ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

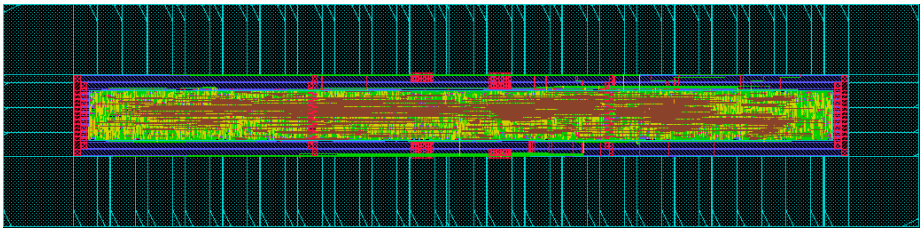
Institut für Integrierte Systeme
Integrated Systems Laboratory

TU Graz
Graz University of Technology

IAIK TU Graz

# Elliptic Curve Cryptography ASIC for Radio Frequency Authentication

## Master Thesis



Author:
Daniel Hein, dhein@student.ethz.ch

Advisors:
Luca Henzen, henzen@iis.ee.ethz.ch
Matthias Brändli, braendli@iis.ee.ethz.ch
Norbert Felber, felber@iis.ee.ethz.ch
Co-Advisor:
Johannes Wolkerstorfer, johannes.wolkerstorfer@iaik.tugraz.at

Prof. Dr. W. Fichtner
Integrated Systems Laboratory (IIS)
ETH Zürich 2008

# Acknowledgements

First I would like to thank Johannes Wolkerstorfer from Graz University of Technology for proposing this thesis and his patience with my questions on elliptic curve cryptography. He also co-examined this thesis.

Next I would like to thank Norbert Felber for paving the way to write this thesis at the Integrated Systems Laboratory of the ETH Zürich, his constructive criticism and of course for examining this document.

My thanks also go to Luca Henzen for being my first point of contact for all hardware relevant questions. His efforts in providing useful answers and discussing my implementation ideas are appreciated. In addition, I would also like to thank him for co-examining this thesis.

For his invaluable advise concerning the tools involved, I would like to thank Mathias Brändli. His aid with my questions about synthesis and backend design is greatly appreciated.

I would also like to thank the enitre IIS team of the ETH Zürich for their support and the warm welcome they provided.

In addition, I would also like to acknowledge the assistence given by the Mobilitätstelle of the ETH Zürich. Thank you, for providing me with an ERASMUS scholarship, without great ado.

Last, but most certainly not least, my special thanks go out to my fiancée Kari for her unwaivering support and understanding. Finally I would like to thank my parents for their long-standing aid over the course of my academic studies.

**Abstract**

Radio Frequency Identification (RFID) technology is currently revolutionizing supply chain management. Cloning-resistant RFID tags could put a permanent stop to product piracy. Tag authentication based on Elliptic Curve Cryptography (ECC), a sound and standardized cryptographic methodology, could provide this copy protection facility.

The small key sizes achievable with ECC render it the only public-key cryptosystem viable for an RFID application. Current low power, small area ECC implementations are realized as Application-Specific Integrated Circuits (ASICs) using full precision datapaths optimized for a specific underlying finite field.

This work presents a new approach for ECC on devices with a fiercely constrained die area and power budget. It employs a datapath with a word size of 16 bit. The core component of the datapath is a $\mathbb{F}_2$ arithmetic unit, which realizes operations in the binary extension field $\mathbb{F}_{2^{163}}$with a fixed standardized irreducible polynomial.

A design space-exploration of word level algorithms led to the development of a new multiplication with interleaved modular reduction method. Minor modifications to a multiply accumulate (MAC) unit produced a hardware component capable to implement this and all other algorithms required for the elliptic curve point multiplication most efficiently. The point multiplication is the operation that lends security to all ECC based cryptographic primitives like authentication.

The *ECCon* ASIC designed and fabricated as part of this thesis is rounded off by an ISO-18000-3-1 compliant digital RFID front-end augmented with the capability to perform a restricted version of the elliptic curve digital signature algorithm (ECDSA). Special attention was paid to the possibility of side channel attacks and measures were implemented to provide high resilience against them.

The *ECCon* processor was fabricated using the UMC L180 GII 180 nm CMOS technology. The area it necessitates is equal to 151125 $\mu m^2$ or 13685 gate equivalents. It performs an elliptic curve point multiplication in 306587 clock cycles and has a power consumption of 11.4 $\mu W$ at a clock frequency of 106 kHz.

**Keywords**: Elliptic curve cryptography, Radio frequency identification, authentication, tag-cloning, security, binary extension fields, ASIC, hardware implementation

# Contents

# List of Algorithms

# List of Tables

# List of Figures

# List of Acronyms

**ECC** Elliptic Curve Cryptography

**RFID** Radio Frequency Identification

**ASIC** Application-Specific Integrated Circuit

**IFF** Identification, Friend or Foe

**WWII** World War II

**ID** Identity

**EPC** Electronic Product Code

**NFC** Near Field Communication

**HF** High Frequency

**EM** Electromagnetic

**NVRAM** Non Volatile Random Access Memory

**MAC** Message Authentication Code

**DSA** Digital Signature Algorithm

**ECDSA** Elliptic Curve Digital Signature Algorithm

**AES** Advanced Encryption Standard

**TTP** Trusted Third Party

**IFP** Integer Factorization Problem

**DLP** Discrete Logarithm Problem

**ECDLP** Elliptic Curve Discrete Logarithm Problem

**PKI** Public-Key Infrastructure

**STS** Station-to-Station

**EC** Elliptic Curve

**ECIES** Elliptic Curve Integrated Encryption Scheme

**SPA** Simple Power Analysis

**FPGA** Field Programmable Gate Array

**ITF** Interrogator Talks First

**IC** Integrated Circuit

**RART** RFID Asynchronous Receiver Transmitter

**RCU** RFID Control Unit

**UID** Unique Identity

**RAM** Random Access Memory

**ROM** Read Only Memory

**ALU** Arithmetic Logic Unit

**MAC** Multiply Accumulate

**FSM** Finite State Machine

**XOR** Exclusive Or

**DSS** Digital Signature Standard

**HDL** Hardware Description Language

**ISA** Instruction Set Architecture

**CPU** Central Processing Unit

**EEA** Extended Euclidean Algorithm

**GCD** Greatest Common Divisor

**CMOS** Complementary Metal-Oxide Semiconductor

**SOF** Start Of Frame

**RTL** Register Transfer Level

**VHDL** Very high speed integrated circuit Hardware Description Language

**LUT** Lookup Table

**DFT** Design For Test

**ATPG** Automated Test Pattern Generator

**BIST** Built-In Self-Test

**MATS++** Modified Algorithmic Test Sequence

**DPA** Differential Power Analysis

**PRNG** Pseudo Random Number Generator

**VLSI** Very Large Scale Integration

**GE** Gate Equivalent

Chapter 1

# Introduction

## 1.1 Overview

Security for Radio Frequency Identification (RFID) devices has received increasing attention over the past few years. This is partially due to privacy issues, which might hamper the acceptance of RFID technology. A second reason is the applicability of RFID technology for product authentication.

Product piracy, the production and selling of counterfeit products, often using the supply chain of the legitimate manufacturer, is an economically crucial problem. The losses incurred by it easily achieve the multi-billion range [LLMF07]. Product authentication has the potential to at least alleviate, if not completely stop, product piracy. RFID technology is already employed in supply chain management as a replacement for the barcode identification system. If it could be enabled with authentication capabilities this would be a great leap forward.

Security in information systems has always been linked with cryptography. The employment of sound cryptographic primitives is difficult for an RFID application. The RFID tag, the device central to all RFID technologies, is fiercely constrained.

Elliptic Curve Cryptography (ECC) is the most resource conservative public-key cryptosystem currently known. The severe conditions entailed by an RFID environment make ECC the only viable solution for sound cryptography for RFID technology. The security of every ECC primitive, including authentication, depends upon the point multiplication operation.

This thesis aims to develop a point multiplication component that could be deployed on an RFID tag. For this an approach novel to low power, small area ECC design for RFID applications was chosen. In addition special care was taken to make the circuit resilient against side channel attacks.

The ECC component is supplemented with a standard compliant RFID front-end. The synthesis of these two components composes the *ECCon*[1] processor, which was fabricated into silicon using the UMC Taiwan L180 GII process.

[1] *ECCon* is a portmanteau of ECC and an abbreviated version of the word economic.

## 1.2 Outline

Chapter §2 will give an overview of RFID technology in general and RFID based entity identification in particular. The latter is a possible application for the Application-Specific Integrated Circuit (ASIC) presented in this work.

The next chapter §3 gives a brief introduction into cryptography. As a comprehensive overview easily attains book length, the aim is to at least graze all concepts that are necessary to understand digital signature based challenge-response authentication protocols. Such protocols allow for secure authentication of one party to another and are a feasible solution for the RFID authentication problem.

The next part of this work finally introduces elliptic curves in §4. Again it is just a quick sketch of the most fundamental ideas that are absolutely essential to define ECC. It discusses the Elliptic Curve Digital Signature Algorithm (ECDSA) as a cryptographic primitive that is applicable to the challenge-response protocols mentioned above and it provides a concrete algorithm to compute the point multiplication, the ECC operation that is central to ECDSA.

In the following chapter (§5) the most important facts about the ISO-18000-3-1 [ISO04] standard are summarized. It is one of the protocols specifically developed for RFID tags in supply chain management and the ASIC developed as part of this thesis employs an ISO-18000-3-1 compliant RFID interface.

This concludes the theoretic background part and the next chapter (§6) covers the architecture of the *ECCon* processor It sets out by defining the exact prerequisites an RFID application demands. Then it continues by detailing the different primary components of the ASIC.

After ascertaining the architecture and assorted design decisions, it becomes necessary to discuss the different options for implementing the binary extension field operations required by the point multiplication. This is done in §7. Amongst others it introduces two algorithms specifically developed for this project, which efficiently solve the multiplication and squaring with interleaved modular reduction problem for a binary extension field with a specific irreducible polynomial.

The implementation chapter details the realization of the *ECCon* processor. First the power saving techniques applied in the design of the final circuit are deliberated and then it highlights its most important specifics of the implementation of the different components of the ASIC. It also gives a brief introduction to side channel attacks and countermeasures against them. It concludes with a presentation of its results and a comparison with related work.

The thesis closes with a discussion of the conclusions gained and an outlook of potential future work in the final chapter (§9).

Chapter 2

# Radio Frequency Identification

Radio Frequency Identification (RFID) enables automated contactless entity identification. Entities in this context include objects, animals or persons. The idea behind RFID is not a new one, a similar approach was already employed in WWII for friend or foe identification (IFF). In the past years RFID technology has received academic and industrial attention because of its applicability in supply chain management, access control and various other fields.

It has been very accurately described as a form of computer vision in [Jue07]. Computer vision deals not only with the physical process of seeing, but more prominently tries to teach computers object recognition. A simple feat for a human, but immensely complex to imitate with machines.

RFID circumvents the problem by enhancing objects with the capability to inform an interested party of their nature. Unlike its barcode predecessor direct line of sight and optical scanning of the labeled entity are not necessary. To stretch above analogy, this is the equivalent to computer X-ray vision.

This chapter gives a concise overview on the subject of RFID. It introduces the RFID tag, the device at the heart of RFID technology, in §2.1. One of the major application of RFID is product identification, where it is the successor of the barcode. Its advantages in comparison to its predecessor are examined in §2.2. The implications of RFID technology for privacy are detailed in §2.3. In §2.4 RFID based item authentication is introduced as a weapon against product counterfeiting.

## 2.1 The RFID tag

Todays most common physical embodiment of an RFID device is the RFID label. An RFID label consists of an RFID tag embedded in an adhesive foil or sticker, similar to those commonly used to add a barcode to an object. RFID tag and tag will be uses synonymously in this text. A tag is a small antenna connected to a micro chip (cf. figure 2.1).

RFID is a set of technologies geared towards entity identification. Several different standards have evolved for RFID communication. They greatly differ in range, transfer speed and frequency used. The more important suites include

**Figure 2.1:** *Photograph of an RFID tag [Mid06]*

the Electronic Product Code (EPC) standard [EPC05] as well as the ISO-14443 [ISO00a], ISO-15693 [ISO00b] and ISO-18000 [ISO04] standard suites.

Many taxonomies are possible to categorize the different RFID tag variants. One that is important for this work is the distinction between battery powered and Electromagnetic (EM) field powered devices. The majority of tags belongs to the second category. This includes all the tag groups that are of primary concern for item identification purposes.

These product identifying tags are the most constrained of all RFID devices. The die area of the digital part of the Integrated Circuit (IC) is a major contributing factor to the cost of a tag. It has to be kept at a minimum. This of course greatly limits the functionality that it is possible to implement on such a device.

Aside from the die area the other limiting factor is the power budget. The power that can be supplied by an EM field is severely restricted. The following deliberations apply to HF-devices using a carrier frequency $f_c$ of 13.56 MHz. According to Feldhofer et al. [FW07a] 15 $\mu A$ is the upper bound to the mean current available. Exceeding this implies reducing the operating range of the tag.

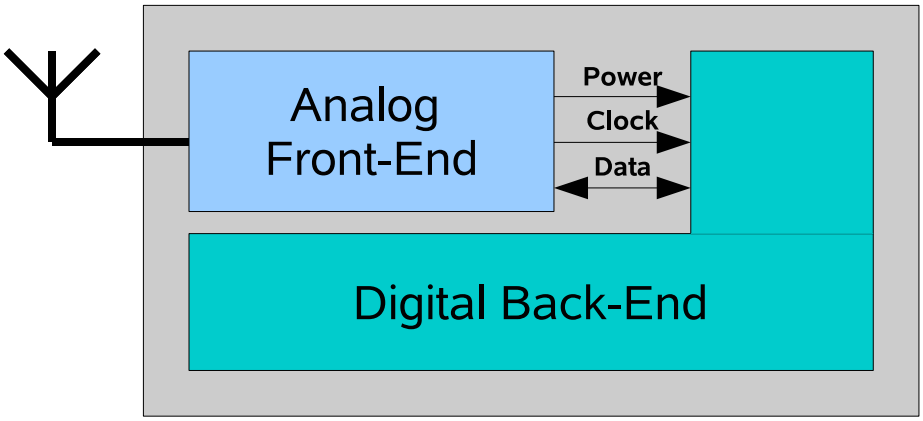



**Figure 2.2:** *Schematic of an RFID tag*

Figure 2.2 depicts a high level schematic of an RFID tag. The antenna is connected to an analog front end. This circuit has three tasks.

**Extract power** from the EM field and provide a stable operating conditions to the integrated circuit.

**Generate a clock** signal. The clock of the digital component is derived from the frequency of the exciting EM field.

**Send and receive data** and act as interface unit for the digital back-end. This part is concerned with the physical details of signal modulation and demodulation.

The digital back-end implements the communication protocol and the higher level functionality of the tag. It processes the commands sent by an interrogator and generates appropriate responses. The bare bone function is to send the Identity (ID) of the tag on request. The ID is stored in a Non Volatile Random Access Memory (NVRAM). More complex devices include commands to read and write data to and from this memory. The idea behind this thesis is to add an ECC module to the digital circuit and extend the communications protocol with cryptographic capabilities.

## 2.2 RFID is poised to supersede the barcode

RFID has two primary advantages compared to a barcode system. The first stems from the radio based communication. With the exception of highly controlled environments barcode scanning requires human intervention. RFID readers, sometimes called interrogators, are not hampered by this limitation. Scanning even hundreds of tags in fast succession is not an issue. Interrogators are impeded by metal structures abundant in todays logistic centers however. This is one reason why RFID technology has not yet completely permeated supply chain management, but it is very likely that this is just a temporary problem.

The second advantage of RFID based item identification protocols is the potential to label every object with its own unique ID. A barcode on the other hand just identifies a product class, a certain type of soft drink for example. The ID of an item in conjecture with a database system facilitates information linkage. Apart from product specific details like the nutrition value of aforementioned soft drink it is also possible to store the transaction history of the beverage back to its bottling. As a consequence massive changes to todays logistics database systems are necessary to take full advantage of RFID technology. Another hampering factor that will vanish over time.

Both advantages in combination offer the potential to revolutionize stock keeping. Reduced storage area and diminished losses due to avoidance of empty shelfs in shops turn the application of RFID technology into hard cash. Other benefits arise from the enhancement of basic identification capabilities with advanced sensor systems. Cool chain monitoring, tire pressure surveillance or constant building stress observation through sensors embedded in the concrete are just some examples that spring to mind.

Returning to the simple item identification tags one should note that these devices do not cease to exist when a product is put into a shopping cart. An intelligent shopping card or a smart phone with an RFID reader function could

automatically compare the contents of the cart with a preconceived shopping list. At the check out all items are billed to an RFID enhanced credit card with a minimum of human interaction.

Even this cursory survey of just a small subset of possible applications for RFID technology reveals a very positive outlook for the future. This is also supported by current sale projections for RFID tags, which anticipate that they will proliferate into the billions within the next few years.

## 2.3 RFID privacy issues

As with every technology there is a distinct downside to the RFID concept. Tags for supply chain and sales applications as mentioned above would adhere to the EPC standard suite, whose development is controlled by the EPCglobal Inc.. This corporation is a joint venture of the regulation bodies that are responsible for the barcode supervision. The particular standards are understandably optimized for automated reader tag interaction and a minimalistic tag functionality. The later limits tag fabrication costs, while the former is at the heart of the whole RFID paradigm.

As is so often the case when security is involved, the goals ease of use and cost minimization clash with at least one of the objectives of security engineering. In this case the privacy requirements of the user. Two very real threats to privacy arise from the fact that tags that comply to an EPC standard[1], responds to an RFID reader without user notification. These are clandestine tracking and inventorying [Jue06].

**Clandestine tracking** As each tag carries an unique identifier and answers freely to interrogators, tracking a subject carrying at least one tag becomes exceedingly simple as long as the target is within range of a reader device.

Spy movie scenarios set aside, tracking customers might be interesting for shops. An RFID enabled establishment that monitors shelf content for stock keeping reasons, already has all the infrastructure in place. The information, how much time a customer spends in front of a certain shelf, is actually usable for marketing purposes. Furthermore, if a credit card is used for payment in such a shop, it becomes possible to link the gathered information to the name of the tracked subject.

This form of privacy attack might even work if each RFID tag of a target is secured against tracking, but the collection of all tags in the vicinity of a victim leaks some form of uniquely identifiable information.

**Clandestine inventorying** RFID technology serves to identify entities and objects in particular. This necessitates a link between the information an RFID tag transmits to an interrogator and the item it identifies. In clandestine scanning this is used to profile a victim.

The information revealed thus could be perceived as harmless, and the subject might very well reveal it freely on request like his shoe size and which kind of clothing brands he prefers. Recalling the afore mentioned example of an RFID enabled credit card and extending it with similarly

[1] Or any equivalent standard for that matter.

enhanced shop bonus cards the disclosed information becomes more critical. Other even more threatening scenarios can easily be devised.

Currently, RFID reader devices are still rare and also quite expensive, but in all probability this situation will change in the future. Ambitions exist to integrate Near Field Communication (NFC) capabilities into mobile phones. The NFC communication standard is compatible with some of the more prominent RFID communication standards. Thus, cheap RFID readers become easily available to everyone, albeit in this case with a very limited range.

The RFID privacy challenge has and still is a very active field of research. Numerous schemes have been devised over the past few years to protect the privacy of the user in an RFID permeated world. Solutions range from simple but brutal tag kill commands to complex schemes using proxy devices and jamming. Many of them rely on cryptographic primitives of varying complexity. Even a brief overview of employed methodologies would go beyond the scope of this introduction. For a detailed survey confer to [Jue06] and for recent research on minimalist cryptography for RFID privacy see [LLM07].

## 2.4 RFID based authentication

Product counterfeiting becomes an increasing problem in todays brand driven markets. There exists a veritable counterfeiting industry copying products of all kinds. While its more amateur outgrowths like Adibas shoes and Naik bags might be considered funny, its professional wing causes great financial harm. If for example medicines are involved product piracy becomes outright dangerous.

Often counterfeits enter the regular distribution channels of genuine products and are thereafter very difficult to differentiate from their genuine counterparts. Automated means to secure these licit supply chains against pirated products have the potential to permanently stop product forgeries.

This is where RFID comes into play. It provides automated identification of items. It is already adopted in supply chains for this virtue. It seems to practically lend itself to product authentication. There is one condition however. Businesses are ultimately profit driven. The cost of a measure to prevent counterfeiting must be smaller than the losses incurred by product piracy.

It is important in this context to distinguish between identification and authentication. The later is similar to the first but also provides corroborative evidence to confirm the claimed ID. For more detailed information confer to §3.4.

Sound cryptography based authentication protocols exist, but the area and power budget constraints in an RFID environment complicate the implementation of hard cryptographic primitives. As a consequence the field of research for alternative authentication schemes has flourished over the past few years. New paradigms for product authentication that do not depend on cryptography were introduced. Also many procedures using only lightweight cryptography, cheaper in terms of die area an power consumption but less secure and often only of academic interest, arose. For a current survey on the topic of RFID product authentication techniques confer to [LSMF06].

In [LLMF07] Lehtonen et al. give an in-depth analysis of trust and security in RFID-based product authentication systems. Among other points of interest tag-cloning, the copying of RFID tags to bestow authenticity to counterfeit

products is revealed as the most probable point of attack on a supply chain. Authentication schemes based on sound cryptography have the inherent advantage of high tag cloning resistance.

Sound cryptographic primitives are pitted against very powerful attacker models. According to [LLMF07] these models are only partially applicable in an RFID enhanced supply chain with the emphasis that a real attacker would be much more limited in his options.

Furthermore the assurance granted by authentication protocols based on hard cryptography is not absolutely required. For a real application a certain level of confidence that a product is genuine might be sufficient. This concept is also formalized by Lehtonen et al. in [LLMF07]. Even the RFID inherent property of item unique IDs already affords a certain protection against product piracy and might be sufficient.

That does not disqualify cryptographic based authentication schemes. They are still potentially the most secure variant and the intrinsic tag cloning resistance is another point in their favor. Yet another advantage that is specific to public key cryptography solutions is that they are relatively easy to deploy and maintain, as little cooperation between different participants in the supply chain is required.

Those alternate methods arose from a need to find a feasible substitute for asymmetric cryptographic primitives. This just emphasizes the point that hard cryptographic primitives have to become smaller and thus cheaper while at the same time require less power.

Elliptic Curve Cryptography is the only public-key cryptosystem that has a chance to fulfill these requirements. Its comparatively small key sizes and the resulting minuscule chip area for hardware implementations makes it the ideal candidate.

Chapter 3

# A brief introduction to cryptography

Cryptography is a vast and complex topic. This chapter only touches on the issues necessary to follow the design decisions made herein. One of its intention is to define what cryptography is and to explain where its applications lie. Public-key cryptography needs to be addressed as to rationalize its employment for digital signature generation which in turn is the tool of choice for entity authentication the concept at the core of this work.

This chapter tries to give a short introduction about the subject of cryptography in general in §3.1 and then briefly immerses the reader in the details of the key-distribution problem (§3.2) and how public-key cryptography (§3.3) can be applied to alleviate it. It continues to give a concise overview of authentication and the use of challenge-response protocols in §3.4. Finally, Digital Signature Algorithms (DSAs) and their applicability in challenge-response protocols is discussed in §3.5.

## 3.1 Cryptography - Definition and goals

The Handbook of Applied Cryptography [MvOV01] defines cryptography as the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. It further stipulates that cryptography is not the only means of providing information security, but rather a set of techniques.

Information security and cryptography are interwoven with antics of three entities: Alice, Bob and Eve. It seems impossible to detail the concepts of information security without these three and it would be a stark omission not to mention them here.

Alice and Bob want to talk. They usually do so over an insecure communications channel and depending on the details of the scenario with optional high background noise.

Specifically Alice and Bob want to

- talk in private, without being overheard

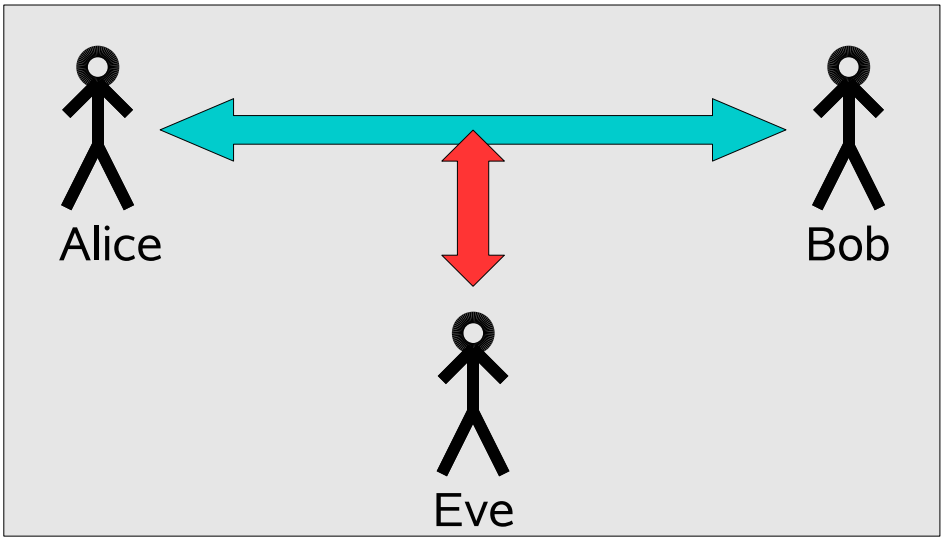



**Figure 3.1:** *Alice, Bob and Eve*

- be certain that what they hear is what the other said, and not garbled by background noise
- know for sure that Alice is Alice and Bob is Bob and not someone else
- ascertain that what one hears originated with the other party and not some hidden ventriloquist

Eve on the other hand wants to listen in, mutilate the content of the communication between Alice and Bob, alternatively impersonate Alice or Bob and intersperse bogus messages that the recipient beliefs have originated with the authorized conversation partner.

Eve is very powerful and it is generally understood that she is has full access to the communication channel used by Alice and Bob. Thus, she is capable to perform the above mentioned malicious actions. The goal of cryptography is to prevent Eve from doing so in spite of her facilities.

Although there has been some speculation on the nature of Alice, Bob and Eve [Gor84], introducing them as entities was deliberate. In the following considerations they represent actors in a communication scenario. This of course encompasses flesh and blood people but also their virtual agents in todays information networks. Alice might be a human who wants to communicate with Bob, but could as well be a smart card authenticating itself to an ATM or an Email application sending a message. Alice et al. will consequently help to provide descriptive examples for the aspects of information security.

### 3.1.1 Goals of Cryptography

**Confidentiality** concerns itself with the protection of data from eavesdropping by illicit third parties. One way to achieve this is to use encryption. Alice

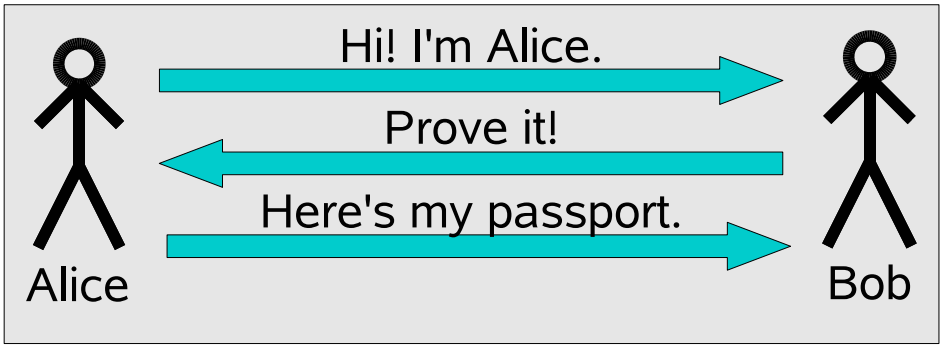



**Figure 3.2:** *Bob challenges Alice*

encrypts a message before she sends it to Bob. Even if Eve intercepts the data, without knowledge on how to decrypt it will be impossible for her to discover its contents. Bob on the other hand has this additional knowledge and thus has access to the content of the message.

**Data integrity** tries to protect the content of messages against accidental or unauthorized modification. Eve might not just be content with reading messages passed between Alice and Bob, she could go a step further and try to alter the data with malicious intend. Multiply Accumulate (MAC) are the cryptographic tools of choice to achieve this goal.

**Entity authentication** has the goal to ascertain the identity of one party to another. In information security it is important to differentiate between identification and authentication. The difference is best explained by an example:

Alice and Bob meet. If Alice claims to be Alice to Bob she identifies herself. To authenticate herself to Bob she needs corroborating evidence. In this example Alice could show adequate photo identification, issued by a third party that Bob trusts.

The corresponding attack by Eve is impersonation, whereas Eve attempts to persuade Alice or Bob that she is the respective other. The cryptographic counter approach is to use challenge-response authentication protocols. Confer to figure 3.2 for a simple illustration of the concept.

**Data origin authentication** ensures that a messages originates from the claimant source. If Eve is not capable of impersonating an authorized communication party, she might be able to intersperse messages into the communication, that the recipient beliefs to have originated from a valid source. Eve could send information to Bob claiming that it originated from Alice. Cryptography applies a technique called keyed-hash functions to combat this hazard.

### 3.1.2 Kerckhoffs' Principle

Kerckhoffs' principle states that the security of a cryptosystem should depend solely on the key that is used. Even if every detail of the system but the key is public knowledge, it should still be secure [Ker83].

In contrast "security through obscurity" hides the intrinsics of a cryptosystem. This is necessary because in this paradigm the security of the system is linked to its inner workings.

The second concept offers a greater point of attack, as the whole system has to be kept secret, which makes it more vulnerable. One example for the catastrophic failure of this paradigm is the repeated leakage of the source code of the Diebold electronic voting machines.

## 3.2 The key distribution problem

There exists several different types of cryptographic primitives to achieve the goals described in §3.1.1. Unkeyed primitives, symmetric-key cryptography and public-key cryptography are distinguished. Of these three only symmetric- and public-key cryptography primitives are serious candidates for challenge-response authentication. Unkeyed primitives are therefore not discussed herein.

### 3.2.1 Symmetric-key cryptography

The name symmetric-key cryptography derives from the fact, that in this cryptographic approach the same key is used for all cryptographic operations. En- and decryption is a good example. An encryption function $E_k$ uses the secret key $k$ to encrypt a message $m$ into the ciphertext $c$. The decryption function $D_k$ applies the same key to decrypt the ciphertext. Equation 3.1 further illustrates this principle. The cryptographic primitive that performs these operations is called a cipher.

$$\begin{aligned} c &= E_k(m) \\ m &= D_k(c) \end{aligned} \tag{3.1}$$

The advantage of the symmetric technique is the efficiency of implementation of its primitive operations. Feldhofer et al. [FW07a] compare different low power hardware implementations of cryptographic primitives of all three families. Not surprising the Advanced Encryption Standard (AES), the de-facto standard symmetric-key cipher, proved to be the most efficient in terms of chip area usage and power consumption.

### 3.2.2 The problem with symmetric-key cryptography

This poses the question why to use ECC, if symmetric-key cryptography seems to be best suited for usage in an RFID application. The reason for this is the so called key-distribution problem.

To be secure, the symmetric approach necessitates that every party participating in a cryptographic protocol needs a unique, authentic secret key for every other participant. The problem is again best illustrated by an example.

Alice and Bob have a common friend Charlie. Alice wants to write Bob a message and make sure that Charlie cannot read it, because it concerns his

birthday present. Alice needs to have a shared key with Bob. The same is true if Alice wanted to discuss Bob's birthday present with Charlie. The subject gets really interesting, when Alice, Bob and Charlie want to exchange messages without Eve being able to eavesdrop. An extra group key could solve this conundrum.

Two problems become immediately apparent.

1. How to exchange the keys in the first place?
2. What happens if there are not just three participants but thousands?

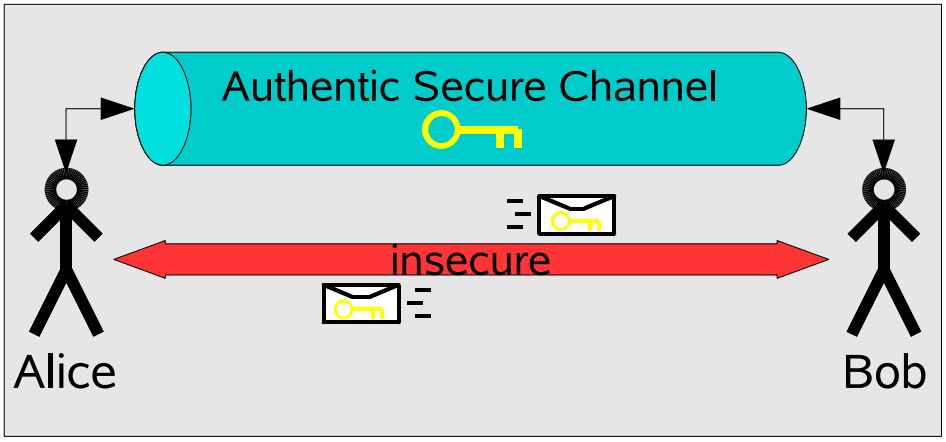



**Figure 3.3:** *Symmetric-key cryptography key exchange*

In the above example the solution is simple. Alice, Bob and Charlie know each other and can meet face to face to exchange their keys. As they know each other it is easy for them to visually authenticate their friends and they can make sure that nobody eavesdrops the keys as they exchange them. The keys are thus exchanged over an authentic and secure (against eavesdropping) channel, cf. figure 3.3.

Todays applications are not that simple. What happens if Alice is a customer and Bob is an Internet based shopping mall. In all probability thousands of kilometers separate Alice and the office of "Bob" and they have never met and very probably never will. In this case it would not be possible to personally exchange keys.

### 3.2.3 Trusted Third Partys

A Trusted Third Party (TTP) is a concept that helps to alleviate part of the problem. A TTP is a trusted agent that handles key exchange on behalf of clients, comparable to an escrow.

Charlie is a TTP to Alice and Bob. Alice hands the key for Bob over to Charlie using the key exchanged with Charlie to secure the communication. Charlie in turn employs the secret agreed upon with Bob to impede eavesdropping and authenticate himself, while he transmits Alices' key to him.

Thus, TTPs greatly facilitate the key exchange, but they do not solve the underlying key-distribution problem. Only through the introduction of public-key cryptography a practical solution becomes available.

## 3.3 Public-key cryptography

As ECC is a form of public-key or asymmetric cryptography and ECC is propagated as a solution to the RFID tag authentication challenge it warrants more than just the cursory explanations spared for the other cryptographic primitives.

Public-key cryptography utilizes not just one secret key, but a pair of keys. They are distinguished as the public- and the private key. The public key can and should be freely shared with all participants in an cryptographic protocol. The private key must be kept secret as the security of the cryptosystem depends solely on the private key.

Public-key cryptography primitives utilize trapdoor one-way functions. These functions are easy to compute in one direction, but close to impossible to solve in the other direction.

At this point the trapdoor part comes into play. It is intractable to compute the inverse of such a function, unless an additional information, the trapdoor, is known. In that case, the calculation of the inverse becomes reasonably simple.

Currently there are three prominent mathematical problems that serve as foundation for trapdoor one-way functions employed by todays public-key cryptosystems.

**Integer Factorization Problem (IFP)** is the mathematical base for RSA public-key cryptography primitives. The IFP is defined in the following way: Given an integer composite number, find the factors that multiplied together again yield that integer. In case that the composite number is large enough, there exists no algorithm that can solve the problem in a reasonable time frame. It is computationally infeasible, unless of course a sufficient number of factors is already known.

**Discrete Logarithm Problem (DLP)** supplies its hardness to the Diffie-Hellman key agreement protocol as well to the ElGamal cryptographic operations. Without going too deep into the number theoretic background, the problem in brief is: Given an equation of the form $y = g^x \, mod \, p$ finding $x$, when $y, g$ and $p$ are known.

**Elliptic Curve Discrete Logarithm Problem (ECDLP)** is a transformation of the DLP onto an elliptic curve. It is essential for the elliptic curve en-/decryption schemes and more relevant to this work the elliptic curve digital signature algorithm. The intrinsics of ECC will be detailed in chapter §4.

For a more detailed and mathematical exhaustive specification of both the IFP and DLP please refer to a cryptographic reference book like the Handbook of Applied Cryptography [MvOV01].

### 3.3.1 Asymmetric cryptography - An example

The concept of public-key cryptography is best illustrated by examining an encryption/decryption process. Alice needs to send a message to Bob and she wants to make sure that no third party is able to read it. She first acquires Bobs public key $Pub_B$ and uses it to encrypt her message $m$. The ciphertext $c$ of the encrypted message is sent to Bob. Bob in turn decrypts $c$ with his private key $Priv_B$.

$$\begin{aligned} c &= E_{Pub_B}(m) \\ m &= D_{Priv_B}(c) \end{aligned} \tag{3.2}$$

Knowledge of the public key does not allow decryption of the message. Even if $c$, $E$, $Pub_B$ and $m$ are known it should be infeasible to find $Priv_B$, as it should be intractable to compute $m$ if an attacker is in possession of $c$, $E$ and $Pub_B$.

### 3.3.2 Key distribution

The fact that every part of a public-key cryptosystem but the private key can be made public, has a fundamental effect on key distribution. Unlike in symmetric systems the total number of keys is restricted to the number of participants ($n$). Table 3.1 compares the symmetric, symmetric with Trusted Third Party and public-key cryptography approaches with respect to the total number of keys ($k$) required.

| Cryptographic system | Number of required keys |
|---|---|
| Symmetric-key | $k = \frac{n \cdot (n-1)}{2}$ |
| Symmetric-key with TTP | $k = n$ |
| Public-key | $k = n$ |

**Table 3.1:** *A comparison of the number of necessary keys in different cryptosystems*

The second point of interest in conjecture with the key-distribution problem is that asymmetric systems do not necessitate the use of a secure and authentic channel to exchange keys. Authenticity is still imperative, but as the public key is not secret it is not necessary to protect against eavesdropping. Figure 3.4 depicts the key exchange process for a public-key cryptosystem.

Providing authenticity to a key exchange is still a formidable challenge taken up by so called Public-Key Infrastructures (PKIs). They use cryptographically secured digital certificates to lend credibility to public keys distributed through a network, most prominently the Internet. Discussing these issues warrants its own chapter and is beyond the scope of this work. It is sufficient to note that, public-key cryptography provides the best known solution to the key exchange problem.

In an RFID supply chain scenario, where the number of participating parties may be equal to the number of RFID tags involved, key management is a critical issue. For this reason asymmetric cryptography is also the theoretical most suitable facility for this field of application.

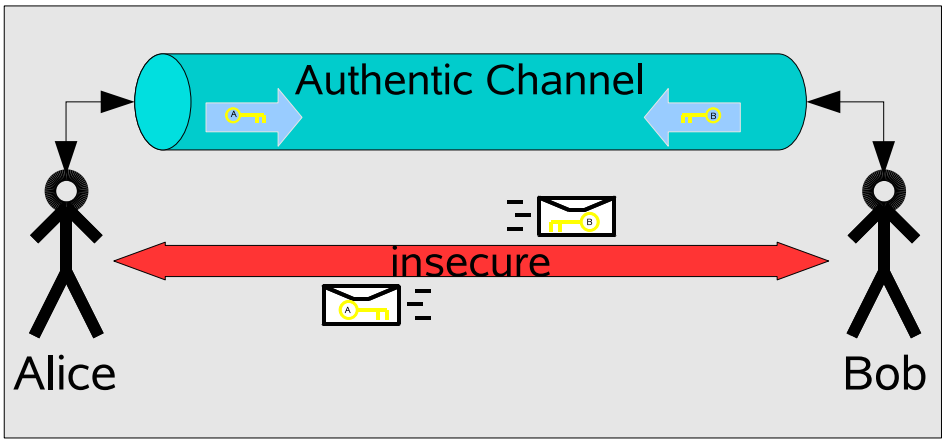



**Figure 3.4:** *Key exchange in a public-key cryptosystem*

## 3.4 Challenge-response authentication

So far no clear means how cryptography can help to proof the identity of an entity have been established. This is partially due to the fact that confidentiality provides more practical examples to illustrate cryptographic concepts. It is time to remedy this omission.

As mentioned before the difference between identification and authentication lies in providing corroborative evidence. There are three paradigms how authenticity can be assured. These are based on:

**Something known.** A person draws close to a door in a dimly lit back alley and knocks. A small viewing window opens up, a pair of suspicious eyes glares at the late night visitor and asks in a deep rumbling voice: “Password?”.

This is an exaggerated example of password based authentication, but it illustrates the point. This kind of authentication depends on the ability to convince the authenticator that the subject knows a secret. This is not quite the same as revealing the information. Cryptographic methods belong to this class.

**Something possessed.** The prime example for this is a passport. The object or person authenticating itself has an additional object to corroborate its identity.

**Something the claimant is.** The idea behind this is that an entity has an intrinsic property that uniquely identifies it and hopefully is not duplicable. Biometric authentication methods fall into this category. For objects or an object class it could be the specific resistance between two points on its surface at a certain voltage.

The principle of challenge-response has already been outlined in figure 3.2. In this case for an authentication process based upon something the authenticating

subject has. The same procedure can be translated to the "something known" paradigm using cryptography.

The authenticator sends a message, the challenge, to the authenticating entity. The receiver, the claimant, uses a cryptographic primitive to perform an operation on the message and transmits the result the validating party. The challenger then either performs the same operation on the original message and compares the result to the received one, this works for shared secret systems, or the received data is verified with the public part of a secret, in case of asymmetric cryptography.

A simple challenge response protocol based on symmetric-key cryptography works like demonstrated by equation 3.3. $A$ and $B$ consistently denominate Alice and Bob respectively in all of the following examples. $E_k$ is a symmetric key encryption and $D_k$ the corresponding decryption function.

$$\begin{aligned} A \rightarrow B: &\quad n \\ A \leftarrow B: &\quad c = E_k(n) \\ A: &\quad E_k(n) = c? \end{aligned} \tag{3.3}$$

Eve wants to attack the authentication. The goal is the impersonation of Bob to Alice. The security of the above protocol greatly depends on the nature of $n$. If $n$ is the same in every protocol execution, in the face of Eve's vast capabilities (cf. §3.1), the procedure is insecure.

The encryption function $E_k(n)$ will always yield the same result if $n$ does not change. Eve has the ability to eavesdrop on the communication between the licit parties. Thus she can easily impersonate Bob by replaying his response to Alice. This attack is called a replay attack. Protocols that are immune against this scheme are termed strong authentication protocols, otherwise they are called weak. The password based authentication that served as an example for the "something known" paradigm belongs to the weak category.

The easiest way to secure the above protocol against this form of attack is to let $n$ be an integer number and to increment it every time it is used. It is important that every $n$ is used only once. Such a number is called a nonce[1].

There exist other more complex forms of deception (cf. to [MvOV01]) and the determination of the security of a protocol against them is a difficult and error prone process. It is therefore advisable to employ proven standards that have undergone rigorous examination by specialists.

## 3.5 Digital Signature Algorithms

Challenge-response authentication protocols that use public-key cryptosystems are more complex then their symmetric counterparts and often rely on a cryptographic primitive called digital signatures.

Signature algorithms allow to create a digital signature that does not only depend on the private key of the signer, but also on the content of the message signed. This linkage of signature and data distinguishes it from its real life equivalent.

To create signatures for arbitrary length messages another cryptographic primitive becomes necessary, the hash function (*hash*). Hash functions are a

[1] Possibly from number used once or number used only once.

subgroup of the aforementioned unkeyed primitives. There sole input is a data stream of arbitrary length $m$. The input is mapped to a finite set of values. Thus there are an infinite number of input message texts that map to the same output value $h$.

It is a required property of a cryptographic hash functions that for any given input text, it is computational infeasible to find a second message that generates the same result $h = hash(m_1) = hash(m_2)$. This is called collision resistance.

The use of public and private key is reversed for signature generation and verification compared to en- and decryption. A signature generation algorithm uses the private key of the signer to create the result, while the verifier uses the corresponding public key to check the signature.

This is important, because only the knowledge of the private key enables signature fabrication, which concords with Kerckhoffs' principle and the advantages of asymmetric cryptography with respect to key distribution hold.

### 3.5.1 The use of digital signature algorithms in challenge-response protocols

Having introduced signature algorithms, the final ingredient to present public-key based challenge-response protocols that can be used for entity authentication is in place.

A variant of the Station-to-Station (STS) protocol witch is limited to entity authentication will serve as an example for such a procedure. The ECDSA an elliptic curve based DSA that is implemented by the processor presented in this thesis will be discussed in the next chapter (§4.1).

The authentication only STS protocol [DvOW92] is a reasonably simple but secure challenge-response protocol. It requires a random number generator $G$ which serves as nonce creator and a signature algorithm $S_k$, whereas $k$ is the public key of the signer. The $|$ denominates the concatenation operator in this context.

$$\begin{aligned} A \rightarrow B: &\quad x = G_A() \\ A \leftarrow B: &\quad y = G_B(),\ s_1 = S_B(y|x) \\ A \rightarrow B: &\quad s_2 = S_A(x|y) \end{aligned} \tag{3.4}$$

Alice generates a random number $x$ and sends it to Bob. He in turn also creates another nonce $y$ and concatenates $y$ and $x$. Next Bob computes the signature of $y|x$ and sends both $y$ and $s_1$ to Alice. She uses her private key to sign $x$ concatenated with $y$ and transmits the result $s_2$ to the other participant. Both parties then use the public key of the other to verify the exchanged signatures. In case the signature validation is successful, mutual authentication is established. Bob is assured that he is communicating with Alice and vice versa.

The STS protocol guarantees strong authentication as the probability for arriving at the same pair of nonces can be held reasonably small by making the image set of the random number generator large enough.

RFID tag authentication only requires that the tag is capable to prove its authenticity to the interrogator. If Alice represents the RFID reading device and Bob the tag in equation 3.4 the protocol run could be aborted after "Bob" sends his response to the RFID reader. This has the downside that the tag has no guarantee that it talks with a legitimate interrogator, but on the upside it removes the necessity for implementing the signature verification procedure on the tag.

Chapter 4

# Elliptic Curve Cryptography

ECC is a huge field and many different variants of cryptography based on Elliptic Curves (ECs) exist. It does not provide a single best solution for all cryptographic problems but a wide range of possibilities. This makes ECC highly versatile as it allows customization of a variety of parameters to find an optimal solution for a specific application.

The application outlined in chapter §1 is entity authentication for RFID tags. In §3 it was ascertained that public-key cryptography based on ECC primitives is an appropriate choice in this case. Next the ideal set of ECC parameters, that allow the implementation of an integrated circuit with optimized die area and minute power consumption, must be found.

It has already been determined in §3.5.1 that the interesting authentication schemes depend on Digital Signature Algorithms. The most prominent ECC based DSA is the ECDSA introduced in §4.1. Therein the basic ECC operation the *point multiplication* is presented. This is the function the *ECCon* ASIC designed in this thesis has to implement.

Section §4.2 finally introduces Elliptic Curves and reveals that more than one possible curve to choose from exists. The structure of the curve has to be adapted to the underlying field, thus the choice of curve goes hand in hand with the selection of the underlying field. The efficiency of an ECC primitive implementation is highly dependent on this decision made in §4.2.1.

Further discussion of EC arithmetic necessitates a brief review of finite fields in general and of binary extension fields more specifically in §4.3. Finally, section §4.4 defines an arithmetic for the chosen EC and details an adequate point multiplication algorithm, which concludes this section.

A last remark: Many of the definitions and algorithms presented in this chapter are either directly taken or adapted from Hankerson et al. [HMV04]. Their "Guide to Elliptic Curve Cryptography" is a comprehensive resource for everything related to ECC and this part relies on it as a primary source of information. References to this book will therefore not be cited explicitly throughout this chapter, as this would only impede its readability. All other sources are cited as needed.

## 4.1 The Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a standardized digital-signature creation and verification method. It has been widely adopted into standards by several organizations including the IEEE, the NIST and the ANSI. The FIPS-186-2 [Fed00] published by the NIST is a specific example.

The ECDSA procedures require an elliptic curve cryptographic primitive and a collision resistant hash function $H$. The parameters $P$ and $n$ are the base point, a special point on the elliptic curve and the order of that point, respectively. Their exact meaning will be specified in detail later in this chapter. The only ECC operation required for signature generation is the so called *point multiplication* $k \cdot P$ in step 2 of algorithm 4.1. The *point multiplication* is also denoted by the term *scalar multiplication*. Those two names are used synonymously throughout this text. The parameter $k$ is called the ephemeral key.

***Algorithm 4.1***: *ECDSA signature generation*

**Input**: $P$, $n$, private key $d$, message $m$.
**Output**: Signature $(r, s)$
1 Select $k \in_R [1, n-1]$;
2 Compute $kP = (x_1, y_1)$ and convert $x_1$ to an integer $\overline{x_1}$;
3 Compute $r = \overline{x_1} \, mod \, n$;
4 **if** $r = 0$ **then**
5 Goto step 1;
6 Compute $e = H(m)$;
7 Compute $s = k^{-1}(e + dr) \, mod \, n$;
8 **if** $s = 0$ **then**
9 Goto step 1;
10 **return** $(r, s)$;

The validation procedure detailed in algorithm 4.2 requires an additional parameter. The public key $Q$ is a point on the elliptic curve and is defined as $Q = d \cdot P$, where $d$ is the private key. The verification is given for completeness sake and to detail how and why signature evaluation works. It has no real relevance in an RFID tag authentication scenario which does not require signature validation on the tag side.

The reason why algorithm 4.2 works is, that given a genuine signature $s = k^1(e + dr) \, mod \, n$, rearranging of this formula gives

$$\begin{aligned} s &= k^{-1}(e + dr) \, mod \, n \\ k &\equiv s^{-1}e + e^{-1}r \\ k &\equiv we + wrd \\ k &\equiv u_1 + u_2 d \, mod \, n \end{aligned} \tag{4.1}$$

and multiplying the last line with $P$ yields equation 4.2.

$$kP = (u_1 + u_2 d)P = u_1 P + u_2 Q = X \tag{4.2}$$

***Algorithm 4.2**: ECDSA signature verification*

**Input**: ECC domain parameters, public key $Q$, message $m$, signature $(r, s)$.
**Output**: Signature validity: accept or reject

**if** $r, s \notin [1, n-1]$ **then**
  **return** *reject*;
Compute $e = H(m)$;
Compute $w = s^{-1} \bmod n$;
Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$;
Compute $X = u_1 P + u_2 Q$;
**if** $X = \infty$ **then**
  **return** *reject*;
Convert the x-coordinate $x_1$ of $X$ to an integer $\overline{x_1}$; compute $v = \overline{x_1} \bmod n$;
**if** $v = r$ **then**
  **return** *accept*
**else**
  **return** *reject*

Thus $v = r$, if $s$ was generated by the legitimate signer.

Under certain conditions, depending on the security of the hash function and the underlying elliptic curve, as well as the randomness of the parameter $k$ the ECDSA is so far assumed to be secure. This and its standardized nature make it an suitable candidate for use in an authentication protocol similar to STS introduced in §3.5.1.

## 4.2 Definition of an elliptic curve

Elliptic curves have been a field of interest for mathematicians for more than a hundred years. In 1985 their use for cryptography was independently suggested by Neal Koblitz and Victor Miller. ECC and with that the ECDSA draws its hardness from the Elliptic Curve Discrete Logarithm Problem (ECDLP). The ECDLP primarily relies on the non-invertibility of the point multiplication. To fully understand the ECDLP it is first necessary to introduce elliptic curves and then define an arithmetic on them.

**Definition 4.1** *An elliptic curve $E$ over a field $K$ is defined by an equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \mid a_1, a_2, a_3, a_4, a_6 \in K \tag{4.3}$$

*and $\Delta \neq 0$, where $\Delta$ is the discriminant of $E$ and is defined as follows:*

$$\left.\begin{array}{rcl} \Delta & = & -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 & = & a_1^2 + 4a_2 \\ d_4 & = & 2a_4 + a_1a_3 \\ d_6 & = & a_3^2 + 4a_6 \\ d_8 & = & a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{array}\right\} \tag{4.4}$$

*If $L$ is any extension field of $K$, then the set of $L$-rational points on $E$ is*

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\} \tag{4.5}$$

*where $\infty$ is the point at infinity.*

Equation 4.3 is called a Weierstrass equation and E is defined over K ($E|K$) because the coefficients $a_1$, $a_2$, $a_3$, $a_4$ and $a_6$ of $E$ are elements of the underlying field $K$. The condition $\Delta \neq 0$ is responsible for ensuring that there exists no points where the curve has more than one tangent line. The L-rational points are all points $(x,\, y)|\, x\,, y \in L$ that satisfy equation 4.3 $E$.

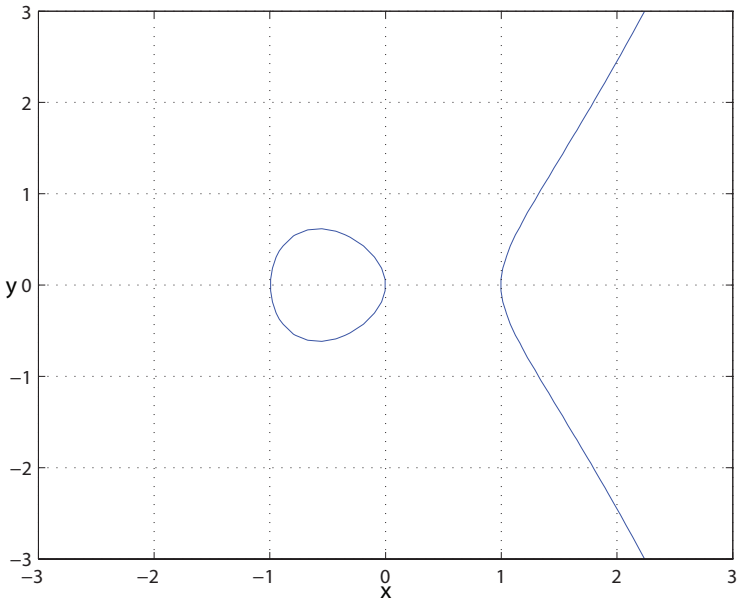



**Figure 4.1:** *Elliptic curve $E : y^2 = x^3 - x$ over $\mathbb{R}$*

### 4.2.1 Simplification and curve selection

It is possible to greatly simplify the Weierstrass equation using a so called admissible change of variables. The curve of interest for this thesis is the non-supersingular curve defined by equation 4.6 derived from equation 4.3 using the admissible change of variables given by equation 4.7.

$$y^2 + xy = x^3 + ax^2 + b \tag{4.6}$$

$$(x,\, y) \rightarrow \left( a_1^2 x + \frac{a_3}{a_1},\, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right) \tag{4.7}$$

Singling out this curve is necessary because there are several valid choices of elliptic curves for cryptographic purposes. The structure of the curve is depending on the underlying field. Equation 4.6 defines a variant where the field $K$ has a characteristic of 2. This makes this curve equation suitable for use with a binary extension field.

Binary extension fields have the great advantage that arithmetics based on them can be implemented very efficiently in hardware. As this work targets RFID applications where fierce restrictions with respect to area usage and power consumption apply, this choice becomes natural.

## 4.3 Binary extension fields

Binary extension fields henceforth also called binary fields, are one of the three most prominent valid choices for cryptographic purposes. The other two are prime fields and optimal extension fields. All three are finite fields.

Before binary fields are discussed in greater detail, an introduction of a mathematical field is in order. The basic building block of a field is the mathematical group. In this case the Abelian group laws are required.

### 4.3.1 Abelian group

**Definition 4.2** *An Abelian group (G, *) consists of a set G with a binary operation *: $G \times G \rightarrow G$ satisfying the following properties:*

*(i) (Associativity)* $a * (b * c) = (a * b) * c$, $\forall\, a, b, c \in G$.

*(ii) (Existence of an identity)* $\exists\, e,\ e \in G |\ a * e = e * a = a$, $\forall\, a \in G$.

*(iii) (Existence of inverses)* $\forall\, a \in G$, $\exists\, b \in G |\ a * b = b * a = e$. *The element b is called the inverse of a.*

*(iv) (Commutativity)* $a * b = b * a$, $\forall\, a, b \in G$.

In layman terms, an Abelian group is nothing else but the definition of an operation and its inverse on a set of numbers $\mathbb{F}$. An identity element is also part of the specification. One such operation, that is widely know is the addition. In this case the identity element is zero, as $a + 0 = a$ and the inverse serves as a base to define the subtraction: $a - a = a + (-a) = 0$ and $a - b = a + (-b)$. In mathematical notation $(\mathbb{F}, +)$ forms an Abelian group with the identity element denoted by zero.

### 4.3.2 Finite fields in general

A field in general is an abstract algebraic structure that provides certain operations on a set of numbers. The most familiar ones are the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$ and the complex numbers $\mathbb{C}$. The set of operations consists of addition, subtraction, multiplication and division.

**Definition 4.3** *A field $\mathbb{F}$ consists of a set $\mathbb{F}$ with two binary operations +,*: $\mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$ that satisfy the following arithmetic properties:*

*(i)* $(\mathbb{F}, +)$ *is an Abelian group with the (additive) identity denominated 0.*

*(ii)* $(\mathbb{F} \setminus \{0\}, \cdot)$ *is an Abelian group with the (multiplicative) identity 1.*

*(iii) The distributive law holds:* $(a + b) \cdot c = a \cdot c + b \cdot c$, $\forall\, a, b, c \in \mathbb{F}$

*If the set of numbers $\mathbb{F}$ is finite, then the field is said to be finite.*

As subtraction is the inverse operation to addition the same is true for the division being the inverse to the multiplication. The division is again defined using the inverse element of the multiplication: $a/a = a * a^{-1} = 1$ and it follows that $a/b = a * b^{-1}$.

The order of a finite field is the number of elements in it. There exists a finite Field $\mathbb{F}$ of order $q$ if and only if $q$ is a prime power ($q = p^m, m \geq 1$), where $p$ is prime. The prime $p$ is called the *characteristic* of $\mathbb{F}$. Finite fields are usually denoted by $\mathbb{F}_p$ (prime fields) or $\mathbb{F}_{2^m}$ (binary fields) to distinguish their nature. Alternatively, finite fields are also known as Galois Fields and therefore the representations $GF(p)$ or $GF(2^m)$ are frequently used.

### 4.3.3 Generators

A generator $g$ is an element of a finite field $\mathbb{F}_q$ that can create all other elements in the field, hence the name, through exponentiation $g^i$, $i \in \{0, \ldots, q-2\}$. This is due to the fact that the nonzero elements form a cyclic group under the multiplication ($\mathbb{F}_q^*$).

### 4.3.4 Prime fields

In case $\mathbb{F}_{q=p^1}$ the field is called a *prime field*. Prime fields are best explained using modular arithmetic. They facilitate the understanding of binary extension fields and prime field arithmetic is also a prerequisite for the ECDSA signature generation, which warrants a brief deliberation of them.

A prime field $\mathbb{F}_p$ is defined by the prime $p$, which is called the modulus of the field. The set of numbers in $\mathbb{F}_p$ is limited to $\{0, 1, \ldots, p-1\}$. All operations in such a field are performed modulo $p$: $(mod\, p)$. If a number is greater than $p$ it is still in $\mathbb{F}$, but not in its basic representation. To obtain that, the number $n$ is divided by $p$. The remainder of the operation $r$ is the base representation and the operation $n \, mod \, p = r$ is called modular reduction.

For a quick example consider the prime field $\mathbb{F}_3$. The addition of two plus two equals four which is congruent to 1 in $\mathbb{F}_3$. Addition, subtraction and multiplication work pretty much the same as in $\mathbb{R}$, apart from the reduction. Division in a prime field on the other hand is surprisingly different. Consider the following: $1 \equiv 2/2 \equiv 2 \cdot 2^{-1} \equiv 2 * 2 \, mod \, 3$. Two is its own multiplicative inverse in $\mathbb{F}_3$.

### 4.3.5 Binary extension fields in detail

If a field has the *characteristic* 2 and $m \geq 2$ it is a called a binary extension field ($\mathbb{F}_{2^m}$). Binary fields can be constructed using different basis representations, where the *polynomial basis representation* is perhaps the most intuitive one[1]. In this representation, elements of $\mathbb{F}_{2^m}$ are *binary polynomials*.

**Definition 4.4** *A binary polynomial is a polynomial of the form*

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_1 z^1 + a_0 \ : a_i \in \{0, 1\}\}$$

*where the coefficients are elements of the prime field $\mathbb{F}_2$. The degree of the polynomial is at most $m - 1$.*

An equivalent to the modulus of prime fields is necessary to define the base representation of polynomials in $\mathbb{F}_{2^m}$. This is the irreducible binary polynomial $f(z)$ of degree $m$. Irreducible polynomials derive their name from the fact, that

[1] $\mathbb{F}_{2^m}$ is interpretable as a vector space

they cannot be factored as a product of binary polynomials each of a degree less than $m$. Finding one efficiently is reasonably simple and they exist for every $m$.

To find the base representation for a binary polynomial $a(z)$ whose degree exceeds $m$, it is necessary to perform the polynomial long division of $a(z)$ by $f(z)$. The residue $r(z)$ is the reduced result. Again this operation is called reduction, only this time modulo $f(z)$.

Binary field arithmetic for polynomial basis representations is similar to the one for polynomials in $\mathbb{R}$, except that the coefficient arithmetic is done modulo 2. The implication of this is that addition is the same as performing an XOR ($\oplus$) operation on coefficients of identical degree. As a consequence addition and subtraction are the same operation. Multiplication in contrast to addition can lead to polynomials with a degree equal to or greater than $m$. Such polynomials necessitate a modular reduction.

Consider the following examples for arithmetic in $\mathbb{F}_{2^3}$ with a irreducible polynomial $f(z) = z^3 + z + 1$.

**Addition** : $(z^2 + z) + (z + 1) \equiv (z^2 + 1) \, mod \, f(z)$

**Multiplication** : $(z^2) \cdot (z^2) \equiv z^4 \, mod \, z^3 + z + 1 \equiv z^2 + z \, mod \, f(z)$

**Inversion** : $(z^2 + z + 1) \cdot z^2 \equiv z^4 + z^3 + z^2 \, mod \, z^3 + z + 1 \equiv 1 \, mod \, f(z)$

## 4.4 ECC arithmetic

To be able to perform the point multiplication on an elliptic curve it is first necessary to define an arithmetic on it. It is well known that a multiplication can be represented by a succession of additions ($3 \cdot 7 = 7 + 7 + 7$).

Translating this to a point multiplication it becomes apparent that an operation to add one point on the EC to itself is required. Many advanced point multiplication algorithms also necessitate the ability to add two different points. Those two functions are termed *point addition* and *point doubling*.

There exists a *chord-and-tangent rule* for closed addition of two points on an EC and together with the set of points $\mathbb{F}_{2^m}$, $E(\mathbb{F}_{2^m})$ forms an Abelian group with respect to addition. The point at infinity serves as identity element.

Both point addition and point doubling are best discussed in geometric context. The following observations are only valid on an EC defined over $\mathbb{R}$, but they illustrate the idea behind the operations. To add two points $P$ and $Q$ to yield a third point on the curve $R$ a line is drawn through $P$ and $Q$. This line will intersect the elliptic curve in a third point. $R$ the sum of $P$ and $Q$ is the reflection of that point about the x-axis. Figure 4.2 details the procedure.

The doubling of a point $P$ is done by creating a tangent in this point. The tangent line will intersect the elliptic curve in yet another point. The point $R = P + P$ is again the reflection of the intersection point about the x-axis. This is illustrated in figure 4.3.

### 4.4.1 Group law

**Definition 4.5** *The elliptic curve* $E/\mathbb{F}_{2^m} : y^2 + xy = x^3 + ax^2 + b$ *forms an Abelian group with respect to the two operations point addition and point doubling, which has the following properties:*

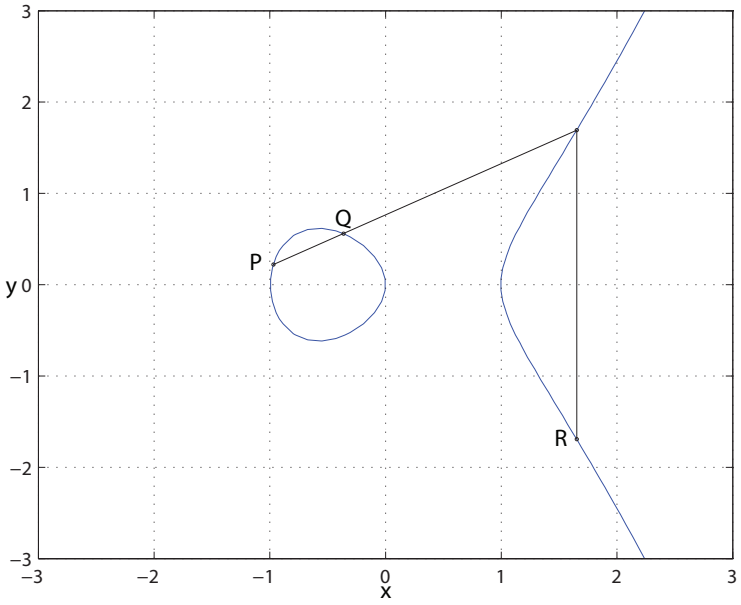



**Figure 4.2:** *Point addition: R=P+Q*

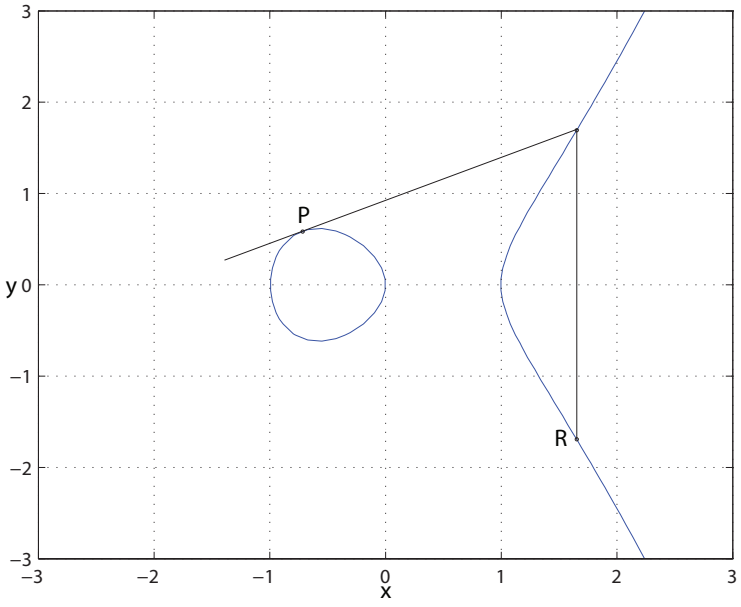



**Figure 4.3:** *Point doubling: R=P+P*

*(i) Identity:* $P + \infty = \infty + P = P, \forall\, P \in E(\mathbb{F}_{2^m})$

*(ii) Negatives: If* $P = (x, y) \in E(\mathbb{F}_{2^m})$*, then* $(x, y)+(x, x+y) = \infty$*. The point* $(x, x + y)$ *is denoted by* $-P$ *and is called the negative of P; note that* $-P$ *is indeed a point in* $E(\mathbb{F}_{2^m})$*. Also* $-\infty = \infty$*.*

*(iii) Point addition: Let* $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ *and* $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$*, where* $P \neq \pm Q$*. Then* $P + Q = (x_3, y_3)$*, where*

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \;\; \textit{and} \;\; y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

*with*

$$\lambda = \frac{(y_1 + y_2)}{(x_1 + x_2)}$$

*(iv) Point doubling: Let* $P = (x_1, x_2) \in E(\mathbb{F}_{2^m})$*, where* $P \neq -P$*. Then* $2P = (x_3, y_3)$*, where*

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \;\; \textit{and} \;\; y_3 = x_1^2 + \lambda x_3 + x_3$$

*with*

$$\lambda = \frac{x_1 + y_1}{x_1}$$

*Point addition and point doubling are performed as a sequence of operations in a finite field, in this case a binary extension field* $\mathbb{F}_{2^m}$*.*

### 4.4.2 Projective coordinates

The two-coordinate representation $(x, y)$ of points in $E(\mathbb{F}_{2^m})$ used so far is termed affine point representation. It is possible to transform the set of *affine points* $\mathbb{A}(K) = (x, y) : x, y \in \mathbb{F}_{2^m}$ with the help of an equivalence relation into the set of projective points $\mathbb{P}(K)^* = (X : Y : Z) : X, Y, Z \in \mathbb{F}_{2^m}, Z \neq 0$

This is a desirable property, because of the inversions necessary to perform the divisions in the point operations. As will become clear later, inversions in $\mathbb{F}_{2^m}$are very time consuming in comparison to the other field operations. There exists coordinate representations that have the advantage of minimizing the number of inversions in $\mathbb{F}_{2^m}$in exchange for an increase of additions and multiplications. The total of the time consumed for the additional operations might be less, than the time needed for the affine variant with its higher number of inversions.

The projective equation for the elliptic curve $y^2 + xy = x^3 + ax^2 + b$ for *standard projective coordinates* (The projective point $(X : Y : Z), Z \neq 0$, corresponds to the affine point (X/Z, Y/Z).) has the form:

$$Y^2Z + XYZ = X^3 + aX^2Z + bZ^3$$

The point at infinity $\infty$ corresponds to $(0 : 1 : 0)$, while the negative of $(X : Y : Z)$ is $(X : X + Y : Z)$.

### 4.4.3 Point multiplication

The point multiplication $Q = kP$ of a point $P$ with a scalar $k$, is the only elliptic curve operation in the ECDSA signature generation algorithm. It is also the most time consuming and complex computation in the whole procedure. This is also true for other EC based cryptographic primitives like the Elliptic Curve Integrated Encryption Scheme (ECIES).

Optimization of this operation is therefore absolutely imperative. As a result thorough research in the field of scalar multiplications was and still is conducted. This has led to a variety of algorithms. They can be categorized into methods that work for an arbitrary $P$ and others that depend on a fixed point. Furthermore, some procedures facilitate precomputation of values, which necessitates additional memory resources, in exchange for greater execution speed.

Memory is one of the main contributing factors to cost of hardware ECC implementations and therefore a scheme that requires precomputation is beyond question. The point $P$, sometimes called the *base point* is a fixed domain parameter in the ECDSA signature creation algorithm, so a fixed point method would be appropriate for this application. Due to the fact that most require precomputation to provide any advantage at all, they can just as easily be dismissed again.

#### Montgomery point ladder

The candidate point multiplication algorithm that was chosen for this master thesis is called *Montgomery's method* or alternatively *Montgomery point ladder*. The procedure works for arbitrary scalars and base points, does not depend on any precomputations in its basic form and despite of that compares very favorable in the performance sector.

It also has a second advantage that makes it the ideal choice for this work. Due to the structure of the algorithm it is inherently immune against Simple Power Analysis (SPA) and timing based side channel attacks.

A side channel attack tries to find the private key used in an cryptosystem, not by tackling the cryptography itself but the implementation. A side channel is an extra source of information like the execution time of an algorithm or the power consumption of the underlying hardware. Side channel attacks are discussed in detail in section §8.8.

The basic structure of this particular scalar multiplication procedure is illustrated by algorithm 4.3. The so called binary method is the ECC equivalent of the square and multiply algorithm for exponentiation. The algorithm works by analyzing the binary string representation of the ephemeral key $k$ denoted by $(k_{l-1}, k_{l-2}, \ldots, k_1, k_0)_2$, where the index $l-1$ identifies the first highest order bit of $(k)_2$ which is not zero.

It is based on an observation by Montgomery, that if $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ denote two point in $E(\mathbb{F}_{2^m})$, where $P_1 \neq P_2$ and $P_3 = P_1 + P_2 = (x_3, y_3)$ and $P_4 = P_1 - P_2 = (x_4, y_4)$, that facilitating the laws defined in §4.4.1 it can be shown that

$$x_3 = x_4 + \frac{x_2}{x_1 + x_2} + \left(\frac{x_2}{x_1 + x_2}\right)^2.$$

This allows the computation of $x_3$ from $x_1, x_2$ and $x_4$. In conjecture with

***Algorithm 4.3***: *Point multiplication, binary method*

**Input**: An integer $k = (k_{l-1}, k_{l-2}, \ldots, k_1, k_0)_2$, where $k_{l-1} = 1$ and a point $P$
**Output**: $Q = kP$

```
1 Set P1 ← P, P2 ← 2P;
2 for i = l − 2 downto 0 do
3     if k_i = 1 then
4         Set P1 ← P1 + P2, P2 ← 2P2;
5     else
6         Set P2 ← P2 + P1, P1 ← 2P1;
7 return (Q = P1);
```

the $y$-coordinate recovery formula

$$y_1 = x^{-1}(x_1 + x)[(x_1 + x)(x_2 + x) + x^2 + y] + y$$

it is possible to derive $Q = (x_q, y_q) = kP$ without need to store or compute a $y$-coordinate until the end of the algorithm. ECDSA signature generation does not even require the $y_q$-coordinate, so its restoration is not necessary.

Montgomery's method works by sustaining the loop invariant $P_2 - P_1 = P$. Maintaining the invariant is important because $P_4 = P_2 - P_1 = P \Rightarrow x_4 = x$. Also in every step only the $x$-coordinate of $k_iP$, where $i$ is the integer represented by the $i$ leftmost bits of $k$, is computed.

An efficient version of the Montgomery point ladder for $\mathbb{F}_{2^m}$ is due to López and Dahab [LD99]. They provide both an affine and a projective variant of their algorithm. As inversions are very expensive on the target hardware, the projective version was chosen and is presented by algorithm 4.4.

The first step of the scalar multiplication algorithm is to check for specific input constellations (zero input) that terminate the execution immediately. Next the two start points $P_1 = P$ and $P_2 = 2P$ are initialized. Only the $x$-coordinates of the $P_1$ and $P_2$ are necessary and stored in standard projective form $X_1/Z_1$ and $X_2/Z_2$.

The main body of the procedure then performs the double and add algorithm, only ever computing the $x$-coordinate of the points involved. After the output of the main loop is tested for certain exit conditions (division by zero), the projective $x$-coordinate is transformed into an affine result and returned.

The functions *Madd* (algorithm 4.5) and *Mdouble* (algorithm 4.6) are responsible for the computation of the point addition and point doubling operations respectively. They expect the input in projective coordinates and deliver the output likewise in projective coordinates. output. The doubling procedure is sped up using a precomputed value $c$, which is derived from the EC domain parameter $b$. As $b$ is a parameter of the EC, it does link the implementation to a certain EC, but $k$ and $P$ can still be chosen arbitrarily.

Algorithm 4.5 requires four multiplications, two additions and one squaring operation in $\mathbb{F}_{2^m}$, as well as one temporary variable $T_1$. The point doubling procedure needs only two multiplications, four squaring runs and one addition. It too necessitates one temporary variable.

***Algorithm 4.4***: *Projective Montgomery point ladder*

**Input**: An integer $k = (k_{l-1}, k_{l-2}, \ldots, k_1, k_0)_2$, where $k_{l-1} = 1$ and a point $P = (x, y) \in E(\mathbb{F}_{2^m})$
**Output**: $x_q$, $Q = (x_q, y_q) = kP$

```
1  if k = 0 or x = 0 then
2      return x_q = 0
3  Set X_1 ← x, Z_1 ← 1, X_2 ← x^4 + b, Z_2 ← x^2;
4  for i = l − 2 downto 0 do
5      if k_i = 1 then
6          Madd(X_1, Z_1, X_2, Z_2), Mdouble(X_2, Z_2);
7      else
8          Madd(X_2, Z_2, X_1, Z_1), Mdouble(X_1, Z_1);
9  if Z_1 = 0 then
10     return x_q = 0;
11 if Z_2 = 0 then
12     return x_q = x;
13 Z_2 = Z_1^{-1};
14 X_1 = X_1 · Z_2;
15 return x_q = X_1;
```

The whole point multiplication has a runtime of approximately

$$6\lfloor log_2(k) \rfloor M + (1I + 1M).$$

$M$ denotes a multiplication and $I$ an inversion. One can see that only one inversion is required.

Additions and square operations require only a fraction of the runtime of a multiplication. It is therefore save to disregard them for a first order approximation runtime analysis.

In case of a fixed curve and a fixed base point but a random $k$, the algorithm needs exactly 6 variables ($k, X_1, X_2, Z_1, Z_2$ and $T_1$), and four constants ($b, x, y$ and $c$). An optimized affine version of algorithm 4.4 requires one variable

***Algorithm 4.5***: *Madd (Point addition algorithm)*

**Input**: The $x$-coordinate of $P(x)$ and the x-coordinates $X_1/Z_1$ and $X_2/Z_2$ for the points $P_1$, $P_2 \in E(\mathbb{F}_{2^m})$
**Output**: The $x$-coordinate $X_1/Z_1$ for the point $P_1 + P_2$

```
1  X_1 = X_1 · Z_2;
2  Z_1 = Z_1 · X_2;
3  T_1 = X_1 · Z_1;
4  Z_1 = Z_1 + X_1;
5  Z_1 = Z_1^2;
6  X_1 = Z_1 · x;
7  X_1 = X_1 + T_1;
8  return X_1/Z_1;
```

***Algorithm 4.6***: *Mdouble (Point doubling algorithm)*

**Input**: The $x$-coordinate $X/Z$ of point $P$ and $c = b^{2^{m-1}} \Rightarrow c^2 = b$
**Output**: The $x$-coordinate $X/Z$ for the point $2P$

1 $X = X^2$;
2 $Z = Z^2$;
3 $T_1 = Z \cdot c$;
4 $Z = Z \cdot X$;
5 $T_1 = T_1^2$;
6 $X = X^2$;
7 $X = X + T_1$;
8 **return** $X/Z$;

less, but it requires 10 times as much execution time on the target hardware.

Chapter 5

# The ISO-18000-3-1 standard

Several RFID standards were mentioned in the introduction, but so far no indication as to which of them is implemented by the RFID tag front-end of the circuit designed in this work. Amongst the multitude of possible choices the EPC class 1 gen 2 [EPC05] and the ISO-18000-3-1 [ISO04] standard are the two most likely candidates to provide a viable solution for supply chain management. Because of the availability of ISO-18000-3-1 compatible Field Programmable Gate Array (FPGA) based test hardware this standard was chosen.

The ISO-18000-3-1 standard document specifies the physical layer and the communications protocol that a compatible tag has to support. The physical layer is implemented by the test hardware(analog front-end) in conjecture with a third party module (digital front-end) available for this project.

## 5.1 Physical layer

The physical layer is still relevant to this work, because of its implications for the architecture of the circuit and the impact it has on the communication protocol.

The operating frequency of the carrier field $f_c$ is 13.56 MHz. The clock for the digital part of the RFID chip derived from this base frequency. That implies that it is easy to create clock signals with a frequency that is an integer fraction of 13.56 ($fclock = \frac{f_c}{i}$, $i \in \mathbb{N}$).

Tag to interrogator communication uses the load modulation technique to transfer information. Load modulation denotes the controlled inclusion of a specific load into the receiver circuit of the RFID tag, that is also responsible for powering the unit. Interrogators are capable of detecting these minimal changes in the inductive coupling area between reader and tag. Load modulation is a passive method, the tag does not emit its own field, which would require comparatively excessive amounts of power. That is a clear advantage in this context.

There is also a distinct downside to this scheme, that occurs if these simple RFID tags are enhanced with additional functionality. The RFID device draws the power required to supply it from the EM field. Thus nominal operation of the tag already loads the field. This is not a problem in itself, as the threshold

the reader reacts to is beyond the variabilities induced by the digital circuit of a pure identification tag.

Problems arise for more complex devices that are complemented with modules for let's say ECC to give a concrete example. The current needed to power an ISO-18000-3-1 protocol controller is diminutive in comparison to that of an ECC unit. If said extension has great differences in its power consumption profile, due to activation and deactivation of larger functional blocks, this can produce changes in the field load that are actually interpreted as attempts at communication by the interrogator.

One property of the load modulation communication scheme of the ISO-18000-3-1 protocol is, that if more than one party tries to load the field with two different values at the same time, the collision this causes on the carrier is detectable by the RFID reader. The communication protocol capitalizes on this fact to distinguish all units currently in the vicinity of the reader.

The operating distance of this protocol is limited to approximately 1.5 meters. This constraint arises from the problem of powering the passive devices and not from the communication range. The data rates specified by the ISO-18000-3-1 standard are between 1.65 kbits/s up to 26,69 kbits/s. The EM-field strength is limited due to internationally varying regulations on acceptable EM emissions.

## 5.2 Interrogator-tag communication

The nature of the communication protocol defined by the ISO-18000-3-1 specification is understandably simple. The communication is always initiated by the reader device. It sends a command to the tag. Depending on the type of the message the tag may have to reply. This is called an Interrogator Talks First (ITF) scheme.

The standard defines a set of instructions or commands that fall into two categories: Mandatory and optional. There are only two mandatory commands that a compliant unit has to support. Both are connected to the primary function of RFID tags, the identification of entities.

The exchanged messages are structured into a logical block called frame. A frame consists of several fields containing information like the type of command or flags. The protocol itself is bit oriented, but every field is an integer multiple of one byte.

The reader device can either broadcast commands to all tags or select a specific recipient in the addressed mode, where the Unique Identity (UID) of the intended target is appended to the request. In this case only that tag will react to the command.

## 5.3 Anti-collision sequence

The first mandatory instruction is the *inventory* command. The primary purpose of RFID technology is the rapid automated identification of entities. The task of an interrogator is to correctly detect all RFID tags in its vicinity, each of which has a 64-bit number uniquely identifying it (UID).

This poses an interesting problem. How to correctly identifiy all units in the EM field, without prior knowledge about their numbers and which IDs to expect. A protocol that sequentially queries all $2^{64}$ possible UIDs is clearly out of question, considering the available data rates.

The solution provided by the ISO-18000-3-1 standard is termed anti-collision sequence. As mentioned before the interrogator is capable of detecting a collision on the carrier field if two or more passive device try to respond to at the same time. The granularity of the detection is fine enough to identify at which transmitted bit the collision occurred.

To start the process the RFID reader sends out an inventory command. All tags in range will answer at exactly the same time $\pm$ a small acceptable variation. The responses contain the 64-bit UID of the tags. If at least two tags react, a collision is ensured, due to the fact that the IDs are unique and have to differ at least in one bit.

The reader notes the first occurrence and starts the second round of the scheme by transmitting a special version of the inventory message. It contains the UID up to and including the bit that caused the collision, setting it either to one or zero. Only tags that have an ID that begins with the received bit sequence will respond to that request.

At this time all UIDs are dividable into two categories. Those which have a zero at the point of collision and those which are fixed to a one. The same is true for all the following collision occurrences. This is evocative of a binary decision tree. At every node of the tree (point of collision) the reader selects one of the two branches and recursively continues to do so until the UID of a tag becomes distinguishable. It then retraces back to the last node and performs the same procedure for the other branch. This is repeated until all tag IDs are identified.

The standard propagates two different variants of the anti-collision scheme. So far, the simpler one of them has been explained. The second form partitions all tags by the first four bits of their UID into 16 different response time slots. The anti-collision is performed normally on each slot.

The second mandatory function is the *stay quiet* instruction. This request is always addressed at a certain recipient device. The targeted RFID tag will not responed to inventory requests any more.

## 5.4 Optional commands

The ISO-18000-3-1 standard defines a range of optional commands. While most of them are of no concern to this work, two of them facilitate the transfer of arbitrary data blocks between interrogator and tag and vice versa.

**Read single block** transfers a block of data from the memory of the RFID tag to the reader. Utilizing this instruction the interrogator requests the results of cryptographic operations computed by the ECC processor.

**Write single block** The opposite of the read single block request. It stores a data value into the memory of the passive device. For the ECC enabled tag presented herein, this command allows amongst other things, to specify a certain $k$ for the point multiplication.

A second group of optional functions that was adopted by this implementation allows to select a specific tag. The *select* instruction is an addressed command that puts the recipient into the selected mode. The interrogator can then use the *Select* flag to talk to that specific device, without the necessity to append the address to every request.

The corresponding counter-command is called *reset to ready*. This is another addressed instruction that resets a target from either quiet or selected state back to the normal mode of operation: The *ready* state.

## 5.5 Custom ECC commands

The final set of instructions concerns the addition of cryptographic capabilities to the RFID functionality. The ISO-18000-3-1 standard reserves a certain part of the valid command identifiers for manufacturer dependent and independent extensions. The following three instructions are implemented as manufacturer dependent extensions.

**ECC reset** is responsible for initializing the Random Access Memory (RAM) of the ECC processor with the start values for the scalar multiplication.

**ECC** $k \times P$ actually performs the point multiplication.

**ECC sign** performs a limited version of the ECDSA signature creation algorithm. A fully standard compliant variant is not possible, due to a lack of certain required hardware components, the design of which was not part of this thesis.

Chapter 6

# Architecture

After the discussion of cryptography in general and Elliptic Curve Cryptography in particular and the choice of the ISO-18000-3-1 standard for the RFID front-end, it is time to select an appropriate architecture for the *ECCon* circuit designed as part of this thesis.

As a prerequisite for understanding the topics treated by this chapter, basic knowledge of digital hardware design concepts is fundamental and it is assumed that the reader is familiar with ideas like single-edge-triggered one-phase clocking (synchronous hardware), gates, registers, datapath and control flow.

The chapter starts out by formulating the requirements an RFID application necessitates in §6.1. Next, §6.2 introduces the high level architecture of the *ECCon*processor. In §6.3, the architecture of the RFID front-end is briefly discussed. The ECC processor the central piece of this work is deliberated in detail in §6.4.

In section §6.4.1 the intended structure of the unit and the design decisions that led to it are considered at great length because of their fundamental importance. Specifically, a standard compliant elliptic curve is selected, the bit-width of the datapath is decided upon and the nature of the arithmetic core and the structure and size of the memory are fixed.

The chapter concludes with a discussion of the control unit and the interface of the ECC processor in §6.4.3 and §6.4.4, respectively.

## 6.1 Prerequisites for a circuit in a RFID application

Prior to a detailed discussion on the architectures of the components of the *ECCon* processor, it is necessary to review the limiting factors of Integrated Circuits (ICs) for RFID applications.

An RFID tag is a tightly constrained environment. On the one hand they have to be as cheap as possible. A substantial amount of the expenses is due to the area of the digital part of the IC at the heart of every passive RFID device. The first directive for the design of any component in this context is to minimize its area.

On the other hand the second important factor is the power budget of the tag. According to [FW07a], the available current supply is restricted to $15\mu A$. This allows a peak power consumption of approximately $27\mu W$ at $1.8V$. It is illusive to believe that goal is achievable with the process used for the ASIC developed by this thesis, but it emphasizes the importance of a power aware design.

The last prerequisite is a constant power consumption. As mentioned in the discussion of the ISO-18000-3-1 physical layer (§5.1), abrupt changes in the supply current may lead to an erroneous transmission of data.

This has a profound effect on the power savings strategy that must be applied. Traditionally, the goal of power aware design is to reduce the consumption to a minimum. If any component is not necessary for a certain amount of time, it is virtually switched off. The reason for this is that applications that use that scheme usually depend on a battery.

A battery is limited reservoir of energy. Once it is depleted it must be exchanged. Therefore, the drain on it at any given time must be minimal to increase its lifetime. The de- and reactivation of components this implies is a problem in the given context.

Passive RFID devices are powered by an EM field created by an RFID reader and for this consideration, it is assumed that the reader has an unlimited power supply. The current available to the tag might be diminutive to say the least, but its supply is inexhaustible for all intents and purposes.

The best applicable low power design strategy is therefore to minimize the current drain of the circuit, such that it does not exceed the power budget and to keep all of its components constantly active.

## 6.2 *ECCon*

*ECCon* the IC designed in this master thesis is subdivided into several modules. These modules are the *analog front-end*, the *RFID Asynchronous Receiver Transmitter (RART)*, the *RCU* that implements the communication protocol and governs the forth part, the *ECC processor*. The exact top level architecture is depicted in figure 6.1

The analog front-end is connected to the antenna and manages data modulation and demodulation, clock generation and power supply. The structure of this interface, although an interesting topic by itself, is not part of this work. Its relevance is therefore limited to the fact, that there exists a device that performs the aforementioned functions and in addition deasserts a reset, whenever the power supply has stabilized enough for a normal mode of operation.

The RART has two tasks. First, it serves as a converter between the bit oriented analog front-end and the system bus that connects the other components. The bus has a width of one byte. The second function is to generate certain protocol specific events that influence the control unit. An efficient implementation of this element was supplied by Mr. Martin Feldhofer from the Institute of Applied Information Processing and Communications at Graz University of Technology.

The RCU implements an extended version of the ISO-18000-3-1 protocol. It handles the RART, the data exchange between the transceiver and the ECC processor and operates the ECC processor.

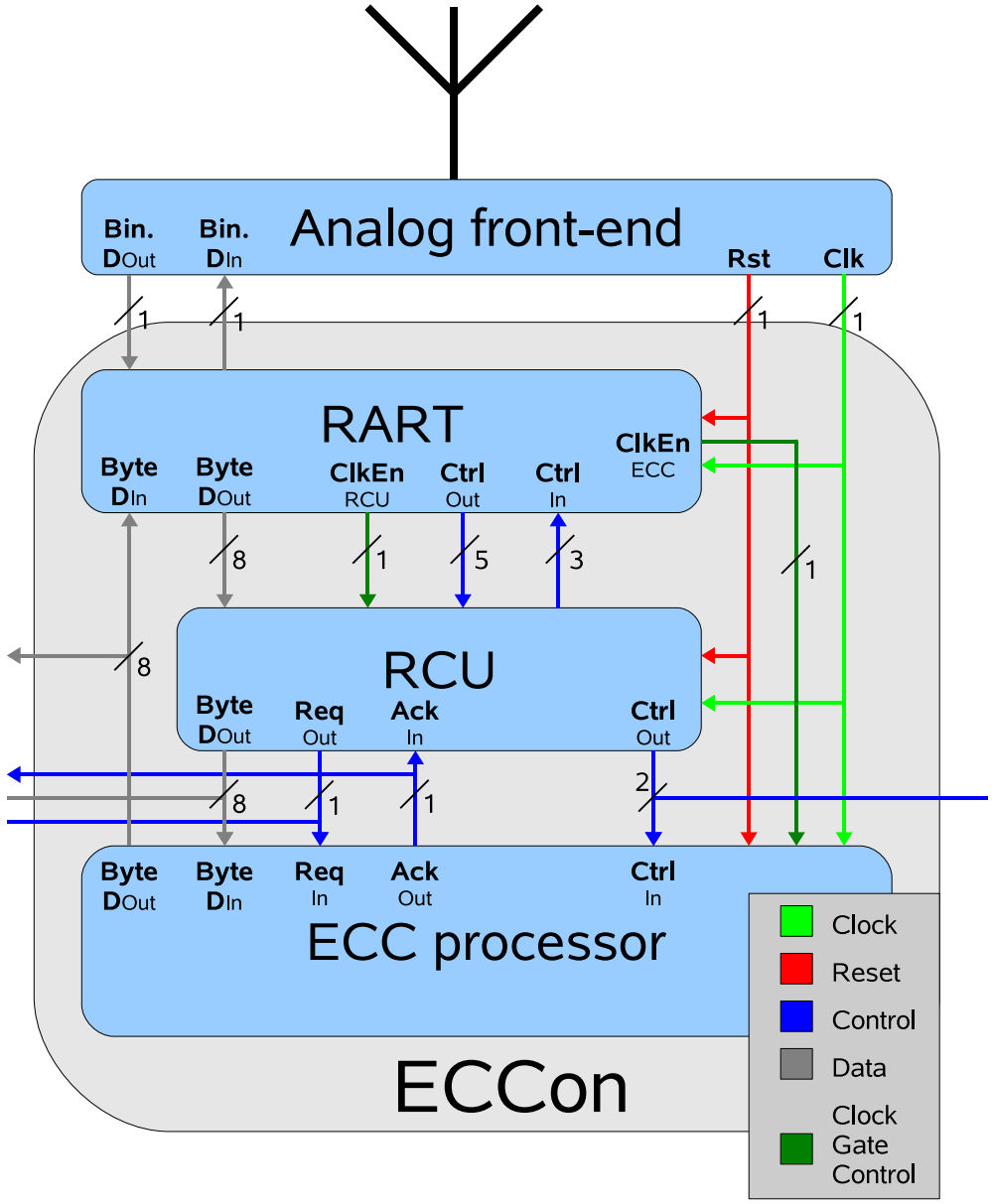



**Figure 6.1:** *Top level architecture*

Last but certainly not least the ECC processor is capable to perform an EC point multiplication as its primary mode of operation. Apart from that it is capable to execute a restricted version of the ECDSA signature generation algorithm with it.

## 6.3 RFID front-end

The RFID part of the circuit consists of the analog front-end, the RART and the control unit. The analog front-end is beyond the scope of this work and therefore warrants no further discussion.

The RART is a third party module. It implements the digital part of the physical layer of the ISO-18000-3-1 standard. The implementation is not complete, but suffices to work with most standard compliant reader devices.

The RFID Control Unit is a straight forward FSM (confer to §6.4.3) based implementation of the protocol commands detailed in chapter §5.

## 6.4 ECC processor architecture

The ECC processor is designed to compute the point multiplication on an elliptic curve. The base requirements for this are:

1. A memory that posses the capability to store the elements of the underlying finite field that make up the points on the EC.
2. An ALU that has the ability to perform operations on them.
3. A control unit that governs the datapath consisting of the memory and the ALU.
4. An interface that establishes a connection to the outside world.

Figure 6.2 illustrates the primary components that make up the ECC processor core. The difference between a register and a module is that a module consists of registers and a combinatoric logic, while the register consists solely of flip-flops and possibly a clock gating component. Both registers and modules all have a clock, reset and clock enable input that was omitted from the illustration.

### 6.4.1 Datapath

Several design decisions with regards to the datapath have to be ascertained. The first concerns the size of the finite field that the EC is defined upon.

#### Curve selection

In 4.2.1 a binary extension field, or simply binary field, was chosen as finite field on which the EC is defined, as it is the most appropriate candidate. The reason for this is that the addition in a binary field corresponds to an Exclusive Or (XOR) operation, which greatly simplifies implementation of all field operations.

The next step is to specify its degree. The degree defines the size of an element in memory, therefore the smallest degree that provides adequate security

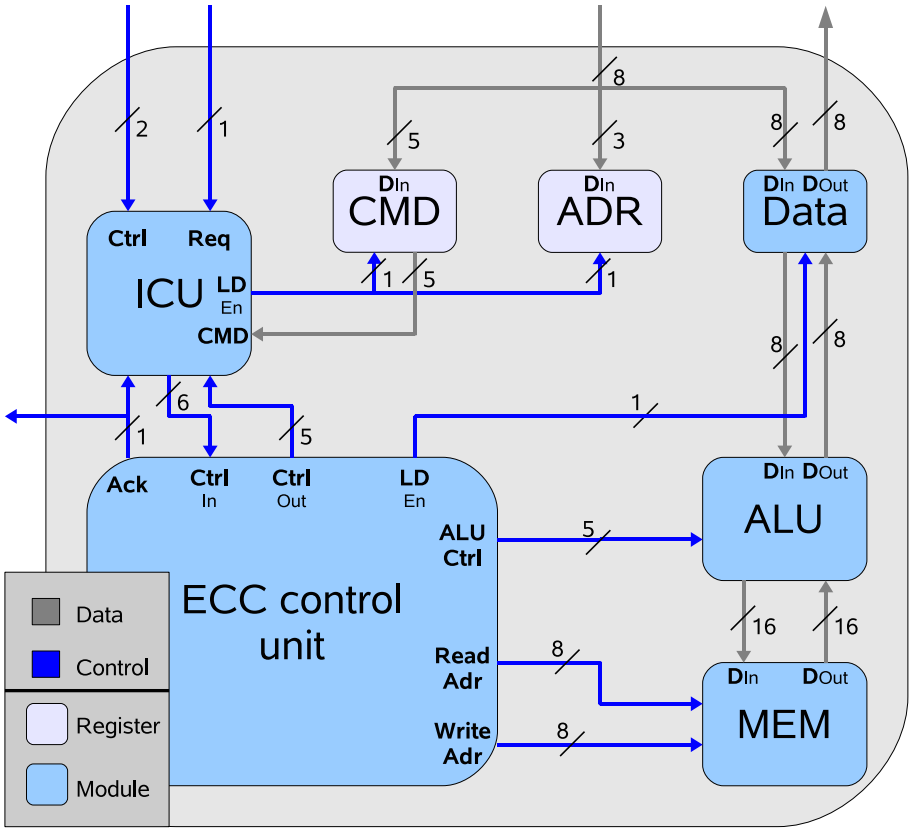



**Figure 6.2:** *ECC processor architecture*

has to be selected. This is due to the fact that memory is one of the main contributing factors to the die area.

The NIST FIPS-186-2 Digital Signature Standard (DSS) [Fed00] provides a set of possible curves categorized by the degree of the associated field. From it *Curve B-163* ($\mathbb{F}_{2^{163}}$), the smallest possible curve provided by that standard, was selected. The definition includes the domain parameters $n$ (degree), $a, b$ (curve coefficients), the base point $P$, and an irreducible polynomial $f(z)$.

| Name | Value |
|---|---|
| $n$ | 163 |
| $a$ | 1 |
| $b$ | $20a601907b8c953ca1481eb10512f78744a3205fd$ |
| $P_x$ | $3f0eba16286a2d57ea0991168d4994637e8343e36$ |
| $P_y$ | $0d51fbc6c71a0094fa2cdd545b11c5c0c797324f1$ |
| $f(z)$ | $z^{163} + z^7 + z^6 + z^3 + 1$ |

**Table 6.1:** *Domain parameters of* Curve B-163

The reason for choosing this particular curve is that, according to Hankerson et al. [HMV04], it provides approximately the same security as an 80-bit AES implementation, while at the same time $n$ is still reasonable small. The degree of the next curve proposed by the NIST standard (233) is significantly higher.

### Datapath width

Traditional hardware architectures for ECC processors [Wol04] employ *full precision datapaths* that are equivalent in width to the size of an element in memory. The advantage of this approach is that it simplifies the algorithms that implement the field operations. The downside is due to the size of the elements. A full precision architecture affords an ALU with an *accumulator* register the size of which is equal to the degree of the field.

In this case 163 registers would be required. For every operation performed at least these 163 registers would change at every clock cycle and that does not consider writing of the results to the memory or the combinatoric logic involved.

Three distinct problems arise from this fact. The first stems from the sheer size of a 163 bit accumulator. The area taken up by it is considerable. The second more stringent problem is the power consumption of such a unit. The problem is not solely the power supply but also the possible disruption of the load modulation based communication. Such a number of registers switching values at once draws a critical amount of current, which can lead to the aforementioned communication disruptions.

This obvious example illustrates that an RFID application necessitates a more suitable solution. Therefore, for the ASIC developed as part of this thesis, a datapath width that is distinctly smaller than the degree of the curve was selected. This has the disadvantage that the operations in $\mathbb{F}_{2^{163}}$need to be implemented by more complex algorithms that work with smaller bit-widths.

To distinct between a full precision element of $\mathbb{F}_{2^m}$in general and $\mathbb{F}_{2^{163}}$in particular and an element fit for the datapath the following definition is made: A full precision element of a field shall furthermore be denoted by the term *element*,

whereas an element with the same width as the datapath shall henceforth be called a *word*.

To determine a suitable word size for the datapath with respect to the requirements formulated in section §6.1, it is necessary to find some kind of estimation for the relation between the word-width and the area requirements and power consumption of the datapath. This is strongly dependent on the nature of the memory and the ALU implementation.

The following procedure was applied to gain the necessary assessment. A Hardware Description Language (HDL) model of the candidate ALU was developed. The bit-width of the ALU is a parameter of this model. The HDL description was then synthesized into a hardware implementation for all word-widths between 8 and 31 bits. The reasons for choosing this range will become clear later. For each bit-width the area and power usage estimations of the synthesizing tool were noted.

For an accurate evaluation of the datapath, the timing is also required. The critical path of the circuit is secondary in the projected application scenario, where the ECC processor is going to be driven by the lowest possible frequency. Therefore, a different more significant replacement metric has to be found. This is the total number of clock cycles necessary to compute a point multiplication, which directly depends on the bit-width. The rule is: The larger the word size, the faster the computation. This value was provided by the Java software model of the ECC processor.

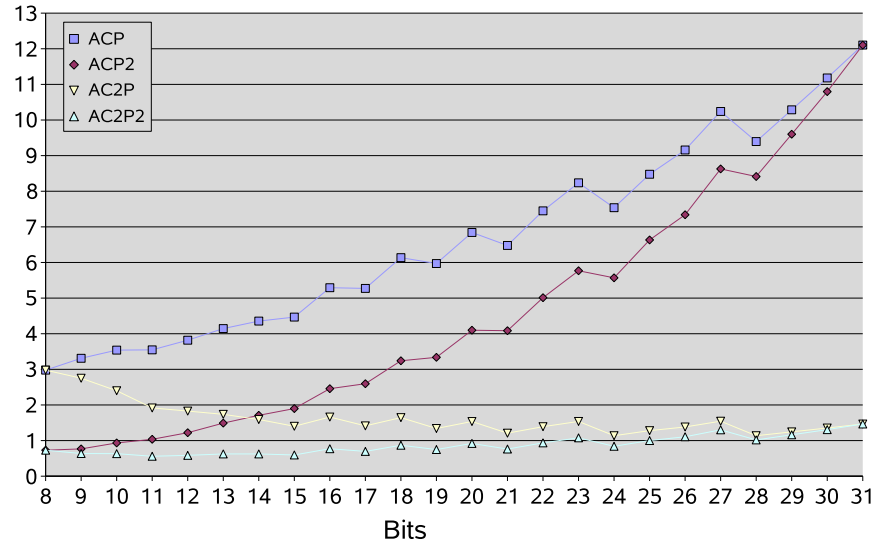



**Figure 6.3:** *A comparison of different datapath widths*

The estimations of all three relevant values, the area, the power consumption and the cycle count were normalized to their respective maximums. The results were then combined to form the 4 different metrics depicted in figure 6.3.

Considered separately the three estimates have the following behavior: The

area $A$ increases more or less linearly with the bit-width. The same is true for power $P$. The number of necessary cycles to compute the point multiplication $C$ decreases quadratically with larger word sizes.

The area×cycles×power product $A \cdot C \cdot P$ is the baseline for comparison with the other metrics. The $A \cdot C \cdot P^2$ product emphasizes the influence of the power consumption, but shows a similar characteristic as the first metric. Both the $A \cdot C^2 \cdot P$ and the $A \cdot C^2 \cdot P^2$ are timing dominated.

Considering figure 6.3 it should become clear, why the 8 to 31 bit range was chosen. The $A \cdot C \cdot P$ and the $A \cdot C \cdot P^2$ functions increase almost quadratically with the bit-width. Every datapath width above 31 would require to much power and area. The number of cycles to compute the point multiplication is inversely proportional to the bit-width and increases quadratically as the bit-width decreases. Thus every datapath below eight bit would be too slow.

No single metric shows a conclusive result, but by combined evaluation one may arrive at the conclusion that a datapath width of 15 bit is possibly the best choice. After the 15-bit point both the $A \cdot C \cdot P$ and the $A \cdot C \cdot P^2$ product begin to increase with before unseen steepness, while both timing driven metrics show a local minimum at 15 bit.

Thus a word size of 15 bit seems optimal. A problem that this evaluation does not take into account is the interface of the processor. Common bus widths are multiples of eight bits. The ECC processor is equipped with an interface that is exactly eight bit wide. Converting words of a width $w = i \cdot 8$, $i \in \mathbb{N} \setminus 0$ to eight bit is much simpler than for arbitrary word widths. Therefore the final choice for the datapath width is 16-bit.

### ALU

The ALU of the ECC processor has to provide facilities to compute basic operations for 16-bit binary polynomials (confer to §4.3.5). These are addition, subtraction and multiplication. As addition and subtraction are the same for binary polynomials (XOR), this list reduces to just addition and multiplication.

As will be seen in the discussion of the algorithms that implement all necessary operations in $\mathbb{F}_{2^{163}}$, it is of advantage when it is possible to perform a multiplication and then an addition on the result in one clock cycle. This operations is termed MAC and an ALU that has the capability to perform this is called MAC unit. The ALU of the ECC processor is a MAC unit.

### Memory

For the memory architecture a choice between using a RAM macro and a register based structure has to be made. A RAM macro facilitates the creation of storage capacity adapted memories built from SRAM or DRAM cells. This approach produces area optimized results and it is also the most power efficient.

The problem with the RAM macros available to implement the ASIC designed as part of this work is that they have been optimized for speed. High speed circuits use gates with stronger driving capabilities which are generally faster but also entail an increased power consumption.

A second point of uncertainty stems from the memory structure. A memory consists of cells that store the data and a selection logic part, also called memory

bus herein. The second component choses which cells to write to and selects a data word for reading.

This memory bus restricts the allowable choices for acceptable element sizes. RAM macros have to generate the selection logic automatically, which is difficult in cases where the element size is not a multiple of eight. Thus to implement a $7 \times 163$ bit RAM with a macro a $80 \times 16$ structure would have to be chosen. This structure squanders 139 bits and complicates addressing words with respect to the element they belong to.

Therefore, a register based implementation built from standard cells was selected for this project. The final memory circuit uses several power optimization strategies to produce a storage unit with a diminutive current drain, while at the same time minimizing the required area.

A further advantage of this approach is that it naturally allows to perform one read and one write operation per clock cycle at the expediency of very little additional power. Concurrent read and write operations facilitate the creation of algorithms that require less clock cycles, which is a considerable advantage.

The memory has a total size of $7 \times 163$ elements. This number arises from the 6 variables the Montgomery point ladder algorithm for scalar multiplication on an EC utilizes plus one which is due to implementation details. Each 163-bit element is split into 10 16-bit words and one 4-bit word that stores the bits 160 to 163.

#### Memory bus

The ALU is capable to perform additions and multiplications. Both operations necessitate two operands to create one result. The algorithms that perform the binary field operations utilizing this MAC unit could be substantially sped up, if it were possible to select two words from the memory at once.

Theoretically it is possible to add more than one read port (read bus) to a memory. Preliminary assessments showed that the hardware resources needed to implement one memory read bus are considerable. The area necessary is approximately twice the area of a 16-bit ALU, compelling the restriction of the system to one read bus. This has a direct impact on the minimum number of cycles needed to perform an operation.

Similarly the write bus allows the write back of one word to memory per cycle. This imposes no further restrictions on the number of atomic operations needed for any operation because the number of memory reads is in all cases equal or greater than the number of writes.

The multiplication will serve as an example to illustrate above considerations. Two operands need to be read and multiplied to generate one result of double length, which in turn takes two cycles to be written back to memory. The multiplication is the only operation that produces a double width result.

### 6.4.2 Datapath design alternatives

Traditional hardware architectures for ECC processors use either bit-serial or bit-parallel, word-serial multiplication schemes [Wol04]. The later is sometimes called a high-radix multiplication or digit-serial multiplication. Bit-serial multipliers are generally more power efficient, while digit-serial multipliers have a

greater throughput and are more energy efficient[1]. In both schemes one operand is a full-precision field element, while the other is either one bit (bit-serial) or one word (digit-serial). These methods are often combined with interleaved modular reduction.

Energy efficient digit-serial multiplication is discussed in some detail by [SP98]. This was taken up by [OP00] to produce a high throughput ECC solution, as does [GSE+02]. In more recent works several proposals for digit-serial multipliers for smart card co-processors have arisen [ABHW04, TWA05], where area cost is at a premium.

Scalable energy efficient bit-serial multiplication schemes have also been proposed [HLRZ03], but more importantly it has been ascertained by [Wol05, KP06, FW07b], that the power consumption and area requirements of such architectures are small enough, that they could be used to add ECC capabilities to RFID tags.

The approach chosen by this work is not uncommon, but has so far been mostly applied to ISA extensions [EWG+05] and flexible, scalable solutions [BL06].

### 6.4.3 Control unit

The control unit governs the datapath. The datapath supplies a set of basic operations, which, executed in a specific sequence, form usable commands, in this instance operations in $\mathbb{F}_{2^{163}}$. The succession of primitive instructions is called control flow.

There exist three prominent techniques to implement a control flow in hardware [Wol04].

1. Hardwired control
2. Micro-programmed control
3. Micro-controller

Hardwired control is an implementation of the Finite State Machine (FSM) paradigm. A FSM is an abstract model of behavior that utilizes a set of states and a next-state function to create a desired sequence of events.

A FSM contains a state variable which holds the current state and in possible conjunction with other input variables is used to compute the control signals for the managed datapath. The next state is equally derived by the next-state function from the current state and depending on the kind of FSM other input variables.

This is the most efficient concept with respect to area usage and power consumption and it maps especially well to synchronous hardware. The drawback is that, even using a state-of-the-art HDL, describing a complex flow of events becomes a tedious task. This can be alleviated in part by hierarchically structuring, where low-level FSMs realize simple operations and high(er)-level ones use these to implement more powerful functions.

[1] Bit-serial-multipliers for low power applications require less power than digit-serial-multipliers, but they need more time to calculate the result, thus this approach requires more energy.

The micro-programmed and micro-controller scheme rely on a set of instruction stored in a Read Only Memory (ROM), which are either executed by a specialized FSM based control unit or a full fledged micro processor respectively. Both are powerful approaches to implement complex algorithms. This comes at a cost of greater area usage and power consumption in comparison to hardwired logic.

The control flow of the ECC processor is implemented using the hierarchical-FSM paradigm. The top level command interface of the control unit provides the three commands already discussed in section §5.5.

### 6.4.4 Interface

The interface connects the ECC processor to the RFID front-end. It implements a two-phase full handshake protocol. It is fully registered, every input and output value is transfered into a register.

It provides two different functionalities. It allows read and write access to the memory core of the ECC processor and it facilitates the invocation of the three cryptographic commands implemented by the processor.

The RFID subsystem uses an 8-bit bus, due to protocol considerations. The internal datapath is 16 bits wide. This necessitates that the interface performs a conversion for memory access operations.

The interface possesses its own control unit which implements the two functions. This unit is again implemented as a FSM.

Chapter 7

# Binary field algorithms

The decisions made in the architecture chapter (§6) of this work require the implementation of binary extension field operations on an elliptic curve defined over the binary field $\mathbb{F}_{2^{163}}$ with the irreducible polynomial $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$ as specified in the NIST FIPS-186-2 DSS [Fed00].

Recall that the mathematical field definition specifies a set of basic operations (confer to §4.3.5). The standard arithmetical operations, addition and multiplication and their inverse functions, are complemented by squaring and modular reduction. While the first is introduced for performance reasons, the later is of course elementary to a finite field. Thus, the list of operations a binary field arithmetic unit must implement is as follows:

- addition/subtraction; $c(z) = a(z) \oplus b(z)$
- multiplication; $c(z) = a(z) \cdot b(z) \, mod \, f(z)$
- inversion; $c(z) = a(z)^{-1} \, mod \, f(z)$
- squaring; $c(z) = a(z)^2 \, mod \, f(z)$
- reduction; $c(z) = c(z) \, mod \, f(z)$

A veritable cornucopia of possible choices for the concrete algorithmic realization of these operations exists. This part of the thesis reviews a selection of candidate methods and introduces two newly developed algorithms.

§7.1 provides material on the conventions used for algorithm descriptions, specifies the representation of elements in the binary field $\mathbb{F}_{2^{163}}$ and presents the concept of ISA. In §7.2, the character of two candidate ISA is discussed. Next §7.3 covers the addition/subtraction operation, while §7.4 talks about multiplication in some detail. Section §7.5 addresses the issue of squaring and §7.6 considers method and implementation of the reduction operation. It also introduces two new, highly optimized algorithms specifically developed to streamline the multiplication, squaring and reduction operations in certain binary extension fields with the support of a special-purpose instruction set. The chapter is concluded by an analysis of possible inversion algorithms in §7.7. This section closes with a presentation of appropriate inversion/scalar-point-multiplication combinations.

# 7.1 Conventions

## 7.1.1 Field element representation

Section §4.3.5 detailed that the elements of a binary field $\mathbb{F}_{2^m}$ can be represented by polynomials in the form $a(z) = a_{m-1}z^{m-1} + \cdots + a_1 z + a_0$, where the coefficients $a_i$ are either 0 or 1. This is called polynomial or standard basis representation. The degree *deg* of a binary polynomial $deg\{a(z)\}$ is defined as the degree of the highest order term in $a(z)$. The length $l$ of a polynomial is $l = deg\{a(z)\} + 1$. This value is equal to the number of flip-flops required to save the coefficients of one polynomial.

Specifically an element of the field $\mathbb{F}_{2^{163}}$ has the following structure: $a(z) = a_{162}z^{162} + \cdots + a_1 z + a_0$. As has been ascertained in section §6.4.1, the term *element* will be used to denote a full precision element of a field.

Polynomials structured as discussed above lend themselves to a binary string representation. Each coefficient is thus represented by a single bit in hardware. Elements in $\mathbb{F}_{2^{163}}$ are stored in a memory structure with a fixed length of 163 bits.

As was deliberated in section §6.4.1, the datapath is not capable to handle full precision elements. Therefore, the field elements need to be broken down into smaller fractions $w(z) = w_{n-1}z^{n-1} + w_1 z + w_0$, $n \leq m$. In accordance with the definition made in the aforementioned section, these will be called *words*. *Digit* is also an acceptable synonym for these polynomials. The length $w$ of a word is equal to the datapath width. Thus a field element is split into $t = \lceil m/w \rceil$ words.

## 7.1.2 Algorithm descriptions

The following sections use precise pseudo-code descriptions to detail the presented algorithms. Many have been adapted from [HMV04]. As a consequence, a similar style is used to describe the algorithms.

To better understand the descriptions, a few remarks on the nomenclature are in order. An element is interpreted as an array of $t$ words, therefore the array index notation common to many programming languages is used, where $A[i]$ selects the $i^{th}$ word of $a(z)$. Variables in lower case always represent a complete polynomial, e.g. $a$ describes $a(z) \in \mathbb{F}_{2^{163}}$. The $(z)$ part is often omitted for brevity. Upper case letters stand for words. Furthermore, little-endian word ordering applies, where the lowest order digit has the index 0.

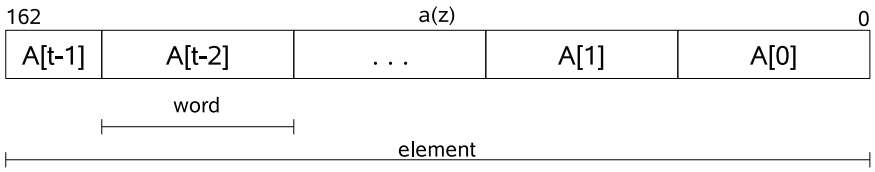



**Figure 7.1:** *An element of $\mathbb{F}_{2^{163}}$ as array of $t$ words.*

### 7.1.3 Instruction Set Architectures

This chapter aims to compare different algorithms for implementing the binary field operations. A problem arises due to the fact that the diverse methods have varying prerequisites with respect to the hardware they are executed upon.

To formalize these requirements, a technique from Central Processing Unit (CPU) design is borrowed. The functionality of such a device is characterizable by the instructions it understands, while at the same time it does not specify too many implementation details. This set of instructions is denoted by the term Instruction Set Architecture (ISA). The ISAs will serve to describe the potential ALUs that could implement the algorithms introduced in this chapter.

A single instruction consists of a set of basic operations. Each of these is composed from an expression and a local storage component descriptor. The latter indicates where the result of the expression should be saved to. The expression itself consists of a mathematical function applied to at least one input operand. An input operand can either be supplied by one of the local storage components or the input port of the ALU described by the ISA.

A storage component, or register of the virtual ALU is denoted by a **bold** type face (**B**, $\mathbf{ACC_H}$ etc.). Together with a mathematical function they form expressions like $\mathbf{A} \cdot \mathbf{B}$. To indicate where the result should be stored a $\leftarrow$ is used ($\mathbf{C} \leftarrow \mathbf{A} \cdot \mathbf{B}$). The letter $\mathcal{I}$ is used to denominate the input of the ALU. The output of the ALU is always the current state of the lowest 16 bits of the accumulator, a special register in the ALU candidate, which is identified by $\mathbf{ACC_L}$. All registers in the potential ALUs, as well as the input and the output, are all 16 bits wide, unless specifically noted differently.

The invocation of an instruction in an algorithm takes the form of a function call, using the name of the instruction to select it. The input value, which can be left unspecified is appended in parenthesis. To indicate that the output value should be stored in a memory word, which is identified using the conventions defined in the above section (§7.1.2), the left arrow $\leftarrow$ is applied.

The *Load register B command* $C[i] \leftarrow LDB(A[i])$ will serve as a concrete example to illustrate an instruction invocation. In this case, the register **B** will be loaded with the value stored in the memory cell $A[i]$ and the current value of $\mathbf{ACC_L}$ will be saved to $C[i]$.

The ISA algorithm representation is employed whenever an implementation-specific variant is presented. For algorithms that illustrate a concept, the simpler pseudo code variant is used.

## 7.2 Definition of potential ALU ISAs

Two different ISAs are of interest. The first ALU candidate is called *Simplex*. It is indeed very simple but has the potential to implement all required binary field operations. It utilizes three registers (**B**, $\mathbf{ACC_H}$, $\mathbf{ACC_L}$), one polynomial multiplication unit and one polynomial adder. It supports the instructions detailed in table 7.1. Table 7.2 explains the mnemonic names identifying the instructions.

$\mathbf{ACC_H}$ and $\mathbf{ACC_L}$ together constitute the *accumulator* **ACC**. This special register stores the result of the basic functions$(\oplus, \cdot)$. The *accumulator* might be treated as one logic unit and not two separated storage containers. An example of this is the *SRE* command. It shifts the whole **ACC** 8 bits to the right.

*Complex* the is the second possible ALU that is considered herein. A look

| | |
|---|---|
| LDB | $\mathbf{B} \leftarrow \mathcal{I}$; $\mathbf{ACC_L} \leftarrow \mathbf{ACC_H}$; $\mathbf{ACC_H} \leftarrow 0$; |
| ADD | $\mathbf{ACC_L} \leftarrow \mathbf{ACC_L} \oplus \mathcal{I}$ |
| MAC | $\mathbf{ACC} \leftarrow \mathcal{I} \cdot \mathbf{B} \oplus \mathbf{ACC_H}$ |
| SRE | $\mathbf{ACC} \gg 8$ |

**Table 7.1:** Simplex *ALU operations*

| | |
|---|---|
| LDB | LoaD B |
| ADD | ADD |
| MAC | Multiply ACcumulate |
| SRE | Shift Right accumulator by Eight (8) bits |

**Table 7.2:** Simplex *ALU operation names*

at table 7.3 gives an idea, how the name was derived. Despite the multitude of instructions it provides, its actual hardware implementation is still relatively simple. It requires 5 internal registers (**B**, **RC**, **MC**, $\mathbf{ACC_H}$ and $\mathbf{ACC_L}$), one polynomial multiplier and two polynomial adders. The **RC** register is only 13 bits wide.

The terms $S_R$ and $S_L$ denote constants for shift operations. $S_R = m \bmod t$ which is equal to 3 if the degree $m = 163$ and the number of words $t = 11$. Then $S_L = w - S_R = 13$ in case the word width $w$ equals 16. They are required to indicate which bits of the 16-bit $\mathbf{ACC_L}$ are stored in the 13 bits wide **RC** register and at which position the content of the **RC** register should be loaded into the accumulator (confer to operations *SMR* and *SAA*).

The pipe | is used as concatenation operator. The *SAA* command contains an operation of the form $\mathbf{ACC_L} \leftarrow \{(\mathbf{ACC_L} \ll S_L)|\mathbf{RC}\}$. This is to be interpreted as shift $\mathbf{ACC_L}$ to the left by $S_L$ bits, concatenate that with the content of **RC** and store the result in $\mathbf{ACC_L}$.

The term $r(z)$ denotes the reduction polynomial, a special variant of the irreducible polynomial $f(z)$. It will be explained in §7.6.4. For the following considerations it is important to distinguish between the *Simplex* and *Complex MAC* operations as they differ.

## 7.3 Addition and Subtraction

As discussed in §4.3.5, addition is a very simple operation in any binary field. Addition is equivalent to the exclusive-or operation. Furthermore, addition and subtraction are identical in binary fields. The sum of two elements is the result of the XOR operation applied digit by digit to the input values. The addition finishes after $t$ operations and order is of no consequence as the XOR function has no carry.

The implementation version of the addition algorithm is basically the same for both the ISAs detailed in §7.2. Therefore, only the simplex variant is given here. For this implementation specific algorithm, it is possible to asses the number of cycles needed to compute an addition of two field elements. As the number of operands that can be read from memory per clock cycle is limited to one, the optimally achievable minimum of cycles is two per word addition.

| | |
|---|---|
| LDB | $\mathbf{B} \leftarrow \mathcal{I}$ |
| LDO | $\mathbf{B} \leftarrow 1$ |
| LBS | $\mathbf{B} \leftarrow \mathcal{I}; \mathbf{ACC} \leftarrow \mathbf{ACC} \gg 16$ |
| SRW | $\mathbf{ACC} \leftarrow \mathbf{ACC} \gg 16$ |
| SRE | $\mathbf{ACC} \leftarrow \mathbf{ACC} \gg 8$ |
| SMR | $\mathbf{ACC} \leftarrow \mathbf{ACC} \gg 16; \mathbf{MC} \leftarrow \mathcal{I}; \mathbf{RC} \leftarrow \mathbf{ACC_L} \gg S_R$ |
| SMC | $\mathbf{ACC_H} \leftarrow 0; \mathbf{ACC_L} \leftarrow \mathbf{MC}; \mathbf{MC} \leftarrow \mathbf{ACC_H}$ |
| SAM | $\mathbf{ACC_H} \leftarrow 0; \mathbf{ACC_L} \leftarrow \mathbf{MC}; \mathbf{MC} \leftarrow \mathcal{I} \oplus \mathbf{ACC_H}$ |
| SAA | $\mathbf{ACC_H} \leftarrow \mathcal{I} \oplus \mathbf{MC}; \mathbf{ACC_L} \leftarrow \{(\mathbf{ACC_L} \ll S_L)\|\mathbf{RC}\}; \mathbf{MC} \leftarrow \mathbf{ACC_H}; \mathbf{RC} \leftarrow \mathbf{ACC_L} \gg S_R$ |
| SLB | $\mathbf{ACC_H} \leftarrow \mathbf{MC}; \mathbf{ACC_L} \leftarrow (\mathbf{ACC_L} \ll S_L)\|\mathbf{RC}; \mathbf{MC} \leftarrow \mathbf{ACC_H}; \mathbf{RC} \leftarrow \mathbf{ACC_L} \gg S_R; \mathbf{B} \leftarrow \mathcal{I}$ |
| SMA | $\mathbf{ACC_H} \leftarrow \mathbf{MC}; \mathbf{ACC_L} \leftarrow \mathcal{I} \oplus \mathbf{ACC_L}; \mathbf{MC} \leftarrow \mathbf{ACC_H}$ |
| SRD | $\mathbf{ACC_H} \leftarrow \mathbf{MC}; \mathbf{ACC_L} \leftarrow \mathcal{I} \oplus \mathbf{ACC_L}; \mathbf{MC} \leftarrow \mathbf{ACC_H}; \mathbf{RC} \leftarrow \mathbf{ACC} \gg S_R$ |
| RAC | $\mathbf{ACC_H} \leftarrow \mathcal{I}; \mathbf{ACC_L} \leftarrow \mathbf{RC}$ |
| ADD | $\mathbf{ACC_L} \leftarrow \mathcal{I} \oplus \mathbf{ACC_L}$ |
| MAH | $\mathbf{ACC} \leftarrow \mathcal{I} \cdot \mathbf{B} \oplus \mathbf{ACC_H}$ |
| MSM | $\mathbf{ACC} \leftarrow \mathcal{I} \cdot \mathbf{B} \oplus \mathbf{MC}; \mathbf{MC} \leftarrow \mathbf{ACC_H}$ |
| MAC | $\mathbf{ACC} \leftarrow \mathcal{I} \cdot \mathbf{B} \oplus \mathbf{ACC}$ |
| MAR | $\mathbf{ACC} \leftarrow \mathbf{ACC_L} \cdot r(z) \oplus \mathbf{ACC_H}$ |

**Table 7.3:** Complex *ALU Operations*

| | |
|---|---|
| LDB | LoaD B |
| LDO | LoaD B with the constant One (1) |
| LBS | Load B and Shift right accumulator by word width (16) |
| SRW | Shift Right accumulator by Word width (16) |
| SRE | Shift Right accumulator by Eight (8) bits |
| SMR | Set Multiplication carry and save Reduction carry |
| SMC | Swap Multiplication Carry register |
| SAM | Swap and Add Multiplication carry register |
| SAA | Swap And Add multiplication carry register and swap reduction carry register |
| SLB | Swap multiplication and reduction carry registers and Load B |
| SMA | Swap Multiplication carry register and add Add A to accumulator |
| SRD | Swap multiplication and reduction carry Registers and Add A to accumulator |
| RAC | Restore ACcumulator with reduction register and A |
| ADD | ADD A to accumulator |
| MAH | Multiply Accumulate with accumulator high word |
| MSM | Multiply and add and Swap multiplication carry register |
| MAC | Multiply Accumulate with the Complete accumulator |
| MAR | Multiply with the special irreducible polynomial r(z) and add accumulator high word |

**Table 7.4:** Complex *ALU Operation Names*

***Algorithm 7.1***: *Addition/Subtraction*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a + b; c \in \mathbb{F}_{2^m}$
1 **foreach** $C[i] \in c$ **do** $C[i] = A[i] \oplus B[i]$;
2 **return** $C$;

***Algorithm 7.2***: *Addition* - Simplex *implementation*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \oplus b; c \in \mathbb{F}_{2^m}$
1 $LDB(1)$;
2 $MAC(A[i])$;
3 $ADD(B[i])$;
4 **for** $i \leftarrow 1$ **to** $t - 1$ **do**
5 $\quad C[i] \leftarrow MAC(A[i])$;
6 $\quad ADD(B[i])$;
7 $C[t-1] \leftarrow LDB(0)$;

The runtime of algorithm 7.2 presented here is

$$t_{cycles} = 2 + 2 \cdot t.$$

It is almost optimal. The architecture adds a certain overhead, due to operand loading and result flushing.

## 7.4 Multiplication

### 7.4.1 Integer multiplication algorithms

There are two prominent integer multiplication algorithms [HMV04], the operand scanning form and the product scanning form. The product scanning form is sometime called Comba's method or Comba multiplication. Both are examined in their applicability with respect to the two ISAs introduced above.

A third and very interesting integer multiplication technique is the so called Karatsuba-Ofmann multiplication [KO63]. It applies the basic divide and conquer paradigm to reduce the length of the operands involved, at the cost of additional additions and subtractions. The overhead of the classic Karatsuba-Ofmann algorithm does not warrant its application for this project.

Peter and Langendoerfer [PL07] however devised an energy-efficient multiplier based on a modified Karatsuba-Ofman method, specifically for ECC circuits. The stringent peak power consumption restrictions prevent its deployment as integer multiplication algorithm, but it might prove a true alternative to the array multiplier core of the MAC unit.

### 7.4.2 Operand scanning form

The operand scanning form integer multiplication algorithm could also be termed schoolbook multiplication algorithm, because it is just that. It is the algorithm

that is presumably taught to most persons, that are educated in basic mathematics. It works by multiplying each digit of multiplicand with one digit from the multiplier for all digits in the multiplier. The resulting rows are aligned appropriately and the added up. Figure 7.2 depicts the process.

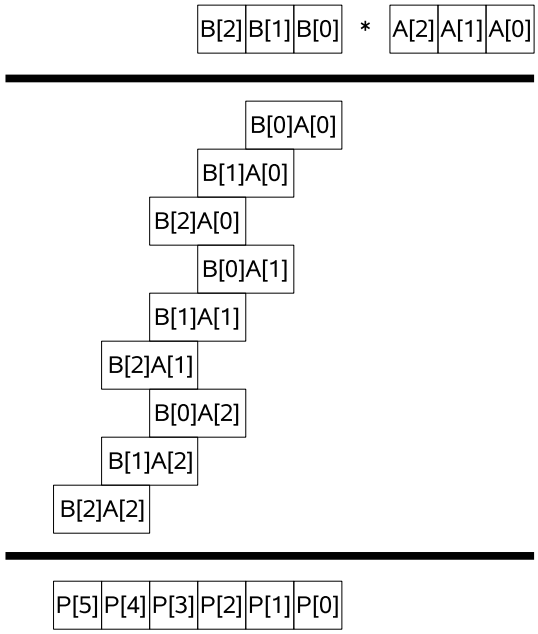



**Figure 7.2:** *The operand scanning form multiplication for two 3-bit integers*

***Algorithm 7.3***: *Multiplication (operand scanning form)*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \cdot b; deg(c) \leq 2m - 2$

```
1 foreach C[i] ∈ c do C[i] = 0;
2 for i ← 0 to t − 1 do
3 |   U ← 0  for j ← 0 to t − 1 do
4 |   |   (U|V) ← C[i + j] + A[i] · B[i] + U;
5 |   |   C[i + j] ← V;
6 |   C[i + t] ← U;
7 return c;
```

Algorithm 7.3 details the necessary steps. The term (U|V) describes the double word width result of the inner product operation $A[i] \cdot B[i]$. In (U|V), U represents the higher order and V the lower word of the result.

The operand scanning form can be efficiently implemented with the *simplex* ALU outlined by the ISA given in table 7.1. The hardware specific implementation of algorithm 7.3 is given by algorithm 7.4.

The inner product operation can be performed in two cycles, which is the minimal possible cycle number with one read bus. The last two operations

***Algorithm 7.4***: *Multiplication (operand scanning form)* - Simplex *implementation*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \cdot b; deg(c) \leq 2m - 2$

```
1 for j ← 0 to t − 1 do
2     C[i + j − 1] ← LDB(B[j]);
3     C[i + j] ← MUL(A[i]);
4     for i ← 1 to t − 1 do
5         ADD(C[i + j − 1]);
6         C[i + j − 1] ← MUL(A[i]);
7     ADD(C[i + j]);
8 C[i + j] ← LDB(0);
9 C[i + j + 1] ← LDB(0);
```

can be used to initiate the reduction process, thus reducing the runtime of a combined multiplication and subsequent reduction operation by two cycles. The number of cycles needed for field multiplication is

$$t_{cycles} = (2 * t + 1) * t + 3.$$

The runtime of the multiplication is quadratic in nature.

### Product scanning form

The product scanning form computes all partial products, that contribute to a certain digit of the result and adds them up, hence it's name. It scans through the product determining one digit of the result at the time. This reduces the number of memory write operations.

***Algorithm 7.5***: *Multiplication (product scanning form)*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \cdot b; deg(c) \leq 2m - 2$

```
1  R0, R1, R2 ← 0;
2  for k ← 0 to 2t − 2 do
3      foreach {(i, j)|i + j = k, 0 ≤ i, j ≤ t − 1} do
4          (UV) ← A[i] · B[i];
5          (ϵ, R0) ← R0 + V;
6          (ϵ, R1) ← R1 + U + ϵ;
7          R2 ← R2 + ϵ;
8      C[k] ← R0, R0 ← R1 ← R2, R2 ← 0;
9  C[2t − 1] ← R0;
10 return C;
```

Algorithm 7.5 gives a detailed description of the product scanning form and figure 7.3 is a visualization of the necessary steps to compute the product of two 3-digit numbers.

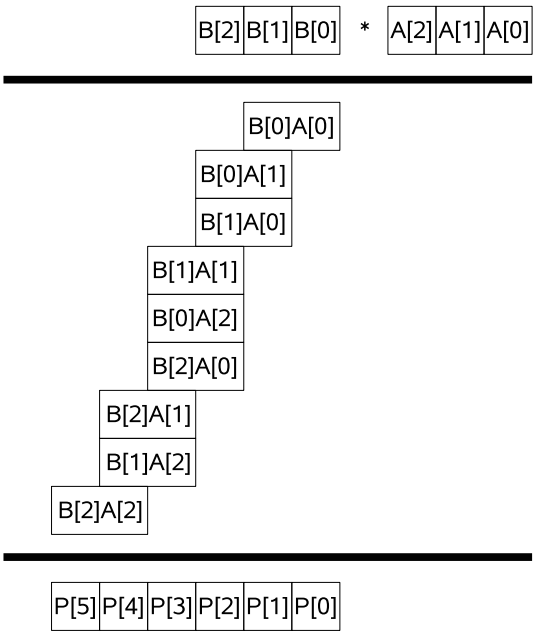



**Figure 7.3:** *An illustration of the product scanning form for two 3-digit integers*

It is possible to implement the multiplication using the product scanning form with the *Complex* ISA. The given architecture produces one partial result in two clock cycles, which is optimal. Also, it needs less write cycles than the operand scanning form because it fully determines one product digit before writing it back to memory. The inherent advantage is counterbalanced by a very complex pattern of operand read operations.

To better understand the runtime analysis, algorithm 7.6 presents a loop-unrolled variant of hardware-specific version of the product scan algorithm for 3-digit operands.

At a first glance at the procedure it becomes apparent that the control logic for this algorithm is more complex, than that of the operand scanning form. The price paid in control logic complexity is rewarded by a shorter runtime.

From the structure of algorithm 7.6, one can see that the computation of the first product digit takes 2 cycles to complete, the second needs 3 cycles, the third 5, the fourth 3 again, the fifth 2 and finally one extra cycle is needed to write back the highest order result. This behavior mirrors the block nature of the multiplication groups depicted in figure 7.3. Studying it, it becomes clear that the first digit of the product needs one multiplication, the second two, and so forth.

After the central block of multiplications the pattern repeats. Only one operation is required to produce the first temporary result of a block, because of the overlapping operand indices. Every other multiplication in a block needs two cycles to compute. These observations lead to the following equation to

***Algorithm 7.6***: *Multiplication (product scanning form) for 3-digit operands (loop unrolled)*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \cdot b; deg(c) \leq 2m - 2$
1 $LBS(A[0])$;
2 $MAC(B[0])$;
3 $C[0] \leftarrow MAH(B[1])$;
4 $LDB(A[1])$;
5 $MAC(B[0])$;
6 $C[1] \leftarrow MAH(B[1])$;
7 $LDB(A[0])$;
8 $MAC(B[2])$;
9 $LDB(A[2])$;
10 $MAC(B[0])$;
11 $C[2] \leftarrow MAH(B[1])$;
12 $LDB(B[2])$;
13 $MAC(A[1])$;
14 $C[3] \leftarrow MAH(A[2])$;
15 $C[4] \leftarrow SRW()$;
16 $C[5] \leftarrow SRW()$;

determine the runtime of algorithm 7.6

$$t_{cycles} = 2 \cdot \left[ \left\{ \sum_{i=2}^{t} (i \cdot 2) - 1 \right\} + 2 \right] + 1.$$

**Comparison and conclusion**

The product scanning form has a clear advantage over the operand scanning form when it comes to the number of clock cycles needed to compute the field multiplication. Table 7.5 gives an overview of cycle times for different MAC input widths. The price for the higher speed is paid with a more complex

| MAC operand width [Bits] | 8 | 12 | 16 | 20 | 24 |
|---|---|---|---|---|---|
| Words for 163 Bit EC | 21 | 14 | 11 | 9 | 7 |
| Op. Scan [cycles] | 906 | 409 | 256 | 174 | 108 |
| Pr. Scan [cycles] | 844 | 368 | 224 | 148 | 88 |

**Table 7.5:** *Multiplication runtime comparison of operand and product scanning form.*

control logic and ALU implementation.

## 7.5 Squaring

Squaring in a binary field is a much simpler operation than multiplication. Informally, to square a binary polynomial is to splay it to double size, inserting a zero at every other position into the binary string representation. For a more mathematically inclined description, as found in [HMV04], consider a binary

polynomial $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$. The squared polynomial $a(z)^2$ has the following structure:

$$a(z)^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0.$$

The binary representation will take the form outlined in figure 7.4. The length of the binary representation is again $2m - 1$ bits, the same as with the multiplication result.

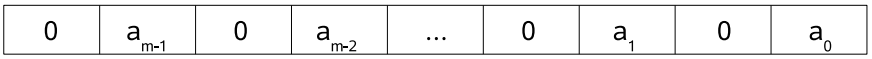



**Figure 7.4:** *Squared binary polynomial* $a(z)^2$.

### 7.5.1 Direct hardware implementation

The obvious solution to the squaring problem is to use rewiring in conjunction with a set of multiplexers to choose the correct input signal and an array of AND gates to mask every other input. The hardware costs for such an implementation are minimal. The power consumption, the other important factor to consider, could also be minimized by circumventing the hardware multiplier unit using operand isolation. This would pose no advantage, as deactivation of the multiplier leads to unwanted fluctuations in the power consumption (refer to §6.1).

### 7.5.2 An alternative squaring method

There exists an implementation that manages the squaring operation without additional hardware resources and that utilizes the hardware multiplication unit of the MAC [GK03a]. The strategy used by this squaring procedure is best explained using figure 7.5. It illustrates the application of the product scanning form multiplication algorithm to the polynomial squaring problem. The cancellation effect in the illustration is due to the following facts:

1. $(\mathbb{F}\backslash\{0\}, \cdot)$ is an Abelian group. Therefore, the multiplication is commutative $a \cdot b = b \cdot a$, $\forall a, b \in \mathbb{F}_{2^m}$.

2. As explained in §7.3, addition in a binary extension field is equivalent to the XOR operation and performing the XOR operation on two identical terms always yields zero: $a \oplus a = 0$, $\forall a \in \mathbb{F}_{2^m}$.

It is easily discernible, that almost all partial products cancel each other. Only multiplications of factors with the same index remain. Applying this observation to the problem at hand yields a simple and fast method to solve it. Squaring a field element is reduced to squaring each of its digits, without the necessity to worry about dependencies between the intermediate results. The steps necessary to compute the final result include squaring of each word and writing back the almost double length result. The runtime of this elegant technique is linear with respect to the number of digits in the field element.

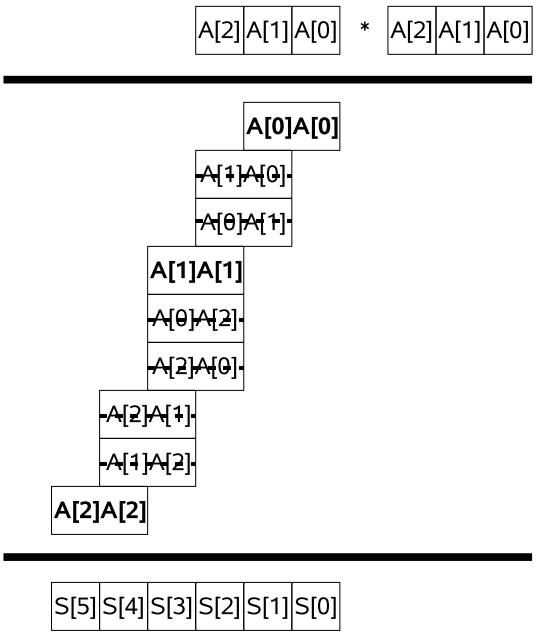



**Figure 7.5:** *Squaring using the multiplication unit, product scanning style*

### 7.5.3 Implementation

Algorithm 7.7 presents a *Simplex* ALU specific implementation of the procedure described in §7.5.2. Adapting it to the ISA of the *Complex* ALU is reasonably easy, therefore only one variant is given. The procedure is indeed very similar to the operand scanning form multiplication method detailed in algorithm 7.4.

***Algorithm 7.7***: *Squaring* - Simplex - *implementation*

**Input**: $a \in \mathbb{F}_{2^m}$
**Output**: $c = a^2; deg(c) \leq 2m - 2$

```
1 LBS(A[0]);
2 MAC(A[0]);
3 for i ← 1 to t − 1 do
4 |   C[2i − 2] ← LDB(A[i]);
5 |   C[2i − 1] ← MAC(A[i]);
6 C[2t − 2] ← LDB(A[i]);
7 C[2t − 1] ← LDB(0);
```

A quick analysis of the necessary steps yields

$$t_{cycles} = 2t + 4$$

for a runtime estimation.

A square operation in $\mathbb{F}_{2^{163}}$without reduction using a 16-bit variant of the ALU would require 26 cycles to complete. That amounts to approximately 10%

of the time needed to perform one multiplication without reduction. Fortunately, this is only marginally worse than what could have been achieved with dedicated square hardware, as the write bus limits the speed of an inner square operation[1] to at least two cycles. As with the multiplication, it is again possible to use the last two steps in the square operation to implement the first two instructions of the reduction. In doing so the number of cycles needed to compute the combined operation is reduced by two.

### 7.5.4 Conclusion

The squaring algorithm 7.7 seems to be the most appropriate choice. The additional resources for a direct hardware implementation are diminutive, but the small speed-up does not justify them. The possibility to switch off the multiplication unit would be a good way to conserve energy, but in this application scenario, this poses no advantage. It might even be detrimental because it would lead to unwanted fluctuations in the power consumption of the circuit. This could interfere with the load-modulation based communication.

## 7.6 Modular reduction

The output polynomials of the multiplication and squaring operations $c(z)$ $deg\{c(z)\} \leq 2m-2$ are not in the base representation of an element of the field $\mathbb{F}_{2^m}$. This necessitates the modular reduction of $c(z)$. As detailed in §4.3.5, this is equivalent to computing $c(z) \, mod \, f(z)$, where $f(z)$ is the irreducible polynomial defining the field. The modular reduction will henceforth also be denoted simply by the term reduction.

### 7.6.1 Methodologies - Interleaved versus stand-alone reduction

There exist two paradigms to apply the reduction [DF96]. Option one is to reduce the intermediate results of the multiplication or squaring operations while performing the respective procedure. The other possibility is to reduce the final result of those operations.

The primary advantage of the first method is that the intermediate results $c_{temp}$ are length limited to $deg(c_{temp} \, mod \, f(z)) \leq m-1$. This is beneficial if bit-parallel word-serial multiplication schemes are used, because it helps to reduce the size of the accumulator. The second advantage is the possible speedup compared to separated reduction. In both word- and bit-serial architectures, it is possible to efficiently implement interleaved reduction. Examples for these techniques can be found in [TWA05] for bit-parallel word-serial and in [Wol04] for bit-serial multiplication.

The benefits of the immediate reduction method are difficult to translate to the architecture chosen for this work. The main drawback of performing a reduction after a multiplication or squaring operation is finished, is that the result takes up two element slots in memory. This is mainly detrimental to the

[1] The squaring of a digit.

area requirements but not to the peak power consumed per clock cycle[2].

Immediate reduction could alleviate this problem, but at a first glance only at considerable cost to runtime. Reduction requires a result the degree of which exceeds the allowed bounds. Utilizing the operand scanning form multiplication that applies to all partial products $a \cdot B[i]$. Using a reduction method similar to that presented in section §7.6.4, reduction of the partial product would need an increasing number of multiplications, depending on the index $i$ of the partial product $a \cdot B[i]$.

Intuitively, one could estimate the runtime in the range of $O(n^2)$ as opposed to the linear relation of the "reducing the final product" scheme. For an implementation that uses this approach, confer to [BL06].

At a second glance, or more correctly after extensive study of the problem, the author of this work arrived at a solution that allows both multiplication and squaring with interleaved reduction, without derogative effects on the speed. The idea was inspired by the Montgomery multiplication[3], as presented in [GK03b]. To facilitate understanding, it is beneficial to consider the other options to implement the modular reduction.

### 7.6.2 Hard-wired reduction

The reduction after computation approach is implementable by a special reduction unit using hard-wired reduction [PLP07]. This method is very fast and has relatively moderate hardware requirements. Unfortunately, there are several severe drawbacks, which inhibited the application of this idea. For an illustration of the operation the reduction unit has to implement, refer to figure 7.6.

The hard-wired reduction module works on the entire target polynomial at once. As the input is a polynomial of a potential degree up to $2m - 2$, the unit would require its own bus system. This and the broad data path impose area and peak power requirements that exceed the acceptable limits.

Apart from that, a structure as large as this would suffer from the same undesirable accidental load modulation condition, that was discussed in §6.1. Considering that the width of an operand is 325 bits, the problem would in all probability even be worse.

It is possible to break down this process to word length. This is exactly what the repeated multiplication reduction algorithm presented in §7.6.4 does.

### 7.6.3 Software reduction algorithms

Next, the applicability of the software reduction procedures detailed in [HMV04] was inspected. These algorithms are specifically tailored to one of the NIST reduction polynomials proposed in [Fed00]. Furthermore, they are optimized for general-purpose processors that lack native $\mathbb{F}_{2^m}$ arithmetic operation support.

An analysis of the appropriate algorithm showed that this method could only be applied to the ISAs introduced in §7.2 with the addition of elaborate shifting capabilities to the accumulator. Although possible, the reduction with this technique would be exceedingly slow and the control logic for the operation would be cumbersome to say the least.

[2]The additional memory only adds to the static power dissipation, because clock gating will be used.

[3]Not to be mixed up with the Montgomery point ladder.

During deliberation of an adaption of the software reduction algorithm, the author of this work devised a method of reduction that was termed repeated multiplication reduction by [PLP07]. For a detailed description of this technique, confer to [GSE+02]. The application domain for this methodology are flexible solutions that support more than one binary field. Surprisingly, this algorithm class produces a nearly perfectly suited incarnation for the NIST reduction polynomial $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$ of the finite field $\mathbb{F}_{2^{163}}$ in conjunction with the ISAs presented in §7.2.

### 7.6.4 Repeated multiplication reduction

As stated in [HMV04] and albeit a little different than [GSE+02], the irreducible polynomial

$$f(z) = z^m + r(z),\ deg\{r(z)\} \leq m - 1,$$

where $r(z)$ is called the reduction polynomial. The result polynomial

$$c(z) |\, deg\{c(z)\} \leq 2m - 2$$

is congruent to

$$\begin{aligned} c(z) &= c_{2m-2}z^{2m-2} + \cdots + c_m z^m + c_{m-1}z^{m-1} + \cdots + c_1 z + c_0 \\ &\equiv (c_{2m-2}z^{m-2} + \cdots + c_m)r(z) + c_{m-1}z^{m-1} + \cdots + c_1 z + c_0 \ (mod\ f(z)). \end{aligned}$$

By using the NIST-specified curve for 163 bits, the irreducible polynomial is defined as $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, and $r(z) = z^7 + z^6 + z^3 + 1$. So, by multiplying $r(z)$ and $c_H = (c_{2m-2}z^{m-2} + \cdots + c_m)$, and adding the result to $c_L = c_{m-1}z^{m-1} + \cdots + c_1 z + c_0$ a new temporary output is derived which only has a degree of $m = 168$. The process described above is then repeated once more and the final reduced result is computed. Confer to figure 7.6 for a visualization of the necessary procedure.

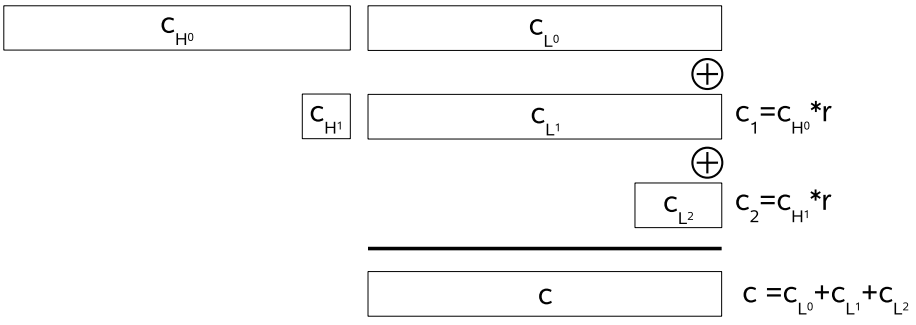



**Figure 7.6:** *Two step reduction for the NIST-163 elliptic curve*

Großschädl and Kamendje introduced an almost identical method in [GK03a]. Their reduction algorithm uses partial reduction, as its projected field of operation are small microprocessors with instruction-set extensions. Due to the fixed memory word size[4] of such processors, there is always a certain amount of unused bits when mapping an element to a memory array. Partial reduction stops

[4] Usually a multiple of eight.

as soon as the element fits into the $t$-word representation. This representation is congruent to an element in $\mathbb{F}_{2^m}$.

If the word width of the MAC equals or exceeds 6 bits, it is possible to compute the reduction with one $element \cdot word$ and two $word \cdot word$ multiplications. If the MAC width is equal to or greater than 7 bits, the same is possible with one $word \cdot word$ multiplication less.

***Algorithm 7.8***: *Repeated multiplication reduction - Implementation*

**Input**: $c(z)|deg(c(z)) \leq 2m-2, r(z)|r(z) = z^7 + z^6 + z^3 + 1$
**Output**: $c(z) \in \mathbb{F}_{2^{163}}$

```
1  LDB(r(z));
2  MAC(C_H[0]);
3  for i ← 0 to t − 2 do
4  |  ADD(C_H[i]);
5  |  C_L[i] ← MAC(C_H[i + 1]);
6  ADD(C_H[t − 1]);
7  C_L[t − 1] ← LDB(r(z));
8  MAC(C_H[0]);
9  ADD(C_L[0]);
10 C_L[0] ← LDB(0);
```

Algorithm 7.8 details the steps necessary to perform the repeated multiplication reduction with the *Simplex* ISA presented in §7.2. The word width $w$ of the implementing ALU must be at least 8.

The memory read and write operations in the reduction procedure necessitate a complex bus logic for the double-width result register file. As for multiplication and squaring, a contiguous write is necessary. The result registers are addressed as one file of almost the double size of a field element. To greatly facilitate the reduction process, it is essential to be able to read $C_H$ and $C_L$ as self-contained register files, independent of their physical position in the result registers. For the multiplication and squaring operations continuous reading is mandatory. This difficulty was termed the $C_H$-selection problem by the author of this work.

Assuming that the second reduction level polynomial is small enough to fit into $\mathbf{ACC_L}$, the correct runtime is given by this equation

$$t_{cycles} = 5 + 2 \cdot (t - 1). \tag{7.1}$$

Optimizations due to combining multiplication and squaring with reduction are taken into account[5]. For a 16-bit MAC, that amounts to 25 clock cycles, or approximately 10% of the runtime of a multiplication without reduction.

### 7.6.5 Multiplication with interleaved reduction

The product scanning form multiplication in conjunction with the repeated multiplication reduction introduced above enable the efficient implementation of interleaved reduction. The reason for this is the property of the Comba multiplication method to generate one digit of the product at a time.

[5] Otherwise the cycle count would be $t_{cycles} = 7 + 2 \cdot (t - 1)$.

For a multiplication $c = a \cdot b$, $a, b \in 2^{163}$, the $11^{th}$ partial product computed with the product scanning form contains the polynomial $c_{175}z^{175}+\ldots+c_{160}z^{160}$. In accordance with the nomenclature used in figure 7.6 the polynomial $c_{162}z^{162}+\ldots+c_{160}z^{160}$ is the last digit of $C_L^0$ ($C_L^0[10]$), while $c_{175}z^{175}+\ldots+c_{163}z^{163}$ is the first part of $C_{H^0}$.

The lower three bits representing $C_L^0[10]$ are normally stored in the last word of the memory element that will contain the final product. The lower part of $C_{H^0}$ is saved in temporary variable **RC**.

The next partial product ($C[11]$) that is ascertained will again be split at the three bit boundary. This time the higher 13 bits ($c_{191}z^{191}+\ldots+c_{176}z^{176}$) are written to a second temporary space **MC**. The lower three bit are then combined with the 13 bits carried over from the last round which are currently stored in **RC**.

Together they constitute the 16-bit word $c_{178}z^{178}+\ldots+c_{163}z^{163}$, the first digit of $C_H^0$. This word is restored in the accumulator. In the next step it is multiplied with $r(z)$: $t = w \cdot r$. The lower 16 bit of the result in $\mathbf{ACC_L}$ are added to $C_L^0[0]$. The carry of the multiplication in $\mathbf{ACC_H}$ is than swapped with the content of **MC**. Thus, **MC** alternately stores the carry of the normal multiplication and the one of the reduction multiplication.

Then the next partial product ($C[12]$) is computed, and the process is repeated. Thus, the repeated multiplication reduction is performed step by step as the partial products of $C_{H^0}$ and after that $C_{H^1}$ become available.

Algorithm 7.9 describes all the steps that are necessary to compute the Comba multiplication with interleaved reduction. A wide array of special purpose instructions is required to implement this method. Thus, only the *Complex* ISA, which in fact is specially tailored to this application, is capable to perform the operation.

The function $cpi(x, y)$ simply returns the difference between the two input values $cpi(x, y) = y - x$. To minimize the number of read operations for the product scanning form multiplication, the operands have to be read in a specific order, as explained in§7.4.2. This optimal sequence of read operations, is computed by the $ind_{i,j}$ function.

The runtime of this multiplication method is given by the following equation

$$t_{cycles} = 11 + 4 \cdot (t-1) + \left(\sum_{i=1}^{t-1} 2 \cdot i\right) + \left(\sum_{i=t-2}^{1} 2 \cdot i\right)$$

For the 16-bit *Complex* ALU, the number of cycles totals to $t_{cycles} = 251$, which is substantially faster than an operand scanning form multiplication with subsequent reduction ($281 = 256 + 25$), and only marginally slower than an optimal Comba multiplication/modular reduction pair ($249 = 224 + 25$).

At the same time, it requires one element-memory space less. The analysis of the Complex ISA showed that two additional registers, the 13 bits wide **RC** and the 15-bit storage unit **MC** are necessary. The net area gain for $\mathbb{F}_{2^{163}}$is therefore equivalent to the size of 136 flip-flops. This method would also work for the NIST curves $\mathbb{F}_{2^{283}}$ and $\mathbb{F}_{2^{571}}$, where the area gain would be accordingly larger.

***Algorithm 7.9***: *Multiplication (product scanning form) with interleaved reduction* - Complex *implementation*

**Input**: $a, b \in \mathbb{F}_{2^m}$
**Output**: $c = a \cdot b \, mod \, f, c \in \mathbb{F}_{2^m}$

```
1  LDB(A[0]);
2  MAC(B[0]);
3  i ← 0, j ← 0;
4  for i ← 1 to t − 1 do
5  |  MAR(B[cpi(ind_{i,j}, i)]) ;
6  |  for j ← 1 to i do
7  |  |  LDB(A[ind_{i,j}];
8  |  |  MAC(B[cpi(ind_{i,j}, i);
9  T[t − 1] ←= SMR(0);
10 i ← 0, j ← 0;
11 LDB(A[ind_{i,j}]);
12 for i ← 0 to t − 2 do
13 |  if i ≠ 0 then
14 |  |  C[i − 1] ← MSM(B[cpi(ind_{i,j}, t + i)]);
15 |  else
16 |  |  MAC(B[cpi(ind_{i,j}, t + i)]);
17 |  for j ← 1 to (t − 2) − i do
18 |  |  LDB(A[ind_{i,j}]);
19 |  |  MAC(B[cpi(ind_{i,j}, t + i)]);
20 |  SAA(T[i]);
21 |  MAR();
22 C[t − 2] ← SMC();
23 SAA(0);
24 MAR;
25 SRD(T[i]);
26 C[i] ← RAC(C[0]);
27 MAR();
28 C[0] ← SRW();
   // The next two lines are only required for ALUs with a word
      width of 8, 11, 12, 14, 15, 21, 24 or 28 bits
29 SMA(C[1]);
30 C[1] ← SRW();
```

### 7.6.6 Squaring with interleaved reduction

The square operation with interleaved reduction algorithm 7.10 utilizes the same principle as the multiplication with interleaved reduction method. The square is computed as outlined in §7.5.2 until the first result exceeds the 163-bit limit. Then, similar to its multiplication counterpart, the algorithm alternates between computing the reduction and the next partial product of the temporary result.

***Algorithm 7.10***: *Squaring with interleaved reduction* - Complex *implementation*

**Input**: $a \in \mathbb{F}_{2^m}$
**Output**: $c = a^2 \, mod \, f(z)$, $c \in \mathbb{F}_{2^m}$

```
1  LBS(A[0]);
2  MAH(A[0]);
3  i ← 1, j ← 0;
4  for i ← 1 to t/2 do
5  |  T[j++] ← LBS(A[i]);
6  |  T[j++] ← MAH(A[i]);
7  T[t−1] ← SMR(T[0]);
8  j ← 0;
9  for i ← ⌊t/2⌋ + 1 to t − 1 do
10 |  SLB(A[i]);
11 |  MAR();
12 |  C[j++] ← MSM(A[i]);
13 |  SAA(T[j]);
14 |  MAR();
15 |  if i = t − 1 then
16 |  |  C[j++] ← SMC();
17 |  else
18 |  |  C[j] ← SAM(T[++j]);
19 SAA(0);
20 MAR();
21 SRD(T[j]);
22 C[j] ← RAC(C[0]);
23 MAR();
24 C[0] ← SRW();
   // The next two lines are only required for ALUs with a word
      width of 8, 11, 12, 14, 15, 21, 24 or 28 bits
25 SMA(C[1]);
26 C[1] ← SRW();
```

The procedure detailed by algorithm 7.10 only works if $t \, mod \, 2 = 1$. For word widths $w$ that result in a $t$ such that $t \, mod \, 2 = 0$, a slightly different variant is required. It will not be discussed here, as it has the same characteristics. The number of cycles necessary to compute one result is given by

$$t_{cycles} = 9 + 2 \cdot \lfloor t/2 \rfloor + (t - 2 - \lfloor t/2 \rfloor) \cdot 6.$$

For a 16-bit ALU implementation of the *Complex* ISA $t_{cycles} = 49$ which is exactly the same as is achievable with two separate operations for squaring and

reduction. This algorithm is required to achieve the area gain outlined for the multiplication with interleaved modular reduction procedure.

## 7.7 Inversion

The inversion of an element in $\mathbb{F}_{2^m}$ is the most complex field operation. Inversion is required to implement division in a field. Recall that $a/b = a \cdot b^{-1}$, $\forall a, b \in \mathbb{F}_{2^m}$.

Again, there are two completely different approaches to perform a field inversion. This section compares several variants of the Extended Euclidean Algorithm (EEA) and the Fermat based inversion and discusses possible hardware implementations thereof.

### 7.7.1 Extended Euclidian Algorithm based inversion

The Extended Euclidean Algorithm for binary polynomials computes the Greatest Common Divisor (GCD) of two elements $a, b \neq 0$, $a, b \in \mathbb{F}_{2^m}$ and is denoted by $d = gcd(a, b)$. The GCD $d$ is the polynomial of highest degree that divides both $a$ and $b$. The EEA is an extension to the GCD that computes two additional elements $g$ and $h$, satisfying $ag + bh = d \, mod \, f$, where $d = gcd(a, b)$. If $a$ is the number of which the inverse is sought, and $b = f$ is the irreducible polynomial of the field $\mathbb{F}_{2^m}$then, $d = gcd(a, f) = 1$, $\forall a \in \mathbb{F}_{2^m}$, $a \neq 0$. Therefore, $g$ is the inverse of $a$, because $ag + bh = d \, mod \, f \equiv ag \equiv 1 \equiv g = a^{-1} \, mod \, f$. For a description of the basic principles of the EEA algorithm, please refer to a textbook like [WT02].

The basic EEA method, as can be found in [HMV04] for example, requires polynomial degree comparison. This is difficult to implement in hardware, therefore a modified approach known as the binary inversion algorithm[6] is more appropriate. It is based on three observations [Lai04]:

1. If $a$ and $b$ are both even, $gcd(a, b) = 2 \cdot gcd(a/2, b/2)$
2. If $a$ is even and $b$ is odd, $gcd(a, b) = gcd(a/2, b)$
3. If both $a$ and $b$ are odd, $gcd(a, b) = gcd(|a - b|/2, b)$

According to [HMV04], which presents the binary field variant of the binary inversion algorithm, the necessary degree calculations can now be reduced to a comparison of the involved values.

A direct hardware implementation of the comparison is still very resource intensive, which led to the development of the current state-of-the-art EEA method for low power, low-die-size applications: the Counting EEA algorithm. It was first introduced in [Lai04] and later adapted by Fürbass for use in an ECC processor for RFID [FW07b].

Algorithm 7.11 is the binary field version of the procedure presented in [FW07b]. The additional exit condition is necessary because in $\mathbb{F}_{2^m}$, $u \oplus v/2 = 0 | \, u = 1, v = 1$, and not $(1 + 1)/2 = 1$ as in $\mathbb{F}_p$.

A direct translation of the Counting EEA technique to an architecture with a word width that differs from the element size is cumbersome. The additional

[6] Take care not mix up the Binary EEA inversion algorithm with inversion in a binary field!

***Algorithm 7.11***: *Counting EEA using reduction polynomial*

**Input**: $a(z)$, $b(z) | \, a(z), b(z) \in \mathbb{F}_{2^m}$
**Output**: $x(z) = a(z) \cdot b(z)^1 \, mod \, f(z)$

```
1  u ← a(z); v ← f(z); x ← b(z); y ← 0; k ← 0;
2  while u ≠ 1 do
3      while u_0 = 0 do
4          u ← u ≫ 1;
5          x ← (x ⊕ x_0 · f(z)) ≫ 1;
6          k ← k + 1;
7      if k ≤ 0 then
8          u ↔ v; x ↔ y;
9          k ← −k;
10     u ← (u ⊕ v) ≫ 1;
11     if u = 0 then
12         break;
13     x ← (x ⊕ y ⊕ (x_0 ⊕ y_0) · f(z)) ≫ 1;
```

instructions needed to augment the ISAs detailed in §7.2 are limited to a 1-bit shift-right instruction, which is negligible. Runtime is the issue. Due to the small-word-size paradigm chosen for this work, almost all operations in the algorithm are slowed down by a factor of $t$, where $t$ is the number of words needed to represent an element.

A conservative estimation of the runtime yielded

$$t_{cycles} = [4 \cdot t + 7 + (4 \cdot t + 3) \cdot l_2] \cdot l_1$$

to compute the cycle count, where $l_1$ is the average number of outer loop executions and $l_2$ the average number of inner loop executions. A simulation with one million test cases produced the values $l_1 \approx 206$, and not surprisingly $l_2 \approx 1$. This amounts to a total of $t_{cycles} \approx 20,188$ for one inversion in $\mathbb{F}_{2^{163}}$ on a 16-bit ALU. That is $80.43 \times t_{mul}$, where $t_{mul}$ is the time required for one multiplication with interleaved modular reduction.

An efficient implementation of the inversion using a distinct hardware unit, would drastically reduce the cycle-count into the range of the multiplication runtime. Unfortunately, the resources required to build such a module include: A full element-width bus, a full precision accumulator and an full precision XOR-adder. Setting aside the additional strain this would put on area size and power management, it would again lead to the same set of problems already discussed in section §6.1.

### 7.7.2 Fermat based inversion

Remember that the inverse of an element in $\mathbb{F}_{2^m}$ can be computed using Fermat's little theorem $a^{2^m-2} \equiv a^{-1} \, mod \, f(z)$. Thus, the inversion problem is reduced to an exponentiation.

There are several computational approaches to exponentiation. Setting the trivial multiplicative technique aside, the square-and-multiply algorithm for ex-

***Algorithm 7.12***: *Square-and-multiply MSB-first method*

**Input**: $a(z) \in \mathbb{F}_{2^m,x}$
**Output**: $a(z)^x$
1 $d(z) \leftarrow 1$;
2 **for** $i \leftarrow m-1$ **to** 0 **do**
3  **if** $x_i = 1$ **then**
4  $d(z) \leftarrow a(z) \cdot d(z)^2 \, mod \, f(z)$
5  **else**
6  $d(z) \leftarrow d(z)^2 \, mod \, f(z)$

ponentiation is probably the best known (algorithm 7.12). It requires $m$ square operations in total and one multiplication for every bit set in the exponent.

Unfortunately, the reduction exponent $2^{163} - 2$ of the field used by this work has 162 of its 163 bits set to one. This amounts to a runtime of 163 square operations and 162 multiplications, which is very inefficient as multiplications have a relatively high cycle count (cf. §7.4.2).

Itoh and Tsujii proposed a recursive method to exponentiation in [IT88]. It focuses on reducing the number of multiplications. Although they use normal basis representation, which has the advantage of especially fast squaring[7], the same premises apply to the problem at hand. The squaring operation in this architecture is also significantly faster than the multiplication. The algorithm they suggest has a complexity of 162 square operations and 18 multiplications for a 163-bit inversion exponent. The Itoh and Tsujii method works with arbitrary exponents and is ideally suited for either a hierarchical multiplier approach in hardware or a recursive software program. For the hardware architecture presented in this work, it has the severe handicap, that it would need 81 temporary element variables. Therefore, it is unacceptable.

To solve that problem and to further improve the cycle count, the structure of the fixed inversion exponent must be exploited. Ignoring the least significant bit of the exponent, which is not set, leaves the 162 upper bits for consideration. Applying the recursive algorithm to those bits, one can discern that all sub-trees are exactly the same. Thus, they can be merged, yielding an implementation that requires only 9 multiplications while the number of square operations stays constant at 162. This is the optimal achievable runtime complexity. Furthermore, this algorithm needs only three storage elements: one for the input value, one for the output value and one for temporary usage.

***Algorithm 7.13***: *Square-and-multiply: recursive method - inner inversion operation (*inInvOp(a(z), b(z), number)

**Input**: $a(z), b(z) \in \mathbb{F}_{2^{163}}, n$
**Output**: $a(z)^{2^n} \cdot b(z) \, mod \, f(z)$
1 **for** $i \leftarrow 0$ **to** $n$ **do** $a(z) \leftarrow a(z)^2 \, mod \, f(z)$;
2 $a(z) \leftarrow a(z) \cdot b(z) \, mod \, f(z)$;

[7] One cyclic shift.

***Algorithm 7.14***: *Square-and-multiply: fully recursive method*

**Input**: $a(z) \in \mathbb{F}_{2^{163}}$
**Output**: $c(z) = a(z)^{2^{163}-2}$

```
1  c(z) ← a(z);
2  inInvOp(c(z), a(z), 1)d(z) ← c(z);
3  inInvOp(c(z), d(z), 2);
4  c(z) ← c(z)^2 mod f(z);
5  c(z) ← c(z) · a(z) mod f(z);
6  d(z) ← c(z);
7  inInvOp(c(z), d(z), 5)d(z) ← c(z);
8  inInvOp(c(z), d(z), 10)d(z) ← c(z);
9  inInvOp(c(z), d(z), 20)d(z) ← c(z);
10 inInvOp(c(z), d(z), 40);
11 c(z) ← c(z)^2 mod f(z);
12 c(z) ← c(z) · a(z) mod f(z);
13 d(z) ← c(z);
14 inInvOp(c(z), d(z), 81)d(z) ← c(z);
15 c(z) ← c(z)^2;
```

The only drawback of the implementation described by algorithm 7.14 is, that it needs the additional temporary variable $d(z)$. Depending on the choice of the ECC level scalar point multiplication scheme, the number of variables required by the inversion algorithm directly contributes to the total memory size. For this case, a modified variant of the recursive approach was developed. It combines the k-ary method detailed in [Gor98] and the optimization idea presented by Guarjardo et al. in [GP97] with the recursive method of Itoh and Tsujii.

***Algorithm 7.15***: *Square-and-multiply: recursive, k-ary method*

**Input**: $a(z) \in \mathbb{F}_{2^{163}}$
**Output**: $c(z) = a(z)^{2^{163}-2}$

```
1  c(z) ← a(z);
2  for i ← 0 to 1 do  a(z) ← a(z)^2 mod f(z); a(z) · c(z) mod f(z);
3  c(z) ← a(z);
4  for i ← 0 to 1 do
5  |   for i ← 0 to 2 do c(z) ← c(z)^2 mod f(z);
6  |   c(z) · a(z) mod f(z)
7  a(z) ← c(z);
8  for i ← 0 to 9 do a(z) ← a(z)^2 mod f(z);
9  a(z) · c(z) mod f(z) c(z) ← a(z);
10 for i ← 0 to 7 do
11 |   for i ← 0 to 17 do c(z) ← c(z)^2 mod f(z);
12 |   c(z) · a(z) mod f(z)
13 c(z) ← c(z)^2 mod f(z);
```

The procedure presented in algorithm 7.15 has a runtime complexity of 162

square operations and 13 multiplications. This amounts approximately to a 10% trade-off compared to the recursive method, but this variant only needs two variables in total. The original value in $a(z)$ is overwritten, so the advantage vanishes, if the original $a(z)$ is required after the inversion.

Notice that in both algorithm 7.14, and algorithm 7.15 the reduction exponent is hard-coded into the structure of the inversion procedure. This is an added bonus, as it frees one slot in the constant memory (ROM).

### 7.7.3 Conclusion

Comparing the different solutions discussed in this section, the selection of an appropriate inversion algorithm becomes rather simple. A direct translation of the state-of-art EEA variant on the MAC architecture, has no advantages over the Fermat based inversion methods. It even performs worse than the optimized Square-and-Multiply techniques, thus it is best discarded.

| Algorithm | Runtime [cycles] |
|---|---|
| Square-and-Multiply | 53509 |
| S-a-M (partially recursive, k-ary) | 11639 |
| S-a-M (fully recursive) | 10551 |
| Counting EEA | 20188 |

**Table 7.6:** *Runtime comparison of inversion algorithms on a 16-bit MAC core*

The speed increase due to an EEA hardware unit such as the one presented in [FW07b] would allow the use of the affine version of the Montgomery scalar multiplication. This would free up one 163-bit memory element. This area gain does not suffice to compensate the additional hardware costs.

If the extra hardware unit is not an option, and a procedure that recovers the y-coordinate is required then, the projective Montgomery-Point-Ladder using the recursive, k-ary Square-and-Multiply method, is the best choice. As the computation of the missing coordinate is the most memory consuming task, and incidentally contains the only occurrence of the inversion, this combination requires in total one temporary variable less .

ECDSA signature generation does not require the above mentioned y-coordinate recovery. Thus, the best scalar point multiplication procedure, with respect to area usage, is the projective Montgomery-Point-Ladder. For this combination, the purely recursive Square-and-Multiply algorithm is best suited, because in total, no extra temporary variable is necessary.

Chapter 8

# Implementation

This chapter discusses the implementation details of the *ECCon* processor. A comprehensive discussion of this subject would go beyond the scope of this work, therefore only the most prominent implementation decisions are highlighted herein.

The chapter starts by introducing the two power-saving methodologies paramount to the design process in §8.2. After that a discussion of the components of the RFID front-end (§8.3) and the constraints for clocking in an RFID environment (§8.3.2) is in order. The RFID section concludes with a deliberation of the realization details of the communication protocol implementation in §8.3.3. Next a comprehensive overview on the components related to the datapath of the ECC processor is given in §8.4 (ALU) and §8.5 (Memory), whereas §8.6 is concerned with the control unit.

Then the design for test strategies applied to the ASIC designed as part of this thesis and an analysis of their efficiency are discussed in §8.7. Side-channel attack resilience was one of the directives for this work and the measures taken to protect the circuit from such an attack are described in section §8.8. Finally, this chapter concludes with a concise presentation of the results of this thesis and a comparison with related work in §8.10.

The subjects discussed herein require fundamental knowledge of digital hardware design. It is assumed that the reader is familiar with the basic hardware building blocks, synchronous design, Complementary Metal-Oxide Semiconductor (CMOS), Very Large Scale Integration (VLSI) and related topics. For further information on these topics please confer to a textbook like [Kae08].

## 8.1 Conventions

The illustrations in this chapter use a fixed color scheme to distinguish different types of signals and logic blocks. The schemes are depicted in figure 8.1.

If space allows, plots might be supplemented with a partial reprint of the color scheme. All logic blocks of type sequential, combination and FSM require a clock, reset and clock enable signal. These are often omitted from the plots for reasons of clarity.

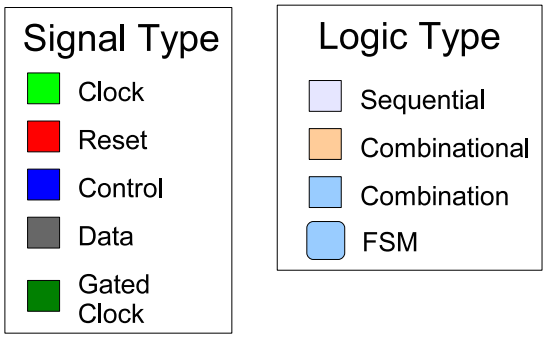



**Figure 8.1:** *Color scheme for signal and logic block types*

## 8.2 Power saving techniques

Power consumption in CMOS circuits is categorized into static power consumption and dynamic power consumption. Recall that CMOS technology mainly dissipates power when a state change occurs. Power usage due to this switching activity is called dynamic power consumption. The main sources for this are the charging and discharging of capacities and crossover currents during a state change.

CMOS technology does not only dissipates power when a state change occurs. Static power consumption, which is primarily caused by different variants of leakage currents, is a constant drain on the power budget, which is not due to switching activity. The simulated static power consumption of the *ECCon* processor does not exceed the nano-Watt range and is therefore deemed negligible.

### 8.2.1 Dynamic power consumption

The major part of dynamic power consumption stems from charging and discharging the input capacities of the logic gates and the wires of the digital circuit. Thus, the power dissipated can be estimated with this equation

$$P_{dyn} = \alpha \cdot f \cdot U^2 \cdot C,$$

where $\alpha$ is the switching probability, $f$ the frequency, $U$ the supply voltage and $C$ the switched capacitance of the circuit in question. Thus, the designer can reduce dynamic power dissipation by decreasing the switching probability and the frequency. The supply voltage and the gate capacities are VLSI technology specific. The total capacitance is dependent on the circuit.

### 8.2.2 Clock gating

Many registers in a digital circuit are not required to change their respective values at every clock cycle. To enable a register to retain its current value, a feedback loop with a multiplexer to select between the input signal and the fed back signal, is employed. This adheres to the rules of synchronous design but is wasteful.

A clock gate is a module, that controls the clock signal to a register. If the clock signal is deactivated, the register retains its value without necessitating an additional multiplexer. Furthermore, the number of clock tree nodes that are charged and discharged is reduced to one, instead of the number of flip-flops in the register. Figure 8.2 depicts a register that is clock gated with a clock gating unit composed of a latch and an AND gate.

Clock gating reduces the dynamic power consumption of the circuit by a factor of 7.4. For more details confer to §8.9.

### 8.2.3 Operand isolation

Operand isolation tackles the problem of unnecessary signal propagation. A combinatoric block computes a new result as soon as any of its inputs changes. If the result is not required, the block wastes energy on calculating it.

To prevent this, the inputs of a combinatoric function, the operands, need to be isolated from the combinatoric core. This is usually achieved by adding an AND gate for every input signal. Figure 8.3 illustrates this process using a polynomial adder as combinatoric block. Operand isolation is also known as sleep logic. Operand isolation is primarily employed by the memory implementation. For further details refer to §8.5.

## 8.3 RFID front-end

The RFID-front end consists of the RART and the RFID control unit. The RART is a third party module but it has been slightly modified to fit the prerequisites of the *ECCon* circuit. For an illustration of the structure of the RFID front-end refer to figure 6.1.

### 8.3.1 RFID Asynchronous Receiver Transmitter (RART)

This module has an *RFID air interface* compatible set of ports on one side and a synchronous, fully registered byte interface on the client component side.

The RFID air interface must provide a clock, a reset, a binary data-in and likewise a binary data-out port. The RART performs a binary-to-byte conversion in the receiver circuit and a byte-to-binary conversion in the sending part. Both operations adhere to the physical layer section of the ISO-18000-3-1 standard.

On the client side it supplies a byte-register that stores the last received byte. In transmitting mode, it saves the data value that should be sent next. In addition, it also provides a set of signals that notify a client component of important events. A mode select signal decides if the component is in receiver or transmitter mode.

### 8.3.2 Clock gate enable operation frequencies

The ISO-18000-3-1 standard [ISO04] defines a carrier frequency $f_c$ of 13.56 MHz. It is easy to derive integer fractions of $f_c$ for use as clock gate enable signals with a frequency: $f_{cge} = \frac{f_c}{i}$, $i \in \mathbb{N} \setminus \{0\}$. The RART module operates at a clock

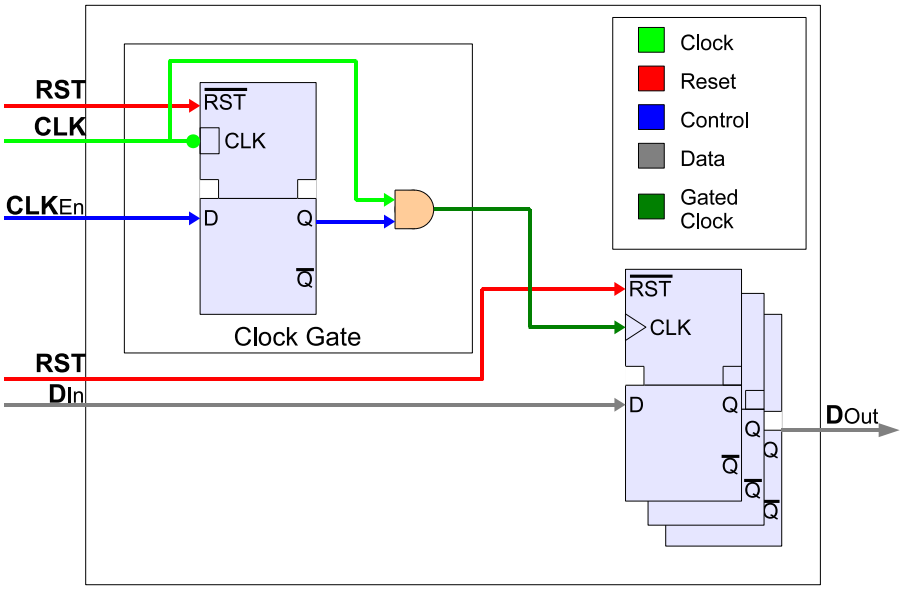



**Figure 8.2:** *Clock gated register*

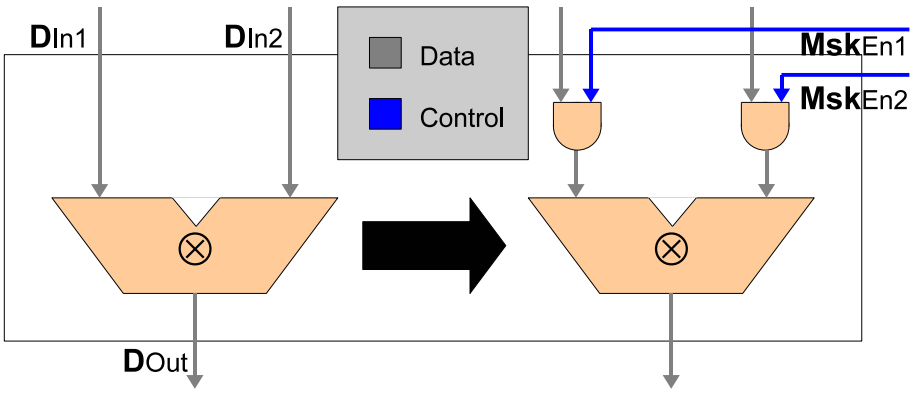



**Figure 8.3:** *Operand isolation*

frequency of $f_c/2 = 6.78$ MHz. The circuit is designed for single-edge-triggered one-phase clocking, where the low-to-high edge is the active one.

The *ECCon* circuit in general relies heavily on clock gating. The primary clock gate enable signals for the different components of the ASIC are created in the RART. These signals are then used by the RCU and the ECC processor to generate the clock gate signals for their registers. The primary clock enable signal for the RCU has a frequency of $f_c/128 = 105.9375$ kHz. The primary clock enable signal for the ECC processor has a frequency of $f_c/16 = 847.5$ kHz.

The RART already defines a clock gating unit that creates a clock gate enable signal a frequency of 106 kHz. This module was augmented with the capability to create the $f_c/16 = 847.5$ kHz clock gate signal for the ECC processor. This allows the calculation of one point multiplication in 0.4 seconds.

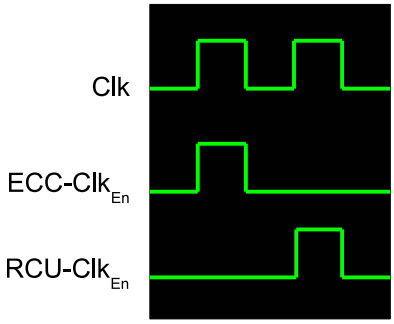



**Figure 8.4:** *Displaced clock gate enable signal creation*

The primary clock gate enable signals are created in a way, such that no two active edges for the RCU and the ECC processor overlap (refer to figure 8.4). Thus, the RCU and the ECC processor are never active at the same time. This limits the maximum amount of dynamic power required by the *ECCon* processor to the power dissipated in the RART and the ECC processor unit. The reason for this is that the consumption of the RART and the RCU combined is negligible in comparison.

### 8.3.3 RFID Control Unit (RCU) implementation

The RFID control unit implements the communication protocol outlined by the ISO-18000-3-1 standard. It does so in a straight-forward way with a complex FSM, a 5-bit counter and an 8-bit comparator. It is capable to execute the commands described in chapter §5, including the simpler version of the anti-collision protocol.

The FSM knows 67 distinct states. Implementation and verification required a month of work. Despite the number of states the implementation itself is rather plain. Because of this, the number of states and the greater relative importance of the ECC processor, no description of the FSM is given herein.

The control component is driven by events generated by the RART. Examples are the *Start Of Frame (SOF)* event, which signals that a new frame of data is about to arrive, the *byte received* event, which indicates that enough binary data arrived to be made available as byte sized chunk, and the *byte sent* event

which indicates that the next byte that should be transmitted, can be written to the send buffer.

The primary difficulty in designing a protocol unit for this standard, is to minimize power consumption while at the same time adhere to the stringent timing constraints imposed. The RART converts the bit stream the air interface provides into byte sized blocks. To minimize necessary storage the received byte must be evaluated before the next arrives. The same holds true in reverse for the case of the tag sending information to the reader.

Therefore, it is of great importance to implement the protocol with the minimal possible virtual clock frequency and the smallest, most power-efficient datapath. A minimum clock frequency is important to reduce dynamic power consumption (confer to §8.2.1). A small datapath reduces the required die area, and less components generally means less capacitance to charge and discharge.

The $f_{cge} = f_c/128 = 106$ kHz clock gate enable frequency already mentioned suffices to perform 32 operations between two *byte received* events. This is enough to implement the complex anti-collision scheme, the most time consuming mandatory operation of the ISO-18000-3-1 communication protocol, with a basic datapath such as the one outlined above. All the other commands, including the simpler anti-collision scheme, fit easily into one such slot.

## 8.4 ALU

The ALU of the ECC processor component of the *ECCon* ASIC is one of the three most crucial components of the circuit. The other two being the memory and the control logic of the ECC core. In §7, two different ISAs were introduced. Prospective implementations for both were developed and compared.

### 8.4.1 The *Simplex* ALU

The *Simplex* ISA design provides only a very limited set of instructions, hence the name. Likewise, the implementation is quite plain. Its detailed Register Transfer Level (RTL) structure is depicted in figure 8.5.

In its 16-bit incarnation, the circuit consists of:

1. Two 16-bit registers **B** and $\mathbf{ACC_L}$.
2. One 15-bit register $\mathbf{ACC_H}$.
3. One $16 \times 16$ $F_2$-polynomial multiplier $\otimes$.
4. One 16-bit $F_2$-polynomial adder $\oplus$, which is composed of 16 XOR gates.
5. A pair of 2-to-1 multiplexers to implement the adder input selection functions.
6. A shift unit capable to right-shift the accumulator by zero, eight or 16 bits.
7. 16 two-input AND gates.

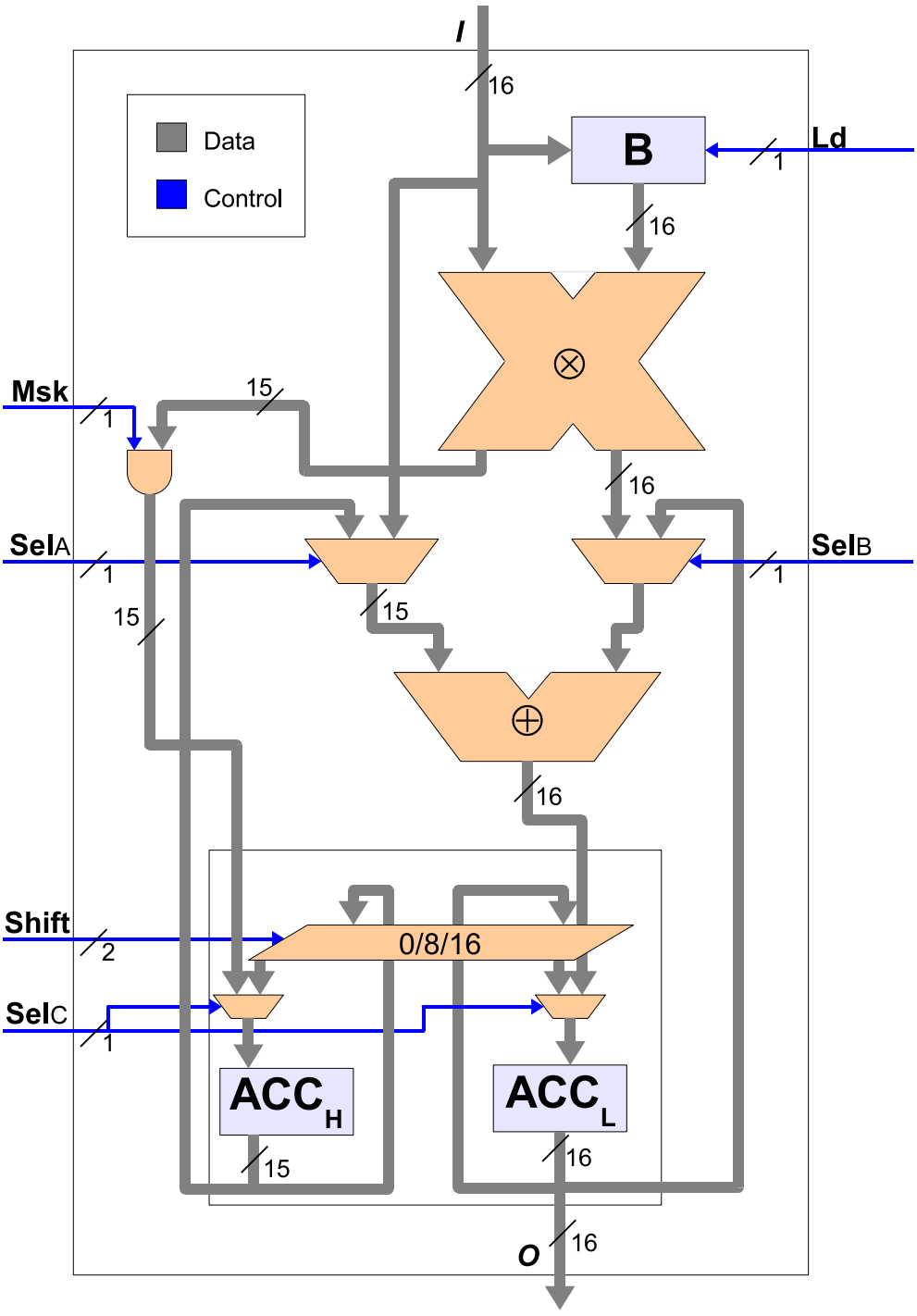



**Figure 8.5:** Simplex-*ISA* - *ALU implementation*

### 8.4.2 The *Complex* ALU

The ALU implementation of the *Complex*-ISA must support a greater variety of instructions in comparison with the *Simplex* architecture. Therefore, it is not very surprising that the concrete hardware realization is more complicated.

Figure 8.6 illustrates the interaction between the major components of the unit, whereas figure 8.7 depicts the select-and-add unit (SAA) in more detail. The select-and-add is not the most area consuming component, but it contains the most complicated datapath of all submodules of the ALU.

**Storage elements.** The module contains two 16-bit registers (**B** and $\mathbf{ACC_L}$), two 15-bit registers (**MC** and $\mathbf{ACC_H}$) and one 13-bit register (**RC**). $\mathbf{ACC_H}$ and $\mathbf{ACC_L}$ together compose the *accumulator* of the component. The accumulator contains a shift unit that is capable of a shift right by eight bits operation. All registers are clock gated if they are not active.

**The $16 \times 16$ $\mathbb{F}_2$-polynomial-multiplier ($\otimes$).** This component is implemented as an array multiplier. The addition part of the multiplication is handled by a tree-adder composed of XOR gates. It is the single most area intensive component of the ALU. The result of the multiplication is only 31 bits wide. This is due to

$$(a_{15}z^{15} + \ldots + a_0) \times (b_{15}z^{15} + \ldots + b_0) = c_{30}z^{30} + \ldots + c_0.$$

**The select-and-add unit.** The two 16-bit $\mathbb{F}_2$-polynomial adders ($\oplus$) are implemented by two times 16 2-input XOR gates. The module actually has two distinct outputs. It uses a set of multiplexer arrays to select the inputs of the addition units and optionally mask them with arrays of AND gates.

This composition as it is depicted in figure 8.7 describes the unit on a functional level. The actual hardware implementation differs, as the synthesizer replaces the multiplexers and AND gates with an and-or-invert gate based structure. A one-bit slice for the right hand side path is depicted in figure 8.8.

Any variable XORed with zero retains its value, thus the select-and-add unit can perform the select operation by masking one of the inputs to one of the two polynomial-adders, which will then perform the identity operation on its other input.

An example will clarify this further. See figure 8.7 and assume that both of the control signals for the AND gate arrays in the high path ($\mathbf{Msk}_{H0}$ and $\mathbf{Msk}_{H1}$) are set to zero. The output of the corresponding polynomial adder will therefore be zero. The multiplexer control signal **Sel** selects $\mathbf{ACC_H}$ as input for multiplexer $M_0$, while $\mathbf{Msk}_{H1} = 0$ masks the other input. Thus, a shift-accumulator-right-16 operation is achieved.

### 8.4.3 Comparison

Both architectures were implemented in the hardware description language VHDL and subsequently synthesized with Synopsys Design Compiler. The results as given by the Design Compiler area and power reports are compared in table 8.1. The two ALUs were synthesized with the compile_ultra command using the constraints given in listing 8.1.

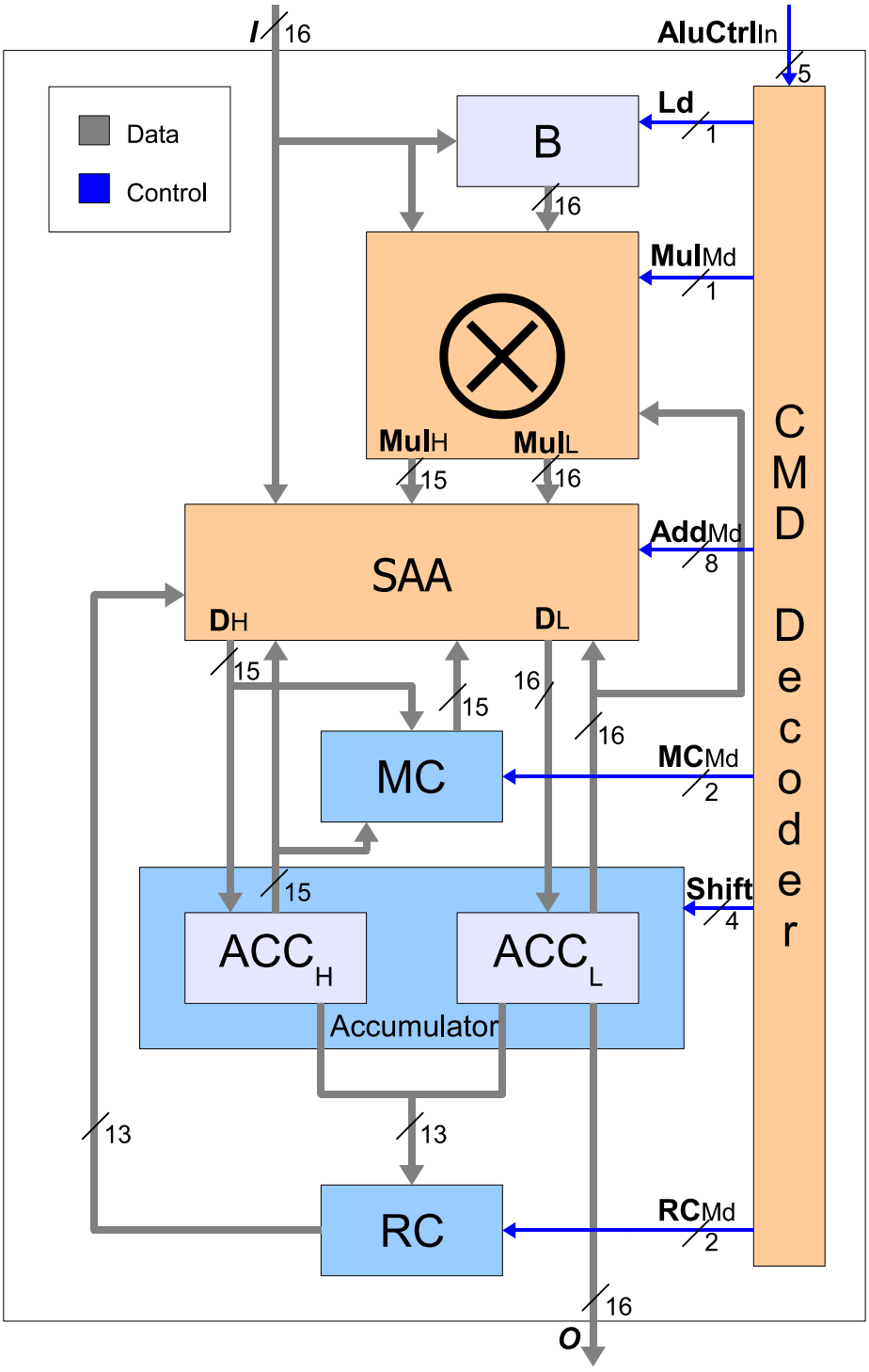



**Figure 8.6:** Complex-*ISA* - *ALU implementation*

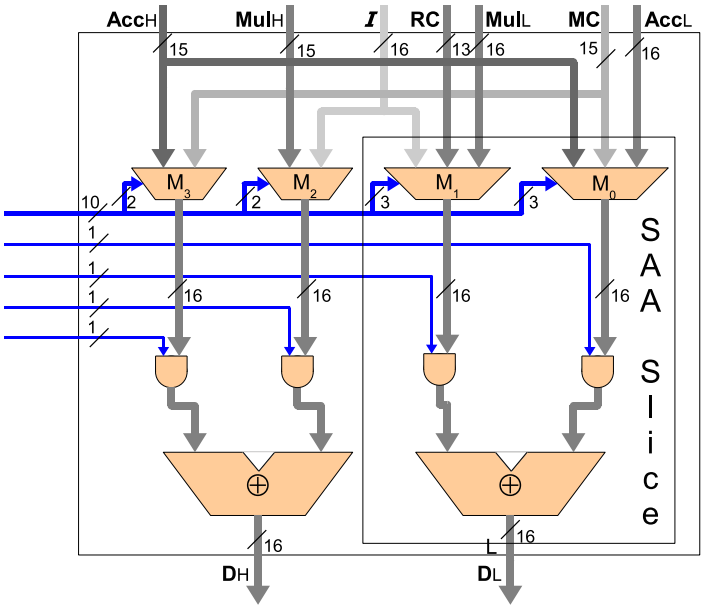



**Figure 8.7:** Complex-*ALU select-and-add (SAA) unit*

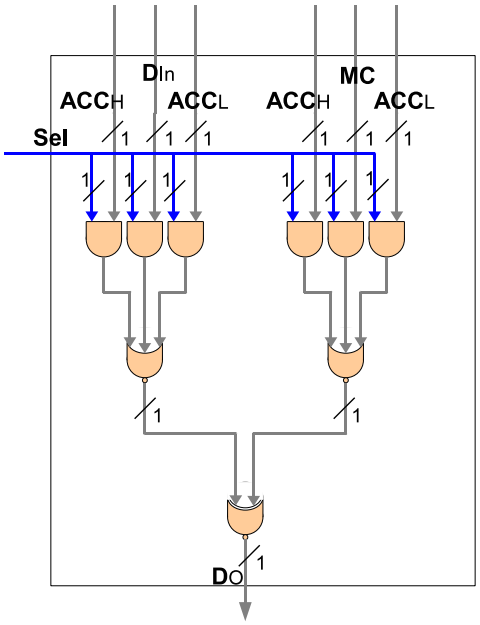



**Figure 8.8:** Complex-*ALU SAA slice*

| | *Simplex*-ALU | *Complex*-ALU | [%] |
|---|---|---|---|
| ALU cell area [$\mu m^2$] | 13,351 | 19,341 | 145 |
| Memory cell area [$\mu m^2$] | 103,767 | 91,347 | 88 |
| Mem+ALU cell area [$\mu m^2$] | 117,118 | 109,878 | 94 |
| Power at 10 MHz [$\mu W$] | 161 | 176 | 109 |
| Scalar multiplication [$cycles$] | 325,245 | 306,587 | 94 |

**Table 8.1:** *Comparison of the* Simplex *and* Complex *ALUs*

The *Simplex* ALU requires significantly less area and marginally less power, while at the same time being approximately 6% slower than its *Complex* counterpart. The *Simplex* ALU necessitates one more 163-bit memory element. Taking this into account, the *Complex* ALU and its $7 \times 163$ bits memory are significantly smaller, than the *Simplex* ALU and the corresponding $8 \times 163$ bits memory.

The *Complex*-ALU has one additional advantages that is not apparent in the comparison made in table 8.1. It solves the $C_H$-selection problem. The $C_H$-selection problem stems from the repeated multiplication modular reduction algorithm introduced in §7.6.3, which is the best solution for the modular reduction operation in conjunction with the *Simplex*-ALU.

This algorithm requires continuous read and write access to the 325-bit result of a multiplication or squaring operation $C = c_{324}z^{324} + \ldots + c_0$. It is necessary to read from and write to $C$ in 16-bit words $C[t]$, where $t \in [0, 21]$. At the same time it necessitates access to $C_H = c_{324}z^{324} + \ldots + c_{163}z^{163}$ and $C_L = c_{162}z^{162} + \ldots + c_0$. The word indices $t$ for $C_H[t]$ and $C_L[t]$ are zero to 11. The memory of this circuit is composed of 163-bit elements. This structure complicates a contiguous read and write access to a virtual 325-bit element.

Different solutions for this particular challenge are conceivable, but none of them are especially attractive. They either disrupt the regular structure of the memory by adding a second read or write bus to the designated multiplication- and squaring-result memory elements, or they require additional registers and multiplexers in the datapath of the ALU. Another option would be to chose a congruent number representation as was done by Großschädl et al. in [GK03b], but that would require 176-bit memory elements.

No concrete numbers for comparison exists but it can be safely assumed that the structure applied to solve this problem would further diminish the power consumption advantage of the *Simplex*-ALU, while, at the same time, increasing the area benefit of a *Complex*-ALU based ECC processor.

For the reasons outline above the *Complex*-ALU was selected for implementation.

## 8.5 Memory

The memory consists of a $7 \times 163$ bits storage unit, a Lookup Table (LUT) implemented as a combinatoric function and a BIST. The LUT is negligible in both area usage and power consumption ($> 1\%$ of the memory component). The BIST unit will be explained in section §8.7.1.

The storage unit is composed of flip-flops (figure 8.9) grouped into 16-bit

**Listing 8.1:** *Constraints*

```
#############################################################
# Setting constraints for ALU comparison
#############################################################

# Set constraints
set_operating_conditions TYPICAL
create_clock ClkxCI -period 100

set_input_delay 1 -clock ClkxCI\
    [remove_from_collection [all_inputs] ClkxCI]
set_output_delay 1 -clock ClkxCI [all_outputs]
set_driving_cell -library umcl18g212t3_tc_180V_25C\
    -lib_cell HDDFERPQ1\
    [remove_from_collection [all_inputs] ClkxCI]\
    > reports/alu.rpt

# all Outputs have the equivalent load of 2 medium
# strength buffers
set_load [expr 2 *\
    [load_of umcl18g212t3_tc_180V_25C/HDBUFD2/A]]\
    [all_outputs]

# Optimize for minimal area
set_max_area 0

# Checks and reports
check_design    >  reports/alu.rpt
report_design   >> reports/alu.rpt
```

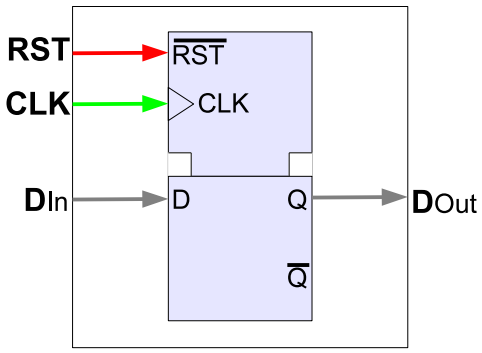



**Figure 8.9:** *Memory flip-flop $w_x$*

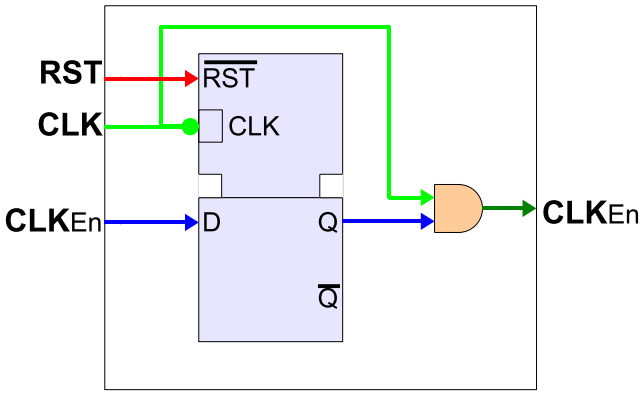



**Figure 8.10:** *Memory clock gating logic*

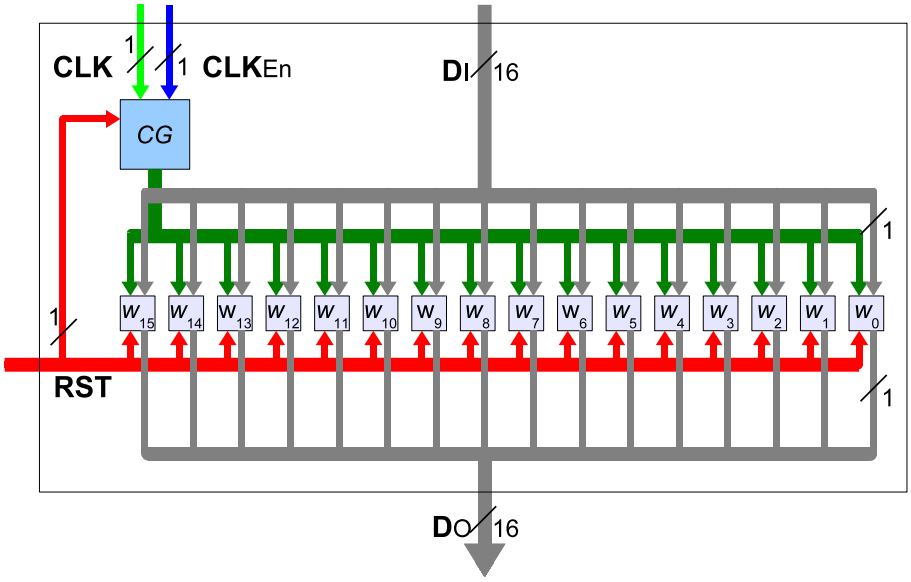



**Figure 8.11:** *Memory word*

words as depicted in figure 8.11 using the clock gate latches illustrated in figure 8.10. The flip-flop block allows concurrent read and write operations.

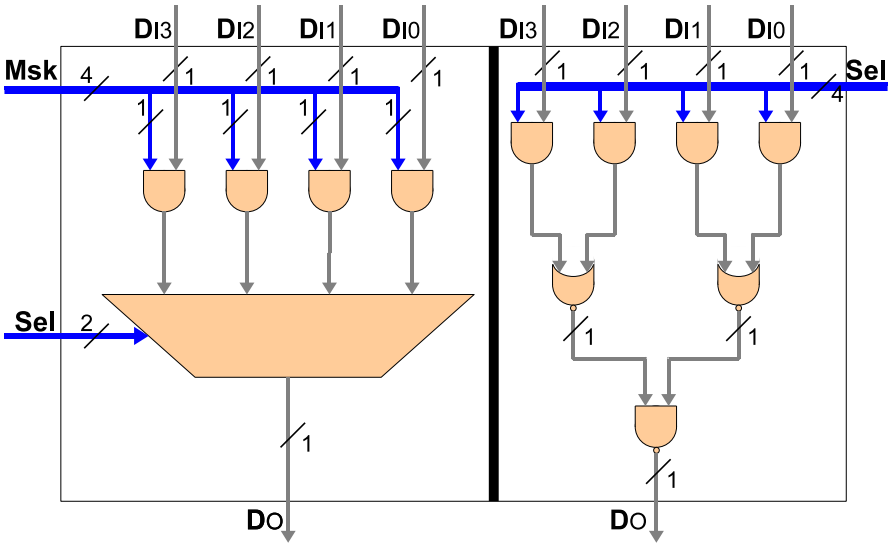



**Figure 8.12:** *Operand isolated multiplexer*

Words are grouped into a structure called a memory unit (figure 8.13). Memory units differ in the number of words and the bit-width of the contained words. The four different configurations of a memory unit are described in table 8.2.

| Type | # Words | # Bits |
|---|---|---|
| A | 4 | 16 |
| B | 3 | 16 |
| C | 4 | 4 |
| D | 3 | 4 |

**Table 8.2:** *Definition of storage unit types*

Each memory unit uses operand isolated 4-to-1 multiplexers as depicted in figure 8.12, to select one word. The outputs of the operand isolated multiplexers are guaranteed to be zero, if the operand isolation is active. This is controlled by the selection signal **Sel**. This in turn is generated from the read address using a decoder with an enable input, which was omitted from the illustration. If the decoder enable input is zero, then so are all of its outputs. Thus, the operand multiplexer is isolated from the inputs.

Apart from minimizing glitch propagation, this also has the advantage that the output signals $\mathbf{D}_O$ of the memory units can be ORed together. This is due to the fact, that always only one memory unit will be active, while all others output zero. Figure 8.14 depicts the memory core component. This implements the $7 \times 163$ bits storage unit. The memory core uses OR gates to generate memory read output.

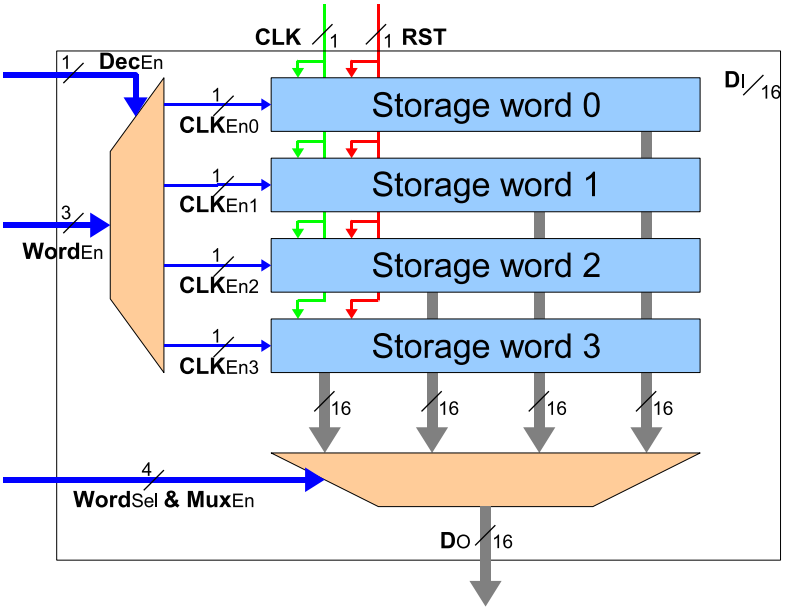



**Figure 8.13:** *Memory unit*

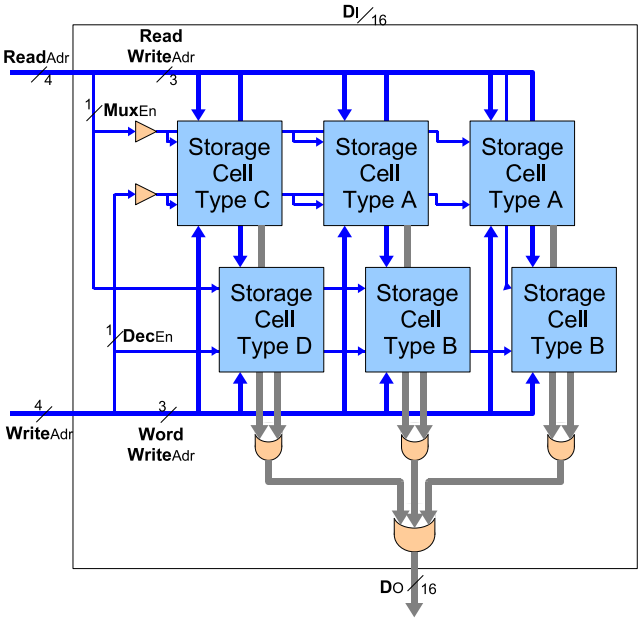



**Figure 8.14:** *Memory core*

Writing to a memory word uses decoders with an enable input as described above, to activate the clock gate of the target memory word. The control signals for the decoders are generated from the write address in a way that always only one word will be selected.

Due to timing constraints on this master thesis, this architecture was only compared to a single competitor. The competing architecture was a trivial VHDL behavioral model memory description. Both storage units were synthesized using the same constraints already applied for ALU comparison, using the compile_ultra command.

The results reported by the Synopsys Design Compiler were a cell area of 84303 $\mu m^2$ and an estimated power consumption of 4.99 $mW$ for the trivial approach. The architecture outlined in this section requires an area of 103876 $\mu m^2$ and has an estimated power consumption of 0.71 $mW$. No clock gating was applied for this comparison.

## 8.6 Control

An ECC point multiplication is a tedious and time consuming task. It requires performing mathematical operations on the EC and in the underlying finite field.

The multitude of functions necessary to perform such an operation require some form of control logic that enables them on the datapath, which consists of the *Complex*-ALU and the memory structure introduced above. As was decided in §6.4.3 all control tasks are realized as a hierarchy of FSMs.

All FSMs are either direct one-to-one implementations of the algorithms given in chapter §7, or so simple in nature that they are best described by their VHDL source code. Therefore, no additional description of the implementation of the FSMs is given herein. This section concentrates on the relationship of the FSMs to each other.

### 8.6.1 The basic operations control unit

The operations in the binary field $\mathbb{F}_{2^{163}}$ form the lowermost layer in the hierarchy of operations that is essential to perform the scalar multiplication. These arithmetic functions include the addition, the multiplication, a squaring operation as a special case of the multiplication and the inversion.

The addition, multiplication, squaring and copying functions necessitate direct control of the *Complex*-ALU, whereas the inversion only requires multiplication, squaring and copying. Therefore, the first four were grouped into the single hierarchical level depicted by figure 8.15.

Of the four FSMs on this level, always only one is active at the same time. For this reason, it is possible to share certain resources amongst the four distinct control units. All of them need a way to implement at least one FOR loop. For this a counter and a subtractor component are required by the hardware realization.

The counter module actually contains two autonomous 4-bit incrementing counters, which allows to realize the two nested loops. These are a prerequisite to the square and multiplication algorithms.

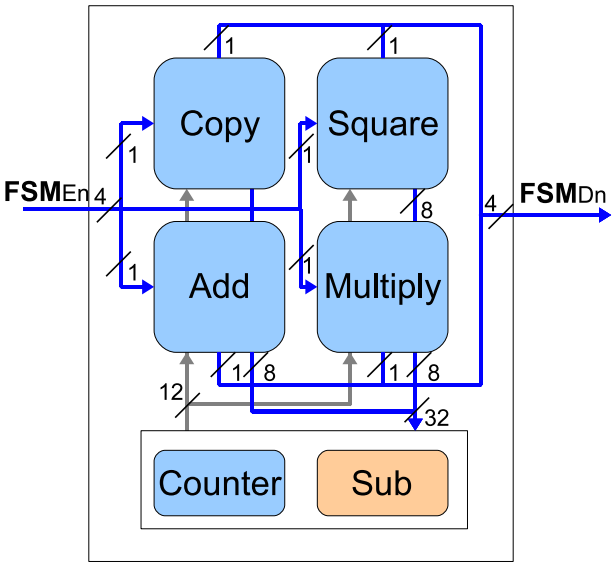



**Figure 8.15:** *The binary extension field operations control module*

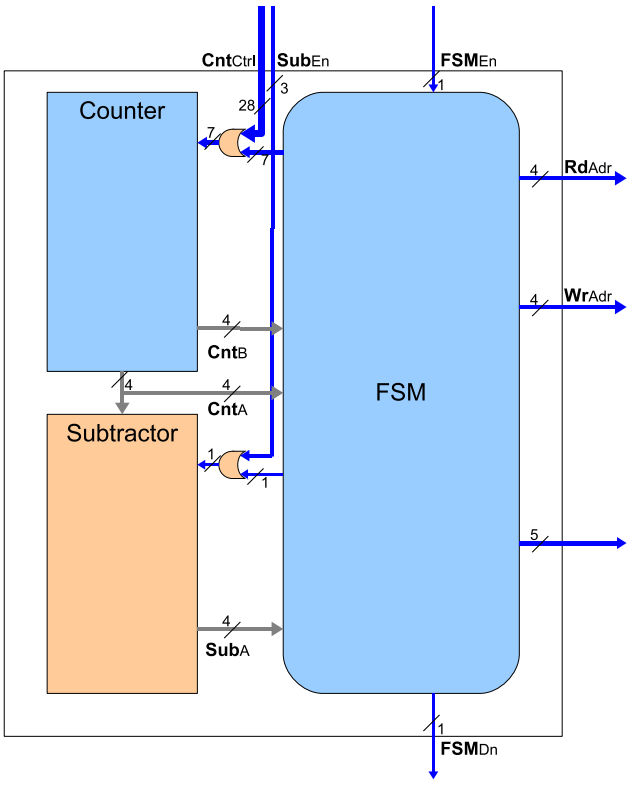



**Figure 8.16:** *Interaction between one FSM and the shared components*

The four bit subtractor is required to help computing the loop indices for the multiplication and squaring operations. It is specified by a table, because not all possible subtrahends actually occur. Because of the self contained nature of each of the four basic functions and their mutual exclusive execution, these two components can be shared.

Figure 8.16 illustrates the relationship between one of the four basic FSMs and the shared components. The FSM is activated using an enable signal, which has to be kept at a logic one, until the FSM signals that it is finished through the done signal. The control signals from the other three FSMs can be ORed with the control signals of the depicted FSM because the control outputs of a deactivated FSM are always zero.

| Function | Algorithm |
|---:|---|
| Copy | N/A |
| Square | Algorithm 7.10 |
| Multiplication | Algorithm 7.9 |
| Add | Modified algorithm 7.2 |

**Table 8.3:** *Mapping of operations in* $\mathbb{F}_{2^{163}}$ *to the chosen algorithms*

Table 8.3 maps the control blocks to the algorithms they realize. The algorithm employed to perform the addition is a direct adaption of the one given for the *Simplex* ISA to the *Complex* ISA. The procedure to copy a memory element is simple enough, therefore it is not explicitly printed herein.

### 8.6.2 The ECC operation control unit

The other tasks realized by the ECC processor control unit are outlined in table 8.4 and their relationship to each other is illustrated by figure 8.17

| Function | Algorithm |
|---:|---|
| Load | N/A |
| Read | N/A |
| Inversion | Algorithm 7.14 |
| Point addition | Algorithm 4.5 |
| Point doubling | Algorithm 4.6 |
| Reset | A portion of algorithm 4.4 |
| Point multiplication | The remainder of Algorithm 4.4 |
| ECDSA signature generation | A portion of algorithm 4.1 |

**Table 8.4:** *Mapping of high level components to the chosen algorithms*

The load, read, reset, point multiplication and the *pseudo* ECDSA signature generation functions can be invoked by the external command interface of the ECC processor. The interface to the other hardware components is fully registered and employs a two-phase full handshaking protocol. It is depicted in figure 6.2.

The load and read commands serve to transfer data to and from the memory core of the ECC processor. As the hardware interface is designed to accommodate an 8-bit bus while the internal datapath is 16 bits wide, the load and read

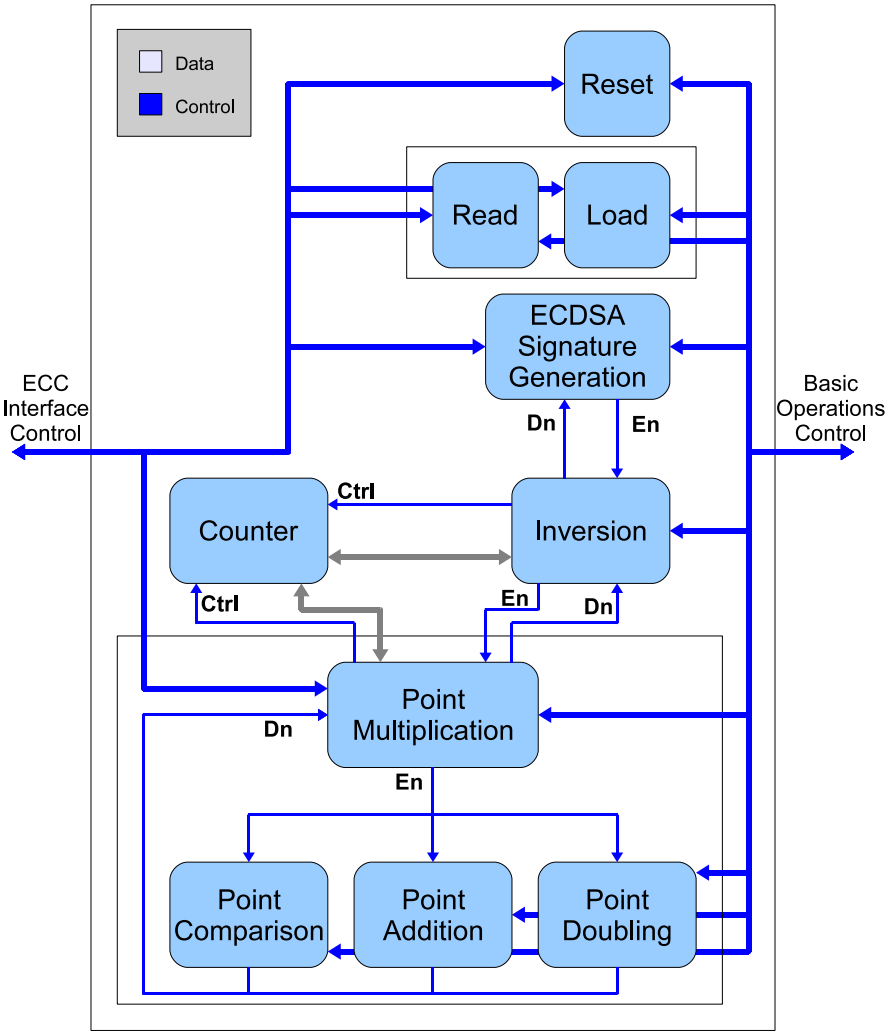



**Figure 8.17:** *The high level operations control module*

operations convert the values as required. Both are reasonable simple, but they necessitate a shift right by eight bits operation, which must be supported by the ALU.

### ECDSA signature generation

The remaining three commands together allow to perform a partially standard compliant ECDSA signature generation. Full standard compliance is not possible, because it would require a Pseudo Random Number Generator (PRNG) and a hash function. Both pseudo random number generation and hash functions are extensively researched topics. For this reason, they were exluded from this project. The partial ECDSA signature generation will also be called *pseudo* ECDSA throughout the remainder of this document.

The reset command realizes line 3 of algorithm 4.4, which initializes the memory with start data for the scalar multiplication. It does so, by copying some values from the LUT-ROM to the memory. Other start values are generated directly from data in the ROM. In addition, a default value for the ephemeral key $k$ is copied into the flip-flop block. Under normal circumstances, it would have to be generated by a random number generator.

At this point an interspersed load command could be utilized to write an arbitrary $k$ to the memory, before the point multiplication is executed. This is cryptographically dangerous, but it allows better evaluation of the circuit.

The next step in the pseudo ECDSA signature generation is the scalar multiplication $k \cdot P$. This is the only ECC operation required by the algorithm. The point multiplication depends on the point comparison, addition, doubling and inversion functions.

After the scalar multiplication, a challenge value must be written to the memory core of the ECC processor. In a fully standard compliant ECDSA signature generation, the message would have to consist of several elements. Each part of it would have to be processed by a hash function. As this is not possible, the following operations are executed on a single element sized message $m$.

To finalize the signature creation process, a sequence of prime field operations is performed: $s = k^{-1}(m + dr) \, mod \, n$, $r = x_1$. As no prime field arithmetic unit is available, the function is executed using the binary extension field arithmetic datapath. Needless to say, that the signature generated by this pseudo ECDSA implementation is not cryptographically secure.

Recall that the primary task of this thesis was to find an efficient implementation of the scalar multiplication. The signature generation was added to illustrate a possible application. It provides a framework, which could be augmented with the omitted functional units. This would complete the framework and allow full standard compliance.

### Component reuse

It is again possible to share a component amongst the different FSMs on this level. In this case an 8-bit counter is reusable by several of the control units, including the point multiplication and the inversion. The counter can function as one coherent 8-bit counter or two independent 4-bit units. The 4-bit units are utilized by the load and read commands, whereas the 8-bit mode is employed

by the point multiplication to index the key bit and the inversion to count the number of square operations performed.

## 8.7 Design For Test

The Design For Test (DFT) strategy of the *ECCon* ASIC consists of an automatically inserted scan chain and a Built-In Self-Test (BIST). For details on scan chains in general and scannable registers please confer to [Fel07].

The scan chain covers all parts of the circuit except the memory and the reset synchronization flip-flop. The memory is not included because it would require scannable flip-flops. These have a higher area usage and power consumption.

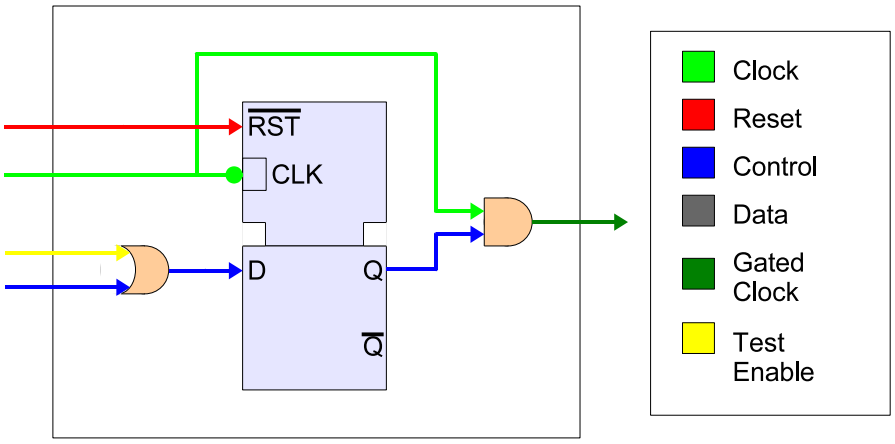



**Figure 8.18:** *Observable clock gate*

A second problem arises in conjunction with clock gating. A clock gated register is not observable. In order for the insertion of a flip-flop into a scan chain to be beneficial, the clock gate must be disabled. This is handled automatically by the synthesis tool, by inserting an additional OR gate (figure 8.18). Sadly, this puts an additional strain on the area usage and power consumption. For these two reasons including the memory into the scan chain is not an acceptable solution.

Utilizing an Automated Test Pattern Generator (ATPG) tool the scan chain was evaluated. The results are given in table 8.5. The low test coverage of only 61.06% is primarily due to the omission of the storage unit from the scan chain. Another reason is that the reset synchronization register is not controllable. A negligible fraction of the undetectable faults is caused by the fact that a minuscule set of logic component inputs and outputs is tied to a certain logic value. Under normal circumstances, these should be eliminated by the boundary optimization of the hardware synthesizer tool.

### 8.7.1 The Built-In Self-Test (BIST)

The memory core is equipped with a BIST unit. This component implements the Modified Algorithmic Test Sequence (MATS++) algorithm, which according to

| #Faults | #Detect | #Posdet | #Undet | #Redund | Testcov | Instance name |
|---|---|---|---|---|---|---|
| 48,980 | 29,440 | 14 | 18,775 | 751 | 61.06% | Chip |
| 48,852 | 29,323 | 14 | 18,764 | 751 | 60.98% | *ECCon* |
| 3,358 | 3,287 | 0 | 69 | 2 | 97.94% | RART |
| 4,032 | 3,886 | 0 | 41 | 105 | 98.96% | RFID control unit |
| 41,342 | 22,032 | 14 | 18,652 | 644 | 54.15% | ECC core |
| 8,136 | 8,032 | 5 | 93 | 6 | 98.82% | ALU |
| 23,400 | 4,725 | 5 | 18,055 | 615 | 20.75% | Memory |

**Table 8.5:** *Fault coverage*

Riedl et al. [RRR95] is guaranteed to detect all unlinked stuck-at and transition faults in the memory.

This remedies the deficiencies created by not adding the RAM to the scan chain. The better part of the 18055 undetectable faults in the memory component should be covered by the BIST. A conservative estimation is that 18000 of the 18055 faults are detectable. This would lead to a total test coverage of 96.86%, without considering the other two fault classes detailed above. Including the undetectable faults caused by the reset, the fault coverage increases to 97.33%.

## 8.8 Side-channel attack resilience

Side channel attacks do not attack the cryptographic primitive but its implementation. A side channel is an additional channel of information, which unintentionally leaks information about the secret key of a cryptosystem. Examples for points of attack are the time an implementation requires to perform a cryptographic operation, its power consumption during the computation or even the error messages it might generate in case of a problem.

Three of the more prominent side channel attacks are the timing attack, the Simple Power Analysis (SPA) and the Differential Power Analysis (DPA). For a comprehensive introduction to side-channel attacks confer to [Wol04]. For detailed information on power consumption based side channel attacks [MOP07] can be recommended.

**The timing analysis attack** tries to derive information about the secret key from the runtime of a cryptographic algorithm. It is applicable if the execution time dependents upon the key.

This is very often the case in software implementations, which tend to contain conditional statements, which execute a set of additional operations if a single bit key has a certain value. Expressed colloquially if the code contains an if statement without an else block that requires the exact same amount of clock cycles to complete, than a timing attack is possible.

**The SPA** measures the power consumption of a circuit and attempts to estimate the secret key of the cryptosystem by analyzing the so called power trace. It distinguishes itself from the DPA because it requires only very few power traces for a successful attack, even a single one might be sufficient.

The vulnerabilities are similar to those outlined for the timing attack. The problem stems again from key dependent operations. If the value of a key bit decides if an assumed datapath performs an AND function in contrast to an XOR one, the difference in the power trace might be distinguishable. That could suffice for an attack.

**The DPA** is similar in nature to the SPA, but is capable of distinguishing minuscule differences in the power consumption profile of a circuit. It utilizes statistical methods to evaluate a set of power traces over the same cryptographic operation computed with the same secret key. This technique often requires 10.000-100.000 power traces to successfully attack an implementation.

Point of attacks for DPA are very difficult to spot in the design phase, because diminutive differences in the power profile, which are not apparent to the designer, are sufficient. The best defense against DPA is to rely on a protocol that changes the secret key once in a while.

The circuit designed in this master thesis should be resilient against all three types of attack methodologies. All operations of the Montgomery point ladder scalar multiplication algorithm introduced in §4.4.3 are key independent. The same functions are performed in every step of the algorithm only the input variables change. Thus, the runtime is the same for every ephemeral key $k$, which thwarts timing based attacks. It is SPA immune for very much the same reasons.

The DPA resilience of the circuit stems from the fact that the ECDSA signature generation algorithm was selected as cryptographic primitive for an authentication protocol. This algorithm requires a random ephemeral key for every execution run. In fact the ECDSA standard requires a different $k$ for every signature creation with the same private key $d$, otherwise it is possible to attack the cryptographic principle directly. A DPA would require thousands of power traces with the same $k$ for a successful attack.

## 8.9 Synthesis

The circuit was synthesized using the Synopsys Design Compiler. The constraints and syntheses scripts were incrementally refined to find the optimal solution for the *ECCon* circuit.

The following optimization steps were performed:

**Step 1** This is the starting point for further optimizations and the baseline for comparison. The design is constrained for typical operating conditions, a 10 MHz clock and the inputs and outputs are set to the values of the respective I/O pads.

**Step 2** The circuit is synthesized with different area and power constraints until the optimal $A \times P$ solution is found. The compile command is used to perform the synthesis.

**Step 3** The synthesis command is changed to compile_ultra. The area constraint loses its effect, as long as a low-power constraint is in place.

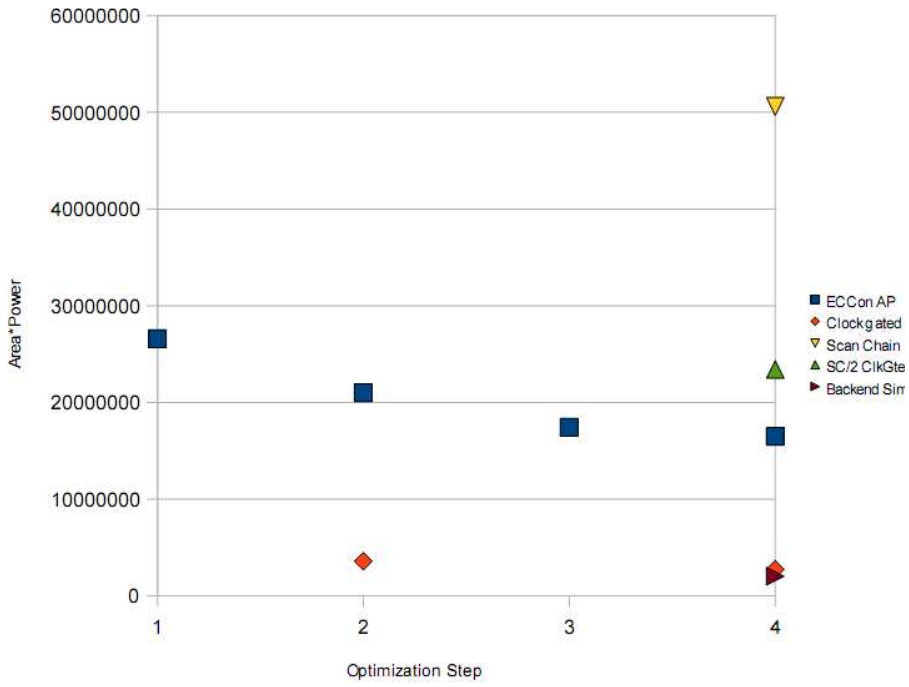



**Figure 8.19:** *$A \times P$ diagram over different optimization steps*

**Step 4** The hierarchy is selectively ungrouped until the optimal structure is found.

**Step 5** Scan chain insertion with default clock gate injection.

**Step 6** Two different clock gate types are injected into the circuit. The memory requires a simpler clock gate logic style than the rest of the circuit.

Figure 8.19 plots the results of the six synthesis steps. The "*ECCon* AP" curve represents the $A \times P$ product over the optimization steps one to four. The "Clock gated" curve depicts the results of optimization step two and four with clock gate injection. The "Scan Chain" point shows the $A \times P$ product after scan chain insertion. Because of the control-point-OR-gates in the clock gates, the Synopsys power report looses its applicability. From this point on, SAIF back-annotation is required to gain a valid power report. The "SC/2 ClkGte" point represents the result of step six. The use of clock gates without a control point for the memory virtually reduces the $A \times P$ product. Finally, the "Backend Simulation" point shows the result of a power simulation of the circuit.

## 8.10 Results

The circuit outlined in this master thesis was developed as a Very high speed integrated circuit Hardware Description Language (VHDL) model, which in

turn was synthesized and then laid out into a fabrication ready netlist. This was then analyzed with respect to area usage, power consumption and timing, the results of which are detailed in this section.

### 8.10.1 Area

The required die area including pads, but not the bonding lands and the seal ring is $2,227\mu m \times 540\mu m = 1,202,580\mu m^2$. The core area of the *ECCon* processor is given by $1,808.96\mu m \times 121.56\mu m = 219,897\mu m^2$. This is the area for a core utilization of 70%. The low utilization was chosen to compensate for the high parasitics as a consequence of the suboptimal routing due to the drawn out rectangular shape.

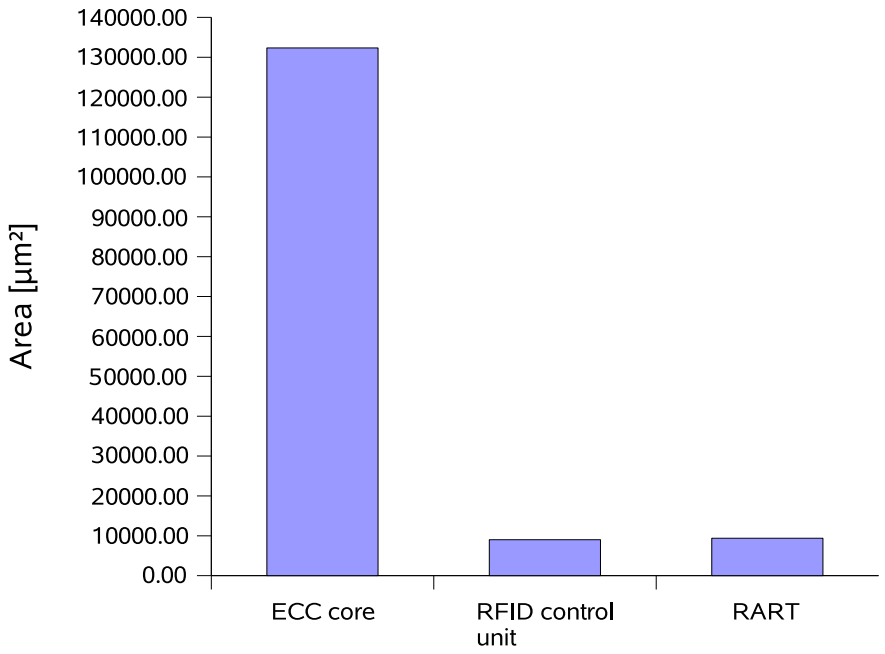



**Figure 8.20:** *Area comparison of the ECCon components*

It is important to note that the following area comparisons employ the estimations made by the synthesis tool. They are astoundingly astute. The synthesis report predicted an area usage of $151,125.7\mu m^2 \approx 219,897 \cdot 0.7 = 153,928\mu m^2$.

Figure 8.20 presents the distribution of the area with respect to the three primary components of the *ECCon* ASIC. It is easily discernible that the lion's share of the area is taken up by the ECC processor. The area usage of the two modules that form the RFID front-end is negligible in comparison.

A further analysis of the ECC processor breaks down the area requirements of its subcomponents in figure 8.21. The values for the ALU and the memory are taken from the synthesis report, whereas the control logic is an estimation based on the total area of the ECC processor and the two known subcomponents.

| | Area [$\mu m^2$] | Area [%] | Area [$GE$] |
|---|---|---|---|
| Total | 151,125.70 | 100.00 | 15,628.30 |
| ECC processor | 132,333.59 | 87.6 | 13,684.96 |
| RFID control unit | 9,028.41 | 6.0 | 933.65 |
| RART | 9,402.61 | 6.2 | 972.35 |

**Table 8.6:** *ECConarea distribution*

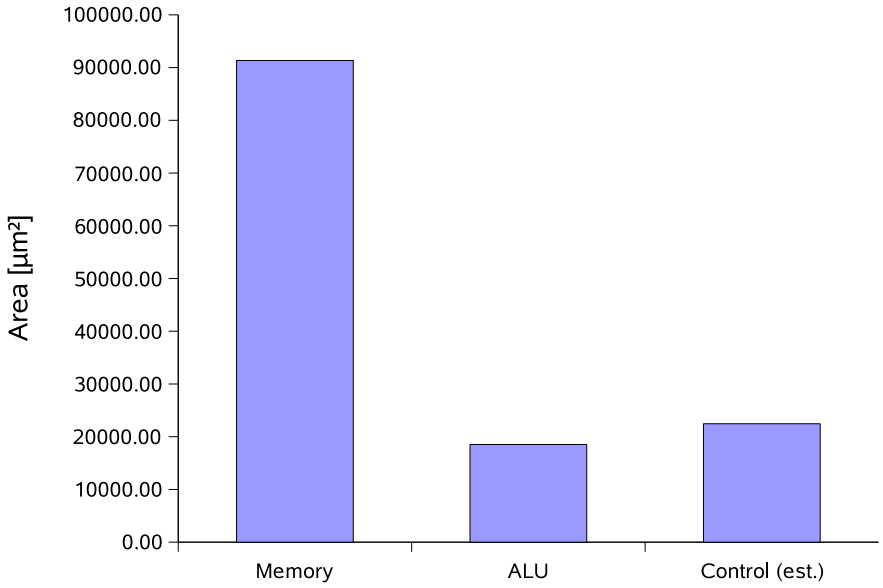



**Figure 8.21:** *Area comparison of the ECC processor components*

| | Area [$\mu m^2$] | Area [%] | Area [$GE$] |
|---|---|---|---|
| ECC processor (total) | 132,333.59 | 100.00 | 13,684.96 |
| Memory | 91,346.45 | 69.03 | 9,446.37 |
| ALU | 18,531.01 | 14.00 | 1,916.34 |
| Control | 22,456.14 | 16.97 | 2,322.25 |

**Table 8.7:** *ECC processor area distribution*

The control unit of the ECC processor being larger than the ALU, is a surprising result, that must be put down to the complexity and sheer number of different operations required for the scalar point multiplication.

### 8.10.2 Power

Next to area, the power consumption is the second most important factor of the *ECCon* circuit. The power analysis was performed with the synthesis tool using parasitics and switching activity back-annotation.

The *ECCon* processor supports two different modes of operation. The first is the RFID mode, where the ASIC functions as the digital part of an ISO-18000-3-1 compatible RFID tag. In this mode of operation the *ECCon* processor requires a clock frequency of 6.78 MHz and all components are active.

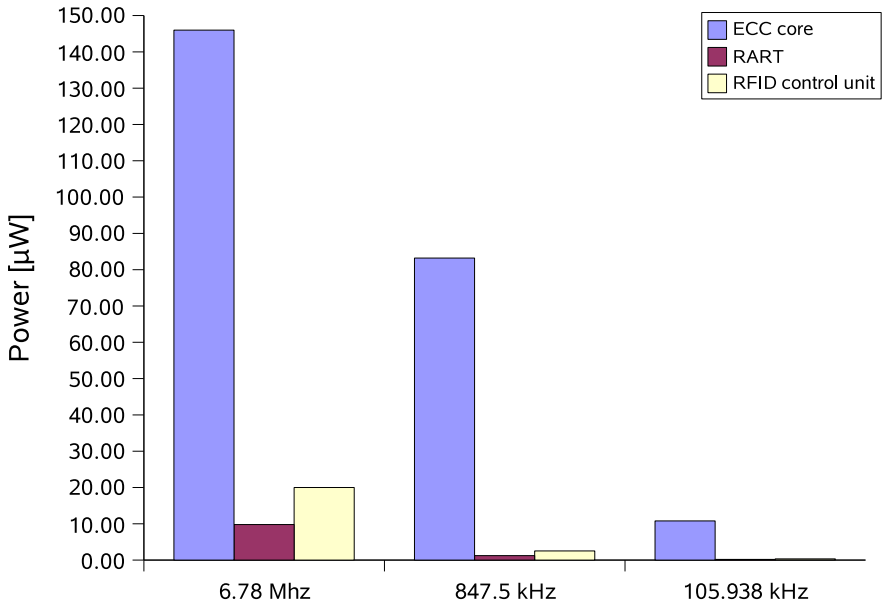



**Figure 8.22:** *Power consumption comparison of the ECCon components*

In the second mode, only the ECC processor is active, while the RFID front-end is deactivated using clock gating. The clock frequency for the ECC processor can be chosen freely within the operation limits of the ASIC. Two frequencies 847.5 kHz and 105.938 kHz were selected as representatives for a power simulation. The first is the nominal clock frequency of the ECC processor, when in RFID mode. The second was selected because it is qualified for comparison with different implementations of ECC primitives.

Note the difference between the power consumption of the ECC processor in RFID mode (6.78 MHz) and in ECC mode (847.5 kHz). In both cases the ECC unit is operated at the same clock frequency of 847.5 kHz, only in the first case this is a virtual clock frequency created using clock gating. This implies a power wastage of 89 $\mu W$ in the clock tree. This is a strong argument for using

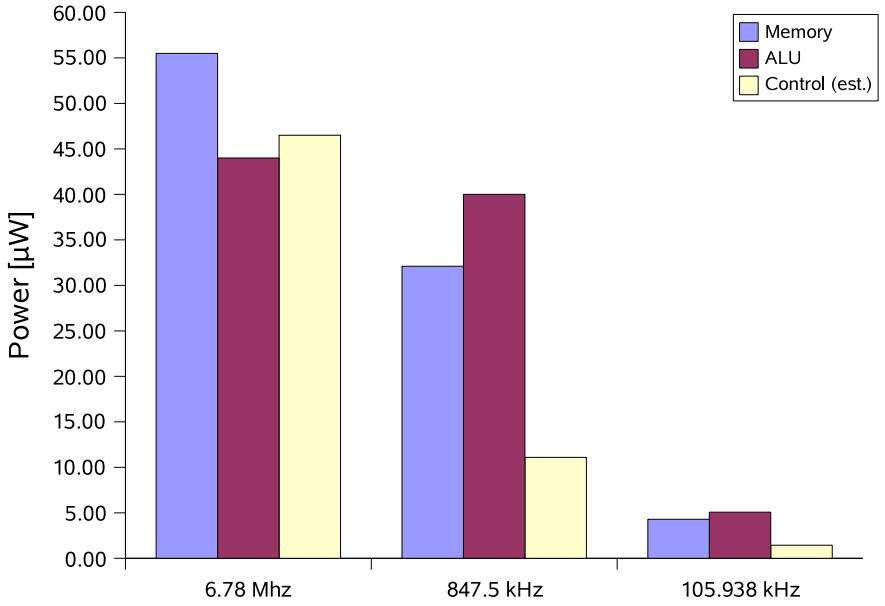



**Figure 8.23:** *Power consumption comparison of the ECC components*

two different clock domains in a real life application.

Figure 8.22 and figure 8.23 illustrate the power distribution amongst the *ECCon* and the ECC processor components respectively. The exact values are detailed in table 8.8 and table 8.9.

| | Power consumption [$\mu W$] | | |
|---|---|---|---|
| | 6.78 MHz | 847.5 kHz | 105.94 kHz |
| *ECCon* (total) | 176.00 | 87.00 | 11.40 |
| ECC processor | 146.00 | 83.20 | 10.80 |
| RFID control unit | 20.00 | 2.53 | 0.35 |
| RART | 9.79 | 1.24 | 0.19 |

**Table 8.8:** *ECCon ASIC power distribution over different frequencies*

The power dissipation values for the memory and the ALU proof that the 16-bit datapath was an adequate selection.

Note that the synthesis tool predicted a power consumption of approximately 100 $\mu W$ for the complete *ECCon* processor at a clock frequency of 6.78 MHz, before the scan-chain insertion. It has to be assumed that the difference between the synthesis estimation and the result of the layout program is either due to the scan chain, which is highly unlikely or that additional routing capacities caused by the unusually rectangular shape of the processor are responsible.

| | Power consumption [$\mu W$] | | |
|---|---|---|---|
| | 6.78 MHz | 847.5 kHz | 105.94 kHz |
| ECC processor (total) | 146.0 | 83.2 | 10.80 |
| Memory | 55.5 | 32.1 | 4.29 |
| ALU | 44.0 | 40.0 | 5.07 |
| Control (est.) | 46.5 | 11.1 | 1.44 |

**Table 8.9:** *ECC processor power distribution over different frequencies*

### 8.10.3 Comparison with related work

Results for ECC implementations are especially difficult to compare. The problem arises in part because of the multitude of ECC realization options. Field and cryptographic primitive selection are the keywords here. A second class of difficulties is due to the level of implementation. Sometimes only the point multiplication is realized, in other cases the results of a complete cryptographic protocol implementation are published.

The next challenge derives from the fact that when it comes to comparing power consumption results, often only the required energy is published. As was already deliberated, this is interesting for battery driven environments, but for an RFID application it is of considerable less interest. Even if the power consumption is given, it is of course dependent on the clock frequency and it is hard to relate results for different VLSI technologies.

The same is true when comparing the area usage. To alleviate that, the area usage is converted from $m^2$ into Gate Equivalents (GEs). A GE is the size of a NAND-2 gate in the target technology. To compute the number of GEs the total area is divided by the area of such a gate. For the technology utilized to implement the *ECCon* circuit (umcL180) a 2-input NAND gate of lowest drive strength requires $9.67\mu m^2$.

| | Area [$GE$] | Cycles [$kCycles$] | Power @ 106 kHz [$\mu W$] | Avg. Cur. [$\mu A$] | ECC-Curve | VLSI tech. |
|---|---|---|---|---|---|---|
| *ECCon* | 13,685 | 306 | 10.80 | 6.00 | B-163 | UMC L180 |
| [FW07b] | 23,656 | 500 | 141.01 | 42.73 | P-192 | AMS C35 |
| [KP06] | 15,094 | 430 | ?.? | ?.? | B-163 | AMI C35 |
| [FW07a] | 23,600 | 502 | 62.21 | 18.85 | P-192 | AMS C35 |
| AES [FW07a] | 3,400 | 1 | 9.9 | 3.0 | 128 | AMS C35 |

**Table 8.10:** *Comparison with other implementations*

Table 8.10 tries to compare the *ECCon* processor introduced herein with other recent implementations of area and power constrained ECC scalar multiplication units. It relates the area usage, the runtime in cycles and the average current at a frequency of 106 kHz. The candidates either implement the point multiplication over a binary extension field of order 163 (B-163) or a prime field of order 192 (P-192). The abbreviation AMS C35 represents the 0.35 $\mu m$ process of Austria Microsystems. AMI C35 is the AMI Semiconductor 0.35 $\mu m$ CMOS technology and UMC L180 stands for the UMC logic 0.18 $\mu m$ process. All results presented in table 8.10 are based on simulations.

The processor discussed in [FW07b] is the closest match with respect to the

chosen application. It implements all ECDSA operations except hashing and pseudo random number generation and works with an EC defined over a 192-bit prime field. The component introduced by Kumar and Paar in [KP06] is a point multiplier for variable field orders specifically for RFID tags. Quite inexplicably, no power or energy consumption results are given. Finally Feldhofer et al. compare different cryptographic primitives in [FW07a] including a low power, small area ECC device by Wolkerstorfer ([Wol05]), which works with a 192-bit prime curve. The AES is a symmetric-key cipher and serves as a baseline for comparison.

## 8.11 Layout

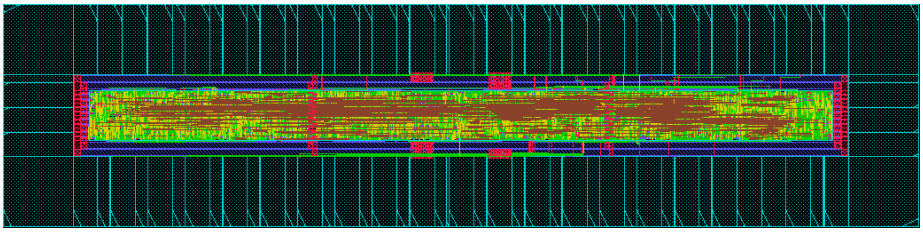


**Figure 8.24:** *The layout of the ECCon processor*

Chapter 9

# Conclusion and outlook

The ambition of this thesis was to develop an ECC hardware module that could serve as foundation for cryptographically sound RFID based product authentication. The projected RFID application determined fierce constraints on the available die area and the power budget.

After a comprehensive discussion of the theoretic background, the fundamental decision to use a small word level architecture to implement the basic operations in the binary extension field $\mathbb{F}_{2^{163}}$ was introduced.

This decisions distinguishes this work from all preceding implementations of ECC for low power, small area applications, which utilize full precision datapaths. Full precision datapaths create great power fluctuations which can cause erroneous data transmissions and require a comparatively higher mean current, but allow for relatively simple algorithms to realize the field operations.

A thorough design-space exploration of word level algorithms and arithmetic units that could implement them was conducted. It yielded a new algorithm for multiplication with interleaved reduction that is both faster and more area efficient than other more obvious approaches.

An ALU was specifically designed to realize the operations required by this procedure. This module also implements all other binary field operations most efficiently. In combination with a highly optimized memory structure the power consumption and area requirements to implement the ECC point multiplication were reduced to hitherto unreachable values.

The ECC processor module was complemented with the digital part of an ISO-18000-3-1 compliant RFID front-end. The communication protocol was extended to allow for cryptographic operations like the scalar multiplication or a limited realization of the ECDSA signature generation procedure. The fully functional RFID front-end is supplemented with an interface that enables direct access to the ECC processor.

The entire architecture was synthesized and a back-end design was performed. The *ECCon* processor was then taped out and fabricated in an 180 $nm$ CMOS process.

That ECC is a viable choice of cryptographic primitive for an RFID application was already ascertained by Wolkerstorfer in 2005 [Wol05]. The primary observation made by this work is that the word-level approach is definitely the

way to go. It requires less area, less power and is substantially faster. A second important reflection is that concentrating all efforts on one specifically selected EC gives an additional degree of freedom to the algorithm design step, which provides an additional benefit.

Other observations include

- Word level algorithms for operations in finite fields were already thoroughly researched for instruction set extensions for smart card processors, but they lack curve specific optimizations and an adapted memory structure.
- In a real world application at least two different clock domains should be implemented. One for the RFID front-end and another one for the ECC processor due to the relatively high power losses in the clock tree.
- Clock gating and operand isolation are indispensable in low power design, but operand isolation is tedious to realize due to synthesizer optimizations.
- Once a scan chain is in place quick synthesis tool based power estimations becomes impossible due to the OR gate overriding the observable clock gating latch.

The logically next step is to perform a design with the same preconditions as this one for prime field arithmetic on a similar fixed elliptic curve. If results are promising a dual field unit might be the next step towards ECC on RFID tags. Unless a simpler ECC based authentication schemes than an ECDSA based one can be found, a fully ECDSA compliant ECC processor would provide further valuable insights.

Appendix A

# Datasheet

## A.1 Key Features

- Low power, small area ECC point multiplication device
- ISO-18000-3-1 compatible digital RFID front-end
- *Technology:* UMC L180 GII 1P/6M 1.8V/3.3V CMOS
- *Core Size:* 219897 $\mu m^2$
- *Clock Frequency:* 46 MHz
- *Total power consumption at 106 kHz:* 11.4 $\mu W$
- *ECC processor power consumption at 106 kHz:* 10.8 $\mu W$
- Built-in memory self-test
- Full scan using one scan chain

## A.2 Circuit Configuration

### A.2.1 Operation Modes

The *ECCon* processor provides two main functionalities. On the one hand, it implements the digital part of an ISO-18000-3-1[ISO04] compatible RFID tag. On the other, it is an Elliptic Curve Cryptography ASIC that implements the point multiplication operation on an Elliptic Curve. These modes of operation are called the RFID mode and the ECC mode.

#### RFID mode

In this mode of operation the *ECCon* processor functions as an ISO-18000-3-1 [ISO04] standard compliant RFID device. The clock input must be set to a frequency of 6.78 MHz. Table A.1 lists the commands supported by this implementation and their respective command codes. For a description of the

| Code [hex] | Name |
|---|---|
| 01 | Inventory |
| 02 | Stay quiet |
| 20 | Read single block |
| 21 | Write single block |
| 25 | Select |
| 26 | Reset to ready |
| E0 | ECC Reset |
| E1 | ECC $k \times P$ |
| E2 | ECC Sign |

**Table A.1:** *RFID command table*

| Code [binary] | Name | Function |
|---|---|---|
| "00010000" | Read element | Read a $7 \times 163$ bits element. |
| "00001000" | Write element | Write a $7 \times 163$ bits element. |
| "00000100" | ECC Reset | Perform the ECC reset operation. |
| "00000010" | ECC $k \times P$ | Perform the point multiplication. |
| "00000001" | ECC Sign | Perform the pseudo ECDSA signature generation. |

**Table A.2:** *ECC interface commands*

commands, refer to §5. For a definition of the expected input and output signals, refer to [ISO04].

### ECC mode

This mode of operation circumvents the RFID front-end and allows direct access to the ECC processor. The ECC processor uses a two-phase full handshake protocol [Kae08]. For an illustration of the command interface, confer to figure 6.2. The clock can be set to any value in the operational range of the *ECCon* processor (106 kHz to 46 MHz).

The interface allows the invocation of the cryptographic commands, and reading and writing data to the memory core of the ECC processor. To perform a command, it is necessary to load it into the command register of the interface. Once a command is loaded, it is automatically executed. The read and write command require an element address, which must be loaded into the address register beforehand.

The *EccIntCtrlxSI* input controls which interface register is loaded.

| Code [binary] | Action |
|---|---|
| "00" | No operation |
| "01" | Load command |
| "10" | Load address |

Table A.2 lists the commands the interface supports. The command code identifier must be applied to the *EccIfcexDI* input port.

The following steps illustrate how to load a command. It is assumed that this is the first communication with the *ECCon* processor and that the handshake request signal was tied to zero, before the reset was released. The acknowledge

signal should output zero after the release of the reset. This signals that the handshake interface is ready for operation.

1. Set *EccIntCtrlxSI*="01" and *EccIfcexDI*="Command code". Toggle the *EccReqxSI*.
2. Wait for the acknowledge signal to follow the value of the request signal. The command is now loaded into the command register.
3. Set *EccIntCtrlxSI*="00", *EccIfcexDI*="00000000" and again toggle the request signal. This will start the execution of the command.
4. Wait for the acknowledge signal to follow the request signal. The command is now executed.

The read and write command work differently. The following steps illustrate how to perform a write command.

1. Set *EccIntCtrlxSI*="10" and *EccIfcexDI*="Element address". Toggle the *EccReqxSI* signal.
2. Wait for *EccAckxSO*. The address is now loaded into the address register. Set *EccIntCtrlxSI*="01", *EccIfcexDI*= "00001000" and again toggle the request signal. This will load the write command.
3. Wait for the acknowledge signal. Apply the lowest order byte of the 163-bit element to the data input port *EccIfcexDO*. Set *EccIntCtrlxSI*="00" and toggle the request value.
4. Wait for the acknowledge signal. The first byte has now been written to the memory. Apply the next byte and toggle the request input. Repeat this 20 times for a total of 21 bytes.

The following steps detail the use of the read command.

1. Set *EccIntCtrlxSI*="10" and *EccIfcexDI*="Element address". Toggle the *EccReqxSI* signal.
2. Wait for *EccAckxSO*. The address is now loaded into the address register. Set *EccIntCtrlxSI*="01", *EccIfcexDI*="00010000" and again toggle the request signal. This will load the read command.
3. Wait for the acknowledge signal and toggle the request signal once more.
4. Wait for *EccAckxSO*. The first byte is now ready to be read from the *EccIfcexDO* output port. Repeat this step 20 times to read the 21 bytes of the 163-bit element from memory.

Four addresses are significant for the pseudo ECDSA signature generation operation. For details on how to perform this function see §8.6.2.

| Element address | Stored value |
|---|---|
| 1 | The ephemeral key $k$. |
| 2 | The result of the scalar multiplication $r$. |
| 3 | The challenge message $m$ to sign. |
| 4 | The second part of the digital signature $s$. |

### A.2.2 Design for Testability (DFT)

#### Scan chain

The *ECCon* processor has a single scan chain to test all registers with the exception of the memory flip-flops. To perform a scan, the *ScanEnxTI* input port must be set to one, before the reset is released. The scan chain uses the same clock as the rest of the circuit.

#### Built-in Memory Self Test (BIST)

To activate the BIST set the *BistEnxTI*=1. This BIST uses the same clock port as the rest of the circuit. *BistEnxTI* must be kept at one for the whole time the BIST is performed (308 cycles). The *BistResxTO* port should switch from a logic zero to a logic one, 78 cycles after the BIST was activated. The BIST was successful if this signal retains a logic one for an additional 230 cycles.

*Note:* Assert these signals *during* active reset, then release the reset signal. The BIST should start and keep running for as long as *BistEnxTI=1*. The test starts anew every 308 cycles.

## A.3 Port Description and Pinout

All ports of the ASIC are listed in table A.3.

| Signal Name | Type | Description |
|---|---|---|
| ClkxCI | in | Clock (single-edge one-phase triggered). |
| RstxRBI | in | Asynchronous reset (active low). |
| Configuration Interface | | |
| ModeSelxSI | in | This port selects the mode of operation for the *ECCon* processor. If this signal is tied to high, the core is in ECC mode. Otherwise the processor is in RFID mode. |
| ECC Data Interface | | |
| EccReqxSI | in | The handshake request signal. |
| EccAckxSO | out | The handshake acknowledge signal. |
| EccIntCtrlxSI | in | The interface control port. |
| EccIfcexDI | in | Byte input data. |
| EccIfcexDO | out | Byte output data. |
| RFID Data Interface | | |
| AirInxDI | in | RFID receiver data. |
| AirOutxDI | out | RFID transmitter data. |
| Status Signals | | |
| SimTrnsOutxTO | out | RFID transmitter control output. |
| DFT Interface (scan test) | | |
| ScanEnxTI | in | This port enables DFT scan and overrides clock gating. |
| ScanInxTI | in | The scan chain input. |
| ScanOutxTO | out | The scan chain output. |
| BIST Interface (RAM self-test) | | |
| BistEnxTI | in | Built-in self test enable. |
| BistResxTO | out | Built-in self test status output. |

**Table A.3:** *Port description of the ASIC.*

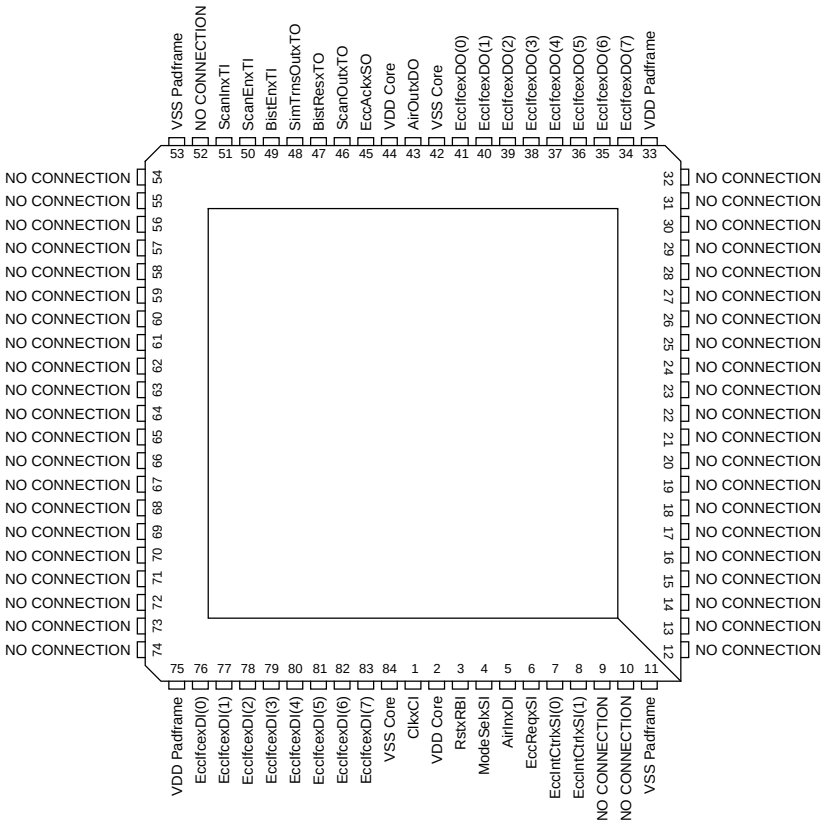



**Figure A.1:** *Pinout of ASIC.*

# Bibliography


[ABHW04] H. Aigner, H. Bock, M. Hütter, and J. Wolkerstorfer. A low-cost ECC coprocessor for smartcards. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2004.

[BL06] M. Benaissa and Wei Ming Lim. Design of flexible GF($2^m$) elliptic curve cryptography processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(6):659–662, 2006.

[DF96] W. Drescher and G. Fettweis. VLSI architectures for multiplication in GF($2^m$) for application tailored digital signal processors. In *Proc. [Workshop on] VLSI Signal Processing, IX*, pages 55–64, 1996.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[EPC05] EPC. EPC™ radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 mhz - 960 mhz. Published Standard 1.10, EPCglobal Inc., December 2005.

[EWG+05] H. Eberle, A. Wander, N. Gura, Sheueling Chang-Shantz, and V. Gupta. Architectural extensions for elliptic curve cryptography over GF($2^m$) on 8-bit microprocessors. In *Proc. 16th IEEE International Conference on Application-Specific Systems, Architecture Processors ASAP 2005*, pages 343–349, 2005.

[Fed00] Federal Information Processing Standards Publication (NIST). Digital Signature Standard (DSS). Federal Information Processing Standards Publication, FIPS PUB 186-2, January 2000.

[Fel07] Norbert Felber. *VLSI II: Design for Testability.* Microelectronics Design Center, ETH Zurich, Autumn Term 2007.

[FW07a] M. Feldhofer and J. Wolkerstorfer. Strong crypto for RFID tags - a comparison of low-power hardware implementations. In *Proc. IEEE International Symposium on Circuits and Systems ISCAS 2007*, pages 1839–1842, 2007.

[FW07b] F. Fürbass and J. Wolkerstorfer. ECC processor with low die size for RFID applications. In *Proc. IEEE International Symposium on Circuits and Systems ISCAS 2007*, pages 1835–1838, 2007.

[GK03a] Johann Großschädl and Guy-Armand Kamendje. Instruction set extension for fast elliptic curve cryptography over binary finite fields gf($2^m$). In *Proc. IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 455–468, 2003.

[GK03b] Johann Großschädl and Guy-Armand Kamendje. Optimized RISC architecture for multiple-precision modular arithmetic. In Dieter Hutter, Günter Müller, Werner Stephan, and Markus Ullmann, editors, *SPC*, volume 2802 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.

[Gor84] John Gordon. The Alice and Bob After Dinner Speech. Zürich Seminar, April 1984.

[Gor98] Daniel M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[GP97] Jorge Guajardo and Christof Paar. Efficient algorithms for elliptic curve cryptosystems. *Lecture Notes in Computer Science*, 1294:342–356, 1997.

[GSE+02] N. Gura, S. Shantz, H. Eberle, D. Finchelstein, S. Gupta, V. Gupta, and D. Stebila. An end-to-end systems approach to elliptic curve cryptography, 2002.

[HLRZ03] Chi Huang, Jinmei Lai, Junyan Ren, and Qianling Zhang. Scalable elliptic curve encryption processor for portable application. In *Proc. 5th International Conference on ASIC*, volume 2, pages 1312–1316 Vol.2, 2003.

[HMV04] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[ISO00a] ISO/IEC/JTC 1/SC 17. ISO/IEC 14443 Identification cards – Contactless integrated circuit(s) cards – Proximity cards. Published standard, International Organization for Standardization, Geneva, Switzerland., 2000.

[ISO00b] ISO/IEC/JTC 1/SC 17. ISO/IEC 15693 Identification cards – Contactless integrated circuit(s) cards – Vicinity cards. Published standard, International Organization for Standardization, Geneva, Switzerland., 2000.

[ISO04] ISO/IEC/JTC 1/SC 31. ISO/IEC 18000 Information technology – Radio frequency identification for item management. Published standard, International Organization for Standardization, Geneva, Switzerland., 2004.

[IT88] T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in GF($2^m$). *Electronics Letters*, 24(6):334–335, 1988.

[Jue06] A. Juels. RFID security and privacy: A research survey. *IEEE Journal On Selected Areas In Communications*, 24(2):381–394, 2006.

[Jue07] A. Juels. The vision of secure RFID. *Proceedings of the IEEE*, 95(8):1507–1508, 2007.

[Kae08] Hubert Kaeslin. *Digital Integrated Circuit Design*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, 2008.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83 Jan., 161–191 Feb., 1883.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Phys. Doklody*, 7(7):595–596, January 1963.

[KP06] S. Kumar and C. Paar. Are standards compliant elliptic curve cryptosystems feasible on RFID? Printed handout of Workshop on RFID Security – RFIDSec 06, July 2006.

[Lai04] G. Lai. Analysis of modular inverse GF(p) implementations. 2004.

[LD99] Julio Lopez and Ricardo Dahab. Fast multiplication on elliptic curves over GF($2^m$) without precomputation. In *Cryptographic Hardware and Embedded Systems*, number Generators, pages 316–327, 1999.

[LLM07] M. Langheinrich, M. Langheinrich, and R. Marti. Practical minimalist cryptography for RFID privacy. *IEEE Systems Journal*, 1(2):115–128, 2007.

[LLMF07] M. O. Lehtonen, M. O. Lehtonen, F. Michahelles, and E. Fleisch. Trust and security in RFID-based product authentication systems. *IEEE Systems Journal*, 1(2):129–144, 2007.

[LSMF06] Mikko Lehtonen, Thorsten Staake, Florian Michahelles, and Elgar Fleisch. From identification to authentication - a review of RFID product authentication techniques. Printed handout of Workshop on RFID Security – RFIDSec 06, July 2006.

[Mid06] Midnightcomm. Blue and purple RFID tag. Wikimedia Commons, `http://commons.wikimedia.org/wiki/Image:Blue_and_Purple_RFID_tag.jpg`, April 2006. Picture is in the public domain.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, Berlin, 1 edition, 2007.

[MvOV01] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[OP00] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for GF ($2^m$). *Lecture Notes in Computer Science*, 1965:31–43, 2000.

[PL07] S. Peter and P. Langendorfer. An efficient polynomial multiplier in gf($2^m$) and its application to ECC designs. In *Proc. Design, Automation & Test in Europe Conference & Exhibition DATE '07*, pages 1–6, 2007.

[PLP07] S. Peter, P. Langendorfer, and K. Piotrowski. Flexible hardware reduction for elliptic curve cryptography in GF($2^m$). In *Proc. Design, Automation & Test in Europe Conference & Exhibition DATE '07*, pages 1–6, 2007.

[RRR95] M. Riedel, M. Riedel, and J. Rajski. Fault coverage analysis of RAM test algorithms. In J. Rajski, editor, *Proc. th IEEE VLSI Test Symposium*, pages 227–234, 1995.

[SP98] Leilei Song and K. K. Parhi. Low-energy digit-serial/parallel finite field multipliers. *J. VLSI Signal Process. Syst.*, 19(2):149–166, 1998.

[TWA05] Wenkai Tang, Huapeng Wu, and M. Ahmadi. VLSI implementation of bit-parallel word-serial multiplier in GF($2^{233}$). In *Proc. 3rd International IEEE-NEWCAS Conference*, pages 399–402, 2005.

[Wol04] J. Wolkerstorfer. *Hardware Aspects of Elliptic Curve Cryptography*. PhD thesis, Graz University of Technology, Austria, 2004.

[Wol05] J. Wolkerstorfer. Is elliptic-curve cryptography suitable to secure rfid tags? Presentation at the Workshop on RFID and Light-weight Cryptography, Graz, Austria, August 2005.

[WT02] Lawrence C. Washington and Wade Trappe. *Introduction to Cryptography: With Coding Theory*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.