19]. The complexity of Miller's algorithm is

dominated by the number of iterations of its main loop. The number of iterations depends on the size of the underlying finite field. The body of the main loop consists of line evaluation functions in order to perform elliptic-curve operations. We can divide the loop body into two sections where one section is executed in every iteration (addition step) and the other section is executed if the ith bit of the binary representation of r is set (doubling step). The two steps contain the elliptic-curve point operation of point addition and point doubling respectively. While the Tate pairing can be applied in various cryptographic schemes it is computationally expensive. Consequently, numerous methods for faster pairing computation based on the Tate pairing have been proposed in literature.

4.5 η_T Pairing

In the following, we want to present a pairing algorithm originally based on the Tate pairing which allows to apply various algorithmic improvements. The so-called η pairing was introduced by Barreto et al. to compute pairings using super-singular curves [5]. Their approach generalizes the preceding results of Duursma-Lee in [16] for computing the Tate pairing in characteristic three in order to use their approach with elliptic and hyper-elliptic curves in characteristic two and three. They define the η to provide non-degeneracy and bilinearity and obtain significant improvements over the previous methods. They introduce the improved version as η_T pairing. The results of their η_T pairing computation outperforms the η pairing by a factor of two according to their experimental results [5]. In [8], Beuchat et al. present a square-root free version of the η_T pairing which is especially interesting for hardware implementations as no circuitry needs to be spent for square-root calculations. As such, this instance of the η_T pairing is particularly interesting regarding a low-area hardware implementation. Algorithmic descriptions of the η_T pairing are provided in finite fields of characteristic two and three. In the following, we consider the η_T pairing based on binary finite fields as the provide efficient implementations in hardware. For a more in-depth discussion on evaluations between characteristic two and three see [8]. For the binary field case, we denote the η_T pairing as:

$$\eta_T : E(\mathbb{F}_{2^m})[r] \times E(\mathbb{F}_{2^m})[r] \rightarrow \mu_r \subset \mathbb{F}^*_{2^{4m}}. \tag{4.9}$$

Using a second exponentiation, it is possible to compute the modified Tate pairing from the reduced η_T pairing [8]. The η_T pairing maps two elements of binary elliptic-curve points to an element of a multiplicative subgroup. The pairing calculation can be separated in two parts where the first part performs accumulative multiplication using line-evaluation functions and the second part assures that the final result is unique by an exponentiation step Figure 4.3. The η_T pairing can be

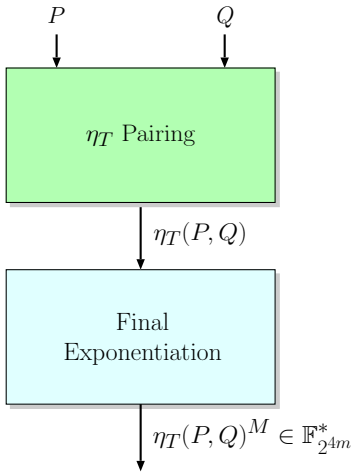


Figure 4.3: Abstract view on η_T pairing

implemented efficiently as it requires a considerably lower number of Miller loop iterations than other pairing algorithms. In the following, we want to present a more detailed illustration on the η_T pairing and the algorithms to compute it.

4.5.1 η_T Pairing Algorithm

In the following, we want to introduce the preliminaries for the η_T pairing according to [8]. Consider a supersingular curve E over $\mathbb{F}_{2^m}$ defined by

$$E : y^2 + y = x^3 + x + b \tag{4.10}$$

where $b \in \{0, 1\}$ and m is an odd integer. Depending on the extension m we define $\delta = b$ when $m = 1, 7 \bmod 8$ and $\delta = 1 - b$ else. The number of rational points of the curve E over $\mathbb{F}_{2^m}$ is given with $N = \#E(\mathbb{F}_{2^m}) = 2^m + 1 + \nu 2^{(m+1)/2}$ where $\nu = (-1)^\delta$ [6]. The embedding degree k is 4 as it is the least positive integer such that $2^{km} - 1$ divides N. According to [5], we choose $T = 2^m - N$ and a prime r (sometimes also denoted as ℓ) dividing N. The η_T pairing is then defined as the pairing of two points P and $Q \in E(\mathbb{F}_{2^m})[r]$ as

$$\eta_T(P, Q) = f_{T',P'}(\phi(Q)) \tag{4.11}$$

where $T' = -\nu T$ and $P' = [-\nu]P$ [8].

Non-linearity of the two input points is assured by applying a *distortion map* ϕ given as $\phi(x, y) = (x + s^2, y + sx + t)$ to satisfy the property of non-degeneracy [6]. The distortion map is used to map points of $E(\mathbb{F}_{2^m}[r])$ to $E(\mathbb{F}_{2^{4m}})[r]$ for all $(x, y) \in E(\mathbb{F}_{2^m})[r]$ [5]. The elements s and t of the distortion map are elements

of $\mathbb{F}_{2^{4m}}$ and satisfy the following equations $s^2 = s + 1$ and $t^2 = t + s$. With this definition we can represent $\mathbb{F}_{2^{4m}}$ as an extension field to $\mathbb{F}_{2^m}$ using the basis $(1, s, t, st)$ leading to Equation 4.12 [8]:

$$\mathbb{F}_{2^{4m}} = \mathbb{F}_{2^m}[s,t] \cong \mathbb{F}_{2^m}[X,Y]/(X^2 + X + 1, Y^2 + Y + X). \tag{4.12}$$

The function $f_{T',P'}$ in Equation 4.13 is an element of the function field of the curve denoted as $\mathbb{F}_{2^m}(E)$. Which, according to [8], is given by

$$\begin{aligned} f_{T',P'} : E(\mathbb{F}_{2^{4m}})[r] &\longrightarrow \mathbb{F}^*_{2^{4m}} \\ \phi(Q) &\longmapsto \left(\prod_{i=0}^{\frac{m-1}{2}} g_{[2^i]P'}(\phi(Q))^{2^{\frac{m-1}{2}-i}} \right) l_{P'}(\phi(Q)), \end{aligned} \tag{4.13}$$

where:

- $[2^i]P'$ represents a point doubling formula given by

$$[2^i]P' = \left(x_{P'}^{2^{2i}} + i, y_{P'}^{2^{2i}} + i x_{P'}^{2^{2i}} + i + \tau(i) \right).$$

- g_V is a rational function defined over $E(\mathbb{F}_{2^{4m}})[r]$ for all $V = (x_V, y_V) \in E(\mathbb{F}_{2^m})[r]$ which corresponds to the doubling of V. Where the doubling operation is defined with

$$g_V(x,y) = x(x_V^2 + 1) + y_V^2 + y + b.$$

- l_V is a rational function according to the addition of $[2^{\frac{m+1}{2}}]V$ with $[\nu]V$, defined for all $V = (x_V, y_V) \in E(\mathbb{F}_{2^m})[r]$. l_V is defined for all $(x,y) \in E(\mathbb{F}_{2^{4m}})[r]$ and given as following [8]:

$$\begin{aligned} l_V(x,y) = x_V^2 + (x_V + \alpha)(x + \alpha) + x + y_V + y + \delta + 1 + \\ (x_V + x + 1 - \alpha)s + t, \end{aligned} \tag{4.14}$$

  where $\alpha = 0$ if $m \equiv 3 \pmod 4$, or $\alpha = 1$ if $m \equiv 1 \pmod 4$.

For further details on the computation of the η_T pairing, we want to refer to the literature as in [5, 7, 8]. A derivation of the line evaluation functions used to calculate the pairing is given in [8].

This algorithmic description presented in the following uses optimizations of Shu et al. [52], who proposed a square-root free version of the η_T pairing, and also applies a reversed-loop approach suggested in [5]. The reversed-loop approach

improves the direct approach by eliminating one $\mathbb{F}_{2^m}$ multiplication [8]. As illustrated before, the algorithms apply tower field extensions which are indicated by uppercase letters. Elements of tower field extensions are either elements of $\mathbb{F}_{2^{2m}}$ or $\mathbb{F}_{2^{4m}}$. Consequently, we can denote the elements of the applied tower field by

$$G = g_0 + g_1 s \text{ where } G \in \mathbb{F}_{2^{2m}}, \tag{4.15}$$

$$G = g_0 + g_1 s + g_2 t + g_3 st \text{ where } G \in \mathbb{F}_{2^{4m}}. \tag{4.16}$$

Algorithm 2 η_T pairing (reversed-loop approach, without square roots [8]); lower case variables $\in \mathbb{F}_{2^m}$; $L, G, F \in \mathbb{F}_{2^{4m}}$

Input: $P, Q \in \mathbb{F}_{2^m}[\ell]$
Output: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

1:	$y_P \leftarrow y_P + \bar{\delta}$	($\bar{\delta}$ XOR)
2:	$x_P \leftarrow {x_P}^2$	(1 S)
3:	$y_P \leftarrow {y_P}^2$	(1 S)
4:	$y_P \leftarrow y_P + b$	(b XOR)
5:	$u \leftarrow x_P + 1$	(1 XOR)
6:	$g_1 \leftarrow u + x_Q$	(1 A)
7:	$g_0 \leftarrow x_P \cdot x_Q + y_P + y_Q + g_1$	(1 M, 3 A)
8:	$x_Q \leftarrow x_Q + 1$	(1 XOR)
9:	$g_2 \leftarrow x_P^2 + x_Q$	(1 S, 1 A)
10:	$G \leftarrow g_0 + g_1 s + t$	
11:	$L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$	(1 A, 1 XOR)
12:	$F \leftarrow L \cdot G$	(2 M, 1 S, 5 A, 2 XOR)
13:	**for** $i \leftarrow 1$ to $\frac{m-1}{2}$ **do**	
14:	$F \leftarrow F^2$	(4 S, 4 A)
15:	$x_Q \leftarrow x_Q^4$	(2 S)
16:	$y_Q \leftarrow y_Q^4$	(2 S)
17:	$x_Q \leftarrow x_Q + 1$	(1 XOR)
18:	$y_Q \leftarrow y_Q + x_Q$	(1 A)
19:	$g_0 \leftarrow u \cdot x_Q + y_P + y_Q$	(1 M, 2 A)
20:	$g_1 \leftarrow x_P + x_Q$	(1 A)
21:	$G \leftarrow g_0 + g_1 s + t$	
22:	$F \leftarrow F \cdot G$	(6 M, 14 A)
23:	**end for**	

The generic cost formulæ for the η_T algorithm variants with and without square roots are provided by Beuchat et al. [8] and are given in Table 4.1. The η_T pairing algorithm without square roots introduces additional addition operations but does

not change the computational cost in terms of costly multiplications. The significant advantage of the square-root free variant is that it allows to save chip area. As the addition in binary finite fields can be implemented cheaply, and low-area is a main design goal, we favor the square-root free variant to be used for a low-area hardware implementation.

From the algorithmic description given in Algorithm 2, we obtain an upper bound for memory consumption by a simple summation of memory costs to store the temporary values without consideration of memory reuse. We find seven variables in $\mathbb{F}_{2^m}$ and three variables in $\mathbb{F}_{2^{4m}}$. Consequently, we find a worst-case memory consumption of 19 words of size m when not considering optimizations of memory usage .

Table 4.1: Comparison of computational cost for η_T pairing algorithms with and without square roots

	η_T pairing without square roots	η_T pairing with square roots	Final Exponentiation
Additions	$11m$	$10 + 17 \cdot \frac{m-1}{2}$	$2m + 53$
XORs	$5 + \bar{\delta} + b + \frac{m-1}{2}$	$3 + \bar{\delta} + \beta + \frac{m+1}{2}$	-
Multiplications	$3 + 7 \cdot \frac{m-1}{2}$	$3 + 7 \cdot \frac{m-1}{2}$	26
Squarings	$4m$	$4m$	$2m + 9$
Square roots	-	$m - 1$	-

4.5.2 Final Exponentiation

To use the η_T pairing in cryptographic applications, we require the result to be unique. However, the result of the η_T pairing is not uniquely defined and thus needs to be transformed. The transformation to a unique result, is called *final exponentiation*. To obtain a unique result the η_T pairing is raised to the Mth power where M is defined by

$$M = \frac{2^{4m} - 1}{N} = (2^{2m} - 1)(2^m + 1 - \nu 2^{\frac{m+1}{2}}), \tag{4.17}$$

where N is number of rational points of the curve E over $\mathbb{F}_{2^m}$. To calculate the final exponentiation, two algorithms for cases of $\nu = 1$ and $\nu = -1$ are proposed in [8].

- Ronan et al. [48] propose an algorithm involving a single inversion of element in $\mathbb{F}_{2^{4m}}$ for the case that $\nu = 1$.

- Shu et al. [52] apply a two-step algorithm for the exponentiation in case $\nu = -1$. They propose raising to $2^{2m} - 1$ first, which involves an inversion in $\mathbb{F}_{2^{2m}}$, and secondly raising to $(2^m + 1 + 2^{\frac{m+1}{2}})$.

The algorithm of Shu et al. contains an inversion in the quadratic extension field where the version given by Ronan et al. requires an inversion of a quadratic extension field element. As field inversion is generally a very costly operation, we apply Shu et al.'s approach and avoid inversion in the quartic extension field.

Algorithm 3 Final exponentiation of the η_T pairing; $m_i \in \mathbb{F}_{2^m}$; T_i, V_i, W_i, and $D \in \mathbb{F}_{2^{2m}}$; V and $W \in \mathbb{F}_{2^{4m}}$; [8]

Input: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}^*_{2^{4m}}$
Output: $V = U^M \in \mathbb{F}^*_{2^{4m}}$, with $M = (2^{2m} + 1)(2^m - \nu 2^{\frac{m+1}{2}} + 1)$

1:	$m_0 \leftarrow u_0^2; m_1 \leftarrow u_1^2; m_2 \leftarrow u_2^2; m_3 \leftarrow u_3^2$	(4 S)
2:	$T_0 \leftarrow (m_0 + m_1) + m_1 s$	(1 A)
3:	$T_1 \leftarrow (m_2 + m_3) + m_3 s$	(1 A)
4:	$T_2 \leftarrow m_3 + m_2 s$	
5:	$T_3 \leftarrow (u_0 + u_1 s) \cdot (u_2 + u_3 s)$	(3 M, 4 A)
6:	$T_4 \leftarrow T_0 + T_2$	(2 A)
7:	$D \leftarrow T_3 + T_4$	(2 A)
8:	$D \leftarrow D^{-1}$	(1 I, 3 M, 1 S, 2 A)
9:	$T_5 \leftarrow T_1 \cdot D$	(3 M, 4 A)
10:	$T_6 \leftarrow T_4 \cdot D$	(3 M, 4 A)
11:	$V_0 \leftarrow T_5 + T_6$	(2 A)
12:	$V_1, W_1 \leftarrow T_5$	
13:	$W_0 \leftarrow T_6$	
14:	$V \leftarrow V_0 + V_1 t$	
15:	$W \leftarrow W_0 + W_1 t$	
16:	$V \leftarrow V^{2m+1}$	(5 M, 2 S, 9 A)
17:	**for** $i \leftarrow 1$ to $\frac{m+1}{2}$ **do**	(611 iterations)
18:	$W \leftarrow W^2$	(4 S, 4 A)
19:	**end for**	
	Return $V \cdot W$	(9 M, 20 A)

Algorithm 3 illustrates the computations used to raise the η_T pairing to a unique result. The algorithm contains several operations in field extensions. Algorithms for arithmetic in the respective extension fields are presented in Section 4.5.3 and 4.5.4. From the algorithmic description given in Algorithm 3, we can again give an

estimate on the upper bound memory consumption. The algorithm uses four variables in $\mathbb{F}_{2^m}$, twelve variables in $\mathbb{F}_{2^{2m}}$, and two variables in $\mathbb{F}_{2^{4m}}$. Consequently, we find a worst-case memory consumption of 36 words of size m without consideration of memory optimizations.

4.5.3 Arithmetic over $\mathbb{F}_{2^{2m}}$

The variables used in Algorithm 2 are elements in $\mathbb{F}_{2^m}$ or field extensions thereof. Base field operations in $\mathbb{F}_{2^m}$ can be calculated using ordinary binary field arithmetic. The operations in extension fields are provided by algorithms according to the field extension degree. In the following, we want to elaborate on the required algorithms for quadratic field extensions $\mathbb{F}_{2^{2m}}$ which are based on operations in the base extension field $\mathbb{F}_{2^m}$.

Multiplication To compute Algorithm 3, we require multiplication of elements in $\mathbb{F}_{2^{2m}}$. Elements of the quadratic extension field are given as $a_0 + a_1 s$ and $b_0 + b_1 s$ (see Equation 4.15) where s signifies the most-significant part. This allows efficient implementation using the Karatsuba algorithm as illustrated in Algorithm 4 [31, 8]. Consequently, we can execute this $\mathbb{F}_{2^{2m}}$ multiplication with modular multiplications and additions in the base extension field $\mathbb{F}_{2^m}$ at a cost of three multiplications and four additions.

Algorithm 4 Multiplication in $\mathbb{F}_{2^{2m}}$ (Karatsuba algorithm)

Input: $A = a_0 + a_1 s, B = b_0 + b_1 s \in \mathbb{F}_{2^{2m}}$
Output: $C = A \cdot B \in \mathbb{F}_{2^{2m}}$.

1: $c_0 \leftarrow a_0 \cdot b_0$	(1 M)
2: $c_1 \leftarrow a_1 \cdot b_1$	(1 M)
3: $a_{01} \leftarrow a_0 + a_1$	(1 A)
4: $b_{01} \leftarrow b_0 + b_1$	(1 A)
5: $a_{01} \leftarrow a_{01} \cdot b_{01}$	(1 M)
6: $a_{01} \leftarrow a_{01} + c_0$	(1 A)
7: $c_0 \leftarrow c_0 + c_1$	(1 A)
Return $C = c_0 + a_{01} s$	
	$\Sigma = (3M, 4A)$

We apply this type of multiplication at line 5, 9, and 10 of the final exponentiation algorithm.

Inversion In final exponentiation, we need to build the denominator expression $U_0^2 + U_0 U_1 + U_1^2 s$ to calculate the terms $U^{2^{2m}-1}$ and $U^{1-2^{2m}}$ in order to invert the

variable $D \in \mathbb{F}_{2^{2m}}$ (see Line 8 of Algorithm 3). We calculate this using Algorithm 5 with three multiplications, one inversion, two additions, and one squaring in the base field $\mathbb{F}_{2^m}$.

Algorithm 5 Inversion of quadratic extension field element $(u_0 + u_1 s)^{-1}$ [8]

Input: $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}, U \neq 0$
Output: $V = U^{-1} = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$.

1: $a_0 \leftarrow u_0 + u_1$ (1 A)
2: $m_0 \leftarrow u_0^2$ (1 S)
3: $m_1 \leftarrow a_0 \cdot u_1$ (1 M)
4: $a_1 \leftarrow m_0 + m_1$ (1 A)
5: $i_0 \leftarrow a_1^{-1}$ (1 I)
6: $v_0 \leftarrow a_0 \cdot i_0$ (1 M)
7: $v_1 \leftarrow u_1 \cdot i_0$ (1 M)
Return $v_0 + v_1 s$
$\Sigma = (3M, 2A, 1S, 1I)$

4.5.4 Arithmetic over $\mathbb{F}_{2^{4m}}$

The multiplication of L and G in $\mathbb{F}_{2^{4m}}$ (Algorithm 2 Line 12) can basically be calculated using the multiplication algorithm as given by Algorithm 7. Beuchat et al. show how to significantly simplify this multiplication by exploiting the sparsity of both operands. To do so, they transform the term $(g_0+g_1s+t)\cdot((g_0+g_2)+(g_1+1)s+t)$ using the formulæ in Equation 4.18 to another form which contains duplicate sub-terms for squaring and addition. As a result, the redundant operations are calculated only once which lowers computational cost:

$$\begin{aligned}(g_0 + g_1 s + t) \cdot ((g_0 + g_2) + (g_1 + 1)s + t) \\ = g_0 \cdot (g_0 + g_2) + g_1^2 + g_1 + (g_0 + g_1 \cdot g_2 + g_1^2 + g_1 + 1)s \\ + (g_2 + 1)t + st.\end{aligned} \tag{4.18}$$

The term $g_1^2 + g_1$ in Equation 4.18 is reused giving an overall cost for this operation of two XOR operations, two multiplications, and just five additions and one squaring. The improved multiplication is given in Algorithm 6.

There are two ways to compute the term V^{2^m+1} in Algorithm 3 Line 16. The first is to multiply V^{2^m} with itself to obtain V^{2^m+1}. This requires a full multiplication over $\mathbb{F}_{2^{4m}}$ and can be calculated with Algorithm 7. Beuchat et al. propose a faster way to raise V to the power of 2^m+1 requiring only five multiplications, two squarings, and nine additions in the base extension field. The improved algorithm is given in Algorithm 8.

Algorithm 6 Simplified computation of $(g_0 + g_1 s + t) \cdot ((g_0 + g_2) + (g_1 + 1)s + t)$ in $\mathbb{F}_{2^{4m}}$ [8]

Input: $U = g_0 + g_1 s + t \in \mathbb{F}_{2^{4m}}, V = (g_0 + g_2) + (g_1 + 1)s + t \in \mathbb{F}_{2^{4m}}$
Output: $W = U \cdot V \in \mathbb{F}_{2^{4m}}$.

1: $s_0 \leftarrow g_1^2$	(1 S)
2: $a_0 \leftarrow g_0 + g_2$;	(1 A)
3: $a_1 \leftarrow g_1 + s_0$;	(1 A)
4: $m_0 \leftarrow g_0 \cdot a_0; m_1 \leftarrow g_1 \cdot g_2$	(2 M)
5: $w_0 \leftarrow m_0 + a_1$	(1 A)
6: $w_1 \leftarrow m_1 + g_0 + a_1 + 1$	(2 A, 1 XOR)
7: $w_2 \leftarrow g_2 + 1$	(1 XOR)
8: $w_3 \leftarrow 1$	
Return $w_0 + w_1 s + w_2 t + w_3 st$	
	$\Sigma = (2M, 5A, 1S, 2XOR)$

The accumulative multiplication in Algorithm 2 Line 22 is calculated according to Algorithm 9.

4.5.5 Computational Complexity

In the following, we consider the computational complexity for a computation of the η_T pairing at a security level of 128 bits using the algorithms presented in Algorithm 2 and Algorithm 3. The most significant factor is the parametrization of the supersingular curve E and the finite field $\mathbb{F}_{2^m}$ over which the curve is defined.

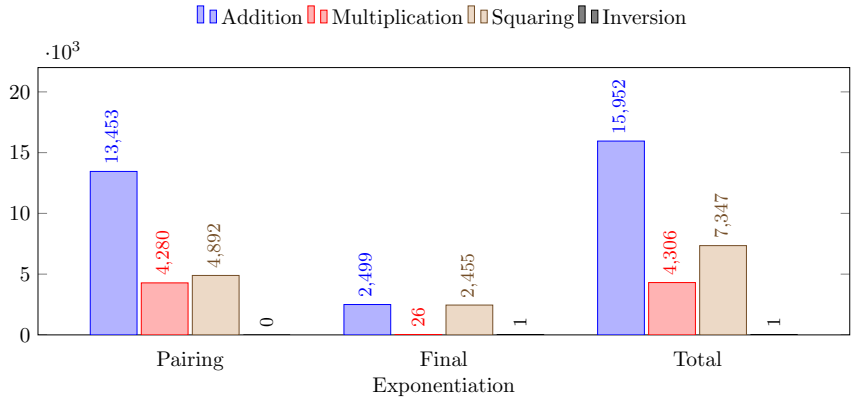


Figure 4.4: Finite-field operations in $\mathbb{F}_{2^m}$ of η_T pairing over $\mathbb{F}_{2^{1223}}$

Algorithm 7 Multiplication in $\mathbb{F}_{2^{4m}}$ [8]

Input: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$ and $V = v_0 + v_1 s + v_2 t + v_3 st \in \mathbb{F}_{2^{4m}}$
Output: $W = U \cdot V \in \mathbb{F}_{2^{4m}}$.

1: $a_0 \leftarrow u_0 + u_1$ (1 A)
2: $a_1 \leftarrow v_0 + v_1$ (1 A)
3: $a_2 \leftarrow u_0 + u_2$ (1 A)
4: $a_3 \leftarrow v_0 + v_2$ (1 A)
5: $a_4 \leftarrow u_1 + u_3$ (1 A)
6: $a_5 \leftarrow v_1 + v_3$ (1 A)
7: $a_6 \leftarrow u_2 + u_3$ (1 A)
8: $a_7 \leftarrow v_2 + v_3$ (1 A)
9: $a_8 \leftarrow a_0 + a_6$ (1 A)
10: $a_9 \leftarrow a_1 + a_7$ (1 A)
11: $m_0 \leftarrow u_0 \cdot v_0; m_1 \leftarrow u_1 \cdot v_1; m_2 \leftarrow u_2 \cdot v_2; m_3 \leftarrow u_3 \cdot v_3;$ (4 M)
12: $m_4 \leftarrow a_0 \cdot a_1; m_5 \leftarrow a_2 \cdot a_3; m_6 \leftarrow a_4 \cdot a_5; m_7 \leftarrow a_6 \cdot a_7; m_8 \leftarrow a_8 \cdot a_9$ (5 M)
13: $a_{10} \leftarrow m_0 + m_1$ (1 A)
14: $a_{11} \leftarrow m_0 + m_4$ (1 A)
15: $w_0 \leftarrow a_{10} + m_2 + m_7$ (2 A)
16: $w_1 \leftarrow a_{11} + m_3 + m_7$ (2 A)
17: $w_2 \leftarrow a_{10} + m_5 + m_6$ (2 A)
18: $w_3 \leftarrow a_{11} + m_5 + m_8$ (2 A)

Return $W = w_0 + w_1 s + w_2 t + w_3 st$

$\Sigma = (9M, 20A)$

Algorithm 8 Raising U to the power of $2^m + 1$ over $\mathbb{F}_{2^{4m}}$ [8]

Input: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$
Output: $V = U^{2^m+1} \in \mathbb{F}_{2^{4m}}$.

1: $a_0 \leftarrow u_0 + u_1$ (1 A)
2: $a_1 \leftarrow u_2 + u_3$ (1 A)
3: $m_0 \leftarrow a_0 \cdot a_1; m_1 \leftarrow u_0 \cdot u_1; m_2 \leftarrow u_0 \cdot u_3$ (3 M)
4: $m_3 \leftarrow u_1 \cdot u_2; m_4 \leftarrow u_2 \cdot u_3$ (2 M)
5: $s_0 \leftarrow a_0^2; s_1 \leftarrow a_1^2$ (2 S)
6: $v_3 \leftarrow m_4 + s_1$ (1 A)
7: $v_2 \leftarrow m_2 + m_3$ (1 A)
8: $v_1 \leftarrow v_3 + m_0 + m_3$ (2 A)
9: $v_0 \leftarrow m_0 + m_1 + m_2 + s_0$ (3 A)

Return $v_0 + v_1 s + v_2 t + v_3 st$

$\Sigma = (5M, 9A, 2S)$

Algorithm 9 Computation of $(g_0 + g_1 s + t) \cdot (f_0 + f_1 s + f_2 s + f_3 st)$ in $\mathbb{F}_{2^{4m}}$ [8]

Input: $G = g_0 + g_1 s + t \in \mathbb{F}_{2^{4m}}, F = f_0 + f_1 s + f_2 t + f_3 st \in \mathbb{F}_{2^{4m}}$
Output: $W = F \cdot G \in \mathbb{F}_{2^{4m}}$.

1: $a_0 \leftarrow g_0 + g_1$ (1 A)
2: $a_1 \leftarrow f_0 + f_1$ (1 A)
3: $a_2 \leftarrow f_2 + f_3;$ (1 A)
4: $m_0 \leftarrow g_0 \cdot f_0; m_1 \leftarrow g_1 \cdot f_1; m_2 \leftarrow g_0 \cdot f_2; m_3 \leftarrow g_1 \cdot f_3$ (4 M)
5: $m_4 \leftarrow a_0 \cdot a_1; m_5 \leftarrow a_0 \cdot a_2$ (2 M)
6: $w_0 \leftarrow m_0 + m_1 + f_3$ (2 A)
7: $w_1 \leftarrow m_0 + m_4 + f_2 + f_3$ (3 A)
8: $w_2 \leftarrow m_2 + m_3 + f_0 + f_2$ (3 A)
9: $w_3 \leftarrow m_2 + m_5 + f_1 + f_3$ (3 A)

Return $w_0 + w_1 s + w_2 t + w_3 st$

$\Sigma = (6M, 14A)$

The choice of an elliptic curve and the underlying finite field on which the curve is defined on has direct influence on the characteristics of the pairing system. Curve and field parameters affect the resulting security level of the pairing and its computational complexity. The presented pairing algorithm is based on supersingular curves defined over fields of characteristic two. As one design goal is to implement a pairing at a 128-bit security level, an underlying binary field $\mathbb{F}_{2^{1223}}$ is used [21]. The applied field is constructed with the irreducible polynomial

$$f(x) = x^{1223} + x^{255} + 1. \tag{4.19}$$

In the following, we want to discuss the parametrization of the η_T pairing according to this field definition. We consider the supersingular elliptic curve

$$E : y^2 + y = x^3 + x, \tag{4.20}$$

where the curve parameter b is zero. Using the finite field $\mathbb{F}_{2^{1223}}$, we obtain $m = 1223$, and hence have $m \equiv 7 \pmod 8$ giving $\delta = b = 0$ [8]. The constant $\bar{\delta}$ is obtained as $\bar{\delta} = 1 - \delta = 1$. The number of rational points of E over $\mathbb{F}_{2^m}$ given by

$$N = \#E(\mathbb{F}_{2^m}) = 2^m + 1 + \nu\, 2^{(m+1)/2}, \tag{4.21}$$

where $\nu = (-1)^\delta = 1$ [6]. With the given curve parameters we have $N = 2^{1223} + 1 + 2^{612}$. The embedding degree k for this curve is 4 as it is a supersingular curve over a binary finite field, and k is the least positive integer such that N divides $2^{km} - 1 = 2^{4892} - 1$. With the definition of the line evaluation function for the addition of $[2^{\frac{m+1}{2}}]V$ with $[\nu]V$ in Equation 4.14, we obtain $\alpha = 0$ as $m = 1223 \equiv 3 \pmod 4$.

Table 4.2: Field-operation cost of η_T pairing in $\mathbb{F}_{2^{1223}}$ considering finite field inversion based on Fermat's little theorem

Operation	η_T without square roots	Final Exponentiation	Total
Additions	13453	2499	15952
XORs	617	-	617
Multiplications	4280	41	4321
Squarings	4892	3677	8579

The computational analysis of the η_T algorithm as given in Table 4.1 and Figure 4.4 served as a guideline for design decisions in the hardware implementation

process. The XOR operation listed in Table 4.2 represents addition of a single digit and is implemented with no extra computational cost in terms of computation cycles as it appended automatically on a preceeding computation. The decomposition of the pairing algorithm into operations in the base extension field together with evaluations of finite-field operation cost provided an important basis for decisions on potential hardware implementations. As the addition, XOR, and squaring operation is relatively cheap special focus was given to the field multiplication which was used to estimate the expected computation time of design candidates. The decisions were made based on the design goal of a small and yet reasonably fast hardware circuit which is able to compute the η_T pairing in less than 400 ms.

Chapter 5

Hardware Architectures

Throughout this chapter, we present the implementation of seven application specific integrated circuits to compute the η_T pairing. The chapter starts with a brief introduction to ASICs to motivate the presented implementations. In the following sections, we will present the so-called front-end design for hardware architectures designed to calculate a bilinear η_T pairing at a security level of 128 bits. Computing a η_T pairing at a high security level requires to perform operations with in a large finite field. As the multiplier performance dominates the overall system performance, special effort was made to optimize the implementation of finite field multiplication. Hence, Section 5.3 presents several multiplier architectures based on Karatsuba's algorithm and digit-serial multiplication. The presented hardware architectures were tested and verified using a testbench setup which is presented in Section 5.4. Finally the chapter concludes with a discussion the applied memory configurations used throughout the designs. In general the presented implementations target application in resource-constrained environments such as smart cards, embedded devices, or even RFID tags. All of which demand low-area and low-power consumption while providing a reasonably short compuation time to be of practicable value in interactive applications.

5.1 Introduction

Since the advent of semiconductor technology in the first half of the twentieth century and integrated circuits in the late 1950s[1], the number of hardware applications has increased continuously. Digital hardware technology enabled the construction and widespread deployment of computers and embedded devices introducing our society to the so-called *digital age*. In the following decades, digital-hardware tech-

[1]Early integrated circuits were presented by Jack Kilby and Robert Noyce in 1958/59.

nology has been subject to rapid technology changes as the demand for new and improved hardware solutions has increased steadily.

The vast majority of hardware systems today is based on ASICs which are virtually ubiquitous in today's electronic devices. ASIC designs offer high performance, low power consumption, and allow low-area designs tailored for a specific application. Offering a high degree of freedom demands to execute a rather complex design cycle to build a custom hardware circuit. The design cycle basically consists of the so-called *front-end* and *back-end* design. Front-end design begins with the definition and specification of a design task and ends with a description of a hardware circuit, usually by means of a Hardware Description Language (HDL). With a so-called synthesizer, the hardware description provided in an HDL can be transformed to a technology-dependent circuit description, usually called a *netlist*, which describes instances of gates and their interconnections. In a standard-cell based design, the transistor-level descriptions are abstracted to standard cells representing the building blocks of the netlist. Back-end design basically consists of the tasks required to transform a netlist to a layout description ready for fabrication by a semiconductor foundry. Another proponent of digital technology is a reconfigurable-hardware device such as a Field Programmable Gate Array (FPGA). FPGAs usually contain a pre-fabricated grid of configurable logic blocks, connected with configurable switches. By configuring the contained logic blocks and their interconnection, a functional circuit can be configured. The configuration of an FPGA is usually called a *bitstream* which is synthesized from a circuit description in HDL. Compared to ASIC designs, FPGA devices offer a short design cycle and provide fast and accurate functionality. This is because there is no need for a back-end design phase as the physical circuit is already pre-built and not subject to errors induced in back-end design or subsequent fabrication. The main drawback of FPGA technology compared to ASIC technology is their considerably bigger size and power consumption [36]. This is why ASIC designs are generally better suited for applications in embedded devices where properties of low-area and low-power consumption are crucial. The gap between ASIC and FPGA designs is also due to the ratio of recurring to non-recurring costs regarding product deployment. Compared to FPGAs, an ASIC deployment contains a high amount of non-recurring design costs for implementation, verification, layout, and initial fabrication runs for physical testing and measurement. On the other hand the recurring costs in ASIC deployment are considerably lower in terms of cost per chip given that a large number of chips is to be produced. So each deployment scenario has a break-even point where the benefits from low recurring cost of an ASIC strategy compensate the large amount of non-recurring ASIC-design costs. So in general ASIC designs offer economic advantages if a hardware circuit is to be produced and deployed in large quantities. As this work targets application in

resource-constrained low-area low-power environments with large scale deployment the effort in an ASIC design is well-founded due to the above reasons.

5.2 System Architecture

Performing a bilinear pairing with the η_T algorithm over the elliptic curve $E(\mathbb{F}_{2^{1223}})$ mandates doing underlying binary finite-field operations. Due to the embedding degree of the pairing finite-field calculations are performed in extension fields of various extension degree. Beside the basic operations in the base-extension field $\mathbb{F}_{2^m}$ also operations in a quadratic extension field $\mathbb{F}_{2^{2m}}$ and in a quartic extension field $\mathbb{F}_{2^{4m}}$ are needed. The operations in the quadratic and quartic extension field can be expressed using operations in the base-extension field. Thus realizing a hardware architecture providing base-extension field operations is adequate to calculate the η_T pairing.

The proposed system architecture to calculate the η_T pairing contains a control unit which controls the execution of the finite-field operations, an arithmetic unit which is capable of calculating base-extension finite-field operations, a random-access memory to store the temporary values for the pairing computation, and an instruction memory holding the instruction words for the instruction set supported by the control unit. The base-extension field at the targeted security level is $\mathbb{F}_{2^{1223}}$ where elements are represented by 1223 bits. An operand size of this dimension directly impacts the area complexity of the arithmetic unit. The presented control setup is designed to be flexible in regard to later redesigns of the arithmetic unit or algorithms to calculate the pairing. Possible design changes could include replacing parts of the arithmetic unit to reduce area consumption or extending the instruction set to support additional instructions. Flexibility is attained using a microcoded approach instead of classic finite-state based implementation. From a top-level perspective, a microcoded approach also satisfies the rather irregular instruction sequence inherent in the η_T algorithms. Furthermore, the pairing algorithm contains a large number of computational steps which are hard to maintain if implemented as classic finite-state machine description. Another significant advantage of a microcoded approach is that it supports making later changes at an algorithmic level. Using an assembler-like metacode to generate the microcode, simplifies resequencing or exchanging computational blocks in algorithms or sub-algorithms. Last but not least, a microcode supports the designer in terms of design verification as the instruction sequence and memory allocation pattern is represented in a structured and concentrated way. Another design decision is implementing handshaked signaling between the control unit and the arithmetic unit. While introducing a small control latency, this makes the design more modular. As such, changing the implementation of the arithmetic unit

does not require adapting the control if instruction latency changes. A simplified system overview covering the main building block of the top-level architecture is depicted in Figure 5.1.

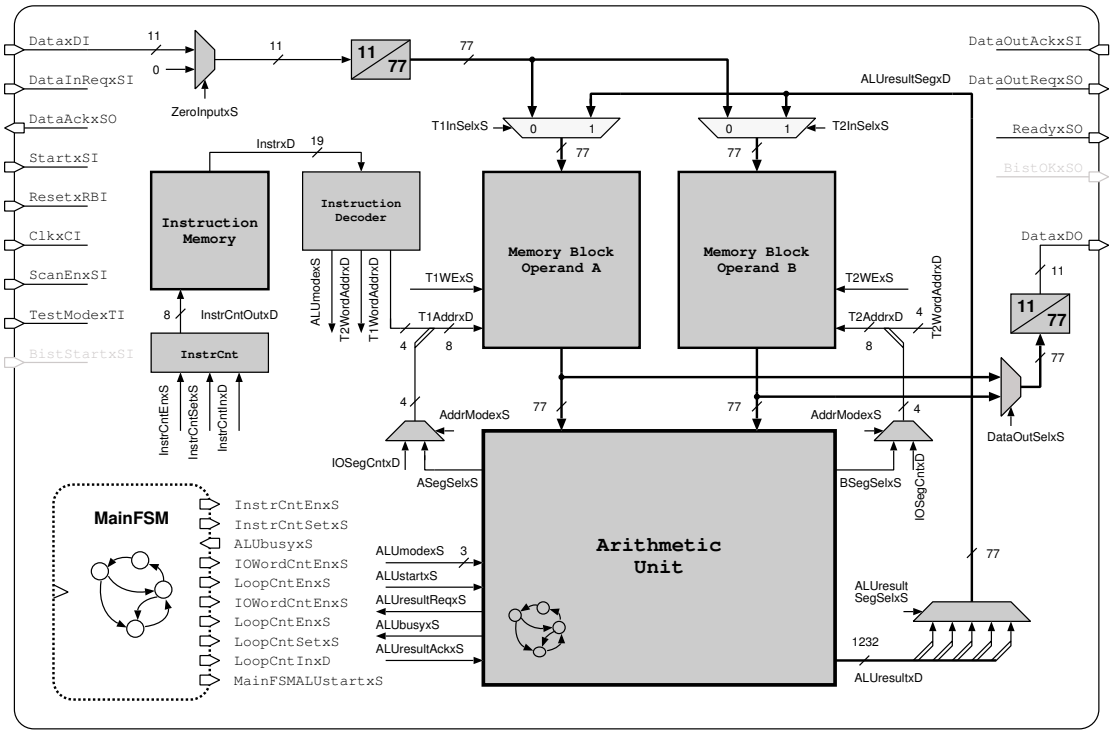


Figure 5.1: Top-level system architecture

Arithmetic on large operands naturally requires an arithmetic unit which is capable of processing large operands. Some arithmetic operations make an efficient implementation difficult as their area scales quadratically with operand size. This is why multiplier circuits usually dominate the area requirement of an arithmetic unit. As this work targets on a low-area implementation, we need to cope especially with area requirements of arithmetic circuits processing large operands. A classic approach to cope with large-scale problems is *divide and conquer*. One of the early and famous proponent of this strategy is a multiplication algorithm by A. Karatsuba [31] which splits the multiplication in smaller-sized multiplications effectively reducing the computational cost. By computing large multiplications in smaller steps it is possible to reduce the size of the arithmetic unit. The so-called Karatsuba algorithm and its adaption for the given design problem is presented in Section 5.3.2. Splitting the operands into smaller segments also affects the memory organization for storing temporary operand values. While segmentation allows having area-efficient aspect ratios for SRAM macrocells it also affects the number

of cycles attributed to read and write operations. Basically two memory configurations are used for the designs presented in this work. The 77-bit architectures split operands into 16 segments which introduce a latency of 16 cycles for memory access of reading or writing an operand. Architectures based on a 153-bit interface benefit from the lower number of operand segments effectively halving the number of read/write cycles.

Operand memory is realized by two separate single-port SRAM macrocell blocks. The memory-output ports are directly connected with the arithmetic unit. This omits a multiplexer cross-over structure which would allow to access segments originating from the same memory block to execute an instruction. Such a cross-over structure would also allow to read both memory blocks at each data input of the arithmetic unit. The given implementation does not spend chip area on such a multiplexer structure at the cost of a more complicated instruction scheduling and allocation pattern. This restriction implies that instructions with two operands require the operands to reside in disjunct memory blocks. Using operand memory-input multiplexers, a result can be stored to each of the two memory block where the respective destination is set by the destination address provided in the instruction word.

Another area optimization was found by constraining the algorithmic flow so that single-operand operations are only supported by one memory block. To save a multiplexer structure, only one port of the arithmetic unit supports the squaring operation. Clearly this limits the freedom for memory allocation and instruction scheduling as squarings can only be performed by reading operands from one of the two memory blocks. In the given implementation, the input port A of the arithmetic unit provides the squaring functionality. The task finding an optimal way to arrange the instructions and place the respective results for the given architecture is not trivial. Nevertheless, the instruction sequence was optimized towards this ideal condition. However, at some points it was not possible to find an instruction sequence and memory allocation pattern which meets the given constraints. To solve such problems, one could apply compiler theory and formulate the architectural constraints. This would allow to apply standard compiler techniques to optimize the algorithmic description for the given architecture. While this certainly would be an interesting encounter, it probably would take a significant amount of time to build a compiler for a custom architecture. So in order to solve situations where an operand does not reside in the required memory block, we apply a simple copy operation of the desired operand. Copying can be realized reusing the binary-field addition instruction where the other operand is containing just zeros. This approach requires a clear instruction to ensure that a zero-valued segment is available in the other operand memory block if needed. To clear a memory segment, we use small multiplexers at the top-level data inputs before

they get extended to the data widths of the respective operand memory. This provides a way to set the macrocell inputs to zero and clear memory content.

The overall area requirement is dominated by the multiplier architecture and memory for temporary variables. The actual memory required for storing temporary variables depends on the instruction scheduling of the algorithms to calculate the pairing. A first upper bound analysis for the amount of temporary memory required to compute the full pairing showed that a pairing computation is feasible using 26 (31 798 bits) words where each has an operand width of 1223 bits. Instruction reordering and in-place execution in sub-algorithms provides an optimized memory-consumption value of 18 words (22 014 bits). This includes memory requirements for storing the point coordinates of the input points P and Q as well as extra memory for calculations in higher extension fields required in sub-algorithms.

5.2.1 Pairing Algorithm

The η_T pairing consists of two main computational steps where one step computes the pairing F and another step to raise the intermediate pairing value to the power of M giving F^M which ensures that the result is a unique value which can be used in cryptographic applications. The second step is called final exponentiation. In the following, we want to describe the first step.

The pairing algorithm given in Algorithm 2 consists of two major computational blocks where the first is basically an initialization step and the second is the accumulative multiplication representing the so-called Miller iterations. The second block is repeated 611 times and reuses the quartic extension-field element denoted as F in each iteration. The initialization step transforms the elliptic curve points P and Q to the elements $L, G \in \mathbb{F}_{2^{4m}}$ which are then iteratively transformed to the multiplicative element $F \in \mathbb{F}_{2^{4m}}$. Each of the contained sub-algorithms is analyzed regarding its memory consumption to find the amount of memory required to implement the pairing in hardware. To analyze the data flow, each algorithm was transformed to a signal flow graph which allows to find sequential dependencies more easily. If a memory segment is read for the last time in the signal-flow graph, its memory is available to be used by new segments. As a result, restructuring the signal-flow graphs allows to reduce the memory footprint. Algorithmic representation as signal-flow graph also provided a good way to find a mapping of variables to the operand memory-blocks (memory allocation) which reduces the number of copy operations and allows to have variables in-place for loop constructions. After restructuring the algorithms, we obtain a peak memory consumption of 15 words located in Algorithm 9. The corresponding data flow used to implement the η_T pairing is illustrated in Figure 5.2. The memory consumption of the main blocks is indicated by the weighted arrows on the right in multiples of the word length 1223 bits.

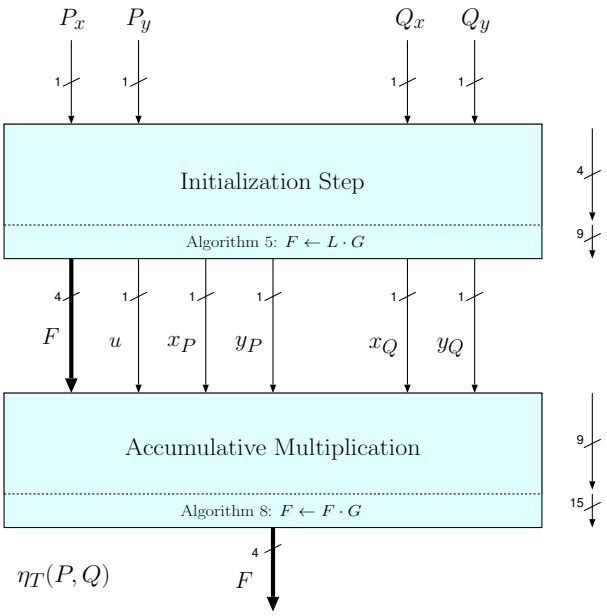


Figure 5.2: Data-flow graph for η_T pairing with memory-allocation sizes

The main loop in Algorithm 2 contains a squaring operation of F. To square F, we need to place all the base-field elements of F in the memory block which supports squaring. However, the operation $F = F \cdot G$ requires the base-field elements f_0, f_1 and f_2, f_3 to be located in distinct memory blocks for the addition operation in Algorithm 9. So if squaring is only supported in one memory block, there has to occur a copy operation at least four times per loop iteration. With 611 iterations and four copy cycles, this gives 2444 copy operations for the implementation when no such multiplexer is applied.

5.2.2 Final Exponentiation

To obtain a unique pairing result a so-called *final exponentiation* is performed. The implementation of Algorithm 3 is structured in various sub-algorithms which are used to calculate operations in quadratic and quartic extension fields. The sub-algorithm given in Algorithm 7 is used to compute a product of two quartic extension-field elements denoted as V and W in Algorithm 3. The implementation of this quartic extension-field multiplication uses 18 words to compute the result. Other sub-algorithms in the final exponentiation are implemented with less operand-memory utilization. As Algorithm 7 represents the maximum memory requirement of the pairing and final exponentiation calculation it determines the memory size of the operand memory blocks. To store the temporary values,

we use two operand memory-blocks each holding 9 words. Other sub-algorithms of the final exponentiation and their cumulative memory footprints are given in Figure 5.3 where the number of operand words is denoted by the weighted arrows on the right.

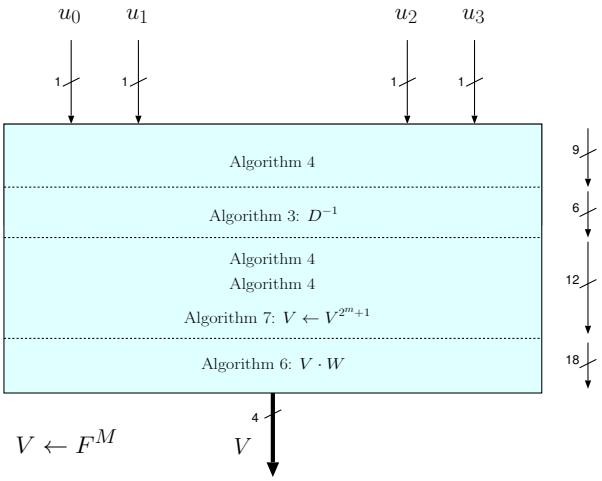


Figure 5.3: Data-flow graph for final exponentiation with memory-allocation sizes

5.2.3 Memory Allocation

Deciding on a hardware architecture means to consider various aspects like chip area, processing speed, power consumption, and also usability at a higher abstraction level. Implementing an algorithmic description on a given arithmetic unit means to formulate a sequence of operations according to the instruction set of that arithmetic unit. While an algorithm is just a description of operations, the respective implementation of an algorithm may have significant influences on calculation time and memory requirements.

The given architecture has two independent memory blocks for intermediate variables. The memory data outputs are connected directly to the data inputs of the arithmetic unit—each operand input of the arithmetic unit has a distinct memory block. Ideally the operands reside in disjunct memory blocks for each operation of the arithmetic unit. Finding a memory-allocation pattern, which provides this property, is not trivial. However, a pragmatic approach of trying to match algorithmic sequences and connect them accordingly already provides satisfactory results. For the case that this property does not hold and two operands of an operation reside in the same operand memory block, a copy operation between the memory blocks is required. For a optimal solution to this memory allocation

problem, we may use a declarative programming language such as Prolog to define a set of pre- and post conditions as well as a constraint set describing the data dependencies. Applying a cost function to this description would allow to find an optimal or near-optimal solution. The given implementation was optimized towards minimal copy operations and hence efficient memory usage where the number of registers for intermediate values is reduced to save chip area.

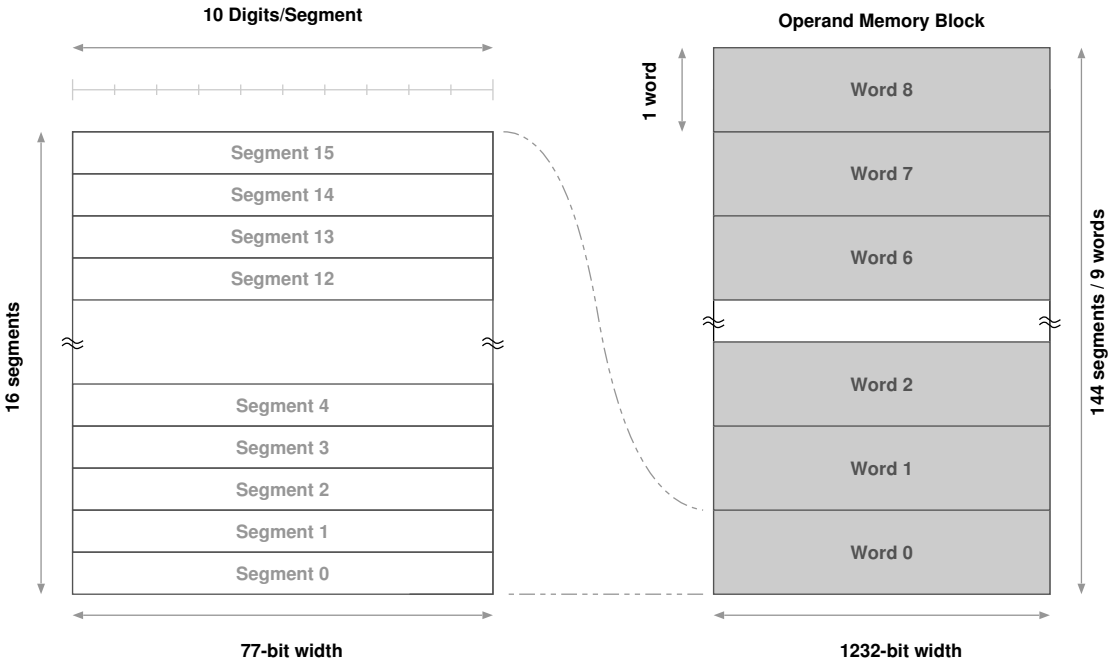


Figure 5.4: Operand-memory block organization of 77-bit architecture

5.2.4 Algorithmic Implementation

Designing an Application Specific Integrated Processor (ASIP) demands finding a trade-off regarding instruction set complexity and corresponding architecture size. A small instruction set may require a large number of instructions and possibly increase computation time but certainly helps maintaining a small architecture size. A complex instruction set may support a variety of instructions which helps the algorithmic implementation but at the same time contains an overhead where parts of the hardware architecture are only seldomly used and thus require more area. The algorithms used to calculate the η_T pairing are selected to only require a minimal amount of instructions, so that a minimal instruction-set processor with low area requirement can be applied. It turns out that we can implement the full pairing with just four elementary binary finite-field operations. The basic

set contains addition, squaring, and multiplication. Addition and squaring can be implemented very efficiently in one cycle and are dominated by read/write cycles to access operands in memory. The base-field multiplication is the most expensive elementary operation in terms of area and time. Hence, Section 5.3.1 focuses especially on an efficient implementation of this operation. The pairing algorithms also require an 'add digit' operation which is basically a XOR operation on the least-significant bit of an operand. It is implemented as small combinatorial circuit which can be enabled by setting a flag in the instruction word. As a consequence, the implementation of this operation does not require additional cycles or significant chip area. Regarding the control in the processor, the 'add digit' operation is a piggyback operation which is combined with the preceding instruction by bit flag in the instruction word. The microcode mnemonics `addd`, `muld`, and `sqrd` indicate the active 'add digit' flag.

Table 5.1: Instruction latencies of 77-bit based architectures. Multiplication latency is given for Karatsuba multiplier applying four iteration levels (K77)

	Read	Calc	Write	Overhead	Total
`add`, `addd`	16	-	16	3	35
`mul`, `muld`	-	134	16	3	152
`sqr`, `sqrd`	16	1	16	3	36
`djnz`, `slc`, `clr`	-	-	-	1	1

Compared to classic cryptographic algorithms, the given pairing algorithm contains many different computational steps to perform. To implement these computational steps and maintain flexibility for instruction sequence restructuring, we use a microcode-controlled architecture. The instruction memory is implemented as look-up table providing low-latency access to instruction words. Due to the fact that the look-up table is realized by a combinatorial network, it requires only a negligible amount of chip area. An instruction word, as stored in the instruction memory, contains an opcode with flag bits, two operand address words, and a result address:

`Opcode`	`Addr Operand A`	`Addr Operand B`	`Addr Result`
4 bits	4 bits	4 bits	5 bits

The program memory for a full pairing calculation contains 234 instruction words. Each instruction word consists of the operand code, addresses for the operands, and a result address. The address widths used in the instruction word

include additional bits which are not used in the final design. Minimal area requirement of the program memory is obtained by using a hardwired combinatorial network to store the program code. The final architectures do not address more than 9 RAM words of full precision requiring just 4 address bits. However the operand addressing widths comprise five bits. One reason is that the number of required operand words was subject to several changes during the design phase so that five bits were chosen to keep flexibility. Secondly, as the fifth address bits are ineffective they get trimmed by the synthesizer for area efficiency, so there is no drawback in terms of chip size and we can keep the signals for future use.

Table 5.2: Clock cycles of main subroutines

Arch	Algorithm 2	Algorithm 3						
		A4	A3(Inv)	2xA4	V·W	SqrW	A7	A6
GeminiK153	626 887	349	26 924	470	133	96 085	533	838

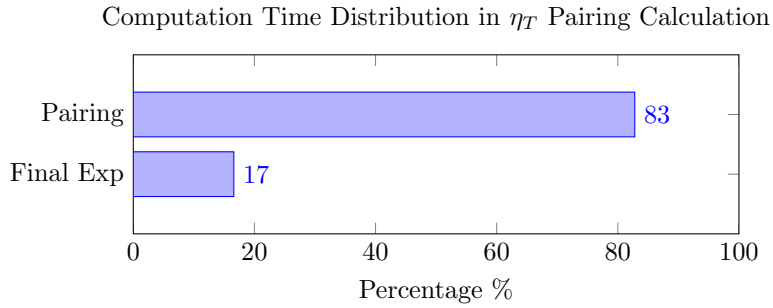


Figure 5.5: Time distribution of total pairing computation based on GeminiK153 architecture

5.2.5 Instruction-Code Translator

The operations to be executed by the micro processor are stored in program memory. Each entry in the program memory represents an instruction to the controller which may use the arithmetic unit to execute operations at certain memory addresses, set a loop counter to a certain value, or jump to another instruction to do functional loops. For the given architecture, an instruction consists of an opcode, on behalf of which the following bits of the instruction are interpreted. Basically, there are three types of instructions based on the number of logical arguments they take. Arithmetic instructions for multiplication and addition take three arguments

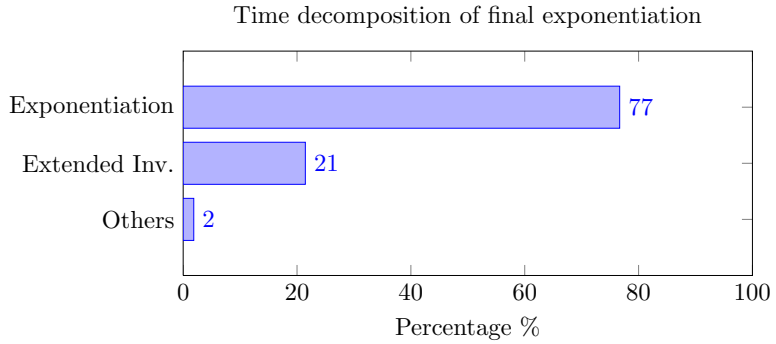


Figure 5.6: Time distribution of final exponentiation based on GeminiK153 architecture

to describe the memory locations of the two operands and the destination address to store the result to. The squaring instruction requires just two arguments as squaring is a single-input operation. Instructions as `clr`, `slc`, and `djnz` use only one input argument where in the case of `slc` and `djnz` it is interpreted as integer describing a loop counter or instruction index while it represents a memory location to be cleared for the `clr` instruction.

Listing 5.1: Metacode example for repeated squaring loop in final exponentiation algorithm

```
# Set loop counter
slc 612

# Algorithm fb4sqr (w0+w1s+w2t+w3st)^2
sqr a0 b1       # w0^2
sqr a1 a1       # w1^2
sqr a5 b2       # w2^2
sqr a6 a6       # w3^2  (result w3)

add a1 b1 b1    # w1+w0
add a6 b1 a0    # w3+w0  (result w0)
add a1 b2 a1    # w1+w2  (result w1)
add a6 b2 a5    # w3+w2  (result w2)
# END Algorithm fb4sqr

# Decrement and jump if not zero
djnz -8
```

Obviously, that modification of an algorithmic description at machine-code level is potentially error-prone and can get quite tedious as instructions are encoded in binary. This is why machine code is usually written and maintained using so-called mnemonics and alphanumeric identifiers for addresses instead of plain binary code. Mnemonics are easier to recall than binary codes and support interpretation of an opcode. A simple translator program transforms the list of mnemonics and address identifiers to actual machine code. Having an intermediate

abstraction level enables the programmer to read the code more easily and spot potential problems faster. Furthermore, architecture changes influencing the syntax or semantic can be realized more easily as only the translator program needs to be adapted to the new design. An example of this intermediate code is illustrated in Listing 5.1 showing the mnemonics for addition, squaring, and looping as well as identifiers for memory locations.

5.3 Arithmetic Unit

The computation of the η_T pairing is based on evaluating an elliptic curve defined over a finite-field. Evaluating an elliptic-curve equation or performing point operations on an elliptic curve is based on finite-field arithmetic. An elliptic-curve point P can be represented by finite-field elements $x, y \in F$ which satisfy the elliptic-curve equation. Elliptic-curve operations are defined by finite-field arithmetic which is applied to the finite-field elements defining points on the elliptic curve. Basically, these finite-field operations represent the bottom abstraction layer to calculate the pairing.

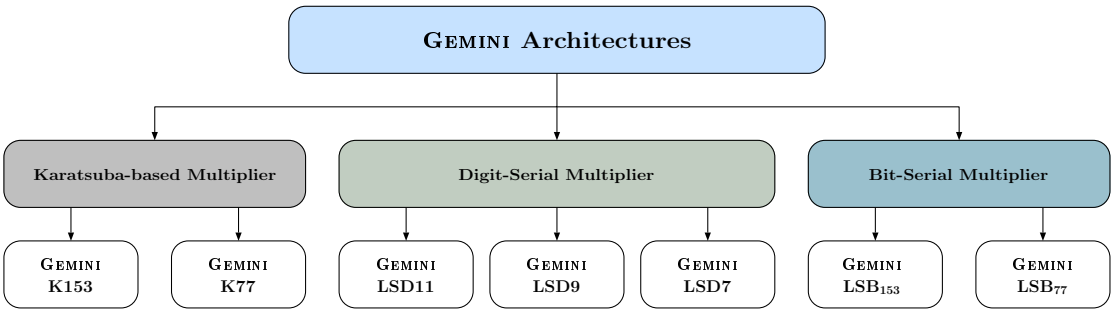


Figure 5.7: Overview on architectures

The operations in the base field are implemented in a unified arithmetic unit which provides modular operations as addition/subtraction, squaring, and multiplication. The η_T pairing requires only one field-inversion operation which is calculated in the final exponentiation. A field inversion can be realized by a combination of modular multiplications and squarings applying Fermat's little theorem. Due to the fact that field inversion contributes only a negligible amount to the total pairing computation time, a dedicated hardware for an inversion circuit seems not very reasonable. If, however, a faster implementation is desired spending an additional inversion circuit can improve calculation time. In case dedicated hardware for a faster field-inversion operation is desired, algorithms, for example, based on Itoh-Tsuhii's inversion algorithm [23, 28] are suitable for hardware implementations.

5.3.1 Multiplier Architectures

In many hardware cryptosystems multiplier performance and chip size contributed to multiplication has utmost influence on overall system characteristics. For binary finite fields, this is even more the case as addition, subtraction, and squaring can be implemented efficiently. Calculating a finite-field inversion with Fermat's little theorem also contributes to the importance of an efficient multiplication.

- Two-step multiplication
- Interleaved multiplication

Finite-field operations require a reduction step which reduces a result to obtain an element out of the finite set of field elements. The reduction operation is directly linked to the irreducible polynomial which defines the given finite field. Basically there are two types of modular multiplications namely two-step modular multiplications and interleaved modular multiplications. Two-step modular multiplications consist of an ordinary binary multiplication step followed by a separate reduction operation. While this approach may benefit from efficient standard multiplier architectures, it requires to hold an intermediate value of twice the operand width to be passed to the reduction step. The second multiplication type integrates the finite-field reduction into the multiplication.

A simple version of a finite-field characteristic-two multiplier in polynomial basis can be realized by performing the following two steps. Let m be the size of the multiplicand and multiplier polynomial both having a maximum degree of $m-1$. At first, we multiply the multiplicand for each degree z^n of the multiplier, where the multiplier coefficient is set to obtain the partial products. In polynomial representation this can be realized by a simple shift operation. Secondly, we accumulate the coefficients of the partial products according to each degree from z^0 to z^{m-1}. The accumulation of coefficients is realized by a simple exclusive-or conjunction as the applied field has characteristic two. Figure 5.8 illustrates a classic multiplication of two binary polynomials with multiplicand a and multiplier b. The filled dots denote an AND gate and the gray vertical bars denote XOR gates. The multiplicand and multiplier in this example have both degree $m-1$ giving an unreduced result of degree $2m-1$. Using Figure 5.8, we can estimate the gate complexity of an unreduced $m \times m$ parallel multiplier in a binary field. Given equal degrees $m-1$ for the multiplier and the multiplicand, we find a quadratic complexity based on the operand width.

$$\begin{aligned} (m-1)^2 &\quad \text{XOR gates} \\ m^2 &\quad \text{AND gates} \end{aligned}$$

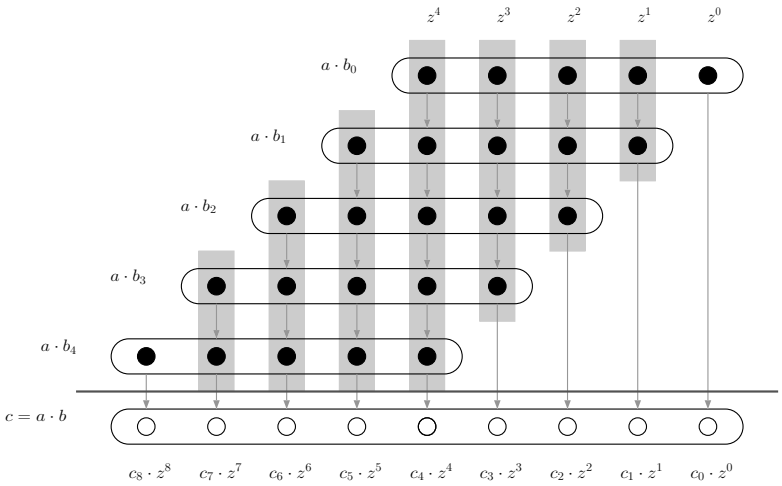


Figure 5.8: Classic binary multiplication of two polynomials a and b

While the multiplier area for moderate or small operand widths m may still be feasible, they are definitely out of reach for large operand sizes, especially if the overall design goal is a low-area implementation. While scaling quadratically with operand width, such multipliers are very fast and able to provide a result within only one clock cycle.

- Parallel multipliers (fully combinatorial)
- Hybrid multipliers (digit-serial/bit-parallel) (MSD/LSD)
- Serial multipliers (bit serial) (MSB/LSB)

So for a smaller multiplier size, we need to reduce the number of bits calculated in parallel. It is obvious that this results in a higher latency giving a slower multiplier. If paramount importance is given to low area, a fully serial multiplier with high latency may provide a good result. However, for a low-area design it is crucial to find a good balance between area and latency as overall computation time has direct influence on energy consumption. To find a good balance between area requirement, power consumption, and computation time, we investigate several different multiplier architectures for further evaluation. These multiplier architectures are discussed in the following sections.

5.3.2 Karatsuba-Based Multiplication

The Karatsuba algorithm is a fast multiplication algorithm introduced in 1962 by Anatolli Alexeevitch Karatsuba. In its classic version, it computes a polynomial

multiplication by three multiplications and two additions on polynomials halved in length.

In general, multiplications are computationally more costly than additions making the Karatsuba algorithm asymptotically faster than the schoolbook multiplication method as it requires less multiplication steps.

Classic multiplication or schoolbook multiplication consists of four multiplications at operand length and three additions to sum up the partial products. The classic multiplication has a computational complexity of $\mathrm{O}(n^2)$ where n is the number of digits of the operand. The following gives an example of the classic binary schoolbook multiplication:

$$\begin{aligned} C = A \cdot B &= (A_1 \cdot 2^m + A_0) \cdot (B_1 \cdot 2^m + B_0) \\ &= A_1 \cdot B_1 \cdot 2^{2m} + (A_1 \cdot B_0 + A_0 \cdot B_1) \cdot 2^m + A_0 \cdot B_0 \end{aligned}$$

With multiplications being a costly operation, an algorithm with less multiplications provides computational advantages. Karatsuba's algorithm does exactly this by eliminating one multiplication.

Karatsuba multiplication Karatsuba's algorithm is a so-called fast multiplication algorithm which applies the concept of *divide and conquer*. It splits the multiplication into sub-multiplications of smaller size and allows to reuse partial products. In contrast to the classic multiplication, Karatsuba's algorithm requires just three multiplications, four additions, and two subtractions. As the addition operation is generally less expensive than a multiplication, saving a multiplication for extra addition operations provides better performance. It turns out that Karatsuba's algorithm has a sub-quadratic complexity of $O(n^{1.58})$ improving the multiplication especially for large operands [31]:

$$\begin{aligned} C = A \cdot B &= (A_1 \cdot 2^m + A_0) \cdot (B_1 \cdot 2^m + B_0) \\ &= A_1 \cdot B_1 \cdot 2^{2m} \\ &\quad + (A_1 \cdot B_1 + A_0 \cdot B_0 + (A_1 - A_0) \cdot (B_1 - B_0)) \cdot 2^m \\ &\quad + A_0 \cdot B_0. \end{aligned}$$

While the original Karatsuba algorithm splits the operands in two segments, also other versions of Karatsuba's algorithm are available (e.g., splitting operands in three segments) [42, 56]. In this work, we investigated two-way and the three-way versions of Karatsuba multiplication and focused especially on the two-way version. A short illustration and according formulæ for the three-way Karatsuba approach is given in the appendix. The classic two-way version splits operands in two parts at each recursion step. So for an operand of bit length m, a recursion step splits the

operands in two $m/2$ bit segments. We observe that by application of Karatsuba's algorithm, the multiplication can effectively be performed on $m/2$ bit operands instead of m bit operands. This allows to reduce the size of the multiplier. Usually Karatsuba multiplication is applied to increase throughput by parallel instantiation of smaller multipliers or sequential reuse of a smaller sized multiplier. In this thesis we apply several recursion levels of Karatsuba's algorithm in an iterative way. The multiplier used to perform the core multiplication at the lowest recursion level can be of arbitrary type. This work uses a fully combinatorial Karatsuba multiplier architecture as core multiplier which is then used in an iterative version of Karatsuba's algorithm.

$$D_1 = A_1 \cdot B_1 \tag{5.1}$$
$$D_0 = A_0 \cdot B_0 \tag{5.2}$$
$$D_{0,1} = (A_1 + A_0) \cdot (B_1 + B_0) \tag{5.3}$$

With the number of recursion levels we can scale the size of the core multiplier. In order to build a 1223×1223-bit multiplication we can apply a low number of recursions and obtain a moderate reduction of the core multiplier size, or apply more recursion levels and obtain an even smaller core multiplier.

As the given finite field size is of prime order, there is no integer divisor so we pad the operands to obtain an integer which can be used as core-multiplier operand width. Padding a single bit to apply the first recursion level, gives a core multiplier operand width of 612 bits. Consequently, we have 306 bits at the second and 153 bits at the third iteration level without any need to pad additional bits. So by padding to 1224 bits we can split each operand into eight 153-bit segments. This allows to use a considerably smaller core multiplier of 153-bit width. Applying the algorithm again at the third level requires padding again which translates into 8+1 pad bits. At the fourth recursion level we again have to pad another bit obtaining a core multiplier width of 77 bits. So for a four level iterative Karatsuba application, we split 1232-bit operands into 16 segments of 77 bits each. Table 5.3 and Figure 5.10 show synthesizer results for area requirements of fully parallel multipliers where ordinary refers to the binary multiplication as generated by the synthesizer, binary refers to a parallel Karatsuba multiplication as proposed in [46], and simple referring to a parallel Karatsuba multiplier with a small modification as proposed in [45]. The simple multiplier uses the fact that when two operands of odd width are split one partial product pair is computed with the halved operand width rounded up and the other partial product pair is computed at the halved operand width rounded down. In this case, one of three partial products (either the product of the more significant segments or the product of the less significant segments) can be calculated with a smaller multiplier circuit.

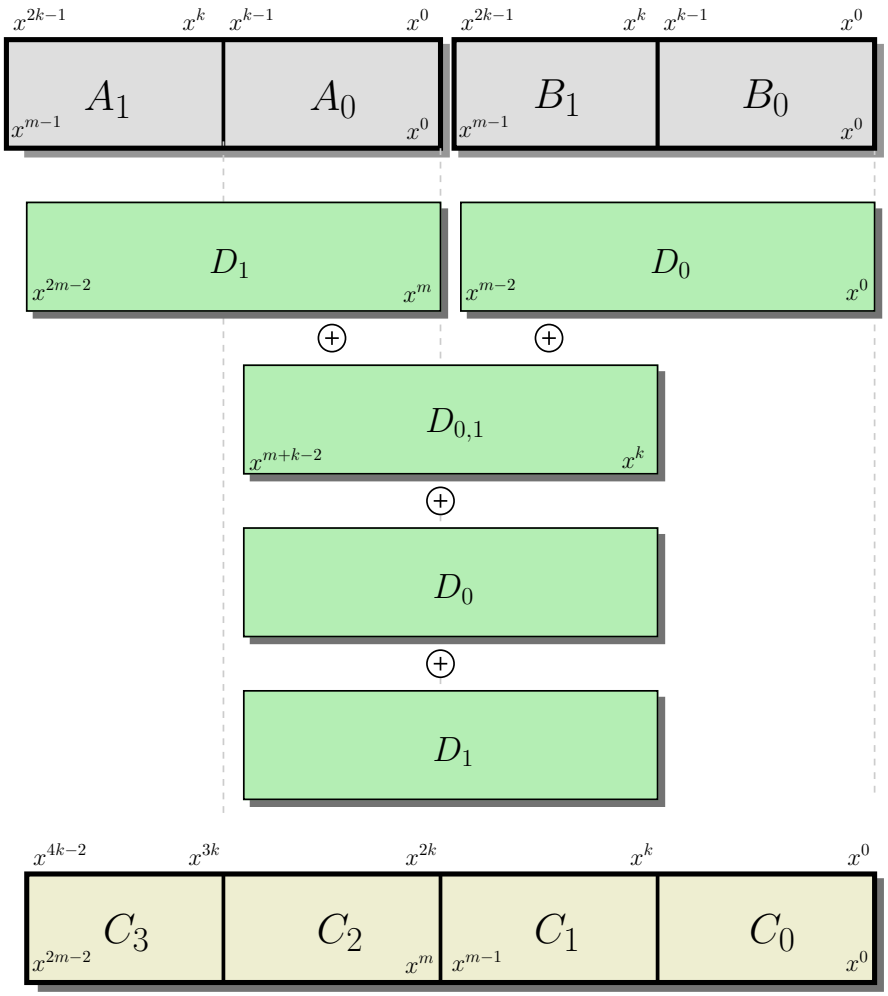


Figure 5.9: Segmentation and combination of one Karatsuba iteration

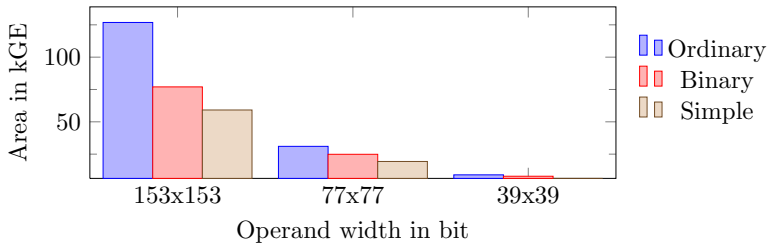


Figure 5.10: Area requirements of fully parallel multipliers

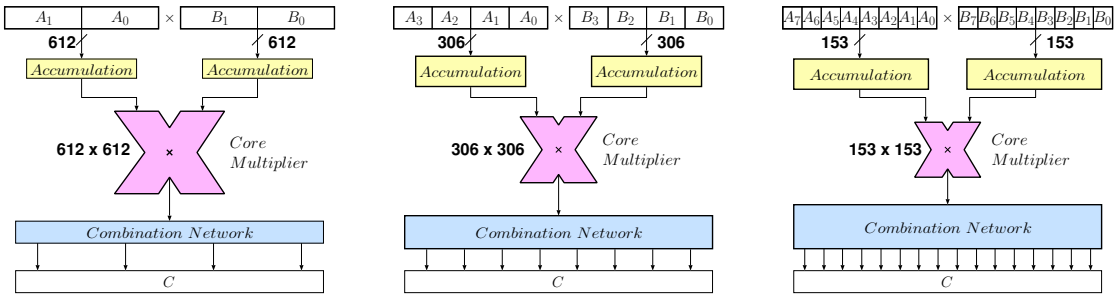
The area requirements of the parallel Karatsuba multipliers in Figure 5.10 show that the core multiplier size can be reduced to about one third with each recursion step halving the operands. Usually the top-level multiplication is performed

Table 5.3: Synthesis result for fully parallel multipliers

Arch		612x612	306x306	153x153	77x77	39x39	20x20
Ordinary	mm^2	26.5	7.1	1.18	0.29	0.08	0.02
	kGE	2825	754	127	31	9	3
Binary	mm^2	6.8	2.2	0.72	0.23	0.07	0.02
	kGE	728	236	77	25	8	2
Simple	mm^2	5.1	1.7	0.55	0.18	0.06	0.02
	kGE	548	181	60	20	6	2

using parallel multiplier instances at lower width allowing to use simple multipliers while maintaining computation speed and area requirement. To benefit from Karatsuba's algorithm area-wise, we perform the calculation of the partial products iteratively. Of course this implies a more complicated design and also infers additional circuitry but it allows to reduce the top-level multiplier area requirement significantly.

In the following, we want to describe a way to use an N-depth Karatsuba multiplication iteratively. At first, we split the problem in two tasks, namely the segmentation of the original operands and the combination of the partial products. In the following, we refer to the first task as *segmentation* and to the latter as *combination* task.



(a) Single-step (b) Two-step (c) Three-step

Figure 5.11: Basic architecture for iterative Karatsuba multiplier

Operand segmentation defines which combinations of the multiplicand/multiplier segments are to be multiplied by the core multiplier. According to Karatsuba's equation, we find that there are partial products which result from multiplying segments without prior addition with other segments (base segments) and that

there are partial products resulting from multiplying a sum of certain segments. The segment configuration of partial-product operands is obtained by executing Karatsuba's algorithm for the desired number of iterations and reordering the final polynomials. The second task is to find the correct combination of the intermediate partial products with respect to the result segments. Using polynomial multiplication, we can derive the result segment locations for each partial product. A partial product is passed to the combination circuit in three ways namely as higher segment, as lower segment, and as cross-term segment which is the sum of higher and lower segment. Each result segment is provided with a circuit which allows to add one of the three partial product types to its current segment value. Figure 5.11 illustrates an abstract view on the architecture and illustrates three examples of different iteration levels for the applied approach.

To compute an N-step Karatsuba multiplication we have to apply Karatsuba's algorithm N times. In each step, we decompose each multiplication in to three multiplications of halved operand width. So to compute a multiplication with an N-step application of Karatsuba's algorithm, 3^N partial products each of operand width $\lceil m/N \rceil$ need to be computed.

Two-step Iterative Karatsuba Multiplier (306-bit core)

The two-step iterative Karatsuba multiplier splits the operands in four segments each of 306 bits. A full 1223×1223-bit multiplication is obtained by calculating nine 306-bit multiplications. While this architecture would provide a low latency, the area costs of 181 kGE for the parallel core multiplier are not practical for low-area environments.

Three-step Iterative Karatsuba Multiplier (153-bit core)

The three-step Karatsuba iterative multiplier design (K153) uses a core multiplier width of 153 bits. The 1223-bit operands are split into $2^3 = 8$ segments each. To obtain a 2446-bit product, $3^3 = 27$ partial products need to be computed by the core multiplier. The maximum segment accumulation depth is eight. To calculate all partial products for a full multiplication, the corresponding operands, consisting of either plain or accumulated segments, and the respective order in the result has to be determined. The dependency graph in Figure 5.13 shows the 27 required core-multiplier operands for the three-step Karatsuba case. The operands are symmetrical for multiplier and multiplicand. Segments, which are directly multiplied by the core multiplier, are denoted as a^i where $i = 0 \ldots 7$. We call these segments which do not need to be accumulated base segments. All the other core-multiplier operands are built by accumulating a certain configuration of base segments where either two, four, or eight segments are accumulated. The superscript index denotes

the indices of the base segments which need to be accumulated to obtain the corresponding core-multiplier operand. So the core multiplier operand a^{0123} is obtained by accumulating the base segments a^0, a^1, a^2, and a^3. Figure 5.13 also illustrates patterns we can use to reuse previously accumulated segments and save computation cycles. In order to accumulate operand segments, we need at least one 153-bit register per operand port to keep the value of a segment to be added. The circuit for segment accumulation (Figure 5.12) provides a path to pass segments directly from the memory output to the core multiplier (base segments) and a path to accumulate segments. As there is only one register per operand port involved, several redundant read operations to the same segments are required. Using one accumulation register per operand port, we obtain a core multiplier saturation of about 65 % requiring 41 cycles.

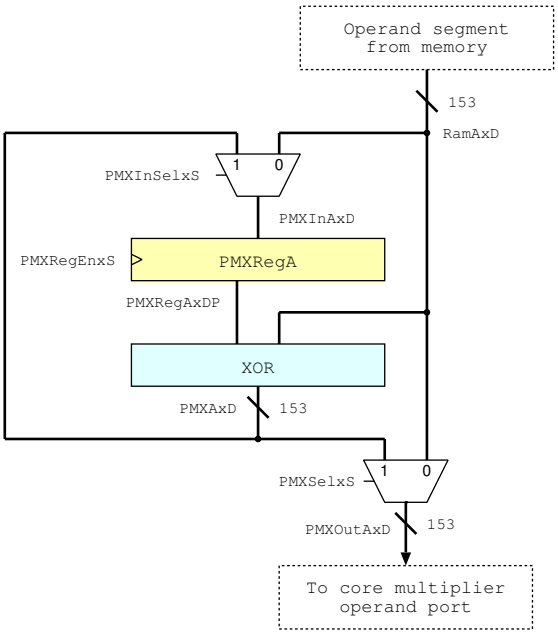


Figure 5.12: Segment-accumulation circuit using one accumulation register of 153-bit Karatsuba-based multiplier

Introducing a second accumulation register to the accumulation circuit allows to store and reuse intermediate results. As an example, the core multiplier operand a^{0123} may be calculated by adding the previously accumulated operands a^{01} and a^{23} so that less redundant read cycles are necessary. With two accumulation registers per operand port, we obtain a core-multiplier saturation of 93 % and can do a multiplication in 29 cycles which is near to the theoretical minimum of 27 cycles. While the two register approach may be more efficient in terms of computation

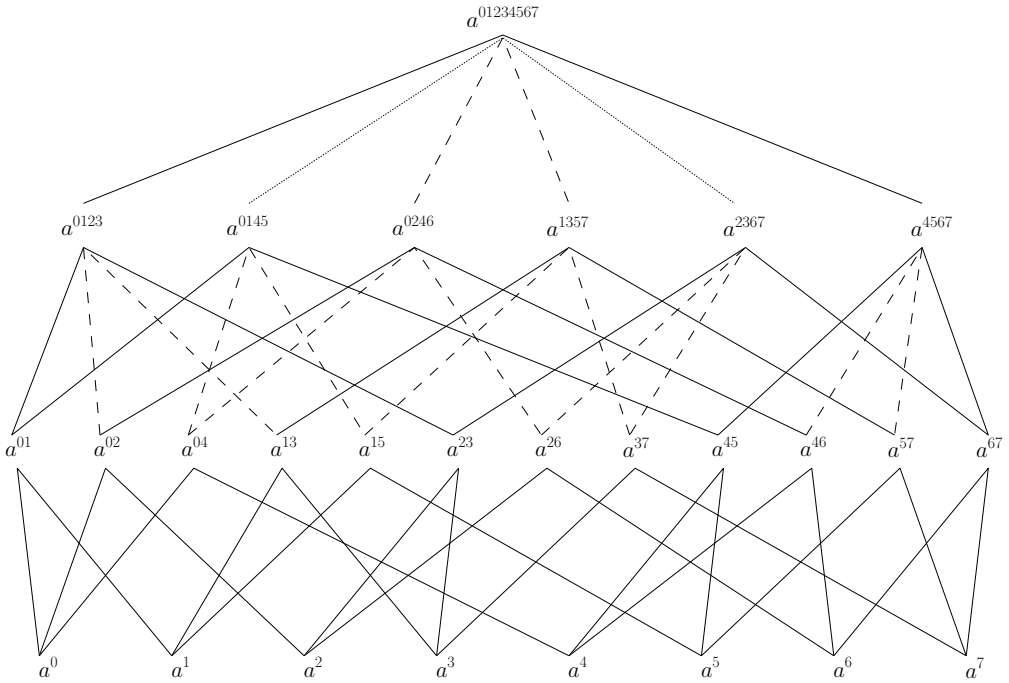


Figure 5.13: Segment-accumulation map for the three-step Karatsuba multiplier illustrating patterns to accumulate operands based on the eight 153-bit base segments.

time, it requires additional area and thus is not considered in further results in favour of the four-step Karatsuba approach presented in the next section.

Table 5.4: Three-step iterative Karatsuba multiplier

Arch	Partial Prod.	Acc.Register	Cycles	Core Saturation
K153	27	1	41	65 %
K153	27	2	29	93 %

Four-step Iterative Karatsuba Multiplier (77-bit core)

The four-step Karatsuba iterative multiplier (K77) splits each of the 1223-bit operands into $2^4 = 16$ segments. To calculate a 1223×1223-bit multiplication, the parallel core multiplier needs to compute $3^4 = 81$ partial products. The maximum segment-accumulation depth for core-multiplier operands is 16. The core multiplier can be significantly reduced in size consuming only 20 kGE. However,

the large number of partial products and the higher accumulation depth require more cycles to complete a multiplication. Also the combination circuit used to accumulate the partial products to the corresponding result segments is more complex and contains more multiplexer units. To compensate the larger number of partial products additional accumulation registers are added. Using the circuit in Figure 5.14, it is possible to save three intermediate results and traverse the accumulation map with fewer cycles. Obtaining a core saturation of 61 % and 132 cycles per multiplication, we have a fair basis to compare the three-step approach with the four-step approach as their core saturation is approximately the same.

Table 5.5: Four-step iterative Karatsuba multiplier

Arch	Partial Prod.	Acc.Register	Cycles	Core Saturation
K77	81	3	141	57 %
K77	81	3	132	61 %

As in the three-step Karatsuba approach, a segment-accumulation map was used to find an efficient control pattern for segment accumulation. Based on segment-accumulation patterns, which were presented in Figure 5.13, we use the same strategy to extend these patterns to the four-step version. However, as scheduling of segment accumulation is considerably more difficult in the four-step version, an automated approach for segment accumulation was applied. Consequently, we built a simulator in MATLAB implementing these patterns which allowed to test large randomized sets of accumulation sequences. With this approach, we could reduce the number of accumulation steps and improve core-multiplier saturation giving a more efficient multiplier. The simulator was then extended to generate VHDL code of the Finite-State Machine (FSM) controlling the four-step multiplier. The minimum obtained cycle count by the generator was 132 states for the given three accumulation register approach. This reduced multiplication latency by about 7 % resulting in 132 computation cycles.

The results for the Karatsuba-multiplier sizes demonstrate the fact that increasing the recursion level allows to reduce the multiplier size. The fourfold recursive multiplier (K77) consumes about 20 % less area than the threefold version (K153) with respect to the result illustrated in Table 5.6 and Figure 5.23.

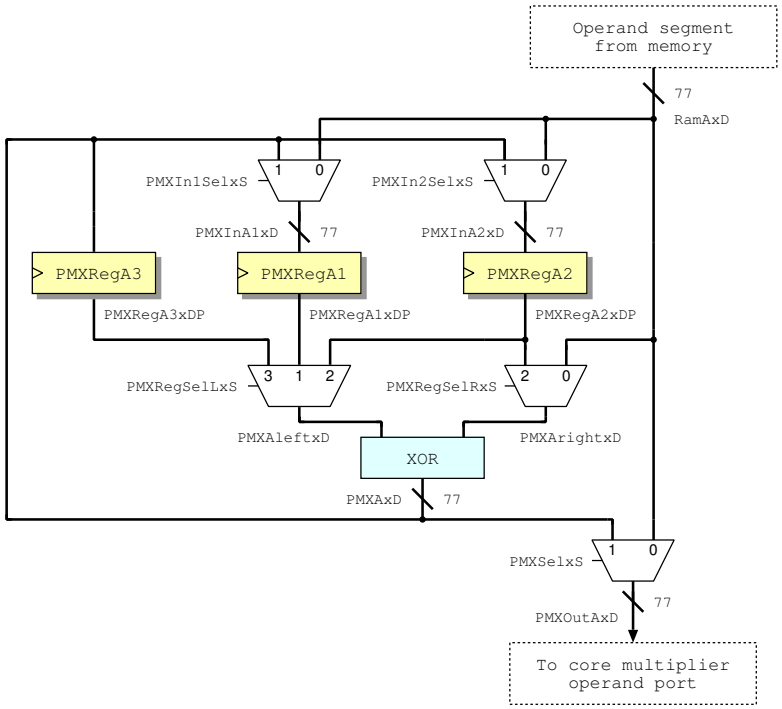


Figure 5.14: Segment-accumulation circuit using three accumulation registers of 77-bit Karatsuba-based multiplier

Table 5.6: Comparison of Karatsuba-based multiplier architectures

Arch	Area in kilo gate equivalents				T_{clk}	$\frac{cycles}{mul}$
	CoreMul	Comb[a]	Non-Comb	Total	[ns]	
K153	51	26	30	107	5 ns	41
K77	20	34	31	85	5 ns	132
LSD11	41	7	14	63	5 ns	112
LSD9	35	7	14	57	5 ns	136
LSD7	26	7	14	47	5 ns	177
LSB	4	7	14	25	5 ns	1223

[a] Excluding combinatorial area of CoreMul

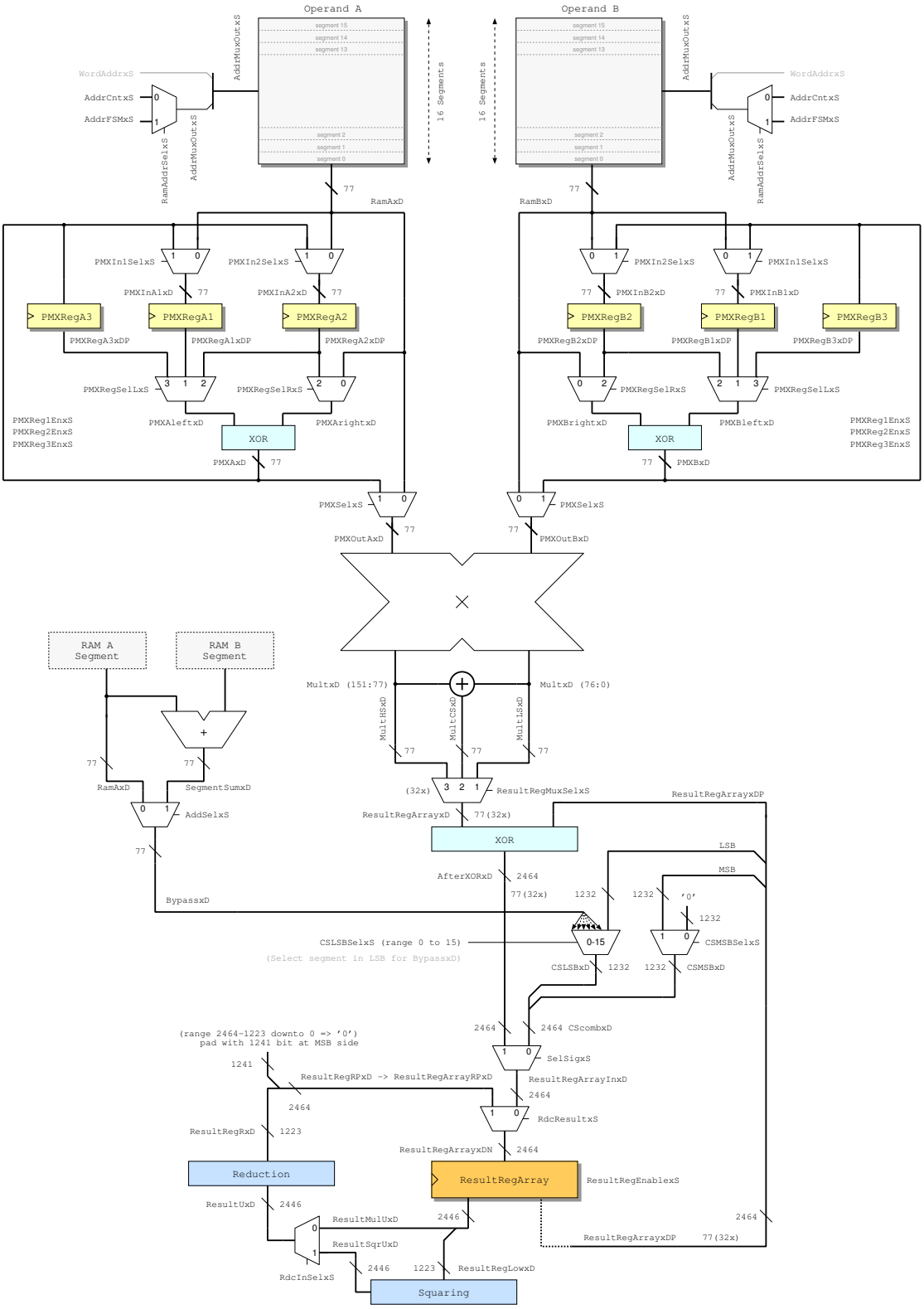


Figure 5.15: Arithmetic unit implementing four-step iterative Karatsuba multiplier

5.3.3 Digit-Serial-Based Multiplication

Designing hardware for resource-constrained environments such as smart cards or embedded devices mandates to balance area and computation time. Digit-serial multiplier architectures allow to find a good balance by selecting a digit size. Descriptions of hardware architectures for digit-serial modular multipliers were first introduced by [54] and discussed further in [27] and [35]. The basic difference to bit-serial multipliers is that $d > 1$ bit are processed in each computation cycle. Hence, increasing digit size yields faster computation times compared to bit-serial multipliers with reasonable increase in area. Digit-serial multipliers can generally be divided in two subtypes namely those processing the multiplier from its most significant to least significant bit and those processing the multiplier from its least significant to its most significant bit. In the following, we refer to the first type as MSD multiplier and to the latter type as LSD multiplier.

Digit-serial multiplier architectures have an interesting property of inherent scalability. The digit size d allows to pick a corresponding trade-off between the multiplier area consumption and its calculation time. The digit size defines the number of segments the multiplier is partitioned in and also defines the number of digit-wise multiplication steps necessary. With a digit size d an $m \times m$ bit multiplication requires $\lceil \frac{m}{d} \rceil$ digit multiplications of $m \times d$ width.

Other than the two-step multipliers where multiplication and reduction is done in two separate steps, digit multipliers allow to perform the modular reduction in an interleaved manner where each digit multiplication result of length $m + d - 1$ is reduced by d bits. For certain types of irreducible polynomials, the reduction step can be performed in a single step. For this type of polynomials, we can implement combinatorial networks for both the multiplication step and the interleaved reduction step.

While it is theoretically possible to increase the digit size up to the multiplier size, this may not always be of practical value. The reduction circuit is implemented with an exclusive-or network combining the multiplicand A with $d - 1$ bit defined by the irreducible polynomial. The propagation delay through this network can be minimized by structuring the XOR-gates as a binary tree of height $\lceil log_2(d) \rceil$ which gives the maximum propagation delay of the XOR-gate delays in the reduction circuit.

The algorithm for an LSD multiplier consists of three parts namely the digit multiplication, the interleaved reduction, and a final reduction circuit as illustrated in Algorithm 10 [35, 57].

For the pairing architecture, we consider a digit-serial GF(2^m) multiplier as it provides inherent scalability in terms of area and time complexity and allows to do interleaved reduction so that the number of registers to store the product can be reduced from $2m - 1$ to $m + d - 1$ registers. However, we need an additional

Algorithm 10 Least Significant Digit (LSD) Multiplier [35]

Input: $A = \sum_{i=0}^{m-1} a_i \alpha^i$ with $a_i \in GF(2)$
Output: $C \cong A \cdot B \bmod p(\alpha) \in \mathbb{F}_{2^{m-1}}$.

1: $C \leftarrow 0$
2: **for** $i \leftarrow 0$ to $\lceil \frac{m}{D} \rceil - 1$ **do**
3: $\quad C \leftarrow B_i A + C$ multiplier core
4: $\quad A \leftarrow A\alpha^D \bmod p(\alpha)$ interleaved reduction
5: **end for**
Return $C = C \bmod p(\alpha)$ final reduction

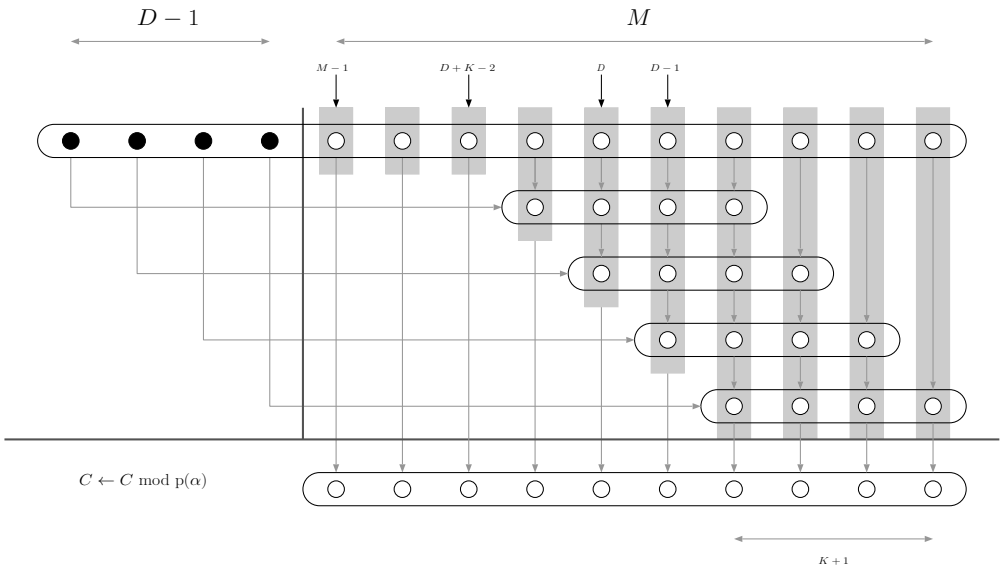


Figure 5.16: Example for a M=10, D=5 final reduction step exploiting reduction trinomial properties [35]

register to hold the $m-1$ bit multiplicand A in a shift register for the interleaved reduction.

Integrating an LSD multiplier mandates to decide on the digit size to be applied. To minimize the integration overhead and avoid generation of extra macro cells, the same infrastructure was used for the LSD multiplier architectures as with the Karatsuba multiplier architecture with a 77-bit operand-segmentation width. While we need to load the multiplicand to a distinct register A for interleaved reduction, we do not need an additional register for the multiplier and can directly read multiplier digits B_i from the SRAM data outputs. To avoid wasting computation cycles by digit sizes other than the prime factors of 77, which would result in additional registers and surplus computation time, we choose from digit sizes of 11,

7, or 1. With varying digit size, the multipliers differ in area complexity, latency, and critical path. Figure 5.17 illustrates the block diagram of the LSD-multiplier architecture using a digit size of 11 bit. Throughout the rest of this thesis digit serial multipliers are denoted by LSD with their digit size appended. So we refer to this multiplier as LSD11.

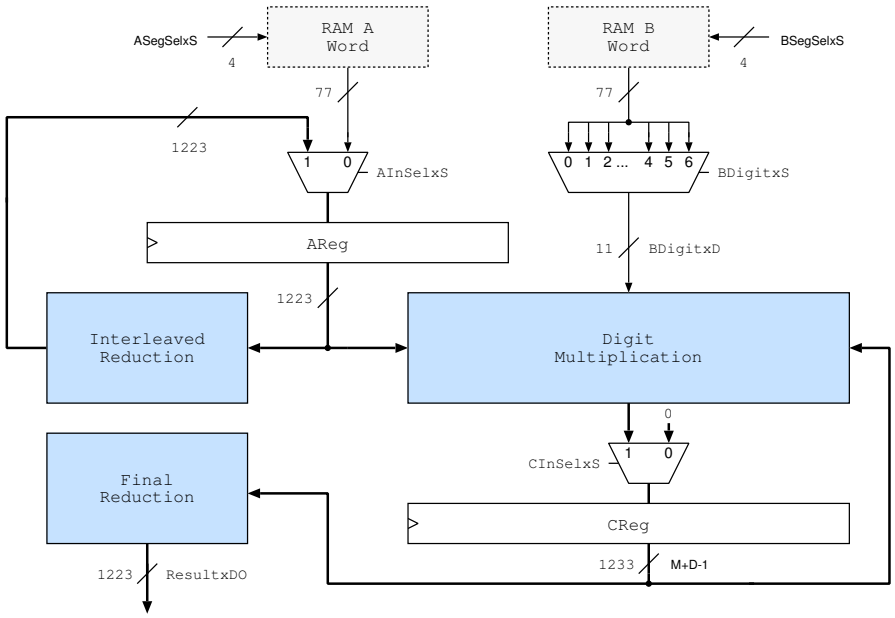


Figure 5.17: Block diagram of LSD11 multiplier

To reduce chip area requirements, we integrate the multiplier in the arithmetic unit sharing the result register with other arithmetic modules. In Figure 5.18, a simplified block diagram of the arithmetic unit is given. The result register is shared between the addition, squaring, and multiplication circuits requiring a multiplexer structure. Synthesis results for a range of clock periods constraints is illustrated in Figure 5.19. Where Figure 5.19a provides synthesis results for the Synopsys executables of `compile` and `compile_ultra` to illustrate the difference of their results. Througout this thesis, presented synthesizer results are generally obtained by executing `compile_ultra -gate_clock`. The given results in Figure 5.19 illustrate the low-area region of the resulting circuits being operational with a clock period of about 5 ns.

Bit-Serial Multiplication

The bit-serial multiplier is a special instance of the digit-serial multipliers with digit size equal to one. As discussed in the description of the digit-serial multiplier, the latency of this multiplier is considerably high. With digit size one

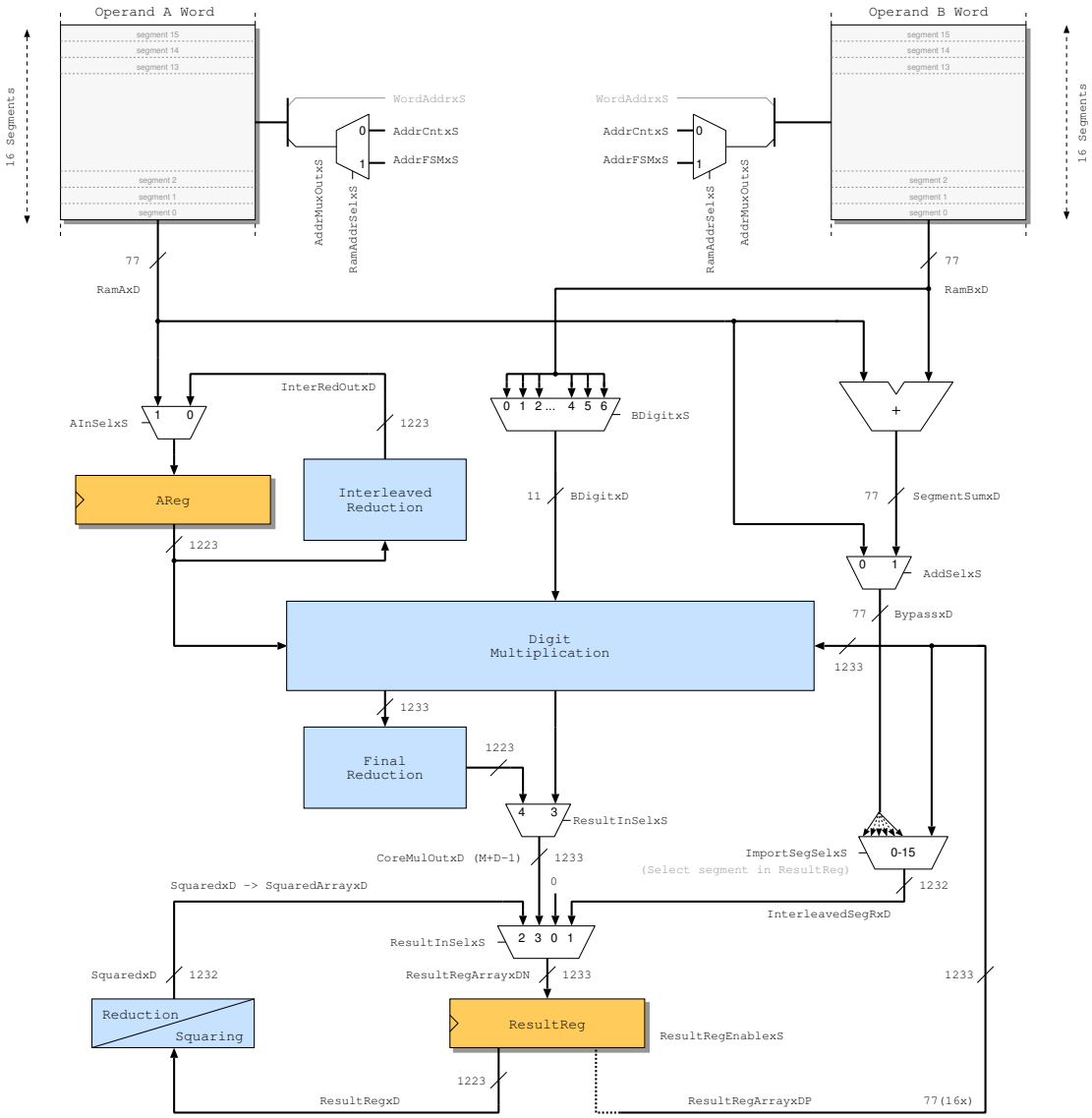
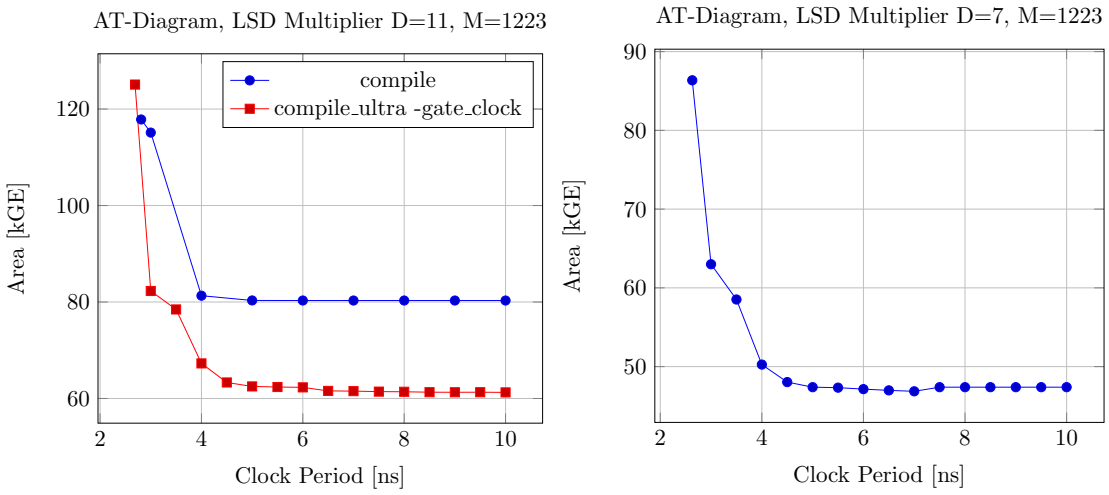


Figure 5.18: Arithmetic unit implementing LSD11 multiplier

($d = 1$) and a field size of $m = 1223$ we obtain a latency of $\lceil \frac{m}{d} \rceil = 1223$ cycles. Although this prolongs the overall computation time of the system it reduces the area requirement considerably as only a multiplication and reduction network of depth one is necessary minimizing area consumption.



(a) LSD11 (b) LSD7

Figure 5.19: Synthesis results of digit-serial multipliers

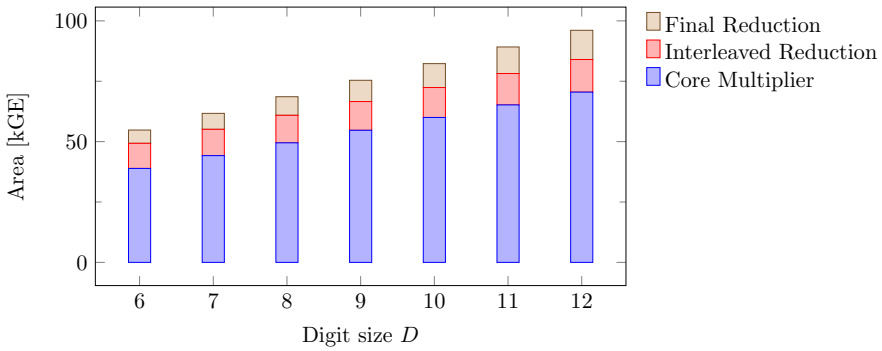


Figure 5.20: Estimated area distribution of digit-serial multipliers

Discussion / Evaluation

With the set of multiplier architectures presented, we now want to discuss and evaluate their performance regarding circuit area and multiplier latency. Throughout the evaluation of the architectural alternatives, we consider the following figures of merit.

Circuit area expressed as mm^2 or gate equivalents (GE). The circuit size is dependent on the applied technology which defines the size of circuit elements such as transistors and as a result standard cells. To quantify the complexity of a circuit, the mm^2 specification has to be accompanied with a technology specification. The area figures in this work refer to 180 nm CMOS technology

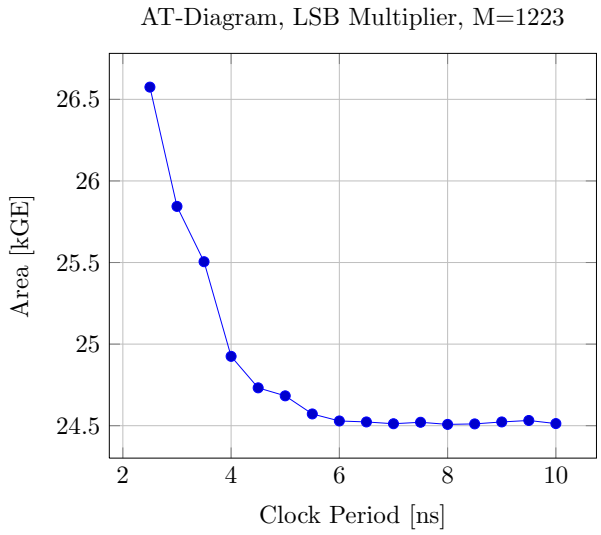


Figure 5.21: Synthesis result for bit-serial multiplier as a function of maximum critical path delay

using UMC/Faraday standard-cell libraries. To have a technology independent approximation of a circuit size, we also use gate equivalents. A gate equivalent (GE) is a technology-specific area measure which refers to the area of an elementary gate (usually the 2-input NAND gate) in that technology[2]. With the help of gate equivalents, we can give numbers on the complexity of a circuit independent of process technology.

Longest path delay or critical path. The inverse of this time period is the maximum operational frequency at which the given circuit can operate reliably.

Computation time or latency indicating the amount of clock cycles which are required for the circuit to compute the result.

Digit-serial multiplier architectures are scalable in terms of their digit size. For irreducible polynomials where $D \leq k$, the interleaved reduction and final reduction can be done in one cycle so that the latency is $\lceil \frac{m}{D} \rceil$ cycles. The area complexity is dominated by the combinatorial circuits for reduction and multiplication (Figure 5.20).

Synthesis results for the implemented multiplier architectures are provided in Figure 5.23 where the designs were constrained with a range of clock periods to

[2]UMC 180 nm: 1 GE $\cong$ 9.3744 μm^2

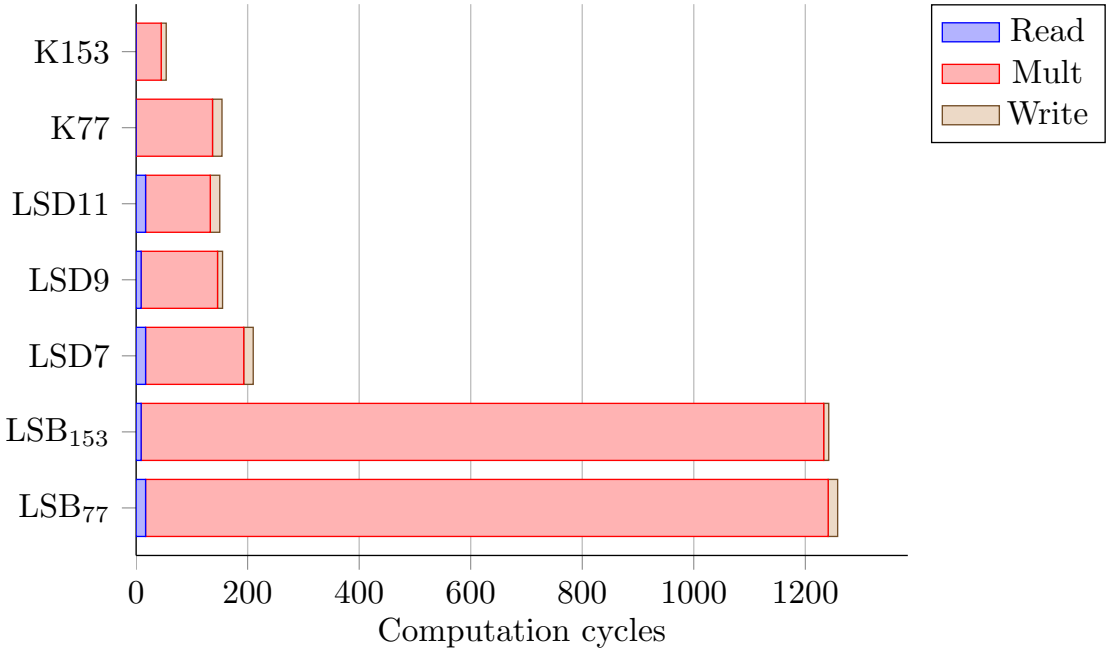


Figure 5.22: Time decomposition of multiplier architectures

evaluate them according to their longest path delay and corresponding circuit size. The results show that the Karatsuba-based multiplier architectures generally begin to saturate at lower clock speeds compared to the other designs. The multiplier architecture K153 contains a large parallel 153×153 bit core multiplier which makes the circuit ramp up in circuit size earlier than the K77 multiplier containing a 77×77 bit core multiplier which is considerably smaller. In both Karatsuba-based multipliers, the critical path of the longest signal propagation passes the parallel core multiplier. In the K153 multiplier, 16 core multiplier cells contribute to the critical path where in the K77 multiplier only 9 cells are part of the critical path. The digit-serial based architectures of digit sizes eleven, nine, and seven show a low-area region without significant area increase up to about 222 MHz. The area size is proportional to the digit size as combinatorial networks for digit multiplication, interleaved reduction, and final reduction directly scale with digit size. Consequently, the digit-serial multiplier with digit size seven consumes the lowest amount of area with about 50 kGE. The third multiplier-architecture type is a special case of the digit-serial multiplier where the digit size is one. The bit-serial multiplier has the lowest area complexity due to low-depth combinatorial networks for core multiplier and interleaved reduction. In contrast to the digit-serial type, the bit-serial multiplier does not need a final reduction step as the excess bit is already reduced by the interleaved reduction circuit. Due to the low combina-

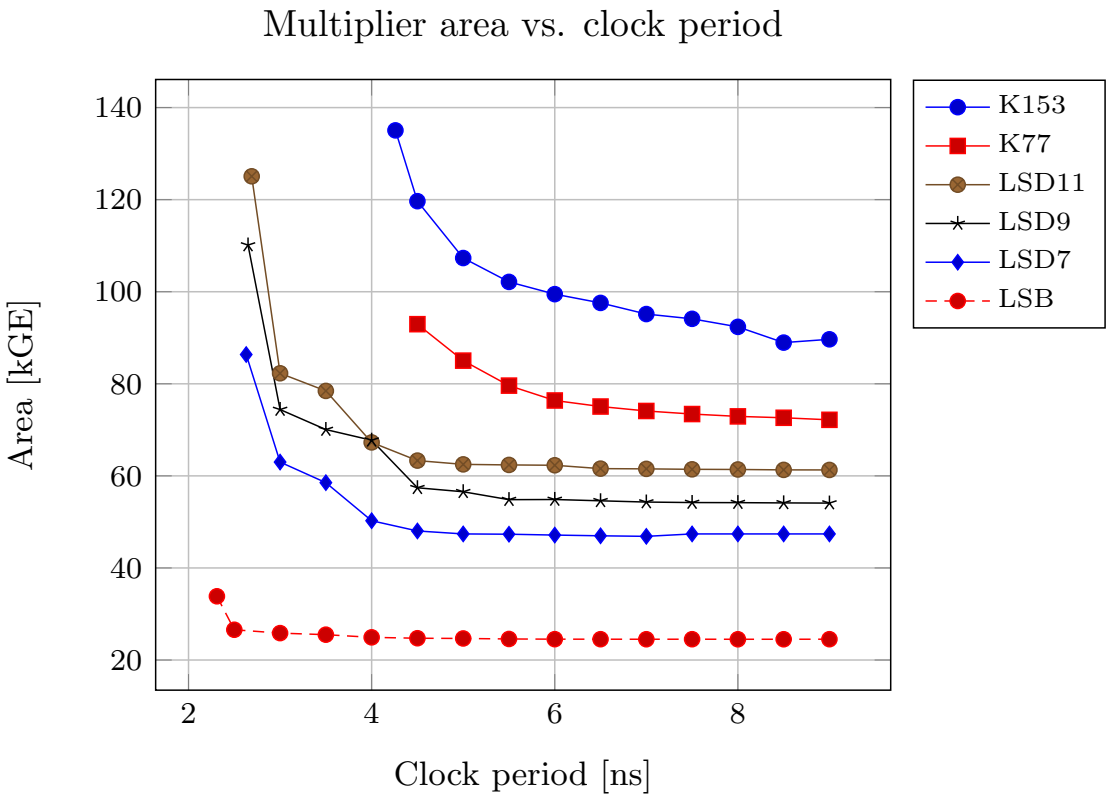


Figure 5.23: Area consumption of multiplier architectures

torial depth of the multiplicatio circuit in the bit-serial multiplier it contains a shorter critical path end. Synthesis results show that the bit-serial multiplier is operational with a clocking up to about 400 MHz without significant increase in circuit size. Finally, the bit-serial multiplier consumes the least amount of area consuming about 25 kGE. However, a latency of 1223 cycles per multiplication is very high compared to the other multiplier designs. Figure 5.22 shows the computation latency of the presented multiplier architectures including the associated read and write cycles according to the corresponding memory configuration based on either a 77-bit or a 153-bit interface.

5.4 Functional Verification

Assuring that a design complies with its specification requires extensive testing. Testing a design thoroughly at various design stages is important to avoid potentially large costs to fix errors in later design stages. Functional design verification was performed at the Register-Transfer Level (RTL) and at a post-layout level with a back-annotated netlist. The test setup of testbench and testvectors is basically the same for both abstraction levels. A detailed description on the testbench used to verify the implemented designs is given in Section 5.4.1.

5.4.1 Testbench

To test hardware models a so-called *testbench* is used that sets the inputs (*stimuli*) and observes the outputs (*actual response*) of a Model Under Test (MUT). The testbench uses so-called *testvectors* which describe the correct input and output values of a circuit. Hence, a testvector contains an arbitrary number of input to output pairs, where the output represents the *expected response* of a circuit corresponding to a stimulus. In the given setup, the testvectors are generated using a software reference model which serves as functional reference and provides correct input to output pairs.

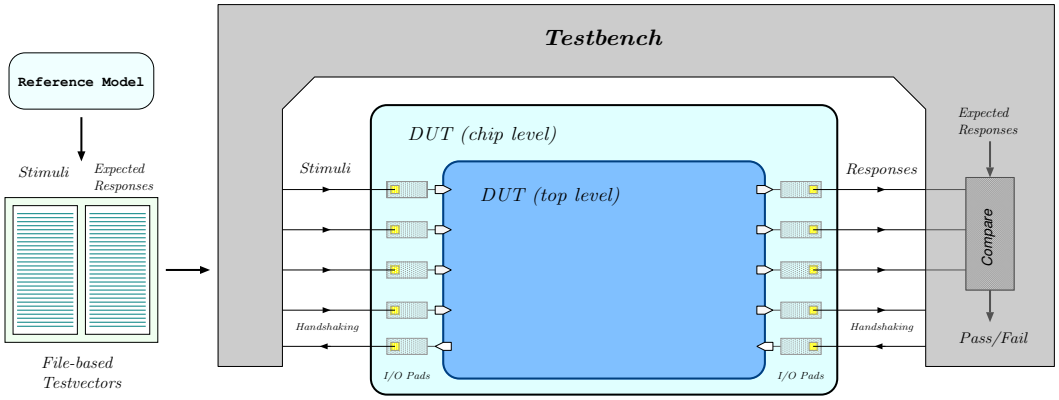


Figure 5.24: Testbench configuration

Due to the large operand size in this high-security pairing architecture, functional verification is prone to consume a considerable amount of time. For circuits of comparably low complexity exhaustive testing with random values is certainly an option as the total test time allowing to assume full functionality is usually acceptable. For our pairing architecture, opting for exhaustive testing alone is probably not a good decision as the large operand sizes result in an up-scaled total testing time. As a consequence, we want to minimize testing time as much as pos-

sible while maintaining good coverage of functionality. Our testvector set consists of elementary function tests designed to cover the mathematical properties of a finite field and an additional set of random elliptic-curve points to finite field pairs serving as random test vectors.

Test-vector Sets

Verification of the design is based on a set of test vectors providing stimuli and expected responses for the module under test. As the design is micro programmed, test vectors are in general linked to a microprogram which implements a certain input to output transformation. The microprogram is integrated as a hardcoded Look-Up Table (LUT).

The top architecture is designed to calculate a full η_T pairing including final exponentiation. Hence we have two elliptic-curve points as input and an $\mathbb{F}_{2^{4m}}$ element as output. The architecture basically allows read and write access to any operand memory location. However, defining fixed memory locations for inputs and outputs has certain benefits. On the one hand, fixed memory locations make testing easier as subsequent algorithmic tasks can be tested without adapting to new allocation patterns. On the other hand, fixed memory locations for stimuli and results allows to drop hardware elements which would be necessary otherwise. As an example, placing our result in a fixed memory block allows to avoid an output multiplexer multiplexing the memory outputs to the top-level output signal.

The functional verification was done using Modelsim SE 6.4 by Mentor Graphics. Simulating the RTL model calculating the full pairing is quite time consuming as about 1.3×10^6 cycles need to be simulated for a single pairing simulation. The effective simulation time is a dependent on the amount of signals contained in the waveform set of the simulator. So to avoid simulation runs taking many hours to complete the simulator waveform sets were adapted according to the required level of detail. Using waveform sets of different verbosity for circuit debugging and algorithmic verification allowed to perform testruns in only a few hours. To further reduce simulation time, another testvector set was constructed focusing on testing the functionality of the ASIP using only a short sequence of instructions. This testvector set basically tests arithmetic operations of addition, squaring, and multiplication (ASM). The AMS micro program also covers implicit tests of other instructions supported by the design such as loop functionality and memory clearing.

- Stimuli: P,Q (4 words of 1223 bits)
- Expected Response: $F^M \in \mathbb{F}_{2^{4m}}$ (4 words of 1223 bits)

The memory locations of stimuli and actual results can be defined arbitrarily. The respective locations are hardwired with constant definitions at compile time.

After starting the processor by raising the start signal `StartxSI`, it begins to read four words of 1223 bits to the memory locations $a0, b1, b0$, and $b2$. The address aN refers to operand memory A where N is the operand word address. The address bN references to operand memory block B using the same word-address scheme. The given placement of the input words is obtained by memory allocation and instruction scheduling optimized for the given architecture. The read and write sequences are implemented using a four-phase handshaking protocol. To indicate valid data ready at the input, the signal `DataInReqxSI` is set by the testbench. The design may then read the input and store it to its operand memory and use `DataInAckxSO` to notify acceptance of input data to the testbench. The testbench then responds by reseting the request signal after which the design resets the acknowledge signal. This completes one four-phase handshake cycle. This four-phase handshaking protocol introduces extra latency as each control signal gets restored to its initial value. Compared to a two-phase handshake protocol, the data rate is effectively halved but has the advantage that it makes the communicating systems more independent and allows to operate them at different clock speeds. This assures that the producer and consumer system can be developed almost independent from each other.

After reading the four input words, the processor automatically starts to compute the pairing. When it has executed the microprogram, it signals its ready state by raising `ReadyxSO`. The testbench then may initiate a read sequence by following the output handshake sequence using the signals `DataOutAckxSI` and `DataOutReqxSO`. The 4892-bit result is generally read from the memory locations $b2, b4, b8$, and $b6$.

5.5 Memory Configuration

At an early design phase, the memory was implemented as behavioral model based on registers. This behavioral memory model provides the same interface and characteristic as a macrocell memory but at the same time offers a high degree of flexibility when memory sizes are not yet fixed and still subject to change.

At a later point, the behavioral SRAM models were exchanged with SRAM macrocells. Customarily a macrocell generator is used to generate appropriate memory macrocells. In this work, a macrocell generator (Memaker FSA0A 2009 1.2.1) by Faraday Technologies of was used to evaluate and generate supported macrocell configurations. The generated macrocell memories conform to UMC 180 nm 1P4M logic process design rules and can be incorporated with Faraday's 180 nm standard cells. All macrocells were generated with an internal macrocell power ring of 2 μm and an output loading of 10 nF. The macrocells used in the final designs are given in Table 5.7 for the two memory interfaces of 77 and 153 bits.

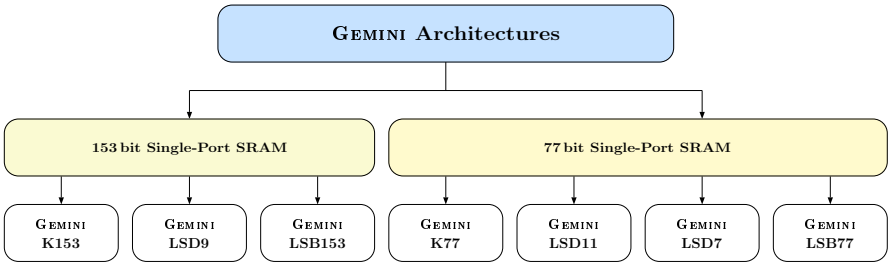


Figure 5.25: Architectures based on memory configuration

To save chip area, we made use of the *byte write* feature provided by the macrocell generator. Usually a memory block provides a *write enable* input which is used to indicate that the current data inputs should be stored to the (synchronous) RAM block at the following clock edge. With a byte write signal, we can enable the respective byte of the data input. The memory generator allows to generate byte write enabled memories and also defines the number of bits per byte. For an efficient memory use, we set the bits per byte setting to an integer divisor of the data width of the memory. As a result, the generated memory has one byte enable signal for each byte. This allows to connect the data input directly to the macrocell inputs and reduce the number of primary inputs at the chip level to the number of bits per byte. Two macrocell memory configurations are used for implementing the final designs. The data widths being 153 bits and 77 bits. The first macrocell uses a byte size of nine bits which translates into a byte-enable signal of length $153/9 = 17$. The second macrocell is configured to have a byte size of eleven bits and a byte-enable signal of seven bits. Table 5.7 gives a macrocell configuration as *number of words* $\times$ *bit/byte* $\times$ *number of bytes*.

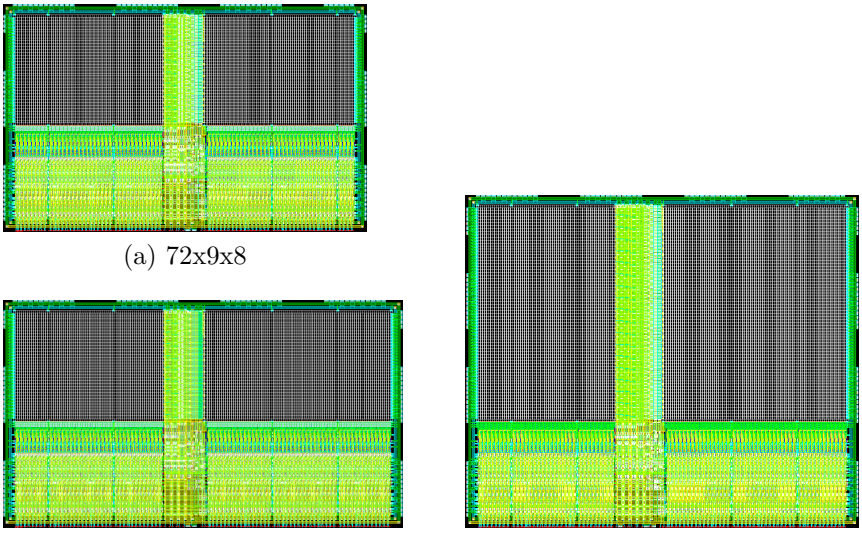
Table 5.7: Design options for operand memory configurations (based on Faraday Memaker, UMC 180 nm).

Width [bit]	RAM type	Configuration	Width [µm]	Height [µm]	Area [kGE]	Total [kGE]
153	Macrocell	72×9×9	333.96	189.04	6.72	12.91
	Macrocell	72×9×8	304.20	189.04	6.19	
77	Macrocell[a]	144×11×7	1183.82	181.36	22.9	22.9
	Macrocell	144×11×7	326.52	276.96	9.65	9.65

[a] Block muxed macrocell (SU-type) with wide aspect ratio

To obtain a memory interface width of 153 bits, we combine two smaller sized macrocells as bit widths exceeding 144 bits are not supported by the macrocell generator. The two macrocells $9 \times 9 + 8 \times 9 = 153$ are then wrapped in a new VHDL entity building the actual memory interface of 153 bits. The first 77-bit macrocell was not used in favor of the second 77-bit macrocell as it requires significantly less area due to a better aspect ratio. The 77 bit configuration has an area per bit ratio of $8.16 \frac{\mu m^2}{bit}$ where the 153 bit configuration has $10.95 \frac{\mu m^2}{bit}$. So in the given implementation the 77 bit memory has a better area per bit ratio than the 153 bit memory as it consists of only one macrocell per operand memory block.

Every SRAM macrocell contains column decoders, row decoders, memory cells, pre-charge amplifier, and sense amplifiers. While the column and row multiplexer are generated according to the configuration passed to the macro generator, the sense amplifiers are not. Practically the area of the sense amplifiers is dimensioned to drive the capacitive load corresponding to a SRAM macrocell at maximum height (i.e., the maximum number of words provided by the macrocell generator). So the sense-amplifier area is constant for the whole range of words supported by the generator. This means that sense-amplifier area dominates the macrocell area for SRAM macrocells of low-word count. Internally, the words are organized as horizontal rows and the bits are organized as vertical columns. So the word width is proportional to the macrocell width and the number of words is proportional to the macrocell height with a constant offset due to the sense amplifiers. Observing the two macrocell configurations for the 77-bit interface we have the same bit capacity and the same logical interface, but very different aspect ratios and area requirements.



(a) 72x9x8

(c) 72x9x9

(b) 144x11x7

Figure 5.26: Internal view of SRAM macrocells illustrating different aspect ratios as used in final ASIC implementations

Chapter 6

Design Flow

In this chapter, we want to discuss the major tasks of a digital semi-custom design flow from a chip designer's perspective regarding the architectures presented in the previous chapter. We start by a discussion of the transition from a behavioral description to a structural description and continue to discuss the final transition to a physical description in order to complete an ASIC design cycle.

6.1 Introduction

In the following, we refer to *design flow* as a set of tasks required to transform a system description at algorithmic level to a physical circuit description of the system. The final description of a physical Very Large Scale Integration (VLSI) circuit complies with functional, electrical, and design rules and is ready for physical production by a semiconductor foundry. A typical design flow is based on a set of software tools often referred to as Electronic Design Automation (EDA) tools. The collaborative set of these modular tools represent the so-called design flow which supports the chip designer to design, simulate, and analyze a digital hardware design. In contrast to a full-custom design flow where each cell is designed individually, a semi-custom design flow reuses descriptions of elementary cells commonly referred to as standard cells. A typical semi-custom design flow consists of two parts namely a front-end part and a back-end part (Figure 6.1).

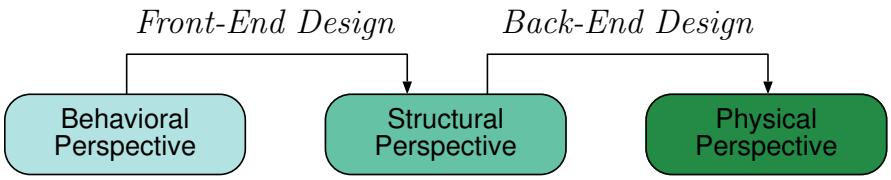


Figure 6.1: Abstract view on design flow

6.2 Front-End Design

Front-end design refers to the tasks required to transform an algorithmic high-level description or behavioral model to a gate-level circuit description. A gate-level circuit description as provided by a front-end flow represents the basic input to a back-end flow. The basis for a front-end flow is a behavioral model or algorithmic description which serves as reference for subsequent design steps. Usually, the behavioral model is provided by a reference model in a high-level software programming language. This reference model is also used as functional reference. Hence, it is sometimes called a *golden model* as it is used to create correct value pairs describing the input to output relation of architectural blocks. For a cryptographic hardware implementation, this may be a pair of plaintext and corresponding ciphertext and vice versa. Based on the reference model a hardware architecture is designed and implemented by means of a hardware description language. This work uses VHDL[1] to implement the presented hardware architectures and also to build testbenches to test the correct operation thereof. In an ASIC front-end design flow, a so-called synthesizer is used to transform a hardware description usually given as VHDL or Verilog code to a gate-level circuit description. The synthesizer is able to recognize structural elements and patterns in the HDL code to synthesize a circuit using standard cells. Standard cells are standardized units which represent small generic subcircuits of units such as NAND gates, multiplexers, or bistabiles. The gate-level netlist is a circuit description based on standard cells and their logical interconnections.

6.2.1 Reference Model

Reference models are essential for implementing and verifying a hardware design. A reference model is usually implemented in a high-level software programming language and used as functional reference providing so-called testvectors. A testvector consists of a correct input/output value pair describing the expected transformation of a stimulus (input) to an output (expected response). Hence, by applying stimuli to a hardware design under test, the correct functionality can be verified by comparing the actual output of the tested module with the expected response in the testvector pair. Testing complex hardware designs requires to have good coverage of the possible state space in a design to detect errors. A reference model can be used to create large sets of testvectors and hence improve the chance to detect errors in the design. Cryptographic hardware implementations naturally benefit from the strong input to output dependency intrinsic in cryptographic algorithms. In general, even single bit errors have a significant effect on the result

[1] Very High-Speed Hardware Description Language

as cryptographic designs are highly deterministic and usually apply non-floating number representations. Reference models can be implemented at various abstraction levels regarding to the actual hardware implementation. At low abstraction levels, a cycle true reference model may allow to spot errors with great detail but at the same time provide little flexibility as later changes require changing the hardware and the reference model. On the other hand, high abstraction levels may not provide enough granularity to identify errors quickly. For implementation of the given hardware architectures, basically two reference models were used. The first being a reference model implemented in MATLAB modelling the operations as provided by arithmetic units. The second being a bit true reference model for the top-level implementation, which provides valid testvectors for testing the design. The top-level reference model uses implementations of the RELIC [4] library which can be built to support operations based on various finite fields and elliptic curves. The implemented top-level hardware architecture is basically a microcoded processor with instruction memory and operand memory. To test this architecture, an algorithmic description corresponding to the algorithms implemented with the reference model is applied. So the reference model and the microcode of the processor architecture process the same sequence of operations. The actual microcode of the microprocessor was generated from an assembler-like metacode using a translator program (see Section 5.2.5) where the metacode is an architecture-based transcription of the algorithm implemented in the software reference model.

6.2.2 Modeling in HDL

Specification of digital hardware may use different abstraction levels for circuit description such as low-level descriptions at gate-level up to a description at a behavioral-level. The hardware description language VHDL is designed to allow circuit descriptions at several abstraction levels. With VHDL, we can describe a circuit in a way which can be further processed by a VHDL compiler for synthesis. A VHDL compiler allows to synthesize an HDL description to a gate-level circuit description which is represented by a so-called *netlist*. A netlist consists of a set of standard cells and their interconnection according to the given circuit description. The synthesizer uses standard-cell descriptions from a standard-cell library which is usually licensed from a standard-cell library vendor to build circuits. The synthesis result represents a circuit which then can be assessed according to area consumption of the contained standard cells and timing characteristics. VHDL also provides functionality to build testbenches for design testing. The Design Under Test (DUT) or MUT is simply instantiated in the testbench and connected to signals providing stimuli and reading actual responses. The testbench uses stimulus/expected response pairs provided by testvectors for testing. A detailed

discussion of the implemented testbench and applied testvectors is given in Section 5.4.1.

6.2.3 Synthesis

Synthesis in the given context is the transformation of a circuit description given as HDL code to a gate-level netlist. The circuits presented in this work were synthesized with a VHDL compiler by Synopsys (D-2010.03-SP1-1). The synthesizer's degree of freedom to build circuits can be defined by setting arbitrary *constraints*. In general, the clock input of a circuit is constrained to ensure that the synthesis result is guaranteed to work at an operational clock frequency. If the synthesizer is not able to build a circuit conforming to a given clock constraint, it reports this fact by a negative *slack*[2]. A negative slack does not mean that the circuit is dysfunctional rather it means that the minimum clock period needs to extended by the absolute value of the negative slack. So if a circuit is constrained by a clock period of 10 ns and the synthesis result is reported to have a negative slack of 1 ns, the circuit is functional at a clock period of 11 ns. A positive slack indicates that the minimum clock period can be reduced by the amount of slack given. Basically, the synthesizer uses timing information of the standard cells in the circuit and their respective driving strength to calculate the clock timing.

The synthesizer needs information on capacitive loads to determine the amount of time a cell requires to drive its output properly. At a basic level, the specifications as provided in the standard-cell documentation is used. In case of a timing violation due to high capacitive load on an output net the synthesizer may increase the driving strength of a cell by using cell subtypes of increased driving strength. A cell with more drive strength is able to drive outputs faster but also requires more chip area as the cell's internal transistors need to be larger in order to drive higher currents. However, the synthesizer has no information about the actual cell placement and according wire interconnections at this stage in the design flow. As such, no accurate information on the capacitive load of a wire can be given. Gate outputs with high fan-out (i.e., gates having a large number of downstream gate inputs) have a high capacitive load due to the gate capacities of the downstream gates. The capacitive load due to gate fan-out can be estimated using the information inherently given in the synthesized netlist. However, the wire routing of these signals and their parasitic capacitance is not known. Long and intersecting wires contribute to parasitic capacity. This is why a layout benefits from routing wires orthogonally in subsequent layers as it reduces the parasitic capacitance. As a result, the metal-layer interconnections share a common orientation of either horizontal or vertical where neighboring metal layers have an orthogonal orienta-

[2]Time difference between actual signal arrival time and required signal arrival time

tion to minimize parasitic capacities. To estimate the capacitive load on a wire, a synthesizer uses a so-called *wire-load model*. A wire-load model basically assumes a given capacitive load corresponding to a certain fan-out numbers. As an example, it may assume a capacitive load of 10 fF, 20 fF, 40 fF, and 60 fF for a fanout below or equal to 1, 2, 4, and 10 respectively. As these assumptions may or may not apply, different wire-load models are used for approximation. A designer may configure the synthesizer to use a certain wire-load model. If no explicit directive is given, the Synopsys synthesizer chooses a wire-load model automatically. The synthesizer's decision on which wire-load model is applied is usually based on the number of gates within a certain block. Given the case that the synthesizer has four different wire-load models A, B, C, and D available, it may pick one of these models for block sizes of about 10, 20, 40, or 100 kGE. If there are interconnections between such design blocks where each is associated with a certain wire-load model, the synthesizer has to decide which wire-load model to use. This decision is made according to the mode of the wire load model (enclosed/unenclosed). In enclosed mode, a sub-block inherits the wire load model from the block which fully encloses the sub-block. So, as a consequence, the results obtained using wire-load models can only give estimates on the actual wire load of an actual layout. This is why the improvement of simulation models is subject to active development in semiconductor industry. As an example, advanced simulations may use three dimensional models instead of just considering horizontal intersections.

The results based on wire-load models differ naturally based on the actual nature of a design. For instance, the modelling accuracy of a structure consisting of shift registers with short wire interconnections may differ significantly from a LUT or RAM structure with long interconnections. A high wire load may require the synthesizer to use standard cells with increased transistor dimensions or duplicate a standard cell for parallel wiring in order do drive higher currents. High wire-load estimations also increase the RC charge time and slows down the signal propagation in the respective path.

As the Synopsys synthesizer software, is proprietary closed-source software the true reasons when the synthesizer may decide to change the wire-load model are obscure to the standard user. During synthetisation runs, we observed an interesting effect of decreasing area results for increased timing constraints. This is sort of atypical compared to normal synthesizer results. We attributed this to be caused by wire-load model switching by the synthesizer as the wire-load model was not explicitly set. This example demonstrates the potential variability of results obtained by a synthesizer. The algorithms used for synthetization are designed to detect structural patterns and perform according optimizations. If these algorithms do not recognize the implicit pattern in a behavioral circuit description, the synthesizer results may be sub-optimal. While automated synthesis helps manag-

ing large circuits, it does not guarantee an optimum solution. Especially, if circuits are described on a behavioral level many degrees of freedom in synthesis may lead to a local minimum instead of a global optimum solution regarding a cost function such as circuit size.

6.3 Back-End Design

In the following sections, we want to discuss the major steps required in the back-end part of a standard-cell based digital design flow.

We refer to *back-end design* as the transformation of a gate-level circuit description to a detailed chip-layout description which is ready for fabrication. As it is important to assure that the design is ready for fabrication (tape-out) and that the produced chip actually works as specified several verification steps are part of a typical back-end design flow. Prominent examples are design rule checks (DRC), electrical rule checks (ERC), and layout versus schematic (LVS) checks. The actual layout of a circuit, which contains placement and routing information, is verified by a so called *post-layout simulation*. A standard-cell back-end flow contains two important tasks. Namely finding an actual standard-cell placement on the chip die and the actual wire routing of standard-cell interconnections. Both of which need to support the timing and area specifications given for a design. The algorithms for placement and routing apply heuristic methods which generally are not determinisitic. So the placement algorithm may give different results based on initializations by pin mapping, core-area aspect ratio, macrocell placement, or certain optimization settings. Routing is basically initialized by preceding steps in back-end flow as well as optimization settings. Consequently variations of the initial setting may have significant effects on the final outcome of a back-end flow.

6.3.1 Floorplanning

Floorplanning is concerned with the coarse-grained organization of chip area. Basically, it contains the tasks of padframe configuration, placement of major circuit blocks such as macrocells, dimensioning the chip-level power distribution network, and the insertion of a so-called clock tree.

In the following, we want to shortly introduce the major steps of floorplanning. The first task of floorplanning usually is to set up a padframe. The padframe basically consists of input and output pads which provide an electrical interface between off-chip circuitry and on-chip circuits. The pads itself are provided as standard cells and allow configuration, as for instance, setting the output driving strength. Usually the padframe is a closed structure of rectangular shape where pad filler-cells are used to fill gaps between input/output pads. The pads are

powered by a pad power ring which is integrated to pad cells and powered by pad power cells. Assigning signals to the pads and placing the pads in the padframe determines the pin diagram of a chip (e.g., Figure C.2). In the following steps, a core area is defined where standard cells and macrocells may be placed. Between the padframe and the core area, a core power-ring is placed to distribute core supply rails of VCC and GND. The core power ring is powered by core power pads located in the padframe and may be extended by power stripes or additional power rings. At this stage, the floorplan is ready for standard-cell placement.

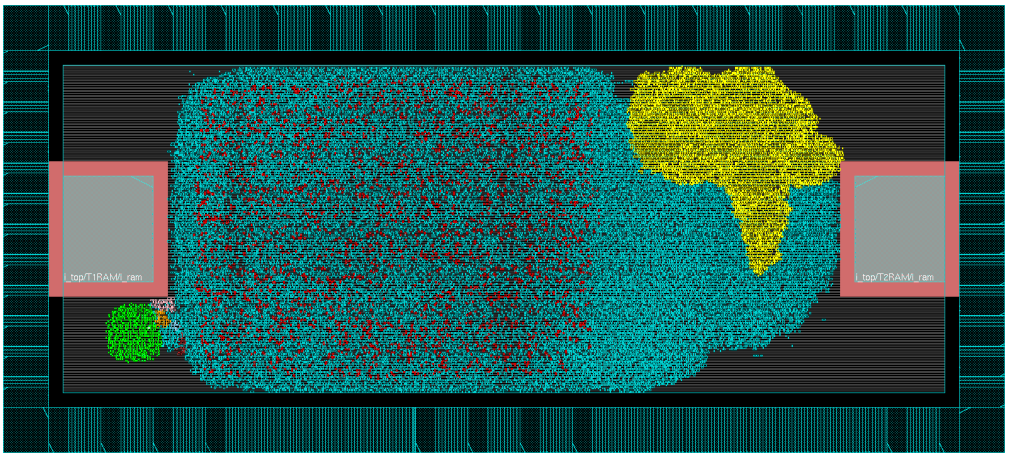


Figure 6.2: Example of a floorplan after macro cell and standard-cell placement. Major VHDL entities are highlighted, (Instruction memory in green, core multiplier in yellow, and result registers in red).

Macrocells are usually placed manually and should be oriented with respect to the location of their input/output signals. Placing large macro cells in corners has advantages as the external macrocell power ring can be combined with the core power ring to minimize the area spent on power distribution. Additional chip area can be saved by overlaying power rings on multiple metal layers. The top metal layer is usually thick compared to lower metal layers and hence suited for power distribution. A possible overlayed configuration in a six metal layer process could distribute the ground ned at layer six (vertical) and layer five (horizontal). The power supply could be placed on layer four (vertical) and layer three (horizontal). The adjacent layers of each net are connected with vias to close the rings. Overlaying power rings allows to save a considerable amount of chip area but requires more complicated routing of interconnections from core to pad cells as they may block signal routing on lower metal layers.

6.3.2 Standard-Cell Placement

In a semi-custom ASIC digital design flow, so-called *standard cells* are used as logic building blocks at the lowest abstraction level. Standard cells are usually supplied by a standard-cell library provider such as Faraday [15] which sells well-defined and quality-assured descriptions thereof. Standard-cell libraries are usually grouped into core cells and I/O cells. Examples of typical core cells are NAND gates (ND), NOR gates (NR), Inverters (INV), Buffer cells (BUF), Multiplexers (MUX), Tie-low/high cells (TIE0/1), and flip-flop cells. These standard cells are small macros sharing a common cell height allowing to place them in standard cell rows. The standard-cell rows are also used to provide power supplies to standard cells. A standard-cell supply line either connects to the core VCC or core GND net. To save chip area the VCC and GND lines are in interleaved order so standard cells connecting from above and below share a common supply connection.

To interface the design to off-chip circuitry, I/O cells are used. Typical examples of I/O cells are I/O-pad cells with various driving strengths, power supply pads for the pad power ring and core supply, pad corner cells, and pad filler-cells. The I/O-cell supply net (3.3 V) is separated from the core supply net (1.8 V). The I/O-cell supply is powered through pad power cells which connect to a pad power ring which is integrated to I/O-cells. The core cells are powered through the core power pads which connect to the core power ring usually located between the pad ring and the core area.

Standard cells can be placed after defining the core area which contains the standard-cell rows. The area as given by the synthesizer serves as a theoretical minimum to place the design. Practically more area is required as power supply nets, buffer cells, and additional signal routing area contribute to the total area consumption especially if the timing is constrained and cells of higher driving strength consuming more area are needed. To accommodate extra area requirements in later design-flow steps, a certain amount of free area is budgeted when placing the design. The ratio between yet unoccupied area and occupied area by standard cells is expressed as placement density. A high density in the placement step makes it more difficult to reach a design without violations in later steps. For an average design starting with a density of about 70 % should leave enough area for subsequent routing, timing optimizations, and re-routing without requiring lengthy optimization iterations.

6.3.3 Signal Routing

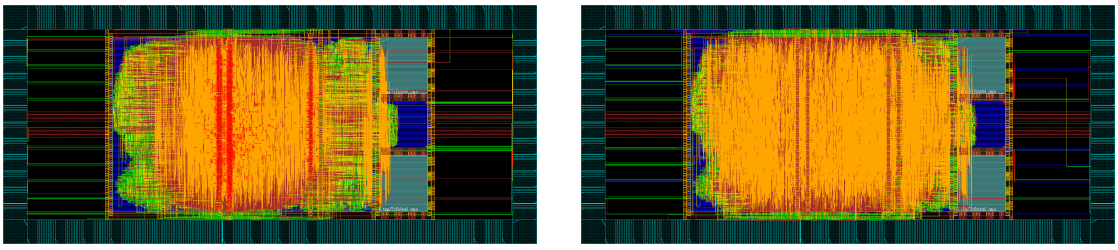
Prior to detailed signal routing, a so-called *trial routing* algorithm is used as coarse estimation to extract timing information. The trial routing algorithm is a fast approximation for actual routing. It partitions the design in blocks and investigates

if wiring between these blocks is possible. Congestion warnings indicate that a block has consumed a certain amount of virtual wire connections to other blocks and actual routing may be complicated. Congestion usually occurs in densely placed regions with accordingly high wire interconnections. If a power stripe limits the number of layers available for wire routing congestion warnings are likely to occur. Figure 6.3 illustrates congestion warnings in the vicinity of the vertical power stripes in the trial routed layout where congestion warnings are indicated in red. Especially the power stripes on the left exhibit many congestion warnings. The actual routing algorithm attempts to resolve congestion and does a *detailed routing* which generally increases wire density. The detailed routing causes an increased wiring visible on top-metal layer usage (vertically oriented wires in yellow represent top-metal layer 6). The effects due to detailed routing is clearly visible in the two layout stages given in Figure 6.3.

6.3.4 Post-Layout Simulation

Layout comprises the steps to transform a discrete schematic description to a physical circuit description. Cell placement and wire routing fixes the physical characteristics of a circuit. Regarding the physical characteristics, a designer is usually interested in the amount of parasitic capacitances. If a wire net is charged with too much capacitive load, signal transitions get malformed and can violate timing constraints of setup and hold times. To verify that the circuit is still operational under the influences of parasitic effects, a post-layout simulation is applied. A post-layout simulation is based on a circuit description which has been back-annotated with information on parasitics which were introduced in the layout step. The so-called *back-annotated* circuit is simulated by a simulator tool[3] using a testbench similar to the circuit simulation at the RTL level. Adjusting the testbench

[3]In this work Modelsim SE 6.4 by Mentor Graphics was used for post-layout simulations



(a) Trial Routed (b) Routed

Figure 6.3: Signal-routing example

clock timing (ATI timing) allows to confirm that the circuit is operational at a given clock frequency.

The described post-layout simulation setup can be extended to extract node-toggling activity informations to be passed to a power-analysis tool for power analysis. A short discussion on power analysis is given in the following chapter.

6.3.5 Power Analysis

In the following, we want to focus on power-rail analysis, current density figures, and power simulation all of which are based on stimuli-based post-layout simulations.

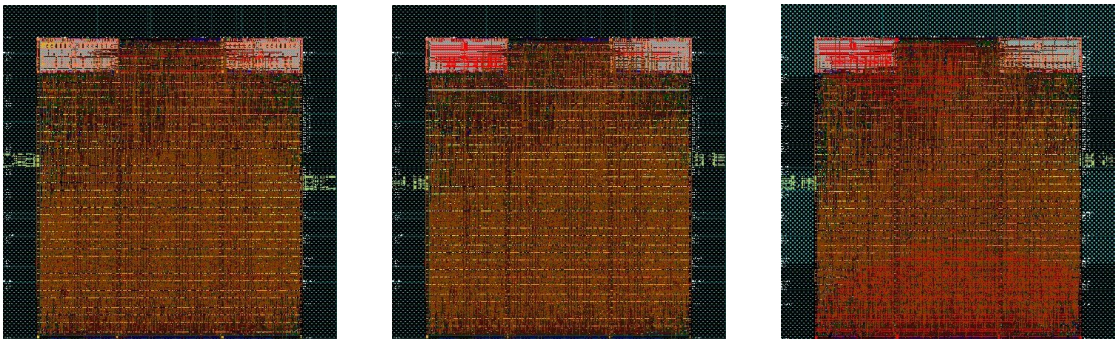
Rail analysis is used to confirm that a circuit provides good power distribution on its power rails of VCC and GND. This is vital for the correct functioning of a circuit as the specified minimum or maximum supply specifications of the standard cells must not be exceeded. Current density figures are important as excessive current densities have negative effects on the expected lifetime of a semiconductor circuit. Basically power simulation approximates a circuit's power consumption using node-toggling activities which were obtained by post-layout simulation of a back-annotated circuit.

Power-Rail Analysis

A power-rail analysis provides the basis to ensure that standard cells are operated within their specified supply ranges. Due to resistance of the conducting material used for power distribution,a certain voltage drops across power-supply lines. This voltage drop is called *IR-drop* as it is proportional to current I and resistance R of the supply line. Reliable circuit design requires that these transient changes of power and ground-supply net remain within tolerable limits in order to guarantee that each cell is provided with its minimum supply voltage. To verify that the power-distribution network is dimensioned well and provides connections of sufficiently low resistance, a power-rail analysis is performed for the supply nets of VCC and GND. A power analysis uses a certain node activity for simulation. One way is using a global value of average node activity which is not very accurate and just a rough estimation. Alternatively, we may use stimuli-based node activities contained in a VCD file[4] as can be obtained by post-layout simulation.

For stimuli-based power analysis, we use Cadence SoC Encounter and VCD files by post-layout simulation runs in Modelsim by MentorGraphics. The applied 180 nm CMOS technology has a nominal supply voltage of 1.8 Volt (VCC) referred to ground (GND). To obtain threshold values for the ground and supply net, we use

[4]Value-Change Dump, describes the logic level of each node for a certain simulated time



(a) IR drop GND (b) IR drop VCC (c) IR drop VCC (mult)

Figure 6.4: Power-rail analyses of Geminicore

the minimum supply voltage as specified in the standard-cell supply specifications. We then divide this value by two and use the result voltage drop margin for ground and supply rail respectively. As a result, we apply a maximum ground-rail voltage of 0.09 V and a minimum supply-rail voltage of 1.71 V. The applied power-analysis tool provides a graphical illustration of critical areas on the chip suffering from insufficient power distribution. Figure 6.4 shows three examples, where the left figure shows analysis of the ground net, the middle an analysis of the supply net, and the right figure an analysis of the supply net during multiplication. IR drops are indicated by a green to red colored gradient. The core-power pads located at the middle left and right of the padframe exhibit only moderate IR drops. The illustration on the right shows IR drop during active calculation where the bottom area exhibits significant IR drop.

Based on the results on a power-rail analysis, a chip designer may extend or reduce the power-distribution network in order to make the design more reliable or efficient.

Current Density

Power analysis provides simulated current-flow values and hence power-density values. Consideration of current density $J = \frac{I}{A}$ is important to conclude on thermal power dissipation and electromigration effects both being proportional to current density. Excessive current densities may cause conducting material to melt and as a consequence destroy the circuit.

High current density also has another more subtle effect which is also detrimental to the correct functioning of a chip. Electromigration is the physical movement of atoms due to a high amount of momentum caused by flowing electrons. This atomic movement may cause gaps and pile ups in conducting material to an ex-

tent potentially destroying the circuit due to short and open wire connections. Consequently, electromigration effectively reduces the lifetime of a chip. Based on fabrication specific process properties such as wire metal type (e.g., silver, copper, or aluminium) and its conductivity as well as metal-layer thickness, a maximum current density J_{max} is defined for metal interconnections by the manufacturer. Hence maximum current density values given in $\frac{mA}{\mu m^2}$ are used as design guidelines for a certain temperature. To avoid the undesired effects caused by high current densities, a chip designer tries to keep current densities sufficiently low ($J/J_{max} < 1$).

Actual results during back-end designs showed power-density violations in connections of VCC-supply pads to VCC core power rings. These violations were caused by near-by placed macro cells which over-strained the supply pad. Placing the macro cells more distant to this pad and closer to other VCC-supply pads helped to reduce the current density considerably. Observing that power-supply pads have metal connects at several metal layers, a designer could apply multiple power ring to power-pad connections using stacked or staggered vias next to the power pad or even use additional power rings on other metal layers. Upper metal layers should be preferred for distributing power as they are generally thicker and have higher maximum current density specification. Of course spending another power pad in the vicinity of an over-strained pad would also represent a viable solution in case an additional supply pad is allocatable.

Layout area attributed to power distribution is clearly a main contributor to chip size. Where the distribution network usually contains the elements such as core power rings, power stripes, and macrocell power rings. In the light of area attributed to power distribution, chip-size estimations solely based on the number of gates lack important information. The amount of area required for power-distribution varies with design as timing and power consumption (node activities) are design-specific.

Power Simulation

Power consumption basically consists of static and dynamic power consumption. As static power is negligible in CMOS circuits, dynamic power is significant. Dynamic power is due to the internal changes in standard cells and power due to the capacitive load a standard-cell output has to drive in order to change the state of its output. A basic model of dynamic power consumption is proportional to the internal and parasitic capacitance, the number of node changes per time (α), and

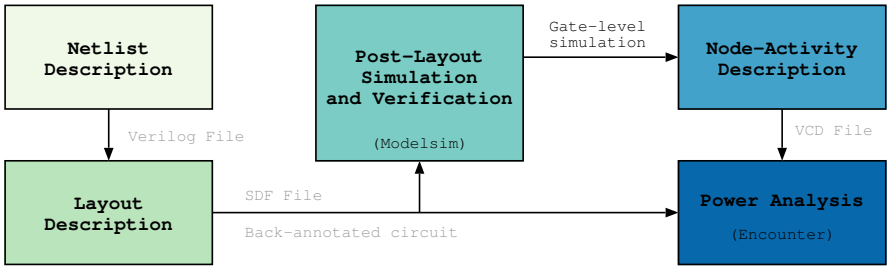


Figure 6.5: Power-analysis flow

the powered supply voltage (Equation 6.1).

$$\begin{aligned} P &= P_{dyn} + P_{stat} \\ P_{dyn} &= P_{internal} + P_{cap} \\ &= (C_{internal} + C_{load}) \cdot V_{CC}^2 \cdot f \cdot \alpha \end{aligned} \tag{6.1}$$

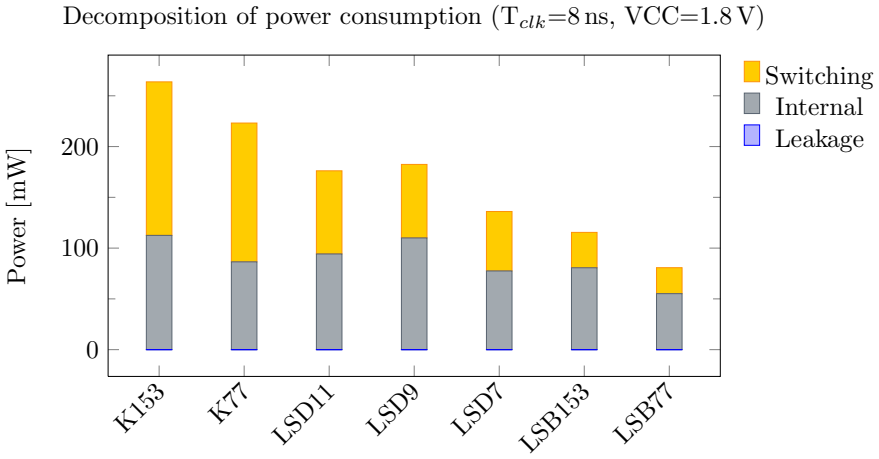
Table 6.1: Power dissipation obtained by stimuli-based power analysis[a] in mW (results from unconstrained area layout with f_{clk}=125 MHz)

Arch	Total	Internal Power	Switching Power	Leakage Power
K153	263.7	112.50	151.20	0.0114
K77	223.1	86.45	136.70	0.0070
LSD11	176.1	94.22	81.89	0.0060
LSD9	182.4	110.00	72.40	0.0090
LSD7	136.0	77.49	58.56	0.0070
LSB153	105.8	76.91	28.90	0.0084
LSB77	80.7	51.14	25.57	0.0048

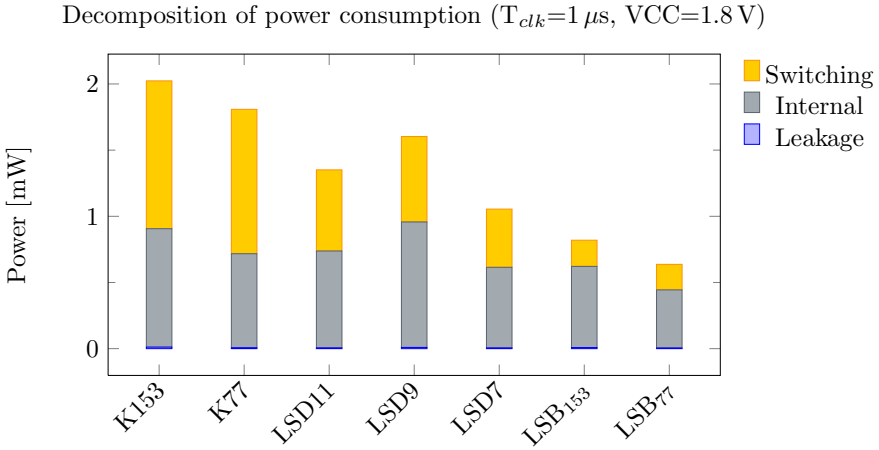
[a] Extracting the electrical activity for a full pairing calculation would require several hundred gigabytes of disk storage for each design. Hence the given analysis is based on circuit activities within the first 30 Miller iterations. As these iterations are highly regular and represent over 80 % of the total computation we regard them as representative and use them in place of an exhaustive power analysis.

Active pad cells and especially output pad cells consume a lot of power. To evaluate the power consumption of the core design, we limit power analysis to the actual computation period and ignore the time for input and output handshaking

where input/output pads are active. As the target application of this design is integration to a system-on-chip or to a crypto-processor module where no such I/O pads are required this approach provides more meaningful power figures. It



(a) 125 MHz



(b) 1 MHz

Figure 6.6: Decomposition of power consumption

should be noted that the power-consumption figures are obtained by averaging the simulated power consumption over a certain period of circuit simulation. To get reasonable results, we first extract the toggle activities of the nodes contained in a circuit based on actual stimuli provided by a pairing calculation. The toggle activities are then stored in a so-called value-change dump (VCD) file. After layout we extract the parasitics for back-annotating the circuit. Simulating the back-annotated circuit using representative stimuli, allows to determine node activities

in the circuit, which are then used to determine the circuit's power consumption in a power-analysis tool (see Figure 6.5). The results provide figures for static and dynamic power consumption as well as leakage power (Table 6.1).

$$P = f_{cp} \cdot E_{cp}, \quad E_{cp} = P_{avg} \cdot T_{cp} \tag{6.2}$$

Power analysis was simulated at 125 MHz and 1 MHz allowing to extrapolate power

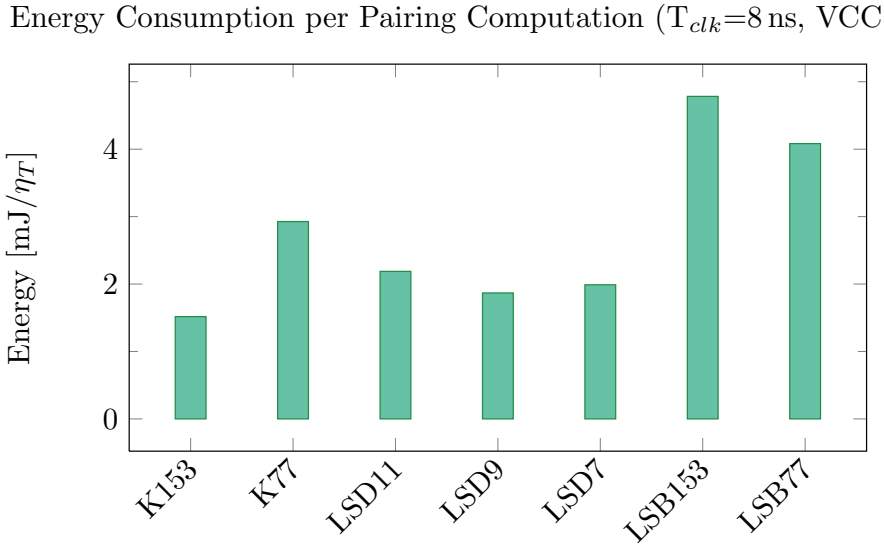


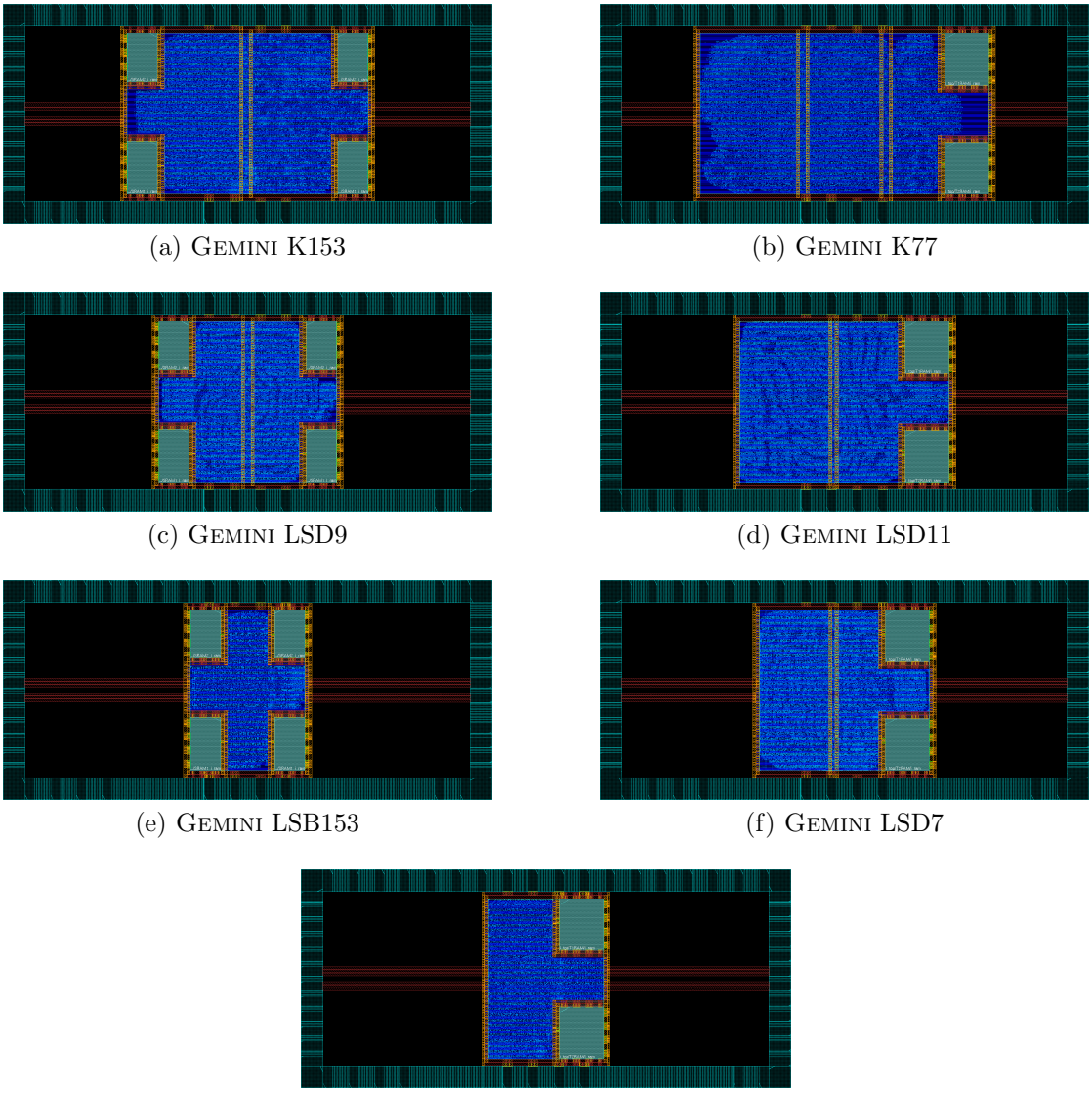
Figure 6.7: Energy consumption per pairing computation

consumption for other clock frequencies. Leakage-power results for the given designs are between 0.0049 mW for GeminiLSB77 and 0.0113 mW for GeminiK153. Compared to the power consumption due to internal and switching power, the amount of leakage power is negligible. Leakage effects are independent of operational frequency so that we find almost identical values for the two simulation runs with 125 MHz and 1 MHz clock speed respectively.

Another way to evaluate a design is considering *energy per computation* E_{cp}. The relation of power to energy per computation is given in Equation 6.2, where T_{cp} is the computation time. Figure 6.7 illustrates the energy consumption of the implemented designs per pairing computation. The long computation time of the bit-serial designs make them consume more energy compared to the other designs making them less efficient in terms of energy consumption. Fast designs such as GeminiK153 benefit from the short computation time exhibiting the lowest energy consumption of all implemented designs.

6.3.6 Layout Results

The actual layout was performed using Cadence SoC Encounter in several back-end iterations. Snapshots of the post-placement and post-routing stage of the last iteration is given in Figure 6.8 and Figure 6.9 respectively. The given layouts were obtained by constraining the effective core area from both sides. The core area of these designs was successivly reduced up to a point were the results started to exhibit design violations. Further optimizing the designs by more back-end iterations is certainly possible but is likely to get excessively time consuming and complex as the optimisation problem for the algorithms get harder and occuring violations need to be fixed.



(a) Gemini K153

(b) Gemini K77

(c) Gemini LSD9

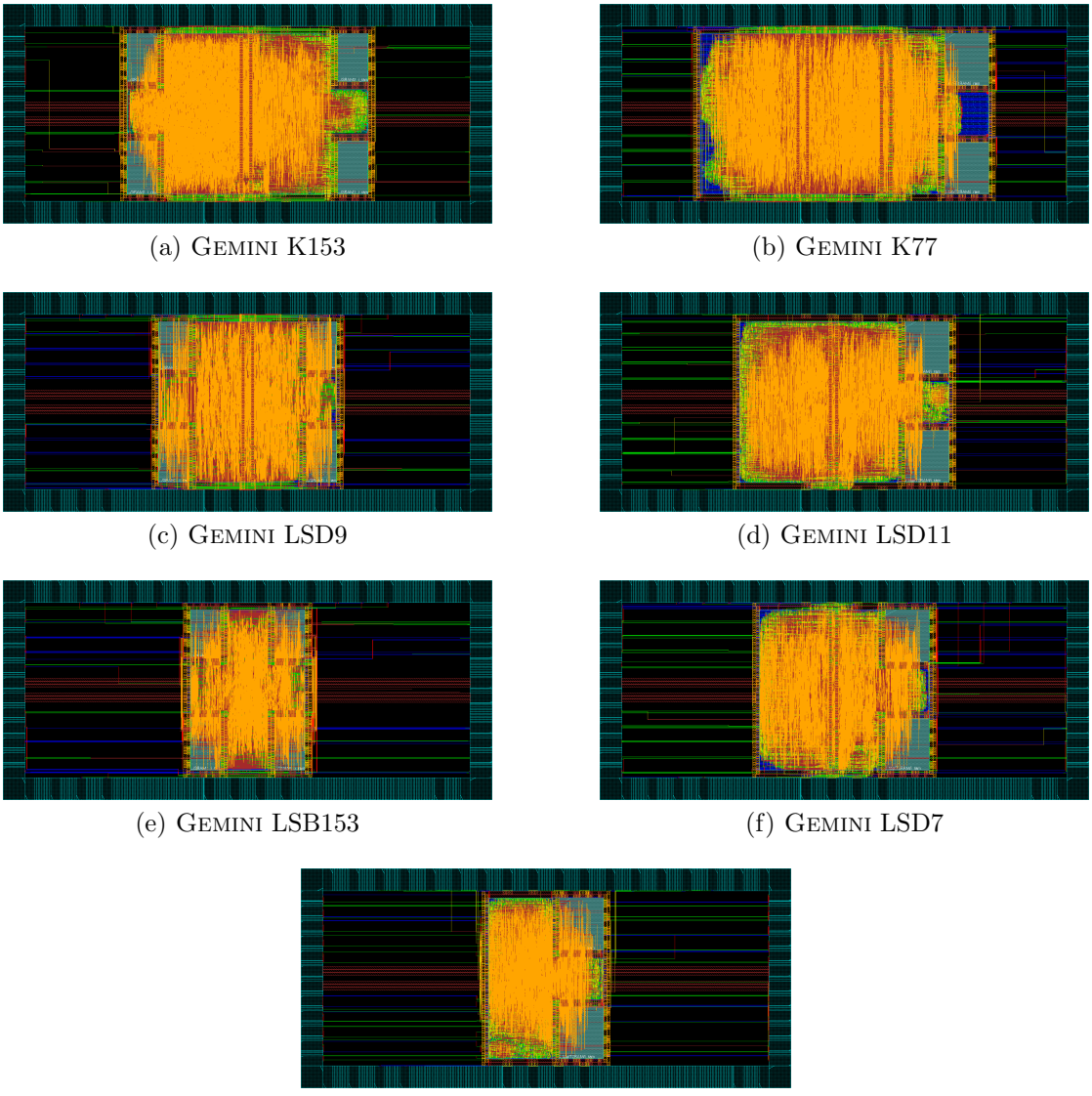
(d) Gemini LSD11

(e) Gemini LSB153

(f) Gemini LSD7

(g) Gemini LSB77

Figure 6.8: Standard-cell placement in back-end design flow



(a) Gemini K153

(b) Gemini K77

(c) Gemini LSD9

(d) Gemini LSD11

(e) Gemini LSB153

(f) Gemini LSD7

(g) Gemini LSB77

Figure 6.9: Routing results in back-end design flow

Chapter 7

Results

In this chapter, we present the final results of the hardware designs described in Chapter 5. The results are based on an ASIC-design flow using 180 nm CMOS UMC technology and Faraday Technology standard cell libraries. For each design,we provide results according to area, computation time, power, and energy. Finally, we conclude the chapter with an evaluation of the final results providing a combined cost metric.

7.1 Computation Time

The main goal of this thesis was to design an ASIC-hardware architecture to calculate a bilinear pairing. The primary design goal was to realize a design with low area requirements. Secondary objectives were low computation time and low power consumption. Design decisions were made so that the full pairing computation time does not exceed 400 ms to be applicable for systems involving human interaction. The results presented in Table 7.1 demonstrate that the designs remain well below this design specification. As the input/output interfaces are small compared to the amount of input and output data, a certain time is required to write input and read results. Even though input/output interaction is handshaked, which requires additional clock cycles, the amount of time attributed to input/output is negligible compared to the actual pairing computation time.

7.2 Synthesis Results

Figure 7.1 and Figure 7.2 show the results of the synthesis runs performed for the implemented designs. Each dot in these figures represents a synthesis and a corresponding circuit result which is operational up to the given clock speed. At some point the circuit's longest propagation delay exceeds the given clock constraint.

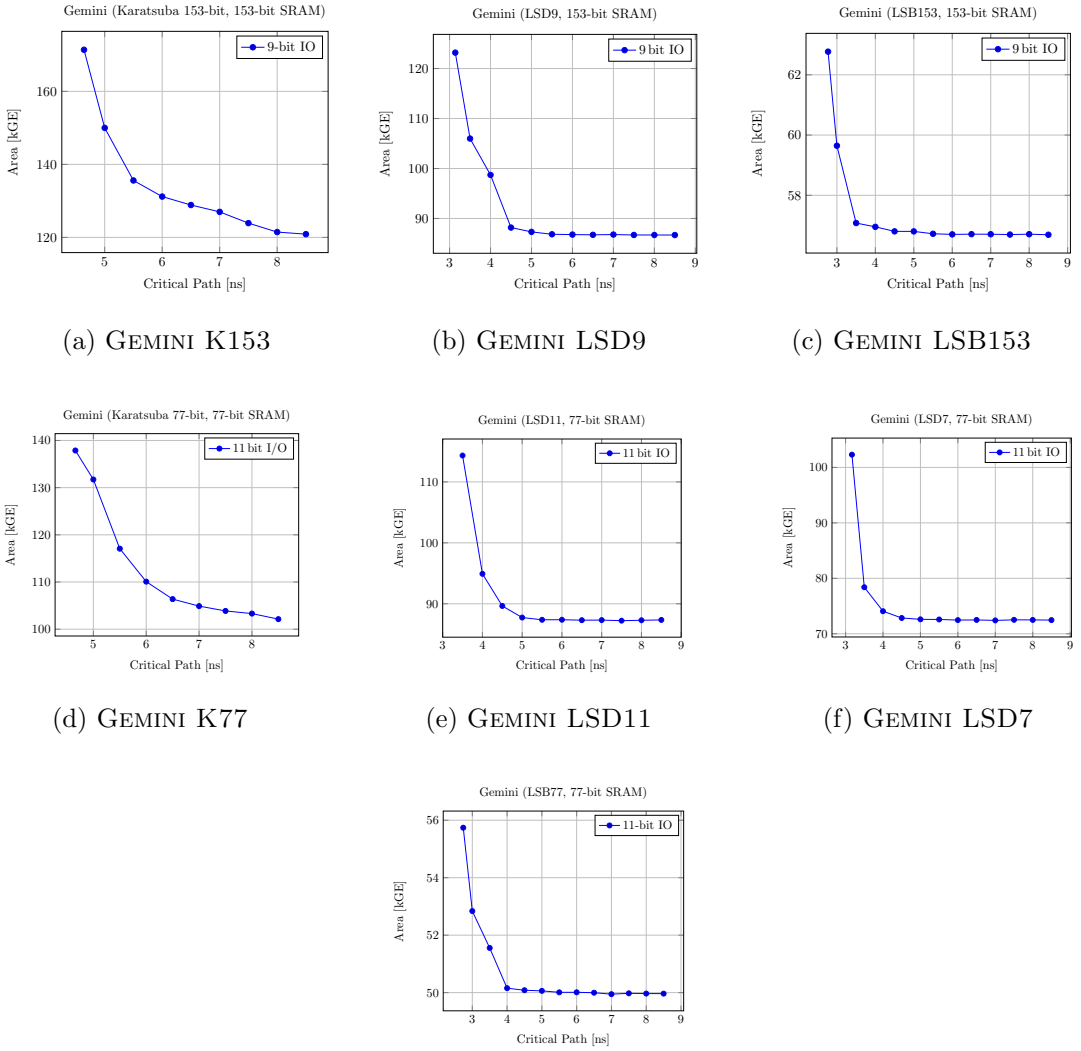
Table 7.1: Pairing computation time

Arch	w/o handshaking		with handshaking	
	cycles	ms†	cycles	ms†
K153	752 929	6.02	756 739	6.05
K77	1 612 620	12.90	1 615 758	12.93
LSD11	1 599 657	12.80	1 602 795	12.82
LSD9	1 202 313	9.62	1 206 123	9.65
LSD7	1 876 201	15.01	1 879 339	15.03
LSB153	5 899 240	47.19	5 903 050	47.22
LSB77	6 434 856	51.48	6 437 994	51.50

† T_{η_T} for $T_{clk} = 8\,ns$

In order to comply with the given clock constraint, the synthesizer begins to optimize the circuit which in turn increases the circuit size, because driving strengths of standard cells need to be increased. With the presented synthesis runs it was possible to explore the circuit specific relation of clock constraint to circuit size and adapt to clock timings providing a low-area circuit. Evaluating several designs requires to define a basis upon which a fair evaluation is possible. To provide a fair basis, we synthesized the circuits under a constant clock constraint, where they do not exhibit steep area increase due to a demanding clock constraint. Based on the synthesis results illustrated in Figure 7.2, a clock period of about 7 to 8 ns was selected as basis for subsequent back-end design flows. To have some additional timing budget in the back-end flow a stricter clock constraint was used for circuit synthesis. This allows to reach the desired clock speed for all the final post-layout designs with a high confidence that no extra timing optimizations are required which tend to make a back-end flow iteration more difficult. As a consequence, all the architectures arrive at a common post-layout clock period which helps to provide a fair basis for evaluation of the implemented designs.

The synthesis results collected in Figure 7.2 show that the implemented designs can be split in three groups based on their circuit size. The Karatsuba-based pairing architectures (K153 and K77), the digit-serial based architectures (LSD11, LSD9, and LSD7), and the bit-serial architectures (LSB153 and LSB77). Interestingly, the digit-serial architecture with digit size eleven (LSD11) shows practically the same area to clock constraint relation as the digit-serial architecture of digit size nine (LSD9). The reason of this area equivalence is that these designs do not share the same datapath width and contain different macrocell-memory configu-



(a) Gemini K153 (b) Gemini LSD9 (c) Gemini LSB153

(d) Gemini K77 (e) Gemini LSD11 (f) Gemini LSD7

(g) Gemini LSB77

Figure 7.1: Synthesis results of pairing architectures

rations. In the given case, the area overhead introduced by the macrocell configuration of design LSD9 cancels out the smaller multiplier size of LSD9 regarding to LSD11. Another interesting fact is that the Karatsuba-based architectures tend to ramp up earlier in circuit size then the digit-serial based architectures. The low combinatorial depth in the digit-serial multipliers allow faster clocking.

Hardware design is generally subject to many trade-off decisions where diametrical or opposing characteristics need to be balanced to obtain a satisfying solution.

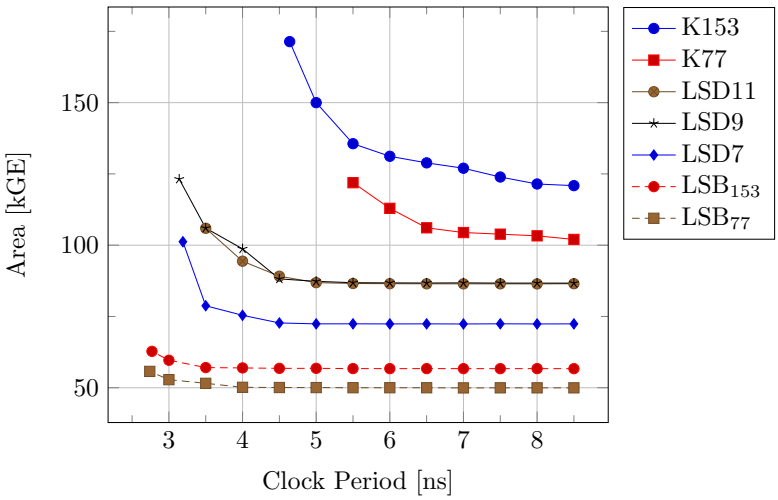


Figure 7.2: Synthesis results of pairing architectures over clock period

These trade-off situations allow to apply architecture transformations in order to comply with design goals. Usually area and computation time are opposing design goals where it is considered optimal to minimize both of them. The relation between area and computation time is customarily illustrated by a so-called AT

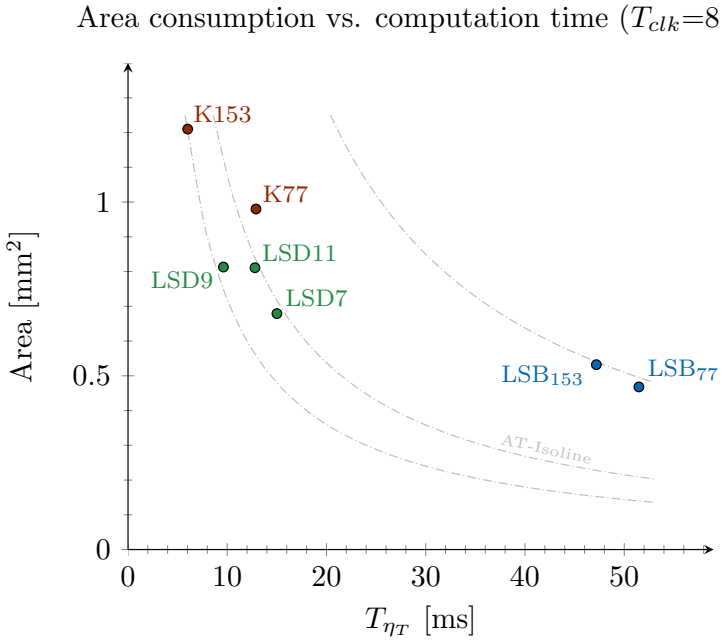


Figure 7.3: Synthesis results over η_T computation time

plot (Figure 7.3) where the area-time product is plotted. In an ideal case, a design transformation allows to move the AT value of a design on an AT isoline where the product of area and time is constant. The illustrated AT plot is based on an 8 ns clocking of each design to provide a fair comparison.

7.3 Area Requirements

Table 7.2 gives chip area results for the design flow stages of synthetization, gate-count based layout, and effective chip area. Synthesis-based area figures represent a theoretical minimum for actual chip area. Area results at the synthesis level are based on the set of standard cells contained in a synthesized netlist. This area figure merely represents the cumulative area of cells contained in the netlist. In an actual back-end flow, standard cells are not placed edge-to-edge to maintain an area margin for subsequent back-end tasks which need area to insert buffers, change placement of standard cells, and to route signal wires for standard-cell interconnection. Gate-count based area figures were obtained in back-end design

Table 7.2: Area consumption

Arch	Synthesis based		Gate-count based		Effective Area	
	mm^2	kGE	mm^2	kGE	mm^2	kGE
K153	1.208	128.9	1.243	132.6	1.77	188.8
K77	0.983	104.9	1.030	109.8	2.10	223.9
LSD11	0.811	86.5	0.848	90.4	1.55	165.3
LSD9	0.813	86.8	0.848	90.4	1.36	144.7
LSD7	0.679	72.4	0.708	75.6	1.29	137.6
LSB153	0.532	56.7	0.560	59.8	0.96	102.2
LSB77	0.468	49.9	0.494	52.7	0.90	95.9

and are still a cumulative sum of standard-cell area contained in the design. The gate-count was extracted post-layout and additionally contains area contributions of buffers and area of standard cells with increased drive strength in order to satisfy given timing constraints. Still this figure does not consider factors contributing to effective chip-area consumption such as area between standard cells, area required to route signal, or area attributed to power distribution such as power rings. The effective area figure in Table 7.2 considers all of the actual area contributors in the design—such as area for power distribution networks (core and macrocell power

rings) and in general area required for signal routing which was not considered in the previous figures. Of course the results given as effective area are subject to change if more time is invested in back-end design iterations.

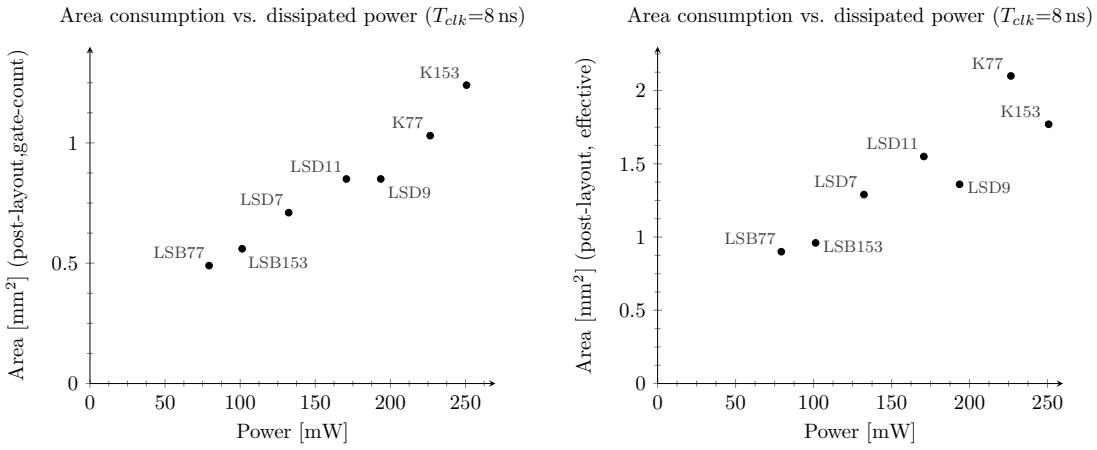
7.4 Power and Energy Consumption

Besides chip area, power and energy consumption figures directly influence the applicability of a design to be used in resource-constrained environments. Power consumption is relevant for integration in smart cards or RFID-enabled devices which can only provide a certain amount of power. Energy consumption is to be considered if integration to a battery-powered embedded device is desired. Basically, it is more sensible to use the energy figure to evaluate a circuit as it combines the power consumption and computation time.

Table 7.3: Power and energy consumption

Arch	Average power consumption		Energy per computation
	mW @ 125 MHz	mW @ 1 MHz	mJ/η_T
K153	263.7	2.023	1.52
K77	223.1	1.808	2.93
LSD11	176.1	1.351	2.19
LSD9	182.4	1.603	1.87
LSD7	136.0	1.054	1.99
LSB153	105.8	0.819	4.78
LSB77	80.7	0.637	4.08

Table 7.3 holds average power and energy figures of the implemented architectures. To allow extrapolation to other clock periods, two power simulation runs were performed giving results for $f_{clk} = 125\,\text{MHz}$ and $f_{clk} = 1\,\text{MHz}$. As the circuits are powered with 1.8 V, we can calculate the average current flow. In the fast clocking scenario, architecture K153 and LSB77 draw about 145 mA and 45 mA respectively. If operated at 1 MHz, the K153 architecture draws 1.12 mA and the bit-serial LSB77 architecture 0.35 mA. Figure 7.4 gives the relation of average power consumption to circuit size considering area based on gate-count figures and on effective chip area. The results in Figure 7.4a indicate the linear relation between the number of gates and their power consumption while the post-layout



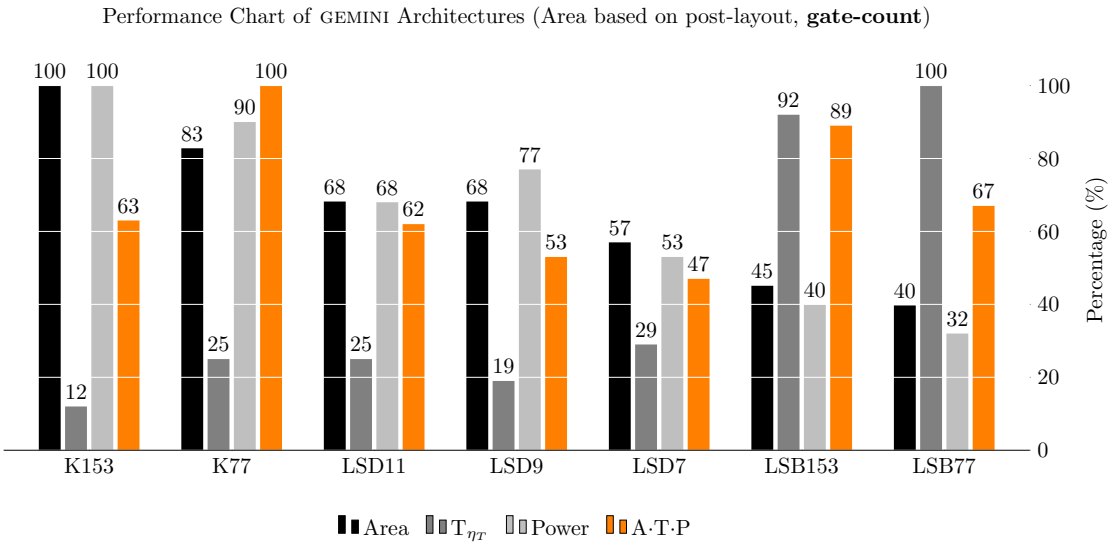
(a) Gate-count based (b) Effective area

Figure 7.4: Area over power consumption

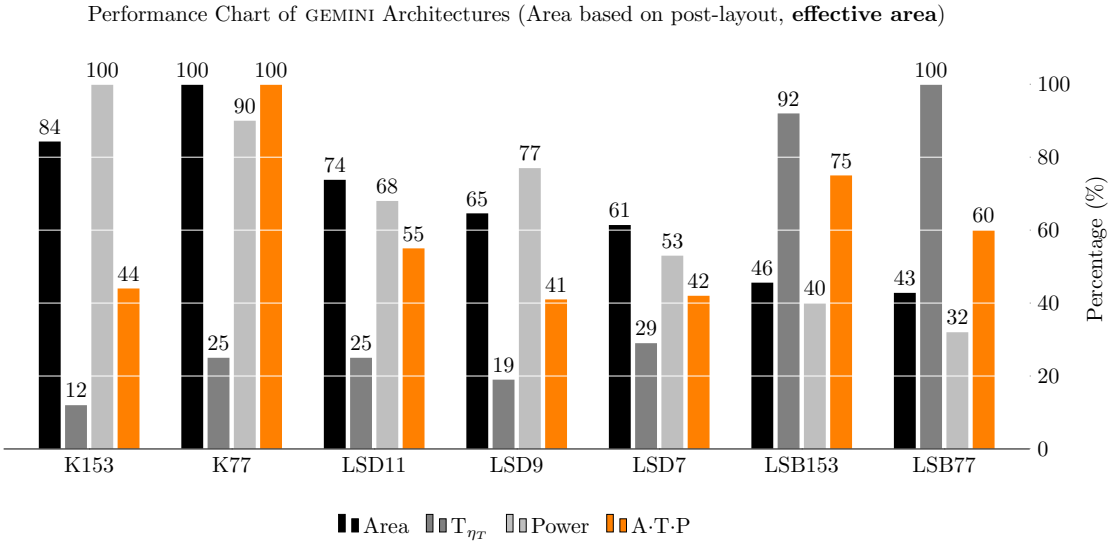
results used in Figure 7.4b deviate from this relation due to parasitic effects induced through actual layout.

7.5 Evaluation

To evaluate the given designs, we want to consider their performance according to all of the demanded design goals. Hence, we normalize the results obtained for area consumption, power and time consumption to define a combined cost metric. In the following, we present an evaluation based on a combined cost metric A·T·P ($area \times time \times power$). As the product of power and time corresponds to the cost in terms of energy, we may also regard it as area-energy cost metric. As this thesis provides area figures based on gate-count and effective chip area, we therefore present also combined cost results for each of them. The gate-count based results are illustrated in Figure 7.5a and those based on effective chip area in Figure 7.5b. While the A·T·P figure is quite similar for both value sets, it shows that the architecture K153 turns out to be favored over the K77 design according to the actual back-end area result. The bit-serial designs suffer from long computation times corresponding to a high demand of energy making them less attractive. Based on the overall cost metric, the designs K153, LSD9, and LSD7 provide a good balance between area and energy consumption. Furthermore, imposing the prime design goal of low area, the designs LSD9 and LSD7 are to be preferred as efficient hardware architectures.



(a) Gate-count based area



(b) Effective chip area

Figure 7.5: Comprehensive comparison of implemented architectures based on normalized values of area, η_T pairing computation time, and power consumption. All designs are constrained with $T_{clk} = 8\,\text{ns}$ for a fair comparison. The product of the normalized values is used as overall cost metric. A low A·T·P value signifies low cost in terms of the considered design goals.

Chapter 8

Previous Work

In [5], Barreto et al. present general techniques for the efficient computation of pairings on supersingular Abelian varieties and provide generalizations on the results by Duursma and Lee in [16] for computing the Tate pairing on supersingular elliptic curves over characteristic three fields. Based on these generalizations they define the so-called eta or η pairing which is non-degenerate and bilinear. They further improve this pairing and denote the improved version as η_T pairing which outperforms their η approach by a factor of two according to their experimental results [5].

Beuchat et al. propose several improvements of the η_T pairing algorithm in characteristic two and characteristic three in [8]. They provide a square-root and cube-root free version of the η_T pairing, for characteristic two and three fields respectively. Furthermore their results improve the final exponentiation step by simplifying the contained field-inversion operation.

Ghosh et al. presented the first implementation of the η_T pairing at 128-bit using supersingular curve in characteristic two [21]. They applied a Karatsuba-based multiplier approach to calculate the η_T pairing on an FPGA platform. Applying versions of one and two Karatsuba-steps they compute a base-field multiplication with a serial use of 612 and 306-bit parallel core multipliers respectively. With the latter approach, they compute a base-field multiplication with nine 306-bit multiplications. Their design takes 15 167 slices and 54 681 LUTs on a Virtex 6 FPGA and computes a pairing within 190 μs.

The recently presented results of Adikari et al. [1] propose a cryptoprocessor for computing the η_T pairing on supersingular elliptic curves over $\mathbb{F}_{2^{1223}}$. To perform the field multiplications in $\mathbb{F}_{2^{1223}}$ they apply a Toeplitz-matrix vector-product based approach and also give results for a Karastuba multiplication splitting segments in two and three parts. They report an area consumption of 716 281 μm^2 and a calculation time for their two-way Karatsuba architecture with 80.6 μsec. Their

results of an ASIC-hardware architecture are based on best-case corner analysis and consume about 500 000 GE being synthesized using a 65 nm TSMC library.

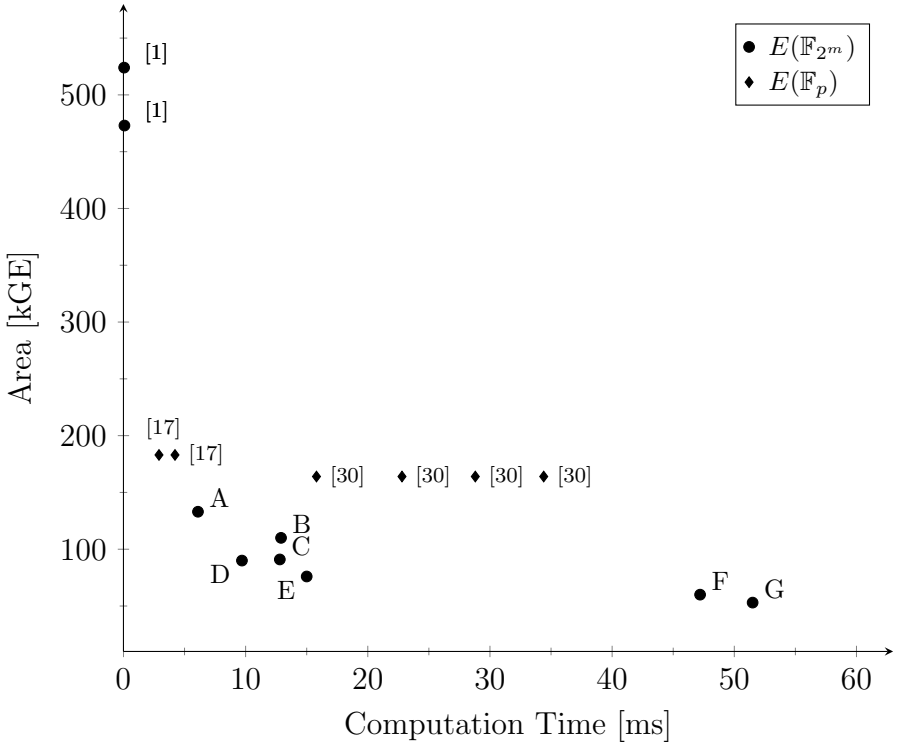
8.1 Hardware Accelerators

To give an overview on previous and concurrent work, Table 8.1 and Figure 8.1 provide a listing of ASIC implementations of bilinear pairings at a security level of 128-bit. The designs presented in this work are listed with area based on gate counts to provide a fair comparison. The figures given in Figure 8.1 provide AT plot illustrations where Figure 8.1a gives the evaluated timing at 8 ns. Figure 8.1b illustrates an AT-plot where the designs presented in this work are clocked with high speed while keeping a low-area circuit size (see Figure 7.2). For this plot, the following clock periods were applied: K77 at 7 ns, LSD11/9 at 4.5 ns, LSD7 at 4 ns, and LSB153/77 at 3 ns.

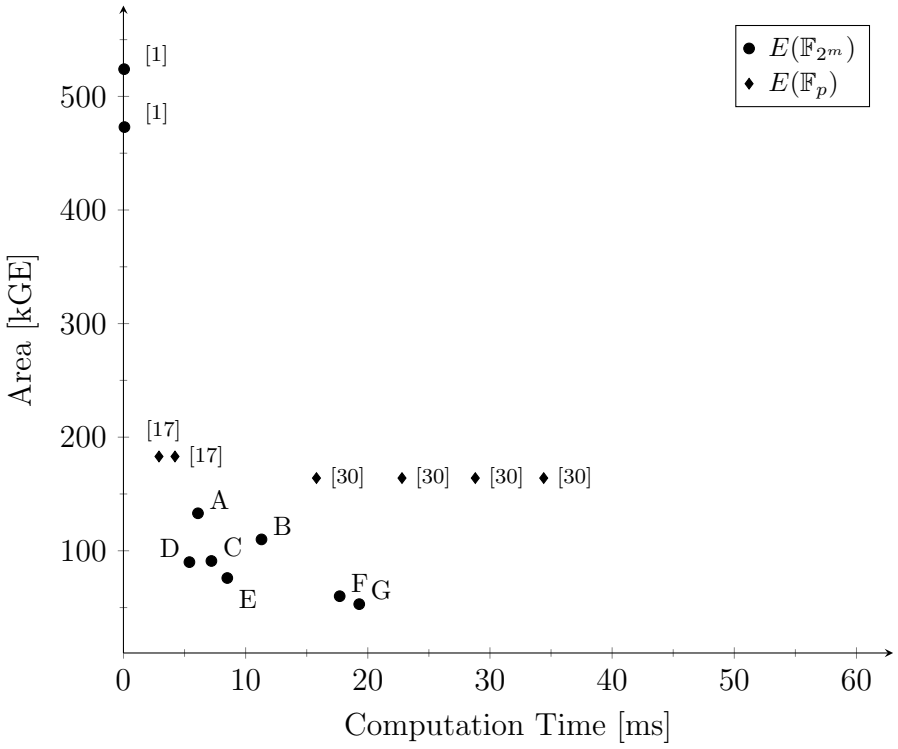
Table 8.1: ASIC-hardware implementations of pairings at 128-bit security level

	Curve	**Pairing**	**Tech.** nm	**Freq.** MHz	**Area** kGE	**Time** ms	**AT**[†]
Fan2009 [17]	$E(\mathbb{F}_p)$	ate	130	204	183[b]	4.22	772
Fan2009 [17]	$E(\mathbb{F}_p)$	R-ate	130	204	183	2.91	533
Kammler2009 [30]	$E(\mathbb{F}_p)$	opt-ate	130	338	164[c]	15.8	2591
Kammler2009 [30]	$E(\mathbb{F}_p)$	ate	130	338	164	22.8	3739
Kammler2009 [30]	$E(\mathbb{F}_p)$	η	130	338	164	28.8	4723
Kammler2009 [30]	$E(\mathbb{F}_p)$	Tate	130	338	164	34.4	5642
Adikari2012 [1][f]	$E(\mathbb{F}_{2^{1223}})$	η_T	65	500	473	0.08	38
Adikari2012 [1][f]	$E(\mathbb{F}_{2^{1223}})$	η_T	65	500	524	0.06	29
(A) **K153**	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	133	6.1	798
(B) **K77**	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	110	12.9	1430
(C) **LSD11**	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	91	12.8	1183
(D) **LSD9**	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	90	9.7	1350
(E) **LSD7**	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	76	15.0	1140
(F) $\mathbf{LSB}_{153}$	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	60	47.2	3111
(G) $\mathbf{LSB}_{77}$	$E(\mathbb{F}_{2^{1223}})$	η_T	180	125	53	51.5	2703

[†] Area-Time product (kGE × ms)
[b] Including 70 kGates for Register File and 25 kGates for controller and ROM.
[c] Includes 97 kGates core area and 67 kGates for data memories.
[d] 124-bit security level
[f] Results based on best-case corner analysis



(a) f_{clk}=8 ns



(b) Applying minimum clock period while maintaining low-area circuit size

Figure 8.1: ASIC hardware implementations of pairings at 128-bit security level

Chapter 9

Conclusion

The computation of bilinear pairings is the central and computationally dominating part in a pairing-based cryptographic scheme. To benefit from the new set of cryptographic schemes introduced by bilinear pairings, implementations applicable in embedded devices such as mobile phones or smart cards are desired. Hence, efficient computation of pairings is of utmost importance for practical realizations of pairing-based schemes to be applicable in resource-constrained environments. The presented hardware architectures were designed to consume low chip area, as this is a crucial figure for high-volume ASIC chip production and large scale deployment. The implemented designs consume as little as 50 kGE and are able to compute a bilinear pairing within several milliseconds. Based on the hardware results presented in this thesis, we can state that computation of cryptographic bilinear pairings is possible with low chip area and within reasonably short computation time. With these results, further integration to build a holistic and applicable cryptosystem based on bilinear pairings is clearly feasible with respect to implementation cost. In general, the presented results demonstrate that pairing-based cryptography is ready for resource-constrained devices and encourages further research in this direction.

9.1 Final Remarks

In the following, we want to address some notable aspects regarding the findings of this thesis.

Synthesis and Post-Layout Area Figures In order to obtain meaningful results for the designed hardware architectures, full ASIC-design flows were executed. This is necessary as it allows to estimate power consumption and more importantly, area consumption after layout and timing is fixed. In addition to

that, we can also give intermediate results at multiple design stages which is especially interesting for the area consumption figure as it illustrates the variance of post-layout results compared to synthesis results. We now want to consider area consumption figures at the following stages:

Synthesis: Area figure considers area consumed by standard cells and macro cells

Post-placement (gate-count): Extends synthesis results by additionally considering buffer and gate strengths in order to fulfill timing constraints, but does generally not consider area for wiring nor area due to power distribution.

Post-layout (effective area): Extends post-placement area figure by considering additional effects due to routing and placement and power distribution network area due to power rings and power stripes.

Based on synthesis results, the four-step iterative Karatsuba design (GeminiK77) consumes less area then its three-step counterpart (GeminiK153). However, the smaller design at synthesis stage becomes effectively bigger in the post-layout results. Basically, the architecture based on the iterative four-step Karatsuba multiplier (GeminiK77) involves a high demand for signal routing, especially in the result register feedback structure. It contains many cells which are dispersed over the chip area requiring widespread interconnections representing a significant routing overhead. Another reason may be due to the fact that the contained signal paths in this design are very wide compared to usual signal widths in hardware design. Hence, the routing algorithms may not recognize these structures properly and fail to find near-optimal routing. A similar situation is also observable for the designs GeminiLSD11 and GeminiLSD9. While they have the same area consumption regarding synthesis and gate-count figures, their effective chip area figures differ by about 14 %. At first, we want to explain why these designs are of equal size at the system level, while their contained multiplier architectures differ by about 6 kGE. The reason is due to the different memory configuration where the GeminiLSD9 architecture contains a macrocell configuration which is less efficient in terms of bit per area than the 77-bit datapath based GeminiLSD11 architecture. The difference due to memory configuration is about 6 kGE (Table 5.7) which results in an equalized area figure at the system level. The reason for the difference in post-layout area is due to the large core multipliers and interleaved reduction circuits, where in the GeminiLSD11 architecture, this structure is more complex. Again, this results in more routing overhead which makes the GeminiLSD11 larger than expected when considering the results of GeminiLSD9.

These observations clearly show that design evaluations limited to synthesis results may lack important information to extrapolate to the effective chip area consumption of a design. Intrinsic system parameters such as routing overhead

due to wide signals contribute to the variability of the effective area consumption results. Consequently, care should be taken when making estimations based on synthesis results to conclude on effective chip area.

Datapath Width The presented designs are based on two datapath widths, namely 77-bit and 153-bit. The width of the datapath determines how many bits are computed per clock cycle. The amount of data which can be computed in a given amount of time is referred to as throughput. For the given architectures a wide datapath using a wide memory interface allows to reduce the number of I/O operations to read from or write to the memory. For the architectures using a 77-bit memory interface we have at least $2 \cdot \lceil 1223/77 \rceil = 32$ read/write cycles per operation. The architectures using memories with twice the interface width require only half the number of cycles per operation which significantly improves overall computation time. While widening the datapath allows to increase throughput it also affects the size of operating units (e.g., multiplier size). Another point to consider is that a wide datapath usually implies using wider memory interfaces so that effects of macrocell configuration have to be considered. As discussed in Section 5.5, a wide memory interface may have a detrimental effect on area consumption, especially in the context of a low area design goal.

Custom Cryptoprocessor vs. General Purpose Processor To calculate the η_T pairing we operate in a finite field of small characteristic. To comply with the design goal of a pairing at 128-bit security level, base-field operations with 1223-bit operands are required. Furthermore, the pairing algorithm itself is rather complex and mandates a large number of finite-field operations resulting a significant computational load. A general purpose processor is generally not well suited for this task as their usually small datapaths would require many intermediate processing steps to process operands of the required dimension. Hence, the design effort in an application-specific dedicated circuit is justified, if a pairing should be calculated within reasonable amount of time and energy. Such circuits may then be used as coprocessors to provide instruction set extensions to general purpose processors or to construct high-level stand-alone pairing processors for embedded devices such as smart cards.

9.2 Future Work

This section proposes potential candidates for future or complementary work regarding the presented results.

Integration of Point Multiplication The implemented architectures are capable of performing finite-field operations in $\mathbb{F}_{2^{1223}}$ with a reduction polynomial $f(x) = x^{1223} + x^{255} + 1$. This provides all the functionality to compute the η_T pairing. However, for actual applications it may be advantageous to further extend the architecture. Extending the instruction set of the implemented ASIP could provide the full feature set of an identity-based encryption system (possibly including the trusted authority).

Generalizing the Iterative Karatsuba Multiplier Approach The presented approach of an iterative Karatsuba multiplier has two scaling factors. The first being the degree of iteration where we can downscale the multiplier area-wise due to smaller core multiplier width. The second being the number of accumulation registers in the operand accumulation circuit. Further work on the second scaling factor could investigate the trade-off between area spent on registers in the accumulation circuit and the corresponding reduction of multiplier latency to find a potentially more efficient multiplier with higher core multiplier saturation. Generalizing the iterative Karatsuba multiplier approach would allow to provide results for a wider range of operand sizes and investigate the effects of accumulation register size scaling. The current multiplier architecture could be used as basis to write a generator for generic multiplier widths and accumulation circuits of generic size.

Optimizing Register Allocation Using a declarative programming language such as Prolog could provide a register allocation pattern which minimizes the computational overhead due to memory copy operations. In a similar approach the memory footprint could also be optimized further.

Low-area Implementation of η_T Pairing in Characteristic Three The η_T pairing can also be implemented using a ternary finite field. A ternary finite field requires to spend two bits per finite-field element instead of one in the binary finite field. While binary finite field arithmetic is basically more efficient to be implemented in digital hardware the finite field size in the ternary case can be reduced. Previous work as given in [8] suggest that there is a slight advantage for ternary implementations. Considering the work in [51] which effectively lowers the security level provided by the ternary field reduces the advantage of the ternary version. As this work presents a low-area hardware implementation of the η_T pairing in fields of characteristic two, a follow-up work could explore low-area implementations based on the η_T pairing in characteristic three.

Bibliography

[1] Jithra Adikari, Anwar Hasan, and Christophe Negre. Towards Faster and Greener Cryptoprocessor for Eta Pairing on Supersingular Elliptic Curve over $\mathbb{F}_{2^{1223}}$. Homepage of CACR, University of Waterloo, Canada, 2012.

[2] ANSI. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.* American National Standards Institute.

[3] ANSI. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).* American National Standards Institute.

[4] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[5] Paulo Barreto, Steven Galbraith, Colm Ó hÉigeartaigh, and Michael Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.

[6] Paulo Barreto, Hae Kim, Ben Lynn, and Michael Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2002.

[7] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three. *Computers, IEEE Transactions on*, 57(11):1454 –1468, Nov. 2008.

[8] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, and Francisco Rodríguez-Henríquez. A Comparison Between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. Cryptology ePrint Archive, Report 2008/115, 2008.

[9] Dan Boneh. The Decision Diffie-Hellman Problem. In *Algorithmic number theory*, pages 48–63. Springer, 1998.

[10] Dan Boneh and Matt Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[11] Dan Boneh, Craig Gentry, and Brent Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In *Advances in Cryptology-ASIACRYPT 2001*, pages 514–532. Springer, 2001.

[13] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 2010.

[14] Don Coppersmith. Fast Evaluation of Logarithms in Fields of Characteristic Two. *Information Theory, IEEE Transactions on*, 30(4):587–594, 1984.

[15] Faraday Technology Corporation. *Faraday ASIC Cell Library, FSA0A_C 180 nm Standard Cell*. Faraday, 2004.

[16] Iwan Duursma and Hyang-Sook Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In *Advances in Cryptology - ASIACRYPT 2003*, Lecture Notes in Computer Science.

[17] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster $\mathbb{F}_p$-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. *CHES*, 2009.

[18] David Freeman, Michael Scott, and Edlyn Teske. A Taxonomy of Pairing-Friendly Elliptic Curves, 2006.

[19] Gerhard Frey and Hans-Georg Rück. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of computation*, 62(206):865–874, 1994.

[20] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for Cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[21] Santosh Ghosh, Dipanwita Roychowdhury, and Abhijit Das. High Speed Cryptoprocessor for η_T Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 442–458. Springer, 2011.

[22] R. Granger, D. Page, and M. Stam. On Small Characteristic Algebraic Tori in Pairing-Based Cryptography. Cryptology ePrint Archive, Report 2004/132, 2004.

[23] Jorge Guajardo and Christof Paar. Itoh-tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes and Cryptography*, 25:207–216, 2002.

[24] Darrel Hankerson, Alfred Menezes, and Michael Scott. Software Implementation of Pairings. In *Identity-Based Cryptography*. IOS Press, 2009.

[25] D.R. Hankerson, S.A. Vanstone, and A.J. Menezes. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004.

[26] Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, pages 310–324. Springer, 2003.

[27] Markus Hütter, Johann Großschädl, and Guy-Armand Kamendje. A Versatile and Scalable Digit-Serial/Parallel Multiplier Architecture for Finite Fields GF(2^m). In *4TH International Conference on Information Technology: Coding and Computing(ITCC 2003)*, pages 692–700. IEEE Computer Society, 2003.

[28] Toshiya Itoh and Shigeo Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. *Information and Computation*, 78(3):171–177, 1988.

[29] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Algorithmic Number Theory*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–393. Springer, 2000.

[30] David Kammler, Diandian Zhang, Peter Schwabe, Hanno Scharwaechter, Markus Langenberg, Dominik Auras, Gerd Ascheid, and Rudolf Mathar. Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. *LNCS / CHES 2009*, 2009.

[31] Anatolii Karatsuba and Yu Ofman. Multiplication of Multidigit Numbers on Automata. In *Soviet physics doklady*, volume 7, page 595, 1963.

[32] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1997.

[33] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[34] Cetin Kaya Koc. *Cryptographic Engineering*. Springer Series. Springer, 2009.

[35] S. Kumar, T. Wollinger, and C. Paar. Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. *Computers, IEEE Transactions on*, 55(10):1306–1311, Oct. 2006.

[36] Ian Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203–215, 2007.

[37] Luther Martin. *Introduction to Identity-Based Encryption*. Information Security and Privacy Series. Artech House, 2008.

[38] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 80–89. ACM, 1991.

[39] Victor S. Miller. Short Programs for functions on Curves. Unpublished manuscript, 1986.

[40] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology - CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1986.

[41] Victor S. Miller. The Weil pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[42] P.L. Montgomery. Five, Six, and Seven-Term Karatsuba-Like Formulae. *Computers, IEEE Transactions on*, 54(3):362 – 369, march 2005.

[43] Dan Page, NP Smart, and Fre Vercauteren. A Comparison of MNT Curves and Supersingular Curves. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):379–392, 2006.

[44] John M Pollard. Monte Carlo Methods for Index Computation (*mod p*). *Mathematics of computation*, 32(143):918–924, 1978.

[45] Chester Rebeiro and Debdeep Mukhopadhyay. Hybrid Masked Karatsuba Multiplier for GF(2^{233}), 2008.

[46] F Rodrıguez-Henrıquez and ÇK Koç. On fully parallel Karatsuba Multipliers for GF(2^m). In *Proc. International Conference on Computer Science and Technology-CST*, pages 405–410, 2003.

[47] Francisco Rodríguez-Henríquez, N.A. Saqib, Arturo Díaz Perez, and Çetin Kaya Koç. *Cryptographic Algorithms on Reconfigurable Hardware*. Signals and Communication Technology. Springer, 2007.

[48] Robert Ronan, Colm O'hEigeartaigh, Colin Murphy, Michael Scott, and Tim Kerins. FPGA Acceleration of the Tate Pairing in Characteristic 2. In *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 213–220. IEEE, 2006.

[49] Richard Schroeppel, Hilarie Orman, Sean O'Malley, and Oliver Spatscheck. *Fast Key Exchange with Elliptic Curve Systems*. Springer, 1995.

[50] Adi Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53. Springer, 1985.

[51] Naoyuki Shinohara, Takeshi Shimoyama, Takuya Hayashi, and Tsuyoshi Takagi. Key Length Estimation of Pairing-based Cryptosystems using η_T Pairing. Cryptology ePrint Archive, Report 2012/042, 2012.

[52] Chang Shu, Soonhak Kwon, and Kris Gaj. FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields. Cryptology ePrint Archive, Report 2006/179, 2006.

[53] Joseph H Silverman. *Advanced Topics in the Arithmetic of Elliptic Curves*. 1994.

[54] Leilei Song and KeshabK. Parhi. Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *Journal of VLSI signal processing systems for signal, image and video technology*, 19:149–166, 1998.

[55] Paul C Van Oorschot and Michael J Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of cryptology*, 12(1):1–28, 1999.

[56] Joachim von zur Gathen and Jamshid Shokrollahi. Efficient FPGA-Based Karatsuba Multipliers for Polynomials over $\mathbb{F}_2$. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 359–369. Springer, 2006.

[57] Johannes Wolkerstorfer. Dual-Field Arithmetic Unit for GF(p) and GF(2^m). In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 500–514. Springer, 2003.

Appendix A

Pairing Applications

In the following we present several applications for pairing-based cryptography which benefit from the new cryptographic primitive available due to bilinear pairings. The pairing operation in generally denoted as a function $e(G_1, G_2) \rightarrow G_3$ where the elements G_1, G_2, are elements of an additive and G_3 is an element of a multiplicative group.

A.1 Identity-Based Encryption

In response to Adi Shamir's idea for an identity-based encryption (IBE) system and quest for a way to implement such a system Boneh and Franklin proposed a solution to implement such identity-based encryption systems using bilinear pairings based on the Weil pairing. As such IBE is historically linked to bilinear pairings and of major importance for new cryptographic schemes and applications. In the following we want to illustrate the steps in the basic Boneh-Franklin IBE for encryption and decryption based on and according to [10] [37].

Let G_1 and G_T be cyclic groups where G_1 is a subgroup of an elliptic curve $E(\mathbb{F}_q)$ and G_2 is a subgroup of $\mathbb{F}^*_{q^k}$. The private key generator (PKG) represents the trusted authority and performs a parameter setup step which defines the master secret key s and an elliptic curve generator point P. The master public key is obtained by multiplying the elliptic curve point P with the master secret key s.

- Trusted Authority
  - Private key: $s \in Z^*_q$
  - Public key: $P, sP \in G_1$
- User

- A user's public key is derived from an *identity string* ID which is mapped to an element in G_1 using a cryptographic hash function[1] H_1, defined as $H_1 : \{0,1\}^* \rightarrow G_1$, such that the user's public key Q_{ID} is: $Q_{ID} = H_1(ID) \in G_1$
- A user's private key corresponding to an identity ID is calculated using the master private key s with: $sQ_{ID} \in G_1$

It should be noted here that the concept of identity strings is not limited to strings which uniquely identify the recipient such as an email address, a phone number, or a social security number. The identity string could also contain a role description which attributes a role to the recipient. As an example, a sender could encrypt messages to `someone@hostpital.net|role=doctor`. The PKG may then require successful authentication according to a role description prior to transmission of a private key. This technique makes it easy to implement so-called *role-based encryption*.

For sending an encrypted message $M \in \{0,1\}^n$ to a recipient of identity ID the sender executes the following steps. Let H_2 be a cryptographic hash function to hash an element of G_T to a string of length n so that $H_2 : G_T \rightarrow \{1,0\}^n$.

1. Choose a random integer $r \in Z_p^*$ to multiply the generator point P giving rP
2. Use identity ID of recipient to calculate the corresponding public key $Q_{ID} = H_1(ID)$. The public key sP of the trusted authority is then paired with rQ_{ID} and hashed to a string of length n. $K_{snd} = H_2(e(rQ_{ID}, sP))$
3. Assemble ciphertext C which consists of $C_1 = rP$ and $C_2 = M \oplus K_{snd}$ such that $C = (C_1, C_2)$.

To decrypt the ciphertext $C = (rP, M \oplus H_2(e(rQ_{ID}, sP)) = (C_1, C_2)$ the recipient executes:

1. Calculate $K_{rcv} = H_2(e(sQ_{ID}, C_1))$ where sQ_{ID} is the private key of the recipient.
2. Calculate $M = C_2 \oplus K_{rcv}$ to obtain the plain text message M.

Due to the bilinearity of the pairing, K_{snd} is equivalent to K_{rcv} which reveals the plaintext message M to the receiver.

$$K_{snd} = H_2(e(rQ_{ID}, sP)) = H_2(e(Q_{ID}, P)^{rs}) \tag{A.1}$$
$$K_{rcv} = H_2(e(sQ_{ID}, C_1)) = H_2(e(Q_{ID}, P)^{sr}) \tag{A.2}$$

[1] see [10], IEEE 1363.3

A.2 Tripartite Key Exchange

An early constructive application of bilinear pairings in cryptography is the one-round tripartite Diffie-Hellman key exchange protocol of Joux [29] which is illustrated in Figure A.1. It is based on the so-called bilinear Diffie-Hellman problem (BDH) which assumes that, given P, aP, bP, and cP, it is hard to compute $e(P,P)^{abc}$.

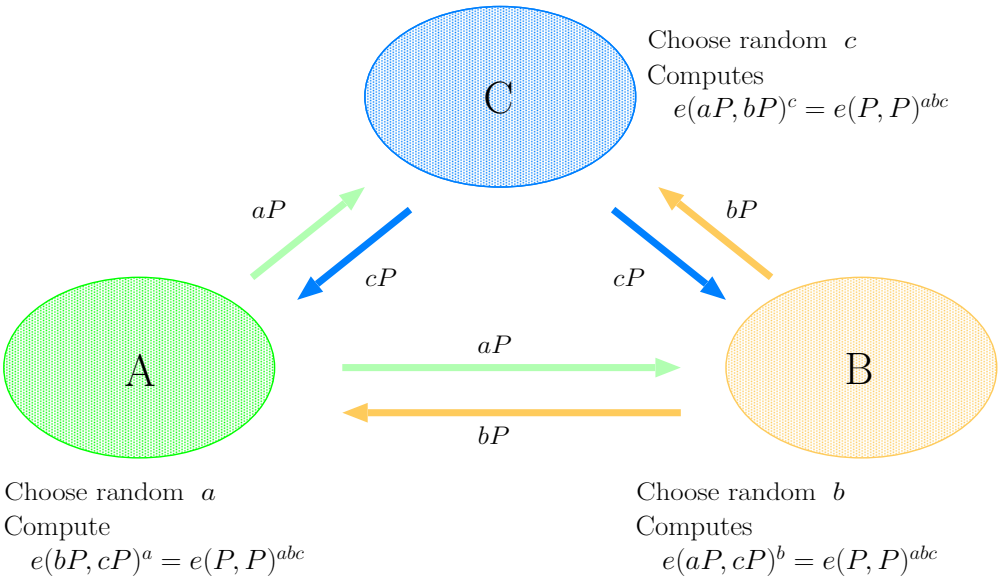


Figure A.1: Tripartite one-round key exchange protocol

A.3 Signature Schemes

Identity-based signature schemes use four algorithms namely setup, extract, sign, and verify, executed by the parties of the signer, verifier, and a trust authority (TA). In [26] an efficient identity-based signature scheme is discussed based on bilinear pairings. It also contains a discussion on the issue of *key escrow* in identity-based signature schemes. The trusted authority has natural access to the signers private key and thus can create signatures which are indistinguishable from those of the authentic signer. Boneh, Lynn, and Shacham [12] propose a so-called short signature scheme which is provable secure assuming random oracles and the intractability of the computational Diffie-Hellman problem.

Appendix B

Signal-Flow Graphs / Operation Scheduling

The algorithms were inspected closely to find an operand scheduling and memory allocation pattern with a low memory footprint and good performance using in-place execution whenever possible. For this examination the various algorithms were transformed to signal-flow graphs, which allowed to find possible enhancements, such as, relocating a variable to avoid a copy operation or reusing memory locations to avoid unnecessary memory consumption for temporary variables.

To keep the total computation time low the operands should be located in distinct operand-memory blocks for every execution to avoid a costly copy operation. While the architecture supports copying operands between the two operand-memory blocks this practice is largely eliminated by reorganizing the instruction sequence and memory allocation.

The signal-flow graphs given here indicate the operand memory-block association with A and B. Input and output variables are also declared in conformance to the main algorithms.

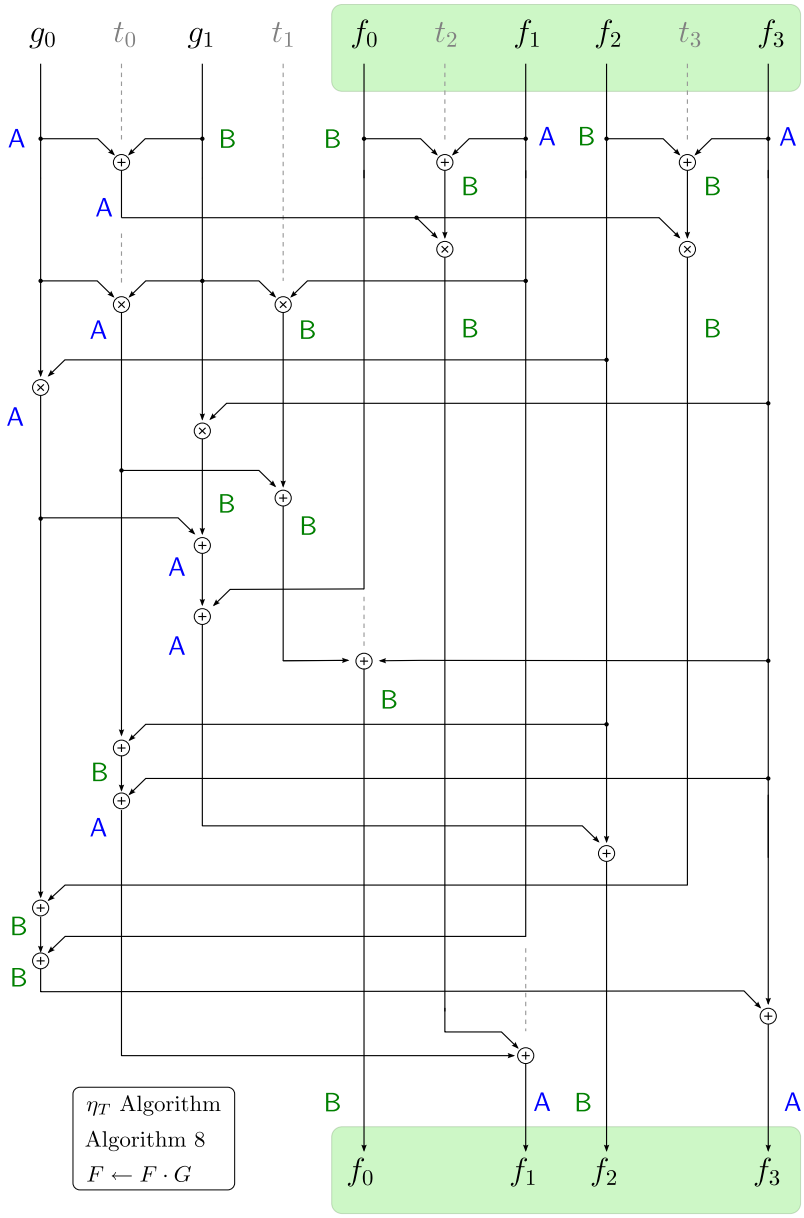


Figure B.1: Example of a signal-flow graph calculating $F \leftarrow F \cdot G$ of Algorithm 2

Appendix C

Geminicore

C.1 Layout

In an early design phase a back-end flow with the Karatsuba 77-bit multiplier was exercised to evaluate post-layout results. The design was constrained by the area provided by a standard miniASIC pad frame. Several steps were necessary to fit the design to the pad frame. One of the first steps was reducing the clock period to 9 ns which provided the smallest circuit. The VCC and GND core power rings were downsized to 9 μm width and overlayed to minimize area for the power supply. Power rail analyses confirmed that safe VCC and GND supply levels are respected using a clock period of 9 ns. Back-end tools were configured not to attempt concentrated module placement of standard cells. Still there were thousands of Design Rule Violations (DRV) due to the imposed area constraint. To gain more core area the input/output interface of the design was changed so that it requires less input/output pads from a 11-bit to a 7-bit I/O interface. This allowed to remove the bottom row of the pad ring and gain of about 15 % additional core area. The area gain allowed to lower the number of DRVs significantly. Still many violations occured and required further optimizations. Rerouting with Engineering Change Order (ECO) options helped to fix some of the occurring violations. Practically every optimization step in back-end design needed several repetitions and optimizations. All in all it was a tedious task to obtain a layout solution which was free of design rule and timing violations. The final layout is depicted in Figure C.1. The design contains several large multiplexers and operates on signal widths of up to 2448 bit which contribute to the design's routing complexity which probably contributed to a intense back-end flow. The dense routing at the top metal layer as can be seen in the layout figure illustrate this fact.

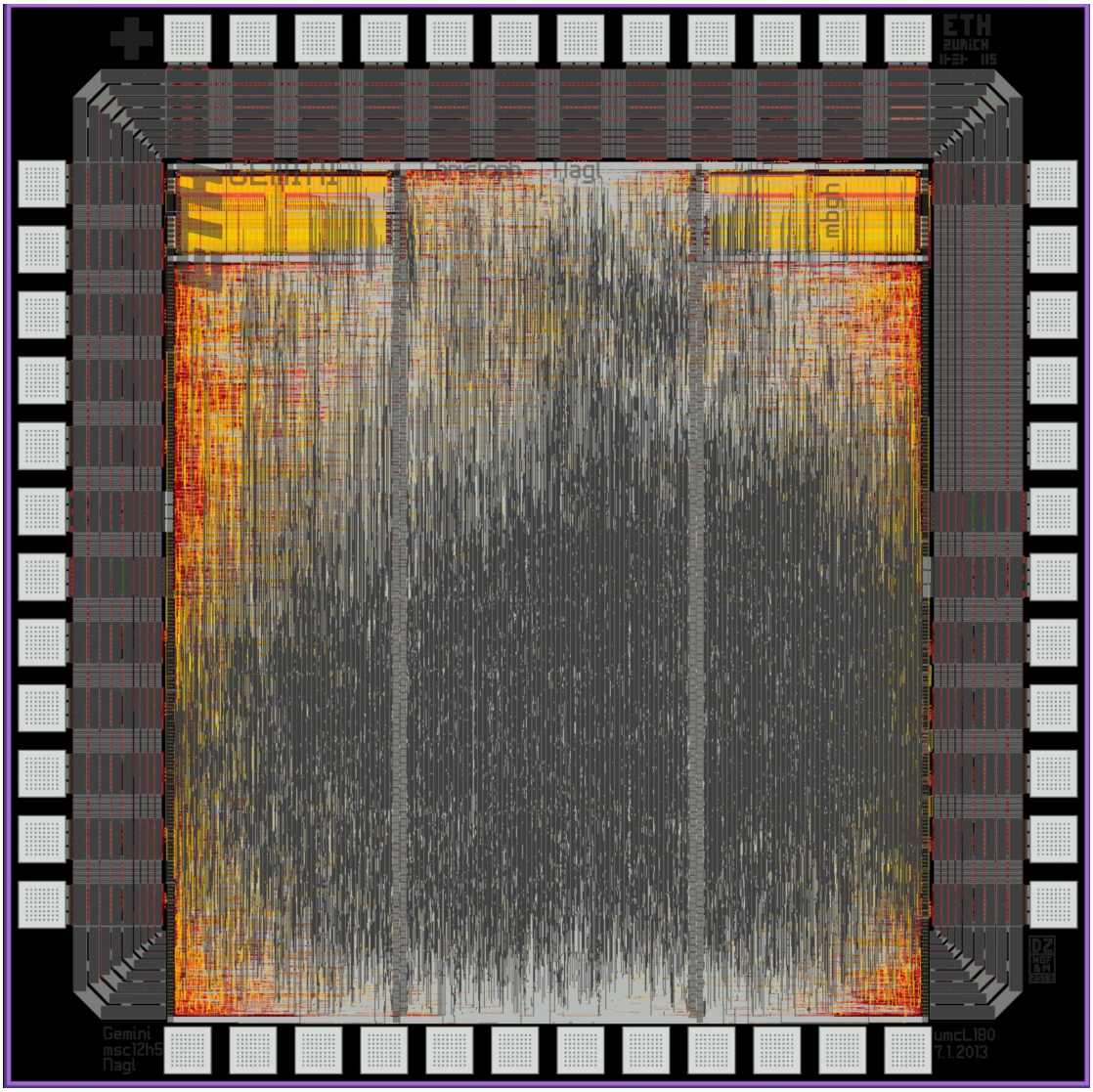


Figure C.1: Layout of Karatsuba 77-bit multiplier architecture

C.2 Datasheet

The chip is enclosed in an quad-flat no-leads (QFN) package which provides 56 pins. The given implementation does not use all of the package-provided pins so that 32 pins require bonding wires to the landing pads. For a detailed pin diagram and pin description see Figure C.2 and Table C.2.

Table C.1: Electrical specifications

Core-Power Supply	1.8 V
Pad-Power Supply	3.3 V
Max. Clock Frequency	90 MHz †

† Based on post-layout simulation

C.2.1 Pinout and Pin Description

The pinout using a 7-bit input/output interface is given in Figure C.2. The pin name prefix indicates the association of pins to the respective landing pads on the chip die.

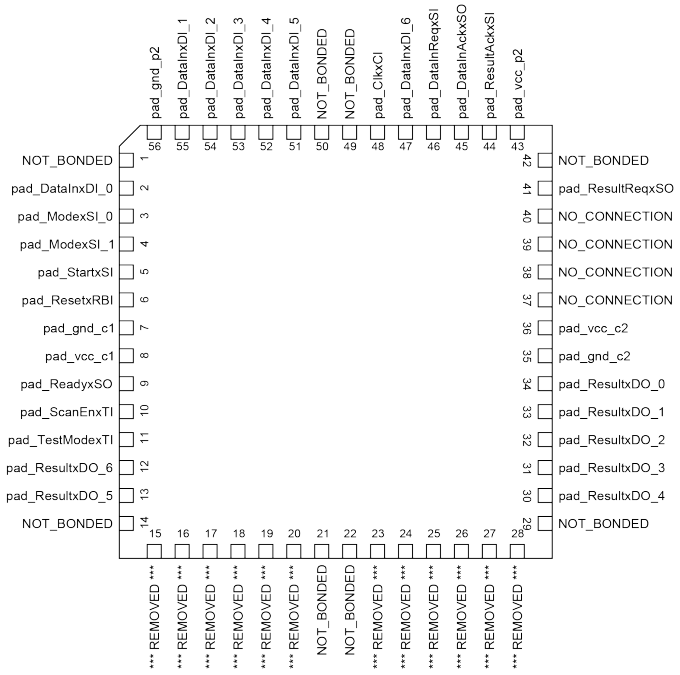


Figure C.2: Pin diagram of Geminicore

C.2.2 Input/Output Interface

The interface supports a four-phased handshake protocol (full handshake) for data input and data output. Input handshaking signals are provided by `DataInReqxSI` and `DataInAckxSO` for data request and data acknowledge respectively. The output handshake interface is available by the pins `DataOutReqxSO` and `DataOutAckxSI`.

Table C.2: Pin descriptions

Pin name	Pin number	Pin type	Description
pad_ClkxCI	48	Clock	Clock network
pad_ResetxRBI	6	Reset	Reset network
pad_StartxSI	5	Input	Start signal
pad_ReadyxSO	9	Output	Ready signal
pad_ModexSI_0	3	Input	Mode select 0
pad_ModexSI_1	4	Input	Mode select 1
pad_DataInxDI_0	2	I/O	Data Input 0 (D/S)
pad_DataInxDI_1	55	I/O	Data Input 1 (D/S)
pad_DataInxDI_2	54	I/O	Data Input 2 (D/S)
pad_DataInxDI_3	53	I/O	Data Input 3 (D/S)
pad_DataInxDI_4	52	I/O	Data Input 4 (D/S)
pad_DataInxDI_5	51	I/O	Data Input 5 (D/S)
pad_DataInxDI_6	47	I/O	Data Input 6 (D/S)
pad_ResultxDO_0	34	I/O	Data Output 0 (D/S)
pad_ResultxDO_1	33	I/O	Data Output 1 (D/S)
pad_ResultxDO_2	32	I/O	Data Output 2 (D/S)
pad_ResultxDO_3	31	I/O	Data Output 3 (D/S)
pad_ResultxDO_4	30	I/O	Data Output 4 (D/S)
pad_ResultxDO_5	13	I/O	Data Output 5 (D/S)
pad_ResultxDO_6	12	I/O	Data Output 6 (D/S)
pad_vcc_c1	8	Power	Core VCC supply
pad_vcc_c2	36	Power	Core VCC supply
pad_gnd_c1	7	Power	Core GND supply
pad_gnd_c2	35	Power	Core GND supply
pad_vcc_p2	43	Power	Pads VCC supply
pad_gnd_p2	56	Power	Pads GND supply
pad_DataInReqxSI	46	Input	Input Request (HS)
pad_DataInAckxSO	45	Output	Input Acknowledge (HS)
pad_ResultReqxSO	41	Output	Output Request (HS)
pad_ResultAckxSI	44	Input	Output Acknowledge (HS)
pad_TestModexTI	11	Input	Enable TestMode
pad_ScanEnxTI	10	Input	Enable ScanMode

HS . . . Handshaking signal
D/S . . . Data port multiplexed with scan chain (input/output)

Appendix D

Factor Graph

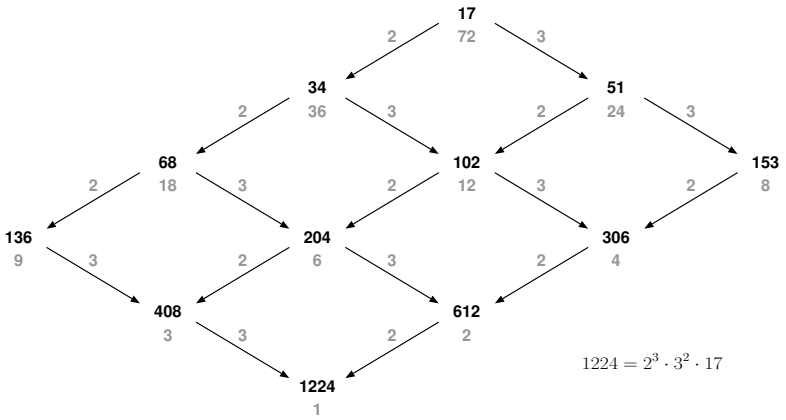


Figure D.1: Factorization tree for 1224

To outline possible Karatsuba segmentations for the finite field operand sizes of 1223 bit we illustrate the factorization of 1224 in Figure D.1. The factors are represented by arrows connecting nodes where the segment size is given above the number of segments. This graph allows to quickly examine possible segmentation configuration. The factorization is given for 1224 as it is the next higher non-prime number to the applied finite field size.

Appendix E

Three-Way Karatsuba

Karatsuba's algorithm usually splits operands in two segments each of halved original operand size. It is also possible to split the operands in three segments which results in sub-operand sizes of one third of the original operand size. In the following example the segment width is denoted as k and the operand width is given as m to illustrate a polynomial multiplication $A \cdot B = C$ where the operand polynomials are of degree $m-1$ and the result polynomial C is of degree $2m-2$. With this version of Karatsuba's algorithm a $k \times k$ core multiplier can be used to compute a $m \times m$ multiplication.

$$
\begin{aligned}
A_0(x) &= (a_{k-1} \cdot x^{k-1} + \ldots + a_0 \cdot x^0) \\
A_1(x) &= (a_{2k-1} \cdot x^{2k-1} + \ldots + a_k \cdot x^k) \\
A_2(x) &= (a_{3k-1} \cdot x^{3k-1} + \ldots + a_{2k} \cdot x^{2k})
\end{aligned}
$$

$$
\begin{aligned}
B_0(x) &= (b_{k-1} \cdot x^{k-1} + \ldots + b_0 \cdot x^0) \\
B_1(x) &= (b_{2k-1} \cdot x^{2k-1} + \ldots + b_k \cdot x^k) \\
B_2(x) &= (b_{3k-1} \cdot x^{3k-1} + \ldots + b_{2k} \cdot x^{2k})
\end{aligned}
$$

$$
\begin{aligned}
D_0 &= A_0 \cdot B_0 \\
D_1 &= A_1 \cdot B_1 \\
D_2 &= A_2 \cdot B_2 \\
D_{0,1} &= (A_1 + A_0) \cdot (B_1 + B_0) \\
D_{1,2} &= (A_2 + A_1) \cdot (B_2 + B_1) \\
D_{0,2} &= (A_2 + A_0) \cdot (B_2 + B_0)
\end{aligned}
$$

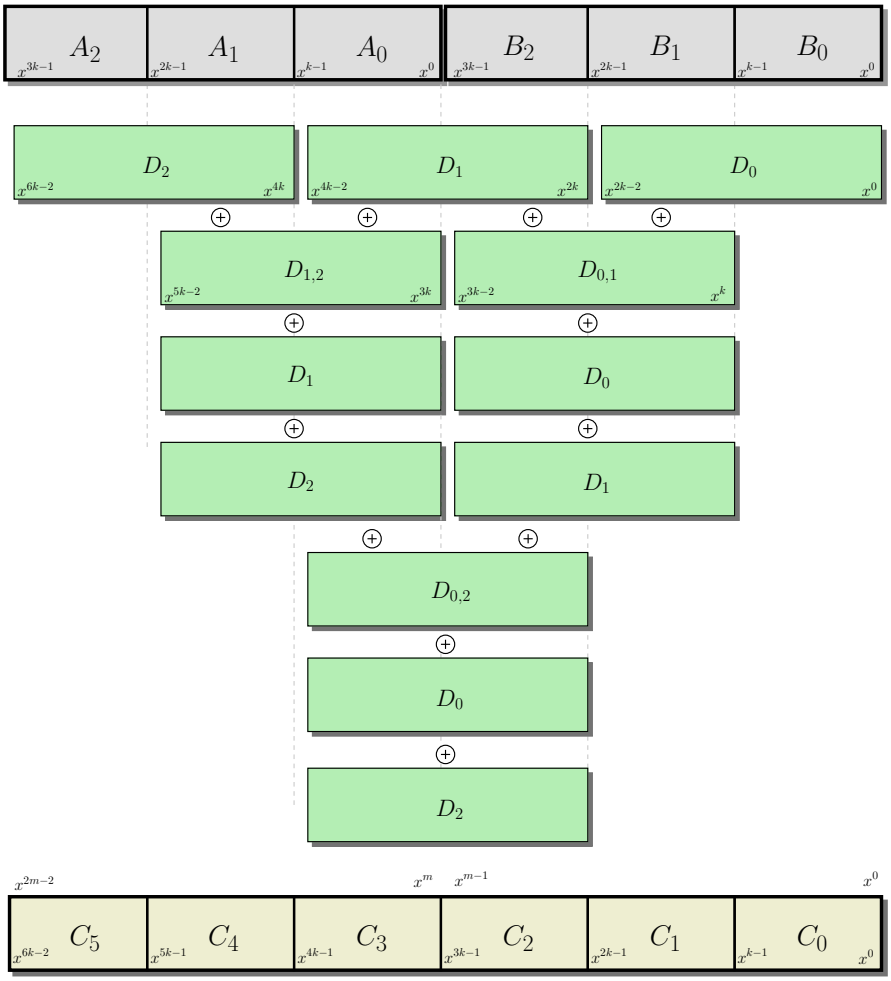


Figure E.1: Segmentation and combination for a one-step 3-way Karatsuba

Appendix F

η_T Pairing Cost in $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$

Table F.1 lists the finite-field operations to calculate the η_T pairing based on the algorithms by [5], [16] and [8]. Originally the characteristic three field $\mathbb{F}_{3^{509}}$ was expected to have a security level of 128 bit. The findings by Shinohara et al. in [51] show that the security level is actually 111 bit.

Table F.1: Comparing the finite field operations of η_T pairing in $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$

Security level		Low		Medium		High	
		67	52	96	78	128	111
Field		$\mathbb{F}_{2^{239}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{2^{557}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{2^{1223}}$	$\mathbb{F}_{3^{509}}$
η_T Pairing	Add	2629	2576	6127	6374	13453	13597
	Mul	836	606	1949	1493	4280	3181
	Squ[a]	956	531	2228	1312	4892	2797
	Inv	0	0	0	0	0	0
Fin.Exp.	Add	531	464	1167	890	2499	1700
	Mul	26	73	26	73	26	73
	Squ[a]	487	294	1123	720	2455	1530
	Inv	1	1	1	1	1	1
Total	Add	3160	3040	7294	7264	15952	15297
	Mul	862	679	1975	1566	4306	3254
	Squ[a]	443	825	3351	2032	7347	4327
	Inv	1	1	1	1	1	1

[a] In $\mathbb{F}_{3^m}$ the squaring operation corresponds to a cubing operation.

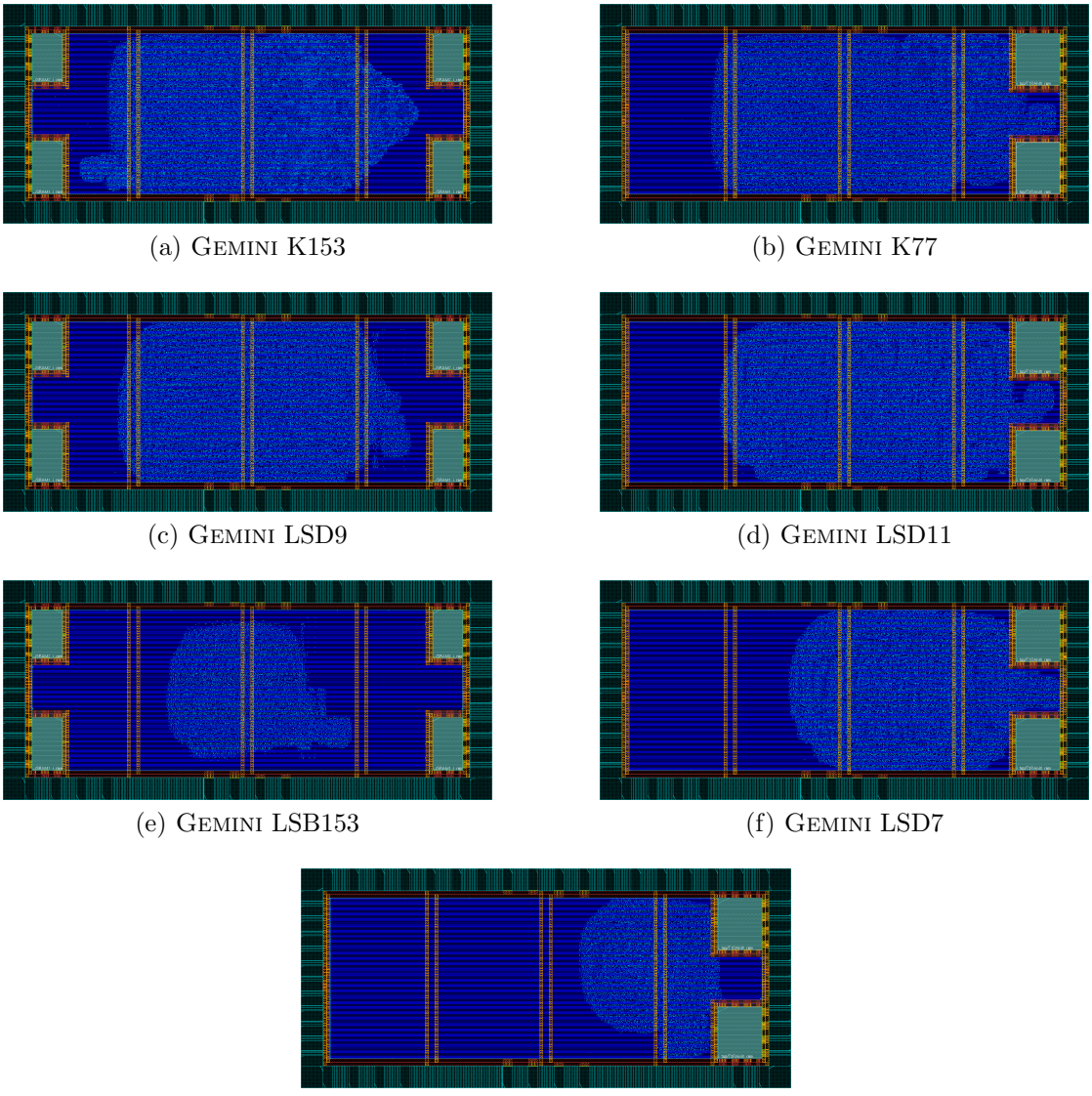
Appendix G

International Standards on Identity-Based Cryptography

- IEEE P1363 Identity-Based Public Key Cryptography
  - IEEE P1363.3 Standard for Identity-Based Cryptographic Techniques using Pairings
- ISO/IEC 11770-3 Key Management / Mechanisms using asymmetric techniques
- ISO/IEC 14888-2 Digital Signatures with appendix/Integer factorization based mechanisms
- ISO/IEC 14888-3 Digital Signatures with appendix/Discrete Logarithm based mechanisms

Appendix H

Layout Results for Unconstrained Area



(a) Gemini K153

(b) Gemini K77

(c) Gemini LSD9

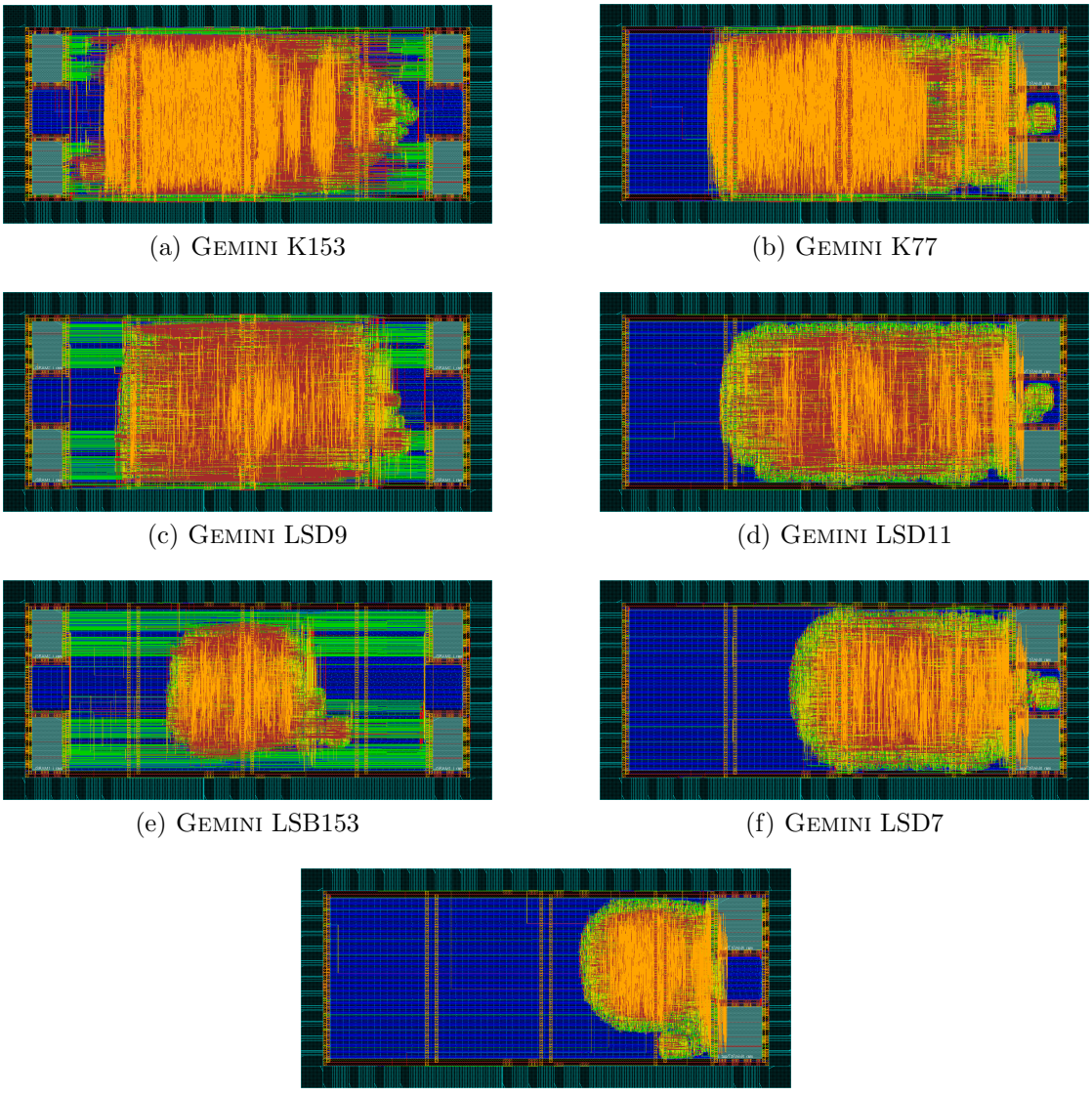
(d) Gemini LSD11

(e) Gemini LSB153

(f) Gemini LSD7

(g) Gemini LSB77

Figure H.1: Standard cell placement in back-end design flow (unconstrained area)



(a) Gemini K153

(b) Gemini K77

(c) Gemini LSD9

(d) Gemini LSD11

(e) Gemini LSB153

(f) Gemini LSD7

(g) Gemini LSB77

Figure H.2: Routing results in back-end design flow (unconstrained area)