

Architectures for Multiplication in Galois Rings

Examensarbete utfört i Datatransmission
vid Tekniska Högskolan i Linköping
av

Björn Abrahamsson

Reg nr: LiTH-ISY-EX-3549-2004
Linköping 2004

Architectures for Multiplication in Galois Rings

Examensarbete utfört i Datatransmission
vid Tekniska Högskolan i Linköping
av


Björn Abrahamsson

Reg nr: LiTH-ISY-EX-3549-2004

Supervisor: **Mikael Olofsson**

Examiner: **Mikael Olofsson**

Linköping 9th June 2004.

 LINKÖPINGS UNIVERSITET	Avdelning, Institution Division, Department Institutionen för systemteknik 581 83 LINKÖPING	Datum Date 2004-06-04
--	---	------------------------------------

Språk Language Svenska/Swedish X Engelska/English	Rapporttyp Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport _____	ISBN ISRN LITH-ISY-EX-3549-2004 Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.ep.liu.se/exjobb/isy/2004/3549/		

Titel Title Författare Author	Arkitekturer för multiplikation i Galois-ringar Architectures for Multiplication in Galois Rings Björn Abrahamsson
--	--

Sammanfattning Abstract This thesis investigates architectures for multiplying elements in Galois rings of the size 4^m , where m is an integer. The main question is whether known architectures for multiplying in Galois fields can be used for Galois rings also, with small modifications, and the answer to that question is that they can. Different representations for elements in Galois rings are also explored, and the performance of multipliers for the different representations is investigated.
--

Nyckelord Keyword Galois ring, VLSI multiplication, Quaternary codes, Normal basis, Dual basis

Abstract

This thesis investigates architectures for multiplying elements in Galois rings of the size 4^m , where m is an integer.

The main question is whether known architectures for multiplying in Galois fields can be used for Galois rings also, with small modifications, and the answer to that question is that they can.

Different representations for elements in Galois rings are also explored, and the performance of multipliers for the different representations is investigated.

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem definition	1
1.3	Outline and reading instructions	2
2	Mathematical background	3
2.1	Groups, rings and fields	3
2.2	Polynomials	5
2.2.1	Irreducible polynomials over fields	6
2.2.2	Basic irreducible polynomials over rings	7
2.3	Extensions of rings and fields	7
2.4	Representation of Galois rings and fields	9
2.4.1	Galois fields as vector spaces	9
2.4.2	Galois rings	10
3	Binary representation of elements	13
3.1	Criteria for choosing representation	13
3.2	Method of optimizing representations	14
3.3	Minimizing the depth and area	15
3.4	Summary of performance	23
4	Polynomial basis representation	25
4.1	Implementation of serial multipliers	25
4.1.1	SSR multiplier	26
4.1.2	MSR multiplier	28
4.1.3	Performance of serial multipliers	28
4.2	Implementation of parallel multipliers	30
4.2.1	Construction of a parallel multiplier	30
4.2.2	Eliminating multiplications by constants	32
4.2.3	Performance of the parallel multiplier	34
4.3	Implementation of systolic multipliers	36
4.3.1	General principles of systolic architectures	36
4.3.2	Implementation for $GR(4^m)$	37

4.3.3	Performance of the systolic multiplier	38
4.4	Summary of performances	42
5	Dual basis representation	43
5.1	Definition and existence	43
5.2	Implementation of serial multipliers	45
5.2.1	Alternative serial multiplier	47
5.2.2	Performance of the serial multipliers	48
5.3	Implementation of systolic multipliers	50
5.3.1	Performance of systolic multiplier	50
5.4	Summary of performances	53
6	Normal basis multipliers	55
6.1	Definition of normal basis	55
6.2	Optimal normal bases	56
6.3	Implementation of serial multipliers	59
6.3.1	Performance of the serial multiplier	60
6.4	A simple parallel multiplier	61
6.4.1	Performance of the parallel multiplier	62
6.5	Summary of performances	62
7	Conclusions	63
7.1	Similarities with field multipliers	63
7.2	Performance aspects	64
7.2.1	Minimizing the chip area needed	64
7.2.2	Maximizing the speed	65
7.3	Possible future research	65
A	Minimal functions for binary representations	69

Chapter 1

Introduction

1.1 Background

In coding theory results and structures from abstract algebra are used extensively. Many of the most popular coding methods draw advantage of the use of finite, or Galois, fields for their descriptions, since they are linear in this context. These codes include cyclic codes, Reed-Solomon codes and BCH codes. For a description of these codes, see for example [13]. Such codes may be used for error detection and correction in for example telecommunications and CD players, and are often implemented in hardware. Since they all use the finite field structure there exists much research on how to implement elementary finite field operations in hardware, most notably VLSI.

Not so long ago (in [7] and [3]) it was shown that some codes that were previously known not to be linear over Galois fields actually were linear, cyclic codes over Galois rings. These codes include the Kerdock and Preparata codes (see [11]). The Galois rings have much in common with the Galois fields, but there are also differences. For example division is not generally possible in Galois rings. Nonetheless, their similarities imply that it could be possible to take the implementations of operations in Galois fields, make small adjustments to them and use for Galois rings, without having to do all the research over again for rings instead of fields. That is precisely what we will strive to do in this thesis.

1.2 Problem definition

For Galois fields the two important operations are multiplication and inversion, since these are more complex than addition and subtraction. Since it is not possible to divide elements in Galois rings, we only have to think about multiplication. When multiplying in Galois fields we may represent the elements in a number of different ways. All these representations are not thoroughly investigated, or even formalized, yet for Galois rings, and therefore we will try to define and explore

equivalent representations for Galois rings. We will also look at the performance of our architectures, both regarding the chip area needed and the speed.

This gives us the following goals for this thesis:

- Investigate if the architectures for multiplying in Galois Fields may easily be adjusted to Galois Rings.
- Investigate if the different types of representations of elements in Galois fields have equivalents in Galois rings.
- Compare the different possible architectures for multiplication with respect to performance and needed chip area.

1.3 Outline and reading instructions

In chapter 2 we describe the mathematical background to the thesis. This is intended as a brief introduction to the concepts used later. The chapter may be useful even to the reader that has knowledge of abstract algebra, because some concepts (i.e. the ones concerning Galois rings) are normally not treated in undergraduate courses or textbooks on the subject.

In chapter 3 we introduce some elementary operations that will be needed for the architectures in later chapters (for example addition and multiplication in the ring formed by the integers 0, 1, 2 and 3), and show how these can be implemented efficiently with logical gates.

In chapter 4 to 6 we present three different representations of the elements in Galois rings, and how multiplication can be implemented in these representations. The representations are polynomial bases (chapter 4), dual bases (chapter 5) and normal bases (chapter 6). Here we will also discuss the performance of the different implementations. The results are then summarized in chapter 7, conclusions.

For the reader who only wants to know how to implement multiplication in a Galois Ring in the best way for a certain application, it is advisable first to take a look at the conclusions chapter. From there it should be possible to see which kind of architecture is advisable, and where the details concerning it can be found, in chapter 3, 4 or 5. In these chapters the serial, parallel and systolic multipliers are presented separately and the different architectures are easy to compare between the chapters. After the architecture has been chosen, chapter 3 gives the details of implementing it with logical gates.

Chapter 2

Mathematical background

In this chapter the mathematics which are utilized throughout the thesis will be described. The presentation will be brief and proofs are not provided. For proofs and a more detailed description the interested reader is referred to [4] or [11]. In [4] the basic theory of groups, rings and fields is treated, while Galois rings are treated more in depth in [11].

2.1 Groups, rings and fields

In this section definitions of the basic mathematical structures that will be used are provided. First we will define some sets that will be used throughout this thesis.

Definition 2.1 *We define the following sets:*

- Z is the set of all integers, positive as well as negative.
- Z_m is the set of all integers modulo the integer m .
- Q is the set of all rational numbers.

Now we turn to the definition of the first of our structures, the group structure.

Definition 2.2 (Group) *A group (G, \circ) is a set G together with an operation \circ that works in the following way:*

- *The group is closed under \circ , that is for $a, b \in G$*

$$a \circ b \in G.$$

- *The operation \circ is associative, that is for $a, b, c \in G$*

$$(a \circ b) \circ c = a \circ (b \circ c).$$

- There exists an element $e \in G$, such that for any element $a \in G$

$$e \circ a = a \circ e = a.$$

- For each element $a \in G$, there exists an inverse element, denoted by a^{-1} , such that

$$a \circ a^{-1} = a^{-1} \circ a = e.$$

A group is called *commutative* if the relation $a \circ b = b \circ a$ holds for all $a, b \in G$. For an element a in a group G we define $a^n = a \circ a^{n-1}$, where $a^0 = e$.

Definition 2.3 (Order) The order of a element a in a group G is the smallest $n > 0$ such that $a^n = e$.

Example 1. The set Z is a commutative group under the operation of addition, with 0 as the element e in definition 2.2. \square

Definition 2.4 (Ring) A ring $(R, +, \cdot)$ is a commutative group $(R, +)$, with a second binary operation \cdot that satisfies the following conditions.

- The ring is closed under \cdot , that is for all $a, b \in R$

$$a \cdot b \in R.$$

- The operation \cdot is associative, that is for $a, b, c \in R$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

- The operation \cdot is distributive over $+$, that is for all elements $a, b, c \in R$

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

$$(a + b) \cdot c = a \cdot c + b \cdot c.$$

Normally we will write ab instead of $a \cdot b$, omitting the \cdot . If we, for all elements $a, b \in R$ have $ab = ba$, R is said to be a *commutative ring*. If there exists an element $1 \in R$, such that $a1 = 1a = a$ for all $a \in R$, we denominate R a *ring with identity*. These definitions can be combined to *commutative rings with identity*, the name being self-explanatory.

Example 2. The set Z_8 with the operations addition and multiplication, performed modulo 8, is a commutative ring with identity. \square

For any ring R , and an element $r \in R$ we denote $r + \dots + r$ (n r:s) by nr .

Definition 2.5 (Characteristic) The characteristic of a ring R is the smallest positive integer n such that for all $r \in R$ we have that $nr = 0$.

Definition 2.6 (Subring) A subring S of R is a subset S of R , for which we have

- $S \neq \emptyset$
- $rs \in S$ for all $r, s \in S$
- $r + s \in S$ for all $r, s \in S$

We may also say that S is a subring of R if and only if S is closed under all operations of the ring.

Example 3. The set Z is a commutative ring, with identity 1, under the normal operations of addition and multiplication. The set $S = \{2n : n \in Z\}$ is the subring consisting of all even integers. \square

Definition 2.7 (Field) A field F is a commutative ring with identity, in which there, for each $a \neq 0 \in F$ exists $b \in F$ such that

$$ab = ba = 1$$

Another way to put it is that each non-zero element has a multiplicative inverse. A field with a finite number of elements is called a *finite field* or a *Galois field*. Subfields are defined in analogy with the definition of subrings.

Example 4. The set Z_7 together with addition and multiplication performed modulo 7 is easily verified to be a Galois field. The set Z_4 on the other hand is *not* a Galois field, since 2 does not have a multiplicative inverse. \square

We will need a theorem from number theory by Fermat. The theorem is actually a special case of a more general theorem for groups.

Theorem 2.1 (Fermat's little theorem) Let p be any prime number, and suppose that p does not divide a . Then

$$a^{p-1} \equiv 1 \pmod{p}.$$

2.2 Polynomials

If we have a ring (or a field) R we can form the *polynomial ring* $R[x]$ by considering all polynomials of the form

$$f(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n \quad (2.1)$$

where $a_i \in R$, $a_n \neq 0$ and n may be any positive integer. A polynomial is called *monic* if $a_n = 1$. The polynomial in equation 2.1 is merely a formal expression, and

we may therefore not assume that it is possible to evaluate the expression by giving x a value, like we are used to with polynomials. We say that two polynomials are equal if all their *coefficients* a_i are identical, and we may add and multiply the formal polynomials just like we are used to with polynomials, bearing in mind that all operations on the coefficients are to be performed in R . If any of the coefficients $a_i = 0$ we usually omit this term from the polynomial.

Theorem 2.2 *Let R be a commutative ring with identity. Then $R[x]$ also is a commutative ring with identity.*

A polynomial ring $F[x]$ over a field F is not necessarily a field, due to the fact that all polynomials need not have an inverse. $F[x]$ is however of course always a commutative ring with identity.

We also have polynomial rings that are formed by equivalence classes modulo a polynomial $p(x)$. Let $R[x]/(p(x))$ denote such a polynomial ring. If R is a ring, $R[x]/(p(x))$ will also be a ring. We call $p(x)$ the *generator polynomial*.

Example 5. Let $R = Z_4$ and $p(x) = x^3 + 2x + 3$. We can perform multiplication between $3x^2 + 2x + 1$ and $x^2 + 3x$ in $R/(p(x))$ as follows.

$$\begin{aligned} (3x^2 + 2x + 1)(x^2 + 3x) &= 3x^4 + 3x^3 + 3x \\ &= 3x(-2x - 3) + 3(-2x - 3) + 3x \\ &= 2x^2 + 3x + 2x + 3 + 3x \\ &= 2x^2 + 3 \end{aligned}$$

The second row is due to the fact that $x^3 \equiv -2x - 3 \pmod{p(x)}$, and in the third row we use the fact that all coefficients should be in Z_4 . \square

2.2.1 Irreducible polynomials over fields

A polynomial $f(x) \in F[x]$ is said to be *irreducible* if it cannot be expressed as a product of two other polynomials in $F[x]$.

Example 6. The polynomial $x^2 + 1$ is irreducible in $Q[x]$, but it is not irreducible in $Z_2[x]$, since we there have

$$(x + 1)(x + 1) = x^2 + 2x + 1 = x^2 + 1$$

On the other hand, $x^2 + x + 1$ is irreducible in both of the fields mentioned. \square

If an irreducible polynomial $p(x)$ of degree n has a root ξ of order $q^n - 1$, where q is the number of elements in F , we say that $p(x)$ is a *primitive* polynomial.

2.2.2 Basic irreducible polynomials over rings

We will need analogy for rings to the concepts of irreducible and primitive polynomials. Define the map α by

$$\begin{aligned}\alpha : Z_4 &\rightarrow Z_2 \\ 0, 2 &\mapsto 0 \\ 1, 3 &\mapsto 1\end{aligned}$$

We will denote this map by just “ $\bar{}$ ”, that is $\bar{0} = \bar{2} = 0$ and $\bar{1} = \bar{3} = 1$. The map can naturally be extended to polynomials by mapping the coefficients.

Example 7. If we have $p(x) = 3x^2 + 2x + 1 \in Z_4[x]$, we also have $\bar{p}(x) = x^2 + 1 \in Z_2[x]$. \square

Now we can define a *basic irreducible (primitive) polynomial* as a monic polynomial $p(x)$ over Z_4 with $\bar{p}(x)$ irreducible (primitive) over Z_2 .

Example 8. The polynomial in example 7 is not basic irreducible, whereas the monic polynomial $x^2 + 3x + 3$ is a basic irreducible polynomial in $Z_4[x]$, since $x^2 + x + 1$ is irreducible in $Z_2[x]$. \square

2.3 Extensions of rings and fields

If we have a ring R (or field E) with a subring S (or subfield F), the ring R (or field E) is called an *extension ring* (or *extension field*) of the *base ring* S (or *base field* F).

Theorem 2.3 *Assume that $p(x) \in F[x]$, where F is a field, is an irreducible polynomial over F . In that case the extension ring $E = F[x]/(p(x))$ is actually a field extension of F . Assume further that $p(x)$ is of degree m , and that F has p (distinct) elements. Then the number of distinct elements, or the cardinality, of E is p^m .*

We are now ready to give the full characterization of all Galois fields.

Theorem 2.4 *All Galois fields of the same size are actually the same¹. The cardinality of a Galois field is either a prime p , or a power of a prime, p^m , where $m \in \mathbb{Z}$.*

¹To be rigid we should say that they are identical up to isomorphism.

Since the characteristics of a Galois field only depends on its size we introduce the notation $GF(p)$ for a Galois field with p elements. Combining the theorems 2.3 and 2.4 we see that we can form the field $GF(p^m)$, where $m \in \mathbb{Z}$, by using an irreducible polynomial $p(x)$ of degree m . We have $GF(p^m) = GF(p)[x]/(p(x))$.

Example 9. The field $GF(4)$ may be described as $\mathbb{Z}_2[x]/(p(x))$, where $p(x) = x^2 + x + 1$ (note that this polynomial is irreducible over \mathbb{Z}_2). Let $p(\alpha) = 0$. Then the elements in $GF(4)$ may be written

$$\begin{aligned} 0 + 0\alpha &= 0 \\ 1 + 0\alpha &= 1 \\ 0 + 1\alpha &= \alpha \\ 1 + 1\alpha &= 1 + \alpha \end{aligned}$$

Note that higher powers of α are not possible, since for example

$$\alpha^2 = \alpha^2 + \alpha^2 + \alpha + 1 = \alpha + 1$$

Below are two tables showing addition and multiplication in $GF(4)$.

\cdot	0	1	α	$1 + \alpha$	$+$	0	1	α	$1 + \alpha$
0	0	0	0	0	0	0	1	α	$1 + \alpha$
1	0	1	α	$1 + \alpha$	1	1	0	$1 + \alpha$	α
α	0	α	$1 + \alpha$	1	α	α	$1 + \alpha$	0	1
$1 + \alpha$	0	$1 + \alpha$	1	α	$1 + \alpha$	$1 + \alpha$	α	1	0

□

We now turn our attention back to the rings. For this purpose we need to remember our definition of a basic irreducible polynomial as described in section 2.2.2. We will limit ourselves to the case of rings with cardinality 4^m , $m \in \mathbb{Z}$. First we state the equivalence of theorem 2.3.

Definition 2.8 Assume that $p(x) \in \mathbb{Z}_4[x]$ is a basic irreducible polynomial of degree m . Then the extension ring $\mathbb{Z}_4[x]/(p(x))$ is called a Galois ring with 4^m elements.

For Galois rings we have the following theorem.

Theorem 2.5 All Galois rings of size 4^m and characteristic 4, where $m \in \mathbb{Z}$, $m > 0$, are actually the same².

²Or identical up to isomorphism, more correctly.

In analogy with Galois fields we introduce the notation $GR(4^m)$ for the Galois ring with 4^m elements and characteristic 4.

Example 10. The ring $GR(16)$ may be described as $Z_4[x]/(p(x))$, where $p(x) = x^2 + x + 3$ (note that $\bar{h}(x) = x^2 + x + 1$, irreducible over Z_2). Let $p(\xi) = 0$. Then the elements in $GR(16)$ may be written

$$\begin{array}{ll} 0 + 0\xi = 0 & 0 + 2\xi = 2\xi \\ 1 + 0\xi = 1 & 1 + 2\xi = 1 + 2\xi \\ 2 + 0\xi = 2 & 2 + 2\xi = 2 + 2\xi \\ 3 + 0\xi = 3 & 3 + 2\xi = 3 + 2\xi \\ 0 + 1\xi = \xi & 0 + 3\xi = 3\xi \\ 1 + 1\xi = 1 + \xi & 1 + 3\xi = 1 + 3\xi \\ 2 + 1\xi = 2 + \xi & 2 + 3\xi = 2 + 3\xi \\ 3 + 1\xi = 3 + \xi & 3 + 3\xi = 3 + 3\xi \end{array}$$

Note that higher powers of ξ are not possible, because for example

$$\xi^2 = \xi^2 + 3p(\xi) = \xi^2 + 3\xi^2 + 3\xi + 1 = 3\xi + 1$$

It is possible to write down tables for multiplying and adding the elements, but since the tables would be very large, we omit them here. \square

2.4 Representation of Galois rings and fields

In this section we will focus on different ways to represent the elements of fields and rings in a way suitable for later use. We start with the fields.

2.4.1 Galois fields as vector spaces

A finite field extension $GF(p^m)$ is a vector space over $GF(p)$. If $\{\alpha_1, \alpha_2, \dots, \alpha_m\}$ is a basis for $GF(p^m)$, then every element $\alpha \in GF(p^m)$ may be written as

$$\alpha = a_1\alpha_1 + a_2\alpha_2 + \dots + a_m\alpha_m$$

where $a_i \in GF(p)$ for $i = 1, \dots, m$. There exists a variety of different bases for a Galois field, but we will limit ourselves to a few ones with desired characteristics.

The most natural basis might be the *polynomial basis*. If $p(x)$ is the generator polynomial to $GF(p^m)$, and α is a root of $p(x)$, the set $\{\alpha^0, \alpha^1, \dots, \alpha^{m-1}\}$ is a basis of $GF(p^m)$. An example of how the elements can be described in a polynomial basis is given in example 9. The elements may also be described as vectors, which is shown in example 11.

Example 11. The table below shows the connection between the polynomial basis and the description as vectors.

Polynomial	Vector
0	(00)
1	(01)
α	(10)
$\alpha + 1$	(11)

□

2.4.2 Galois rings

Elements in Galois rings may, as an analogy to the polynomial basis for fields, be described as polynomials in a root ξ to the generator polynomial, as in example 10. The elements may also be described as vectors, even though the Galois rings are not vector spaces. Instead they are *modules*. A module is a more general structure than a vector space, but for all our needs they will have the same characteristics, and we will use the terms vector and vector space also when we mean vector (in a module) and module. An example of the representation is shown in example 12.

Example 12. The table below shows the connection between the polynomial description and the description as vectors.

Polynomial	Vector
0	(00)
1	(01)
2	(02)
3	(03)
ξ	(10)
$\xi + 1$	(11)
$\xi + 2$	(12)
$\xi + 3$	(13)
2ξ	(20)
$2\xi + 1$	(21)
$2\xi + 2$	(22)
$2\xi + 3$	(23)
3ξ	(30)
$3\xi + 1$	(31)
$3\xi + 2$	(32)
$3\xi + 3$	(33)

□

2-adic representation

We will now explore a representation of the elements in $GR(4^m)$ which will serve us for theoretical rather than computational purposes, the 2-adic representation. We will need the definition of a *basic primitive polynomial* $p(x)$, which means that $\bar{p}(x)$ is primitive, and $p(x)$ is monic. It can be shown that there exists at least one basic primitive polynomial with degree m for every positive integer m . We now have the following theorem.

Theorem 2.6 (2-adic representation) *In the Galois ring $GR(4^m)$ there exists a nonzero element ξ of order $2^m - 1$ which is a root of a basic primitive polynomial.*

- Let $\mathcal{T} = \{0, 1, \xi, \dots, \xi^{2^m-2}\}$. Now any element $c \in GR(4^m)$ may be written uniquely as $c = a + 2b$ where $a, b \in \mathcal{T}$.
- An element c is invertible if and only if $a \neq 0$.
- An element c is a multiple of 2 if and only if $a = 0$.
- The order of c is a divisor of $2^m - 1$ if and only if $a \neq 0$ and $b = 0$.

We define a function that will be useful for us further on.

Definition 2.9 (Frobenius map) *Write $c = a + 2b$ in 2-adic representation. Define the function f as*

$$\begin{aligned} f : GR(4^m) &\rightarrow Z_4 \\ c = a + 2b &\rightarrow c^f = a^2 + 2b^2 \end{aligned}$$

The function is called the Frobenius map.

Example 13. Let $R = Z_4[x]/(p(x))$, where $p(x) = x^3 + 2x^2 + x + 3$. Let further $p(\xi) = 0$. Now ξ is an element of order $2^3 - 1 = 7$. Hence we can use ξ to represent all elements in the 2-adic form. We have for the different powers of ξ :

$$\begin{aligned} \xi^0 &= 1 \\ \xi^1 &= \xi \\ \xi^2 &= \xi^2 \\ \xi^3 &= 2\xi^2 + 3\xi + 1 \\ \xi^4 &= 3\xi^2 + 3\xi + 2 \\ \xi^5 &= \xi^2 + 3\xi + 3 \\ \xi^6 &= \xi^2 + 2\xi + 1 \\ \xi^7 &= 1. \end{aligned}$$

Hence for this example we have

$$\mathcal{T} = \{0, 1, \xi, \xi^2, 2\xi^2 + 3\xi + 1, \\ 3\xi^2 + 3\xi + 2, \xi^2 + 3\xi + 3, \xi^2 + 2\xi + 1\}$$

and all elements $c \in R$ may be written as $c = a + 2b$, where $a, b \in \mathcal{T}$. As an example of this we see that the element $\alpha = \xi^2 + 3\xi + 2$ may be described as $\alpha = \xi^4 + 2\xi^2 = 3\xi^2 + 3\xi + 1 + 2\xi^2 = \xi^2 + 3\xi + 2$. We calculate α^f :

$$\alpha^f = (\xi^4)^2 + 2(\xi^2)^2 = \xi^8 + 2\xi^4 = \xi + 2\xi^4. \quad (2.2)$$

□

Theorem 2.7 *For the Frobenius map we have*

$$\begin{aligned} (cd)^f &= c^f d^f \\ (c+d)^f &= c^f + d^f \\ c^{f^m} &= c \\ n^f &= n \end{aligned}$$

where $c, d \in GR(4^m)$ and $n \in Z_4$.

We will also need the definition of the so called *trace function* T .

Definition 2.10 (Trace function) *Suppose that $c = a + 2b$ in 2-adic representation. Define the trace function from $GR(4^m)$ to Z_4 as*

$$\begin{aligned} T(c) &= c + c^f + c^{f^2} + \dots + c^{f^{m-1}} \\ &= (a + 2b) + (a^2 + 2b^2) + (a^{2^2} + 2b^{2^2}) + \dots + (a^{2^{m-1}} + 2b^{2^{m-1}}) \end{aligned}$$

The trace function has some useful characteristics that will be valuable later.

Theorem 2.8 *For the trace function T the following properties hold*

- $T(c + c') = T(c) + T(c')$ for all $c, c' \in GR(4^m)$
- $T(ac) = aT(c)$ for all $a \in Z_4$ and $c \in GR(4^m)$
- T is surjective.

We see from the first two properties that the trace function is linear over Z_4

Example 14. We continue from example 13, and calculate $T(\alpha)$. We know that $T(\alpha) = \alpha + \alpha^f + \alpha^{f^2}$, and that $\alpha^f = \xi + 2\xi^4$. We now also have

$$\alpha^{f^2} = (\xi)^2 + 2(\xi^4)^2 = \xi^2 + 2\xi. \quad (2.3)$$

This gives us

$$\begin{aligned} T(\alpha) &= \xi^2 + 3\xi + 2 + \xi + 2\xi^4 + \xi^2 + 2\xi = \\ &= \xi^2 + 3\xi + 2 + \xi + 2\xi^2 + 2\xi + \xi^2 + 2\xi = 4\xi^2 + 8\xi + 2 = 2. \end{aligned}$$

□

Chapter 3

Binary representation of elements

In this chapter we will deal with the two-bit binary representation of the elements of Z_4 , namely 0, 1, 2 and 3. We will investigate how the choice of representation controls the performance of the basic operations needed when multiplying in $GR(4^m)$.

3.1 Criterias for choosing representation

To decide which binary representation is the best, we need to establish criterias for what we mean by “best”. First of all we need to define the operations which we wish to implement. We will study the operations

- multiplication between two elements in Z_4
- addition between two elements in Z_4
- subtraction of one element from another in Z_4 .

These are the basic binary operations that exist in Z_4 , since division is not defined for the ring. Later we will also see that all these operations will be needed when implementing our architectures. Note that multiplication and addition are commutative operations, whereas subtraction is not. Apart from these general operations we will need a few more special operations. We will at times need to multiply with a constant element, known while constructing the circuit. If this constant is 0 or 1 the implementation is of course trivial, but if it is 2 or 3 logical gates may be needed for the implementation. Note that a multiplication by 3 in Z_4 is equal to a negation. This gives us five different operations of interest, the last two being

- multiplication of elements in Z_4 by the constant 2

- negation of elements in Z_4 , which also can be viewed as multiplication by the constant 3.

We also need to consider what the objective of the optimization is. Here we have two choices, namely

- minimize number of gates needed
- minimize depth of net, i.e. minimize the largest number of gates in any path from input signal to output signal.

The reason for choosing these two objectives is that they will give nice properties when implemented in VLSI. Minimizing the number of gates will demand the smallest chip area, and minimizing the depth will give the opportunity to use the highest possible clock frequency. Which is most important, a small chip area or a fast circuit will of course differ from time to time. We will treat both the case of minimizing the depth, and the case of minimizing the number of gates.

To simplify our search for the best implementation we will limit ourselves in some ways. First of all, we will only allow gates with one or two inputs. This means that for example 3-input and gates will not be allowed. This is a simplification we do to make it easier to compare the different representations. We will also assume that all gates delay the signal equally much, and need the same area on a chip.

Note that these simplifications make it impossible to state that the logical circuits we say are the best will always be the best when implemented in VLSI. All types of gates do not need the same number of transistors (and hence not the same chip area), and do not cause equal delay to the signal. It is also possible that allowing gates with more than two inputs would make the implementations faster or smaller. For a discussion of VLSI considerations see for example [9].

3.2 Method of optimizing representations

To find the best possible representation, we have to look at all possible representations and see which representation gives us the best performance for the operations we have chosen. Simple combinatorics tells us that we have 24 possible representations of the numbers. However, 12 of these are equivalent to the 12 other. This can easily be realized if we take in mind that the order of the two bits is not significant. Switching the bit-order of a representation will generate the same output (with the bit-order reversed, of course). From now on, whenever we talk about the properties of a representation, the same properties are valid for the representation with reversed bit-order.

Before going into the different representations and the results they will bring us we will look at what results we might expect, in the best case. For multiplication and addition we must take a few things into account when considering the least possible depth and number of gates for the implementation. First of all, the operations are commutative, which means for the implementations that they are symmetric. Hence, if for example the calculation of an output signal needs x_2 , also

y_2 is needed. Furthermore, all input signals are needed to calculate the total output, and no output signal is independent of the inputs (since both multiplication and addition are surjective). Nor is it possible that each bit depends on only one of the input bits (of both operands). This last claim is not as obvious as the others, and we will only briefly explain the reason for it here. Assume that one bit states if the number is odd or even. Then the other must indicate to which pair of one odd and one even number it belongs. The information of odd-even of the input signals is used to decide if the output is odd or even, but the pairs the inputs belong to are not sufficient to say which pair the output will belong to, here we also need the odd-even information. For example, if we know that both inputs are either 1 or 2, this is not sufficient to tell if the product of them is 0, 1 or 2. This implies that for at least one of the outputs we need all four inputs to calculate this, and for the other we need at least two input signals. It can be shown that the same is true even if no bit has the odd-even significance, but rather divides Z_4 into two other pairs. It is easily understood that four input signals means at least three gates, and two inputs necessitates one gate. Now consider subtraction. It is obvious in the same way as for multiplication and addition that all input signals are significant, and therefore needed for one of the output signals. The other can not be independent of the input signals, and since it's never possible that only one input signal determines an output signal at least two input signals will be needed for the other output signal. In total this means that we need at least 1 respective 3 gates for the outputs, just as with addition and multiplication. In the ideal case no gates at all are needed for negation (the operation is "free"). This might sound surprising, but when 3 and 1 are represented with one 0 and one 1, and 0 and 2 with two 0:s or two 1:s, we easily see that switching the bitorder is equal to negation. In the same way we see that if we for example represent 0 with 00 and 2 with 10 the second output bit when multiplying by 2 will always be 0, and the first output bit will be equal to the second input bit (which is 1 for 1 and 3. Therefore both negation and multiplication by 2 is possible to implement without any gates at all.

3.3 Minimizing the depth and area

For the rest of the chapter, let m_1m_2 denote the binary result of multiplying the binary numbers x_1x_2 and y_1y_2 , a_1a_2 the result when adding them, and s_1s_2 the result when subtracting y_1y_2 from x_1x_2 . Let also n_1n_2 denote the result of negating x_1x_2 , and d_1d_2 the result of multiplying x_1x_2 by 2.

In appendix A the 12 different representations (remember that shifting the bitorder doesn't change anything) are listed, together with the minimal functions for the operations we are interested in. These have been obtained from the Karnaugh diagrams for the different representations and operations and then simplified as much as possible, using all possible gate types.

Looking at the functions in appendix A we see that there exists only one representation with both multiplication and addition optimal (a depth of 2), and that is the *natural representation*, where $0 = 00$, $1 = 01$, $2 = 10$ and $3 = 11$. It is

however not theoretically optimal when it comes to subtraction, one input signal needs to be inverted, for a total depth of 3, but as we can see from the table no other representation is better. The natural representation needs one gate depth for negation, but all representations for which negation is free needs gates for multiplying by 2 and need far more gates for addition and subtraction, and hence we draw the conclusion that the natural representation is the best one. The only exception is when we need to perform a large number of negations, and not so many other operations. The natural representation and the representation with the bit-order shifted are shown in table 3.1.

Below we show how the minimal functions can be obtained from the minimal polynomials extracted from the Karnaugh diagrams.

$$\begin{aligned}
m_1 &= x_1y_1'y_2 + x_1x_2'y_2 + x_1'x_2y_1 + x_2y_1y_2' \\
&= x_1y_2(x_2' + y_1') + x_2y_1(x_1' + y_2') \\
&= x_1y_2(x_2y_1)' + x_2y_1(x_1y_2)' \\
&= (x_1y_2) \oplus (x_2y_1) \\
m_2 &= x_2y_2 \\
a_1 &= x_1y_1'y_2' + x_1x_2'y_1' + x_1'x_2'y_1 + x_1'y_1y_2' + x_1'x_2y_1'y_2 + x_1x_2y_1y_2 \\
&= x_1y_1'(x_2' + y_2') + x_1'y_1(x_2' + y_2') + x_2y_2(x_1'y_1' + x_1y_1) \\
&= (x_1 \oplus y_1)(x_2y_2)' + x_2y_2(x_1 \oplus y_1)' \\
&= (x_1 \oplus y_1) \oplus (x_2y_2) \\
a_2 &= x_2 \oplus y_2 \\
s_1 &= x_1y_1'y_2' + x_1x_2y_1' + x_1'x_2'y_1'y_2 + x_1x_2'y_1y_2 + x_1'x_2y_1 + x_1'y_1y_2' \\
&= x_1y_1'(y_2' + x_2) + x_1'y_1(x_2' + y_2') + x_2'y_2(x_1'y_1' + x_1y_1) \\
&= (x_1'y_1 + x_1y_1')(x_2'y_2)' + x_2'y_2(x_1'y_1' + x_1y_1) \\
&= (x_1 \oplus y_1)(x_2'y_2)' + x_2'y_2(x_1 \oplus y_1)' \\
&= (x_1 \oplus y_1) \oplus (x_2'y_2) \\
s_2 &= x_2y_2' + x_2'y_2 = x_2 \oplus y_2 \\
n_1 &= x_1 \oplus x_2 \\
n_2 &= x_2 \\
d_1 &= x_2 \\
d_2 &= 0
\end{aligned}$$

In figures 3.1-3.5 the implementation of the above equations are shown implemented with logical gates. We can see that 4 gates are needed for multiplication, 4 for addition, 5 for subtraction, 1 for negation and no gates are needed for multiplication by 2. We see that for all operations except negation this is the least number of gates needed by any of the representations.

We can also see from the equations above that since it is the negation of x_2 that

Element	Representation 1	Representation 2
0	00	00
1	01	10
2	10	01
3	11	11

Table 3.1. Representations for minimum depth except for negation.

makes the depth of subtraction grow to 3 this will not necessarily mean that the subtraction will contribute with depth 3 to the critical path. Since the depth for the second bit in addition and multiplication is only 1 we can input an extra inverter after this without increasing the depth of the total operation. Hence, whenever a subtraction is directly preceded by an addition or multiplication, the addition to the length of the critical path is 2 for the subtraction.

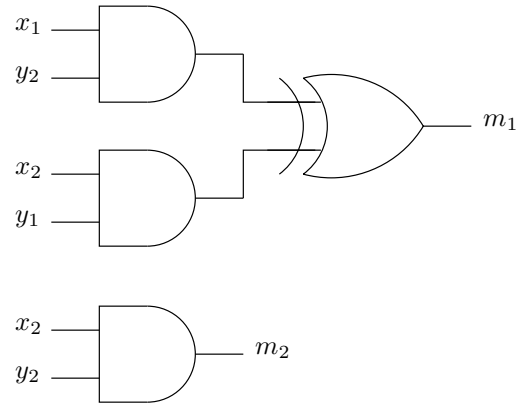


Figure 3.1. Implementation of multiplication for representation in 3.1.

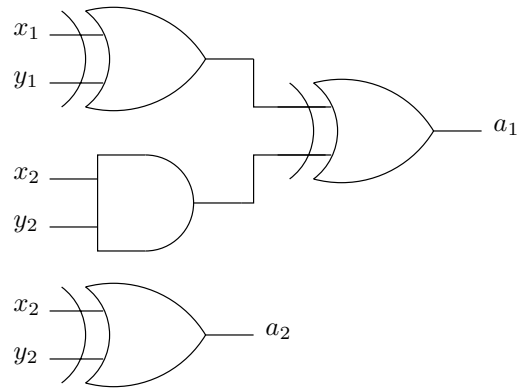


Figure 3.2. Implementation of addition for representation in table 3.1.

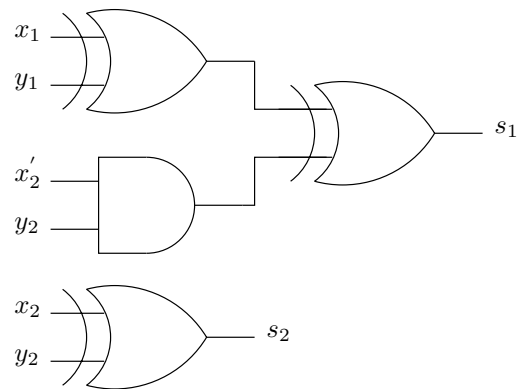


Figure 3.3. Implementation of subtraction for representation in table 3.1.

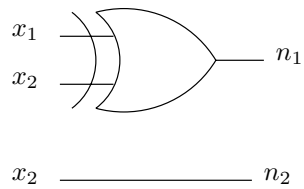


Figure 3.4. Implementation of negation for representation in table 3.1 .

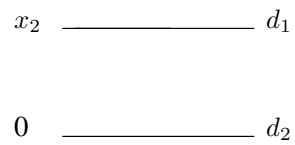


Figure 3.5. Implementation of multiplication by 2 for representation in table 3.1 .

The only downside to the natural representation is, as we have seen, that it needs one gate for negation. Hence, another representation that doesn't need any gates for negation could be better when many negations are to be performed. Of the representations in appendix A there are two that doesn't need any gates for negation the first representation in table 3.2 is obviously the better one, since it doesn't need any input signals to be inverted for the other operations. Also in the table we see the representation with reversed bit-order.

Element	Representation 1	Representation 2
0	00	00
1	01	10
2	11	11
3	10	01

Table 3.2. Representations for minimum depth of negation.

Below are the minimal functions for this representation.

$$\begin{aligned}
 m_1 &= (x_1 y_2) \oplus (x_2 y_1) \\
 m_2 &= (x_2 y_2) \oplus (x_1 y_1) \\
 a_1 &= (x_1 \oplus y_1) \oplus \underline{((x_1 \oplus x_2)(y_1 \oplus y_2))} \\
 a_2 &= (x_2 \oplus y_2) \oplus \underline{((x_1 \oplus x_2)(y_1 \oplus y_2))} \\
 s_1 &= (x_1 \oplus y_2) \oplus \underline{((x_1 \oplus x_2)(y_1 \oplus y_2))} \\
 s_2 &= (x_2 \oplus y_1) \oplus \underline{((x_1 \oplus x_2)(y_1 \oplus y_2))} \\
 n_1 &= x_2 \\
 n_2 &= x_1 \\
 d_1 &= x_1 \oplus x_2 \\
 d_2 &= x_1 \oplus x_2
 \end{aligned}$$

The underlines indicate that the same gates are used more than once. In the figures 3.6-3.10 the implementations for this representation are shown.

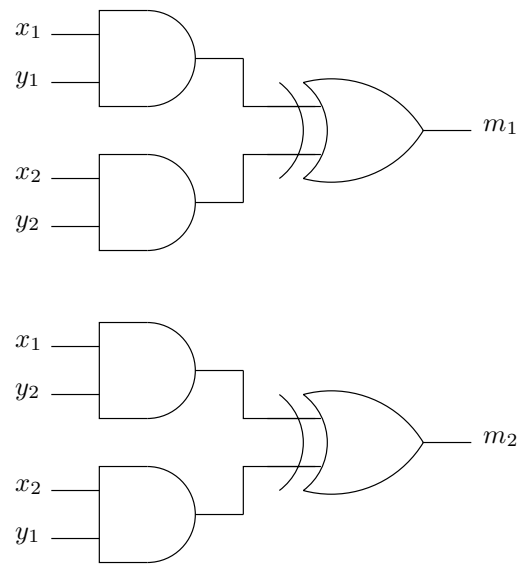


Figure 3.6. Implementation of multiplication for representation in table 3.2.

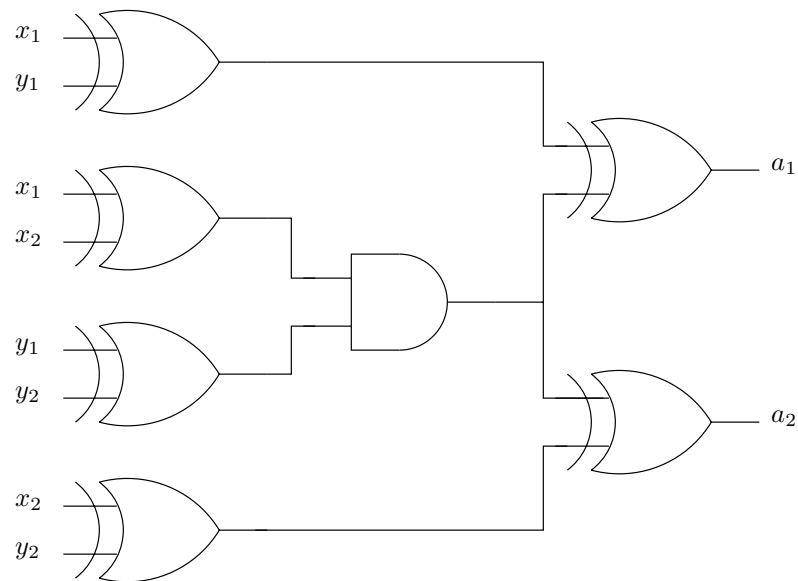


Figure 3.7. Implementation of addition for representation in table 3.2.

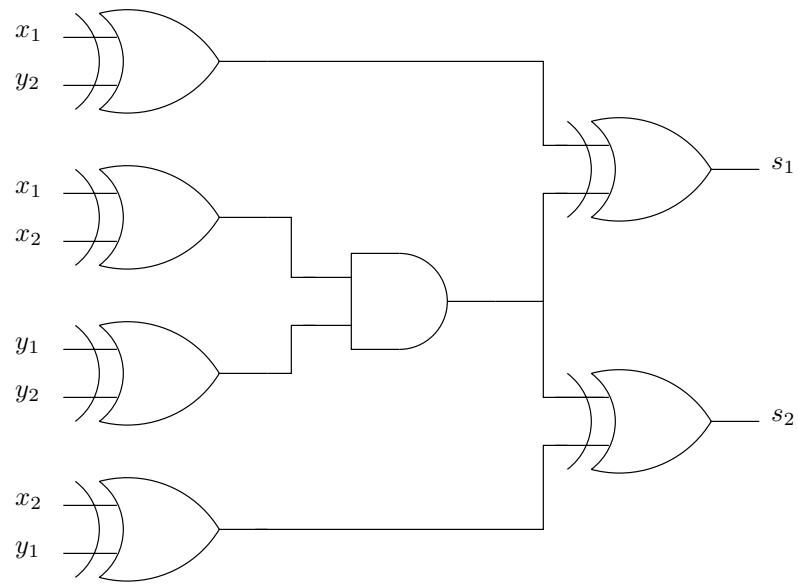


Figure 3.8. Implementation of subtraction for representation in table 3.2.

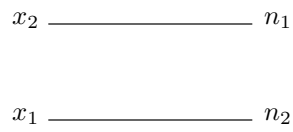


Figure 3.9. Implementation of negation for representation in table 3.2.

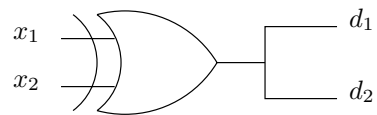


Figure 3.10. Implementation of multiplication by 2 for representation in table 3.2 .

3.4 Summary of performance

We end this chapter by giving the performances of the representations discussed. This is done in the table below. Remember that the same performances may be obtained by switching the bit-order of the representations.

	Representation			
	0 = 00, 1 = 01, 2 = 10, 3 = 11		0 = 00, 1 = 01, 2 = 11, 3 = 10	
	Depth	Gates	Depth	Gates
$x \cdot y$	2	4	2	6
$x + y$	2	4	3	7
$x - y$	3	5	3	7
$-x$	1	1	0	0
$2x$	0	0	1	2

Table 3.3. Performance of the two representations.

Chapter 4

Polynomial basis representation

In this chapter structures for performing multiplication in $GR(4^m)$ using the polynomial basis representation will be described. The polynomial basis representation has been presented in section 2.4.2. We will explore three types of implementations, serial multipliers, parallel multipliers and systolic multipliers. The implementations will be described in terms of operations in Z_4 . How the different operations can be implemented in gates has been discussed in chapter 3. When studying the performances, regarding speed and needed chip area, of our implementations we will use the results from chapter 3.

4.1 Implementation of serial multipliers

For the rest of this section, we will assume that we have a ring generated by the (basic irreducible) polynomial

$$p(x) = \sum_{i=0}^m p_i x^i = p_0 + p_1 x + \dots + x^m \quad (4.1)$$

in which we wish to multiply the two polynomials $a(x)$ and $b(x)$:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + \dots + a_{m-1} x^{m-1}$$

$$b(x) = \sum_{i=0}^{m-1} b_i x^i = b_0 + b_1 x + \dots + b_{m-1} x^{m-1}.$$

The result of the multiplication $a(x)b(x) \pmod{p(x)}$ is denoted $c(x)$, and written

$$c(x) = \sum_{i=0}^{m-1} c_i x^i = c_0 + c_1 x + \dots + c_{m-1} x^{m-1}.$$

4.1.1 SSR multiplier

The SSR (Standard Shift-Register) multiplier is the perhaps most intuitive, and oldest, serial multiplier for Galois fields. Here we will transform the multiplier presented in [6] into a multiplier for the Galois ring $GR(4^m)$. We have

$$\begin{aligned} c(x) &= a(x)b(x) \pmod{p(x)} \\ &= a(x)(b_0 + b_1 x + \dots + b_{m-1} x^{m-1}) \pmod{p(x)} \\ &= b_0 a(x) + b_1 x a(x) + \dots + b_{m-1} x^{m-1} a(x) \pmod{p(x)} \quad (4.2) \\ &= (b_0 a(x) \pmod{p(x)}) + (b_1 x a(x) \pmod{p(x)}) + \dots + \\ &\quad + (b_{m-1} x^{m-1} a(x) \pmod{p(x)}) \\ &= \sum_{i=0}^{m-1} (b_i x^i a(x) \pmod{p(x)}), \end{aligned}$$

where the terms $b_i x^i a(x) \pmod{p(x)}$ may be computed recursively by multiplying by one x at a time, and calculating the result modulo $p(x)$. An example of how this is done for b_3 is shown below.

$$\begin{aligned} b_3 x^3 a(x) &= (b_3 x^2 a(x) \pmod{p(x)}) x \pmod{p(x)} \\ &= (((b_3 a(x) \pmod{p(x)}) x \pmod{p(x)}) x \pmod{p(x)}) x \pmod{p(x)}. \end{aligned}$$

Figure 4.1 shows the implementation of the SSR multiplier. The polynomials $a(x)$ and $b(x)$ are loaded serially into the r_i registers. During the first clock cycle $b_{m-1} a(x)$ is calculated and the result is stored in the z registers. The registers containing $b(x)$ and $z(x)$ are then shifted left one step, corresponding to a multiplication by x . This gives us, after shifting $z(x)$:

$$z(x) = z_m x^m + z_{m-1} x^{m-1} + \dots + z_1 x + z_0,$$

where $z_0 = 0$. To reduce this modulo $p(x)$ we subtract $z_m p(x)$ from $z(x)$:

$$\begin{aligned} z(x) - z_m p(x) &= z_m x^m + z_{m-1} x^{m-1} + \dots + z_0 + \\ &\quad + (-z_m x^m - z_m p_{m-1} x^{m-1} - \dots - z_m p_0) \\ &= (z_{m-1} - z_m p_{m-1}) x^{m-1} + \dots + (z_0 - z_m p_0) \\ &= \sum_{i=0}^{m-1} (z_i - z_m p_i) x^i. \end{aligned}$$

After this reduction modulo $p(x)$ we add $b_{m-2} a(x)$, with the reduction and addition performed in the E_i cells of figure 4.1. In the z registers we now have

$b_{m-1}xa(x) + b_{m-2}a(x)$ and we see that after repeating the same procedure as above for all b_i we will have our result in the z registers. The result is thereafter returned serially using the upper r_i registers.

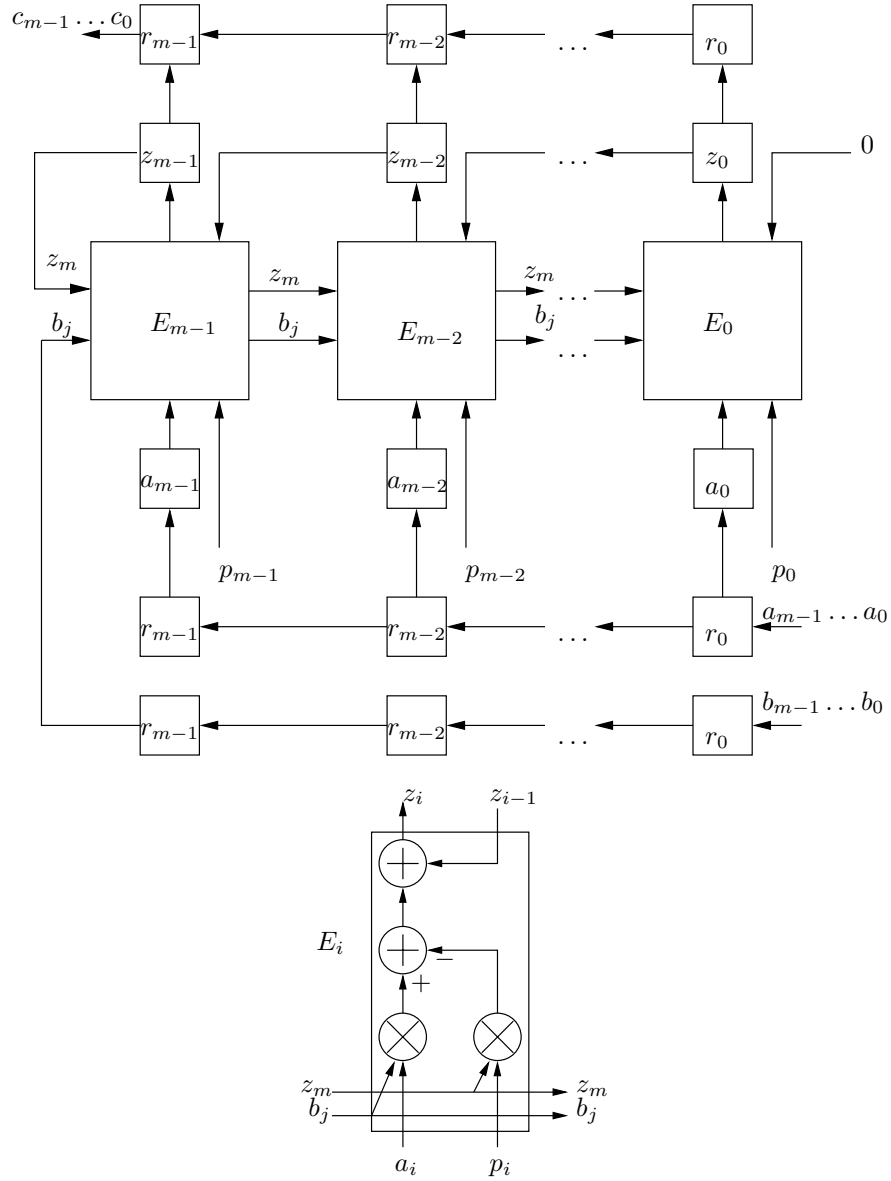


Figure 4.1. Implementation of the SSR multiplier for $GR(4^m)$.

4.1.2 MSR multiplier

In [6] a minor modification of the SSR multiplier is proposed. The new multiplier for fields is called the Modified Shift-Register (or MSR) multiplier. This can also be used for Galois rings. Remembering equation 4.2 we have

$$c(x) = b_0a(x) + b_1xa(x) + \dots + b_{m-1}x^{m-1}a(x) \bmod p(x)$$

Now we can define polynomials $Z_{-,j}(x)$ as

$$Z_{-,j}(x) = \sum_{i=0}^{m-1} z_{i,j}x^i = x^j a(x) \bmod p(x). \quad (4.3)$$

This gives us

$$c(x) = \sum_{j=0}^{m-1} b_j Z_{-,j}(x).$$

In matrix notation we can write

$$C = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} z_{0,0} & z_{0,1} & \dots & z_{0,m-1} \\ z_{1,0} & z_{1,1} & \dots & z_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m-1,0} & z_{m-1,1} & \dots & z_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix} = ZB$$

From equation 4.3 we can see that the columns in the matrix are formed by merely multiplying the former column by x (and reducing modulo $p(x)$). This is equal to that $Z_{-,1}$ is formed by shifting $Z_{-,0}$ and reducing modulo $p(x)$. Therefore we need to first calculate $Z_{-,0}b_0$, then calculate $Z_{-,1}b_1$ and add to the former result, and repeat this for all columns in Z . The implementation of the MSR multiplier is shown in figure 4.2. In the figure the upper part is responsible for the shifting and reducing modulo $p(x)$, while the lower part sums up the terms for the different c_i 's, through a feedback of the temporary sum. After m clockcycles the result will be given in parallel form (it can, of course, be put in registers and serially shifted out, as in the SSR case, to provide the result in serial form).

4.1.3 Performance of serial multipliers

From the figures 4.1 and 4.2 we can easily determine the performance of the architectures in terms of speed and area. We will use the natural representation from chapter 3, since it's been shown to be the best except for the case where we have an abundance of negations, which is not the case here. For the SSR multiplier we see that the longest path a signal has to travel through during one clockcycle contains one multiplication, one subtraction and one addition. Since, according to chapter 3, these operations has a depth of 2, 3 and 2, the critical path should contain 7 gates. But, as noted in section 3.3, when a subtraction is preceded by a multiplication, it only adds a depth of 2 gates to the critical path. Therefore the critical

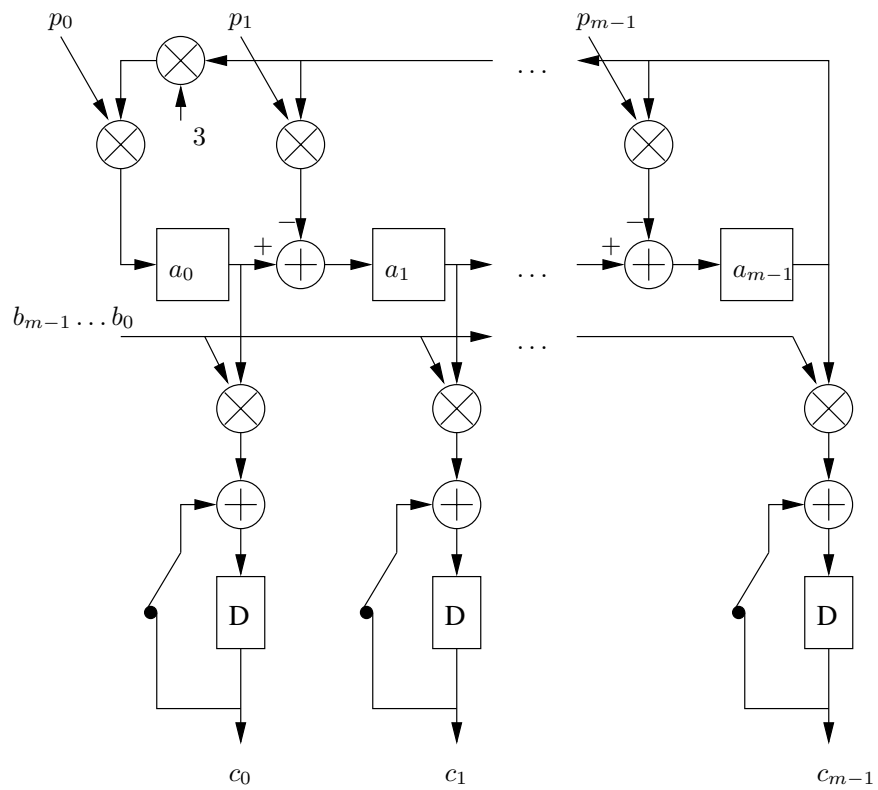


Figure 4.2. Implementation of the MSR multiplier for $GR(4^m)$.

path consists of 6 gates for the SSR multiplier. We see also that the delay, i.e. the time from the input reaches the circuit until the output begins leaving it, is $2m$ clock cycles. Of these, m cycles are needed for the actual calculations, and m for the serial input and output of the data. The throughput is decided by how often we may introduce new data into the circuit, and since the actual calculations need m clock cycles, we may input new data each m clock cycles, and new output will be given just as often. This means the throughput is $1/m$ results per clock cycle. We see further that the SSR multiplier is comprised by m cells, all performing 2 multiplications, 1 subtraction and 1 addition. Since multiplication needs 4 gates, subtraction 5 and addition 4, this gives a total of $17m$ gates. Adding to this, we also need $5m$ registers, as can be seen in the figure.

Turning our attention to the MSR multiplier we see that the critical path here contains 1 multiplication and 1 subtraction. Since the subtraction here is preceded by another subtraction, we must count 3 gates as its addition to the critical path, for a total of 5 gates in the critical path. In the same way as for the SSR case we see that the delay is $2m$ clock cycles, and the throughput $1/m$ results per clock cycle. For the area, we see that the upper part of the circuit needs m multiplications, $m - 1$ subtractions and 1 negation (multiplication by 3). The lower part needs m multiplications and additions, for a total of $2m$ multiplications, m additions, $m - 1$ subtractions and 1 negation. This sums up to a total of $17m - 4$ gates. Furthermore a total of $5m$ registers are needed. This is not shown in the figure, but considering that we need the same registers for input and output of the data serially as in the SSR case we get this number.

From the calculations above we see that the MSR multiplier is slightly better than the SSR. They need approximately the same chip area, have the same delay and throughput but the critical path is one sixth shorter, which can be used for clocking the circuit faster.

4.2 Implementation of parallel multipliers

The standard polynomial parallel multipliers for fields are normally more complicated to construct than their serial counterparts. This is primarily due to that their implementation is dependent upon the generator polynomial $p(x)$, which means that there is the additional problem of choosing the most suitable polynomial. For Galois rings the parallel multiplier may be constructed similarly as for Galois fields. We will begin by describing the general procedure when constructing a parallel multiplier. After that the role of the generator polynomial for the construction procedure and final architecture will be treated briefly.

4.2.1 Construction of a parallel multiplier

Assume that we wish to multiply two elements in the Galois ring generated by the (basic irreducible) polynomial $p(x) = x^4 + x + 1$, $GR(4^4)$. Denote the multiplicands

as

$$\begin{aligned} a(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 \\ b(x) &= b_0 + b_1x + b_2x^2 + b_3x^3. \end{aligned}$$

First we note that we have

$$\begin{aligned} x^4 &= 3x + 3 \\ x^5 &= 3x^2 + 3x \\ x^6 &= 3x^3 + 3x^2. \end{aligned}$$

We now perform the laborious task of multiplying $a(x)$ and $b(x)$ by hand.

$$\begin{aligned} c(x) = a(x)b(x) &= (a_0 + a_1x + a_2x^2 + a_3x^3)(b_0 + b_1x + b_2x^2 + b_3x^3) \\ &= a_0b_0 + [a_0b_1 + a_1b_0]x + [a_0b_2 + a_1b_1 + a_2b_0]x^2 + \\ &\quad + [a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0]x^3 + [a_1b_3 + a_2b_2 + a_3b_1]x^4 \\ &\quad + [a_2b_3 + a_3b_2]x^5 + a_3b_3x^6 \\ &= a_0b_0 + [a_0b_1 + a_1b_0]x + [a_0b_2 + a_1b_1 + a_2b_0]x^2 + \\ &\quad + [a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0]x^3 + \\ &\quad + [a_1b_3 + a_2b_2 + a_3b_1](3x + 3) + [a_2b_3 + a_3b_2](3x^2 + 3x) + \\ &\quad + a_3b_3(3x^3 + 3x^2) \\ &= [a_0b_0 + 3a_3b_1 + 3a_2b_2 + 3a_1b_3] + \\ &\quad + [a_1b_0 + (a_0 + 3a_3)b_1 + (3a_2 + 3a_3)b_2 + (3a_1 + 3a_2)b_3]x + \\ &\quad + [a_2b_0 + a_1b_1 + (a_0 + 3a_3)b_2 + (3a_2 + 3a_3)b_3]x^2 + \\ &\quad + [a_3b_0 + a_2b_1 + a_1b_2 + (3a_3 + a_0)b_3]x^3. \end{aligned}$$

The result of the multiplication may be expressed with matrices. Let

$$Z = \begin{pmatrix} a_0 & 3a_3 & 3a_2 & 3a_1 \\ a_1 & a_0 + 3a_3 & 3a_2 + 3a_3 & 3a_1 + 3a_2 \\ a_2 & a_1 & a_0 + 3a_3 & 3a_2 + 3a_3 \\ a_3 & a_2 & a_1 & 3a_3 + a_0 \end{pmatrix}. \quad (4.4)$$

Then we have

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = Z \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Now that we know an expression for the multiplier, the question is how to implement it. We choose to use the same architecture as is used for a Galois field multiplier in section 4.2 in [6]. This multiplier is often referenced to as the Mastrovito multiplier, and is possible to translate almost entirely to work for Galois rings.

First we note that the Z matrix is a function of the a_i :s, and therefore we let

$$Z = (f_{i,j}(a_0, \dots, a_3)) \quad (4.5)$$

where $0 \leq i, j \leq 3$. This gives us

$$c_i = \sum_{j=0}^3 f_{i,j}(a_0, \dots, a_3) b_j \quad (4.6)$$

and we now see that all c_i are computed as inner products between the functions $f_{i,j}$ and the b_j :s. Hence we can divide the multiplication into two parts, one that computes the values of the functions $f_{i,j}$ using the a_i :s, and one that implements the inner products. Looking at the matrix Z we see that some elements are equal, which means that some of the functions are actually the same. To benefit from this we introduce a third part into our implementation, a bus used to connect the part computing the functions and the inner products. All together we see the implementation in figure 4.3, where we, from left to right, calculate the functions in Z , transmit them via the bus and calculate the inner products. We see that the rightmost part, calculating the inner products, only depends on the size of the Galois Ring, not on the generator polynomial, while the two other parts depends on the polynomial itself. This incurs the drawback of having to reconstruct the network for each new generator polynomial we want to use. It also means that some polynomials will be more suitable as generator polynomials than others, since the complexity of the implementation to some degree depends on the generator polynomial. This inconvenience of having to reconstruct the network for a new generator polynomial is the reason for not using subtractions in figure 4.3. Where we have a multiplication by 3 (or negation), followed by a multiplication and then by an addition, we could have instead used just the multiplication followed by a subtraction. This would have shortened the critical path. The down-side, however, would have been that the rightmost part of the figure would now also be dependent on the generator polynomial, making the construction procedure a little less straightforward. For this reason we have chosen not to do this optimization here, but if speed is really important, it should of course be done.

4.2.2 Eliminating multiplications by constants

In this section we will discuss a detail regarding the generating polynomial that can be observed when studying section 4.2.1. As we can see from the description of the Z array in 4.4, an implementation of the parallel multiplier in the ring $GR(4^4)$, generated by the polynomial $x^4 + x + 1$ needs to perform several different multiplications by coefficients with the constant 3. As can be seen from the calculations, all these 3:s originates from the fact that $x^4 = 3x + 3$. If we instead had chosen the polynomial $x^4 + 3x + 3$, we would have had $x^4 = x + 1$, and all multiplications by 3 would have disappeared. We see that the same thing goes for all polynomials of the form $x^m + ax + b$. If possible, a and b should be chosen to 3 if a trinomial of the form above is to be used.

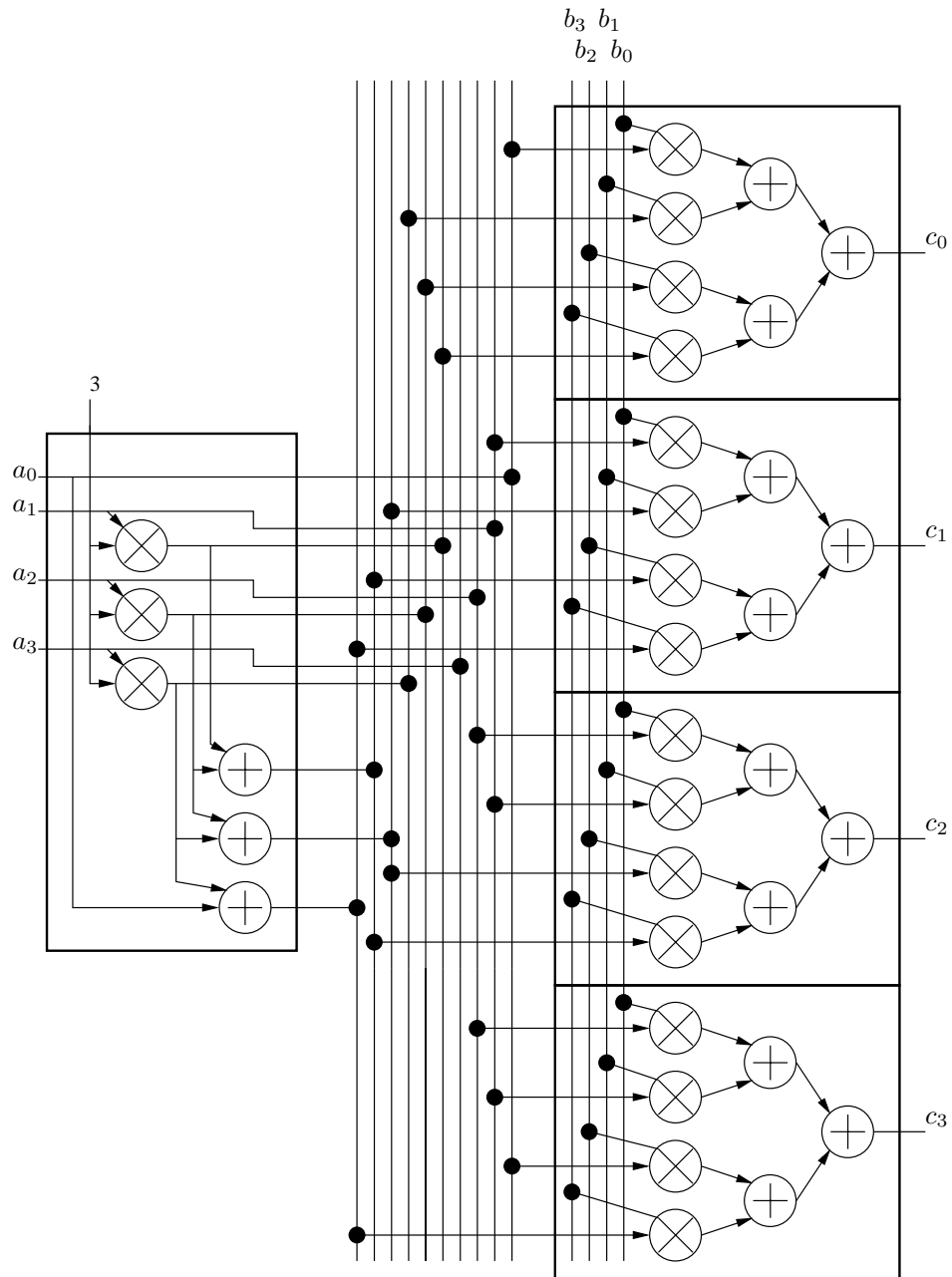


Figure 4.3. Implementation of parallel polynomial multiplier for $GR(4^4)$, with $P(x) = x^4 + x + 1$.

4.2.3 Performance of the parallel multiplier

It is hard, to not say impossible, to explicitly state the number of gates needed and the critical path length for a parallel multiplier, given the generator polynomial. We will state upper bounds on the complexity, bounds that can often be beaten by quite a lot. Therefore we will also discuss some specific classes of generator polynomials that will show better performances.

First we look at the right part of figure 4.3. We see that this only depends on the size of the Galois ring, and not on the generator polynomial. The depth is 1 multiplication, and $\lceil \log_2 m \rceil$ additions. This gives a total depth of $2 + 2\lceil \log_2 m \rceil$ gates for the right part of the circuit. The number of gates in each cell is m multipliers, and $m-1$ adders, totaling to $8m-4$ gates per cell, which gives $8m^2-4m$ gates for the whole right part, since we have m cells.

The left part is a bit more tricky, since it depends on the generator polynomial used. However, we know that it consists of constant multiplications followed by additions. Since all constant multiplications except negations are free in terms of gates, we assume that negation is needed for any of the coefficients, which will add 1 gate to the length of the critical path. Furthermore, the largest possible depth of the additions is the same as for the right part of the circuit, $2\lceil \log_2 m \rceil$. Summing up the depths we get a critical path of at most $3 + 4\lceil \log_2 m \rceil$ gates for the parallel multiplier. In [6] an upper limit of the number of gates needed for the left part when multiplying in fields is given. The result is valid for rings also, but we must adjust it, bearing in mind that we work in Z_4 instead of Z_2 and that we may have to negate elements, which costs us an extra gate. Therefore, from corollary 4.8 in [6] with adjustments, we get an upper bound of $5(m-1)(w_p-2)$, where w_p is the number of non-zero coefficients in the generator polynomial. We see that when $w_p = m+1$, its maximum, we get an upper bound of $5(m-1)^2$ gates. For the whole circuit, this means that the number of gates needed is less than $8m^2 - 4m + 5(m-1)(w_p-2) \leq 8m^2 - 4m + 5(m-1)^2 = 13m^2 - 14m + 5$. As far as the throughput and delay is concerned, since there are no registers, we will get one result each clock cycle, and when applying input data we will get the output the next clock cycle.

Performance for specific polynomials

In [6] the performance for different classes of generator polynomials for field multipliers is explored. Above we have used a formula for the number of gates needed, depending on the number of coefficients in the generator polynomial. Now we will take a look at the results regarding the critical path length of the left part of the multiplier in figure 4.3. In [6] results for a few different classes of polynomials are shown, and the proofs for their respective critical paths hold in rings also. We must, however, still keep in mind that we need more gates for the operations of addition and multiplication than in the case of a Galois field $GF(2^k)$, and that we also might have negations of all coefficients. This said, we see that using the results from [6] we get the following results for the left part of the multiplier.

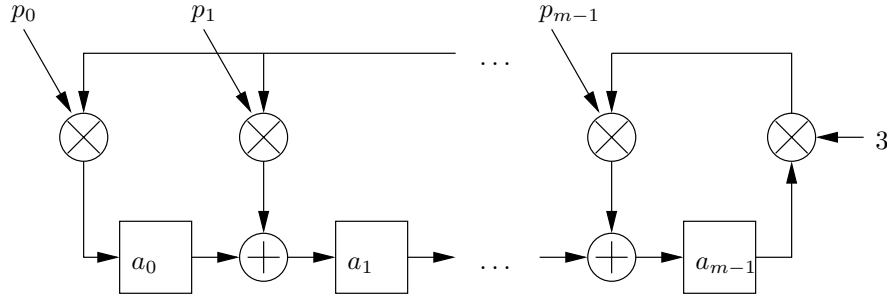


Figure 4.4. Shift register used for calculating Z .

- If the generator polynomial is $x^m + ax + b$, $a, b \neq 0$, the critical path will be at most 3 gates. We note that in section 4.2.2 we have seen that no negations are needed if the polynomial is $x^m + 3x + 3$, so if this is the case the critical path will become at most 2 gates.
- If the generator polynomial is of the form $x^m + ax^k + b$, $a, b \neq 0$ and $0 < k < m/2$, the critical path will be at most 5 gates. Also here the best is if $a = b = 3$, because then no negations are needed so the critical path will become 4 gates.
- If the generator polynomial is a polynomial of the form

$$x^{ns} + x^{(n-1)s} + \dots + x^s + 1,$$

for any integer s , the critical path will be at most 3 gates.

For the right part of the multiplier we have already seen that the critical path is $2 + 2\lceil \log_2 m \rceil$ gates, so to get the full critical path we only have to add this to the results above. As we have stated these results are proven in [6], but we will show an alternate way of justifying them here. This method is used in the first of the cases in [6], but here we will extend it to be used for all cases, even though we only prove the third statement, the one with $x^m + x^{m-1} + \dots + x + 1$. First we remember from section 4.1.2 that the columns in the multiplication matrix Z can be calculated by rotating the columns and reducing modulo the generator polynomial. Bearing in mind that the left-most column contains a_0, a_1, \dots, a_{m-1} , we see that we can use the shift-register in figure 4.4 to calculate the columns one after another, by loading it with a_0, a_1, \dots, a_{m-1} , and then shifting the data in the registers $m - 1$ times. Each shift will give us a new column in Z . We note that this figure is equivalent to the upper part of the MSR multiplier, as shown in figure 4.2.

We now wish to compute the columns of the matrix for the polynomial $x^m + x^{m-1} + \dots + x + 1$. Obviously, we can not perform this computation entirely, because we don't know the size of m , but the first few columns are calculated in the table 4.1 (the columns of Z are shown as rows in the table).

R_0	R_1	R_2	\dots	R_{m-1}
a_0	a_1	a_2	\dots	a_{m-1}
$3a_{m-1}$	$a_0 + 3a_{m-1}$	$a_1 + 3a_{m-1}$	\dots	$a_{m-2} + 3a_{m-1}$
$a_{m-1} + 3a_{m-2}$	$3a_{m-2}$	$a_0 + 3a_{m-2}$	\dots	$a_{m-3} + 3a_{m-2}$
$a_{m-2} + 3a_{m-3}$	$a_{m-1} + 3a_{m-3}$	$3a_{m-3}$	\dots	$a_{m-4} + 3a_{m-3}$
\vdots	\vdots	\vdots	\dots	\vdots
$a_2 + 3a_1$	$a_3 + 3a_1$	$a_4 + 3a_1$	\dots	$a_0 + 3a_1$

Table 4.1. The columns in the Z matrix for the generator polynomial $x^m + x^{m-1} + \dots + x + 1$.

After the first few lines we discover a pattern and may thus conclude how all columns in Z will look. We see that no element in Z will need more than one addition and one negation (or one subtraction instead of both), and thus the maximum depth will be 3 gates, just as we have stated. In the same way we may make tables for the other polynomials for which we have stated good critical path lengths and see that it's correct. Or, as we have said earlier, we may rely on the proofs in [6].

4.3 Implementation of systolic multipliers

Another class of architectures for multiplying in Galois fields are systolic architectures. Their advantages include highly regular structures and that they are quite fast. The systolic multiplier for Galois fields, as described in [10] can easily be adapted to Galois rings.

4.3.1 General principles of systolic architectures

The principle behind systolic architectures appears to be quite simple and easy to understand. A *systolic array* comprise an array of identical cells, performing some kind of operation. The cells are put together in such a way that each cell only uses signals from the cells next to it. By introducing flip-flops at well chosen points (for example at all points where signals go from one cell to another) we can now get very short signal paths. This means that the architecture may be clocked very fast. The data which to process is normally introduced at the top and left side of the array. The systolic arrays we will look at would without flip-flops be strictly parallel architectures, but the adding of flip-flops gives them a certain serial flavour. Since data only flows from a cell to it's neighbours, and no feedback circuits are allowed, after a few clock cycles we will normally have cells that are no longer used in the computations. They can be used for beginning the next computation. This means that with a good design of the systolic array all cells may be in work all the time, thus maximizing the throughput.

The downside of systolic arrays are that they will soon become very large. The number of cells is normally (as it will be in our case) in the order of m^2 . This demands a very large chip area. Another negative thing about systolic multipliers is that they often require a larger number of clockcycles before they are done than the serial architectures.

4.3.2 Implementation for $GR(4^m)$

We have seen in section 4.1.2 that

$$c(x) = b_0a(x) + b_1xa(x) + \dots + b_{m-1}x^{m-1}a(x) \pmod{p(x)}.$$

Interchanging $a(x)$ and $b(x)$, which is possible since multiplication is commutative, we might just as well write

$$c(x) = a_0b(x) + a_1xb(x) + \dots + a_{m-1}x^{m-1}b(x) \pmod{p(x)},$$

which may be described as adding up the different terms, and then reducing modulo $p(x)$. In equation 4.3 it is shown that the different terms may be computed by succesively adding the terms without multiplying by x , and after each new term left-shift one step, which is equivalent to multiplying by x . This can be done recursively, and we may reduce modulo $p(x)$ in each step, instead of at the end like in the equation. This gives us an algorithm for computing $c(x)$, as is shown below.

$$\begin{aligned} R^{(0)}(x) &= 0 ; \\ \text{for } i &= 1 \text{ to } m \\ &R^{(i)}(x) = (R^{(i-1)}(x)x + a_{m-i}b(x)) \pmod{p(x)} \\ \text{end} \\ c(x) &= R^{(m)}(x). \end{aligned}$$

To transform this into an algorithm working on the coefficients instead of the full polynomial $b(x)$ we observe that we may write

$$\begin{aligned} R^{(i-1)}(x)x \pmod{p(x)} &= \sum_{k=0}^{m-1} r_k^{(i-1)} x^{k+1} \\ &= r_{m-1}^{(i-1)} x^m + \sum_{k=0}^{m-2} r_k^{(i-1)} x^{k+1} \\ &= r_{m-1}^{(i-1)} \sum_{j=0}^{m-1} (-p_j x^j) + \sum_{j=1}^{m-1} r_{j-1}^{(i-1)} x^j \\ &= \sum_{j=0}^{m-1} (r_{j-1}^{(i-1)} - r_{m-1}^{(i-1)} p_j) x^j, \end{aligned}$$

as long as we let $r_{-1}^{(i-1)} = 0$.

Now we may rewrite our algorithm above as follows:

```

 $R^{(0)}(x) = 0 ; \quad r_{-1}^{(k)} = 0 \quad \forall k ;$ 
for  $i = 1$  to  $m$ 
   $R^{(i)}(x) = \sum_{j=0}^{m-1} (r_{j-1}^{(i-1)} - r_{m-1}^{(i-1)} p_j + a_{m-i} b_j) x^j$ 
end
 $c(x) = R^{(m)}(x).$ 

```

Now we can let the sum in the for loop be represented by a row in the systolic array, and each cell in a row corresponds to one term in the sum.

Using another description of $R^{(i)}(x)$, namely

$$R^{(i)}(x) = \sum_{j=0}^{m-1} r_j^{(i)} x^j,$$

we easily see that

$$r_j^{(i)} = (r_{j-1}^{(i-1)} - r_{m-1}^{(i-1)} p_j + a_{m-i} b_j).$$

This shows that $r_j^{(i)}$ depends on $r_{m-1}^{(i-1)}$. This is the most significant coefficient of the upper row, and because of this we calculate the most significant coefficients first (i.e. higher and to the left of the less significant). Each cell in our array will therefore depend on the leftmost cell on the row above. We also see that each cell will depend on the cell to its upper right ($r_{j-1}^{(i-1)}$), and on the cell to its left and the cell above it (since the b and a coefficients are introduced at the top respectively at the left). This, together with the equation 4.3.2 gives us the cells in figure 4.5. To interconnect the cells we use the systolic array in figure 4.6. We haven't yet discussed how many flip-flops should be introduced. Between all cells we need at least one flip-flop, but this is not enough. Since every cell depends both on the cell above it and on the cell on its upper right, we need two flip-flops between the cells vertically. This is because to calculate $r_j^{(i)}$ we need $r_{j-1}^{(i-1)}$. Since this needs data from the cell to the left of it we need a two clock-cycles gap between each cell and the cell above it.

We also see that to make sure that all coefficients meet each other in the right cells at the right times we must delay the inputs. Since it takes the b coefficients two clock cycles to "fall down" one row in our matrix, the inputs at the left must be delayed two clock cycles in the second row, four in the third row and so on. Since we only have one flip-flop between each column in our matrix, we only need to delay the data at the upper row one clock cycle in the second column, two in the third column and so on. This delay of the inputs at the top also means that we will have the same delays of the second, third and so on coefficient at the bottom. In figure 4.6 the notation $a_i^{(j)}$ signifies that the coefficient a_i is delayed j clock cycles.

4.3.3 Performance of the systolic multiplier

The systolic multiplier contains m^2 equal cells. The longest path a signal has to go through in such a cell contains one multiplication, one subtraction and one addition.

Chapter 3 tells us that this translates into 6 gates in the critical path (remember that a subtraction preceded by a multiplication only contributes with two gates to the critical path). We may apply new data each clock cycle at the inputs, and for each clockcycle we will get a new calculated result, thus giving a throughput of 1 result per clock cycle. The delay will be $3m - 1$ clock cycles, of which $2m$ is the delay between the cells, and $m - 1$ is the extra delay for not applying all inputs at the same time, but delaying some before they can enter the array. The total number of operations in one cell is two multiplications, one subtraction and one addition for a total of 13 gates. The whole array then contains $13m^2$ gates, plus $6m^2$ registers, or flip-flops.

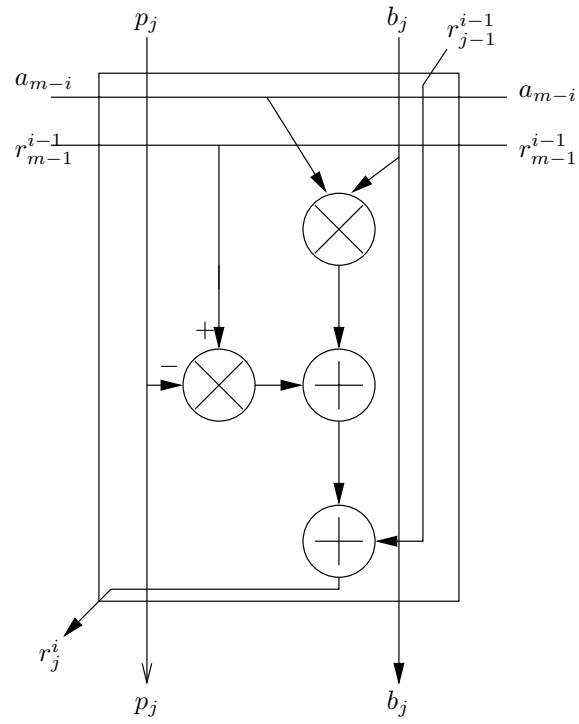


Figure 4.5. A cell in the systolic multiplier.

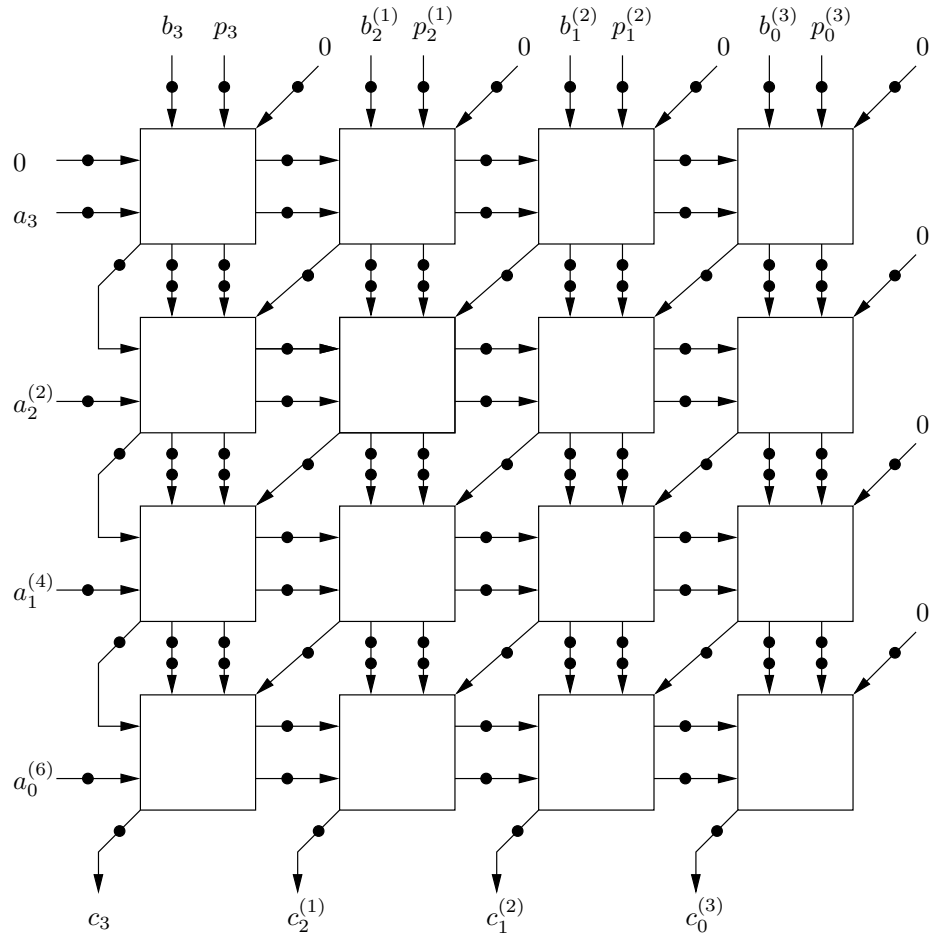


Figure 4.6. Implementation of a systolic multiplier for the $GR(4^4)$. The filled circles between the cells are flip-flop registers.

4.4 Summary of performances

In the tables below we summarize the speed and the area needed for the four architectures explored in this chapter. For the parallel multiplier a few comments are necessary. First, when computing the area w_p equals the number of non-zero coefficients in the generator polynomial. For the critical path we give several different values. One is an upper limit for any generator polynomial, and the others are values for specific generator polynomials, as shown in 4.2.3.

Architecture	Area (in gates)	Registers
SSR serial	$17m$	$5m$
MSR serial	$17m - 4$	$5m$
Parallel	$\leq 8m^2 - 4m + 5(m - 1)(w_p - 2)$	0
Systolic	$13m^2$	$6m^2$

Table 4.2. Area of polynomial basis architectures.

Architecture	Critical path	Delay	Throughput
SSR serial	6	$2m$	$1/m$
MSR serial	5	$2m$	$1/m$
Parallel	$\leq 3 + 4\lceil \log_2 m \rceil$	1	1
Parallel, $x^m + ax + b$	$5 + 2\lceil \log_2 m \rceil$	1	1
Parallel, $x^{sn} + x^{s(n-1)} + \dots + 1$	$5 + 2\lceil \log_2 m \rceil$	1	1
Parallel, $x^m + ax^k + b, k \leq m/2$	$7 + 2\lceil \log_2 m \rceil$	1	1
Systolic	6	$3m - 1$	1

Table 4.3. Speed of polynomial basis architectures.

Chapter 5

Dual basis representation

In this chapter we will explore a representation called the dual basis representation for multiplying elements in $GR(4^m)$. We begin by defining what we mean by a dual basis, and prove the existence of such a basis for all Galois Rings. This theory is much inspired by [5] and [11].

5.1 Definition and existence

We start by defining what we mean when we say that two bases are dual.

Definition 5.1 (Dual basis) *A pair of bases $\{\alpha_0, \dots, \alpha_{m-1}\}$ and $\{\beta_0, \dots, \beta_{m-1}\}$ are called dual bases if and only if*

$$T(\alpha_i \beta_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, 0 \leq i, j \leq m-1$$

where T is the trace function from definition 2.10.

It is important to notice that a basis by itself never can be a dual basis, it must be dual with respect to another basis. At times we will, however, state that a basis is a dual basis, omitting to which basis it is dual. This is only done when the other basis is obvious from the context, and this other basis will normally be a standard polynomial basis. Now we will state a theorem that guarantees a dual basis for any basis in the Galois Ring $GR(4^m)$.

Theorem 5.1 (Existence of unique dual basis) *Every basis of $GR(4^m)$ has a unique dual basis.*

When proving this theorem we will need the following lemma.

Lemma 5.1 *All linear transformations from $GR(4^m)$ to Z_4 may be written uniquely as $L_\gamma(\alpha) = T(\gamma\alpha)$ for different values of $\gamma \in GR(4^m)$.*

Proof. Since the trace function T is linear (according to theorem 2.8), L_γ is a linear transformation from $GR(4^m)$ to Z_4 . For $\gamma_1 \neq \gamma_2$ we have that $L_{\gamma_1}(\alpha) - L_{\gamma_2}(\alpha) = T((\gamma_1 - \gamma_2)\alpha)$. Write $\gamma_1 - \gamma_2 = a + 2b$ in 2-adic representation. If $a \neq 0$ we know that $a + 2b$ is invertible according to theorem 2.6. Therefore we may choose an α such that $(\gamma_1 - \gamma_2)\alpha$ obtains any value in $GR(4^m)$. Since T is surjective onto Z_4 we can choose α such that $T((\gamma_1 - \gamma_2)\alpha) \neq 0$. If, on the other hand, $a = 0$ we have $T((\gamma_1 - \gamma_2)\alpha) = 2T(b\alpha) \neq 0$ for some α since b is invertible. From this we may conclude that all transformations L_γ are different, and if their number equals the total number of linear functions from $GR(4^m)$ to Z_4 they actually form the set of all such functions.

The number of linear functions from $GR(4^m)$ can be obtained by considering that a linear function is formed by assigning a value in Z_4 to every basis element of a certain basis in $GR(4^m)$. Since we have m elements in any basis, and each may map to 4 different values in Z_4 , there are 4^m possible linear functions from $GR(4^m)$ to Z_4 . Since we also have 4^m different linear functions $L_\gamma(\alpha)$, one for each element $\gamma \in GR(4^m)$, the lemma is proven. \square

Now we turn to the proof of theorem 5.1.

Proof (Existence of unique dual basis) Assume that we have a basis for $GR(4^m)$, $\{\alpha_0, \dots, \alpha_{m-1}\}$, and that ξ is a root of a primitive polynomial of degree m , with the order of ξ being $2^m - 1$. Let further, for any element $\alpha \in GR(4^m)$,

$$\alpha = \sum_{i=0}^{m-1} c_i(\alpha) \alpha_i \quad (5.1)$$

be the unique representation of α in the basis, where the $c_i(\alpha)$:s are m linear functions from $GR(4^m)$ to Z_4 . We wish to show that we can write $c_i(\alpha) = T(\beta_i \alpha)$, for all these functions, and that $\{\beta_0, \dots, \beta_{m-1}\}$ forms a basis for $GR(4^m)$.

All c_i in equation 5.1 are linear transformations from $GR(4^m)$ to Z_4 . Now lemma 5.1 tells us that for each c_i there exists a β_i such that we have $c_i(\alpha) = T(\beta_i \alpha)$, for all α .

To prove our assumption we must also show that the set $\{\beta_0, \dots, \beta_{m-1}\}$ is a basis of $GR(4^m)$, dual to $\{\alpha_0, \dots, \alpha_{m-1}\}$. Since we have

$$\alpha_j = \sum_{i=0}^{m-1} c_i(\alpha_j) \alpha_i = \sum_{i=0}^{m-1} T(\beta_i \alpha_j) \alpha_i = \sum_{i=0}^{m-1} L_{\beta_i}(\alpha_j) \alpha_i,$$

we know that $T(\beta_i \alpha_j) = 0$ if $i \neq j$ and 1 if $i = j$. Since we know how the L_{β_i} :s work on all basis elements of the original basis they are fully determined, and from lemma 5.1 we draw the conclusion that the β_i :s are uniquely determined. Furthermore,

if $\sum_i d_i \beta_i = 0$, where $d_i \in Z_4$, we have

$$\begin{aligned} \left(\sum_{i=0}^{m-1} d_i \beta_i \right) \alpha_j &= 0 \Leftrightarrow \\ T \left(\sum_{i=0}^{m-1} d_i \beta_i \alpha_j \right) &= 0 \Leftrightarrow \\ \sum_{i=0}^{m-1} d_i T(\beta_i \alpha_j) &= 0 \Leftrightarrow \\ d_j &= 0 \end{aligned}$$

for all $j = 0, \dots, m-1$. Therefore the β_i :s are linearly independent and we conclude that $\{\beta_0, \dots, \beta_{m-1}\}$ is a basis for $GR(4^m)$, and also the only basis dual to the basis $\{\alpha_0, \dots, \alpha_{m-1}\}$. \square

We give a few examples of dual bases.

Example 1. Let $R = Z_4[x]/(x^4+x+1)$, and let $\alpha^4+\alpha+1=0$, so that $\{1, \alpha, \alpha^2, \alpha^3\}$ is the standard polynomial basis of R . The dual basis is $\{\alpha^3+1, \alpha^2, \alpha, 1\}$. This also means that if the coefficients in the first basis for an element are (a_0, a_1, a_2, a_3) they will be $(a_3, a_2, a_1, a_0 - a_3)$ in the second basis.

Let also $S = Z_4[x]/(x^4+3x+3)$, and $\beta^4+3\beta+3=0$. Now we have the pair of dual bases $\{1, \beta, \beta^2, \beta^3\}$ and $\{3\beta^3+1, 3\beta^2, 3\beta, 3\}$. Also we have for an element in the first basis with coefficients (b_0, b_1, b_2, b_3) the coefficients $(3b_3, 3a_2, 3a_1, 3a_0+a_3)$ in the second basis.

As a third example, let $T = Z_4[x]/(x^3+2x^2+x+3)$, and $\gamma^3+2\gamma^2+\gamma+3=0$. Now the basis and dual basis are $\{1, \gamma, \gamma^2\}$ and $\{2\gamma^2+2\gamma+3, 2\gamma^2+3\gamma+1, 2\gamma^2+\gamma+2\}$. If the coefficients for an element are (c_0, c_1, c_2) in the first basis, they are $(3c_0+2c_1+2c_2, 2c_0+2c_1+c_2, 2c_0+c_1+2c_2)$ in the second basis.

We note that conversion between the dual bases may be very simple, as in the first example, or more complicated, as in the third. In the first example we only have to change the order of the elements, and add or subtract a constant 1 to one of them, while as for the last example it is much more difficult. \square

Now we will use this result to construct multipliers that multiplies two elements, one represented in a polynomial basis and the other in its dual basis. The multiplier we present first is similar to the serial dual-basis multiplier in [6].

5.2 Implementation of serial multipliers

Assume that $\{1, \alpha^1, \dots, \alpha^{m-1}\}$ and $\{\beta_0, \beta_1, \dots, \beta_{m-1}\}$ are dual bases for $GR(4^m)$, and that α is a root of the basic irreducible polynomial

$$p(x) = p_m + p_{m-1}x^{m-1} + \dots + p_1x + p_0$$

. The first basis is hence a standard polynomial basis. We know that

$$T(\alpha_i \beta_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} . \quad (5.2)$$

Assume further that we wish to multiply the two elements A and B , where

$$\begin{aligned} A &= \sum_{i=0}^{m-1} a_i \alpha^i \\ B &= \sum_{i=0}^{m-1} b_i \beta_i, \end{aligned}$$

and $a_i, b_i \in Z_4$. We denote the result

$$C = AB = \sum_{i=0}^{m-1} c_i \beta_i,$$

where $c_i \in Z_4$. Note that A is in polynomial basis representation while B and C are in dual basis representation. We have

$$T(\alpha^j B) = T\left(\alpha^j \sum_{i=0}^{m-1} b_i \beta_i\right) = \sum_{i=0}^{m-1} b_i T(\beta_i \alpha^j) = b_j, \quad (5.3)$$

due to the linearity of the trace function, and equation 5.2. Let $Y = \alpha B$. Then equation 5.3 gives us

$$y_j = T(\alpha^j Y) = T(\alpha^{j+1} B) = \begin{cases} b_{j+1}, & j = 0, 1, \dots, m-2 \\ T(\alpha^m B), & j = m-1 \end{cases} \quad (5.4)$$

Since $p(\alpha) = 0$, and $p(x)$ is monic, we know that

$$\alpha^m = -(p_0 \alpha^0 + p_1 \alpha + \dots + p_{m-1} \alpha^{m-1}) \quad (5.5)$$

and, combining this with 5.3, we get

$$\begin{aligned} T(\alpha^m B) &= T(-[p_0 \alpha^0 + p_1 \alpha + \dots + p_{m-1} \alpha^{m-1}] B) \\ &= -[p_0 T(B) + p_1 T(\alpha B) + \dots + p_{m-1} T(\alpha^{m-1} B)] \\ &= -(p_0 b_0 + p_1 b_1 + \dots + p_{m-1} b_{m-1}) \\ &= -(p_0, p_1, \dots, p_{m-1}) \cdot (b_0, b_1, \dots, b_{m-1}) \end{aligned} \quad (5.6)$$

where \cdot is the scalar product between vectors. We see from equations 5.4 and 5.6 that we can implement a multiplication by α in the dual basis by a shift of coefficients, letting each coefficient take the value of the former next higher coefficient.

The exception to this is that the new highest coefficient will take the value of the equation 5.6. Let

$$\alpha^j B = \sum_{i=0}^{m-1} b_{i+j} \beta^i.$$

Obviously this is valid for $j = 0$, since we then have

$$\alpha^0 B = \sum_{i=0}^{m-1} b_{i+0} \beta^i.$$

This gives us the following recursive formula for $\alpha^{j+1} B$:

$$\alpha^{j+1} B = \sum_{i=0}^{m-2} b_{i+j+1} \beta^i - \left(\sum_{i=0}^{m-1} b_{i+j} p_i \right) \beta^{m-1}. \quad (5.7)$$

The calculation of αB is shown in the upper part of figure 5.1.

Now we can compute the coefficients in C .

$$c_j = T(\alpha^j C) = T(\alpha^j AB), \quad j = 0, 1, \dots, m-1$$

We compute the first coefficients as examples.

$$\begin{aligned} c_0 &= T(AB) = T(a_0 B) + T(a_1 \alpha B) + \dots + T(a_{m-1} \alpha^{m-1} B) \\ &= a_0 b_0 + a_1 b_1 + \dots + a_{m-1} b_{m-1} \end{aligned} \quad (5.8)$$

$$= (a_0, a_1, \dots, a_{m-1}) \cdot (b_0, b_1, \dots, b_{m-1}), \quad (5.9)$$

where \cdot still denotes the scalar product of vectors. Now c_1 may be computed as the scalar multiplication of A and αB , $c_2 = A \cdot (\alpha^2 B)$ and so on. Hence the multiplication may be divided in two parts, one recursively calculating $\alpha(\alpha^{i-1} B)$, and the other performing a scalar product with A .

$$\begin{aligned} c_1 &= T(\alpha AB) = T(a_0 \alpha B) + T(a_1 \alpha^2 B) + \dots + T(a_{m-1} \alpha^m B) \\ &= a_1 b_1 + a_1 b_2 + \dots + a_{m-2} b_{m-1} - a_{m-1} (b_0 p_0 + b_1 p_1 + \dots + b_{m-1} p_{m-1}) \end{aligned}$$

Note that C is given in dual basis representation.

These calculations gives us a implementation of the dual basis serial multiplier as shown in figure 5.1. In this figure we can see that the upper part is used to calculate αB as described in equations 5.4 and 5.6, while in the lower part we perform the scalar product from equation 5.8. The output is given serially, starting with c_0 .

5.2.1 Alternative serial multiplier

Looking at figure 5.1 we see that the critical path is very long, and depends on m , which is not a very good property of the multiplier. Inspired by [9], more

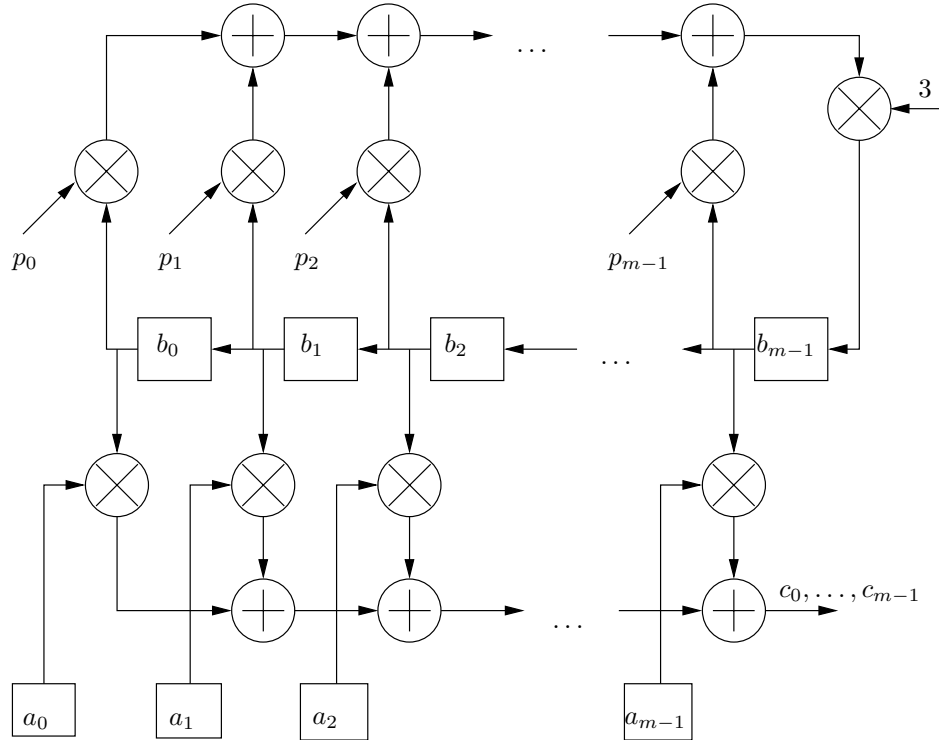


Figure 5.1. Implementation of the serial dual basis multiplier for $GR(4^m)$.

specifically the transformation of figure 4.2 into figure 4.7 in that thesis, we see that by moving around some of the registers we could instead use the architecture in figure 5.2 as our serial multiplier. Some explanations about how the input of data is done is needed. As we see from the figure the registers in the middle are not sufficient to store all b_i , but only half of them. Therefore we introduce our b_i :s serially the first m clock cycles, and after this we turn the switch in the figure and close the feedback circuit, which has now calculated the very first result that is to be fed back. This is immediately used by the lower part that has added all but the last term to obtain c_0 . Thereafter we continue operation for $m - 1$ clock cycles, calculating the rest of the c_i :s.

5.2.2 Performance of the serial multipliers

We see from figure 5.1 that the critical path depends on the size of the Galois ring, as opposed to the serial architecture for the polynomial basis. The addition networks in the figure are not optimal, we should of course use addition trees for adding m values. The critical path will then contain one multiplication, $\lceil \log_2(m - 1) \rceil$ ad-

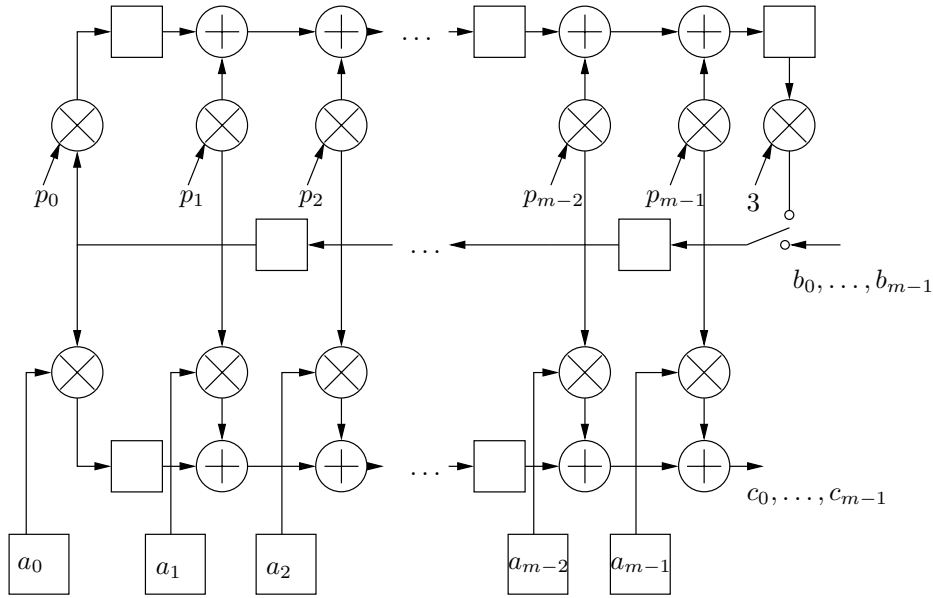


Figure 5.2. Implementation of the alternative serial dual basis multiplier for $GR(4^m)$.

ditions and one negation, bringing the total length to $3 + \lceil \log_2(m-1) \rceil$ gates, which is horrible compared to the 5 gates needed for the MSR multiplier in section 4.1.2. The total number of operations needed is $2m$ multiplications, $2m-2$ additions and 1 negation, which gives an area of $16m-7$ gates. Furthermore $4m$ registers are needed ($2m$ registers are necessary to be able to load the b_i and a_i registers in a parallel fashion). We see that we can perform one multiplication each m clock cycles, for a throughput of $1/m$. The delay is $2m$, just as in the standard polynomial basis case.

The alternative multiplier in figure 5.2 has much better performance in some ways. First we see that the critical path contains just one multiplication, and two additions, for a total of 6 gates. The number of gates needed is the same as above, $16m-7$. Apart from the registers in the figure we need m registers to serially input the a_i 's, and $m/2$ registers to input the b_i 's (since they can also use the $m/2$ registers in the middle of the figure). This totals to $4m$ registers. We can here only perform one multiplication every $2m$ clock cycles, due to the fact that we cannot start applying new data until we're completely done with the previous ones, so therefore the throughput is only $1/2m$ results per clock cycle. The delay is still $2m$ clock cycles.

5.3 Implementation of systolic multipliers

In this section we will look at how a systolic dual basis multiplier may be constructed. For a brief introduction into systolic array architectures the reader is referred to section 4.3.1.

In [2] a parallel, systolic, dual basis multiplier is proposed for the Galois field $GF(2^n)$. We will show that a similar multiplier for the Galois ring $GR(4^m)$ can be constructed. Bearing in mind the dual basis multiplier in section 5.2, and using the same denotation for the multiplicands, we know that

$$c_j = T(\alpha^j AB) = A \cdot (\alpha^j B),$$

according to the discussion after equation 5.8. We have also shown a recursive formula for calculating $\alpha^j B$ in the equation 5.7. We see now that we can write the multiplication in the following matrix form.

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} b_0 & b_1 & \dots & b_{m-1} \\ b_1 & b_2 & \dots & b_m \\ \vdots & \vdots & \ddots & \vdots \\ b_{m-1} & b_m & \dots & b_{2m-2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{pmatrix}. \quad (5.10)$$

This can be used to implement the multiplication with a systolic array. Let us input the A coefficients at the left, and the B coefficients at the top, and expect the C coefficients to turn out as the result at the bottom of the array. We see from the array above that the B coefficients must go from one cell to the left lower cell, and that we need to input the negative cross product with the P coefficients in the right-most cell of each column. Therefore we need to calculate this cross-product at each row, to be able to input it at the row below. This is done by letting the P coefficients be inputted in the same way as the A coefficients, at the left. The full systolic array is shown in figures 5.3 and 5.4.

As we see two flip-flops are used between the cells vertically, but only one flip-flop is used horizontally. This is because horizontally the cells only depends directly on the cell to the left of them, while as vertically they depend on the cell to their upper right, and on the cell above them. Since their upper right cell also depends on the cell above them, it takes two clock cycles from the moment the above cell has its correct value until it can be used. Alternatively we might say that the data from the upper cell first has to travel to the right, and then down to the left, thus needing two clock cycles. This also demands the input coefficients to be delayed in time, which is shown by the notation $a_i^{(j)}$, meaning coefficient a_i should be delayed j clock cycles.

5.3.1 Performance of systolic multiplier

As we see in figure 5.4 the critical path of the dual basis systolic multiplier consists of one multiplication and one subtraction. Since the subtraction isn't preceded by multiplications, it needs a depth of 3 gates, and the multiplication needs 2 gates,

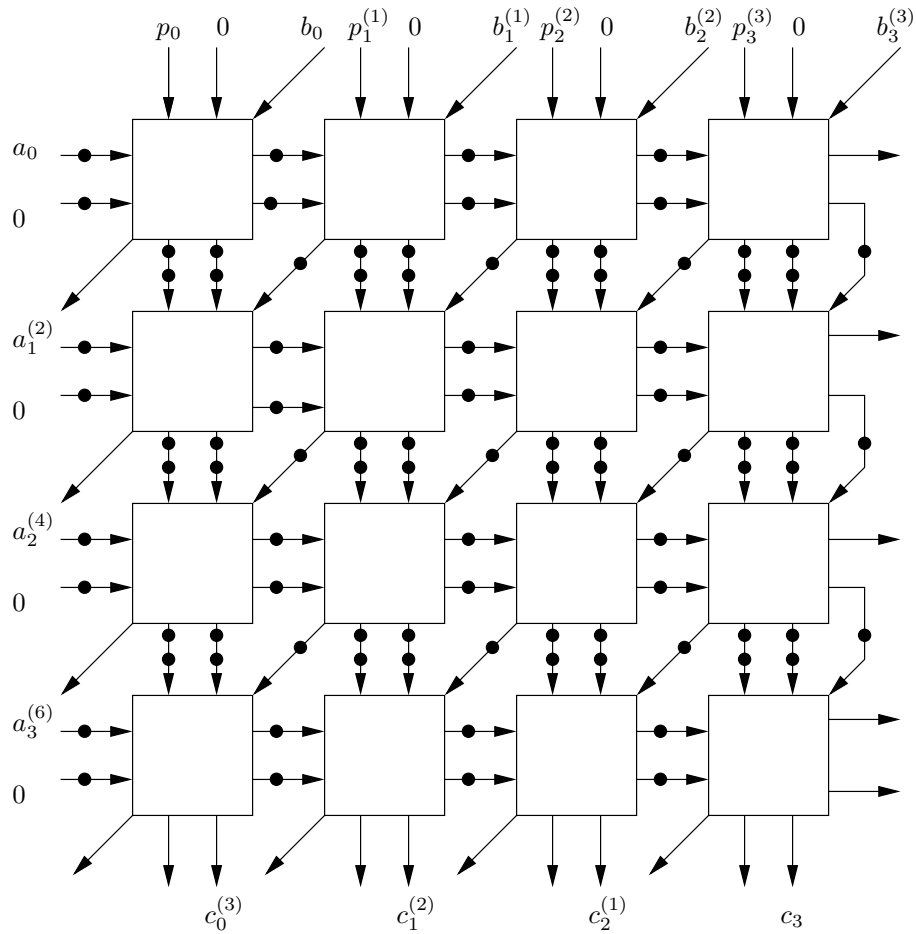


Figure 5.3. Implementation of a systolic dual basis multiplier for the $GR(4^4)$. The filled circles between the cells are flip-flop registers.

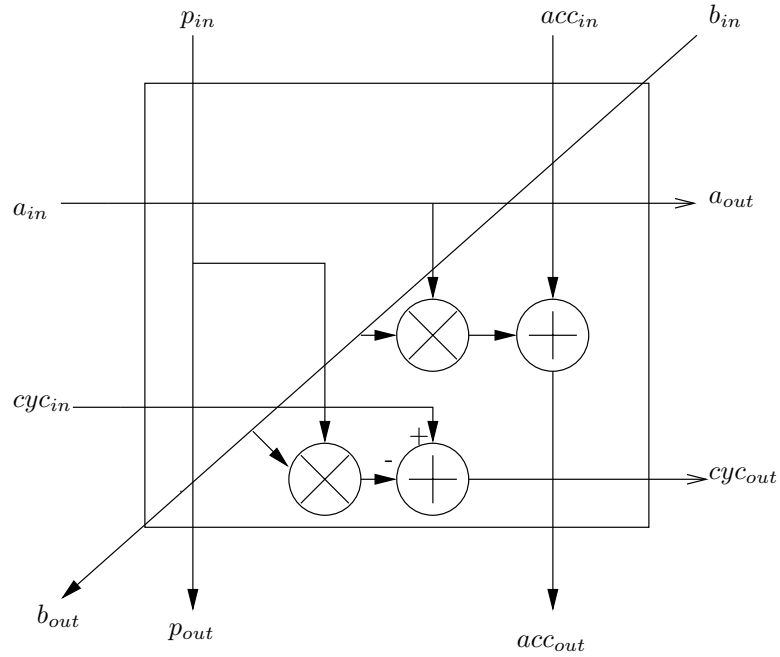


Figure 5.4. A cell in the systolic dual basis multiplier.

which means the depth of the critical path is 5 gates. The delay and throughput are the same as for the standard polynomial basis multiplier, $3m - 1$ clock cycles and 1 result per clock cycle. The total number of operations in a cell is two multiplications, one subtraction and one addition, for a total number of 17. For the whole array this implies that $17m^2$ gates are necessary. Also, we will need $6m^2$ flip-flops.

5.4 Summary of performances

Here we summarize the performances of the architectures presented. In the tables below their speed and area are shown. Note that the numbers given do not include the structures for performing basis conversions, rather it is assumed that all data exist in the needed bases. For some choices of basis this could be a significant part of the circuit, and therefore must be taken into account when comparing performances.

Architecture	Area (in gates)	Registers
Serial	$16m - 7$	$4m$
Alternative serial	$16m - 7$	$4m$
Systolic	$13m^2$	$6m^2$

Table 5.1. Area of dual basis architectures.

Architecture	Critical path	Delay	Throughput
Serial	$3 + \lceil \log_2(m - 1) \rceil$	$2m$	$1/m$
Alternative serial	6	$2m$	$1/2m$
Systolic	5	$3m - 1$	1

Table 5.2. Speed of dual basis architectures.

Chapter 6

Normal basis multipliers

In this chapter we will discuss another possible basis for a Galois Ring, which we will call a normal basis. The definition (and many of the results) is analogue to the definition of a normal basis for a Galois Field, as described in for example [1], [6] and [9]. The theoretical discussions here owes most to chapter 5 in [1], while the presentation of the implementation is inspired by [6]. We begin by defining and justifying the concept of the normal basis.

6.1 Definition of normal basis

Definition 6.1 (Normal basis) *A normal basis over $GR(4^m)$ is a basis of the form $\{\alpha, \alpha^f, \alpha^{f^2}, \dots, \alpha^{f^{m-1}}\}$ where $\alpha \in GR(4^m)$, and f is the Frobenius map defined in definition 2.9.*

It is not obvious that any normal bases exist, but they do. One example of a class of normal bases will be given in theorem 6.2.

Since f is a linear function that maps the elements of Z_4 on themselves, and $c^{f^m} = c$ according to 2.8, we have the following.

$$(a_0\alpha + a_1\alpha^f + \dots + a_{m-1}\alpha^{f^{m-1}})^f = a_{m-1}\alpha + a_0\alpha^f + a_1\alpha^{f^2} + \dots + a_{m-2}\alpha^{f^{m-2}}$$

This means that applying the Frobenius map on an element expressed in a normal basis is a simple cyclic right-shift of the coefficients. Furthermore, we know that if $c = ab$ then, according to theorem 2.7, $c^f = (ab)^f = a^f b^f$, which means that if we have a function $g(a, b)$ for computing the coefficient c_{m-1} , we can apply the same function on the elements a^f, b^f to obtain the coefficient c_{m-2} . This can be widened, so that all coefficients of c can be calculated using the same function g , by just shifting the inputs. This function can be expressed as a matrix, and instead of shifting the inputs, we may of course rotate this matrix in two dimensions, so that we get m different matrices (all containing the same elements, but not at the same positions), one for each output coefficient.

Definition 6.2 We define the multiplication matrix for coefficient k as the matrix M_k satisfying

$$c_k = (a_0 \quad \dots \quad a_{m-1}) M_k \begin{pmatrix} b_0 \\ \vdots \\ b_{m-1} \end{pmatrix} \quad (6.1)$$

The number of non-zero elements in the matrix is called the *complexity* of the normal basis, and denoted c_N , if the normal basis is denoted N . In fact this is a bit off-target, since the actual complexity of the multiplier will depend also on what values the non-zero elements will take, since a value that is not 1 will force a multiplication by a constant. However, for simplicity of the theoretical reasoning we have chosen not to bother about the actual values of the non-zero elements. In the next section we will give a limit for the complexity, as well as discuss a class of low-complexity normal bases.

6.2 Optimal normal bases

A trivial upper bound for the complexity of a normal basis is obtained by considering a $m \times m$ matrix containing no zeros. This matrix of course contains m^2 non-zero elements, which we will take as an upper bound for the complexity. A lower bound is given in the following theorem.

Theorem 6.1 For any normal basis of the Galois Ring $GR(4^m)$ we have a lower bound on the complexity, $c_N \geq 2m - 1$.

Proof. Let $\{\alpha_0, \alpha_1, \dots, \alpha_{m-1}\}$ be a normal basis for the Galois Ring $GR(4^m)$. Let further

$$\alpha_0 \alpha_i = \sum_{j=0}^{m-1} s_{ij} \alpha_j \quad (6.2)$$

where $0 \leq i \leq m-1$ and $s_{ij} \in Z_4$. Sum the left and right sides of the m equations 6.2 respectively. The left side of the equation turns out as follows.

$$\alpha_0(\alpha_0 + \alpha_1 + \dots + \alpha_{m-1}) = \alpha_0(\alpha_0 + \alpha_0^f + \dots + \alpha_0^{f^{m-1}}) = \alpha_0 T(\alpha_0) \quad (6.3)$$

Since $T(\alpha_0) \in Z_4$ we now know that

$$\sum_{i=0}^{m-1} s_{ij} = \begin{cases} T(\alpha_0), & j = 0 \\ 0, & 1 \leq j \leq m-1 \end{cases} \quad (6.4)$$

Now let $S_k = (s_{ij}^{(k)})$ denote the multiplication matrix of definition 6.2 for calculating the k th coefficient of the product, and let also $S = (s_{ij})$. This gives us

$$\alpha_0 \alpha_i = \sum_{j=0}^{m-1} s_{ij} \alpha_j = \sum_{j=0}^{m-1} s_{0i}^j \alpha_j \quad (6.5)$$

where $0 \leq i \leq m-1$. As we let i assume all values from 0 to $m-1$ we will therefore use all elements from the row with index zero in matrix S_j . We will now show that this is the same as using all elements from each row in S_0 . We know that if S_0 gives us the $(m-1)$ th coefficient of the result, we can get the $(m-2)$ th coefficient by rotating the multiplicand vectors to the right. This is the same as rotating the rows in the matrix upwards, and the columns to the left, and keeping the same multiplicands. Therefore, to get all coefficients in the result we may successively rotate the matrix left and upwards, thereby letting the elements of one row at a time from S_0 form the upper row of the matrices. Therefore, using all elements from the first row of all matrices is the same as using all elements from the matrix S_0 (or any other S_k). Hence the matrix S contains the same elements as any of the S_k 's.

Since the first column in S sums to a number between 0 and 3 the least number of non-zero elements in the column is of course 1. For the other columns, we know that they sum up to 0. Since $\alpha_0 \neq 0$ and $\{\alpha_0 \alpha_i : 0 \leq i \leq m-1\}$ is also a basis for $GR(4^m)$, S is invertible. This means each column must contain at least one non-zero element. Since the columns sum to 0, they must contain at least two elements. All together this means that the matrix S has at least $2m-1$ non-zero elements, and therefore this is also the least possible complexity of the basis. \square

For easier reading, we give a name to such a basis.

Definition 6.3 (Optimal normal basis) *A normal basis $\{\alpha, \alpha^f, \dots, \alpha^{f^{m-1}}\}$ with complexity $2m-1$ is called an optimal normal basis.*

Observe that we have not shown that every Galois Ring has such a basis. In fact we haven't even shown that there exists such a basis for any Galois ring! For that reason, we will show the existence of a class of optimal normal bases, and how to construct them.

Theorem 6.2 (Type-I Optimal Normal Bases) *Let $m+1$ be a prime larger than 2 and assume that 2 is primitive in Z_{m+1} , i.e. $2^m = 1 \pmod{m+1}$ and $2^k \neq 1 \pmod{m+1}$ for integers $0 < k < m$. Then we have the following:*

(i) *The m nonunit $(m+1)$ th roots of unity are linearly independent and form an optimal normal basis of $GR(4^m)$.*

(ii) *The optimal normal basis is $\{\alpha, \alpha^2, \dots, \alpha^{2^{m-1}}\}$, where α is a root of the polynomial $x^m + x^{m-1} + \dots + x + 1$.*

Proof. Let $\alpha \neq 1$ be a $(m+1)$ th root of 1, i.e. $\alpha^{m+1} = 1$ where $m+1$ is a prime larger than 2, in accordance to the theorem. We have

$$0 = \alpha^{m+1} - 1 = (\alpha - 1)(\alpha^m + \alpha^{m-1} + \dots + 1). \quad (6.6)$$

According to theorem 2.6 all elements that aren't multiples of 2 are invertible, and hence all zero divisors are multiples of 2. Therefore either $(\alpha - 1)$ or $(\alpha^m + \alpha^{m-1} + \dots + 1)$ must be zero, both must be divisible by 2, or else their product can't be zero.

Assume that $\alpha - 1$ is divisible by 2. Then, according to the same theorem, its 2-adic representation is $\alpha - 1 = 2b$. This means we can write $\alpha = 1 + 2b$, which is a 2-adic representation since $1, b \in \mathcal{T}$. But this would mean that $\alpha^2 = (1 + 2b)^2 = 1 + 4b + 4b^2 = 1$, and this is not possible since the order of α is $m + 1$ which is larger than 2.

Since $\alpha - 1$ isn't a multiple of 2, and not zero (because $\alpha \neq 1$) we must have $(\alpha^m + \alpha^{m-1} + \dots + 1) = 0$. Therefore α has to be a root of the polynomial $x^m + x^{m-1} + \dots + x + 1$, as is pointed out in part (ii) of the theorem.

Let the 2-adic representation of α be $a + 2b$. According to Fermat's little theorem, theorem 2.1 in this thesis,

$$2^m \equiv 1 \pmod{m + 1},$$

since $m + 1$ is prime. Another way to put it is that

$$2^m - 1 = n(m + 1),$$

for some integer n . Therefore $(m + 1)$ is a divisor of $2^m - 1$. Since $m + 1$ is the order of α theorem 2.6 now tells us that $b = 0$, which in turn guarantees that $\alpha^f = \alpha^2$.

Since α is of order $m + 1$, the m elements $\{\alpha, \alpha^2, \dots, \alpha^m\}$ will all be $(m + 1)$ th roots of unity, because we can't possibly raise any of the elements to a smaller number than $m + 1$ and get the result 1, since $(m + 1)$ is prime and hence relatively prime to $2, \dots, m$. Furthermore, 2 generates $Z_{m+1} - \{0\}$, and therefore we have (defining N)

$$N = \{\alpha, \alpha^2, \alpha^{2^2}, \dots, \alpha^{2^{m-1}}\} = \{\alpha, \alpha^2, \alpha^3, \dots, \alpha^m\}, \quad (6.7)$$

if we don't take the order of the basis elements into account. Since we have seen that $\alpha^2 = \alpha^f$, this is a normal basis if the elements in N are linearly independent. In [1] it is stated that $x^m + x^{m-1} + \dots + x + 1$ is irreducible over Z_2 when $m + 1$ is prime and 2 is primitive in Z_{m+1} . It is then also basic irreducible over Z_4 , and since α is a root to this polynomial $\{1, \alpha, \dots, \alpha^{m-1}\}$ is a basis of the Galois ring the polynomial generates. Since $\alpha^m = -(\alpha^{m-1} + \dots + \alpha + 1)$ we must have that $\{\alpha, \alpha^2, \dots, \alpha^m\}$ are also linearly independent, and hence N is a normal basis.

We see that

$$\alpha\alpha^i = \alpha^{i+1} \in N, \quad 1 \leq i < m \quad (6.8)$$

and also, according to 6.7 and the definition of the trace function

$$\alpha\alpha^m = 1 = -\alpha - \alpha^2 - \dots - \alpha^m = -T(\alpha). \quad (6.9)$$

This guarantees that the first $m - 1$ rows of the multiplication matrix contains only one non-zero element, while in the last row all elements are non-zero, and therefore we have $2m - 1$ non-zero elements in the matrix, and therefore the basis is optimal. □

In the next section we will use our knowledge of the normal basis so far to implement a serial normal basis multiplier.

6.3 Implementation of serial multipliers

We will show the implementation of a multiplier using the polynomial $p(x) = 1 + x + x^2 + x^3 + x^4$ as generator polynomial for $GR(4^4)$, and α is a root to this polynomial. As we see this polynomial satisfies the conditions in theorem 6.2, and therefore we expect the number of non-zero elements in our multiplication matrix to $2m - 1 = 7$.

We wish to calculate $C = AB$, where we have

$$\begin{aligned} A &= \sum_{i=0}^3 a_i \alpha^{2^i} \\ B &= \sum_{i=0}^3 b_i \alpha^{2^i} \\ C &= \sum_{i=0}^3 c_i \alpha^{2^i}. \end{aligned}$$

We begin by calculating a transformation matrices between the standard polynomial basis and the normal basis. We have

$$\begin{pmatrix} \alpha \\ \alpha^2 \\ \alpha^4 \\ \alpha^8 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \end{pmatrix} \quad (6.10)$$

and this also gives us polynomial basis and the normal basis. We have

$$\begin{pmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 3 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \alpha^2 \\ \alpha^4 \\ \alpha^8 \end{pmatrix}. \quad (6.11)$$

Multiplying AB , in normal basis, but with the result as a polynomial *without* reducing modulo $p(x)$ we find that

$$ab = \sum_{i=0}^3 a_i \alpha^{2^i} \cdot \sum_{j=0}^3 b_j \alpha^{2^j} = \sum_{i=0}^3 \sum_{j=0}^3 a_i b_j \alpha^{2^i + 2^j}. \quad (6.12)$$

We can further write

$$\alpha^{2^i + 2^j} = \sum_{l=0}^3 \lambda_l(i, j) \alpha^{2^l} \quad (6.13)$$

for some functions λ_l . Since c_3 is the coefficient for α^{2^3} in the result we see that

$$f(A, B) = c_3 = \sum_{i=0}^3 \sum_{j=0}^3 \lambda_3(i, j) a_i b_j. \quad (6.14)$$

In the table below we have written down the λ function values for the different i :s and j :s.

i	j	$2^i + 2^j$	α	α^2	α^4	α^8
0	0	2	0	1	0	0
0	1	3	0	0	0	1
0	2	5	3	3	3	3
0	3	9	0	0	1	0
1	0	3	0	0	0	1
1	1	4	0	0	1	0
1	2	6	1	0	0	0
1	3	10	3	3	3	3
2	0	5	3	3	3	3
2	1	6	1	0	0	0
2	2	8	0	0	0	1
2	3	12	0	1	0	0
3	0	9	0	0	1	0
3	1	10	3	3	3	3
3	2	12	0	1	0	0
3	3	16	1	0	0	0

Now we see that

$$c_3 = f(A, B) = a_0b_1 + 3a_0b_2 + a_1b_0 + 3a_1b_3 + 3a_2b_0 + a_2b_2 + 3a_3b_1. \quad (6.15)$$

Hence the multiplication matrix contains 7 non-zero terms, as foreseen. We now turn to the implementation of this normal basis multiplier. We know that we only have to implement the above function once, and then we will get the different coefficients in the result by shifting the inputs. We therefore implement the multiplication network and connect it to 4 registers, in which we will rotate our input values. This is shown in figure 6.1.

6.3.1 Performance of the serial multiplier

The critical path of the multiplier will, as we have seen, depend on the normal basis chosen. For an optimal normal basis we have shown that we first need $2m - 1$ multiplications, that are parallel and therefore contributing to the depth with one multiplication. After this we need a network of depth $\lceil \log_2(2m - 1) \rceil$ with additions and subtractions. In the worst case, we have one subtraction for each value that comes from a multiplication, and these are spread amongst the different levels of the network, so that the full depth is composed of subtractions. Since in this case (when we have to subtract from a negated value) we need negations in front of the subtractions, we can't use the fact that they are preceded by multiplications to optimize, but the depth of each subtraction will be three. Hence the total depth will be $2 + \lceil \log_2 3(2m - 1) \rceil < 5 + \lceil \log_2 m \rceil$. We can, as for the serial multipliers in previous chapters, introduce data every m th clock cycle, for a throughput of $1/m$

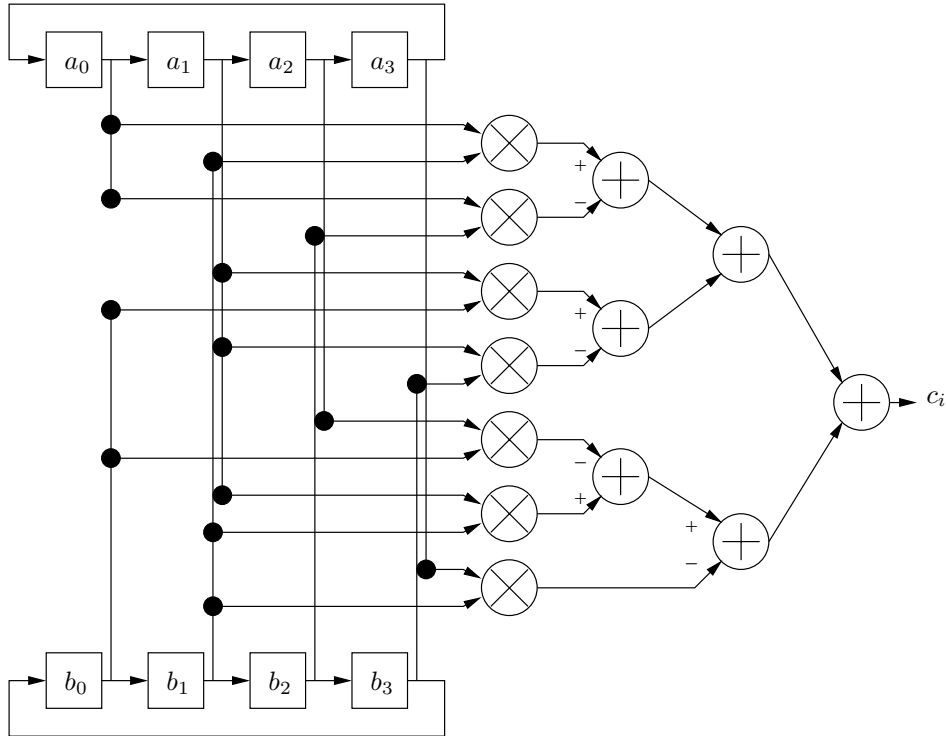


Figure 6.1. Implementation of normal basis serial multiplier for $GR(4^4)$, with $P(x) = x^4 + x^3 + x^2 + x + 1$.

results per clock cycle. The delay is $2m$ clock cycles, whereof m cycles are used for the actual calculations. The number of gates are $2m - 1$ for multiplication, and another $2m - 2$ additions or subtractions. This will incur a need of at least $16m - 12$ gates. Furthermore, we need $4m$ registers (of which half is not shown in the figure, but needed to take care of the serialized input).

6.4 A simple parallel multiplier

We can easily construct a parallel multiplier from the serial one described. In the serial case we compute the different output coefficients successively by rotating the inputs. We can of course choose to, instead of having one circuit and rotating the inputs, use m circuits, with the inputs hard-wired as “rotated”. This would give us a parallel multiplier. It is easy to imagine how it would look, just m cells of the gate network in figure 6.1 stacked in a parallel fashion, each giving a different output coefficient.

6.4.1 Performance of the parallel multiplier

The performance of the parallel multiplier is very easy to calculate from the performance of the serial multiplier, described in section 6.3.1. The critical path is the same, i.e. approximately at least $5\lceil\log_2 m\rceil$, and the number of gates will be m times the number needed for the serial multiplier, i.e. $16m^2 - 12m$ gates. Since the multiplier doesn't contain any registers we have a throughput of 1 result each clock cycle, and a delay of just 1 clock cycle.

6.5 Summary of performances

In the tables below the performance of the normal basis multipliers we have discussed are shown. Note that all values are approximations of the performances for optimal bases, which do not exist for more than a small part of all Galois rings.

Architecture	Number of gates	Number of registers
Serial	$16m - 12$	$4m$
Parallel	$16m^2 - 12m$	0

Table 6.1. Chip area of normal basis architectures.

Architecture	Critical path	Delay	Throughput
Serial	$5 + \lceil\log_2 m\rceil$	$2m$	$1/m$
Parallel	$5 + \lceil\log_2 m\rceil$	1	1

Table 6.2. Speed of normal basis architectures.

Chapter 7

Conclusions

In the thesis we have discussed a number of architectures for multiplying in Galois rings. Mainly we have modified similar architectures used for Galois fields to also work for Galois rings.

7.1 Similarities with field multipliers

One of the goals of this thesis was to investigate if it was possible to transform the architectures for multiplying in Galois fields into multipliers in Galois rings. For every architecture we have explored, we have found it possible to construct a similar architecture for multiplying in a ring, with just small modifications. We have also found that for the different types of bases used for multiplications in Galois fields there exist analogies for Galois rings, which make possible similar architectures. Our review of architectures has been exhaustive, though not complete, and we may therefore not state that there always will exist an implementation of multiplication in Galois rings similar to one for Galois fields. We may state, however, that generally it will be worth a try transforming an architecture for fields with desirable properties, since it will often be very similar for a Galois ring. The architectural similarities of the implementations translate into similarities in the performances.

Even though the over-all architecture of the multipliers for rings and fields may be similar, the detailed implementation will differ quite a bit. We have seen the following differences:

- Multiplication and addition in Z_4 is not as easily implemented as it is in Z_2 . Most existent architectures for multiplying in Galois fields are used for $GF(2^k)$, whereas we want architectures for $GR(4^m)$. For Z_2 multiplication of two elements is equivalent to a binary and-operation, and addition to a binary xor. For Z_4 we need to use the implementations presented in chapter 3.
- When we reduce modulo the generator polynomial we must use subtraction. Subtraction is actually needed for Galois fields also, but since subtraction and addition is the same in Z_2 , addition can be used there.

- In the architectures where the generator polynomial is needed to construct the circuit, i.e. the polynomial basis parallel multiplier and the normal basis multipliers, we get multiplications by constants from Z_4 , which we don't have in the field case. Or rather, we have them for fields also, but the constants will always be from Z_2 , i.e. 0 or 1.

Observing the differences above we see that none of them stem from the fact that we work with Galois *rings* instead of *fields*. Rather they are implications of the use of extensions over Z_4 instead of Z_2 . If we would have tried to implement multipliers in extension fields over larger primes than 2 they would behave very much like the multipliers for the Galois rings. This is not very surprising, since the main difference between fields and rings does not lie in multiplication, but in the fact that we can't divide in rings, which we can in fields.

7.2 Performance aspects

We have given the performance of the multipliers in terms of two measures, the area needed for an implementation and the speed. Depending on the application, in some cases a small chip area may be desired, while in other cases speed is of higher importance.

7.2.1 Minimizing the chip area needed

From the summaries in the end of chapters 4, 5 and 6 we see that the architecture needing the least number of gates is the normal basis serial architecture for an optimal basis, which needs $16m - 12$ gates, 5 less than the two dual basis serial architectures. The dual basis multipliers need, however, also a circuit for performing basis conversion, which may need some extra gates. This is not needed by the standard polynomial basis multipliers, but these need $m + 7$ gates and m registers more than the dual basis multipliers. Therefore our choices for choosing an architecture that minimizes the area of the circuit are the following.

- If we can choose an *optimal* normal basis, the normal basis serial multiplier is the best choice.
- If we can perform basis conversion with few gates a dual basis serial multiplier should be used, if not the above criteria is met.
- If none of the above criterias are met, the standard polynomial basis MSR multiplier is the best.

In the second case, with the dual basis multipliers we have not stated which one to use. This is because the alternative serial dual basis multiplier has shorter critical path, which allows for a higher clock frequency, but it also has less throughput each clock cycle. Therefore, we should use the alternative multiplier if we can choose the clock frequency, but not if the frequency is fixed (low) by other parts of the circuit.

7.2.2 Maximizing the speed

When it comes to speed mainly two things are of importance, how fast we can perform calculations, and what the delay of the results will be. For the speed of calculations we want high throughput and short critical path.

For the speed of calculations we see that the systolic multipliers are outstanding. They produce one result each clock cycle, just as the parallel multipliers, but their critical paths are of constant lengths, while the critical paths for the parallel multipliers have a length of at least $\lceil \log_2 m \rceil$, allowing a lower clock frequency. Of the systolic multipliers the dual basis multiplier has a shorter critical path, and is hence the better if the basis conversion does not require to large a circuit.

If, on the other hand, it is vital that we have a low delay, the parallel architectures are better since they only will delay the signal one clock cycle. Of these we see that the critical path is shortest for an optimal normal basis, or for the standard polynomial basis of a ring generated by one of the polynomials in table 4.3. If it is not possible to use one of these polynomials as generator polynomial we can't beforehand say if the standard polynomial or the normal basis multiplier will be the best, it has to be checked for the specific case.

Hence, the following should be our choices for maximizing speed.

- If the delay is not of vital importance, and we can perform basis conversion easily, the dual basis systolic multiplier is the best.
- If the delay is not of vital importance, but basis conversion for the dual basis multiplier would be too complicated, the standard polynomial basis systolic multiplier is the best.
- If a short delay is important one of the parallel multipliers should be chosen. Which one (the standard polynomial basis or the normal basis) depends on how we can choose the generator polynomial, and must be decided for each specific case.

7.3 Possible future research

Some questions have been left unanswered, and some have never been stated in this thesis. The following is a list of possible subjects for future research. Apart from these, just about everything that has been done on Galois field multipliers could also be investigated for Galois rings.

- Tower fields: In for example [9] tower field implementations for Galois fields are described. The basic idea is that instead of extending from Z_2 directly to $GF(2^n)$ the extension is made in steps. If this is possible, and how the multipliers would look for Galois rings could be interesting.
- More research on generator polynomials: Both for the standard basis polynomial basis parallel multiplier and for the normal basis multipliers the choice

of generator polynomial affects the implementation. More research on the performance for different polynomials should be interesting.

- Dual basis conversion: To be fully able to analyze the performance of the dual basis multipliers it would be good to know how complex the dual basis conversion is for different generator polynomials, and how to choose the generator polynomial in the best way.
- Multiplication by constant: Sometimes one of the multiplicands is known beforehand. A question that seeks an answer is how the different implementations could be simplified when this is the case.
- All our research in this essay has been inspired by results for fields. Perhaps some things could be discovered for rings, that is not possible for fields. For example, other types of generator polynomials could be interesting for the normal basis or standard basis parallel multipliers.

Bibliography

- [1] Alfred J. Menezes (Editor). *Applications of Finite Fields*. Kluwer Academic Publishers, 1993.
- [2] S.T.J. Fenn, M. Benaïssa, and D. Taylor. Dual basis systolic multipliers for $gf(2^m)$. *IEE Proceedings - Computers and Digital Techniques*, 144(1):43–46, January 1996.
- [3] A.R. Hammons, P.V. Kumar, A.R. Calderbank, N.J.A. Sloane, and P. Solé. The z_4 -linearity of kerdock, preparata, goethals, and related codes. *IEEE Transactions on Information Theory*, 41:301–319, 1995.
- [4] Thomas W. Judson. *Abstract Algebra - Theory and Applications*.
- [5] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, 1984.
- [6] Edoardo D. Mastrovito. *VLSI Architectures for Computations in Galois Fields*. Phd thesis 242, Department of Electrical Engineering, Linköping University, Linköping, Sweden, February 1991.
- [7] A.A. Nechaev. Kerdock code in a cyclic form. *Discrete Math. Appl.*, 1:365–384, 1991.
- [8] Ivar Tångring. A design study of an arithmetic unit for finite fields. Master’s thesis LiTH-ISY-EX-3396, Department of Electrical Engineering, Linköping University, Linköping, Sweden, June 2003.
- [9] Mikael Olofsson. *VLSI Aspects on Inversion in Finite Fields*. Phd thesis 731, Department of Electrical Engineering, Linköping University, Linköping, Sweden, February 2002.
- [10] W.C. Tsai and S.-J. Wang. Two systolic architectures for multiplication in $GF(2^m)$. *IEEE Proceedings on Computers and Digital Technology*, 147(6):375–382, November 2000.
- [11] Zhe-Xian Wan. *Quaternary Codes*, volume 8 of *Series on Applied Mathematics*. World Scientific, 1997.

- [12] Chin-Liang Wang and Jung-Lung Lin. Systolic array implementation of multipliers for finite fields $GF(2^m)$. *IEEE Transactions on Circuits and Systems*, 38(7), July 1991.
- [13] Stephen B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, 1995.

Appendix A

Minimal functions for binary representations

In this Appendix we will give all the minimal functions for different binary representations of the integers 0-3.

Representation: 00 = 0, 01 = 1, 11 = 3, 10 = 2

$x \cdot y$	$m_1 = (x_1 y_2) \oplus (x_2 y_1)$
	$m_2 = x_2 y_2$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$a_2 = x_2 \oplus y_2$
$x - y$	$s_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$s_2 = x_2 \oplus y_2$
$-x$	$n_1 = x_1 \oplus x_2$
	$n_2 = x_2$
$2x$	$d_1 = x_2$
	$d_2 = 0$

Representation: 00 = 0, 01 = 1, 11 = 2, 10 = 3

$x \cdot y$	$m_1 = (x_1 y_2) \oplus (x_2 y_1)$
	$m_2 = (x_2 y_2) \oplus (x_1 y_1)$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$a_2 = (x_2 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$x - y$	$s_1 = (x_1 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$s_2 = (x_2 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$-x$	$n_1 = x_2$
	$n_2 = x_1$
$2x$	$d_1 = x_1 \oplus x_2$
	$d_2 = x_1 \oplus x_2$

Representation: $00 = 0, 01 = 3, 11 = 1, 10 = 2$

$x \cdot y$	$m_1 = ((x_2 y_2) \oplus (x_1 y_2)) \oplus (x_2 y_1)$
	$m_2 = x_2 y_2$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$a_2 = x_2 \oplus y_2$
$x - y$	$s_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$s_2 = x_2 \oplus y_2$
$-x$	$n_1 = x_1 \oplus x_2$
	$n_2 = x_2$
$2x$	$d_1 = x_2$
	$d_2 = 0$

Representation: $00 = 3, 01 = 1, 11 = 0, 10 = 2$

$x \cdot y$	$m_1 = x_1 + y_1$
	$m_2 = (x_2 + y_1) \oplus (x_1 + y_2)$
$x + y$	$a_1 = x_1 \oplus y_1$
	$a_2 = (x_2 \oplus y_2) \oplus (x_1 + x_2)$
$x - y$	$s_1 = x_1 \oplus y_1$
	$s_2 = (x_2 \oplus y_2) \oplus (x_1 y_1)$
$-x$	$n_1 = x_1$
	$n_2 = x_1 \oplus x_2$
$2x$	$d_1 = 1$
	$d_2 = x_1$

Representation: $00 = 2, 01 = 1, 11 = 3, 10 = 0$

$x \cdot y$	$m_1 = ((x_1 y_2) \oplus (x_2 y_1)) \oplus (x_2 + y_2)$
	$m_2 = x_2 y_2$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$a_2 = (x_2 \oplus y_2)$
$x - y$	$s_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$s_2 = x_2 \oplus y_2$
$-x$	$n_1 = x_1 \oplus x_2$
	$n_2 = x_2$
$2x$	$d_1 = x_2$
	$d_2 = 0$

Representation: 00 = 1, 01 = 0, 11 = 3, 10 = 2

$x \cdot y$	$m_1 = (x_1 y_2)' \oplus (x_2 y_1)$
	$m_2 = (x_1 y_1)' \oplus (x_2 + y_2)'$
$x + y$	$a_1 = (x_2 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$a_2 = (x_1 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$x - y$	$s_1 = (x_2 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$s_2 = (x_1 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$-x$	$n_1 = x_2$
	$n_2 = x_1$
$2x$	$d_1 = (x_1 \oplus x_2)'$
	$d_2 = x_1 \oplus x_2$

Representation: 00 = 3, 01 = 1, 11 = 2, 10 = 0

$x \cdot y$	$m_1 = x_1 + y_1$
	$m_2 = ((x_2 y_1)' \oplus (x_1 y_2)) \oplus (x_1 + y_1)'$
$x + y$	$a_1 = x_1 \oplus y_1$
	$a_2 = (x_2 \oplus y_2) \oplus (x_1 + y_1)'$
$x - y$	$s_1 = (x_1 \oplus y_1)$
	$s_2 = (x_2 \oplus y_2) \oplus (x_1 y_2)'$
$-x$	$n_1 = x_1$
	$n_2 = (x_1 \oplus x_2)'$
$2x$	$d_1 = 1$
	$d_2 = x_1'$

Representation: 00 = 3, 01 = 0, 11 = 1, 10 = 2

$x \cdot y$	$m_1 = (x_1 y_1) \oplus (x_2 + y_2)'$
	$m_2 = (x_2 + y_1)' \oplus (x_1 y_2)$
$x + y$	$a_1 = (x_2 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$a_2 = (x_1 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$x - y$	$s_1 = (x_2 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$s_2 = (x_1 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$-x$	$n_1 = x_2$
	$n_2 = x_1$
$2x$	$d_1 = (x_1 \oplus x_2)'$
	$d_2 = x_1 \oplus x_2$

Representation: $00 = 2, 01 = 1, 11 = 0, 10 = 3$

$x \cdot y$	$m_1 = (x_2 + y_2)' \oplus (x_1 + y_1)$
	$m_2 = (x_1 + y_2) \oplus (x_2 + y_1)'$
$x + y$	$a_1 = (y_1' \oplus x_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$a_2 = (y_2 \oplus x_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$x - y$	$s_1 = (x_1 \oplus y_2) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
	$s_2 = (x_2 \oplus y_1) \oplus ((x_1 \oplus x_2)(y_1 \oplus y_2))$
$-x$	$n_1 = x_2$
	$n_2 = x_1$
$2x$	$d_1 = (x_1 \oplus x_2)'$
	$d_2 = (x_1 \oplus x_2)'$

Representation: $00 = 1, 01 = 3, 11 = 0, 10 = 2$

$x \cdot y$	$m_1 = x_1 + y_1$
	$m_2 = (x_1 y_1) + ((x_2 y_1)' \oplus (x_1 y_2)')$
$x + y$	$a_1 = x_1 \oplus y_1$
	$a_2 = (x_2 \oplus y_2) \oplus (x_1 + y_1)'$
$x - y$	$s_1 = (x_1 \oplus y_1)$
	$s_2 = (x_2 \oplus y_2) \oplus (x_1 y_1)$
$-x$	$n_1 = x_2$
	$n_2 = x_1'$
$2x$	$d_1 = 1$
	$d_2 = x_1$

Representation: $00 = 2, 01 = 3, 11 = 1, 10 = 0$

$x \cdot y$	$m_1 = (x_1 + y_2) \oplus (x_2 y_1)'$
	$m_2 = x_2 y_2$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus (x_2 y_2)$
	$a_2 = x_2 \oplus y_2$
$x - y$	$s_1 = (x_1 \oplus y_1) \oplus (x_1 y_2)$
	$s_2 = x_2 \oplus y_2$
$-x$	$n_1 = x_1 \oplus x_2$
	$n_2 = x_2$
$2x$	$d_1 = x_2$
	$d_2 = 0$

Representation: $00 = 1$, $01 = 0$, $11 = 2$, $10 = 3$

$x \cdot y$	$m_1 = (x_2 y_1)' \oplus (x_1 y_2)'$
	$m_2 = x_2 + y_2$
$x + y$	$a_1 = (x_1 \oplus y_1) \oplus (x_2 + y_2)'$
	$a_2 = (x_2 + y_2)'$
$x - y$	$s_1 = (x_1 \oplus y_1) \oplus (x_2 y_1)'$
	$s_2 = (x_2 + y_2)'$
$-x$	$n_1 = (x_1 \oplus x_2)'$
	$n_2 = x_2$
$2x$	$d_1 = x_2$
	$d_2 = 1$

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

© [Björn Abrahamsson]