Linköping Studies in Science and Technology.
Dissertation No. 425

VLSI Architectures and Arithmetic Operations with Application to the Fermat Number Transform

Lars-Inge Alfredsson

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1996

ISBN 91-7871-694-2 ISSN 0345-7524

Printed in Sweden by LJ Foto & Montage/VTT-Grafiska, Vimmerby 1996

To my wife

Anneli

and to our children

Michaela, Sebastian, Jonathan, and Antonia

Abstract

The properties of arithmetic operations in Fermat integer quotient rings $\mathbb{Z}_{2^m+1}$, where $m = 2^t$, are investigated. The arithmetic operations considered are mainly those involved in the computation of the Fermat number transform. We consider some ways of representing the binary coded integers in such rings and investigate VLSI architectures for arithmetic operations, with respect to the different element representations. The VLSI architectures are mutually compared with respect to area (A) and time (T) complexity and area-time performance (AT^2). The VLSI model chosen is a linear switch-level RC model.

In the polar representation, the nonzero elements of a field are represented by the powers of a primitive element of the field. In the thesis we particularly investigate the properties of arithmetic operations and their corresponding VLSI architectures with respect to the polar representation of the elements of Fermat prime fields. Some new results regarding the applicability of the Fermat number transform when using the polar representation are also presented.

Acknowledgements

My time as a PhD student has come to an end. I have really enjoyed teaching, studying, and doing research, which have been my main duties during these years. One of the main reasons why I wanted to join the Data Transmission group was the friendly and inspiring atmosphere that was — and still is — prevalent among the people in the group. I would like to thank all members of the Data Transmission group for providing this friendly and inspiring atmosphere.

I particularly would like to thank my supervisor, Professor Thomas Ericson, for giving me the opportunity to join the Data Transmission group. He has been an excellent guide on my tour into the world of science and he has always supported my work with a proper balance between friendly encouragements and educating directions.
I also appreciate the fruitful discussions with Professor Stefan Dodunekov, Professor Christer Svensson, and Dr. Edoardo Mastrovito.

Finally, I would like to thank my wonderful family, to whom I dedicate this thesis. The seemingly never-ending process of writing the thesis has come to an end. From now on, I will spend a lot more time with You!

Linköping, March 1996

Lasse Alfredsson

There are certain privileges of a writer,
the benefit whereof, I hope, there will be no reason to doubt;
Particularly, that where I am not understood, it shall be concluded,
that something very useful and profound is couched underneath.

– Jonathan Swift
(Tale of a Tub, preface 1704)

Not that the story need to be long,
but it will take a long while to make it short.

– Henry David Thoreau
(Letter, 16 Nov. 1867.)

Contents

1 Introduction **1**

2 Binary Arithmetic in the Fermat Integer Quotient Ring **3**

2.1 The Integer Quotient Ring 3

2.2 The Number Theoretic Transform 4

2.2.1 Suitable Integer Rings 6

2.3 The Fermat Number Transform 10

2.3.1 Fermat Numbers 10

2.3.2 The Transform Kernel 12

2.3.3 Butterfly Computations 16

2.4 Element Representation 23

3 Applications **25**

3.1 Convolution and Correlation of Real Integer Sequences . . . 26

3.2 Decoding of Reed-Solomon Codes 30

4 The VLSI Model **33**

4.1 Introduction 34

4.2 Complexity and Performance 36

4.2.1 The Delay Model 36

4.2.2 Area and Time Complexities 38

4.3 Basic CMOS Building Blocks 42

4.3.1 The Inverter and the Transmission Gate 42

4.3.2 The Two-Input Multiplexer 44

4.3.3 Two-Input Gates 44

4.3.4 The Single-Bit Adder 47

4.3.5 The Register 51

4.3.6 Table of Complexity Parameters 55

4.4 Implementing the Fermat Number Transform 57

5 The Normal Binary Coded Representation **59**

5.1 Architectures for Arithmetic Operations 59

5.1.1 Modulus Reduction 60

5.1.2 Negation 68

5.1.3 Addition and Subtraction 72

5.1.4 Multiplication by Powers of 2 77

5.1.5 General Multiplication 81

5.1.6 Exponentiation of the Transform Kernel 84

5.2 Summary 87

6 The Diminished–1 Representation **89**

6.1 Linearly Transformed Representations 89

6.1.1 Arithmetic Operations 90
6.2 The Use of a Zero Indicator 93
6.3 The Diminished–1 Representation 98
6.3.1 Code Translation 98
6.3.2 Modulus Reduction 106
6.3.3 Negation 106
6.3.4 Addition and Subtraction 108
6.3.5 Multiplication by Powers of 2 122
6.3.6 General Multiplication 128
6.3.7 Exponentiation of the Transform Kernel 152
6.4 Summary 152

7 The Polar Representation **155**
7.1 Introduction 155
7.2 Arithmetic Operations 156
7.2.1 Discrete Exponentiation 157
7.2.2 The Discrete Logarithm 157
7.2.3 Modulus Reduction 159
7.2.4 Negation 159
7.2.5 Addition and Subtraction 160
7.2.6 General Multiplication 161
7.2.7 Multiplication by Powers of ω 162
7.3 Zech's Logarithm 165
7.4 Properties of the $\mathcal{D}_m$ Matrix 167
7.4.1 Discrete Exponentiation 173

7.4.2 The Discrete Logarithm . . . 177
7.4.3 Zech's Logarithm . . . 179
7.5 The Mirror Sequence $\mathcal{M}_m$ . . . 182
7.5.1 Discrete Exponentiation Using a Look-Up Table . . . 183
7.5.2 The Discrete Logarithm Using a Look-Up Table . . . 183
7.5.3 The Mirror Properties of $\mathcal{M}_m$ . . . 185
7.5.4 Finding the Unique Distinct Positions in $\mathcal{M}_m$ . . . 189
7.5.5 Addressing the Look-Up Table for Discrete Logarithm 195
7.6 Architectures for Arithmetic Operations . . . 197
7.6.1 Discrete Exponentiation . . . 197
7.6.2 The Discrete Logarithm . . . 205
7.6.3 Negation . . . 206
7.6.4 Addition . . . 208
7.6.5 General multiplication . . . 219
7.6.6 Multiplication by powers of ω . . . 223
7.7 Summary . . . 231

8 Comparisons Between Element Representations 233

8.1 Arithmetic Operations . . . 233
8.1.1 Modulus Reduction . . . 233
8.1.2 Code Translation . . . 234
8.1.3 Negation . . . 235
8.1.4 Addition . . . 236
8.1.5 General Multiplication . . . 238
8.1.6 Multiplication by Powers of ω . . . 239

8.1.7 Butterfly Computations 242
8.2 Other element representations 246

9 Conclusions **249**

A Proofs of Some Theorems **251**
A.1 Proof of Theorem 2.1 251
A.2 Proof of Theorem 2.3 253
A.3 Proof of Theorem 2.5 254

B A Table of Some Primes **257**

C Further Properties of Zech's Logarithms **261**

Bibliography **269**

Chapter 1

Introduction

In 1972 Rader [77] proposed transforms in the ring of integers modulo a Mersenne or a Fermat number ($2^n - 1$ and $2^m + 1$; $m = 2^t = 1, 2, 4, 8, \ldots$, respectively) to compute error-free convolutions of real integer sequences. Later, Agarwal and Burrus [2] showed that for some transform lengths the radix-2 Fermat number transform can be implemented using only addition, subtraction, and bit shifting, i.e. without using multiplication. This transform was shown to be faster than the conventional fast Fourier transform over the complex field.

There are also other applications of the Fermat number transform. Justesen [54] was one of the first to consider Reed-Solomon codes over the finite field of integers modulo a Fermat prime. He stated that the decoding complexity of such codes can be reduced if the Fermat number transform is used to evaluate the syndromes and error magnitudes. This was further investigated by Reed et al. [82] and others.

The special attributes of the Fermat number transform have led several researchers to consider the VLSI (Very Large Scale Integration) implementation of arithmetic operations in the ring of integers modulo a Fermat number. These operations are traditionally implemented using binary logic circuits, which means that the elements of the ring have a binary coded form of representation. The $2^m + 1$ binary coded elements of the ring of integers modulo a Fermat number can be represented using $m + 1$ bits. We thus get numerous ways of representing the elements of the ring. The complexity and per-

formance of architectures for arithmetic operations depend inter alia on the representation chosen.

The most known representations are the ones proposed by McClellan [65] and Leibowitz [58]. Their coding schemes are linear coordinate transformations of the normal binary coded representation of the elements in the ring. Using their representations, operations like addition, multiplication by two, and the code translation can be carried out fairly easy in VLSI. Also, for some relatively small transform lengths, the transform multiplications by powers of the transform kernel can be carried out as binary shifts. This is a well known property of the Fermat number transform. One of the main disadvantages of using McClellan's or Leibowitz' element representation is that for most other possible transform lengths, the resulting transform computation involves general multiplications (by powers of the transform kernel). Nevertheless, Leibowitz' so called *diminished–1* representation is used by most people who consider the VLSI implementation of the Fermat number transform.

In this thesis we investigate various ways of representing the binary coded elements of the ring of integers modulo a Fermat number. For each element representation considered, the properties of the arithmetic operations involved in the computation of the Fermat number transform are thoroughly investigated. Some other (arithmetic) operations are also considered. We also investigate VLSI architectures for the arithmetic operations. Some architectures are previously known and some are new. We show how each of these architectures is derived from its associated analytical expression for the arithmetic operation in question.

One of our main goals is to find a representation that makes it possible to compute the Fermat number transform with favourable area-time performance for *all* possible transform lengths. In particular, we focus on the arithmetic operations obtained when using the *polar representation* of the elements of Fermat prime fields. In the polar representation, the elements of a field are represented by powers of some primitive element of the field.

Chapter 2

Binary Arithmetic in the Fermat Integer Quotient Ring

In this chapter we give a formal introduction to the number theoretic transform in general and the Fermat number transform in particular. The chapter contains several known results from the area of number theory. We also consider some fast Fourier transform algorithms for implementing the Fermat number transform. For each algorithm, we find out which arithmetic operations are needed and the complexity of computing the transform. The purpose of the survey is to get our work into perspective. The chapter is concluded by presenting some aspects of representing the binary coded integers of the Fermat integer quotient ring.

2.1 The Integer Quotient Ring

A *ring* is an algebraic system consisting of a set of elements together with addition, subtraction, and multiplication. The result of any of these arithmetic operations is always an element of the original set. It may also be possible to divide in a ring. Then the *multiplicative inverse* of the divisor must exist in the ring.

A natural example of a ring is $\mathbb{Z}$, the ring of integers; for $a,\ b\ \in\ \mathbb{Z}$, we have $a\ +\ b,\ a\ -\ b,\ a\ \cdot\ b\ \in\ \mathbb{Z}$. Denote by $\mathbb{Z}_q$ the *quotient ring* of integers modulo an integer q: It consists of the set $\{0,\ 1,\ 2,\ \ldots,\ q\ -\ 1\}$ of integers and the result of every arithmetic operation is reduced modulo q. Thus, an integer c maps into $\mathbb{Z}_q$ as the remainder r of c divided by q. If we have $c\ =\ r\ +\ dq$ for some integer

d, then c and r are *congruent* modulo q. The notation for such a congruence is

$$c \equiv r \pmod{q}.$$

The multiplicative inverse of an element of $\mathbb{Z}_q$ exists if and only if the element is relatively prime to the modulus q.[1] If q is a prime number, then every non-zero element of $\mathbb{Z}_q$ has a multiplicative inverse and thus division becomes a general operation in the ring. Then $\mathbb{Z}_q$ is called a *field*.[2] For a detailed mathematical survey on the theory of rings and fields, see for example Lidl and Niederreiter [60] or Herstein [50].[3]

In this thesis we investigate VLSI architectures for arithmetic operations in the integer quotient ring $\mathbb{Z}_q$, where q is a Fermat number. Even though the development of multiple-valued logic has progressed over the years [29] it is still a difficult problem to design q-valued logic circuits for large q. Therefore, we restrict ourselves to representations of the integers modulo q as *binary* coded symbols and use binary logic circuits in the VLSI architectures for the arithmetic operations in $\mathbb{Z}_q$.

2.2 The Number Theoretic Transform

Before going into details about the Fermat number transform, we give the definition of the number theoretic transform in an arbitrary integer quotient ring $\mathbb{Z}_q$. We also discuss which moduli q are most suitable, with respect to the complexity of computing the number theoretic transform. The computation of the number theoretic transform (NTT) involves integer ring arithmetic operations. The NTT is a DFT-like (discrete Fourier transform) transform which is computed in the ring of integers modulo some integer:

Definition 2.1 *In the ring $\mathbb{Z}_q$ of integers modulo a positive integer $q = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ the number theoretic transform of the sequence $\mathbf{x} = \{x_n\}_{n=0}^{N-1}$ of elements $x_n \in \mathbb{Z}_q$ is a sequence $\mathbf{X} = \{X_k\}_{k=0}^{N-1}$, $X_k \in \mathbb{Z}_q$, given by*

[1] If $a \in \mathbb{Z}_q$ and q are relatively prime, then we have $1 = ab + dq \equiv ab \pmod{q}$ where b and d are integers. The integer $b \bmod q$ is then referred to as the multiplicative inverse of a under multiplication modulo q.

[2] Thus, a field is a ring in which it is also possible to divide.

[3] The quotient ring $\mathbb{Z}_q$ is denoted by $\mathbb{Z}/(q)$ and J_q in [60] and [50] respectively. The notation $\mathbb{Z}_q$, which we conveniently use in this thesis, is very common in many other books on abstract algebra and number theory.

$$X_k \triangleq \sum_{n=0}^{N-1} x_n \omega^{kn} \pmod{q}; \qquad k = 0, 1, \ldots, N-1\,, \tag{2.1}$$

where ω is any element with order N in $\mathbb{Z}_q$.

The factors $p_1,\ p_2, \ldots,\ p_k$ of q are distinct primes.

Remark: Let ω and q be relatively prime positive integers. Then, the least positive integer N such that $\omega^N \equiv 1 \pmod{q}$ is called the *order of ω modulo q*. We denote the order of ω modulo q by $\mathrm{ord}_q\,\omega$. Thus, for the *transform kernel* ω we get $\mathrm{ord}_q\,\omega = N$. Sometimes, ω is said to be a *primitive Nth root of unity*.

Because we have $\mathrm{ord}_q\,\omega = N$, the product kn in the exponent of ω in (2.1) is calculated modulo N. It is easy to show that the NTT, as well as the DFT, possesses the cyclic convolution property, i.e. the transform of a cyclic convolution of two sequences is equal to the product of their transforms. There are also other properties of the DFT that have their counterparts in the NTT. The inverse number theoretic transform is given by

$$x_n \triangleq N^{-1} \sum_{k=0}^{N-1} X_k \omega^{-kn} \pmod{q}; \qquad n = 0, 1, \ldots, N-1, \tag{2.2}$$

where N^{-1} is the multiplicative inverse of N modulo q, i.e. the least positive integer M for which $N \cdot M \equiv 1 \pmod{q}$. Such an inverse exists if and only if $\gcd(N,\, q) = 1$. The factor ω^{-kn} in (2.2) is congruent to $\omega^{N-kn \bmod N} \bmod q$. Therefore, (2.2) involves multiplication by *positive* powers of ω modulo q.

It is sometimes convenient to use the multiplicative inverse ω^{-1} of ω modulo q instead of ω as the transform kernel of the inverse NTT.[4] If there exists an integer ω with order N modulo q, then its inverse $\omega^{-1} \equiv \omega^{N-1} \pmod{q}$ also exists.

Thus, we can say that a number theoretic transform of length N and its inverse transform exist in $\mathbb{Z}_q$ if there is an integer ω with order N modulo q and N has a multiplicative inverse modulo q. The following theorem may be useful when determining the possible lengths of an invertible transform in an integer quotient ring:

[4] We have $\omega^{-kn} = \left(\omega^{-1}\right)^{kn}$.

Theorem 2.1 *There exists an invertible NTT of length N in $\mathbb{Z}_q$ if and only if $N \mid (p_i - 1)$ for every prime p_i that divides q.*

Proof: See Section A.1 of Appendix A. □

Thus, the theorem says that the transform length N must satisfy

$$N \mid \gcd(p_1 - 1,\, p_2 - 1, \ldots,\, p_k - 1), \tag{2.3}$$

where $q = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$. In particular, if $q = p$ is a prime, then every nonzero element of the *prime field* $\mathbb{Z}_p$ has a multiplicative inverse and there exists an NTT of every length N that divides $p - 1$.

2.2.1 Suitable Integer Rings

There exist infinitely many number theoretic transforms. The modulus $q = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ should be chosen in a suitable way with respect to the complexity and performance of the architectures for the binary coded integer arithmetic operations modulo q, and with respect to the possible NTT lengths that will be obtained. Multiplication by powers of the transform kernel ω is usually the most complex arithmetic operation involved in the computation of the NTT. Therefore, the efficiency of a VLSI implementation of an NTT is often largely determined by the efficiency by which such multiplications can be carried out.

The direct computation of an NTT of length N requires in the order of N^2 multiplications and $N(N-1)$ additions. If the transform length is composite the NTT can be decomposed into several transforms of smaller sizes which may be computed using some fast Fourier transform (FFT) algorithm [17, Ch. 4]. The FFT algorithm is most efficiently computed if the transform is a single-radix transform with a small radix, i.e. if the transform length can be expressed as a power of a small integer. For example, if $N = r^b$, for some r and b, the NTT can be computed using a *radix-r* FFT algorithm. Such an algorithm requires in the order of $k(r-1)N \log_r N$ multiplications and $(r-1)N \log_r N$ additions, where k depends on N and the choice of ω [33, 35]. Hence, the complexity of computing the NTT can be significantly reduced by choosing a suitable transform length and using an FFT algorithm. From (2.3) it follows that it is the modulus that determines the possible transform lengths.

From a VLSI implementation point of view, the reduction modulo q of a binary coded integer is simplest to perform when q is close to a power of two or when the binary coded representation of q contains few ones. The modulus reduction in $\mathbb{Z}_{2^m}$ is very simple and straightforward, but since 2 is a prime factor of

$q = 2^m$ the maximum possible NTT length in any ring of size 2^m is 1. The same conclusion holds for every even modulus q. Integer quotient rings with even modulus are therefore not interesting from an NTT application point of view.

Any odd natural number q can be written on the form $q = a \cdot r^m + 1$ for some natural numbers $a,\ r$, and m, where r does not divide a. When q is a prime, we see from (2.3) that the possible transform lengths are the ones that divide $a \cdot r^m$. Therefore, the maximum radix-r transform length in the prime field $\mathbb{Z}_{a \cdot r^m + 1}$ is r^m. Because a radix-r transform of length $N = r^b$ involves in the order of $(r-1)N \log_r N$ multiplications and additions, the transform is most efficiently computed if N is highly composite, i.e. r is small.

Chevillat gives a table [33, Tab. II] of 8-bit to 16-bit moduli whose associated integer quotient rings each contains a single-radix transform of length $N \geq 16$. Some of these moduli are composite, but most of them are prime numbers. The modulus should be chosen such that the modulus reduction is not a very complex operation. As an example we consider $\mathbb{Z}_q$ with a prime modulus $q = 39367$, for which $q - 1 = 2 \cdot 3^9$. This is one of Chevillat's numbers. The maximum transform length of a single-radix NTT in $\mathbb{Z}_{39367}$ is $3^9 = 19683$. However, because the normal binary representation of $q = 39367$ is 1001100111000111, the reduction modulo q may not be as simply performed as when q can be represented by much fewer ones or when it is closer to a power of two.

We mentioned above that multiplication by powers of the transform kernel should be carried out as simply as possible. The complexity of such a multiplication depends inter alia on the kernel chosen. However, in an arbitrary integer quotient ring there may not exist a suitable kernel for which this complexity is low. In general, even if there exist single-radix transforms of great lengths in an integer ring, it is not certain that a transform multiplication can be computed using a procedure that is simpler than general multiplication.

Mersenne numbers

A set of integers of particular interest is the set of *Mersenne numbers*. These numbers are of the form $2^m - 1$, where $m = 2, 3, 4, \ldots$. We denote such numbers by M_m. The NTT in a Mersenne integer quotient ring $\mathbb{Z}_{M_m}$ is usually called the Mersenne number transform. One of the first to consider Mersenne number transforms was Rader in 1972 [77]. Arithmetic operations are easily carried out in $\mathbb{Z}_{M_m}$ if the elements are represented as normal binary coded m-bit integers, because then the complexity of performing the operations equals the complexity of one's-complement arithmetic: Because $2^m \equiv 1 \pmod{2^m - 1}$, the modulus reduction is equivalent to the procedure for handling overflow in one's-complement arithmetic.

m	$M_m = 2^m - 1$	$M_m - 1 = 2(2^{m-1} - 1)$
3	7	$2 \cdot 3$
5	31	$2 \cdot 3 \cdot 5$
7	127	$2 \cdot 3^2 \cdot 7$
13	8 191	$2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
17	131 071	$2 \cdot 3 \cdot 5 \cdot 17 \cdot 257$
19	524 287	$2 \cdot 3^3 \cdot 7 \cdot 19 \cdot 73$
31	2 147 483 647	$2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$

Table 2.1: *The first 7 Mersenne prime numbers.*

There is, however, no general fast algorithm for the computation of the Mersenne number transform. Let $m = \lambda k$ where λ is a prime number. Then $2^\lambda - 1$ divides $2^{\lambda k} - 1$. This is easily shown by using the relation $x^k - 1 = (x-1)(x^{k-1} + x^{k-2} + \cdots + x + 1)$ for $x = 2^\lambda$ which gives $2^{\lambda k} - 1 = (2^\lambda - 1)(2^{\lambda(k-1)} + 2^{\lambda(k-2)} + \cdots + 2^\lambda + 1)$, and thus we get $(2^\lambda - 1) \mid (2^{\lambda k} - 1)$. If $m = 2k$ is even then $2^2 - 1 = 3$ is a prime factor of M_m which, from (2.3), implies that the transform length divides 2. Thus, a transform of meaningful length can only be obtained when m is odd. Furthermore, if $M_m = 2^m - 1$ is prime then m must also be prime, i.e. k equals 1 in the previous factorisation of $2^{\lambda k} - 1$. The converse, however, is not always true – for example $2^{11} - 1 = 2047 = 23 \cdot 89$ is not a prime number. This shows, by applying (2.3) to the prime factorisation of M_m, that the possible lengths of the NTT in $\mathbb{Z}_{M_m}$ are relatively small when m is odd and M_m is composite.

When M_m is prime the NTT length must divide $M_m - 1 = 2^m - 2$. The third column of Table 2.1 shows the prime factorisations of $M_m - 1$ for the first 7 Mersenne numbers. We see that for large M_m the number $M_m - 1$ is *not* highly composite. Therefore, there may not exist any efficient FFT-type algorithm to compute transforms of great lengths in $\mathbb{Z}_{M_m}$. Properties of Mersenne number transforms and some applications are further discussed in Chapter 6.3 of Blahut [17] and by Rader [77].

Numbers of the form $2^n - 2^m + 1$

The final set of numbers to be considered here are prime numbers of the form $q = 2^n - 2^m + 1$, where $0 < m < n$. Several of these numbers can also be found in the set of Chevillat numbers. In 1976, Pollard [73] stated that such numbers are good choices as integer ring moduli.

The normal binary representation of the n-bit modulus $q = 2^n - 2^m + 1$ is

$$\overbrace{\underbrace{1\,1\,\cdots\,1}_{n-m\text{ ones}}\;\underbrace{0\,0\,\cdots\,0}_{m-1\text{ zeros}}\;1}^{n\text{ bits}},$$

i.e. a block of $n - m$ ones followed by a block of $m - 1$ zeros and a one in the least significant bit position. It is quite easy to perform the modulus reduction in VLSI when the modulus has this form and if the integer to be reduced is less than 2^n. Because $2^n - 2^m + 1 \equiv 0 \pmod{2^n - 2^m + 1}$ we get $2^n - 2^m \equiv -1 \pmod{2^n - 2^m + 1}$. Therefore, when the $n - m$ most significant bits of an integer not greater than $2^n - 1$ are all ones, the modulus reduction is carried out by changing these bits to zero and subtract one (1) from the resulting binary coded integer. When the integer to be reduced is greater than $2^n - 1$, the modulus reduction procedure is just slightly more complicated.

Example: $n = 8,\ m = 5 \Rightarrow q = 2^8 - 2^5 + 1 = 225$.
Modulus reduction $235 \equiv 10 \pmod{225}$:

$$\begin{array}{rcr} 1\,1\,1\,0\,1\,0\,1\,1 & = & 235 \\ \downarrow\downarrow\downarrow & & \downarrow \\ 0\,0\,0\,0\,1\,0\,1\,1 & = & 11 \\ -\qquad\qquad 1 & & -\ 1 \\ \hline 0\,0\,0\,0\,1\,0\,1\,0 & = & 10 \end{array}$$

In Section 5.1.1 we show how subtraction by one can be carried out in VLSI in a simple way.

For prime moduli q, the possible transform lengths divide $q - 1 = 2^m(2^{n-m} - 1)$,which implies that there exist radix-2 NTTs of length $N = 2^b$, where $b \leq m$, in the corresponding prime fields $\mathbb{Z}_q$. In Table B.1 of Appendix B we present the factorisations of $q - 1$ together with n, m, and q for all primes q of the form $q = 2^n + 2^m + 1$ where $0 < m < n \leq 32$. These primes were found by computer search. In order to obtain a transform of great length, m should be large. On the other hand, in order to make efficient use of the n-bit representation of the integers in $\mathbb{Z}_q$, m should be as small as possible ($m \geq 1$). The best choice of m with respect to n may differ, depending on the NTT application in question.

We have not found any general structure of the prime factorisations of composite moduli $q = 2^n - 2^m + 1 = 2^m(2^{n-m} - 1) + 1$. However, it may be profitable to consider subsets of this set of moduli for which the NTT in the corresponding integer rings possesses some of the desirable properties. Such

a subset may, for example, consist of moduli for which $n-m$ is constant. Properties of the NTT in $\mathbb{Z}_q$ can then be examined separately in each subset.

We see in Table B.1 of Appendix B that there are several prime moduli for which $n - m$ is small. A report on primes of the form $k \cdot 2^m + 1$ was published by Robinson in 1958 [83]. In [83] he also presented a table of all such primes for $k < 100$ and $m < 512$. Liu et al. [62] considered primes of the form $2^m(2^m - 1) + 1$, i.e. for $n = 2m$, for some values of m. Number theoretic transforms in the integer ring modulo $2^m(2^m - 1) + 1$ have also been considered by Dubois and Venetsanopoulos [38, 39]. Some other researchers have investigated properties of moduli of the form $3 \cdot 2^m + 1$, i.e. for $n - m = 2$, see for example Golomb [47] and Golomb et al. [48]. In [48] the authors discuss how to perform arithmetic operations in $\mathbb{Z}_{3\cdot 2^m+1}$.

The above-mentioned numbers are all special cases of numbers of the form $q^{(p-1)n} + q^{(p-2)n} + \cdots + q^n + 1 = (q^{pn} - 1)/(q^n - 1)$ for some integers q, p, and n. In a recent article by Dimitrov et al. [37], the authors define number theoretic transforms in integer quotient rings with such moduli for $q = 2$, $p = 3,\ 5$, and 7, and for some appropriate values of n.

In the present thesis we consider moduli $q = 2^m(2^{n-m} - 1) + 1$ for $n - m = 1$, i.e moduli of the form $q = 2^m + 1$. For m equal to a power of two, such numbers are called *Fermat numbers*.

2.3 The Fermat Number Transform

2.3.1 Fermat Numbers

In this section we study number theoretic transforms in integer quotient rings with moduli of the form $2^m + 1$ for some m.

Theorem 2.2 *If $2^m + 1$ is a prime then m is a power of two.*

Proof: (From [42, pp. 23–24]) Suppose m has an odd factor k, say $m = nk$. Using the factorisation $x^k+1 = (x+1)(x^{k-1}-x^{k-2}+x^{k-3}-\cdots+x^2-x+1)$ for $x = 2^n$ we get $2^m+1 = 2^{nk}+1 = (2^n+1)(2^{n(k-1)}-2^{n(k-2)}+2^{n(k-3)}-\cdots+2^{2n}-2^n+1)$, which apparently is composite. The only numbers that have no odd factor are the powers of two. □

We have shown that $2^m + 1$ is *not* a prime when m is *not* a power of two, but for which $m = 2^t$ do we get a prime? The number

$$F_t \triangleq 2^m + 1; \qquad m \triangleq 2^t,$$

where $t \in \mathbb{N}$, is defined as the tth *Fermat number*.[5] Fermat observed that the first five such numbers are all prime:

$$\begin{aligned} F_0 &= 2 + 1 &&= 3, \\ F_1 &= 2^2 + 1 &&= 5, \\ F_2 &= 2^4 + 1 &&= 17, \\ F_3 &= 2^8 + 1 &&= 257, \\ F_4 &= 2^{16} + 1 &&= 65\,537. \end{aligned}$$

Fermat expressed his belief that *every* F_t is a prime, but admitted that he had no proof.

From Fermat's little theorem [84, Th. 5.3] it follows that if p is a prime and a is a positive integer, then $a^p \equiv a \pmod{p}$, that is $p \mid (a^p - a)$. [6] In general, if a is a positive integer and q is a composite positive integer that divides $a^q - a$, then q is usually called a *pseudoprime to the base* a. One of the reasons for Fermat's statement that every F_t is a prime may have been that in fact all Fermat numbers are either primes or *pseudoprimes*.

We see that for every Fermat number $F_t = 2^{2^t} + 1$, where $t \in \mathbb{N}$, the relation $F_t \mid (2^{F_t} - 2)$ holds [93, Exerc. 2]: For *any* positive integer t we have $t + 1 \leq 2^t$, and thus $2^{t+1} \mid 2^{2^t}$. Consequently, we have $(2^{2^{t+1}} - 1) \mid (2^{2^{2^t}} - 1) = 2^{F_t - 1} - 1$. Because $2^{2^{t+1}} - 1 = (2^{2^t} + 1)(2^{2^t} - 1)$ we get $F_t = (2^{2^t} + 1) \mid (2^{F_t - 1} - 1)$ and hence $F_t \mid (2^{F_t} - 2)$.

Therefore, all composite Fermat numbers F_t are pseudoprimes to the base 2. When trying to find the factors of composite Fermat numbers, the following theorem is of good use:

[5] Henceforth, whenever the number $2^m + 1$ appears in the thesis we always mean the Fermat number F_t, i.e. we implicitly assume $m = 2^t$ for some natural number t.

[6] Even the ancient Chinese had a test for primality which is similar to Fermat's little theorem. The test said that an integer p is a prime if *and only if* $p \mid (2^p - 2)$. By Fermat's little theorem we know that the test is correct when p *is* an odd prime, but the converse is not always true. For example, the ancient Chinese did not discover that the smallest composite integer that passes their test is $341 = 11 \cdot 31$. It can easily be verified that $2^{341} \equiv 2 \pmod{341}$ and thus $341 \mid (2^{341} - 2)$.

Theorem 2.3 *Every prime divisor of the Fermat number $F_t = 2^{2^t} + 1$, where $t \geq 2$, is of the form $k \cdot 2^{t+2} + 1$, for some natural number k.*

Proof: See Section A.2 of Appendix A. The proof involves Euler's theorem and the concept of quadratic residues. □

Thus, every prime divisor of F_t is congruent to 1 modulo 2^{t+2} for $t \geq 2$. Actually, because the product of two numbers of the form $k \cdot n + 1$ is also of this form, *any* divisor of F_t is congruent to 1 modulo 2^{t+2} for $t \geq 2$. Lucas [36, pp. 376–379] was the first to prove that every prime factor of F_t is of the form $k \cdot 2^{t+2} + 1$. Prior to Lukas' proof Euler showed that $5 \cdot 2^7 + 1 = 641$ is a factor of F_5. The complete factorisations of F_5 and F_6 are

$$F_5 = (5 \cdot 2^7 + 1)(3 \cdot 17\,449 \cdot 2^7 + 1); \qquad \text{Euler 1732}$$

$$F_6 = (3^2 \cdot 7 \cdot 17 \cdot 2^8 + 1)(5 \cdot 47 \cdot 373 \cdot 2\,998\,279 \cdot 2^8 + 1); \qquad \text{Landry 1880}$$

To this day, *no Fermat prime greater than F_4 has been found*. Since the days of Euler, finding the prime factors of composite Fermat numbers or proving that certain Fermat number *are* composite have been two of the most famous problems in number theory. In 1958, Robinson presented a list [83, Table 2] of all known prime factors of composite Fermat numbers together with the dates of discovery. Using today's powerful computing tools still more prime factors have been found. In [28, page lxxxviii], Brillhart et al. published a table of all factors of composite Fermat numbers known in 1988. To the author's knowledge, the largest Fermat number with known factorisation is $F_{11} = 2^{2^{11}} + 1$, which was factored by Brent and Morain in 1988 using the elliptic curve method [24], [25]. The ninth Fermat number F_9 was factored by A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard in 1990 by means of the number field sieve [59]. The complete factorisation of F_{10} is still not known. The largest Fermat number with a known factor is F_{23471}. It is divisible by $5 \cdot 2^{23473} + 1$.

2.3.2 The Transform Kernel

The number theoretic transform in the Fermat integer quotient ring $\mathbb{Z}_{F_t}$ is often referred to as the Fermat number transform (FNT). A great advantage of the FNT is that the possible transform lengths are all highly composite. As shown in Section 2.3.1, a composite Fermat number F_t can be factorised into prime

powers as

$$F_t = (k_1 2^{t+2} + 1)^{n_1} (k_2 2^{t+2} + 1)^{n_2} \cdots (k_l 2^{t+2} + 1)^{n_l},$$

for some $k_1, k_2, \ldots, k_l$ and $n_1, n_2, \ldots, n_l$. Let $2^{\hat{k}}$ be a common factor of $k_1, k_2, \ldots, k_l$ for some $\hat{k}$.[7] Equation (2.3) then implies that there exist radix-2 transforms in $\mathbb{Z}_{F_t}$. The transform length N must divide $2^{t+\hat{k}+2}$. Furthermore, when F_t is prime the possible lengths N divide $F_t - 1 = 2^{2^t}$. Thus, the radix-2 FNT in $\mathbb{Z}_{F_t}$ is of length

$$N = 2^b \; ; \quad \begin{cases} 0 \leq b \leq t + \hat{k} + 2 \; ; & F_t \text{ is composite} \\ 0 \leq b \leq m \; (= 2^t) \; ; & F_t \text{ is prime} \end{cases} . \tag{2.4}$$

Because the FNT length N is a power of two the transform can be computed using a fast and efficient algorithm. Using the radix-2 Cooley-Tukey FFT algorithm [35], a transform of length $N = 2^b$ in a Fermat integer quotient ring $\mathbb{Z}_{F_t}$ can be computed using only $(N/2)\log_2 N$ multiplications and $N \log_2 N$ additions modulo F_t. Since elements of the sequence that is to be transformed are multiplied by powers of the kernel ω, the complexity of computing the transform depends strongly on the choice of ω.

Using binary arithmetic, multiplication by a power of two can be implemented in VLSI as binary shifts. We see by the congruence

$$1 = (-1)^2 \equiv 2^{2m} \pmod{2^m + 1}$$

that the integer 2 has order $2m = 2^{t+1}$ modulo $2^m + 1$ and hence can be used as the kernel of an FNT of length $2m$. Then, all multiplications involved in the transform computation can be carried out as binary shifts. Equation (2.4) implies that N must divide $2^{t+\hat{k}+1}$ when F_t is composite, i.e. for $t > 4$. In particular it can be verified that $\hat{k}$ is zero for F_5, F_6, and F_7, i.e. the k_i's in the factorisations of these numbers are all odd (see page 12 and [28, page lxxxviii]). Thus for F_5, F_6 and F_7 the maximum transform length is $2^{t+2} = 4m$.

A suitable kernel of a $4m$-length transform is an integer that has 2 as its square. Such an integer exists if the congruence $x^2 \equiv 2 \pmod{F_t}$ has a solution. By the definition of quadratic residues in Section A.2, the integer 2 is then called a *quadratic residue* modulo F_t. The least positive solution x to the mentioned congruence is often denoted $\sqrt{2}$ in the literature. The following theorem says that there really exists such a solution x.

[7] In general, we have $\gcd(k_1, k_2, \ldots, k_l) = k' \cdot 2^{\hat{k}}$ for some k' and $\hat{k}$, but here we are only interested in the cases when the transform length is a power of two.

Theorem 2.4 *The integer 2 is a quadratic residue modulo each Fermat number F_t for $t \geq 2$.*

Proof: From the proof of Theorem 2.3, given in Section A.2, we know that the integer 2 is a quadratic residue modulo every odd prime factor p_i of the Fermat number $F_t = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ for $t \geq 2$. Then, 2 is also a quadratic residue modulo $p_i^{n_i}$ (see for example Stewart, [95, Prop. A.13]). By Proposition A.10 of [95] we then get that the integer 2 is a quadratic residue modulo F_t for $t \geq 2$. □

The square of the element $\sqrt{2}$ can be expressed in the following way:

$$\begin{aligned}\left(\sqrt{2}\right)^2 &= 2 = (-1)\cdot 2 \cdot (-1) \equiv -2 \cdot 2^m = 2^{\frac{m}{2}}(-1 + 1 - 2 \cdot 2^{\frac{m}{2}}) \\ &\equiv \left(2^{\frac{m}{4}}\right)^2 \left((2^{\frac{m}{2}})^2 + (-1)^2 + 2(-1 \cdot 2^{\frac{m}{2}})\right) \\ &= \left(2^{\frac{m}{4}}(2^{\frac{m}{2}} - 1)\right)^2 \pmod{2^m + 1},\end{aligned}$$

and thus we get

$$\sqrt{2} \equiv 2^{\frac{m}{4}}(2^{\frac{m}{2}} - 1) \equiv 2^{\frac{3m}{4}} + 2^{\frac{5m}{4}} \pmod{2^m + 1}.$$

Powers of $\sqrt{2}$ can be written as

$$\left(\sqrt{2}\right)^n = \begin{cases} 2^{\frac{n}{2}}; & n \text{ even} \\ 2^{\frac{n-1}{2}}\sqrt{2} \equiv 2^{\frac{3m+2n-2}{4}} + 2^{\frac{5m+2n-2}{4}} \pmod{2^m + 1}; & n \text{ odd} \end{cases}, \quad (2.5)$$

which means that multiplication by powers of $\sqrt{2}$ can be implemented in VLSI as binary shifts when the exponent n is even, and two binary shifts and one addition when the exponent is odd. This is the reason why the element $\sqrt{2}$ is practically always used as the kernel of the FNT of length $4m$ in $\mathbb{Z}_{2^m+1}$. It can easily be shown that the order of $\sqrt{2}$ modulo F_t is $4m$ for $t \geq 2$ [2, App. C].

Because we have $4m = 2^m = N_{\max}$ for $m = 4$, the kernel $\sqrt{2}$ will yield the maximum length FNT in $\mathbb{Z}_{F_2}$. The same kernel will also yield the maximum length FNT in $\mathbb{Z}_{F_t}$ for $t = 5$, 6, and 7. However, in several applications the transform length $4m$ is still relatively small. In general, one-dimensional prime field FNTs of length greater than $4m$ require nontrivial multiplications. For a maximum length FNT ($N = 2^m$) in a Fermat prime field, the transform kernel must be a *primitive element*.

t	$m = 2^t$	$F_t - 1 = 2^m$	N for $\omega = 2$	N for $\omega = \sqrt{2}$	N for $\omega = \alpha$
0	1	2	**2**	—	—
1	2	4	**4**	—	**4**
2	4	16	8	**16**	**16**
3	8	256	16	32	**256**
4	16	65 536	32	64	**65 536**
5	32	4 294 967 296	64	**128**	—
6	64	$2^{64} \approx 1.8 \times 10^{19}$	128	**256**	—

Table 2.2: *Some parameters for the FNT. The boldfaced numbers are the maximum obtainable transform lengths. The kernel α is any primitive element modulo F_t.*

Every primitive element of a prime field $\mathbb{Z}_p$ has maximum order $p-1$ modulo p.[8] In Chapter 7 we find use of the following property:

Theorem 2.5 *The integer 3 is a primitive element of each Fermat prime field $\mathbb{Z}_{F_t}$ where $t \geq 1$.*

Proof: See Appendix A.3. □

Remark: Cunningham (see [36, page 199]) noted that for $t \geq 2$, the integers $3, 5, 6, 7, 10,$ and 12 are all primitive elements of the field of integers modulo a Fermat prime F_t for $t \geq 2$.

By Theorem 2.5 the maximum length FNT in a Fermat prime field can be computed using the primitive element 3 as transform kernel. Table 2.2 shows the relations between some kernels and their corresponding FNT lengths for the seven first Fermat numbers.

For each primitive element $\alpha \in \mathbb{Z}_{2^m+1}$, where 2^m+1 is a prime, we have $\alpha^{2^m} = (\alpha^{2^{m-b}})^{2^b} \equiv 1 \pmod{2^m+1}$. Because the order of the element $\alpha^{2^{m-b}}$ modulo 2^m+1 equals 2^b, it may be chosen as the kernel of an FNT of arbitrary length $N = 2^b$ for $0 \leq b \leq m$. This is further discussed in Section 7.2.7.

As previously mentioned, we would like to calculate the radix-2 FNT with as low complexity and high performance as possible for *every* such transform

[8] In general, if α and $q > 0$ are relatively prime integers such that $\mathrm{ord}_q\, \alpha = \phi(q)$, where ϕ denotes Euler's totient function, then α is called a *primitive root modulo q*.

length $N = 2^b$. Hence, we would like its approximately $N \log_2 N$ additions together with its $(N/2) \log_2 N$ multiplications by powers of the kernel to be carried out as simply as possible. In the present section we do not go into detail about what we mean by 'simple'. Complexity issues are further discussed in Chapter 4.

One purpose of our work is to find suitable ways of representing the binary coded integers of $\mathbb{Z}_{2^m+1}$, in order to simplify the arithmetic operations (especially multiplication by powers of the transform kernel) involved in the computation of the FNT of every possible length $N = 2^b$. We are particularly interested in the rings for which $2^m + 1$ is a prime, i.e. the Fermat prime fields.

2.3.3 Butterfly Computations

The Radix-2 Decimation-In-Time Algorithm

We mentioned above that the FNT of length $N = 2^b$ can be computed using a radix-2 FFT algorithm. When using the well known *decimation-in-time* algorithm, which is due to Cooley and Tukey [35], the FNT of the form in (2.1) is first split into two parts as follows.[9]

$$\begin{aligned} X_k &\equiv \sum_{n=0}^{N-1} x_n \omega^{kn} = \sum_{n \text{ even}} x_n \omega^{kn} + \sum_{n \text{ odd}} x_n \omega^{kn} \\ &= \sum_{r=0}^{N/2-1} x_{2r}\, \omega_{N/2}^{kr} + \omega^k \cdot \sum_{r=0}^{N/2-1} x_{2r+1}\, \omega_{N/2}^{kr} \\ &= G_k + \omega^k \cdot H_k \pmod{F_t}, \qquad k = 0, 1, \ldots, N-1, \end{aligned}$$

where G_k and H_k are the $N/2$-point FNTs of the sequences $\{x_{2r}\}_{r=0}^{N/2-1}$ and $\{x_{2r+1}\}_{r=0}^{N/2-1}$, respectively. The order of the kernel $\omega_{N/2} \triangleq \omega^2$ modulo F_t is $N/2$. Because $\omega^{N/2} \equiv -1 \pmod{F_t}$ we have $\omega^{k+N/2} \equiv -\omega^k \pmod{F_t}$ and thus the FNT can be expressed as

$$\begin{aligned} X_k &\equiv G_k + \omega^k \cdot H_k \pmod{F_t} \\ X_{k+N/2} &\equiv G_k - \omega^k \cdot H_k \pmod{F_t}, \end{aligned}$$

[9] The derivation of the decimation-in-time FFT algorithm can be found in most books on digital signal processing, e.g. [74, Ch. 9.3.3].

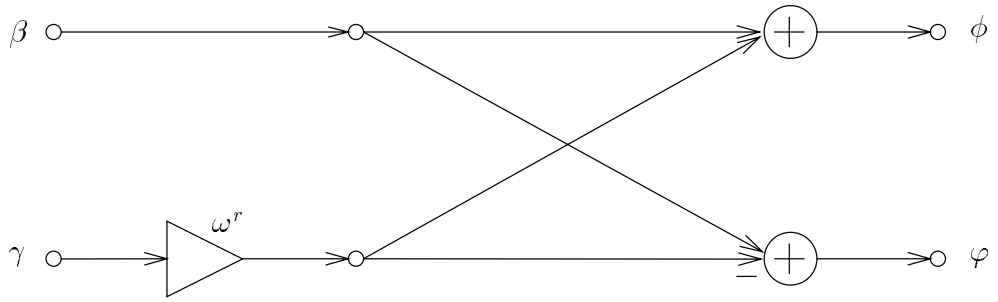


Figure 2.1: *Butterfly of a radix-2 decimation-in-time FFT.*

for $k = 0, 1, \ldots, N/2 - 1$. The name decimation-in-time is due to the decimation of x_n by a factor of 2. A repeated decimation of the sequences $\{x_{2r}\}$ and $\{x_{2r+1}\}$ will result in four $N/4$-point FNTs after the second step, eight $N/8$-point FNTs after the third step and so on, until we end up in $N/2$ 2-point FNTs after step $\log_2 N-1$. Thus, the computation of the FNT of length $N = 2^b$ may be carried out in $\log_2 N$ stages, where each stage consists of $N/2$ 2-point FNTs [74, Fig. 9.14]. Hence, the FNT can be computed as $(N/2)\log_2 N = (N/2)b$ FNTs of length 2. Figure 2.1 illustrates how such a basic 2-point FNT is computed. Because of the flow graph symmetry of the 2-point transform, it is usually called a *butterfly*. The two output signals from the decimation-in-time butterfly of Figure 2.1 are

$$\phi \equiv \beta + \omega^r \gamma \pmod{F_t}$$

$$\varphi \equiv \beta - \omega^r \gamma \pmod{F_t},$$

for some r and where β and γ are the butterfly inputs. Because each butterfly involves two additions and one multiplication, the total number of additions modulo F_t equals $N \log_2 N$ and the total number of multiplications modulo F_t equals $(N/2)\log_2 N$, as we have previously indicated.[10]

When the FFT algorithm is used for computing the ordinary DFT, the real and imaginary parts of the factors $\omega^r = e^{-j2\pi r/N}$, which are often called the *twiddle factors*, are usually stored in a table. This yields the fastest algorithm, to the cost of a look-up table. Concerning the FNT, by choosing a suitable kernel for the transform it may not be necessary to store the different powers of the kernel modulo F_t. For example, for $\omega = \sqrt{2}$ and $N = 4m = 2^{t+2}$ multiplication by powers of ω can be carried out as two binary shifts and one addition, as mentioned in connection with (2.5). For such kernels the b-bit exponents r may be

[10] Subtraction is regarded as addition, because it can be carried out by adding the minuend to the negated subtrahend (see Section 5.1.3).

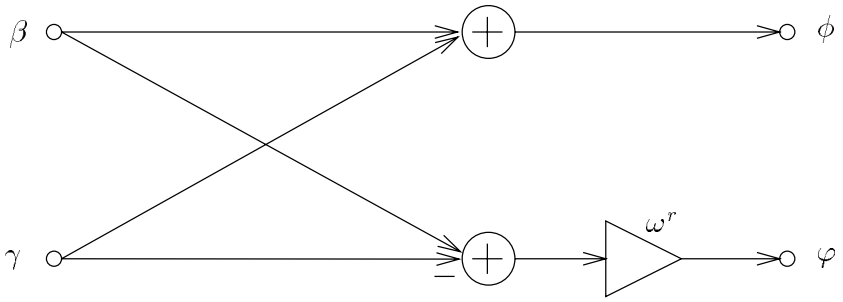


Figure 2.2: *Butterfly of a radix-2 decimation-in-frequency FFT.*

generated by some control logic for small transform lengths [101]. For larger transform lengths, the exponents are preferably stored in a table [90]. However, if there is no suitable kernel ω for which multiplication by powers of ω can be carried out simpler than the procedure for general multiplication, then we may still want a table of the twiddle factors involved in the computation of the transform.

The Radix-2 Decimation-In-Frequency Algorithm

When using the decimation-in-time FFT algorithm, the input sequence must appear in a bit-reversed order [74, Ch. 9.3.3]. The transformed sequence is, however, obtained in natural order. Using the radix-2 *decimation-in-frequency* FFT algorithm, we have the opposite situation. Then the input occurs in the right order while the output is obtained in bit-reversed order. The decimation-in-frequency algorithm is obtained by repeatedly divide the transform into two transforms, one which depends on the first half of the sequence and the other depending on the second half of the sequence. This algorithm is due to Gentleman and Sande [45].

As for the decimation-in-time algorithm, the decimation-in-frequency algorithm also divides the N-length transform into $\log_2 N$ stages of $N/2$ butterflies. Figure 2.2 shows the butterfly for the decimation-in-frequency algorithm.

For the butterfly input variables β and γ, we have the output variables

$$\phi \equiv \beta + \gamma \pmod{F_t}$$

and

$$\varphi \equiv (\beta - \gamma)\omega^r \pmod{F_t},$$

for some r.

The computations in both the decimation-in-time and decimation-in-frequency algorithms are done *in place*, which means that the same memory locations that hold the N elements of the sequence $\{x_n\}$ can be used to store the results of the butterfly computations at each of the $\log_2 N$ stages. Also, both algorithms involve $(N/2)\log_2 N$ butterfly operations, each consisting of one multiplication by a twiddle factor and two additions. The two algorithms can be arranged such that both the input and output sequences are maintained in natural order. However, the resulting algorithms are no longer in-place algorithms, which implies that additional memory is required.

Remark: Because of the similarity between the FNT and its inverse transform, they can be computed using the same FFT algorithm. The two transforms differ only in the factor $1/N$ and the sign of the exponent of ω.

Radix-4 Algorithms

If $b = \log_2 N$ is even we have $N = 4^{b/2}$ and thus the transform can be computed using a radix-4 FFT algorithm. Such an algorithm can be obtained by repeatedly dividing the input sequence into four parts in a manner that is similar to the procedure for deriving a radix-2 algorithm [74, Ch. 9.3.4]. The radix-4 FFT algorithm consists of $b/2$ stages of $N/4$ butterflies. The four outputs, say $\phi_0,\ \phi_1,\ \phi_2$, and ϕ_3, of a decimation-in-time butterfly can be expressed in matrix form as

$$\begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} \equiv \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{N/4} & (\omega^{N/4})^2 & (\omega^{N/4})^3 \\ 1 & (\omega^{N/4})^2 & (\omega^{N/4})^4 & (\omega^{N/4})^6 \\ 1 & (\omega^{N/4})^3 & (\omega^{N/4})^6 & (\omega^{N/4})^9 \end{pmatrix} \begin{pmatrix} \omega^0\beta_0 \\ \omega^r\beta_1 \\ \omega^{2r}\beta_2 \\ \omega^{3r}\beta_3 \end{pmatrix} \pmod{F_t},$$

for some r and where $\beta_0,\ \beta_1,\ \beta_2$, and β_3 and the four butterfly input data. Because the order of $\omega^{N/4}$ modulo F_t is 4 we get the congruences $(\omega^{N/4})^2 \equiv (\omega^{N/4})^6 \equiv -1 \pmod{F_t}$, $(\omega^{N/4})^3 \equiv -\omega^{N/4} \pmod{F_t}$, $(\omega^{N/4})^4 \equiv 1 \pmod{F_t}$, and $(\omega^{N/4})^9 \equiv \omega^{N/4} \pmod{F_t}$. In order to reduce the number of additions, the butterfly is usually derived from the following factorised twiddle-factor matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{N/4} & \omega^{2N/4} & \omega^{3N/4} \\ 1 & \omega^{2N/4} & \omega^{4N/4} & \omega^{6N/4} \\ 1 & \omega^{3N/4} & \omega^{6N/4} & \omega^{9N/4} \end{pmatrix} \equiv$$

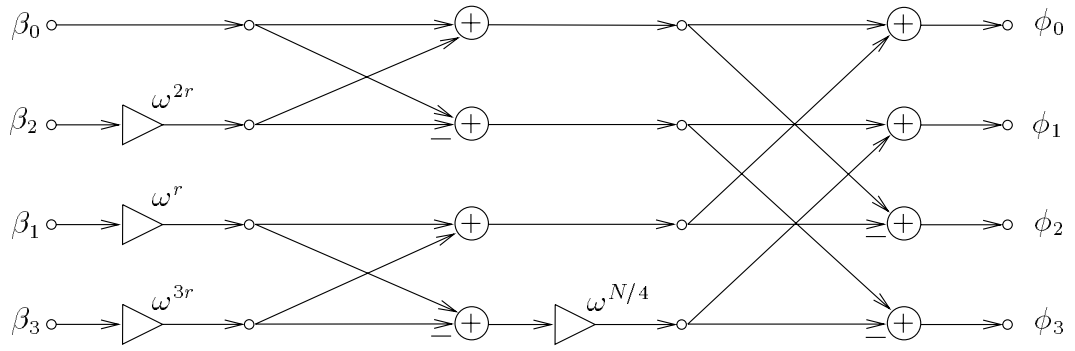


Figure 2.3: *Butterfly of a radix-4 decimation-in-time FFT.*

$$\equiv \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & \omega^{N/4} \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -\omega^{N/4} \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} \pmod{F_t}.$$

A radix-4 decimation-in-time butterfly is shown in Figure 2.3. The two-stage structure of the butterfly is due to the factorisation of the twiddle-factor matrix. Note that the input is in bit-reversed order, because then the computations can be carried out in place.

For the ordinary DFT, which has kernel $\omega = e^{-j2\pi/N}$, we have $\omega^{N/4} = -j$ (see [74, Eq. 9.3.44]).

Let $\vartheta \equiv \omega^{N/4} \pmod{F_t}$. It can be proved that for prime $F_t \geq 5$, the four incongruent solutions to the congruence $\vartheta^4 \equiv 1 \pmod{F_t}$ are ± 1 and $\pm 2^{m/2}$ modulo F_t. By Theorem 8.8 in [84] there are $\phi(4) = 2$ incongruent integers of order 4 modulo a prime F_t. Obviously, the integers

$$\begin{aligned} \vartheta_0 &\equiv 2^{m/2} \pmod{F_t} \\ \vartheta_1 &\equiv -2^{m/2} \equiv 2^{3m/2} \pmod{F_t} \end{aligned}$$

are these two incongruent integers. In particular, we see that for the FNTs of length $N = 2m$ and $N = 4m$ and with kernels $\omega = 2$ and $\omega \equiv \sqrt{2} \pmod{F_t}$, respectively, we have $\vartheta \equiv \omega^{N/4} \equiv 2^{m/2} \pmod{F_t}$.

We showed earlier that for composite F_t, the maximum radix-2 FNT length is at least $4m = 2^{t+2}$. [11] Because the order of 2 modulo F_t is $2m = 2^{t+1}$ it

[11] For most composite F_t with at least one factor known, the maximum length is *exactly* $4m$.

follows that for all transform lengths $N = 2^b$, where $0 \leq b \leq t+1$, we have $\mathrm{ord}_{F_t}\, 2^{2m/N} = N$. Therefore, for every such transform length there exists a kernel which is a power of two.

Hence, by choosing a suitable kernel ω, the radix-4 butterfly multiplication by $\omega^{N/4}$ can simply be carried out as some binary shifts modulo F_t for every Fermat number F_t and every possible transform length in $\mathbb{Z}_{F_t}$. Therefore, using *three* general multiplications and eight additions modulo F_t per butterfly, a radix-4 FNT can be computed using $3 \cdot (N/4) \cdot \log_4 N = (3N/8) \log_2 N$ multiplications and $8 \cdot (N/4) \cdot \log_4 N = N \log_2 N$ additions modulo F_t.

Compared with the radix-2 FNT algorithm, the radix-4 algorithm requires 25% less multiplications but the same number of additions, i.e. we get the same complexity reduction as is obtained for the "ordinary" radix-4 DFT (see [74, Ch. 9.3.4]).

By using appropriate decimating procedures, it is also possible to define fast algorithms for radix-r transforms for $r > 4$. These algorithms are quite similar to the radix-2 and radix-4 algorithms, and they do not result in a significant reduction of the number of arithmetic operations. Therefore, they are not considered here.

The Split-Radix Algorithm

The *split-radix* algorithm, which is due to Duhamel and Hollman [40], [41], is presently the most efficient radix-2 FFT algorithm. The decimation-in-frequency algorithm is derived by using a radix-2 decomposition of the even-indexed terms and a radix-4 decomposition of the odd-indexed terms. In the first stage, the even-indexed terms are inputs to a radix-2 transform of length $N/2$ and the odd-indexed terms are again decomposed into two sequences of length $N/4$, which becomes the inputs of two radix-4 transforms. The even-indexed terms are given by

$$X_{2k} \equiv \sum_{n=0}^{N/2-1} \left(x_n + x_{n+N/2}\right) \omega^{2kn} \pmod{F_t},$$

for $k = 0,\, 1, \ldots,\, N/2-1$ and the two radix-4 transforms are given by

$$X_{4k+1} \equiv \sum_{n=0}^{N/4-1} \left[\left(x_n - x_{n+N/2}\right) + \omega^{N/4}\left(x_{n+N/4} - x_{n+3N/4}\right)\right] \omega^n \omega^{4kn}$$

and

$$X_{4k+3} \equiv \sum_{n=0}^{N/4-1} \left[\left(x_n - x_{n+N/2}\right) - \omega^{N/4}\left(x_{n+N/4} - x_{n+3N/4}\right)\right] \omega^{3n} \omega^{4kn},$$

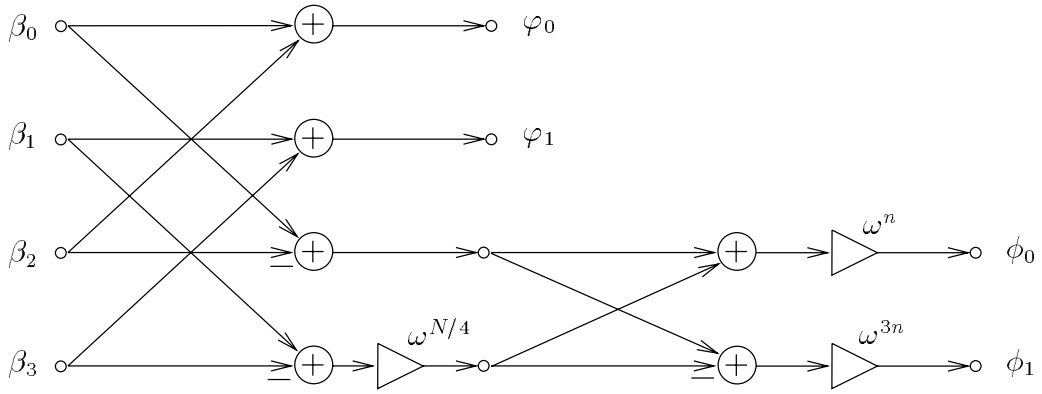


Figure 2.4: *Butterfly of a split-radix decimation-in-frequency FFT.*

for $k = 0,\ 1,\ 2, \ldots,\ N/4 - 1$ and where both congruences are reduced modulo F_t. As shown on page 21, in $\mathbb{Z}_{F_t}$ the factor $\omega^{N/4}$ equals some power of two. Thus, using a binary coded element representation, multiplication by $\omega^{N/4}$ can be carried out as binary shifts modulo F_t. Figure 2.4 shows a butterfly of a split-radix decimation-in-frequency FFT.

In the first stage of the algorithm, the input variables $\beta_0,\ \beta_1,\ \beta_2$, and β_3 are $x_n,\ x_{n+N/4},\ x_{n+N/2}$, and $x_{n+3N/4}$, respectively, for some n. The output variables φ_0 and φ_1 are used to calculate some of the even-indexed terms of the transformed sequence, and ϕ_0 and ϕ_1 are used to calculate the terms with odd indices of the forms $4k + 1$ and $4k + 3$, respectively, for some k.

Because the split-radix algorithm is a kind of mixture of a radix-2 and a radix-4 FFT, it does not progress stage by stage. Therefore, the indexing will be more complicated compared with for example a fixed-radix FFT algorithm. It has been shown that a split-radix FFT can be computed using in the order of $(N/3)\log_2 N$ multiplications and $N \log_2 N$ additions for great transform lengths N (see for example Proakis et al. [75, Ch. 2.14] or Skodras and Constantinides [94]).

As seen above, the only arithmetic operations that are involved in the computation of the FNT and its inverse transform are addition, subtraction (i.e. negation followed by addition), multiplication by powers of the transform kernel, and multiplication by powers of two modulo F_t. In this thesis we mainly focus on these arithmetic operations and others that may be needed in connection with the transform computation. Examples of such operations are general multiplication, the discrete logarithm, and exponentiation modulo F_t. We do not care about which FFT algorithm is used (radix-2, radix-4, split-radix, or

Integer	Normal binary coded repr.
2^m	$1000 \cdots 000$
$2^m - 1$	$0111 \cdots 111$
$2^m - 2$	$0111 \cdots 110$
$2^m - 3$	$0111 \cdots 101$
.	.
.	.
.	.
3	$0000 \cdots 011$
2	$0000 \cdots 010$
1	$0000 \cdots 001$
0	$0000 \cdots 000$

Table 2.3: *The normal binary coded integer representation.*

any other). We are only interested in the arithmetic operations involved in the computation of the transform.

2.4 Element Representation

We mentioned in Section 2.1 that we represent the elements of the Fermat integer quotient rings $\mathbb{Z}_{2^m+1}$ as *binary coded integers* and use binary logic circuits in the VLSI architectures for the arithmetic operations in $\mathbb{Z}_{2^m+1}$. It is clear that $m+1$ bit positions are needed to represent the 2^m+1 elements of $\mathbb{Z}_{2^m+1}$. Thus, there are

$$\underbrace{2^{m+1} \cdot (2^{m+1}-1) \cdot (2^{m+1}-2) \cdots (2^m+1) \cdot 2^m}_{2^m+1 \text{ factors}} = \frac{2^{m+1}!}{(2^m-1)!}$$

different ways of representing these elements. The very well known normal binary coded representation of integers is illustrated in Table 2.3.

This representation, however, may not be the best one with respect to the complexity and performance of the VLSI architectures for arithmetic operations in $\mathbb{Z}_{2^m+1}$. Depending on how complexity and performance are defined, it may require a great effort to find the 'optimum' representation among the $2^{m+1}!/(2^m+1)!$ possible ones, e.g. there are about 2×10^{23} ways to represent the 5-bit binary coded integers of $\mathbb{Z}_{2^4+1}$. We therefore choose to restrict our-

selves to consider a subset of representations that can be expressed as elementary functions of the normal binary coded representation.

The first form of representation considered is the normal binary coded representation. In Chapter 5 we study VLSI architectures for arithmetic operations using this representation. Linear coordinate transformations of the normal binary coded representation and the corresponding VLSI architectures are considered in Chapter 6. Finally, in Chapter 7 we particularly focus on the polar representation, which can be regarded as a nonlinear coordinate transformation of the normal binary coded representation.

Chapter 3

Applications

The Fermat number transform (FNT) is one of the most useful and powerful number theoretic transforms. As mentioned in Chapter 1, in the beginning of the 1970's the interesting properties of the FNT attracted several researches. In this chapter we describe some of the main applications of the FNT. In particular, we consider digital convolution and correlation in Fermat integer quotient rings and Reed-Solomon codes over Fermat prime fields.

There are also other applications of the FNT. Siu and Constantinides [87] have shown that the number of multiplications required to compute the discrete Fourier transform can be reduced by using number theoretic transforms. In [88] they particularly consider the FNT for reducing the complexity of computing the discrete Fourier transform. Truong et al. [102] later considered the computation of the discrete Fourier transform using the FNT in a quadratic residue Fermat number system. Several other researchers have also studied the computation of the discrete Fourier transform using number theoretic transforms.

Boussakta and Holt have shown that the discrete Hartley transform can be calculated using the FNT [20, 21]. In [22], the same authors showed how to compute the Walsh-Hadamard transform using the FNT and vice versa. Two decades ago, Rader [78] discussed number theoretic transforms for use in a block-mode image filtering scheme. A microprocessor-based architecture for block-mode image filters using the FNT was later implemented in VLSI by Shakaff et al. [90].

Boussakta et al. [23] showed that the FNT of periodic data has a regular structure with many transform components equal to zero. Any small imperfection in the periodic data significantly changes the high regularity of its FNT. As a consequence of the results in [23], the authors conclude that the FNT is highly applicable in areas like for example the detection of errors in maskmaking for integrated circuit design and defect detection in industrial inspection. They also suggest applications for image compression and data storage, where only the nonzero elements of the FNT of periodic data need to be stored together with their locations.

3.1 Convolution and Correlation of Real Integer Sequences

Discrete convolution and correlation are two very common operations in digital signal processing (see for example Blahut [17]). The cyclic convolution of two sequences $\{x_n\}_{n=0}^{N-1}$ and $\{h_n\}_{n=0}^{N-1}$ is given by the sum

$$y_n = \sum_{k=0}^{N-1} x_k h_{n-k \,(\mathrm{mod}\, N)} \; ; \qquad n = 0, 1, \ldots, N-1 \tag{3.1}$$

Correlation and convolution are computationally equivalent. The cross-correlation of two sequences $\{x_n\}$ and $\{h_n\}$ is obtained by convolving $\{x_n\}$ with $\{h_{-n}\}$.

Like the discrete Fourier transform the FNT also has the cyclic convolution property, i.e. the transform of a cyclic convolution of two sequences is equal to the product of their transforms. Because the method of computing the convolution sum using transform calculations is often faster than the direct computation of the sum, the procedure is sometimes called *fast convolution*. The method is particularly efficient when the sequence length is highly composite, because then some FFT algorithm can be applied to compute the transform.

It is often possible – and sometimes preferable – to let computations in one algebraic field be carried out in another field, which is then usually called a *surrogate* field. Depending on the application in question, this computational procedure may also apply to rings. A computation of interest where this is applicable is convolution via transform calculations. Using a computer or a digital signal processor, these calculations are often carried out in the complex field $\mathbb{C}$, i.e. the discrete Fourier transform is used. However, if the sequences that are to be convolved consist of *real integers*, the convolution can instead be computed in an integer quotient ring $\mathbb{Z}_q$, for some suitable modulus q [2].

There are some advantages of computing the transforms in $\mathbb{Z}_q$ rather than in the complex field: A complex multiplication requires *several* real multiplications while a multiplication in $\mathbb{Z}_q$ is a *single* and often simpler operation (integer multiplication). The computation precision is also improved since computations in a finite ring are exact. Another very important consequence of the simplified arithmetic is that, depending on q, the complexity and performance of the hardware implementation of a transform in $\mathbb{Z}_q$ can be smaller than the complexity and performance of the corresponding implemented transform in $\mathbb{C}$.

The modulus q must be chosen such that every element x_n, h_n, and y_n, for $n = 0, 1, \ldots, N-1$, is contained in the ring $\mathbb{Z}_q$. Because of the congruence relation modulo q in the ring $\mathbb{Z}_q$, negative integers are represented as positive integers, in accordance with the congruence $-x \equiv q - x \pmod{q}$.

In the following example we illustrate how discrete cyclic convolution of real integers can be computed in an integer quotient ring.

Example 3.1 If the convolution of two positive real integer sequences $\mathbf{x}$ and $\mathbf{h}$ are to be carried out in the surrogate field $\mathbb{Z}_q$, then the greatest integer in the convolution sum must be less than the modulus q, i.e. q must not exceed the dynamic range of y_n (and x_n and h_n). For $\mathbf{x} = \{4, 11, 7, 4\}$ and $\mathbf{h} = \{9, 0, 8, 14\}$, by (3.1) we can compute the convolution $\mathbf{y} = \{246, 229, 151, 180\}$. The prime modulus $q = 257 = 2^8 + 1$ is greater than the maximum value of $\mathbf{y}$. Consequently, this convolution can be carried out in $\mathbb{Z}_{2^8+1}$. Furthermore, because the sequences involved have length 4, which divides $257 - 1 = 256$, the convolution $\mathbf{y}$ can be obtained by using FNT calculations in $\mathbb{Z}_{2^8+1}$.

Because the order of the integer 16 is 4 modulo $2^8 + 1$, it can be chosen as the kernel ω of an FNT of length $N = 4$. Thus, the 4-point FNT of $\mathbf{x}$ is

$$X_k \equiv \sum_{n=0}^{3} x_n 16^{kn \bmod 4} \pmod{257}; \qquad k = 0, 1, 2, 3$$

with a similar relation for the transform of $\mathbf{h}$. Using matrix notations we have

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} \equiv \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 16 & 256 & 241 \\ 1 & 256 & 1 & 256 \\ 1 & 241 & 256 & 16 \end{pmatrix} \begin{pmatrix} 4 \\ 11 \\ 7 \\ 4 \end{pmatrix} \equiv \begin{pmatrix} 26 \\ 109 \\ 253 \\ 142 \end{pmatrix} \pmod{257}$$

and

$$\begin{pmatrix} H_0 \\ H_1 \\ H_2 \\ H_3 \end{pmatrix} \equiv \begin{pmatrix} 31 \\ 34 \\ 3 \\ 225 \end{pmatrix} \pmod{257}.$$

Each component Y_k of the FNT of y is then obtained by multiplying X_k by H_k modulo 257, which gives

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} \equiv \begin{pmatrix} 35 \\ 108 \\ 245 \\ 82 \end{pmatrix} \pmod{257}.$$

Regarding the inverse transform we need to know N^{-1} and ω^{-1}. From the congruences $4 \cdot 193 \equiv 1 \pmod{257}$ and $16 \cdot 241 \equiv 1 \pmod{257}$ we get $N^{-1} = 4^{-1} \equiv 193 \pmod{257}$ and $\omega^{-1} = 16^{-1} \equiv 241 \pmod{257}$, respectively. Hence, the inverse transform is

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \equiv 193 \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 241 & 256 & 16 \\ 1 & 256 & 1 & 256 \\ 1 & 16 & 256 & 241 \end{pmatrix} \begin{pmatrix} 35 \\ 108 \\ 245 \\ 82 \end{pmatrix} \equiv \begin{pmatrix} 246 \\ 229 \\ 151 \\ 180 \end{pmatrix} \pmod{257},$$

which agrees with the convolution y obtained when using the conventional convolution sum in (3.1). □

Let x be the input sequence of a linear time-invariant system with impulse response h. Let A be the dynamic range of x_n, i.e. we have $\mid x_n \mid \leq A$ for $n = 0, 1, \ldots, N-1$. If x_n can take on negative numbers, the convolution sum yields

$$\mid y_n \mid \leq A \sum_{k=0}^{N-1} \mid h_k \mid \leq \frac{q-1}{2},$$

and thus

$$A \leq \frac{q-1}{2 \sum_{k=0}^{N-1} \mid h_k \mid}.$$

If A is also the dynamic range of h_n and the computations are carried out in $\mathbb{Z}_{2^m+1}$, we get $\mid h_k \mid \leq A$, $q = 2^m + 1$, and $N = 2^b$. Thus, we have

$$A \leq \sqrt{\frac{2^m + 1 - 1}{2 \cdot 2^b}} = 2^{\frac{m-b-1}{2}}$$

b	m					
	2	4	8	16	32	64
1	1	2	8	128	2^{15}	2^{31}
2	0	1	4	64	2^{14}	2^{30}
3	–	1	4	64	2^{14}	2^{30}
4	–	0	2	32	2^{13}	2^{29}
5	–	–	2	32	2^{13}	2^{29}
6	–	–	1	16	2^{12}	2^{28}
7	–	–	1	16	2^{12}	2^{28}
8	–	–	0	8	–	2^{27}
9	–	–	–	8	–	–
10	–	–	–	4	–	–
11	–	–	–	4	–	–
12	–	–	–	2	–	–
13	–	–	–	2	–	–
14	–	–	–	1	–	–
15	–	–	–	1	–	–
16	–	–	–	0	–	–

Table 3.1: *The dynamic range of x_n and h_n, for which the corresponding sequences are of length $N = 2^b$. In $\mathbb{Z}_{2^m+1}$ we have $0 \leq b \leq m$ for $m = 1, 2, 4, 8,$ and 16, and $2^b \leq 4m$ for $m = 32$ and $m = 64$.*

which implies that the maximum dynamic range is

$$A = \lfloor 2^{\frac{m-b-1}{2}} \rfloor,$$

i.e. the greatest integer less than or equal to $2^{(m-b-1)/2}$. Table 3.1 shows the dynamic range of x_n and h_n for some values of m and b. Because of the relatively poor dynamic range for small m, digital filtering of real integer sequences is generally considered to be applicable primarily for $m \geq 32$.

A common situation in filtering applications is the filtering of a relatively long sequence by an FIR filter of much shorter length. This involves a linear convolution of great length which can be impractical to compute. There exist, however, two well known techniques that simplify the computation of great-length linear convolutions: Using the *overlap-add method* or the *overlap-save method*, the longer sequence is sectioned into shorter length subsequences that are cyclically convolved with the impulse response [79, Ch. 2.25]. Truong et al. [103, 107] have devised a general overlap-save method for filters of arbitrary length using the Fermat number transform.

An important application of the cyclic convolution property is multiplication of (large) integers. Let u and v be two m-bit normal binary coded integers, i.e. $u = \sum_{n=0}^{m-1} u_n 2^n$ and $v = \sum_{n=0}^{m-1} v_n 2^n$ where $u_n, v_n \in \mathbb{Z}_2$. The procedure for multiplying u by v is equivalent to the convolution $(u * v)_n$. The direct convolution requires in the order of m^2 bit operations. If m is a power of 2, this complexity can be reduced to approximately $m^{\log_2 3}$ bit operations by using the Karatsuba-Ofman algorithm [55], [4, Ch. 2.6].

The most efficient algorithm for multiplication of large m-bit integers, where m is a power of two, is due to Schönhage and Strassen [86]. The algorithm multiplies two m-bit normal binary coded integers u and v, where $m = 2^t$. The output is the $(m+1)$-bit product of u and v modulo the Fermat number $F_t = 2^m + 1$. The product is computed using the FNT in $\mathbb{Z}_{F_s}$ for $s = (t+3)/2$ if t is odd and for $s = (t+2)/2$ if t is even. The algorithm, which requires in the order of $m \cdot \log_2 m \cdot \log_2(\log_2 m)$ bit operations , is described in English by Aho et al. in [4, Ch. 7.5].

3.2 Decoding of Reed-Solomon Codes

Denote by $GF(q)$ an algebraic finite field of order q.[1] The *Galois field Fourier transform* (GFT) can be regarded as a generalisation of the well known discrete Fourier transform. The GFT of the vector $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{N-1})$ over $GF(q)$ is the vector $\mathbf{V} = (V_0, V_1, V_2, \ldots, V_{N-1})$, where

$$V_j = \sum_{i=0}^{N-1} v_i \omega^{ij} \; ; \qquad j = 0, 1, \ldots, N-1.$$

The transform kernel ω is an element of $GF(q^n)$ of order N, where N divides $q^n - 1$ for some positive integer n, see for example Blahut [16, Def. 8.1.1]. The inverse GFT of $\mathbf{V}$ is given by

$$v_i = N^{-1} \sum_{j=0}^{N-1} V_j \omega^{-ij} \; ; \qquad i = 0, 1, \ldots, N-1,$$

where the multiplicative inverse N^{-1} is computed modulo the characteristic p of the field $GF(q)$. Each transform component V_j is an element of $GF(q^n)$.

[1] The only finite field we have considered so far is the prime field $GF(p)$, where p is a prime number. Because the set $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ of integers modulo the prime p forms the prime field $GF(p)$ under addition and multiplication modulo p, the prime field of order p is often denoted $\mathbb{Z}_p$.

Let $\mathbf{C} = (C_0, C_1, C_2, \ldots, C_{N-1})$ be the GFT of a Reed-Solomon codeword $\mathbf{c} = (c_0, c_1, c_2, \ldots, c_{N-1})$ of length N. For Reed-Solomon codes the exponent n in the GFT computation equals one, i.e. the transform kernel ω is an element of $GF(q)$. The codeword polynomial $c(x) = \sum_{i=0}^{N-1} c_i x^i$, which is associated with the codeword $\mathbf{c}$, has $2t$ consecutive powers of ω as its roots, where t is the number of errors that can be corrected by the code. Thus, we have

$$c(\omega^{u+l}) = \sum_{i=0}^{N-1} c_i (\omega^{u+l})^i = C_{u+l \pmod N} = 0$$

for some u and $l = 0, 1, \ldots, 2t-1$. Consequently, each cyclically contiguous transform component $C_{u+l \pmod N}$ equals zero. This property may be used to construct Reed-Solomon codes in the transform domain: The encoder first sets $2t$ consecutive[2] components of $\mathbf{C}$ equal to zero. The remaining $K = N - 2t$ positions of $\mathbf{C}$ are filled with message symbols. Next, the resulting transform is inverted to produce the desired codeword $\mathbf{c}$. Depending on the choice of q, N, and K, this procedure may yield a computational complexity that is smaller than the complexity of the 'direct' computation of the codeword, i.e. by means of polynomial multiplication in the time domain.

The decoding procedure at the receiver's end may also take place in the transform domain (see Blahut [16, Ch. 8–9]). The receiver first computes the GFT vector $\mathbf{R}$ from the received vector $\mathbf{r} = \mathbf{c} + \mathbf{e}$, where $\mathbf{e}$ is an error vector of length N. In the transform domain we have the relation $\mathbf{R} = \mathbf{C} + \mathbf{E}$. The transmitted codeword $\mathbf{c}$ can be obtained as the inverse GFT of $\mathbf{C} = \mathbf{R} - \mathbf{E}$. When the encoding take place in the transform domain, the message symbols may be obtained directly from $\mathbf{C}$. Because the encoder has set $2t$ consecutive positions of $\mathbf{C}$ equal to zero, $\mathbf{E}$ equals $\mathbf{R}$ in these positions. The $2t$ corresponding components of $\mathbf{R}$ are called the *syndromes* of $\mathbf{r}$. If not more than t errors have occurred, the remaining $N - 2t$ unknown components of $\mathbf{E}$ can be recursively computed from the syndromes using for example the Berlekamp-Massey algorithm.

When q is a Fermat prime and n equals one, the FNT in the prime field $\mathbb{Z}_{F_t}$; $t = 1, 2, 3, 4$ is obtained as a special case of the GFT. Justesen [54] was among the first researchers to consider Reed-Solomon codes over Fermat prime fields. He stated that the decoding complexity of such codes can be reduced if the FNT is used to calculate the syndromes.

Reed, one of the originators of the Reed-Solomon codes, have coauthored several articles concerning fast decoding of Reed-Solomon codes using the FNT.

[2] Or cyclically contiguous.

For example, in [80], Reed et al. show how to use the FNT and continued fractions in the decoding procedure. In [82], Reed et al. conclude that a decoder for Reed-Solomon codes of length 2^{t+4} over $GF(F_t)$ using an FNT is simpler than corresponding decoders for a code of length $2^t - 1$ using a GFT in $GF(2^t)$. Liu et al. [63] considered Reed-Solomon codes over $GF(F_3)$ for use in space communication applications. In a recent article by Shiozaki et al. [91], the authors consider a Reed-Solomon code as a special case of a redundant residue polynomial code. They present a fast algorithm for decoding Reed-Solomon codes over $GF(F_t)$ using the FNT and the Euclidean algorithm.

Chapter 4

The VLSI Model

We use complementary metal-oxide-semiconductor (CMOS) circuits in the VLSI architectures presented in this thesis. The CMOS technology offers high packing density, high yield, wide noise margin, low power dissipation, and low cost. Because of these attractive properties, CMOS has become one of the most important VLSI technologies of today (see for example Weste and Eshraghian [113, Ch. 1]). The VLSI model adopted in this thesis (and defined in the present chapter) is only valid for CMOS and nMOS circuits.

In integer quotient rings, all arithmetic operations involve modulus reduction. When performing modulus reduction of a binary coded integer, the value in each bit position of the reduced binary coded integer may depend on the value in *every* bit position of the original binary coded integer. Therefore, depending on the modulus, bit-serial architectures are often impracticable for arithmetic operations in integer quotient rings. This particularly applies to integer arithmetic operations modulo a Fermat number $2^m + 1$. Most of the architectures presented in the subsequent chapters are based on bit-parallel transmission and processing of the data. The main exceptions are the bit-serial/parallel multipliers in Sections 5.1.5, 6.3.6, and 7.6.6 and the bit-serial multipliers in Sections 7.6.5 and 7.6.6.

4.1 Introduction

In the VLSI circuit design process it is important to consider aspects like floorplanning and interconnections. These aspects play a major role when minimising parameters like clock skew, noise, and power dissipation (see the book of Bakoglu [14]). Over the years, two of the main goals for integrated circuit designers have been to minimise the area and maximise the performance of the implemented circuits. During the last years, there has also been an increasing interest in low-power digital CMOS design, see Chandrakasan et al. [30, 31] and Liu [61]. One reason for this is the increasing number of portable equipment requiring low power. Another reason is that the scaling of digital CMOS circuits results in a higher power consumption.

In Chapters 5, 6, and 7 we investigate different architectures for arithmetic operations. These architectures are mutually compared mainly with respect to their area complexity and time performance. The chip area occupied by the corresponding implemented circuit is denoted by A and the time required to perform the operation is denoted by T. In order to take both chip area and computation time into account, we also consider the area-time performance AT^2 of each architecture. The AT^2 performance is a cost function to be minimised. Thompson [100] is one of the originators of this area-time performance measure. In his paper of 1979 [100] Thompson proposed a VLSI model of computations. Based on this model, he derived a lower bound N^2 on the AT^2 performance of computing the discrete Fourier transform of length N, i.e $AT^2 = \Omega\left(N^2\right)$ for such a computation.[1] Brent and Kung [26] also did some basic works on VLSI models and complexity. They derived the lower bound $AT^{2\alpha} = \Omega\left(N^{1+\alpha}\right)$, for $0 \leq \alpha \leq 1$, on the performance of N-bit binary multiplication. A survey of computational algorithms and their VLSI implementation is given by Ullman [104]. For example, in Chapter 2 of [104], Ullman gives an introduction to the area of AT^2 performance.

As indicated above, two very important steps in the VLSI design process are floorplanning and the routing of interconnections and communication paths. Interconnections usually occupy a large part of the chip, typically more than fifty percent of the total chip area. The placement of the different modules of the chip is crucial to the interconnection delay. The wire lengths between modules and within each module should be as small as possible in order to get a small interconnection delay.

[1] By $a(m) = \mathcal{O}\left(b(m)\right)$ (or "$a(m)$ is $\mathcal{O}\left(b(m)\right)$") we mean that, for increasing m, the function $a(m)$ does not grow faster than the function $b(m)$. The notation $a(m) = \Omega\left(b(m)\right)$ (or "$a(m)$ is $\Omega\left(b(m)\right)$") is used to bound the growth rate of $a(m)$ from below. The notations $\mathcal{O}$ and Ω are conventionally used in the area of VLSI complexity, see for example Ullman [104].

The choice of VLSI model differ between researchers. The modelling of the chip area is quite uncontroversial. The main difference lays in the modelling of the interconnection delay. The time for a signal to propagate along a wire of length l is usually modelled as either $\mathcal{O}(1)$ (synchronous model), $\mathcal{O}(\log l)$ (capacitive model), $\mathcal{O}(l)$ (transmission line model), or $\mathcal{O}(l^2)$ (RC model), see Bilardi et al. [15] and Bakoglu [14, Ch. 5-6]. The capacitive model, adopted by for example Thompson, is appropriate for short wires and the RC model for long wires. It is, however, common to divide long wires into shorter subsections using repeaters (buffers). These repeaters have the effect of reducing the interconnection delay from $\mathcal{O}(l^2)$ to $\mathcal{O}(l)$, see for example Bakoglu [14, Ch. 5.4.2].

Because device dimensions are getting smaller and chips are getting larger, the lengths of on-chip wires are increasing. Therefore, the interconnection delay is more and more becoming a major factor when determining the overall circuit performance. In this thesis we assume that interconnection delays within each module are $\mathcal{O}(1)$, i.e we adopt the synchronous model. For *large* systems, the synchronous model gives a gross simplification of the true interconnection delay. Because in general the architectures studied here do *not* involve global routing (interconnections between modules on the chip) our delay estimations should not, however, considerably deviate from the true intramodular delays.

If we go a couple of steps further in the design process and consider the implemented circuit, then it is simpler to estimate the delays caused by the wiring. It is also simple to estimate the true interconnection delay after the floorplanning and routing steps of the design process. One way of estimating the average lengths of the chip interconnections is to partition the circuit design into different sections and calculate the number of connections between the sections. The average lengths can then be modelled by using Rent's rule, which is described by, for example, Bakoglu [14, Ch. 9.8.1].

Adopting the synchronous model for the interconnection delays does not mean that we disregard the wiring effects. Our effort is to design architectures with a high degree of regularity and with wires only connecting neighbouring gates.

In general, the architectures presented in this thesis do not contain logic circuits for generating control signals. For example, every clock signal is assumed to be available wherever needed and without any clock skew involved.

The phrase "low power" can be found in many current publication titles. Several aspects of low power digital CMOS design can be found in the recently published PhD thesis by Liu [61]. For example, in his thesis Liu considers low

power CMOS device design, low power circuit and system techniques, and power estimations in digital CMOS VLSI chips.

In this thesis, we do not give estimates of the power dissipated by the investigated circuits. However, with a low-power design strategy in mind, we often follow the guidelines suggested by Liu [61] and others when choosing clocking strategy and combinational logic circuits.

4.2 Complexity and Performance

4.2.1 The Delay Model

Over the years, the linear switch-level RC model for CMOS transistors has been adopted by many researchers when investigating the timing properties of digital VLSI circuits. We refer to the articles by Ousterhout [70] and Rubinstein et al. [85], and Chapter 1 in Mead's and Conway's book [66]. A linear switched RC model for the nMOS transistor is shown in Figure 4.1.

For the sake of simplicity, all nMOS and pMOS transistors are modelled to have the same characteristics, e.g. they have equal size and the pull-down and pull-up times of the nMOS and pMOS transistors, respectively, are the same. The RC model for the pMOS transistor is similar to the nMOS transistor model in Figure 4.1. When a transistor is off, the switch is open and the transistor acts only as a capacitive load to the rest of the circuit. The variables C_g, C_d, C_s, and R_0 are the gate, drain, and source capacitances, and the channel resistance, respectively.

A consequence of modelling the non-linear MOS transistors of a circuit as linear switched RC circuits is that the estimated circuit delays are more or less erroneous. A circumstance which is often neglected is the fact that the transistor capacitances and the channel resistance are actually functions of voltage. Inaccuracies in delay computations may also occur because of the difficulties in including input waveform effects. Nevertheless, for most RC models the delay estimations do not deviate more than 20 percent from SPICE simulations, see the articles by Ousterhout [70], Sundblad and Svensson [96], and Hedenstierna and Jeppson [49].

In this thesis we adopt the Penfield-Rubenstein model [85] in which the input voltages are modelled as step waveforms and transistors are modelled as the transistor in Figure 4.1. The delay calculation of a circuit is based on Elmore's

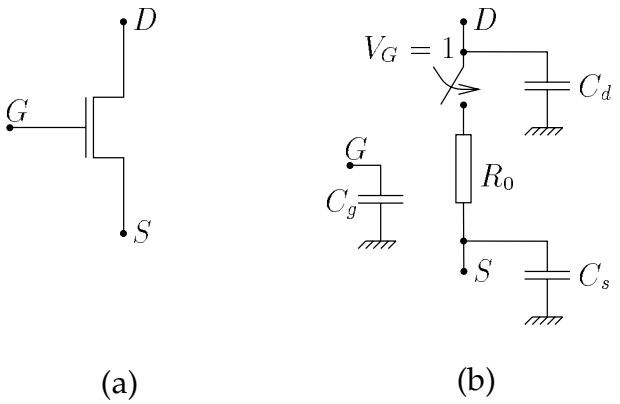


Figure 4.1: *Model of an nMOS transistor.* (a) *Symbolic description.* (b) *A switched linear RC model of the transistor. The transistor is switched on when the gate voltage V_G is high.*

delay model for an RC tree without side branches, i.e. an RC chain [43], [85]. For example, the Elmore delay T_d from the input to the output of the RC chain in Figure 4.2 equals

$$\begin{aligned} T_d &= \sum_{i=1}^{4} \left(\sum_{j=1}^{i} R_j \right) C_i \\ &= R_1C_1 + (R_1 + R_2)C_2 + (R_1 + R_2 + R_3)C_3 + (R_1 + R_2 + R_3 + R_4)C_4, \end{aligned}$$

i.e. *each capacitor contributes to the delay as the product of the capacitance and the total resistance between the capacitor and the signal source (or ground)*[2]. The delay is defined as the time from the 50-percent level of the input signal waveform to the 50-percent level of the output signal waveform.

Without considering wire capacitances, the capacitive loads in different nodes of most of today's digital CMOS combinational logic circuits are dominated by gate capacitances [113, Ch. 4.3.4], [49]. The main reason for this is that the number of gates connected to a node is often several times greater than the number of drains and sources connected to the node. Furthermore, according to Weste and Eshraghian [113, Ch. 4.3.4] and others, the gate capacitance is

[2] Each capacitor is assumed to be charged (or discharged) through all resistors between the capacitor and the signal source (ground).

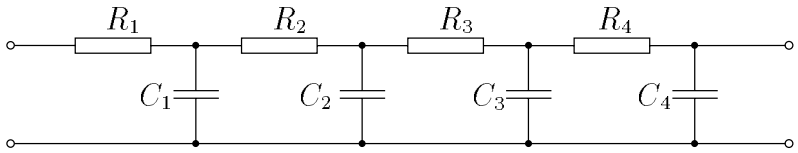


Figure 4.2: *An RC chain.*

typically several times greater than the drain and source capacitances. Due to these facts and in order to obtain a measure of time complexity on a simple form, we generally disregard the effects of the drain and source capacitances on the delays. Thus, the only capacitances and resistances that are involved in our delay computations are gate capacitances and transistor channel resistances, respectively.

4.2.2 Area and Time Complexities

Denote by A the chip area occupied by an implemented circuit. Because the CMOS technology changes so fast, it is essential to have a measure of the chip area that is technology-independent. The area complexities of the architectures considered in this thesis are given in terms of the *sizes* of the architectures.

Definition 4.1 *The* size *of an architecture is the number of CMOS transistors that form the architecture and is denoted by $\mathcal{C}$.*

Consequently, with equally sized transistors the chip area A occupied by the implemented circuit, not including the wire area, is proportional to $\mathcal{C}$. If the total circuit area is to be determined, the circuit interconnections must also be considered. Even though the area occupied by these interconnections is *not* considered here, we still strive to design modular and regular architectures in order to reduce the interconnection area and simplify the interconnection work.

The clock frequency of a circuit is related to the length of its *critical path*. The critical path is the longest path along which signals are pulling up and pulling down circuit transistors, or propagating through them, during one clock interval. The critical path usually starts and ends with a clocked latch or flip-flop. Thus, the minimum clock cycle time of a circuit is proportional to the length of

its critical path. Suppose the circuit needs m clock cycles to perform its operation. Then, the computation time is proportional to m times the critical path length.

When determining the critical path, we use the same strategy as the one used in the timing verification program Crystal, which is described by Ousterhout in [70]. The circuit to be examined is decomposed into chains of transistors called *stages*. A stage runs from the supply voltage source or ground through a number of transistors to the gate inputs of some other transistors.

Definition 4.2 *Let s denote a certain stage of a circuit and let T_{s} denote the delay of that stage. Then, the length $\mathcal{L}_{\mathrm{s}}$ of stage s is the ratio of T_{s} and the time constant $R_0 C_g$, i.e.*

$$\mathcal{L}_{\mathrm{s}} = \frac{T_{\mathrm{d}}}{R_0 C_g},$$

where R_0 and C_g are the linearised MOS transistor channel resistance and gate capacitance, respectively.

The delay of a stage is calculated as Elmore's delay of the RC chain model of the stage. The critical path through a circuit is formed by an ordered set of stages, where each stage gives a separate contribution to the total circuit delay.

Definition 4.3 *The length $\mathcal{L}_{\mathrm{CP}}$ of the critical path (CP) equals the sum of the lengths of the stages that forms the critical path.*

One of the transistors in each stage is called the *trigger*. The trigger is the last transistor to turn on in a stage.

Consider, as an example, the circuit in Figure 4.3. This circuit has no relevance, except for being an example. The CP through the circuit is the ordered set $\{s_1,\, s_2,\, s_3,\, s_4\}$ of stages. These stages, which become active for $in = 0$, are signified by dotted lines in the figure. The RC circuits in the bottom of the figure correspond to the equivalent RC models of the four stages $s_1,\, s_2,\, s_3$, and s_4. The total delay T_{d} of the circuit is approximately equal to the sum

$$T_{\mathrm{d}} = T_1 + T_2 + T_3 + T_4 = R_1 \cdot 2C_g + 2R_0 \cdot 3C_g + R_0 C_g + (R_2 + R_0) \cdot nC_g, \quad (4.1)$$

where $T_1,\, T_2,\, T_3$, and T_4 are the delay contributions of the stages $s_1,\, s_2,\, s_3$, and s_4, respectively and where n is the *fan-out* of the circuit, i.e. the number of transistor gates that are driven by the circuit output signal. The trigger of stage $s_2/s_3/s_4$ is the transistor in the end of stage $s_1/s_2/s_3$, respectively. By Definitions 4.2 and 4.3, the length of the CP through the circuit in Figure 4.3 equals $\mathcal{L}_{\mathrm{CP}} = T_{\mathrm{d}}/R_0 C_g$.

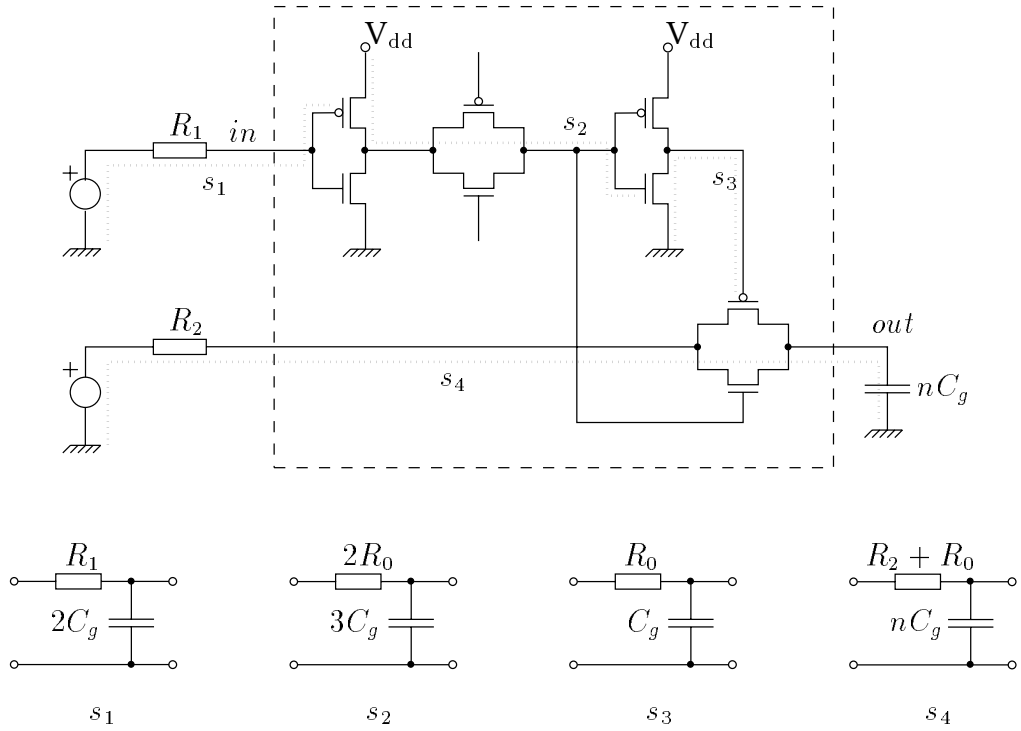


Figure 4.3: *An example of a circuit (the one within the dashed box) that has CP length $\mathcal{L}_{\text{CP}} = 2r_1 + 7 + n(r_2 + 1)$ and size $\mathcal{C} = 8$. Here, the output load is strictly capacitive; n is the circuit fan-out. The RC equivalent circuits of the stages s_1, s_2, s_3, and s_4 are shown in the bottom of the figure.*

Definition 4.4 *We define the* normalised resistance r *of a resistor with resistance R as the ratio*

$$r = \frac{R}{R_0},$$

where R_0 is the linearised MOS transistor channel resistance.

By letting $R_1 = r_1 R_0$ and $R_2 = r_2 R_0$ we get, from (4.1), the CP length $\mathcal{L}_{\text{CP}} = 2r_1 + 7 + n(r_2 + 1)$ for the circuit in Figure 4.3. Hence, the time from the rising/falling of the input signal to the rising/falling of the output signal is proportional to $\mathcal{L}_{\text{CP}} = 2r_1 + 7 + n(r_2 + 1)$. We also note that the circuit has size $\mathcal{C} = 8$, because it comprises 8 transistors.[3] The area-time performance AT^2 of the circuit is proportional to $\mathcal{C}\mathcal{L}_{\text{CP}}^2 = 8 \cdot (2r_1 + 7 + n(r_2 + 1))^2$.

[3] In this example we do not consider the area of the two resistors R_1 and R_2.

The architectures of this thesis mainly comprise basic building blocks like inverters, transmission gates, 2-input gates, adder elements, and registers. In the following Section 4.3 we present the sizes and time performances of such CMOS logic circuits, with respect to the adopted delay model. The size of an architecture was defined above as the number of nMOS and pMOS transistors that form the circuit. We describe the time performance of an architecture in terms of its *fan-in*, *internal CP length*, and *output normalised resistance*:

Definition 4.5

1. *The* fan-in f *of a circuit is the number of transistor gates of the circuit that are driven by the circuit input signal.*
2. *An* internal stage *of a circuit is a stage whose associated RC model does not depend on how the circuit is connected to other circuits. The* internal CP length $\mathcal{L}_{\mathrm{CP}}$ *is the sum of the lengths of the internal stages.*
3. *The* output normalised resistance r *is the total normalised resistance from the circuit output node back to the supply voltage source (or ground).*

The length of the CP is used as a measure of time performance. When characterising the CP through a circuit, it is partitioned into three parts, in accordance with Definition 4.5-*1*, *2*, and *3*:

1. The first part of the CP is the input stage of the circuit. The delay of the input stage depends on the *total* capacitance $n_{\mathrm{in}} C_g$ at the input node, where n_{in} is the fan-out of the preceding circuit that has this input stage as its output stage. Henceforth, we describe the input stage of a circuit in terms of its contribution to n_{in}, i.e. we only state the fan-in f of the circuit.
For example, the fan-in of the circuit in Figure 4.3, whose input stage is s_1, equals 2.

2. The second part of the CP is the set of internal stages, which is described by the internal CP length.
For example, the internal stages of the circuit in Figure 4.3 are s_2 and s_3. Their respective lengths equal $2 \cdot 3 = 6$ and $1 \cdot 1 = 1$. Hence, the internal CP length of the circuit equals $6 + 1 = 7$.

3. The third part of the CP is the output stage. The output normalised resistance determines, together with the subsequent resistive and capacitive loads of the output stage, the length of the stage. The circuit contribution to this length is described in terms of its output normalised resistance.
For example, the output stage of the circuit in Figure 4.3 is s_4 and its output normalised resistance equals $r_2 + 1$.

Remark: There are architectures in Chapters 5, 6, and 7 for which the delays of some stages are proportional to m, where m is the exponent of 2 in the Fermat number $2^m + 1$. These delays are generally due to the fact that single logic gates or inverters are driving large capacitive loads. For example, if a logic gate with output normalised resistance r is driving m logic gates, each with fan-in equal to f, the delay of that stage is proportional to its length $r \cdot fm$. The traditional way of reducing the delay of a stage with a large capacitive load is to properly buffer the stage by using a number of cascaded drivers (inverters) of gradually increasing size. Then, the resulting total delay can be bounded to be proportional to $\log m$, see Mead an Conway [66, Sec. 1.5]. Note, however, that *regarding the architectures in Chapters 5, 6, and 7, we generally do not consider the problem of driving large capacitive loads*.

4.3 Basic CMOS Building Blocks

In this section we derive the sizes, fan-ins, internal CP lengths, and output normalised resistances of the inverter, the transmission gate, the two-input multiplexer, two-input gates, the single-bit adder, and the register (D flip-flop), with respect to the VLSI model defined in the previous section. In Section 4.3.6, these parameters are all listed in a table.

4.3.1 The Inverter and the Transmission Gate

The Inverter

Because the CMOS inverter comprises two MOS transistors, its size equals $\mathcal{C}_{\text{inv}} = 2$. The inverter is shown in Figure 4.4. When the inverter input signal changes from high to low, the stages marked by the dashed lines in Figure 4.4(b) are activated. For a low-to-high input signal transition, the stages marked by the dotted lines are activated. Because the two possible input stages, as well as the two output stages, are actually equivalent, the inverter contribution to the CP is simply its fan-in, which equals $f_{\text{inv}} = 2$, and its output normalised resistance $r_{\text{inv}} = 1$. There is no internal stage.

In Figure 4.4(c), n_{in} is the total number of transistor gate inputs that are connected to the inverter input node and n is the fan-out of the inverter. Furthermore, r_{in} is the output normalised resistance of the circuit prior to the inverter.

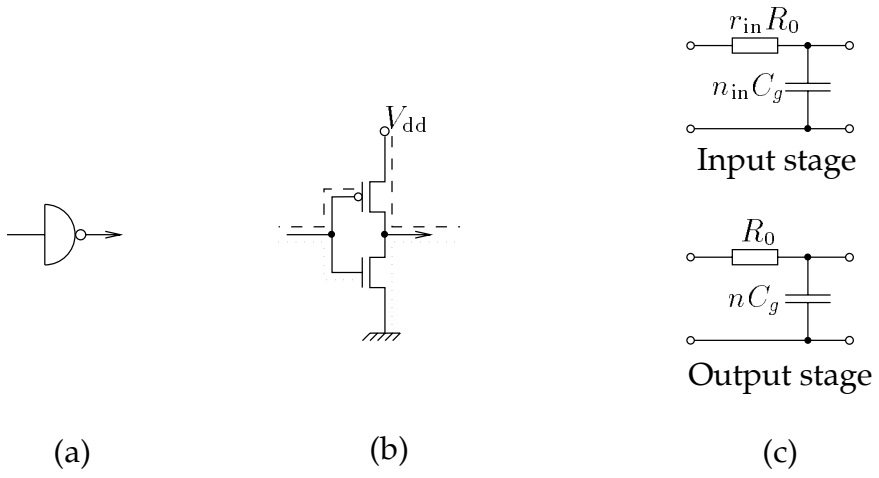


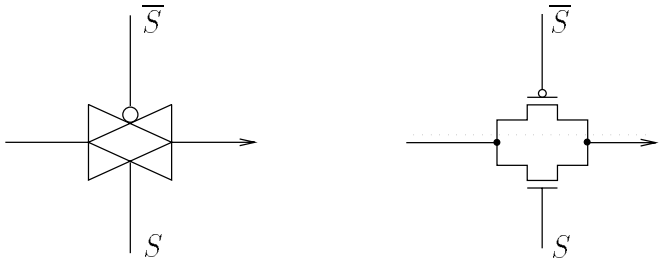
Figure 4.4: *A CMOS inverter.* (a) *Symbolic description.* (b) *Schematic description. The CP is formed either by the dotted or the dashed stages.* (c) *Simple RC equivalents of the stages.*

The Transmission Gate

The transmission gate has the same size $\mathcal{C}_{\text{TG}} = 2$ as the inverter. Figure 4.5 shows how the transmission gate is formed by an nMOS and a pMOS transistor in parallel. The dotted path in Figure 4.5(b) is the output stage of the CP. This stage is also the output stage of a preceding circuit whose output signal is the input signal of the transmission gate. The stage runs through *one* of the transmission gate transistors. Therefore, the output normalised resistance equals $r_{\text{TG}} = r_{\text{prior}} + 1$, where r_{prior} is the output normalised resistance of the mentioned circuit prior to the transmission gate. Note that because the transmission gate is not connected to the supply voltage source or ground, its equivalent pass transistor resistor is a *series* resistor (and not a Thevenin equivalent resistor).

If one of the transistors of the transmission gate is the trigger of the output stage, then the stage that ends up in the gate input of this transistor also belongs to the CP; it becomes the input stage. Then, the fan-in f_{TG} equals the fan-in of the trigger,[4] i.e. we have $f_{\text{TG}} = 1$. Otherwise, the fan-in equals zero. Like the inverter, the transmission gate has no internal stage.

[4] In accordance with Definition 4.5-*1*, by the fan-in of the trigger we mean the number of gates of a circuit that are driven by the signal on the gate of the trigger.



(a) (b)

Figure 4.5: *A transmission gate.* (a) *Symbolic description.* (b) *Schematic description. The dotted line is the output stage of the transmission gate.*

4.3.2 The Two-Input Multiplexer

The two-input multiplexer is simply constructed using two transmission gates, as shown in Figure 4.6.

Because the multiplexer comprises two transmission gates, each of size 2, the total size of the two-input multiplexer equals $\mathcal{C}_{\text{MUX}} = 4$. Like the output stage of the transmission gate, the multiplexer output stage (s_2 in Figure 4.6) is also the output stage of another circuit. Hence, the output normalised resistance of the multiplexer equals $r_{\text{MUX}} = r_{\text{prior}} + 1$, where r_{prior} is the output normalised resistance of the circuit prior to the multiplexer.

Furthermore, if the transmission gate transistor of stage s_1 is the trigger of stage s_2, stage s_1 also belongs to the CP. Then, the multiplexer fan-in equals $f_{\text{MUX}} = 2$, because the control signal S in stage s_1 controls two of the multiplexer transistors.[5] If s_1 does not belong to the CP, the multiplexer has no input stage and thus its fan-in equals zero. The internal CP length equals zero.

4.3.3 Two-Input Gates

NAND/NOR Gates

Schematic descriptions of the 2-input NAND and NOR gates are given in Figures 4.7(a_2) and (b_2) respectively. The NAND and NOR gates have equal size

[5] If $\overline{S}$ controls the trigger of the output stage, the fan-in f_{MUX} also equals two.

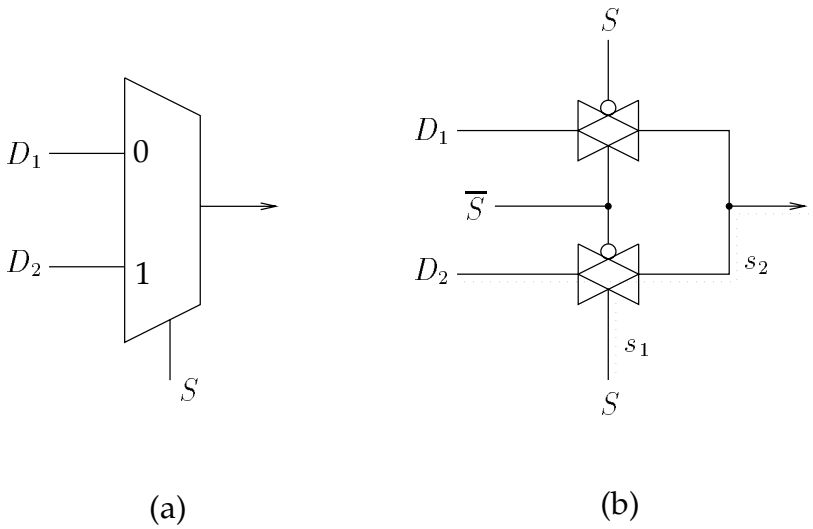


Figure 4.6: *A two-input multiplexer.* (a) *Symbolic description.* (b) *Schematic description. The dotted lines show the two stages of the CP when the signal* $S = 1$ *opens the trigger of stage* s_2.

$\mathcal{C}_{\text{NAND/NOR}} = 4$. With respect to the switch-level transistor model, the gate delays are also the same. The RC equivalents of the NAND and NOR gates are given in Figure 4.7(c). In the worst case delay, D_2 is the input signal of the trigger of stage s_2. Because each of the input signals controls the switching of two transistors, the fan-in equals $f_{\text{NAND/NOR}} = 2$ for both the NAND gate and the NOR gate. We also get the same output normalised resistance $r_{\text{NAND/NOR}} = 2$ for both gates. The NAND and NOR gates have no internal stage.

In a more realistic transistor model, the NAND gate is often preferable to the NOR gate. For example, if the gates are designed to have symmetric switching, the area occupied by the NAND gate is smaller than the area required for the NOR gate, see Uyemura [106, Ch. 6.5.3]. Conversely, for transistors of the same size, the rise-time and fall-time asymmetry is greater for the NOR gate than for the NAND gate.

AND/OR Gates

Two-input AND gates and OR gates are usually designed as NAND gates and NOR gates, respectively, each followed by an inverter. Thus, AND gates and OR gates have size $\mathcal{C}_{\text{AND/OR}} = 6$. The fan-in $f_{\text{AND/OR}}$ equals the fan-in $f_{\text{NAND/NOR}} = 2$ of the NAND (and NOR) gate and the output normalised re-

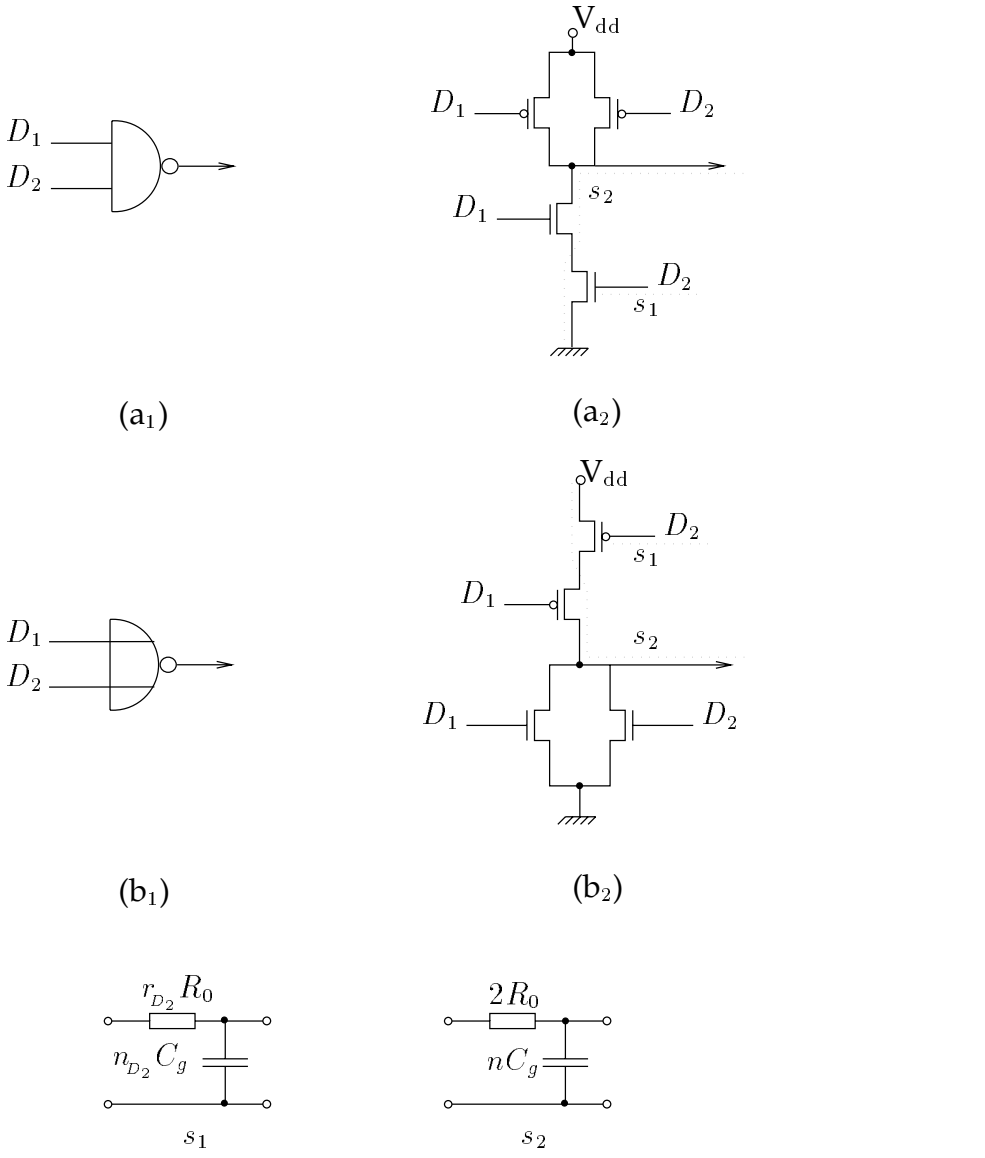


Figure 4.7: *Two-input NAND and NOR gates.* (a) *A NAND gate.* (b) *A NOR gate.* (c) *RC equivalents of the CP stages when there are no side branches or extended branches. The NAND gate and the NOR gate have similar CP stages, see the dotted lines in* (a_2) *and* (b_2).

sistance $r_{\text{AND/OR}}$ of the AND and OR gates equals the normalised resistance $r_{\text{inv}} = 1$ of the inverter. The internal CP length $\mathcal{L}_{\text{AND/OR}}$ equals the product of the output normalised resistance of the NAND (and NOR) gate and the inverter fan-in, i.e. we have $\mathcal{L}_{\text{AND/OR}} = r_{\text{NAND/NOR}} f_{\text{inv}} = 2 \cdot 2 = 4$.

XOR/XNOR Gates

The XOR gate can be designed in several ways. For example, a transmission gate-based XOR gate can be built with as few as six transistors [113, Fig. 8.11]. However, it may be rather difficult to track down CPs of circuits that contain such XOR gates. Therefore, we instead consider the realisation shown in Figure 4.8(b), which has size $\mathcal{C}_{\text{XOR}} = 12$. This gate contains more transistors than the transmission gate-based XOR gate, but it is quite easy to find the stages of its CP.

There are eight different stages in the XOR gate in Figure 4.8. Which ones will be activated depends on the input signals D_1 and D_2 and their last values. Among the 16 different transitions of the input signals that may occur, the one from $(D_1,\, D_2) = (1,\, 0)$ to $(D_1,\, D_2) = (0,\, 0)$ activates the stages s_1, s_2, and s_3 in the listed order. These stages, which are signified by the dotted lines in Figure 4.8(b), form a CP through the XOR gate.[6]

The fan-in of the XOR gate equals $f_{\text{XOR}} = 4$ and the normalised resistance of the output stage s_3 equals $r_{\text{XOR}} = 2$. The RC equivalent of the internal stage s_2 of the CP through the XOR gate is shown in Figure 4.8(c). The length $\mathcal{L}_{\text{XOR}}$ of stage s_2 equals 2.

The two-input XNOR gate can be constructed by interchanging the connections of γ and its binary inverse (i.e. its one's complement) $\overline{\gamma}$ in the rightmost part of the circuit in Figure 4.8(b). Consequently, the XNOR gate have the same size and delay characteristics as the XOR gate.

4.3.4 The Single-Bit Adder

Addition is a fundamental operation in all arithmetic processes. There are many ways to implement an m-bit binary adder. In general, it consists of a some *single-bit* full adder elements. A parallel m-bit adder can be formed by cascading m such adder element. Figure 4.9 shows the Karnaugh maps for the sum output σ and carry output c of the full adder element. The adder has three

[6] There are also other stages of the XOR gate whose lengths sum up to the length of the CP chosen.

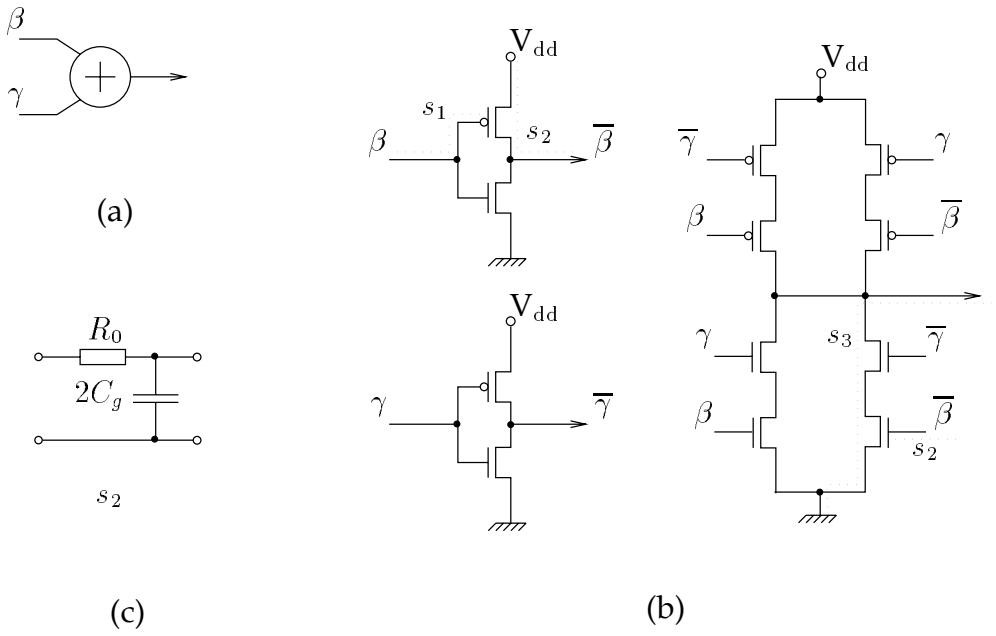


Figure 4.8: *A static two-input XOR gate.* (a) *Symbolic description.* (b) *Schematic description. The dotted lines show the three stages of a possible CP through the gate.* (c) RC *equivalent circuit of the internal stage* s_2 *of the CP.*

inputs; the signals β and γ and the carry input c_{in}. According to the Karnaugh maps, the carry and sum outputs of the full adder element can be expressed as the Boolean expressions

$$c = \beta\gamma + c_{\text{in}}(\beta + \gamma) \tag{4.2}$$

$$\sigma = \beta \oplus \gamma \oplus c_{\text{in}}, \tag{4.3}$$

respectively. The symbol $\oplus$ denotes the XOR function, i.e. addition modulo 2. Figure 4.10(a) shows the symbolic description of the single-bit full adder element.

There are various ways of implementing the full adder element. Here, we use the conventional static full adder element shown in Figure 4.10(b), which is based on the carry output Boolean function given by (4.2) and the sum output Boolean function

$$\sigma = \beta\gamma c_{\text{in}} + \overline{c}(\beta + \gamma + c_{\text{in}}),$$

which is obtained by rewriting (4.3). The delays of this adder element can easily be estimated when using the adopted switch-level RC delay model. Another advantage is that the adder outputs are driven by inverters. However,

σ

c_{in} \ $\beta\gamma$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

c

c_{in} \ $\beta\gamma$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Figure 4.9: *Karnaugh maps of the sum output σ and carry output c of the full adder element.*

compared with dynamic adders it has at least one disadvantage: From an investigation of various adder elements, Liu and Svensson [61, Paper 5] conclude that the power consumption of the static adder in Figure 4.10(b) is typically two to three times greater than the power consumption of dynamic full adder elements. The size of the chosen full adder, which equals $\mathcal{C}_{\text{FA}} = 28$, is comparable to the sizes of most other dynamic and static full adder elements.

There are 64 different input signal transitions of the adder element that may occur. From a CP search point of view, however, most of them are ruled out. Yuan and Svensson [109] propose two principles of determining the number of significant transitions. Firstly, the start stage of each transition should include as many transistors as possible. Secondly, the final stage should have as few transistors in parallel as possible. Using these principles, the number of interesting input transitions are reduced to 14 [109, Fig. 5]. When investigating these transitions, we have found that they all give rise to paths of the same lengths.

For example, one such CP is obtained when the input signals change from $(\beta,\ \gamma,\ c_{\text{in}}) = (1,\ 1,\ 1)$ to $(\beta,\ \gamma,\ c_{\text{in}}) = (0,\ 0,\ 1)$. If this transition occurs synchronously for the three adder inputs, the CP from the input to the *sum* output is equal to the set $\{s_1,\ s_2,\ s_5,\ s_6\}$ of stages, see the dotted lines in Figure 4.10(b). The internal signal $\overline{c}$ opens the trigger of stage s_5. Moreover, the CP from the input to the *carry* output is equivalent to the set $\{s_1,\ s_2,\ s_3\}$. If β opens the trigger of stage s_2, then s_1 is replaced by s_8 in the above sets of stages. However, because both γ and β drive eight of the full adder transistors, the fan-in of the trigger of stage s_2 will still be the same; $f_1 = f_8 = 8$.

If the trigger of stage s_5 is the transistor with gate input signal c_{in}, i.e. if c_{in} appears at the adder input later than the moment when the end node of stage s_2

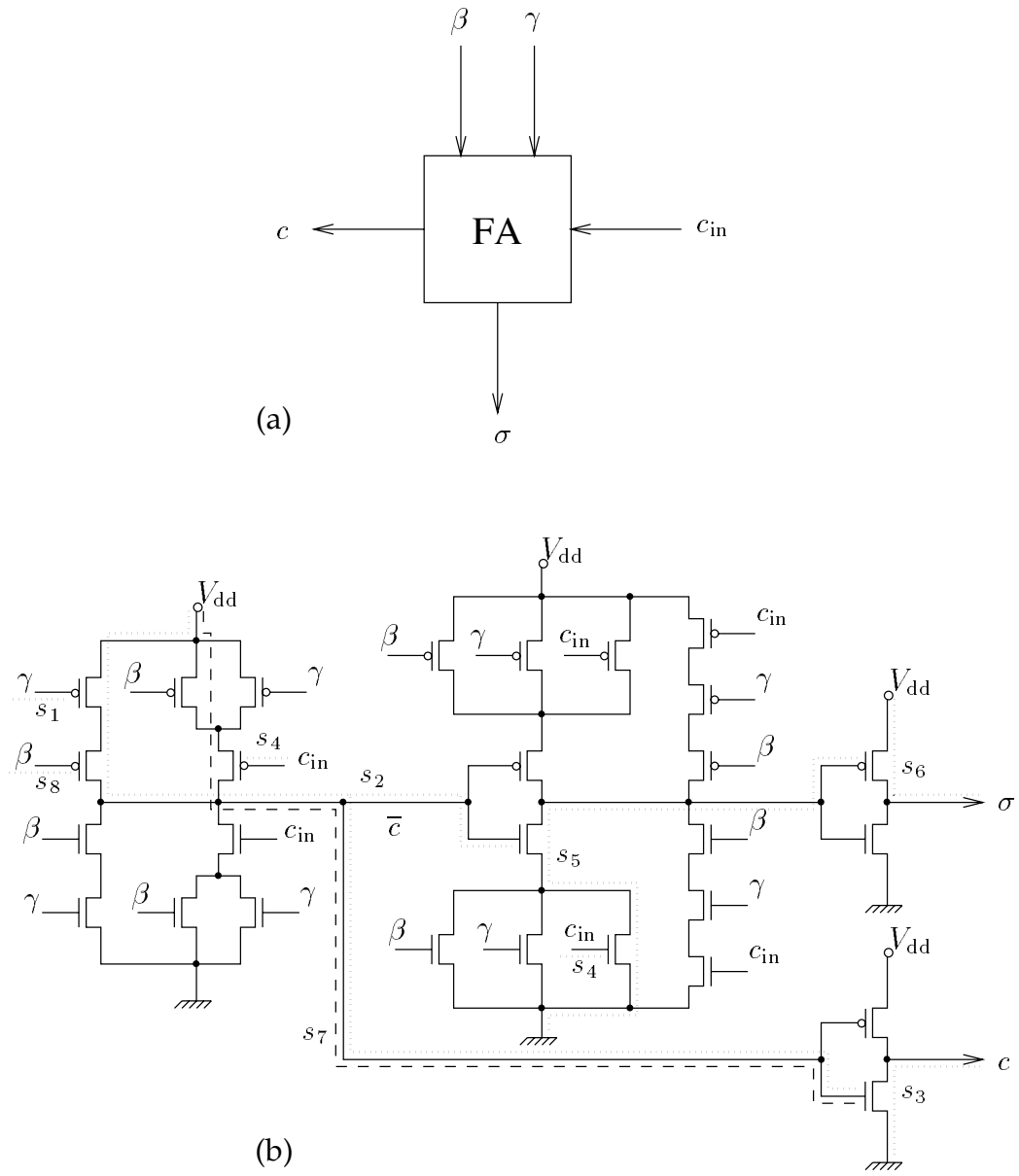


Figure 4.10: *The single-bit binary full adder element.* (a)*Symbolic description.* (b) *Schematic description. The dotted and the dashed lines are stages that form the different CPs of the adder element.*

is fully charged, then stage s_4 is included in the input-to-sum CP. This situation typically occurs in parallel adders, for which the CP is usually the carry chain through the full adder elements. Then, the CP from the carry input to the carry output is the set $\{s_4,\ s_7,\ s_3\}$, for which only one of the two parallel transistors (controlled by β and γ) in stage s_7 is switched on. Because the carry input c_{in} is connected to the gates of six transistors of the full adder, the fan-in of the trigger of stage s_5 (and of the trigger of stage s_7) equals $f_4 = 6$. Using the switch-level RC model, we obtain the lengths $\mathcal{L}_2 = 2 \cdot 4 = 8$, $\mathcal{L}_5 = 2 \cdot 2 = 4$, and $\mathcal{L}_7 = 2 \cdot 4 = 8$ of the internal stages s_2, s_5, and s_7, respectively.

From the above reasoning we get that the full adder fan-ins equal $f_{\text{FA,signal}} = 8$ and $f_{\text{FA,carry}} = 6$, with respect to the signal and carry input nodes, respectively. When the CP through the full adder leads to the sum output, the internal CP length equals $\mathcal{L}_{\text{FA,sum}} = \mathcal{L}_2 + \mathcal{L}_5 = 8 + 4 = 12$ and when it leads to the carry output, the internal CP length equals $\mathcal{L}_{\text{FA,carry}} = \mathcal{L}_2 = \mathcal{L}_7 = 8$. In both cases we get the same output normalised resistance $r_{\text{FA}} = 1$.

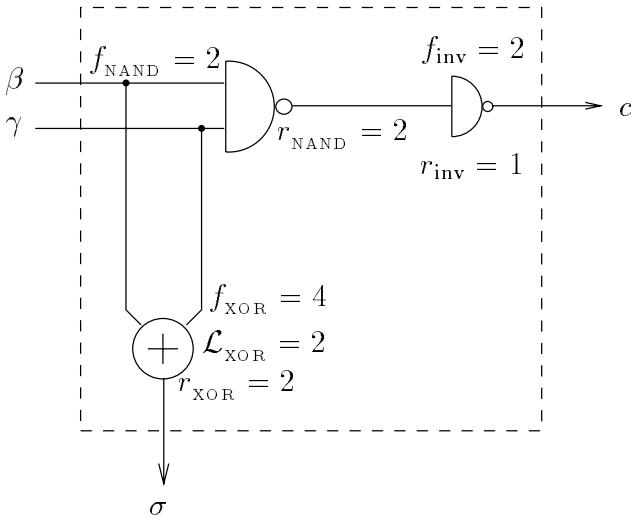
If one of the inputs, say the carry input, of the full adder element is always equal to zero, a *half adder* may be used instead of a full adder. Then, the sum and carry outputs of the half adder are the Boolean functions $\sigma = \beta \oplus \gamma$ and $c = \beta\gamma$, respectively. These functions may be directly implemented using one XOR gate for the sum output and one AND gate for the carry output, where the latter gate is realised as a NAND gate followed by an inverter.

The half adder element is depicted in Figure 4.11. The size of this half adder equals $\mathcal{C}_{\text{HA}} = \mathcal{C}_{\text{XOR}} + \mathcal{C}_{\text{NAND}} + \mathcal{C}_{\text{inv}} = 12 + 4 + 2 = 18$. Its CP delay parameters are shown in the bottom of the figure.

4.3.5 The Register

As for many other CMOS circuits, there are several ways of designing a register. Here, we consider the dynamic true single-phase clock master-slave D flip-flop depicted in Figure 4.12. This positive edge-trigged flip-flop is an extended version of a precharged inverting D flip-flop suggested by Yuan et al. [108]. The size of the flip-flop in Figure 4.12, which equals $\mathcal{C}_{\text{reg}} = 16$, is less than the sizes of ordinary static D flip-flops. Also, Liu and Svensson [61, Ch. 3.3], [99] found that the power consumption of this flip-flop is less than the power consumption of other known static and dynamic master-slave D flip-flops.

The flip-flop in Figure 4.12 has an asynchronous reset input. In some circuits, one may need settable registers and when using a plain bit-serial shift register there is no need for settable or resettable registers. There are also other types of



- Total fan-in: $f_{\mathrm{HA}} = f_{\mathrm{NAND}} + f_{\mathrm{XOR}} = 6$
- Internal length, input to carry output: $\mathcal{L}_{\mathrm{HA,carry}} = r_{\mathrm{NAND}} f_{\mathrm{inv}} = 4$
- Internal length, input to sum output: $\mathcal{L}_{\mathrm{HA,sum}} = \mathcal{L}_{\mathrm{XOR}} = 2$
- Normalised resistance of the carry output stage: $r_{\mathrm{HA,carry}} = r_{\mathrm{inv}} = 1$
- Normalised resistance of the sum output stage: $r_{\mathrm{HA,sum}} = r_{\mathrm{XOR}} = 2$

Figure 4.11: *A half adder element, realised using one NAND gate, one XOR gate, and one inverter.*

registers and D flip-flops. We make the following assumptions regarding the register elements (and D flip-flops) in the architectures considered in Chapters 5, 6, and 7:

- Every register element (and D flip-flop) has the same size and delay parameters as the D flip-flop in Figure 4.12.
- Data is fed from the output of one register through a block of combinational logic to the input of another register during one clock cycle. Consequently, each CP starts with the output stages of the first register and ends with the input stages of the destined register.
- The register input data obeys the setup and hold time constraints of the register.
- The clock signal clk of a register is the output signal of an inverter that only drives this particular register clock input. The delay time of any

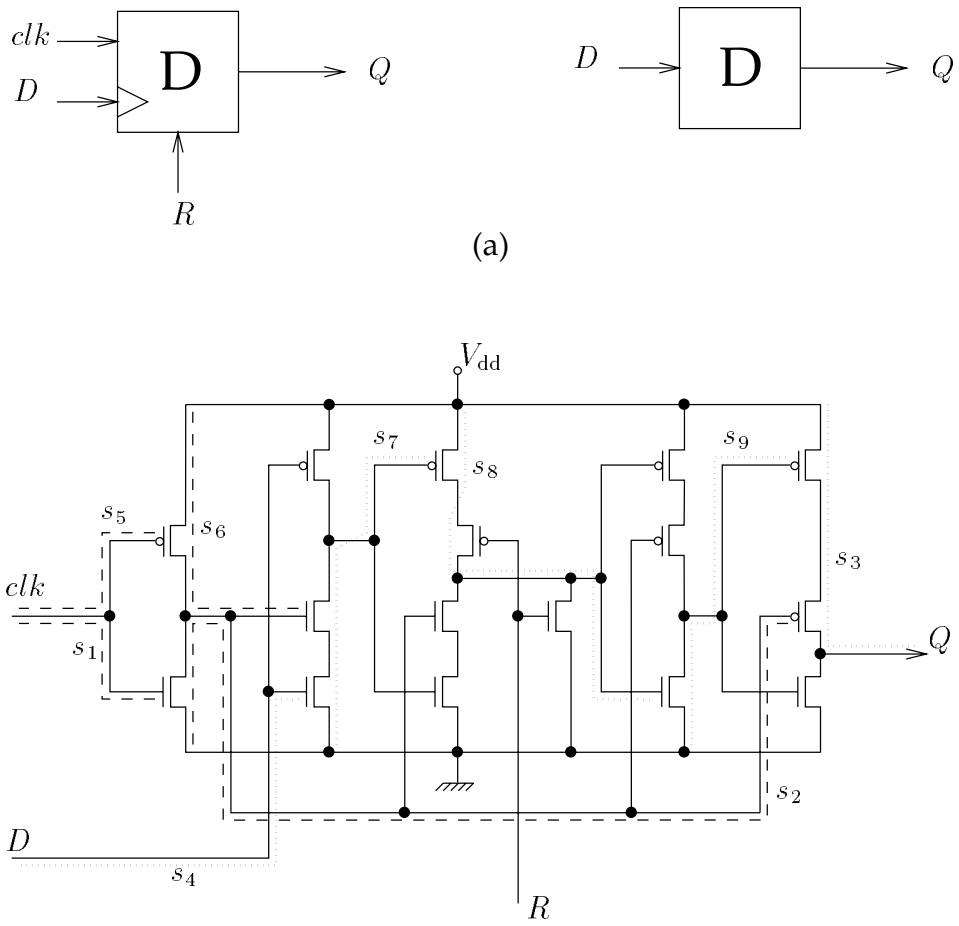


Figure 4.12: *A resettable register, realised as a dynamic, true single-phase clock, positive edge-trigged master-slave D flip-flop. The register is reset for* $R = 1$. (a) *Symbolic descriptions.* (b) *Schematic description.*

Stage s_i	Stage length $\mathcal{L}_i$
s_1	$1 \cdot 2 = 2$
s_2	$1 \cdot 4 = 4$
s_5	$1 \cdot 2 = 2$
s_6	$1 \cdot 4 = 4$
s_7	$2 \cdot 2 = 4$
s_8	$2 \cdot 2 = 4$
s_9	$1 \cdot 2 = 2$

Table 4.1: *Lengths of the internal stages of the register in Figure 4.12.*

stages ahead of the clock input stages s_1 and s_5 are *not* included in the total delay of the register.

- In a particular CMOS system, the various architectures for arithmetic operations all share the same registers for storing the input data and they share the same registers for storing the output data. Therefore, the input and output registers are generally not considered when deriving the total size of an investigated architecture. In some architectures, like architectures for serial/parallel multiplication, the input data are loaded into registers which are used throughout the whole execution time (during several clock cycles). An output register may also be used in a similar manner, for example as a feedback shift register. The size of every register involved in the computation in this way is included in the total size of an architecture.

The D flip-flop stages that are included in the CP are marked with dotted (signal stages) and dashed (clock stages) lines in Figure 4.12. The CP stages s_1, s_2, $\ldots$, s_9 in the figure are activated when an output (input) signal of the CP start (end) register changes from low to high. The output stages of the start register are the stages s_1 (due to clock rising), s_2, and s_3. The input stages of the end register are s_4, s_5, s_6, s_7, s_8, and s_9.

The lengths of the register stages are tabulated in Table 4.1. The total internal length of the start register equals the sum $\mathcal{L}_{\text{reg,out}} = 2 + 4 = 6$ of the lengths of stages s_1 and s_2. The total internal length of the end register equals the sum $\mathcal{L}_{\text{reg,in}} = 2+4+4+4+2 = 16$ of the lengths of stages s_5, s_6, s_7, s_8, and s_9. Hence, the register contribution to the *total* CP length equals $\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{reg,out}} + \mathcal{L}_{\text{reg,in}} = 22$. The fan-in f_{reg} of the register and its output normalised resistance r_{reg} of the output stage s_3 both equal 2.

4.3.6 Table of Complexity Parameters

In Table 4.2 we have listed the complexity parameters of the circuits that have been analysed in the previous Sections 4.3.1 to 4.3.5. For each circuit we state its size $\mathcal{C}$, fan-in f, internal CP length $\mathcal{L}$, and its output normalised resistance r.

CMOS circuit	Size $\mathcal{C}$	Fan-in f	Internal CP length $\mathcal{L}$	Output norm. resistance r
Inverter	2	2	—	1
Transmission gate[7]	2	1(0)	—	$r_{\text{prior}} + 1$
2-input Multiplexer[7]	4	2(0)	—	$r_{\text{prior}} + 1$
2-input NAND and NOR gates	4	2	—	2
2-input AND and OR gates	6	2	4	1
2-input XOR and XNOR gates	12	4	2	2
Full adder element	28			1
• Signal input to sum output		8	12	
• Signal input to carry output		8	8	
• Carry input to sum output		6	12	
• Carry input to carry output		6	8	
Half adder element	18	6		
• Signal input to sum output			2	2
• Signal input to carry output			4	1
(Shift) Register, D flip-flop	16		16+6= 22	
• Input path		2	16	
• Output path			6	2

[7] If a transmission gate transistor is the trigger of the output stage, then the fan-in equals 1 for the transmission gate and 2 for the multiplexer. If not, the fan-in equals zero in both cases and only the output stage contributes to the CP. The normalised resistance r_{prior} equals the output normalised resistance of the circuit that is prior to the transmission gate (or multiplexer).

Table 4.2: *The sizes, fan-ins, internal CP lengths, and output normalised resistances of some frequently used CMOS circuits.*

4.4 Implementing the Fermat Number Transform

In the previous chapters, we have mentioned several advantages of number theoretic transforms in general and the Fermat number transform in particular. For example, digital convolution of real integer sequences can be implemented using Fermat number transforms for which multiplication by powers of the transform kernel can be carried out as binary shifts (rotations). Also, no round-off errors occur during the computations, because the arithmetic operations involved are carried out in a finite ring or field.

As mentioned in Section 2.3.2 the Fermat number transform, whose length is a power of two, can be computed using a suitable fast Fourier transform algorithm. In Section 2.3.3 we considered the conventional radix-2, radix-4, and split-radix algorithms, in which the transform additions and multiplications are partitioned into so called butterfly computations. These algorithms exploit different degrees of parallelism.

Since the publication in 1976 of McClellan's [65] hardware implementation of a Fermat number transform, several Fermat number transform architectures have appeared in the literature. Truong et al. [103, 107] considered the implementation of fast digital filtering using a generalised overlap-save method and a parallel pipelined Fermat number transform architecture.

Based on the work of Truong et al., Towers et al. [101] designed a cascadable nMOS VLSI circuit for fast convolution, involving a pipelined Fermat number transformer.

Shakaff et al. [90] investigate the practical aspects of using the Fermat number transform as a block-mode image filtering tool on small microprocessor based systems. Their transform architecture is based on a gate array implementation of the butterfly computational unit.

Several aspects and techniques for implementing the Fermat number transform in (nMOS) VLSI are investigated in the theses by Pajayakrit [71, Ch. 4–] and Shakaff [89].

Finally, we also would like to mention the recent paper by Benaissa et al. [13], in which the authors present a CMOS VLSI design of a high-speed Fermat number transform-based convolver/correlator. The VLSI chip comprises a complete 64-point pipeline transformer that can be used for both the forward and the inverse Fermat number transform.

In all the above papers, except the one by McClellan, the authors have adopted the *diminished–1* representation of the elements in Fermat integer quotient rings. The diminished–1 representation is thoroughly investigated in Chapter 6.

As mentioned before, we are primarily interested in the *arithmetic operations* required to compute the Fermat number transform. Architectures for the complete transform or the transform butterflies are not further considered in this thesis.

Chapter 5

The Normal Binary Coded Representation

We only consider element representations that can be expressed as simple elementary functions of the normal binary coded (NBC) representation. In the present chapter, we study integer arithmetic operations modulo $2^m + 1$ with respect to the NBC representation itself.

5.1 Architectures for Arithmetic Operations

We are mainly interested in VLSI architectures for the arithmetic operations that may be involved in the computation of the Fermat number transform and its inverse transform. Therefore, we consider architectures for modulus reduction, negation, addition, subtraction, multiplication by powers of 2, general multiplication, and exponentiation, with respect to a binary coded representation of the integers of $\mathbb{Z}_{2^m+1}$. All these operations may not be involved in the computation of the Fermat number transform, but for completeness they are still considered. For example, general multiplication can be avoided using a suitable transform kernel, see Section 2.3.2. We do not consider division, because it is not needed when computing the Fermat number transform and it is not a general operation in every Fermat integer quotient ring.

The architectures for some of the arithmetic operations considered in the thesis are based on architectures for operations on ordinary two's complement binary coded numbers. There is a wide variety of VLSI designs available for these operations. For example, an adder circuit can be implemented in sev-

eral ways. It can be a carry ripple adder, a carry select adder, a carry save adder, a carry look-ahead adder, a conditional-sum adder, or some other type of adder [113, Ch. 8.2.1]. All architectures in Chapters 5, 6, and 7 are not optimal with respect to chip area, computation time, or area-time performance. We primarily consider architectures that can be mutually compared in order to decide which form of *element representation* is most advantageous, with respect to some area and/or time complexity of the resulting architectures. Thus, for a certain element representation and a certain arithmetic operation there *may* exist architectures that have better area-time performance than the one (or the ones) presented here.

Henceforth, most of the architectures presented are valid for arithmetic operations in the Fermat integer quotient ring $\mathbb{Z}_{2^4+1}$, i.e. for $m = 4$. However, in general the architectures are regular in such a way that they can easily be expanded (or contracted) to become applicable in any ring $\mathbb{Z}_{2^m+1}$, where m is a power of two. The only exceptions are the architectures in Chapter 7. They are based on the polar representation and are applicable only when $\mathbb{Z}_{2^m+1}$ is a field, i.e. for $m = 1,\ 2,\ 4,\ 8,\ 16$.

The $2^m + 1$ binary coded integers of $\mathbb{Z}_{2^m+1}$ are represented as $(m+1)$-bit NBC numbers. Therefore, by a congruence $a \equiv b \pmod{2^m+1}$ we generally consider a to be the least nonnegative $(m+1)$-bit residue of b modulo $2^m + 1$.

5.1.1 Modulus Reduction

It is important that the reduction modulo $2^m + 1$ is carried out as simply and fast as possible, because it is involved in all arithmetic operations in $\mathbb{Z}_{2^m+1}$. For some operations, the modulus reduction may be included in the overall computation.

Let β be an n-bit normal binary coded integer. This integer $\beta = \beta_{n-1}2^{n-1} + \beta_{n-2}2^{n-2} + \cdots + \beta_1 2 + \beta_0$, where $\beta_i \in \mathbb{Z}_2$ for $0 \le i \le n-1$, may also be represented by the n-bit binary vector $\beta^{(n-1)} \triangleq (\beta_{n-1}, \beta_{n-2}, \ldots, \beta_1, \beta_0)_2$. The notation $\beta^{(n-1)}$ is occasionally used also for the integer β.[1]

The residue of an $(m+1)$-bit integer $\beta > 2^m \pmod{2^m+1}$ is simply calculated by first changing the one (1) in the most significant bit position β_m of β to a zero and then subtracting a one from the modified number, i.e. $\beta = \beta_m 2^m + \beta^{(m-1)} \equiv \beta^{(m-1)} - \beta_m = \beta^{(m-1)} - 1 \pmod{2^m+1}$. In a hardware realisation, a simple way to subtract 1 from the binary coded m-bit positive integer $\beta^{(m-1)}$ is to add

[1] This is illustrated, for $n = m + 1$, in Table 2.3 of Section 2.4.

the two's complement of 1 to the integer;

$$\begin{aligned} \beta &\equiv \beta^{(m-1)} - 1 \pmod{2^m + 1} \\ \Rightarrow \beta &\equiv \sigma \pmod{2^m}, \end{aligned}$$

where $\sigma \triangleq \beta^{(m-1)} + (2^m - 1)$. It follows by (4.2) and (4.3) that the carry output c_{i+1} and the sum output σ_i of a binary full adder element can be expressed as the Boolean functions

$$c_{i+1} = \beta_i \gamma_i + c_i(\beta_i + \gamma_i)$$

$$\sigma_i = \beta_i \oplus \gamma_i \oplus c_i$$

respectively, where β_i and γ_i are the adder input signals and c_i is the carry input signal. By letting $\gamma = 2^m - 1$, i.e. let $\gamma_i = 1$ for $0 \leq i \leq m-1$, the carry and the sum output functions reduce to

$$c_{i+1} = \beta_i + c_i \tag{5.1}$$

$$\sigma_i = \overline{\beta_i \oplus c_i}, \tag{5.2}$$

respectively. Hence, we get $\sigma = c_m 2^m + \sigma_{m-1} 2^{m-1} + \cdots + \sigma_1 2 + \sigma_0$, where c_m and $\sigma_i;\ 0 \leq i \leq m-1$ are given in (5.2) and (5.2), respectively.

The full adder elements can be connected in different ways. We consider two types of two-operand parallel adders, which are based on how the internal carries between the adder elements are generated. One of the adder types is the *carry ripple* (or ripple carry) adder, for which the carry output of each full adder element is connected to the carry input of the subsequent full adder element (the one in the next higher-order bit position). The second adder type is the *carry look-ahead* adder, for which the internal carry signals are precomputed. The carry look-ahead adder is usually faster than the carry ripple adder, but the penalty paid for this is a greater area complexity, see for example Weste and Eshraghian [113, Ch. 8.2.1] or Hwang [52, Ch. 3].

A Carry Ripple-Based Architecture

Figure 5.1 shows an architecture that performs the modulus reduction $\varphi \equiv \beta \pmod{2^4 + 1}$ using essentially a simplified carry ripple adder followed by two-input multiplexers. The multiplexers, which are formed by the transmission gate pairs at the outputs, let either β (if $\beta \leq 2^m$) or σ (if $\beta > 2^m$) pass to the output. The signal h and its inverse control the multiplexers. The output residue is the $(m+1)$-bit normal binary coded integer $\varphi = \varphi_m 2^m + \varphi_{m-1} 2^{m-1} + \cdots + \varphi_1 2 + \varphi_0$. The architecture is based on the following algorithm:

1. If $0 \leq \beta \leq 2^m$ $(h = 0)$, then let $\varphi = \beta$

2. If $2^m + 1 \leq \beta \leq 2^{m+1} - 1$ $(h = 1)$, then let $\varphi = (0, \sigma^{(m-1)})$, where $\sigma^{(m-1)} = \beta^{(m-1)} - 1$.

The carry signals are realised using a chain of OR gates. The Boolean function $h = \beta_m c_m$ is used to indicate whether β is greater than 2^m. We assume that β is always an $(m+1)$-bit integer, i.e. the maximum reducible overflow is $2^{m+1} - 1$.

The architecture in Figure 5.1 for reduction modulo $2^m + 1$ comprises $m - 1$ OR gates, $m - 1$ XNOR gates, one NAND gate, $m + 1$ two-input multiplexers,[2] and two inverters. Using the size parameters of Table 4.2, the size[3] of this architecture equals

$$\begin{aligned} \mathcal{C}_{\text{mod},1} &= (m-1)(\mathcal{C}_{\text{OR}} + \mathcal{C}_{\text{XNOR}}) + \mathcal{C}_{\text{NAND}} + (m+1)\mathcal{C}_{\text{MUX}} + 2\mathcal{C}_{\text{inv}} \\ &= (m-1)(6+12) + 4 + 4(m+1) + 2 \cdot 2 \\ &= 22m - 6. \end{aligned} \tag{5.3}$$

The critical path[3] (CP) through the circuit is the path from β_0 to h together with the path from β_{m-1} to φ_{m-1}, as signified by the dotted lines in the figure. The fan-in,[3] with respect to the β_0-input node, equals

$$f_{\text{mod},1} = f_{\text{inv}} + f_{\text{OR}} + f_{\text{XNOR}} = 2 + 2 + 4 = 8.$$

The output normalised resistance[3] equals

$$r_{\text{mod},1} = r_{m-1} + 1,$$

where r_{m-1} is the total normalised resistance from the β_{m-1}-input node to the supply voltage source, i.e. the output normalised resistance of the preceding circuit.

Regarding $r_{\text{mod},1}$, the length of the output stage, which is the dotted path from β_{m-1} to φ_{m-1} in Figure 5.1, actually equals $r_{m-1}(f_{\text{OR}} + f_{\text{XNOR}}) + (r_{m-1} + 1)n_{m-1}$, where n_{m-1} is the fan-out seen from the φ_{m-1}-output node. However, because the β_{m-1}-input node is fully charged when the output multiplexer opens, we do not include the former part of the expression. Thus, only the term $(r_{m-1} + 1)n_{m-1}$ contributes to the length of the output path and hence the output normalised resistance $r_{\text{mod},1}$ equals $r_{m-1} + 1$. The internal CP length of the

[2] For the sake of simplicity, the single inverter of the output circuitry in bit position m is regarded as a transmission gate.

[3] The size, critical path, fan-in, and output normalised resistance of an architecture was defined in Section 4.2.2.

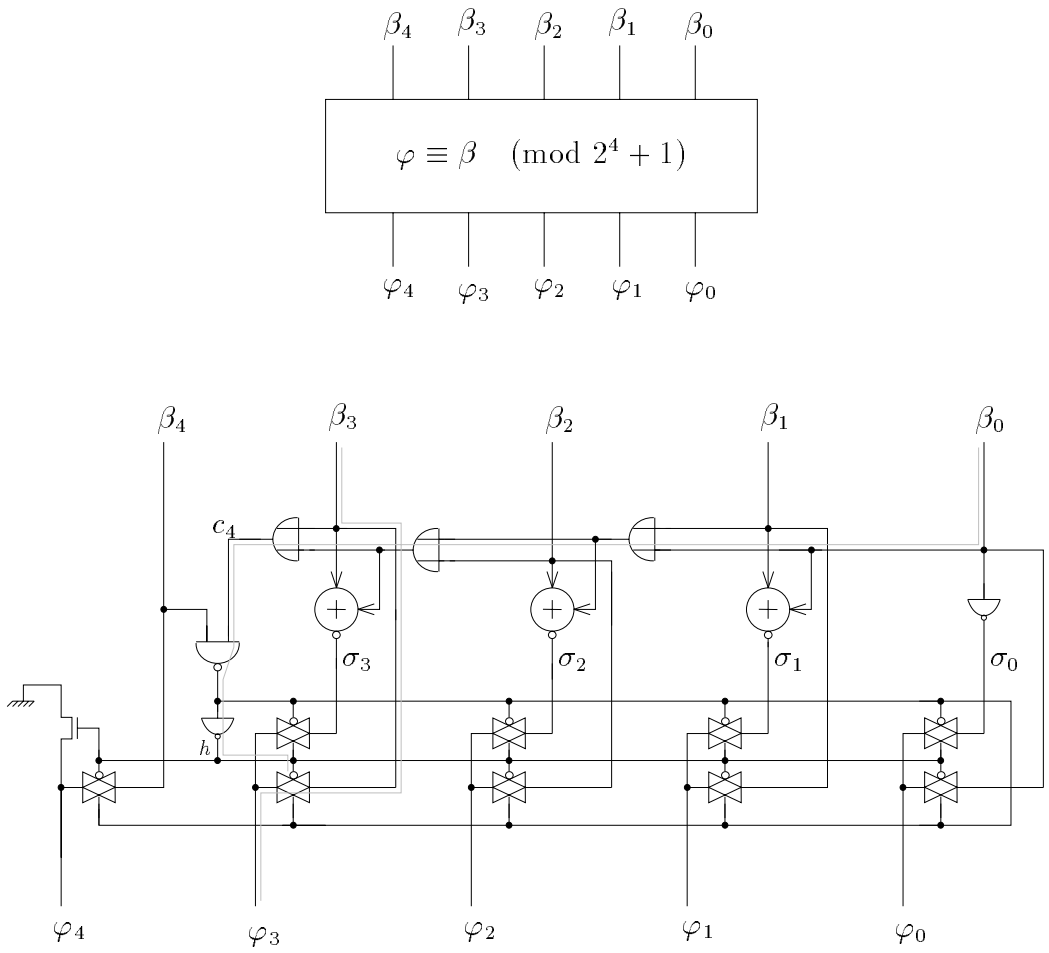


Figure 5.1: *A carry ripple-type circuit for reduction modulo* $2^m + 1$*, where* $\sigma^{(m-1)} \equiv \beta^{(m-1)} - 1 \pmod{2^m}$ *and* $m = 4$*. If* $\beta > 2^m$ *(overflow) then* $h = 1$*, otherwise* $h = 0$*. The two dotted lines indicate the set of stages that form the critical path through the circuit.*

circuit equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,mod,1}} &= (m-1)\mathcal{L}_{\mathrm{OR}} + (m-2)r_{\mathrm{OR}}(f_{\mathrm{OR}} + f_{\mathrm{XNOR}}) + r_{\mathrm{OR}}f_{\mathrm{NAND}} \\ &\quad + r_{\mathrm{NAND}}(2m+1+f_{\mathrm{inv}}) + r_{\mathrm{inv}} \cdot 2(m+1) \\ &= (m-1)\cdot 4 + (m-2)\cdot 1\cdot(2+4) + 1\cdot 2 \\ &\quad + 2(2m+1+2) + 1\cdot 2(m+1) \\ &= 16m - 6.\end{aligned}$$

The circuit parameters forming $\mathcal{L}_{\mathrm{CP,mod,1}}$ are obtained from Table 4.2.

In some situations, the modulus reduction may directly succeed operations for which the signal β_{m-1} appears on its input node *after* the time when the internal carry signal c_{m-1} appears on its node in the carry chain. See for example the adder architecture of Figure 5.7. Then, the modulus reduction circuit contribution to the total CP length may be much smaller than $\mathcal{L}_{\mathrm{CP,mod,1}}$.

In Chapter 4 we mentioned that, with respect to size and time performance, NAND and NOR gates are preferable to AND and OR gates, respectively. Accordingly, it may seem advantageous to realise the carry chain using a chain of alternating NAND and NOR gates instead of a chain of OR gates. We have designed such an architecture. The size of that architecture equals $21m - 4$, which is slightly less than the size $\mathcal{C}_{\mathrm{mod,1}} = 22m - 6$ of the OR-type architecture in Figure 5.1. However, the internal CP length of the NAND/NOR-type architecture equals $18m + 33$, which is *greater* than the corresponding length $\mathcal{L}_{\mathrm{CP,mod,1}} = 16m - 6$ of the architecture in Figure 5.1. The increase of the first term by $2m$ (from $16m$ to $18m$) equals the difference between the contributions $(\mathcal{L}_{\mathrm{OR}} + r_{\mathrm{OR}}(f_{\mathrm{OR}} + f_{\mathrm{XNOR}}))m = (4 + 1\cdot 6)m = 10m$ and $r_{\mathrm{NAND/NOR}}(f_{\mathrm{NAND/NOR}} + f_{\mathrm{XNOR}})m = (2\cdot 6)m = 12m$ to the CP lengths of the OR-type and the NAND/NOR-type architectures, respectively. Thus, for each bit position, the increase of the CP length (by 6) due to the doubling of the normalised resistance in a stage (when exchanging an OR gate for a NAND or NOR gate) is greater than the decrease of the length (by 4) due to the elimination of the internal length $\mathcal{L}_{\mathrm{OR}}$ of the OR gate.

A Carry Look-Ahead-Based Architecture

In Figure 5.2 we present a carry look-ahead-type circuit for modulus reduction. Here, the carry signals are generated in parallel using the tree of NAND and NOR gates that precedes the row of XNOR gates in the figure. The structure of this simplified and distributed carry look-ahead tree is similar to the structure of Brent and Kung's carry look-ahead tree [27].

The depth of the tree is $\log_2\ m$ and there are $m - 2^i$ NAND or NOR gates in level i of the tree, starting with $i = 0$ for the input level. Thus, there are a total of

$$\sum_{i=0}^{\log_2\ m-1} (m - 2^i) = m(\log_2\ m - 1) + 1$$

such gates in the tree, distributed such that the NOR gates are only in the even numbered levels of the tree and the NAND gates are only in the odd numbered levels. Also, there are 2^i inverters in level i of the tree, which means that the total number of inverters in the tree is $m - 1$. Hence, the size of the carry look-ahead-type modulus reduction circuit in Figure 5.2 equals

$$\begin{aligned}
\mathcal{C}_{\mathrm{mod},2} &= (m(\log_2\ m - 1) + 1)\mathcal{C}_{\mathrm{NAND/NOR}} + m\mathcal{C}_{\mathrm{inv}} \\
&\quad + (m-1)\mathcal{C}_{\mathrm{XNOR}} + \mathcal{C}_{\mathrm{NAND}} + (m+1)\mathcal{C}_{\mathrm{MUX}} \\
&= (m(\log_2\ m - 1) + 1) \cdot 4 + m \cdot 2 + (m-1) \cdot 12 + 4 + (m+1) \cdot 4 \\
&= 4m \cdot \log_2\ m + 14m. \qquad (5.4)
\end{aligned}$$

The CP through the circuit is the set of stages along the two dotted lines in the figure.[4] The fan-in of the circuit equals

$$f_{\mathrm{mod},2} = f_{\mathrm{XNOR}} + 2f_{\mathrm{NOR}} = 4 + 2 \cdot 2 = 8$$

and its output normalised resistance $r_{\mathrm{mod},2}$ equals

$$r_{\mathrm{mod},2} = r_{\mathrm{mod},1} = r_{m-1} + 1.$$

Thus the architectures in Figures 5.1 and 5.2 have equal fan-in and output normalised resistance. The internal CP length of the carry look-ahead-type architectures equals

$$\begin{aligned}
\mathcal{L}_{\mathrm{CP,mod},2} &= (\log_2\ m - 1)r_{\mathrm{NAND/NOR}}(f_{\mathrm{NOR/NAND}} + f_{\mathrm{inv}}) + r_{\mathrm{NAND}}f_{\mathrm{NAND}} \\
&\quad + r_{\mathrm{NAND}}(2m + 1 + f_{\mathrm{inv}}) + r_{\mathrm{inv}} \cdot 2(m+1) \\
&= (\log_2\ m - 1) \cdot 2(2+2) + 2 \cdot 2 \\
&\quad + 2(2m + 1 + 2) + 1 \cdot 2(m+1) \\
&= 6m + 8\log_2\ m + 4.
\end{aligned}$$

As mentioned above, in some situations the CP may enter the circuit via the β_{m-1}-input node. In such a situation, the carry ripple-type architecture in Figure 5.1 is preferable to the carry ripple-type architecture in Figure 5.2, because the path from the β_{m-1} input to the last carry signal c_m is shorter in the former case than in the latter case. Regarding the architecture in Figure 5.2, we would like to mention the following:

[4] Actually, the CP output stage is any of the stages from a β_i-input node to the corresponding φ_i-output node, where $0 \leq i \leq m-1$.

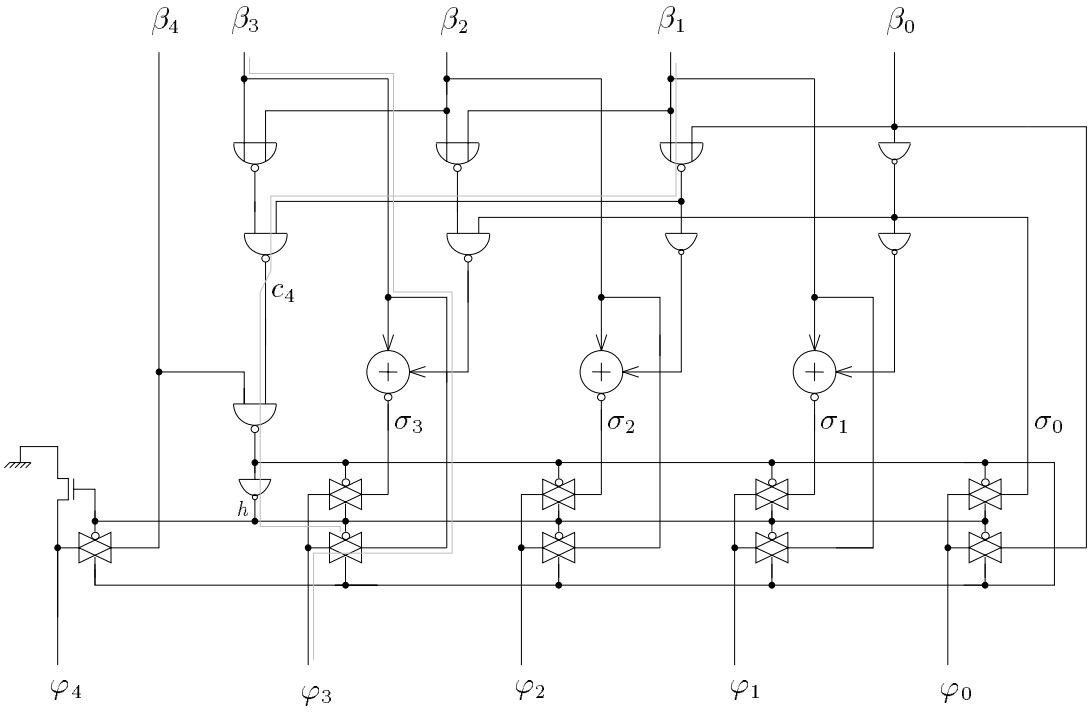


Figure 5.2: *A carry look-ahead-type circuit for reduction modulo* $2^m + 1$*, where* $m = 4$*. The two dotted lines signify the set of stages that form the critical path through the circuit.*

1. With respect to both area and time complexity, NAND and NOR gates are preferable to OR gates in the carry look-ahead tree. The difference in complexity is, however, not significant.

2. The placement of the row of $m/2$ inverters in the last level of the carry look-ahead tree differs, depending on whether $\log_2 m$ is odd or even. The inverters can be omitted if the subsequent row of $m/2$ XNOR gates is exchanged for a row of XOR gates.

3. A disadvantage of the carry look-ahead tree may be its relatively long internal wires.

Regarding the architectures in both Figure 5.1 and Figure 5.2, for large m there may be a problem for the NAND gate and the inverter (whose output signals are $\overline{h}$ and h, respectively) to each drive $2(m + 1)$ multiplexer transistors. The

delay of a stage with a large capacitive load can be significantly reduced by using properly sized drivers. Such drivers are, however, not used here.

Our comparison between different architectures with respect to their area-time performance is made under the assumption that the architectures are both preceded and followed by a parallel register. Then, using the architecture in Figure 5.1 or the one in Figure 5.2, the time T to perform the modulus reduction operation is proportional to the lengths

$$\begin{aligned}\mathcal{L}_{\mathrm{mod},1} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} f_{\mathrm{mod},1} + \mathcal{L}_{\mathrm{CP},\mathrm{mod},1} + r_{\mathrm{mod},1} f_{\mathrm{reg}} \\ &= 22 + 2 \cdot 8 + 16m - 6 + 3 \cdot 2 \\ &= 16m + 38\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{\mathrm{mod},2} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} f_{\mathrm{mod},2} + \mathcal{L}_{\mathrm{CP},\mathrm{mod},2} + r_{\mathrm{mod},2} f_{\mathrm{reg}} \\ &= 22 + 2 \cdot 8 + 6m + 8 \log_2 m + 4 + 3 \cdot 2 \\ &= 6m + 8 \log_2 m + 48,\end{aligned}$$

respectively, where $r_{\mathrm{mod},1} = r_{\mathrm{mod},2} = r_{\mathrm{reg}} + 1 = 3$. Using the size paramaters $\mathcal{C}_{\mathrm{mod},1}$ and $\mathcal{C}_{\mathrm{mod},2}$ in (5.3) and (5.4), respectively, and the above lengths $\mathcal{L}_{\mathrm{mod},1}$ and $\mathcal{L}_{\mathrm{mod},2}$, the area-time performances AT^2 of these modulus reduction circuits are proportional to the products

$$\begin{aligned}\mathcal{C}\mathcal{L}^2_{\mathrm{mod},1} &\triangleq \mathcal{C}_{\mathrm{mod},1}(\mathcal{L}_{\mathrm{mod},1})^2 = (22m - 6)(16m + 38)^2 \\ &= \mathcal{O}\left(m^3\right)\end{aligned}$$

$$\begin{aligned}\mathcal{C}\mathcal{L}^2_{\mathrm{mod},2} &\triangleq \mathcal{C}_{\mathrm{mod},2}(\mathcal{L}_{\mathrm{mod},2})^2 = (4m \log_2 m + 14m)(6m + 8 \log_2 m + 48)^2 \\ &= \mathcal{O}\left(m^3 \log_2 m\right),\end{aligned}$$

respectively. The sizes, CP lengths, and AT^2 performances of the above two circuits for modulus reduction are plotted in Figure 5.3. We see that the size $\mathcal{C}_{\mathrm{mod},2}$ of the carry look-ahead-type architecture in Figure 5.2 is greater than the size $\mathcal{C}_{\mathrm{mod},1}$ of the carry ripple-type architecture in Figure 5.1. On the other hand, for the CP lengths of the architectures we have the reverse relation. The ratio of the CP lengths $\mathcal{L}_{\mathrm{mod},1}$ and $\mathcal{L}_{\mathrm{mod},2}$ converges relatively fast to $8/3$ with growing m. We conclude that, with respect to the time complexities and the AT^2 performances, the architecture in Figure 5.2 is preferable to the architecture in Figure 5.1.

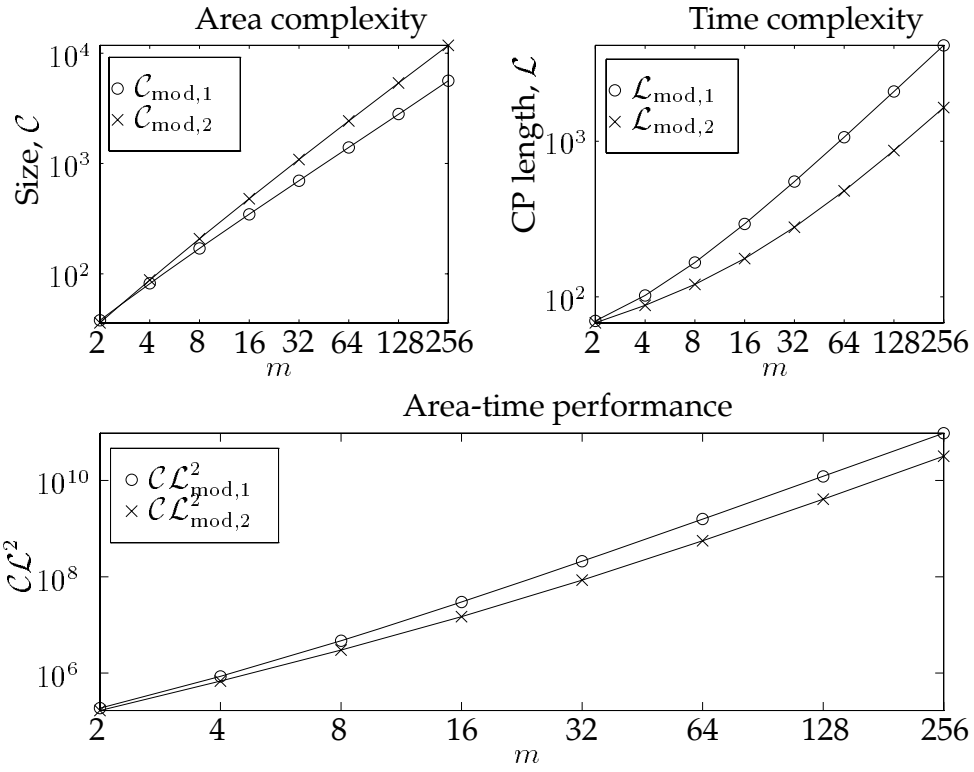


Figure 5.3: *Sizes, CP lengths, and AT^2 performances of the two modulus reduction architectures. The parameters are plotted versus m for $m = 2, 4, 8, 16, 32, 64, 128, 256$. The lines connecting the parameter values are plotted only to clearly illustrate how the complexity parameters grow with m.*

5.1.2 Negation

Let β be an $(m+1)$-bit NBC integer. Then we have

$$\begin{aligned} \varphi &\equiv -\beta \equiv (2^{m+1}-1) - \beta - 2 \cdot 2^m + 1 \\ &\equiv \overline{\beta} + 3 \pmod{2^m+1}, \end{aligned} \tag{5.5}$$

where $\overline{\beta} = (\overline{\beta_m}, \overline{\beta_{m-1}}, \ldots, \overline{\beta_1}, \overline{\beta_0})_2$ is the one's complement of $\beta = (\beta_m, \beta_{m-1}, \ldots, \beta_1, \beta_0)_2$. Adding 3 to $\overline{\beta}$ seems to be a simple operation, but we also would like to perform the modulus reduction in the same computation step. We therefore expand $\overline{\beta}$ as

$$\overline{\beta} = \overline{\beta_m} 2^m + \overline{\beta^{(m-1)}} \equiv \overline{\beta^{(m-1)}} - \overline{\beta_m} = \overline{\beta^{(m-1)}} + \beta_m - 1 \pmod{2^m+1}$$

where, by Definition 2.4, $\beta^{(m-1)} = (\beta_{m-1}, \beta_{m-2}, \ldots, \beta_1, \beta_0)_2 = \beta_{m-1}2^{m-1} + \beta_{m-2}2^{m-2} + \cdots + \beta_1 2 + \beta_0$. Hence, (5.5) can be written as

$$\begin{aligned} \varphi &\equiv -\beta \equiv \overline{\beta^{(m-1)}} + \beta_m - 1 + 3 \\ &= 2(\gamma + 1) + \overline{\beta_0} + \beta_m \qquad (\mathrm{mod}\ 2^m + 1), \end{aligned}$$

where $\gamma = \overline{\beta_{m-1}}2^{m-2} + \overline{\beta_{m-2}}2^{m-3} + \cdots + \overline{\beta_2}2 + \overline{\beta_1}$ is an $(m-1)$-bit binary coded integer for which $\gamma_i = \overline{\beta_{i+1}}$; $0 \le i \le m-2$. Let $\sigma \triangleq \gamma + \theta$, where $\theta = 1$. This sum can be computed using a simplified $(m-1)$-bit parallel adder. By (4.3) and (4.2) the sum output and the carry output of the adder element in bit position i are the Boolean functions $\sigma_i = \theta_i \oplus \gamma_i \oplus c_i$ and $c_{i+1} = \theta_i\gamma_i + c_i(\theta_i + \gamma_i)$ respectively, where $0 \le i \le m-2$. We have $\gamma_i = \overline{\beta_{i+1}}$ for $0 \le i \le m-2$, $\theta_i = 0$ for $1 \le i \le m-2$, $\theta_0 = 1$, and $c_0 = 0$. Hence, the sum and carry output functions simplifies to

$$\sigma_i = \gamma_i \oplus c_i = \overline{\beta_{i+1}} \oplus c_i$$

$$c_{i+1} = \gamma_i c_i = \overline{\beta_{i+1}} c_i$$

respectively, for $i = 0, 1, \ldots, m-2$. We identify these functions as the sum and carry outputs of the half adder element, see Section 4.3.4 (Figure 4.11).

From the above it follows that the desired integer φ equals the $(m+1)$-bit NBC integer $\varphi = (\varphi_m, \sigma_{m-2}, \sigma_{m-3}, \ldots, \sigma_1, \sigma_0, \varphi_0)_2$. We obtain the bit values φ_m and φ_0 as Boolean functions of the carry signal c_{m-1} and the input signals β_m and β_0. There are four situations that have to be handled:

1. If $\beta = 0$, then $\sigma = 2^{m-1}$, $\beta_0 = 0$, and $\beta_m = 0$.
   Let $\varphi_m = \overline{c_{m-1}} = 0$ and $\varphi_0 = \beta_0 = 0$.
2. If $\beta = 1$, then $\sigma = 2^{m-1}$, $\beta_0 = 1$, and $\beta_m = 0$.
   Let $\varphi_m = c_{m-1} = 1$ and $\varphi_0 = \overline{\beta_0} = 0$.
3. If $2 \le \beta \le 2^m - 1$, then $1 \le \sigma \le 2^{m-1} - 1$, β_0 is arbitrary, and $\beta_m = 0$.
   Let $\varphi_m = c_{m-1} = 0$ and $\varphi_0 = \overline{\beta_0}$.
4. If $\beta = 2^m \equiv -1 \pmod{2^m + 1}$, then $\sigma = 2^{m-1}$, $\beta_0 = 0$, and $\beta_m = 1$.
   Let $\varphi_m = \overline{c_{m-1}} = 0$ and $\varphi_0 = \overline{\beta_0} = 1$.

These special cases yield the Karnaugh maps for φ_m and φ_0 in Figure 5.4. The respective Boolean functions are

$$\varphi_m = c_{m-1}\beta_0 = \overline{\overline{c_{m-1}} + \overline{\beta_0}},$$

$$\varphi_0 = \beta_m + \overline{c_{m-1}}\,\overline{\beta_0} = \overline{\overline{\beta_m} \cdot (\overline{\overline{c_{m-1}}\,\overline{\beta_0}})}\,.$$

β_m \ $c_{m-1}\beta_0$	00	01	11	10
0	0	0	1	0
1	X	X	X	0

(a) φ_m

β_m \ $c_{m-1}\beta_0$	00	01	11	10
0	1	0	0	0
1	X	X	X	1

(b) φ_0

Figure 5.4: *Karnaugh maps for the output variables φ_m and φ_0 of the negation circuit.* X = *"don't care".* (a) $\varphi_m = c_{m-1}\beta_0$. (b) $\varphi_0 = \beta_m + \overline{c_{m-1}}\overline{\beta_0}$.

Figure 5.5 shows a realisation in $\mathbb{Z}_{2^4+1}$ of the negation circuit. As seen in the figure, the above-mentioned simplified parallel adder consists of a row of half adder elements, each comprising one AND gate and one XOR gate. In order to generate the signal $\overline{c_{m-1}}$, the carry output of the half adder in the most significant bit position is inverted. The inversion is realised by exchanging the half adder AND gate for a NAND gate.

The size of the architecture in Figure 5.5 equals

$$\begin{aligned}\mathcal{C}_{\text{neg}} &= (m+2)\mathcal{C}_{\text{inv}} + (m-3)\mathcal{C}_{\text{HA}} + \mathcal{C}_{\text{XOR}} + 3\mathcal{C}_{\text{NAND}} + \mathcal{C}_{\text{NOR}} \\ &= 2(m+2) + 18(m-3) + 12 + 3\cdot 4 + 4 \\ &= 20m - 22.\end{aligned}$$

The CP of the architecture is the dotted path from the β_1 input along the carry chain to $\overline{c_{m-1}}$ and finally through two NAND gates to φ_0. Hence, the fan-in of the architecture equals

$$f_{\text{neg}} = f_{\text{inv}} = 2$$

and its output normalised resistance equals

$$r_{\text{neg}} = r_{\text{NAND}} = 2.$$

The length $\mathcal{L}_{\text{CP,neg}}$ of the internal CP equals

$$\begin{aligned}\mathcal{L}_{\text{CP,neg}} &= r_{\text{inv}}(f_{\text{inv}} + f_{\text{HA}}) + (m-3)(\mathcal{L}_{\text{HA,carry}} + r_{\text{HA,carry}} f_{\text{HA}}) \\ &\quad + r_{\text{NAND}}(f_{\text{NOR}} + f_{\text{NAND}} + f_{\text{NAND}}) \\ &= 1\cdot(2+6) + (m-3)(4+1\cdot 6) + 2\cdot(2+2+2) \\ &= 30m - 10.\end{aligned}$$

The negater in Figure 5.5 is a carry ripple type of architecture. The AND-gate carry chain can be exchanged for a chain of alternating NAND and NOR gates,

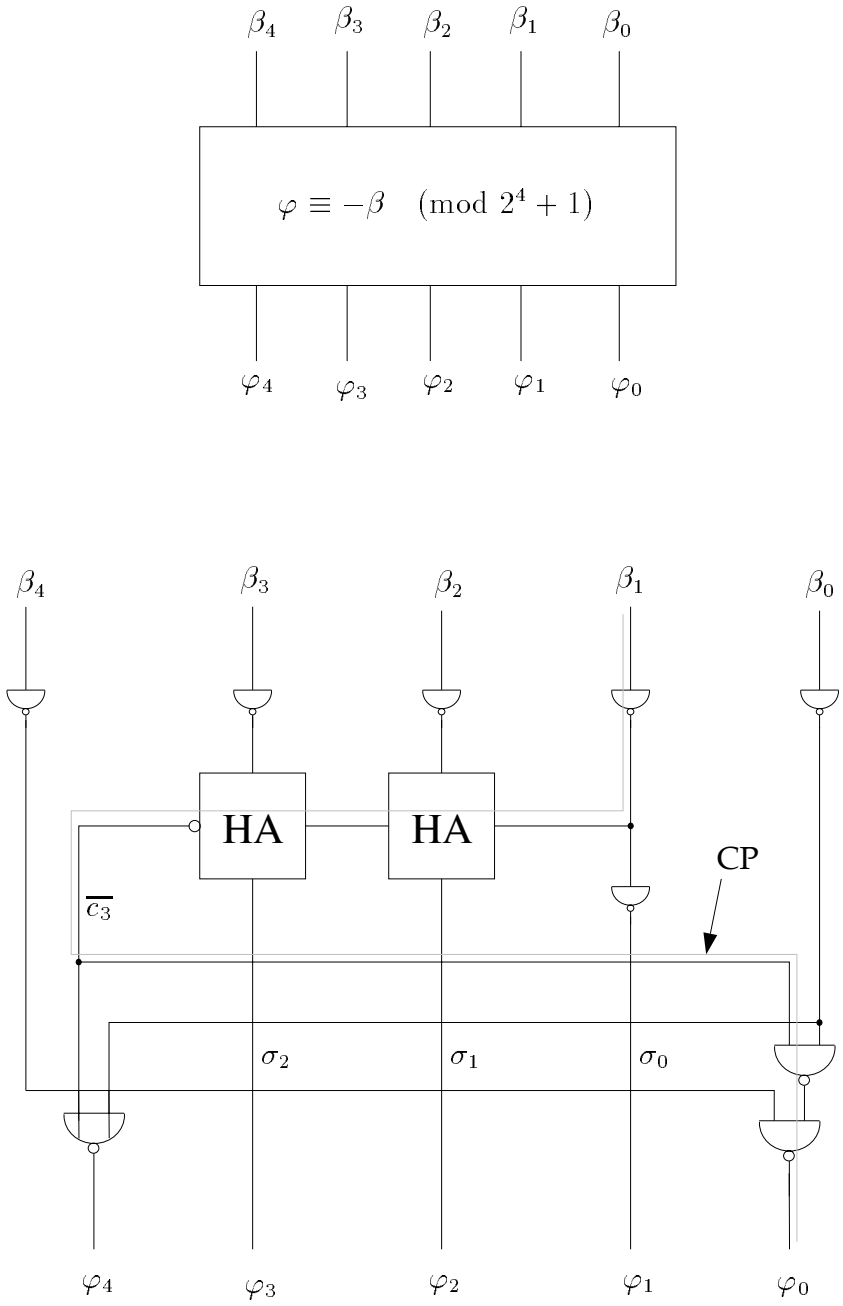


Figure 5.5: *Negation modulo* $2^m + 1,\ m = 4$.
$\varphi \equiv -\beta = 2\sigma + \beta_m + \overline{\beta_0} \pmod{2^m+1}$, *where* $\sigma = (\overline{\beta^{(m-1)} - \beta_0})/2 + 1$.

resembling the modification of the OR-gate carry chain of the modulus reduction circuit in Figure 5.1. Such a modification slightly reduces the size of the architecture, but the CP length will, however, increase. This was also the case for the circuit in Figure 5.1.

It is possible to design a carry look-ahead type of architecture for NBC negation. The carry look-ahead part of such a circuit may be similar to the carry look-ahead part of the modulus reduction architecture in Figure 5.2. The difference in area-time performance between that carry look-ahead negater and the architecture in Figure 5.5 is in the same order as the difference in area-time performance between the architecture in Figure 5.2 and the one in Figure 5.1.

When the negater in Figure 5.5 is preceded and followed by parallel registers, its total CP length equals

$$\begin{aligned}\mathcal{L}_{\text{neg}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{neg}} + \mathcal{L}_{\text{CP,neg}} + r_{\text{neg}} f_{\text{reg}} \\ &= 22 + 2 \cdot 2 + 30m - 10 + 2 \cdot 2 \\ &= 30m + 20,\end{aligned}$$

which means that the area-time performance AT^2 of the suggested negater is proportional to

$$\mathcal{C}\mathcal{L}_{\text{neg}}^2 \triangleq \mathcal{C}_{\text{neg}}(\mathcal{L}_{\text{neg}})^2 = (20m - 22)(30m + 20)^2 = \mathcal{O}\left(m^3\right).$$

In Section 8.1.3 we compare the complexity parameters of the above negater with the complexity parameters of other negation circuits.

5.1.3 Addition and Subtraction

Addition

We consider the addition $\varphi \equiv \beta + \gamma \pmod{2^m + 1}$, where the $(m+1)$-bit NBC integers φ and γ are elements of $\mathbb{Z}_{2^m+1}$, i.e. we have $0 \leq \beta, \gamma \leq 2^m$. In order to simplify the arithmetic operation, we expand the above addition in the following way.

Let $\sigma \triangleq \beta^{(m-1)} + \gamma^{(m-1)}$. This sum can be obtained by using an m-bit parallel carry ripple adder. We write σ on the form $\sigma = c_m 2^m + \sigma_{m-1} 2^{m-1} + \sigma_{m-2} 2^{m-2} + \cdots + \sigma_0$, where c_m is the carry output of the full adder element in bit position $m-1$ and σ_i is the sum output of the adder element in bit position i for $0 \leq i \leq m-1$. Because the first carry input signal is always equal to zero, the adder element in the least significant bit position may be implemented as a half adder.

Furthermore, let $\theta \triangleq \sigma^{(m-1)}+2^m-1$. This is the same type of addition as the one that resulted in the carry ripple-type modulus reduction circuit of Figure 5.1 in Section 5.1.1. With σ_i being the input of a simplified adder element of the type described in Section 5.1.1, the corresponding output is θ_i. We denote by g_m the carry output from the simplified adder element in the most significant bit position. Finally, we need a binary control signal, say h, that lets either $\theta^{(m-1)}$ or $\sigma^{(m-1)}$ pass to the adder output. Thus, we define $h \in \mathbb{Z}_2$ such that $\varphi^{(m-1)} = h\theta^{(m-1)} + (1-h)\sigma^{(m-1)}$, i.e. $\varphi_i = h\theta_i + \overline{h}\sigma_i$ for $0 \le i \le m-1$.

With regard to the above definitions, we consider the following seven special cases of input signal combinations.

1. If $\beta = \gamma = 0$, then $(\beta_m,\, \gamma_m) = (0,\, 0), \quad (c_m,\, g) = (0,\, 0)$.

   Let $\quad h = 0, \quad \varphi_m = 0, \quad \varphi^{(m-1)} = \sigma^{(m-1)}$.

2. If $\beta = 0, \;\; 1 \le \gamma \le 2^m - 1 \;\;$ or $\;\; 1 \le \beta \le 2^m - 1, \;\; \gamma = 0$,
   then $(\beta_m,\, \gamma_m) = (0,\, 0), \quad (c_m,\, g) = (0,\, 1)$.

   Let $\quad h = 0, \quad \varphi_m = 0, \quad \varphi^{(m-1)} = \sigma^{(m-1)}$.

3. If $\beta = 0, \;\; \gamma = 2^m \;\;$ or $\;\; \beta = 2^m, \;\; \gamma = 0$,
   then $(\beta_m,\, \gamma_m) = (0,\, 1)$ or $(1,\, 0), \quad (c_m,\, g) = (0,\, 0)$.

   Let $\quad h = 0, \quad \varphi_m = 1, \quad \varphi^{(m-1)} = \sigma^{(m-1)}$.

4. If $1 \le \beta, \gamma \le 2^m - 1$ and $\sigma = 2^m$,
   then $(\beta_m,\, \gamma_m) = (0,\, 0), \quad (c_m,\, g) = (1,\, 0)$.

   Let $\quad h = 0, \quad \varphi_m = 1, \quad \varphi^{(m-1)} = \sigma^{(m-1)}$.

5. If $1 \le \beta, \gamma \le 2^m - 1$ and $\sigma > 2^m$,
   then $(\beta_m,\, \gamma_m) = (0,\, 0), \quad (c_m,\, g) = (1,\, 1)$.

   Let $\quad h = 1, \quad \varphi_m = 0, \quad \varphi^{(m-1)} = \theta^{(m-1)}$.

6. If $1 \le \beta \le 2^m - 1, \;\; \gamma = 2^m \;\;$ or $\;\; \beta = 2^m, \;\; 1 \le \gamma \le 2^m - 1$,
   then $(\beta_m,\, \gamma_m) = (0,\, 1)$ or $(1,\, 0), \quad (c_m,\, g) = (0,\, 1)$.

   Let $\quad h = 1, \quad \varphi_m = 0, \quad \varphi^{(m-1)} = \theta^{(m-1)}$.

7. If $\beta = \gamma = 2^m$, then $(\beta_m,\, \gamma_m) = (1,\, 1), \quad (c_m,\, g) = (0,\, 0)$.

   Let $\quad h = 1, \quad \varphi_m = 0, \quad \varphi^{(m-1)} = \theta^{(m-1)}$.

$\beta_m\gamma_m$ \ $c_m g_m$	00	01	11	10
00	0	0	1	0
01	0	1	X	X
11	1	X	X	X
10	0	1	X	X

(a) h

$\beta_m\gamma_m$ \ $c_m g_m$	00	01	11	10
00	0	0	0	1
01	1	0	X	X
11	0	X	X	X
10	1	0	X	X

(b) φ_m

Figure 5.6: *Karnaugh maps of the Boolean function h and φ_m.*
(a) $h = \beta_m\gamma_m + g_m(c_m + \beta_m + \gamma_m)$. (b) $\varphi_m = \overline{h} \cdot \overline{\overline{\beta_m}\,\overline{\gamma_m}\,\overline{c_m}}$.

The binary control signal h and the output bit φ_m can be expressed as a Boolean functions of the variables β_m, γ_m, c_m, and g_m. From the Karnaugh map in Figure 5.6(a) we derive the Boolean function

$$h = \beta_m\gamma_m + g_m(c_m + \beta_m + \gamma_m) = \overline{\overline{\beta_m\gamma_m} \cdot \overline{g_m \cdot \overline{\overline{c_m}(\overline{\beta_m + \gamma_m})}}}.$$

By writing h on this form we see that it can be realised using four NAND gates, one inverter, and one XOR gate. The Boolean function

$$\varphi_m = \overline{\overline{(\overline{c_m + (\beta_m \oplus \gamma_m)})} + g_m}$$

is derived from the Karnaugh map in Figure 5.6(b). However, by comparing the Karnaugh maps in Figure 5.6(a) and (b), we see that the map for φ_m is the inverse of the map for h, except in the positions $(\beta_m,\ \gamma_m,\ c_m,\ g_m) = (0,\ 0,\ 0,\ 0)$ and $(0,\ 0,\ 0,\ 1)$. Therefore, the function φ_m can also be expressed as

$$\varphi_m = \overline{h} \cdot \overline{\overline{c_m}\,\overline{\beta_m}\,\overline{\gamma_m}} = \overline{h + \overline{c_m}(\overline{\beta_m + \gamma_m})}.$$

Because the inverse of the term $\overline{c_m}(\overline{\beta_m + \gamma_m})$ is a part of the expression for h, we consequently only need one inverter and one NOR gate to generate φ_m from the gates producing the signal h.

Figure 5.7 shows an adder architecture whose structure is based on the above reasoning. The sum $\sigma = \beta^{(m-1)} + \gamma^{(m-1)}$ is computed using an ordinary m-bit parallel carry ripple adder. This part of the architecture may be replaced by a carry look-ahead adder, if desirable. The gates in the leftmost dashed box in

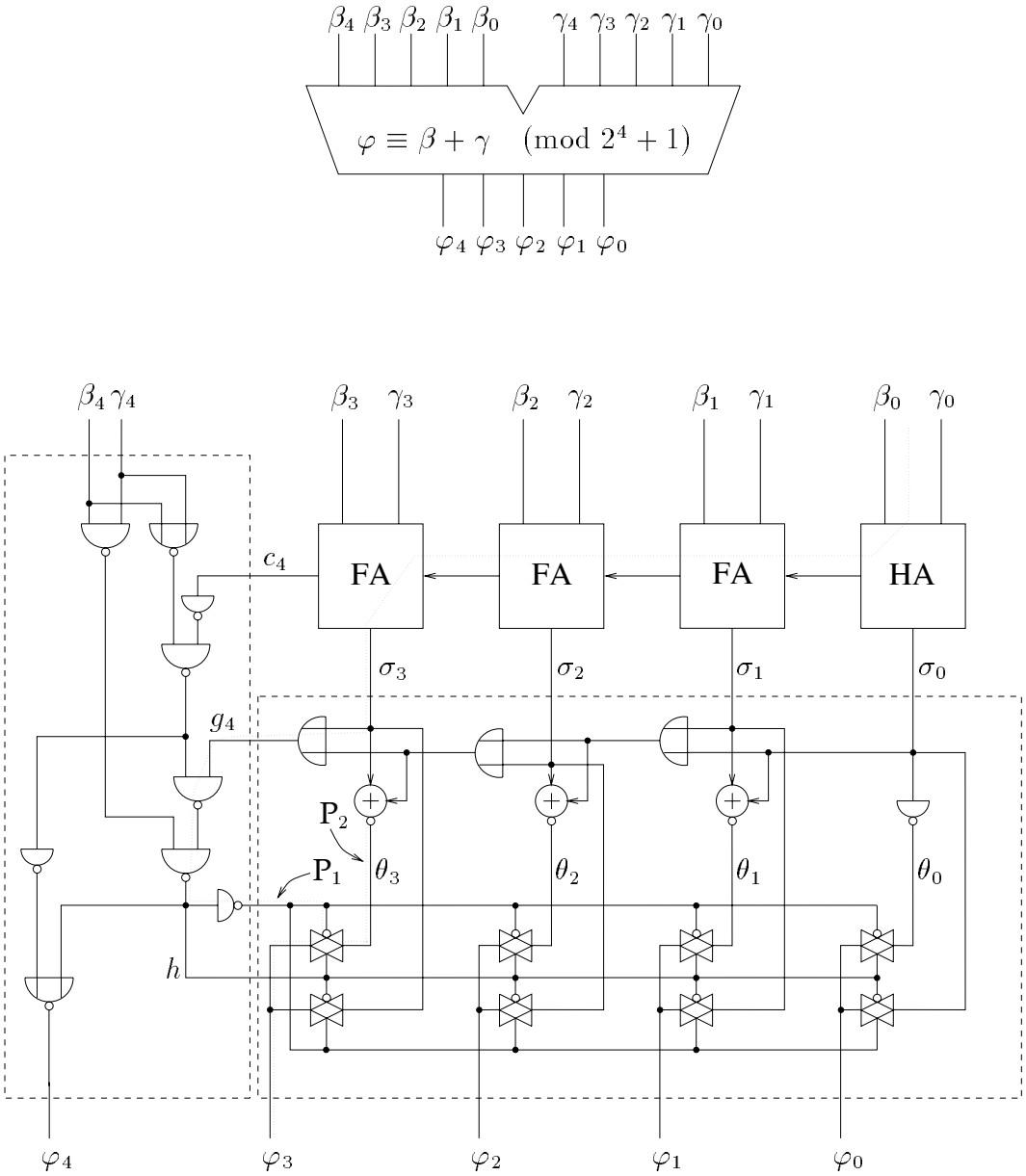


Figure 5.7: *A circuit performing the addition* $\varphi \equiv \beta + \gamma \pmod{2^m+1}$ *for* $m = 4$. *The dotted paths* P_1 *and* P_2 *form the CP through the circuit. The gates within the rightmost dashed box performs the modulus reduction. This part of the architecture can also be found in Figure* 5.1. *The gates within the leftmost dashed box generate the output bit* φ_m *and the control signal* h.

Figure 5.7 generate the control signal h and the output bit φ_m. The gates in the rightmost dashed box in the figure generate the m least significant bits of the output binary coded integer φ by subtracting, if necessary, one (1) from $\sigma^{(m-1)}$ modulo 2^m.

As a result of a timing analysis based on the RC delay model described in Chapter 4, we found that the dotted paths marked by P_1 and P_2 form the CP through the adder architecture. The fan-in f_{add} of the architecture equals the fan-in of the half adder, i.e. we get

$$f_{\text{add}} = f_{\text{HA}} = 6.$$

The internal length $\mathcal{L}_{\text{CP,add}}$ through the adder equals the length of path P_1, i.e.

$$\begin{aligned}
\mathcal{L}_{\text{CP,add}} &= \mathcal{L}_{\text{HA,carry}} + (m-1)r_{\text{FA}}f_{\text{FA,carry}} + (m-2)\mathcal{L}_{\text{FA,carry}} \\
&\quad + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}}(f_{\text{XNOR}} + f_{\text{OR}}) + \mathcal{L}_{\text{OR}} + (r_{\text{OR}} + r_{\text{NAND}})f_{\text{NAND}} \\
&\quad + r_{\text{NAND}}(2m + f_{\text{NOR}} + f_{\text{inv}}) + r_{\text{inv}} \cdot 2m \\
&= 4 + (m-1)\cdot 6 + (m-2)\cdot 8 + 12 + 1\cdot(4+2) + 4 + (1+2)\cdot 2 \\
&\quad + 2(2m + 2 + 2) + 2m \\
&= 20m + 18.
\end{aligned}$$

The m two-input multiplexers at the circuit output are opened simultaneously by the control signal h and its inverse signal. The maximum normalised resistance of the stage that runs through the multiplexer at bit position i equals $r_{\text{XNOR}} + 1 = 3$ for $1 \leq i \leq m-1$ and $r_{\text{HA,sum}} + 1 = 3$ for $i = 0$. Therefore, the CP output stage is *any* of the stages associated with the m least significant bits of φ and thus, the output normalised resistance of the adder equals

$$r_{\text{add}} = 3.$$

The size of the addition circuit equals

$$\begin{aligned}
\mathcal{C}_{\text{add}} &= (m-1)(\mathcal{C}_{\text{FA}} + \mathcal{C}_{\text{OR}} + \mathcal{C}_{\text{XNOR}}) + \mathcal{C}_{\text{HA}} + m\mathcal{C}_{\text{MUX}} + 4\mathcal{C}_{\text{NAND}} + 2\mathcal{C}_{\text{NOR}} + 4\mathcal{C}_{\text{inv}} \\
&= (m-1)(28 + 6 + 12) + 18 + m\cdot 4 + 4\cdot 4 + 2\cdot 4 + 4\cdot 2 \\
&= 50m + 4.
\end{aligned}$$

Assuming that β and γ are outputs of $(m+1)$-bit parallel register and that φ is also stored in such a register, we get the total CP length

$$\begin{aligned}
\mathcal{L}_{\text{add}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}}f_{\text{add}} + \mathcal{L}_{\text{CP,add}} + r_{\text{add}}f_{\text{reg}} \\
&= 22 + 2\cdot 6 + 20m + 18 + 3\cdot 2 \\
&= 20m + 58. \qquad (5.6)
\end{aligned}$$

The area-time performance of the circuit is proportional to

$$\mathcal{C}\mathcal{L}_{\text{add}}^2 \triangleq \mathcal{C}_{\text{add}}(\mathcal{L}_{\text{add}})^2 = (50m+4)(20m+58)^2 = \mathcal{O}\left(m^3\right).$$

Subtraction

The most straightforward method of performing subtraction is to first negate the subtrahend and then add it to the minuend, i.e.

$$\varphi \equiv \beta - \gamma = \beta + (-\gamma) \pmod{2^m+1}.$$

Subtraction can thus be realised using the architectures of Figures 5.5 and 5.7.

5.1.4 Multiplication by Powers of 2

Multiplication of an NBC number by two is easily carried out as a binary shift of the number. Because the modulus reduction operation is not so straightforward and we use an $(m+1)$-bit representation of the binary coded integers of $\mathbb{Z}_{2^m+1}$, multiplication by an arbitrary power of two is preferably carried out as *repeated* multiplication by two. The modulus reduction is carried out after every single shift, i.e. according to the congruence

$$2^n\beta \equiv 2\left(2^{n-1}\beta \bmod 2^m+1\right) \pmod{2^m+1}.$$

Multiplication by 2

Figure 5.8 shows an architecture for computing $\varphi \equiv 2\beta \pmod{2^m+1}$, where $m=4$. The modulus reduction part of the circuit may for example be the architecture in Figure 5.1 or the one in Figure 5.2. Here, due to its favourable AT^2 performance, we only consider the carry look-ahead-type architecture in Figure 5.2.

The input to the residue circuit is $2\beta = \beta_{m-1}2^m + \beta_{m-2}2^{m-1} + \cdots + \beta_1 2^2 + \beta_0 2$ when $0 \leq \beta \leq 2^m-1$, and $2\beta \equiv -2 \equiv 2^m - 1 = (011\cdots111)_2 \pmod{2^m+1}$ when β equals 2^m. This is easily implemented using simplified two-input multiplexers prior to the reduction circuit, as shown in Figure 5.8. Hence, the complete circuit has size

$$\begin{aligned}\mathcal{C}_{\text{mult2}} &= \mathcal{C}_{\text{inv}} + (m-1)(\mathcal{C}_{\text{TG}}+1) + \mathcal{C}_{\text{mod},2} \\ &= 4m\cdot\log_2 m + 17m - 1.\end{aligned}$$

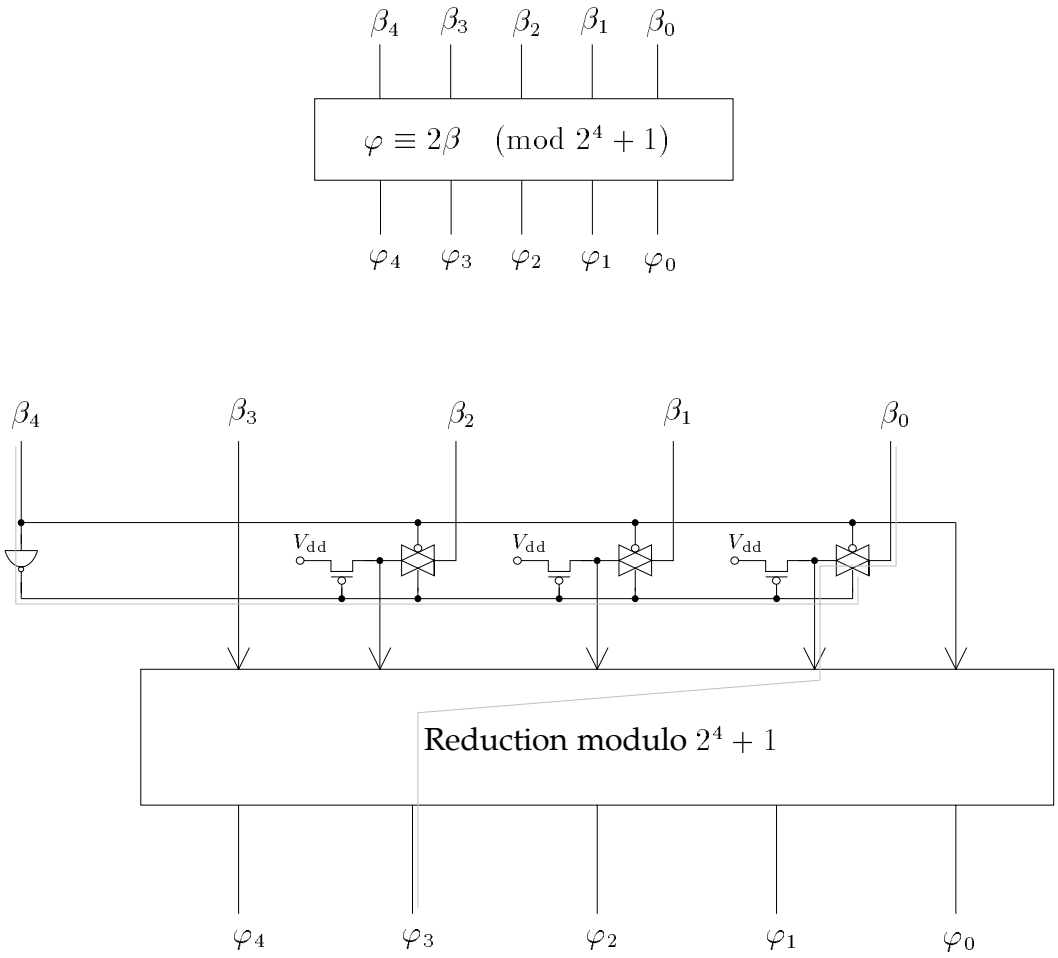


Figure 5.8: *Multiplication by two;* $\varphi \equiv 2\beta \pmod{2^m+1}$ *for* $\beta \in \mathbb{Z}_{2^m+1}$*, where* $m = 4$.

The CP is formed by the two dotted paths in the figure. Because the fan-in of the reduction circuit in Figure 5.2, with respect to its least significant bit position, equals $f_{\text{NOR}} + f_{\text{inv}} = 4$, the total fan-in of the architecture in Figure 5.8, with respect to the β_m-input node, equals

$$f_{\text{mult2}} = f_{\text{inv}} + m - 1 + 4 = m + 5.$$

The output normalised resistance of the architecture is

$$r_{\text{mult2}} = r_{\text{mod},2} = r_0 + 2,$$

where r_0 equals the normalised resistance from the β_0-input node to the supply voltage source (or ground). The internal CP length equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,mult2}} &= r_{\mathrm{inv}} \cdot 2(m-1) + (r_0+1)f_{\mathrm{mod},2} + \mathcal{L}_{\mathrm{CP,mod},2} \\ &= 2(m-1) + 8(r_0+1) + 6m + 8\log_2\, m + 4 \\ &= 8(m + \log_2\, m + r_0) + 10.\end{aligned}$$

If the input β is obtained from parallel register and φ is stored in a similar register, then $r_0 = r_{\mathrm{reg}} = 2$ and the total CP length equals

$$\begin{aligned}\mathcal{L}_{\mathrm{mult2}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} f_{\mathrm{mult2}} + \mathcal{L}_{\mathrm{CP,mult2}} + r_{\mathrm{mult2}} f_{\mathrm{reg}} \\ &= 22 + 2(m+5) + 8(m + \log_2\, m + 2) + 10 + (2+2)\cdot 2 \\ &= 10m + 8\log_2\, m + 66. \qquad (5.7)\end{aligned}$$

The AT^2 performance of the architecture in Figure 5.8 is proportional to the product

$$\begin{aligned}\mathcal{CL}^2_{\mathrm{mult2}} &\triangleq \mathcal{C}_{\mathrm{mult2}}(\mathcal{L}_{\mathrm{mult2}})^2 \\ &= (4m \cdot \log_2\, m + 17m - 1)\,(10m + 8\log_2\, m + 66)^2 \\ &= \mathcal{O}\left(m^3 \log_2\, m\right).\end{aligned}$$

The row of transmission gates and pMOS transistors at the input of the multiplication-by-2 circuit in Figure 5.8 may be exchanged for a row of $m-1$ OR gates. The OR gate in bit position i, for $0 \le i \le m-2$, would have β_m and β_i as its input signals. If such a row of OR gates is used, the fan-in and the output normalised resistance of the circuit are reduced to $(m-1)f_{\mathrm{OR}} = 2(m-1)$ and $r_{\mathrm{OR}} + 1 = 2$, respectively. The total CP length $\mathcal{L}_{\mathrm{mult2}}$ decreases by 30 but the circuit size increases by $3m-5$. With respect to the AT^2 performance, the row of OR gates is preferable to the row of transmission-gates-and-pMOS-transistors only for $m \le 64$. For $m > 64$, the circuit in Figure 5.8 has better area-time performance, compared with an architecture with a row of OR gates at the input.

Multiplication by 2^n

Multiplication by *powers* of two can be carried out by using a feedback coupled multiplication-by-2 circuit with a parallel register in the feedback loop. A block diagram of such a circuit is shown in Figure 5.9. Here, the multiplication-by-2 block is the circuit in Figure 5.8.

For $\beta \in \mathbb{Z}_{2^m+1}$ and $n \in \mathbb{N}$, the arithmetic operation $\varphi \equiv 2^n\beta \pmod{2^m+1}$ is carried out by first, during an initial clock cycle, loading β into the parallel register and then run the circuit for an appropriate number of clock cycles. Because the the integer 2 has order $2m = 2^{t+1}$ modulo 2^m+1 (see Section 2.3.2) we have $\varphi \equiv 2^{n^{(t)}}\beta \pmod{2^m+1}$, where $t = \log_2\, m$. Thus, only the $t+1$

least significant bits of n have to be considered. This implies that the desired product $2^n\beta \bmod 2^m+1$ is present at the circuit output after $n^{(t)}$ clock cycles (not counting the initial clock cycle for loading the feedback register with β).

The chip area A occupied by the circuit in Figure 5.8 is proportional to

$$\mathcal{C}_{\text{mult2n}} = \mathcal{C}_{\text{mult2}} + (m+1)\mathcal{C}_{\text{reg}} = 4m \cdot \log_2\, m + 33m + 15.$$

The internal CP of the circuit is the feedback path from the output of the register element in the most significant bit position, through the multiply-by-2 circuit, to the input of any of the other register elements. Assuming that the output $2^n\beta \bmod 2^m+1$ is stored in an $(m+1)$-bit parallel register, the length of the internal CP equals

$$\begin{aligned}\mathcal{L}_{\text{CP,mult2n}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{mult2}} + \mathcal{L}_{\text{CP,mult2}} + r_{\text{mult2}} \cdot 2 f_{\text{reg}} \\ &= 22 + 2 \cdot (m+5) + 8(m + \log_2\, m + 2) + 10 + (2+2) \cdot 2 \cdot 2 \\ &= 10m + 8\log_2\, m + 74.\end{aligned}$$

After $n^{(t)}+1$ clock cycles, including one clock cycle for initiating the feedback register, the result is shifted out to the output register. Hence, the time T required to perform the entire operation is proportional to

$$\mathcal{L}_{\text{mult2n}} = (n^{(t)}+1)\mathcal{L}_{\text{CP,mult2n}} = (n^{(t)}+1)(10m + 8\log_2\, m + 74),$$

where $0 \le n^{(t)} \le 2m-1$, and the area-time performance AT^2 of the multiply-by-2^n architecture is proportional to the product $\mathcal{C}_{\text{mult2n}}(\mathcal{L}_{\text{mult2n}})^2$.

Note that, because we have $n^{(t)} = n_t 2^t + n_{t-1}2^{t-1} + \cdots + n_0 = n_t \cdot m + n^{(t-1)}$, we can write

$$\varphi \equiv 2^{n^{(t)}}\beta = (2^m)^{n_t} 2^{n^{(t-1)}}\beta \equiv (-1)^{n_t} 2^{n^{(t-1)}}\beta \pmod{2^m+1}. \tag{5.8}$$

Hence, $2^n\beta \bmod 2^m+1$ can be computed by first running the feedback multiplication-by-two circuit $n^{(t-1)}$ clock cycles. Then, if $n_t = 0$ the desired product $\varphi \equiv 2^n\beta \pmod{2^m+1}$ is present at the circuit output and if $n_t = 1$ the result must be negated to obtain φ. The area-time performance of the resulting circuit, which consequently also comprises a negater, is smaller (but not significantly smaller) than the area-time performance of the circuit in Figure 5.9.

It is also possible to design a strictly parallel architecture that performs multiplication by powers of two in one clock cycle. The structure of such an architecture would be similar to the structure of a *barrel shifter*. Such an architecture is considered in Chapter 6 but, however, not in the present chapter.

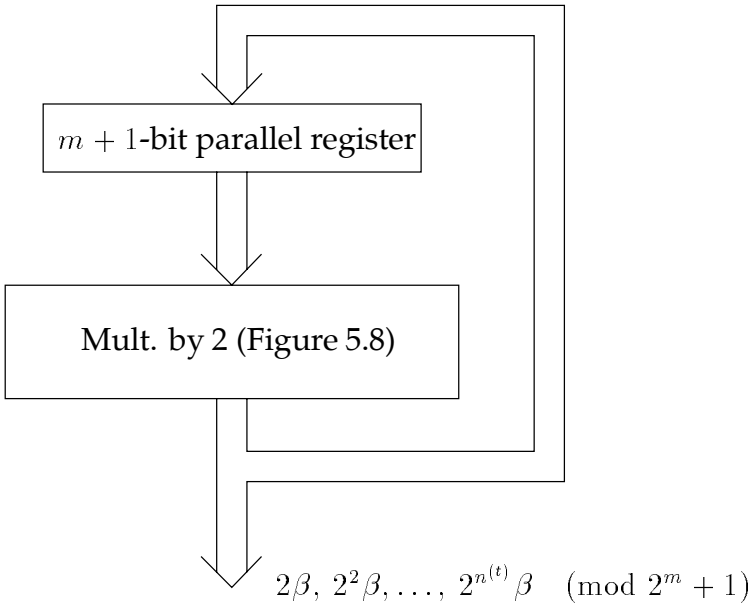


Figure 5.9: *Block diagram for multiplication by powers of two.*

5.1.5 General Multiplication

An overview of some well known approaches to the binary multiplication problem can be found for example in Hwang [52, Ch. 5] and Weste and Eshraghian [113, Ch. 8.2.7]. In principle, there are three types of architectures for general multipliers, namely the serial-type, the serial/parallel-type, and the parallel-type architecture. Factors like form of data transmission, circuit area and computation time requirements, potential for pipelining (to increase the clock frequency), and power dissipation constraints may govern the choice of architecture type. For multiplication in an integer quotient ring, a serial/parallel or strictly parallel architecture is generally preferable to a serial architecture, inter alia with respect to the complexity of performing the modulus reduction operation. This issue was briefly discussed in the beginning of Chapter 4.

Independently of the type of architecture, multiplication of NBC integers is generally performed as sequential addition of partial products. For the multiplicand $\beta = \sum_{i=0}^{m} \beta_i 2^i$ and the multiplier $\gamma = \sum_{i=0}^{m} \gamma_i 2^i$, where as usual $\beta_i,\ \gamma_i \in \mathbb{Z}_2$, we get the product

$$\varphi \equiv \beta \cdot \gamma = \sum_{i=0}^{m} \gamma_i (2^i \beta) \qquad (\mathrm{mod}\ 2^m + 1).$$

A common approach when designing a fast multiplier is to find a way to quickly sum up all the partial products. The serial/parallel multiplier, which is also known as the shift-and-add multiplier, is one of the most well known multipliers. It successively adds the partial products together using *one* feedback parallel adder. In each clock interval, a partial product $2^i\beta \bmod 2^m+1$ is calculated as $2 \cdot (2^{i-1}\beta) \bmod 2^m+1$, i.e. using repeated multiplication by 2 modulo 2^m+1.

A block diagram for a serial/parallel multiplier over $\mathbb{Z}_{2^m+1}$ is shown in Figure 5.10. The parallel-input multiplicand β and the serial-input multiplier γ are initially loaded into the registers R_1 and SR, respectively. The register R_2 is initiated with the all-zero word. These initiations are carried out during one clock cycle. After the following i clock cycles, the $(m+1)$-bit parallel register R_1 contains the partial product $2^i\beta \bmod 2^m+1$. Each output bit from register R_1 is fed both to one of the inputs of a two-input AND gate and to the input of the multiplication-by-2 circuit. The bit-serial output γ_i of the shift register SR is connected to the second input of each of these $m+1$ AND gates, making the fan-out of the shift register equal to $(m+1)f_{\text{AND}}$. Hence, the value of the least significant bit of SR controls, in each clock interval, whether the all-zero word (for $\gamma_i = 0$) or the partial product in R_1 (for $\gamma_i = 1$) is to be added to the contents of R_2.

The CP of the serial/parallel multiplier architecture in Figure 5.10 is the path from the output of shift register SR through an AND gate, the parallel adder, and into one of the registers elements in R_2.[5] Using the carry ripple-type adder in Figure 5.7, the length of this CP equals

$$\begin{aligned}\mathcal{L}_{\text{CP,mult}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} \cdot (m+1)f_{\text{AND}} + \mathcal{L}_{\text{AND}} \\ &\quad + r_{\text{AND}} f_{\text{add}} + \mathcal{L}_{\text{CP,add}} + r_{\text{add}} f_{\text{reg}} \\ &= 22 + 2(m+1)2 + 4 + 6 + 20m + 18 + 3 \cdot 2 \\ &= 24m + 60.\end{aligned}$$

After $m+1$ clock cycles, the product $\varphi \equiv \beta \cdot \gamma \pmod{2^m+1}$ is obtained in register R_2. An initial clock cycle is required for loading the registers with their initial values and an extra clock cycle is required to shift φ from register R_2 into an output register (not shown in the figure). Hence, the total computation time T is proportional to

$$\mathcal{L}_{\text{mult}} = (m+2)\mathcal{L}_{\text{CP,mult}} = 24m^2 + 132m + 180.$$

[5] If the adder architecture in Figure 5.7 is adopted here, the CP ends in the register element in the next most significant bit position $(m-1)$ of the parallel register R_2.

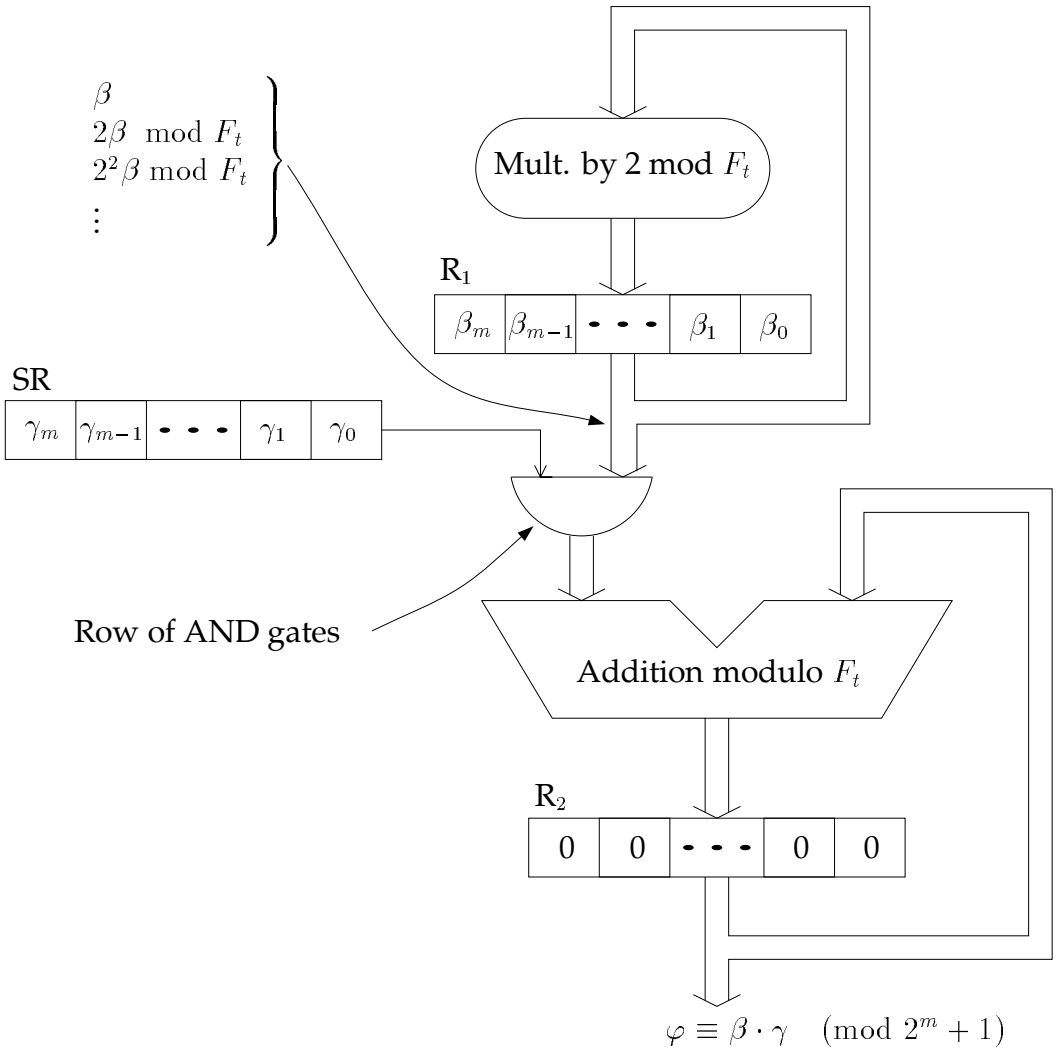


Figure 5.10: *The block diagram for a serial/parallel multiplier. The product* $\varphi \equiv \beta \cdot \gamma \pmod{F_t}$*, where* $F_t = 2^m + 1$*, is generated and stored in register* R_2 *after* $m + 1$ *clock cycles. The initial contents of the registers* R_1 *and* R_2 *are* β *and the all-zero word, respectively, and the shift register* SR *is initiated with* γ*. These initial values are shown in the respective registers in the figure.*

Using the multiplication-by-two circuit in Figure 5.8, the chip area A occupied by the circuit in Figure 5.10 is proportional to its size

$$\begin{aligned}\mathcal{C}_{\text{mult}} &= \mathcal{C}_{\text{mult2}} + 3(m+1)\mathcal{C}_{\text{reg}} + (m+1)\mathcal{C}_{\text{AND}} + \mathcal{C}_{\text{add}} \\ &= 4m \cdot \log_2\, m + 17m - 1 + 3(m+1)16 + (m+1)\cdot 6 + 50m + 4 \\ &= 4m \cdot \log_2\, m + 121m + 57.\end{aligned}$$

The area and/or time complexities of the serial/parallel multiplier may be reduced by for example replacing the parallel carry ripple-type adder with an adder that has better area-time performance. For standard binary serial/parallel multiplication of binary coded two's complement numbers, the efficiency of computing the sum of partial products may be speeded up by adopting a different multiplication scheme, see for example Chapter 5 in Hwang's book on computer arithmetic [52]. However, using the NBC representation of the integers of $\mathbb{Z}_{2^m+1}$, it seems as if none of these schemes yields a serial/parallel architecture whose area-time performance is significantly improved, compared to the area-time performance of the serial/parallel multiplier in Figure 5.10. The area-time performance AT^2 of the latter multiplier is proportional to

$$\mathcal{CL}^2_{\text{mult}} \triangleq \mathcal{L}_{\text{mult}}(\mathcal{C}_{\text{mult}})^2 = \mathcal{O}\left(m^5 \log_2\, m\right).$$

Remark: We have not investigated the properties of any bit-parallel architecture for NBC multiplication in $\mathbb{Z}_{2^m+1}$. However, a promising candidate for such an architecture is a modified version of the pipelined array multiplier suggested by Benaissa et al. [11, Fig. 4]. Their multiplier is based on an NBC representation of both the multiplier and the multiplicand. Because this multiplier is basically a *diminished–1* multiplier, its properties are investigated in Section 6.3.6 (see page 138). A block diagram of the multiplier is shown in Figure 6.21.

5.1.6 Exponentiation of the Transform Kernel

Consider the exponentiation

$$\varphi \equiv \beta^n \pmod{2^m + 1}, \tag{5.9}$$

where $\beta \in \mathbb{Z}_{2^m+1}$ and the exponent n is an integer. Because the order of every element of $\mathbb{Z}_{2^m+1}$ divides $\phi(2^m+1)$, where ϕ is Euler's totient function,[6] the

[6] See Corollary 8.1.1 in Rosen's book [84]. The totient function $\phi(2^m+1)$ equals 2^m in the Fermat prime fields, i.e. for $m = 1,\ 2,\ 4,\ 8,\ 16$.

only part of the exponent n that has to be considered is $n \bmod \phi(2^m+1)$. In particular, when β is the transform kernel ω, the order of β modulo 2^m+1 equals the transform length $N = 2^b$ for some integer b (see (2.4) in Section 2.3.2). Therefore, when computing powers of the transform kernel, we use the exponent $n \bmod N$.

There exist several algorithms for integer exponentiation. Probably the most well known method is the so called *binary method*, which is described by Knuth in [56, Ch. 4.6.3]. It is based on the NBC extension of the exponent n. The r-bit NBC integer n can be written on the form

$$n \equiv n_{r-1}2^{r-1} + n_{r-2}2^{r-2} + \cdots + n_1 2 + n_0 \pmod{q},$$

where $r \triangleq \lfloor \log_2 (n \bmod q) \rfloor + 1$,[7] $n_0, \ldots, n_{r-1} \in \mathbb{Z}_2$, and $q = \phi(2^m+1)$ (for arbitrary $\beta \in \mathbb{Z}_{2^m+1}$) or $q = N$ (for $\beta = \omega$). Consequently, the congruence in (5.9) can be written as

$$\varphi \equiv \left(\left(\cdots \left(\left(\beta^{n_{r-1}} \right)^2 \beta^{n_{r-2}} \right)^2 \beta^{n_{r-3}} \cdots \beta^{n_2} \right)^2 \beta^{n_1} \right)^2 \beta^{n_0} \pmod{2^m+1}.$$

In the binary method, the right-hand side of this congruence is evaluated using repeated squaring and multiplication. Hence, depending on n, $r-1$ squarings and at most the same number of multiplications are required to perform the exponentiation. In a conventional circuit for exponentiation we use a full-width exponent representation, i.e. we have $r = \lfloor \log_2 (\phi(2^m+1)-1) \rfloor + 1$ in the general case[8] (for arbitrary nonzero $\beta \in \mathbb{Z}_{2^m+1}$) and $r = \lfloor \log_2 (N-1) \rfloor + 1 = b$ for $\beta = \omega$. The multiplications required to perform the exponentiation are *general* multiplications. For some choices of base β, these multiplications may be carried out in a simpler way, but such simplified multiplications are not considered here.

Zuras [115] discusses how to find the fastest way to square (and multiply) large integers in software: Denote by T_{mult} and T_{square} the computation times for general multiplication and squaring, respectively. Because squaring is a special case of multiplying, we trivially have $T_{\text{square}} \leq T_{\text{mult}}$. There is no known algorithm for exponentiation that is *significantly* faster than general multiplication. From the equation

$$A \cdot B = \frac{(A+B)^2 - (A-B)^2}{4}$$

it follows that a multiplication can be carried out as two squarings, three additions, and one multiplication by 2^{-2}. Assuming that addition and multiplication by 2^{-2} takes at most $\mathcal{O}(m)$ time (see for example (5.6) and (5.7)), where

[7] For $x \in \mathbb{R}$, the expression $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x.

[8] When 2^m+1 is prime we get $r = m$.

m is the operand word bit length, we thus get

$$T_{\text{mult}} \leq 2T_{\text{square}} + \mathcal{O}(m).$$

Hence, as stated by Zuras, even though someone may discover an algorithm for squaring that is faster than any *existing* multiply algorithm, any squaring algorithm can be used to construct a multiply algorithm that is not more than a constant slower than the squaring algorithm. Regarding the NBC representation of the elements of $\mathbb{Z}_{2^m+1}$ we do not consider any specially designed architecture for squaring. Squarings are performed as general multiplications, which means that exponentiation requires at most $2(r-1)$ multiplications, where $r = \lfloor \log_2(\phi(2^m+1)-1) \rfloor + 1$ for arbitrary nonzero $\beta \in \mathbb{Z}_{2^m+1}$ and $r = b$ for $\beta = \omega$. Hence, using the NBC representation and the binary method as described above, exponentiation in $\mathbb{Z}_{2^m+1}$ can be performed in $\mathcal{O}(2(r-1)\mathcal{L}_{\text{mult}})$ time.

Alternative methods of performing integer exponentiation are described for example in Chapter 4.6.3 in Knuth's book [56] and in Zuras' paper [115]. Bocharova and Kudryashov [18], [19] investigate exponentiation schemes based on different source codes. Gollmann et al. [46] consider exponentiation based on a signed-digit representation of the exponent. See also the articles on integer exponentiation in the reference list in Gollmann's paper [46]. Compared with the binary method, most other algorithms for integer exponentiation reduce the number of true multiplications, often by processing several bits of the binary (or signed-digit) representation of the exponent at a time, which for some algorithms is done to the cost of a precomputed look-up table. The number of squarings are approximately the same for most algorithms. In the present chapter we only consider the above binary method.

In Section 2.3.2 we showed that for some sequence lengths N there exist suitable choices of the kernel ω for which the different powers of the kernel are easily calculated. For example, for the combinations $(N, \omega) = (2m, 2)$ and $(N, \omega) = (4m, \sqrt{2})$, multiplication by a power of ω can be simply carried out as binary shifts in the former case and a pair of binary shifts and one addition in the latter case.

For transforms of arbitrary lengths, the powers of the transform kernel are either directly calculated when needed or precomputed and stored in a memory (look-up table) from which they are read when needed. For the direct calculations, we use the above binary method for exponentiation. When the powers of ω are precomputed, the exponentiations are suitably carried out as repeated multiplication by ω, i.e. $\omega^2 = \omega \cdot \omega$, $\omega^3 = \omega^2 \cdot \omega$, $\omega^4 = \omega^3 \cdot \omega, \ldots$. We do not consider the complexity of these precomputations.

In Fermat prime fields, i.e. in $\mathbb{Z}_{2^m+1}$ for $m = 1, 2, 4, 8, 16$, there are more ways of performing exponentiation. For example, in Section 7.2.1 we describe methods of performing exponentiation with respect to the *polar representation*.

5.2 Summary

In Table 5.1 we have summarised the sizes, the fan-ins, the internal and total CP lengths, the output normalised resistances, and the area-time performances AT^2 of the architectures in the present chapter. Note that the modulus reduction operation is included in the other four operations.

Operation	Figure	Subscript name	Size $\mathcal{C}$	Fan-in f	Internal CP length $\mathcal{L}_{\text{int}}$	
Modulus reduction	5.1	mod,1	$22m - 6$	8	$16m - 6$	
	5.2	mod,2	$4m \cdot \log_2 m + 14m$	8	$6m + 8\log_2 m + 4$	
Negation	5.5	neg	$20m - 22$	2	$30m - 10$	
Addition	5.7	add	$50m + 4$	6	$20m + 18$	$\cdots$
Mult. by 2	5.8	mult2	$4m \cdot \log_2 m + 17m - 1$	$m + 5$	$8(m + \log_2 m + r_0) + 10$	
Mult. by 2^n	5.9	mult2n	$4m \cdot \log_2 m + 33m + 15$	—	$10m + 8\log_2 m + 74$	
General mult.	5.10	mult	$4m \cdot \log_2 m + 121m + 57$	—	$24m + 60$	
Exponentiation	—	—	$\longrightarrow$	$\longrightarrow$		

	Norm. output res. r_o	Total CP length $\mathcal{L}$ (including registers)	Area-time perf. $\mathcal{CL}^2$
	$r_{m-1} + 1$	$16m + 38$	$\mathcal{O}(m^3)$
	$r_{m-1} + 1$	$6m + 8\log_2 m + 48$	$\mathcal{O}(m^3 \log_2 m)$
	2	$30m + 20$	$\mathcal{O}(m^3)$
$\cdots$	3	$20m + 58$	$\mathcal{O}(m^3)$
	$r_0 + 2$	$10m + 8\log_2 m + 66$	$\mathcal{O}(m^3 \log_2 m)$
	—	$(n^{(t)} + 1)(10m + 8\log_2 m + 74)$	$\mathcal{O}\left((n^{(t-1)})^2 m^3 \log_2 m\right)$
	—	$24m^2 + 132m + 180$	$\mathcal{O}(m^5 \log_2 m)$
At most $2(r-1)$ multiplications are required, where $r = \lfloor \log_2(\phi(2^m + 1) - 1) \rfloor + 1$ or $r = b$			

Table 5.1: *The complexity parameters of the architectures in the present chapter.*

Chapter 6

The Diminished–1 Representation

6.1 Linearly Transformed Representations

In this chapter investigate properties of arithmetic operations in $\mathbb{Z}_{2^m+1}$, with respect to a *linear coordinate transformation* of the $(m+1)$-bit normal binary coded (NBC) integers of $\mathbb{Z}_{2^m+1}$. In the resulting number system, an NBC integer $\gamma \in \mathbb{Z}_{2^m+1}$ is represented by the binary coded integer

$$T(\gamma) \equiv k\gamma + l \pmod{2^m+1}, \tag{6.1}$$

where $k,\, l \in \mathbb{Z}_{2^m+1}$. The reverse code translation (from $T(\gamma)$ to γ) can be written as $\gamma \equiv k^{-1}(T(\gamma) - l) \pmod{2^m+1}$. Consequently, the reverse code translation only exists if k has a multiplicative inverse k^{-1} in $\mathbb{Z}_{2^m+1}$, i.e. if

$$\gcd(k,\, 2^m+1) = 1.$$

Depending on the constants k and l, we obtain various VLSI architectures for arithmetic operations in $\mathbb{Z}_{2^m+1}$. Trivially, for $k = 1$ and $l = 0$ we get the NBC representation of $T(\gamma) = \gamma$ and consequently the architectures considered in Chapter 5.

Because every translated integer $T(\gamma)$ is an $(m+1)$-bit NBC integer in $\mathbb{Z}_{2^m+1}$, reduction modulo 2^m+1 can be performed using the procedure described in Section 5.1.1 for any k and l. In Section 6.2 we investigate how to choose the constants k and l so that the modulus reduction operation can be incorporated into the various arithmetic operations in a straightforward way, i.e. in a way that minimises the computational complexity of each operation.

For the sake of convenience, we occasionally denote a translated NBC integer $T(\gamma)$ by $\hat{\gamma}$, i.e. for $\hat{\gamma}_m,\ \hat{\gamma}_{m-1},\ \ldots,\ \hat{\gamma}_1,\ \hat{\gamma}_0 \in \mathbb{Z}_2$ we have

$$T(\gamma) \triangleq \hat{\gamma} = 2^m\hat{\gamma}_m + 2^{m-1}\hat{\gamma}_{m-1} + \cdots + 2\hat{\gamma}_1 + \hat{\gamma}_0.$$

6.1.1 Arithmetic Operations

For arbitrary $\beta,\ \gamma,\ k,\ l \in \mathbb{Z}_{2^m+1}$, where $\gcd(k,\ 2^m+1) = 1$, we get the following arithmetic:

Negation

By (6.1) we get $T(\gamma) + T(-\gamma) \equiv 2l \pmod{2^m+1}$ and thus

$$T(-\gamma) \equiv -T(\gamma) + 2l \pmod{2^m+1}.$$

By (5.5) we get the congruence $-T(\gamma) \equiv \overline{T(\gamma)} + 3 \pmod{2^m+1}$, which gives

$$T(-\gamma) \equiv \overline{T(\gamma)} + 3 + 2l \pmod{2^m+1}. \tag{6.2}$$

Thus, the integer $-\gamma$ is represented by $\overline{T(\gamma)} + 3 + 2l \mod 2^m+1$, where $\overline{T(\gamma)}$ is the one's complement of the $(m+1)$-bit translated NBC integer $T(\gamma)$.

Addition

$$T(\beta+\gamma) \equiv k(\beta+\gamma) + l \equiv T(\beta) + T(\gamma) - l \pmod{2^m+1}. \tag{6.3}$$

For the sake of simplicity we sometimes use the symbol $\oplus$ to denote addition between translated symbols.[1] Hence, we define such an addition as

$$T(\beta) \oplus T(\gamma) \triangleq T(\beta+\gamma) \pmod{2^m+1}. \tag{6.4}$$

Subtraction

The congruence

$$T(\beta-\gamma) \equiv T(\beta) \oplus T(-\gamma) \pmod{2^m+1} \tag{6.5}$$

follows directly from (6.3) and (6.4), i.e. subtraction is performed in the traditional way by first negating the subtrahend and then adding it to the minuend.

[1] Chang et al. [32] use the same notation.

Note that the symbol $\oplus$ denotes the logical XOR function whenever it appears in a Boolean expression.

Multiplication by Powers of 2

We expand $T(2^n\gamma)$ as

$$T(2^n\gamma) \equiv k2^n\gamma - l + 2^n l - 2^n l \equiv 2^n T(\gamma) - (2^n - 1)l \tag{6.6}$$

$$\equiv \sum_{i=1}^{2^n-1} T(\gamma) \pmod{2^m + 1}, \tag{6.7}$$

where $\sum$ denotes the special summation of translated symbols. The special case of simple multiplication by 2,

$$T(2\gamma) \equiv 2T(\gamma) - l \pmod{2^m + 1}, \tag{6.8}$$

can also be directly obtained from the addition formula (6.3). The product $2T(\gamma)$ is simply obtained by shifting the NBC integer $T(\gamma)$ one bit to the left. The translated product $T(2^n\gamma)$ can be calculated in a way that is computationally more efficient than the direct computation of (6.6). We have

$$\begin{aligned} T(2^n\gamma) &\equiv k2^n\gamma + l \\ &= 2(k2^{n-1}\gamma + l) - l \\ &= 2T(2^{n-1}\gamma) - l \pmod{2^m + 1}, \end{aligned} \tag{6.9}$$

which is simply computed using repeated multiplication by 2 and addition. The modulus reduction is performed after every multiplication by 2 and addition of $-l$.

General Multiplication

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv k\beta\gamma + l \equiv kk^{-1}(T(\beta) - l)k^{-1}(T(\gamma) - l) + l \\ &= k^{-1}\left(T(\beta)T(\gamma) - l(T(\beta) + T(\gamma) - l)\right) + l \pmod{2^m + 1}. \end{aligned} \tag{6.10}$$

Because $T(\beta)$ and $T(\gamma)$ are NBC integers, it is possible to simplify (6.10). By writing $T(\gamma)$ on the form $T(\gamma) = \sum_{i=0}^{m} \hat{\gamma}_i 2^i$, where $\gamma_i \in \{0, 1\}$, we get

$$
\begin{aligned}
T(\beta\cdot\gamma) &\equiv k\beta\gamma + l = \beta(k\gamma + l) - l\beta + l \\
&\equiv \beta\cdot T(\gamma) - lk^{-1}(k\beta + l) + l^2k^{-1} + l \\
&\equiv \beta\sum_{i=0}^{m}\hat{\gamma}_i 2^i + lk^{-1}(l + k - T(\beta)) \\
&= \sum_{i=0}^{m}(\hat{\gamma}_i 2^i\beta + l) - ml + lk^{-1}(l + k - T(\beta)) - l \\
&\equiv \left(\bigoplus_{i=0}^{m} T(\hat{\gamma}_i 2^i\beta)\right) \oplus lk^{-1}(l + k - T(\beta)) \pmod{2^m + 1}. \qquad (6.11)
\end{aligned}
$$

In some applications one may wish to represent either the multiplicand or the multiplier as an NBC number. For example, constants or the Fermat number transform coefficients $\omega^{\pm kn}$ may just as well be stored in that format. By writing the NBC multiplier γ on the form $\gamma = \sum_{i=0}^{m}\gamma_i 2^i$ we get

$$
\begin{aligned}
T(\beta\cdot\gamma) &\equiv k\beta\gamma + l = k\beta\sum_{i=0}^{m}\gamma_i 2^i + l \\
&= \sum_{i=0}^{m}(k\gamma_i 2^i\beta + l) - l(m+1) + l \equiv \sum_{i=0}^{m} T(k\gamma_i 2^i\beta) - lm \\
&\equiv \bigoplus_{i=0}^{m} T(k\gamma_i 2^i\beta) \pmod{2^m + 1}, \qquad (6.12)
\end{aligned}
$$

which apparently has a simpler structure than both (6.10) and (6.11). Apparently, the efficiency of computing (6.10) and (6.11) depends on which values are assigned to k and l. The multiplication procedure according to (6.12) depends on the value of k, but not l. It is also possible to obtain an expression for general multiplication which involves l but not k:

$$
\begin{aligned}
T(\beta\cdot\gamma) &\equiv k\beta\gamma + l = (k\beta + l)\sum_{i=0}^{m}\gamma_i 2^i - l\gamma + l \\
&= \sum_{i=0}^{m}\gamma_i 2^i(k\beta + l) - l\gamma + l \\
&= \sum_{i=0}^{m}\gamma_i(k2^i\beta + l) - l\sum_{i=0}^{m}\gamma_i + l\sum_{i=0}^{m}\gamma_i 2^i - l\gamma + l
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=0}^{m} \gamma_i T(2^i\beta) + l\left(1 - \sum_{i=0}^{m} \gamma_i\right) \\
&= \sum_{i=0}^{m} \gamma_i T(2^i\beta) + l\left(\sum_{i=0}^{m} (1-\gamma_i) - m\right) \\
&= \sum_{i=0}^{m} (\gamma_i T(2^i\beta) + \overline{\gamma_i} l) + m - m(l+1) \\
&= \sum_{i=0}^{m} (\gamma_i T(2^i\beta) + \overline{\gamma_i} l) - m(l+1) \pmod{2^m+1}. \qquad (6.13)
\end{aligned}
$$

In Section 6.3.6 we consider various multiplication procedures which are based on the above congruences (6.10), (6.11), (6.12), and (6.13).

Exponentiation

The formula for general exponentiation is

$$T(\gamma^n) \equiv k\gamma^n + l \equiv k^{1-n}\,(T(\gamma) - l)^n + l \pmod{2^m+1}. \qquad (6.14)$$

6.2 The Use of a Zero Indicator

Generally speaking, the best choice of k and l in (6.1) yields optimum complexity and performance of the corresponding VLSI architectures for arithmetic operations. Among the arithmetic operations considered in the previous section, the code translation (Equation (6.1)), general multiplication according to (6.10), (6.11), and (6.12), and exponentiation (Equation (6.14)) are the only ones involving the constant k. It is involved in these operations in the following ways:

- Multiplication by k and k^{-1}.
- Multiplication by lk^{-1} (or $-lk^{-1}$) and addition by $l + k$.
- Multiplication by k^{1-n}.

These operations are simplest carried out if $k = 1$, $k = -l$, and $k = 1$ respectively. The operations then reduce to multiplication by one (for all equations involving k) and addition by zero (Equation (6.11)). Hence, we assert

that *choosing $k = 1$ is the best choice with respect to the simplification of the above operations.*

Now, let us consider the choice of the constant l. We have seen that addition of translated symbols occurs in several operations (Equations (6.3), (6.5), (6.7), (6.10), (6.11), (6.12), and (6.13)). We therefore first focus on the sum

$$T(\beta) \oplus T(\gamma) = T(\beta + \gamma) \equiv T(\beta) + T(\gamma) - l \pmod{2^m + 1}.$$

in (6.3). Multiplication by two,

$$T(2\gamma) \equiv 2T(\gamma) - l \pmod{2^m + 1},$$

is another operation of special interest, because it is involved in the computation of general multiplication. Multiplication by two is of course a special case of addition, but the product $2T(\gamma)$ is preferably carried out as a binary shift of $T(\gamma)$ instead of ordinary addition. We would, however, like to carry out the addition by $-l$ followed by the reduction modulo $2^m + 1$ as simply as possible.

The following $(m+1)$-bit NBC integers modulo $2^m + 1$ are the 2^m elements of $\mathbb{Z}_{2^m+1}$:

$$\begin{aligned} 1\,00\cdots00 &= 2^m \\ 0\,11\cdots11 &= 2^m - 1 \\ 0\,11\cdots10 &= 2^m - 2 \\ &\vdots \\ 0\,00\cdots01 &= 1 \\ 0\,00\cdots00 &= 0 \end{aligned}$$

It would be very convenient, at least from an implementation point of view, if 2^m represents the zero element. Because the NBC integer 2^m is the only element of $\mathbb{Z}_{2^m+1}$ which has a one in its most significant bit position , it would then be enough to check in *one* bit position whether an element is zero. Such a procedure can be helpful for example when computing sums and products; addition by zero ($T(\gamma + 0) = T(\gamma)$) and multiplication by zero ($T(\gamma \cdot 0) = T(0)$) are two operations that can be simply carried out in VLSI during a single clock interval.

When representing the zero element by the integer 2^m, the nonzero integers $1, 2, 3, \ldots, 2^m$ are consequently represented by the (m-bit) NBC integers of $\mathbb{Z}_{2^m}$. Hence, we can use an m-bit arithmetic for the nonzero elements of $\mathbb{Z}_{2^m+1}$, which from a complexity point of view is preferable to the $(m+1)$-bit arithmetic associated with the NBC representation in Chapter 5.

Henceforth, the translated element 2^m is called the *zero indicator*. Thus, by letting $T(\gamma) = 2^m$ represent $\gamma = 0$ we get

$$l = -1$$

from (6.1), which is the same value of l that is obtained from the choice of k:

For $k = 1$ and $k = -l$, we get $l = -1$.

The congruences (6.3) and (6.8) then change to

$$T(\beta + \gamma) \equiv T(\beta) + T(\gamma) + 1 \pmod{2^m + 1} \tag{6.15}$$

$$T(2\gamma) \equiv 2T(\gamma) + 1 \pmod{2^m + 1}, \tag{6.16}$$

respectively. It is also interesting to note that, for $l = -1$, negation (Equation (6.2)) and general multiplication according to (6.13) simplify to

$$T(-\gamma) \equiv \overline{T(\gamma)} + 1 \pmod{2^m + 1} \tag{6.17}$$

$$T(\beta \cdot \gamma) \equiv \sum_{i=0}^{m} (\gamma_i T(2^i \beta) - \overline{\gamma_i}) \pmod{2^m + 1}, \tag{6.18}$$

respectively. Obviously, the addition by 1 modulo $2^m + 1$ appears in the three congruences (6.15), (6.16), and (6.17) (and actually also in the congruence (6.18), which is formed by m additions of the type in (6.15)).[2]

For an arbitrary $(m+1)$-bit NBC integer $\hat{\theta} < 2^{m+1}$ the congruence $\hat{\varphi} \equiv \hat{\theta} + 1 \pmod{2^m + 1}$ can be simplified as

$$\begin{aligned} \hat{\varphi} &\equiv \hat{\theta} + 1 = \hat{\theta}^{(m-1)} + 1 + \hat{\theta}_m 2^m \\ &\equiv \hat{\theta}^{(m-1)} + 1 - \hat{\theta}_m \\ &= \hat{\theta}^{(m-1)} + \overline{\hat{\theta}_m} \pmod{2^m + 1}, \end{aligned} \tag{6.19}$$

where $\hat{\theta}^{(m-1)} = \left(\hat{\theta}_{m-1}, \hat{\theta}_{m-2}, \ldots, \hat{\theta}_0\right)_2$. We thus have

$$\hat{\varphi} \equiv \begin{cases} \hat{\theta}^{(m-1)}; & \text{for } \hat{\theta}_m = 1 \\ \hat{\theta}^{(m-1)} + 1; & \text{for } \hat{\theta}_m = 0 \end{cases} \pmod{2^m + 1}, \tag{6.20}$$

which can easily be computed using for example a chain of half adders. This is further discussed in Section 6.3.1. The sum $\hat{\varphi}$ may be associated with

[2] Note that by letting $3 + 2l$ of (6.2) be equal to $-l$ of (6.3), we get $l = -1$ and thus $3 + 2l = 1$ and $-l = 1$.

$T(\beta+\gamma)$, $T(2\gamma)$, or $T(-\gamma)$ and the addend $\hat{\theta}$ may be associated with $T(\beta)+T(\gamma)$, $2T(\gamma)$, or $\overline{T(\gamma)}$ in (6.15), (6.16), or (6.17), respectively.

From the above arguments we conclude that, from a computational complexity point of view and with respect to the complexity and performance of the VLSI architectures for arithmetic operations, the best choice of the constants k and l in (6.1) is $(k,l) = (1,-1)$.

McClellan's Representation

In 1976, McClellan [65] proposed a way of representing the integers of $\mathbb{Z}_{2^m+1}$. By letting the $(m+1)$-bit binary coded number $T(\gamma) \triangleq \hat{\gamma} = (\hat{\gamma}_m, \hat{\gamma}_{m-1}, \ldots, \hat{\gamma}_0)_2$ represent the NBC integer $\gamma \in \mathbb{Z}_{2^m+1}$, the coding scheme is defined as follows:

$$\begin{array}{ll} \text{If } \hat{\gamma}_m = 1, & \text{then } \gamma = 0. \\ \text{If } \hat{\gamma}_m = 0, & \text{then } \gamma \equiv \sigma_{m-1}2^{m-1} + \sigma_{m-2}2^{m-2} + \cdots + \sigma_0 \pmod{2^m+1} \end{array},$$

where

$$\sigma_j = \begin{cases} 1 & \text{if } \hat{\gamma}_j = 1 \\ -1 & \text{if } \hat{\gamma}_j = 0 \end{cases}.$$

Thus, McClellan uses binary weightings with ± 1 instead of 0 and 1. The core of his representation is that the binary coded integer 2^m represents the integer 0, i.e. he uses 2^m as a zero indicator. Consequently, all the nonzero elements have a zero in their most significant bit position and thus it is possible to perform m-bit arithmetic operations on these elements.

For a nonzero integer γ, for which $\hat{\gamma}_m = 0$, we have the relation $\sigma_j = 2\hat{\gamma}_j - 1$. Therefore, γ can be expanded as

$$\begin{aligned} \gamma &\equiv (2\hat{\gamma}_{m-1}-1)2^{m-1} + (2\hat{\gamma}_{m-2}-1)2^{m-2} + \cdots + (2\hat{\gamma}_1 - 1)2 + (2\hat{\gamma}_0 - 1) \\ &\equiv 2\left(\hat{\gamma}_{m-1}2^{m-1} + \hat{\gamma}_{m-2}2^{m-2} + \hat{\gamma}_{m-3}2^{m-3} + \cdots + \hat{\gamma}_1 2 + \hat{\gamma}_0\right) + 2 \\ &= 2\hat{\gamma}^{(m-1)} + 2 \pmod{2^m+1}. \end{aligned}$$

Because $\hat{\gamma}_m$ equals zero we get $\gamma \equiv 2\hat{\gamma}^{(m-1)}+2 = 2\hat{\gamma}+2 = 2T(\gamma)+2 \pmod{2^m+1}$. It shows that this congruence also holds for the zero element; $2T(0)+2 = 2\cdot 2^m + 2 \equiv 2\cdot -1 + 2 = 0$. Hence, the congruence

$$\gamma \equiv 2T(\gamma) + 2 \pmod{2^m+1}$$

holds for every element $\gamma \in \mathbb{Z}_{2^m+1}$. Because we have $2^{-1} \equiv (2^m+2)2^{-1} = 2^{m-1}+1 \pmod{2^m+1}$ the code translation from γ to $T(\gamma)$ is performed according to the congruence

$$T(\gamma) \equiv 2^{-1}\gamma - 1 \equiv (2^{m-1}+1)\gamma - 1 \pmod{2^m+1}.$$

We thus get McClellan's element representation by choosing $k = 2^{m-1} + 1$ and $l = -1$ in (6.1).

Leibowitz' Representation

Also in 1976, Leibowitz [58] presented another way of representing the integers of $\mathbb{Z}_{2^m+1}$. In his article, he mentions that McClellan's element representation belongs to the set of element translations of the form

$$T(\gamma) \equiv k\gamma - 1 \pmod{2^m + 1}; \qquad k,\, k^{-1} \in \mathbb{Z}_{2^m+1},$$

which all give the same simplified binary arithmetic modulo $2^m + 1$. However, this is true only for operations like negation (Equation (6.2)), addition (Equation (6.3)), multiplication by two (Equation (6.8)), and general multiplication according to (6.12). The integer k is not involved in any of these operations. Leibowitz claimed that that the choice $k = 1$ will give the simplest code translation. Then, γ is simply obtained from $T(\gamma)$ by adding 1 to $T(\gamma)$ modulo $2^m + 1$. The reverse operation is carried out by diminishing γ by 1 modulo $2^m + 1$. Owing to this fact, Leibowitz calls his element representation the *diminished–1* representation.

The diminished–1 representation has been adopted in most published architectures since 1976. As indicated above, the two main reasons for that should be the utilisation of the element 2^m as a zero indicator ($l = -1$), and the simplified element translation ($k = 1$). A very common application to Fermat integer quotient ring computations is the computation of the Fermat number transform of lengths $2m$ and $4m$, for which the transform kernel ω is preferably chosen as 2 and $\sqrt{2}$, respectively.[3] Then it is possible to compute the Fermat number transform using bit shifts and additions but no general multiplication or exponentiation. Hence, *no operation involving k is needed to compute such a transform*. If the translation from the normal binary representation to the diminished–1 representation and vice versa must take place, then, as asserted above, $k = 1$ is the best choice.

[3] See Section 2.3.2.

6.3 The Diminished–1 Representation

6.3.1 Code Translation

Obviously, the code translation from the NBC integer $\gamma \in \mathbb{Z}_{2^m+1}$ to the diminished–1 integer $T(\gamma) \equiv \gamma - 1 \pmod{2^m+1}$ and the reverse translation $\gamma \equiv T(\gamma) + 1 \pmod{2^m+1}$ only involve subtraction by one and addition by one modulo $2^m + 1$, respectively.

NBC to Diminished–1 Representation

The forward translation $T(\gamma) \triangleq \hat{\gamma} \equiv \gamma - 1 \pmod{2^m+1}$, where $0 \leq \gamma \leq 2^m$, can quite easily be carried out using a simplified parallel adder. As in Section 5.1.1, we first compute the sum $\sigma \triangleq \gamma^{(m-1)} + (2^m - 1) = (c_m, \sigma_{m-1}, \sigma_{m-2}, \ldots, \sigma_1, \sigma_0)_2$, where c_m is the carry output from the most significant bit position of the adder and σ_i is the adder sum output in bit position i, for $i = 0, 1, \ldots, m-1$. With one of the adder input signals in bit position i high and the second input signal equal to γ_i we get, in accordance with (5.2) and (5.2), the carry and sum outputs

$$c_{i+1} = \gamma_i + c_i$$

$$\sigma_i = \overline{\gamma_i \oplus c_i},$$

respectively. The first carry input c_0 equals zero. In order to determine the desired sum $\hat{\gamma}$, three cases have to be considered:

1. If $\gamma = 0$, then let $\hat{\gamma}_m = \overline{c_m} = 1$ and $\hat{\gamma}^{(m-1)} = \sigma^{(m-1)} - (2^m - 1) = 0$, i.e. $\hat{\gamma}_i = 0$ for $0 \leq i \leq m-1$.
2. If $1 \leq \gamma \leq 2^m - 1$, then let $\hat{\gamma}_m = \overline{c_m} = 0$ and $\hat{\gamma}^{(m-1)} = \sigma^{(m-1)}$.
3. If $\gamma = 2^m$, then let $\hat{\gamma}_m = c_m = 0$ and $\hat{\gamma}^{(m-1)} = \sigma^{(m-1)}$.

From these three cases we form the two Karnaugh maps in Figure 6.1 for the bit values of $\hat{\gamma}$. According to the maps, for $0 \leq i \leq m-1$ the Boolean functions for $\hat{\gamma}_m$ and $\hat{\gamma}_i$ can be expressed as

$$\hat{\gamma}_m = \overline{\gamma_m + c_m}$$

$$\hat{\gamma}_i = \gamma_m + c_m \sigma_i,$$

γ_m \ c_m	0	1
0	1	0
1	0	X

(a) $\hat{\gamma}_m$

σ_i \ $\gamma_m c_m$	00	01	11	10
0	X	0	X	X
1	0	1	X	1

(b) $\hat{\gamma}_i$ for $0 \leq i \leq m-1$

Figure 6.1: *Karnaugh maps for the bit values $\hat{\gamma}_i$ of $\hat{\gamma}$.* X = *"don't care".*
(a) $\hat{\gamma}_m = \overline{\gamma_m} \cdot \overline{c_m} = \overline{\gamma_m + c_m}$. (b) $\hat{\gamma}_i = \gamma_m + c_m \sigma_i$.

respectively. The sum σ may be computed using either a carry ripple or a carry look-ahead type of architecture. In the previous chapter we concluded that, due to its favourable AT^2 performance, the carry look-ahead-type architecture in Figure 5.2 is preferable to the carry ripple-type architecture in Figure 5.1. We have designed a carry look-ahead type of architecture for the computation of $T(\gamma)$ (i.e. $\hat{\gamma}$) from γ. The architecture, which is shown in Figure 6.2, is similar to the architecture in Figure 5.2. The row of combined AND–NOR gates at the output generates the one's complement $\overline{\hat{\gamma}^{(m-1)}}$ of the NBC integer $\hat{\gamma}^{(m-1)}$, i.e. the gate in bit position i generates the signal $\overline{\hat{\gamma}_i} = \overline{\gamma_m + c_m \sigma_i}$. A schematic description of such a gate is shown in the bottom of Figure 6.2. The gate has size $\mathcal{C}_{\text{AND-NOR}} = 6$, fan-in $f_{\text{AND-NOR}} = 2$, and output normalised resistance $r_{\text{AND-NOR}} = 2$. It has no internal stage. The output array of inverters generates the desired output $\hat{\gamma}^{(m-1)}$.

The total size of the 'NBC-to-diminished–1' architecture in Figure 6.2 equals[4]

$$\begin{aligned}
\mathcal{C}_{\text{NBC2Dim}} &= (m(\log_2 m - 1) + 1)\mathcal{C}_{\text{NAND/NOR}} + (m-1)(\mathcal{C}_{\text{inv}} + \mathcal{C}_{\text{XNOR}}) \\
&\quad + \mathcal{C}_{\text{NOR}} + m(\mathcal{C}_{\text{AND-NOR}} + \mathcal{C}_{\text{inv}}) \\
&= (m(\log_2 m - 1) + 1) \cdot 4 + (m-1)(2+12) + 4 + m(6+2) \\
&= 4m \cdot \log_2 m + 8m - 6.
\end{aligned}$$

The CP is the dotted path from the γ_1-input node through the circuit and to the $\hat{\gamma}_0$-output node. The fan-in and the output normalised resistance of the architecture, with respect to this CP, equal

$$\begin{aligned}
f_{\text{NBC2Dim}} &= f_{\text{mod},2} = 8 \\
r_{\text{NBC2Dim}} &= r_{\text{AND-NOR}} = 1,
\end{aligned}$$

[4] Compare the derivation of this expression with the derivation of $\mathcal{C}_{\text{mod},2}$ in (5.4).

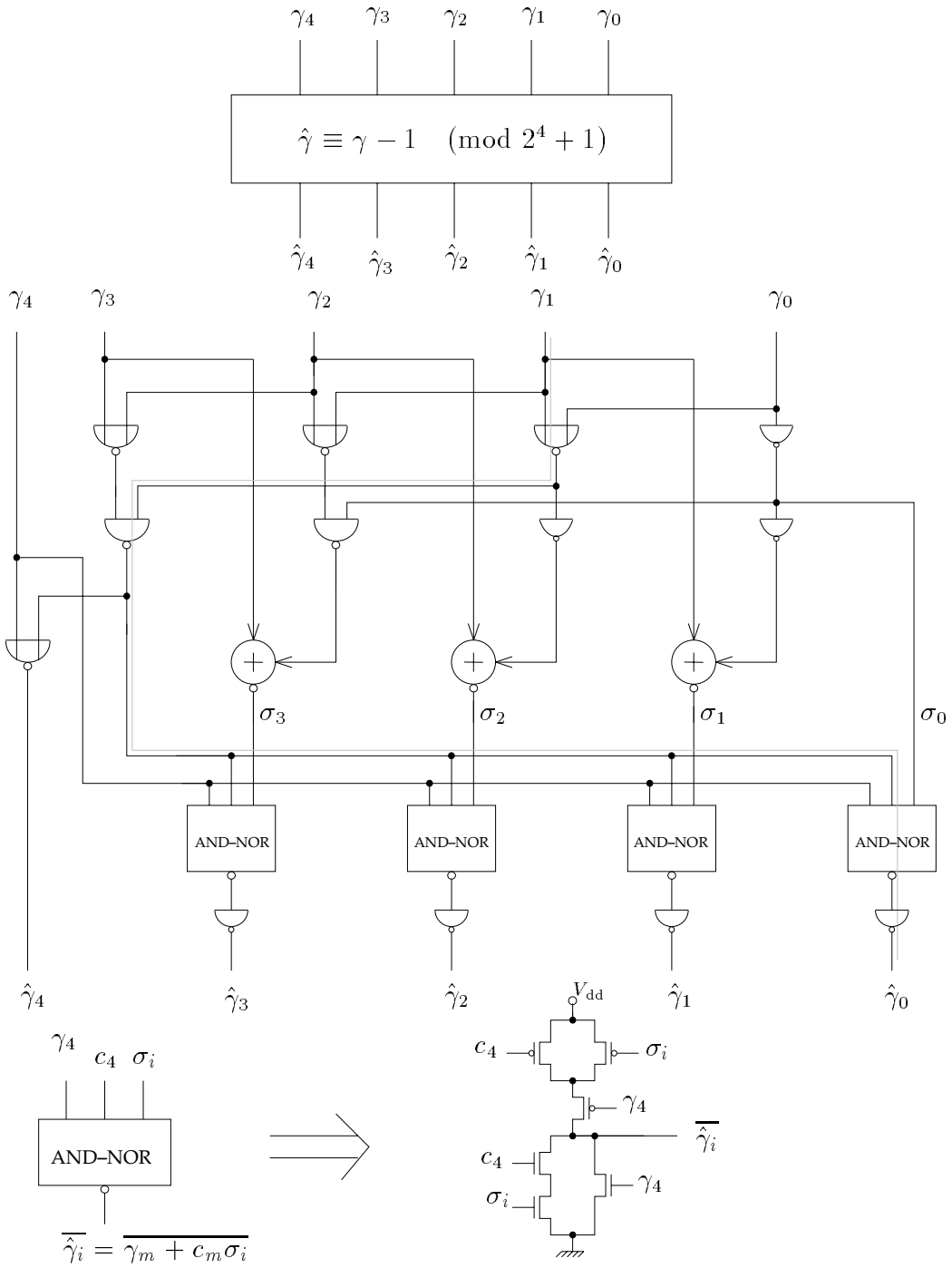


Figure 6.2: *An architecture performing the code translation from the NBC to the diminished–1 representation (from γ to $T(\gamma) = \hat{\gamma}$). The dotted line indicates the CP through the circuit. The bottom part of the figure shows how each combined AND–NOR gate is designed.*

respectively, and the internal CP length equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,NBC2Dim}} &= (\log_2\, m-1)r_{\mathrm{NAND/NOR}}(f_{\mathrm{NOR/NAND}}+f_{\mathrm{inv}})\\ &\quad + r_{\mathrm{NAND}}(f_{\mathrm{NOR}}+mf_{\mathrm{AND-NOR}})+r_{\mathrm{AND-NOR}}f_{\mathrm{inv}}\\ &= (\log_2\, m-1)\cdot 2(2+2)+2(2+2m)+2\cdot 2\\ &= 4m+8\log_2\, m.\end{aligned}$$

As in Chapter 5, when determining the AT^2 performance of the architecture, we assume that it is both preceded and followed by $(m+1)$-bit parallel registers. Therefore, the time T required to evaluate the congruence $\hat{\gamma} \equiv \gamma - 1 \pmod{2^m+1}$ is proportional to

$$\begin{aligned}\mathcal{L}_{\mathrm{NBC2Dim}} &= \mathcal{L}_{\mathrm{reg}}+r_{\mathrm{reg}}f_{\mathrm{NBC2Dim}}+\mathcal{L}_{\mathrm{CP,NBC2Dim}}+r_{\mathrm{NBC2Dim}}f_{\mathrm{reg}}\\ &= 22+2\cdot 8+4m+8\log_2\, m+2\\ &= 4m+8\log_2\, m+40,\end{aligned}$$

which implies that the AT^2 performance of the circuit is proportional to the product

$$\begin{aligned}\mathcal{CL}^2_{\mathrm{NBC2Dim}} &\triangleq \mathcal{C}_{\mathrm{NBC2Dim}}(\mathcal{L}_{\mathrm{NBC2Dim}})^2\\ &= (4m\cdot\log_2\, m+8m-6)(4m+8\log_2\, m+40)^2=\mathcal{O}\left(m^3\log_2\, m\right).\end{aligned}$$

This product is less than the area-time product $\mathcal{CL}^2_{\mathrm{mod},2}$ of the modulus reduction circuit in Figure 5.2.

An alternative procedure for performing the subtraction $\hat{\gamma} = \gamma - 1 \pmod{2^m+1}$, for $\gamma = (\gamma_m,\, \gamma_{m-1},\ldots,\, \gamma_1,\, \gamma_0)_2 = \gamma_m 2^m + \gamma^{(m-1)}$, is the following:

1. If $0 \le \gamma \le 2^m-1$, i.e. if $\gamma_m = 0$ and $0 \le \gamma^{(m-1)} \le 2^{m-1}-1$, then let $\beta \triangleq \gamma^{(m-1)} - 1 \equiv 2^m + \gamma^{(m-1)} = (1,\, \gamma_{m-1},\ldots,\, \gamma_1,\, \gamma_0)_2 \pmod{2^m+1}$. The diminished–1 integer $\hat{\gamma}$ is obtained by reducing β modulo 2^m+1.

2. If $\gamma = 2^m$, i.e. if $\gamma_m = 1$ and $\gamma^{(m-1)} = 0$, then let $\beta \triangleq \gamma - 1 = 2^m - 1 = (0,\, 1,\ldots,\, 1,\, 1)_2$. The diminished–1 integer $\hat{\gamma}$ equals β.

In case 1, β can be obtained from γ simply by inverting its most significant bit γ_m. By letting β be the input of a modulus reduction circuit, for example the one in Figure 5.1 or the one in Figure 5.2, we get the desired integer $\hat{\gamma}$ as the output of the circuit. In case 2, β can be obtained by inverting *all* bits of γ. Even though this β is the desired $\hat{\gamma}$, we avoid an unnecessarily complex control logic

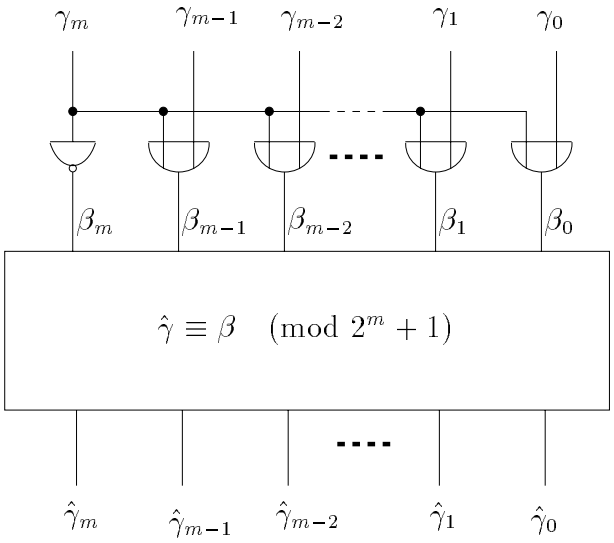


Figure 6.3: *An alternative architecture for performing the code translation from the NBC integer γ to the diminished–1 coded integer $T(\gamma) = \hat{\gamma} \equiv \gamma - 1 \pmod{2^m + 1}$ using a modulus reduction circuit.*

by letting β pass through the modulus reduction circuit also in case 2. Still, it is the procedure in case 1 that determines the overall time performance of the operation. Figure 6.3 shows an architecture that generates the binary coded diminished–1 integer $\hat{\gamma}$ from the NBC integer γ using the above procedures in cases 1 and 2.

It may be convenient to utilise a modulus reduction circuit to perform the code translation but, however, both the size and the total CP length of such an architecture are greater than the corresponding parameters of an architecture that is specially designed for the operation. For example, if the modulus reduction part of the circuit in Figure 6.3 is the carry look-ahead-type architecture in Figure 5.2, its total size equals $\mathcal{C}_{\mathrm{mod},2} + m\mathcal{C}_{\mathrm{OR}} + \mathcal{C}_{\mathrm{inv}} = 4m \log_2 m + 20m + 2$ and its total CP length (including the delay contribution of one input and one output register) equals $\mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}}(m f_{\mathrm{OR}} + f_{\mathrm{inv}}) + \mathcal{L}_{\mathrm{OR}} + r_{\mathrm{OR}} f_{\mathrm{mod},2} + \mathcal{L}_{\mathrm{CP,mod},2} + r_{\mathrm{mod},2} f_{\mathrm{reg}} = 10m + 8 \log_2 m + 46$. These two complexity parameters should be compared with the smaller size $\mathcal{C}_{\mathrm{NBC2Dim}}$ and the smaller CP length $\mathcal{L}_{\mathrm{NBC2Dim}}$, respectively, of the architecture in Figure 6.2.

In his paper of 1976 Leibowitz [58] suggests that the code translation should be carried out as an ordinary diminished–1 addition (see Section 6.3.4) of γ and

the NBC integer $2^m - 1$.[5] This is a good solution if the diminished–1 adder is readily available and if the time requirements for the code translation are fulfilled. However, at least from a time performance point of view, a special-purpose architecture like the one in Figure 6.2 is preferable to a general-purpose architecture (like the diminished–1 adder).

Diminished–1 to NBC Representation

Leibowitz [58] described how to perform the translation from a binary coded diminished–1 integer $T(\gamma)$ $(= \hat{\gamma})$ to an NBC integer γ by adding $\overline{\hat{\gamma}_m}$ to $\hat{\gamma}^{(m-1)}$. Thus, we have

$$\gamma \equiv \hat{\gamma} + 1 \equiv \hat{\gamma}^{(m-1)} + 1 - \hat{\gamma}_m = \hat{\gamma}^{(m-1)} + \overline{\hat{\gamma}_m} \pmod{2^m + 1}.$$

This operation, which in hardware does not require any modulus reduction, can be performed using a row of half adder elements. Consider an m-bit parallel carry ripple adder with input signals β and γ. In bit position i, where $0 \leq i \leq m-1$, the signal input bits are $\beta_i = 0$ and $\hat{\gamma}_i \in \mathbb{Z}_2$, which implies that, by (4.2) and (4.3), the carry and sum outputs are equal to

$$\begin{aligned} c_{i+1} &= c_i \hat{\gamma}_i \\ \gamma_i &= c_i \oplus \hat{\gamma}_i, \end{aligned}$$

respectively. Because these functions are also the respective carry and sum outputs of the half adder element (see the end of Section 4.3.4), the parallel adder may be formed by a row of half adder elements, where the first carry input c_0 equals $\overline{\hat{\gamma}_m}$. An architecture that performs the diminished–1-to-NBC coordinate transformation using the above procedure is shown in Figure 6.4. The size of this architecture equals

$$\begin{aligned} \mathcal{C}_{\text{Dim2NBC}} &= m\mathcal{C}_{\text{HA}} + \mathcal{C}_{\text{inv}} \\ &= 18m + 2. \end{aligned}$$

The CP runs from the $\hat{\gamma}_m$-input node through the inverter and the chain of cascaded half adder elements. Denote by n_s and n_c the fan-out with respect to the γ_{m-1}-output node and the γ_m-output node, respectively. If $\mathcal{L}_{\text{HA,sum}} + r_{\text{HA,sum}} n_s = 2 + 2n_s \geq \mathcal{L}_{\text{HA,carry}} + r_{\text{HA,carry}} n_c = 4 + n_c$, i.e if $n_s \geq n_c/2 + 1$, the CP runs from the input to the *sum* output γ_{m-1} of the half adder element in the most significant bit position. Otherwise, the CP runs to the *carry* output γ_m. The former path is the one most likely to belong to the CP.[6] Therefore, the fan-in, the output

[5] We have $\varphi \equiv \gamma - 1 \equiv \gamma + 2^m - 1 + 1 \equiv \gamma \oplus (2^m - 1) \pmod{2^m + 1}$.

[6] For example, if $n_s = n_c$, the CP runs from the input of the half adder element to its *carry* output only if $n_c = 1$.

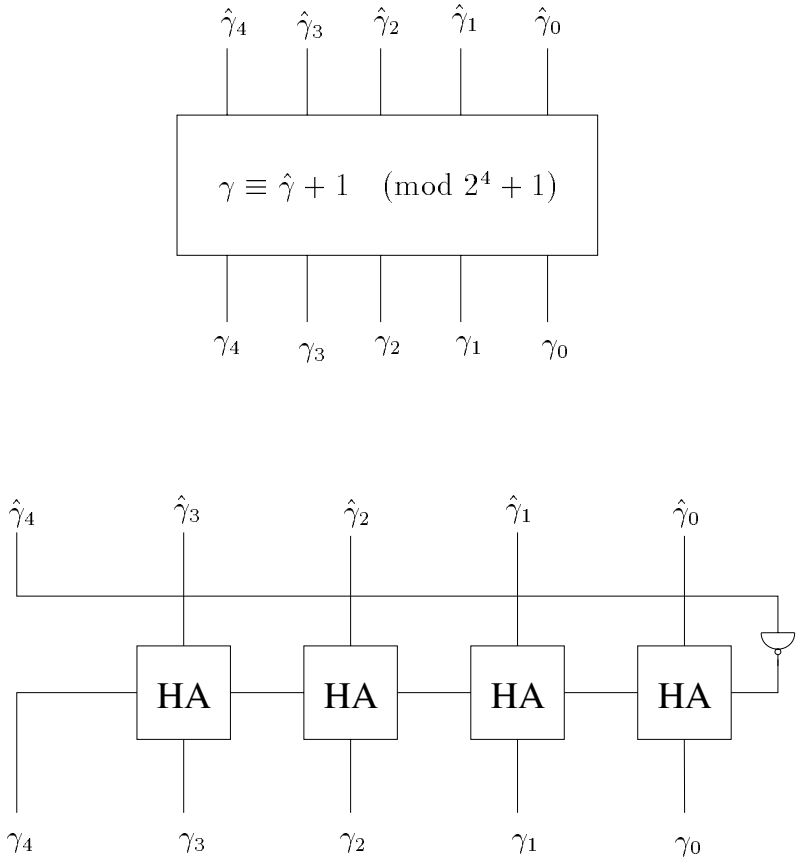


Figure 6.4: *A simple architecture for performing the code translation from the binary coded diminished–1 integer $T(\gamma) = \hat{\gamma}$ to the NBC integer $\gamma \equiv \hat{\gamma} + 1 \pmod{2^4 + 1}$.*

normalised resistance, and the internal CP length of the circuit equal

$$\begin{aligned} f_{\text{Dim2NBC}} &= f_{\text{inv}} = 2, \\ r_{\text{Dim2NBC}} &= r_{\text{HA,sum}} = 2, \\ \mathcal{L}_{\text{CP,Dim2NBC}} &= r_{\text{inv}} f_{\text{HA}} + (m-1)(\mathcal{L}_{\text{HA,carry}} + r_{\text{HA,carry}} f_{\text{HA}}) + \mathcal{L}_{\text{HA,sum}} \\ &= 6 + (m-1)(4+6) + 2 \\ &= 10m - 2, \end{aligned}$$

respectively. When the input and the output of the coordinate transformation circuit in Figure 6.4 are each connected to an $(m+1)$-bit register, the time T to perform its operation is proportional to the total CP length

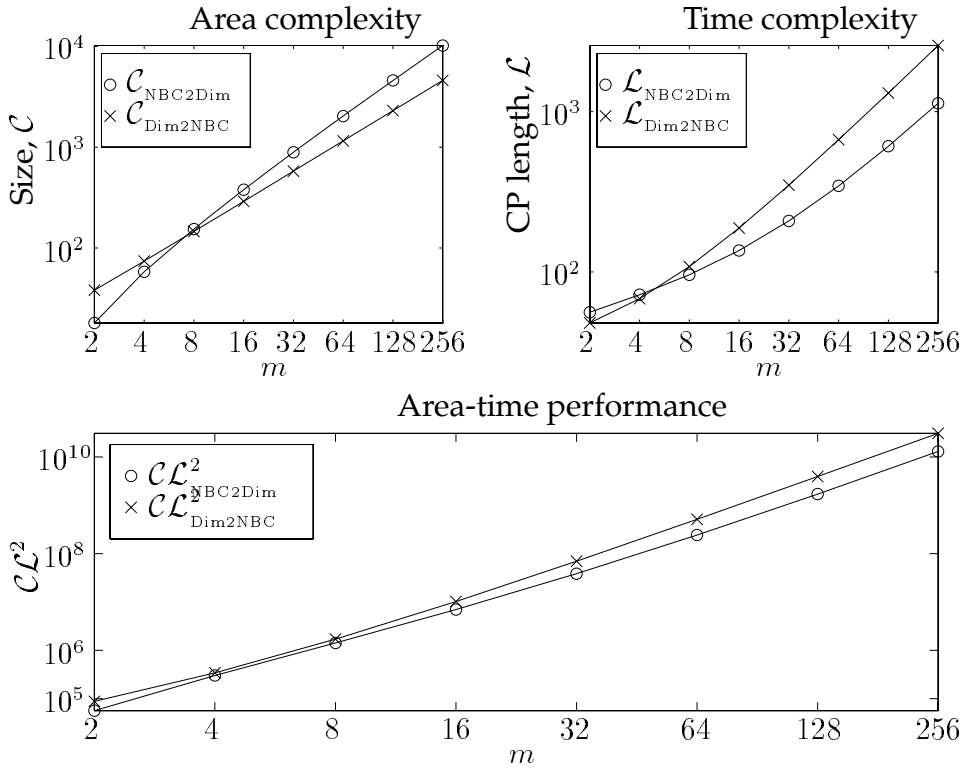


Figure 6.5: *The sizes, CP lengths, and AT^2 performances of the code translation architectures. The parameters are plotted versus m for $m = 2, 4, 8, 16, 32, 64, 128, 256$.*

$$\begin{aligned}\mathcal{L}_{\mathrm{Dim2NBC}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} f_{\mathrm{Dim2NBC}} + \mathcal{L}_{\mathrm{CP,Dim2NBC}} + r_{\mathrm{Dim2NBC}} f_{\mathrm{reg}} \\ &= 22 + 2 \cdot 2 + 10m - 2 + 2 \cdot 2 \\ &= 10m + 28.\end{aligned}$$

Hence, the area-time performance AT^2 of this circuit is proportional to

$$\begin{aligned}\mathcal{C}\mathcal{L}^2_{\mathrm{Dim2NBC}} &\triangleq \mathcal{C}_{\mathrm{Dim2NBC}} (\mathcal{L}_{\mathrm{Dim2NBC}})^2 = (18m + 2)(10m + 28)^2 \\ &= \mathcal{O}\left(m^3\right).\end{aligned}$$

The row of half adder elements in Figure 6.4 is a carry ripple type of architecture. Other classes of architectures, like the parallel carry look-ahead (half) adder, are not considered here.

In Figure 6.5 we have plotted the sizes, total CP lengths, and area-time performances of the architectures in Figure 6.2 and Figure 6.4. When comparing

these parameters we see that

$$\begin{array}{ll} \mathcal{C}_{\text{NBC2Dim}} < \mathcal{C}_{\text{Dim2NBC}}; & \text{for } m = 2 \text{ and } m = 4. \\ \mathcal{C}_{\text{NBC2Dim}} > \mathcal{C}_{\text{Dim2NBC}}; & \text{for } m \geq 8. \end{array}$$

$$\begin{array}{ll} \mathcal{L}_{\text{NBC2Dim}} > \mathcal{L}_{\text{Dim2NBC}}; & \text{for } m = 2 \text{ and } m = 4. \\ \mathcal{L}_{\text{NBC2Dim}} < \mathcal{L}_{\text{Dim2NBC}}; & \text{for } m \geq 8. \end{array}$$

$$\mathcal{CL}^2_{\text{NBC2Dim}} < \mathcal{CL}^2_{\text{Dim2NBC}}; \qquad \text{for all } m.$$

6.3.2 Modulus Reduction

Because the diminished–1 integers are represented by the NBC integers in $\mathbb{Z}_{2^m+1}$, the residue modulo $2^m + 1$ of an $(m+1)$-bit binary coded diminished–1 integer can be computed using any of the modulus reduction circuits in Section 5.1.1. However, one of the nice properties of the diminished–1 representation is that it yields arithmetic operations for which the modulus reductions *together* with the arithmetic operations can be carried out in a more straightforward way than what is possible when using the ordinary NBC representation. This is demonstrated in the following sections.

6.3.3 Negation

By letting $l = -1$ in (6.2) we get

$$T(-\gamma) \equiv \overline{T(\gamma)} + 1 \pmod{2^m + 1}. \tag{6.21}$$

This congruence was also considered in (6.17). The computational complexity of computing $\overline{T(\gamma)} + 1 \bmod 2^m + 1$ may seem to be in the same order as the complexity of performing the code translation from the diminished–1 integer $T(\gamma) \in \mathbb{Z}_{2^m+1}$ to the NBC integer $\gamma \equiv T(\gamma) + 1 \pmod{2^m + 1}$. However, by expanding (6.21) as

$$\begin{aligned} T(-\gamma) &\equiv (1 - \hat{\gamma}_m)2^m + \overline{\hat{\gamma}^{(m-1)}} + 1 \\ &\equiv \overline{\hat{\gamma}^{(m-1)}} + \hat{\gamma}_m = \begin{cases} \overline{\hat{\gamma}^{(m-1)}} & \text{if } \hat{\gamma}_m = 0 \\ \hat{\gamma} = 2^m & \text{if } \hat{\gamma}_m = 1 \end{cases} \pmod{2^m + 1}, \end{aligned}$$

where $\hat{\gamma} = T(\gamma)$, it shows to be quite easy to implement. The negative of a nonzero integer γ (for which $\hat{\gamma}_m = 0$) is simply derived by inverting its m least

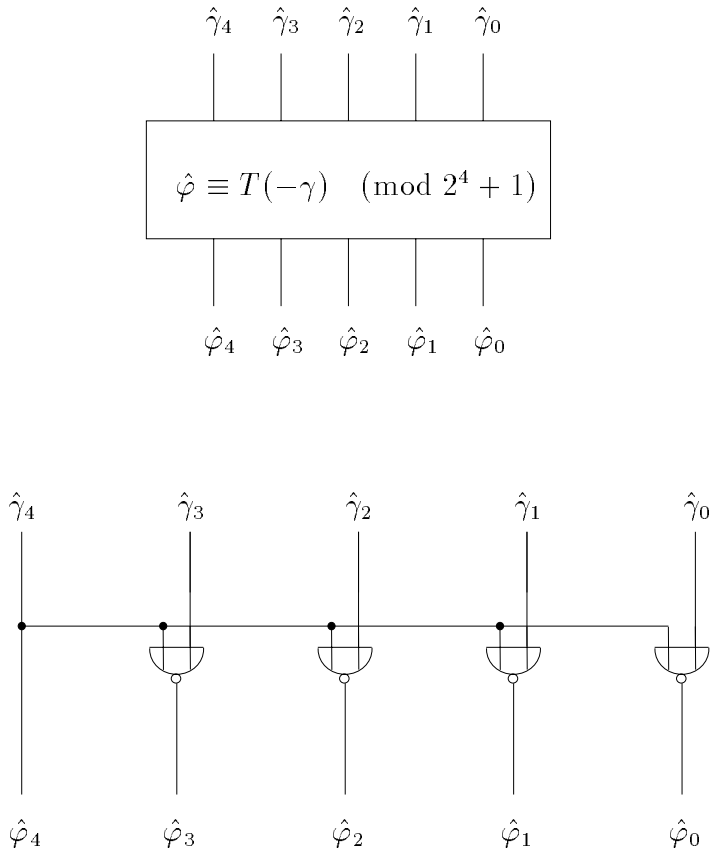


Figure 6.6: *Negation modulo* $2^4 + 1$. *Here, we have* $\hat{\gamma} = T(\gamma)$ *and* $T(-\gamma) = \hat{\varphi}$.

significant bits. For the zero element we have the relation $T(-0) = T(0) = 2^m$, which means that the symbol $T(0)$ stays unmodified. For the $(m+1)$-bit binary integer $T(\gamma) = (\hat{\gamma}_m, \hat{\gamma}_{m-1}, \ldots, \hat{\gamma}_0)_2$ we thus have

$$T(-\gamma) \equiv \hat{\varphi} = (\hat{\varphi}_m, \hat{\varphi}_{m-1}, \ldots, \hat{\varphi}_0)_2,$$

where the Boolean function for $\hat{\varphi}_i$ equals

$$\hat{\varphi}_i = \begin{cases} \overline{\hat{\gamma}_i} \cdot \overline{\hat{\gamma}_m} = \overline{\hat{\gamma}_i + \hat{\gamma}_m}; & \text{for } i = 0, 1, \ldots, m-1 \\ \hat{\gamma}_m; & \text{for } i = m \end{cases} .$$

Figure 6.6 shows an architecture that realises such a negation, using a row of NOR gates. This simple circuit is generally used for negation of diminished–1 numbers, see for example Pajayakrit [71, Ch. 3.4], Benaissa et al. [11, Fig. 8], and Sunder et al. [97, Fig. 3].

The CP through the negater circuit is the path from the $\hat{\gamma}_m$-input node through one of the NOR gates to the circuit output. The fan-in and the output normalised resistance, with respect to this path, equal

$$\begin{aligned} f_{\text{dimneg}} &= n_m + m f_{\text{NOR}} = n_m + 2m \\ r_{\text{dimneg}} &= r_{\text{NOR}} = 2, \end{aligned}$$

respectively, where n_m is the negater fan-out with respect to the $\hat{\varphi}_m$ node. Note that the delay of the input stage will be excessively long if both m and the normalised resistance of the input stage are large. One way of reducing this delay is to properly buffer the circuit input stage, i.e. by using drivers in the stage. However, as mentioned before, such a buffering is not considered here. Therefore, the chip area A occupied by the negater circuit in Figure 6.6 is proportional to its size

$$\mathcal{C}_{\text{dimneg}} = m\mathcal{C}_{\text{NOR}} = 4m.$$

There is no internal CP of the architecture. When the negater input is taken from an $(m+1)$-bit parallel register and the output is stored in a similar register, we get $n_m = f_{\text{reg}} = 2$. Then, the negation time T is proportional to the total CP length

$$\mathcal{L}_{\text{dimneg}} = \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{dimneg}} + r_{\text{dimneg}} f_{\text{reg}} = 4m + 30.$$

Hence, the product AT^2 is proportional to

$$\mathcal{C}\mathcal{L}^2_{\text{dimneg}} \triangleq \mathcal{C}_{\text{dimneg}} (\mathcal{L}_{\text{dimneg}})^2 = \mathcal{O}\left(m^3\right).$$

6.3.4 Addition and Subtraction

In Section 6.2 (Equations (6.3) and (6.15)) we showed that the choice $l = -1$ in (6.1) yields

$$T(\beta + \gamma) \equiv T(\beta) + T(\gamma) + 1 \pmod{2^m + 1} \tag{6.22}$$

For $0 \le T(\beta), T(\gamma) \le 2^m$, we expand this equation as

$$\begin{aligned} \hat{\varphi} &\triangleq T(\beta+\gamma) \equiv \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} - \hat{\beta}_m - \hat{\gamma}_m + 1 \\ &= \begin{cases} \hat{\lambda} + 1 \pmod{2^m+1}; & \text{if } (\hat{\beta}_m, \hat{\gamma}_m) = (0,0) \\ \hat{\lambda} \pmod{2^m+1}; & \text{if } (\hat{\beta}_m, \hat{\gamma}_m) \in \{(0,1),\ (1,0)\} \\ \hat{\lambda} - 1 = 2^m \pmod{2^m+1}; & \text{if } (\hat{\beta}_m, \hat{\gamma}_m) = (1,1) \end{cases}, \end{aligned}$$

where $\hat{\beta} = T(\beta)$, $\hat{\gamma} = T(\gamma)$, and $\hat{\lambda} \triangleq \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)}$. The three cases in the above equation are handled in the following way:

1. $\hat{\varphi} \equiv \hat{\lambda} + 1 \pmod{2^m + 1}$:
   Using the congruence $\hat{\lambda} = \hat{\lambda}_m 2^m + \hat{\lambda}^{(m-1)} \equiv \hat{\lambda}^{(m-1)} - \hat{\lambda}_m \pmod{2^m + 1}$ we get
   $\hat{\varphi} \equiv \hat{\lambda}^{(m-1)} + 1 - \hat{\lambda}_m = \hat{\lambda}^{(m-1)} + \overline{\hat{\lambda}_m} \pmod{2^m + 1}$

2. $\hat{\varphi} \equiv \hat{\lambda} \pmod{2^m + 1}$:
   In this case we have either $\hat{\beta}^{(m-1)} = 0$ or $\hat{\gamma}^{(m-1)} = 0$, which implies $\hat{\lambda}_m = 0$. Therefore, no modulus reduction is needed. We let
   $\hat{\varphi} = (\hat{\varphi}_m, \hat{\varphi}^{(m-1)}) = (0, \hat{\lambda}^{(m-1)})$.

3. $\hat{\varphi} \equiv \hat{\lambda} - 1 \equiv 2^m \pmod{2^m + 1}$:
   In this case we have $\hat{\beta}^{(m-1)} = \hat{\gamma}^{(m-1)} = 0$, which gives $\hat{\lambda} = 0$. Therefore, let
   $\hat{\varphi} = (\hat{\varphi}_m, \hat{\varphi}^{(m-1)}) = (1, \hat{\lambda}^{(m-1)})$.

A Carry Look-Ahead Adder

For a bit-parallel transmission of both $\hat{\beta}$ and $\hat{\gamma}$, the sum $\hat{\lambda} = \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)}$ may be calculated using an m-bit parallel adder. In the above case 1, the supplementary addition by $\overline{\hat{\lambda}_m}$ may be carried out by letting the carry in c_0 in the least significant bit position be equal to $\overline{\hat{\lambda}_m}$. The bit value $\hat{\lambda}_m$ must then be generated by a carry look-ahead circuit. In cases 2 and 3, the initial carry in c_0 equals zero. Hence, c_0 can be generated according to the Boolean function

$$c_0 = \overline{\hat{\beta}_m} \cdot \overline{\hat{\gamma}_m} \cdot \overline{\hat{\lambda}_m} = \overline{(\hat{\beta}_m + \hat{\gamma}_m) + \hat{\lambda}_m}. \tag{6.23}$$

The most significant bit $\hat{\varphi}_m$ of the output $\hat{\varphi}$ must be generated separately. Let c_m denote the final carry of the addition $\hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} + c_0$. Table 6.1 shows the possible states of $\hat{\beta}_m,\ \hat{\gamma}_m,\ \hat{\lambda}_m,\ c_0,\ c_m$, and the resulting sum $\hat{\varphi}$. We see that $\hat{\varphi}_m$ is high only for $(\hat{\beta}_m,\ \hat{\gamma}_m) = (1,\ 1)$ and for $(c_0,\ c_m) = (1,\ 1)$, which means that it can be described as the Boolean function

$$\hat{\varphi}_m = \hat{\beta}_m \cdot \hat{\gamma}_m + c_0 \cdot c_m = \overline{\overline{\hat{\beta}_m \cdot \hat{\gamma}_m} \cdot \overline{c_0 \cdot c_m}}.$$

Case	$\hat{\beta}_m$	$\hat{\gamma}_m$	$\hat{\lambda}_m$	c_0	c_m	$\hat{\varphi}$	$\hat{\varphi}_m$
1	0	0	0	1	0	$\geq 0\,,\ \leq 2^m - 1$	0
			0	1	1	2^m	1
			1	0	1	$\geq 0\,,\ \leq 2^m - 1$	0
2	0/1	1/0	0	0	0	$\geq 0\,,\ \leq 2^m - 1$	0
3	1	1	0	0	0	2^m	1

Table 6.1: *The possible states of some variables involved in the computation of* $\hat{\varphi} = T(\beta + \gamma)$ *(see also Figure 6.7).*

Figure 6.7 shows one possible architecture of a diminished–1 carry look-ahead adder. The carry-in of the carry look-ahead block equals zero. The adder has about the same structure as McClellan's adder [65, Fig. 7]. However, because of an incorrect gating of the carry c_m, McClellan's adder gives an erroneous output when $\hat{\lambda}_m$ equals one (the third line of Table 6.1). On the other hand, the gating is correctly realised for the carry look-ahead adder in Figure 8 of [65].

Benaissa et al. [11] and others also use an adder that is based on the adder in Figure 6.7. However, in Figure 9 of [11], the authors use AND and OR gates to form the output bit $\hat{\varphi}_m$, in contrast to the NAND gates used in Figure 6.7. Pajayakrit [71, Fig. 3.3] also considers an adder whose architecture slightly differs from the one presented in [11]. In Pajayakrit's adder, there is an AND gate that has $\hat{\lambda}_m$ (which by Pajayakrit is named D) as one of its input signals. This signal is exchanged for c_0 in Benaissa's adder. Using the RC model adopted in this thesis, it can easily be verified that when c_0 is chosen as input signal to the AND gate, the internal delay from the $\hat{\lambda}_m$-output node of the carry look-ahead circuit to the c_0 carry input node of the parallel adder is less than the corresponding delay if $\hat{\lambda}_m$ is chosen as the input signal of the AND gate.

Remark: Pajayakrit's adder is actually a corrected version of the adder considered by Towers et al. [101, Fig. 9]. In Towers' adder, which is based on McClellan's adder, the carry-in signal c_0 was improperly formed as the Boolean function $c_0 = \hat{\beta}_m \hat{\gamma}_m \hat{\lambda}_m$ instead of the correct one given by (6.23).

When comparing the adder architectures described above we find that the adder in Figure 6.7 is preferable to the others, with respect to correctness and both area and time complexity. So far, we have not considered the choice of

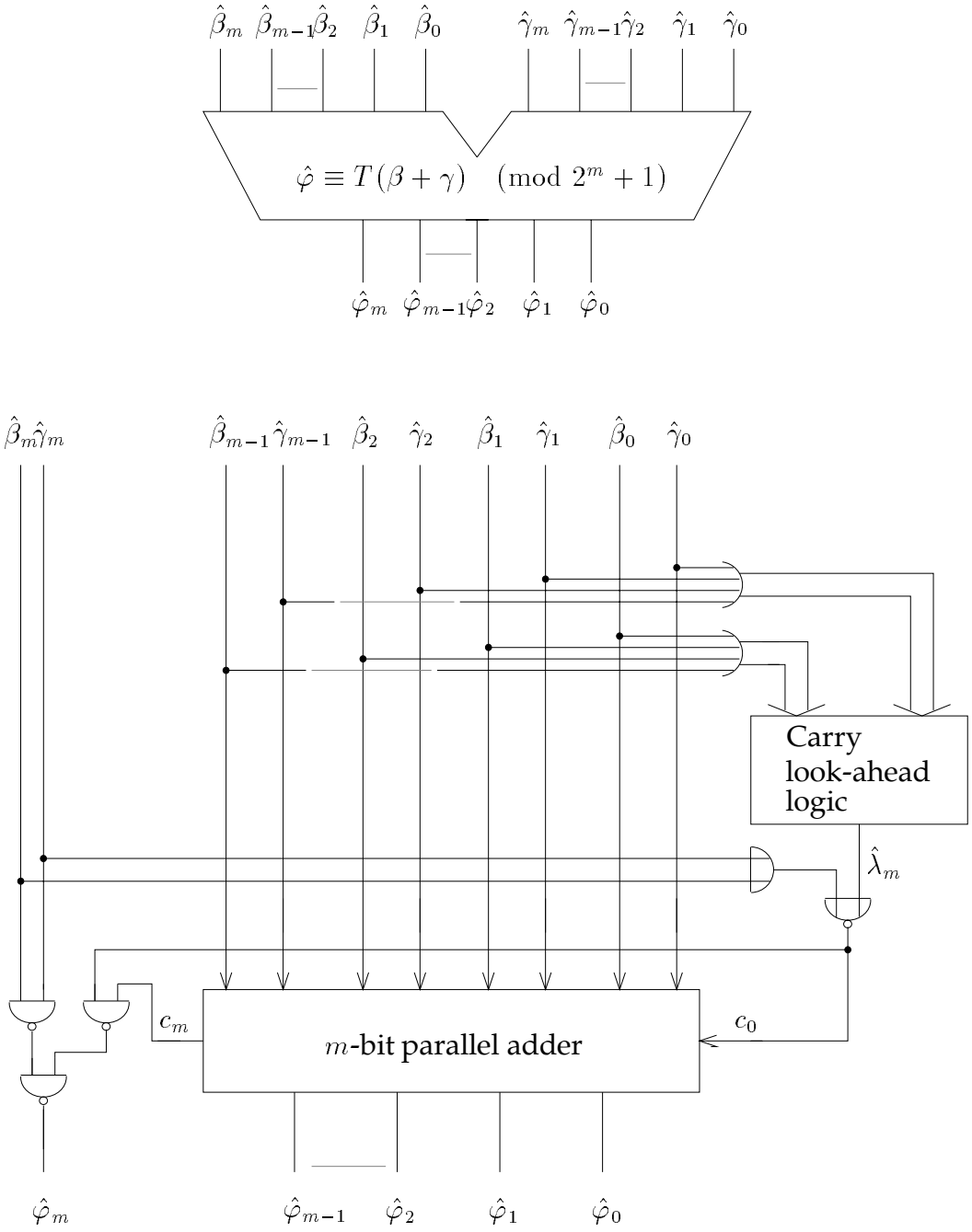


Figure 6.7: *Diminished–1 addition modulo* $2^m + 1$*:* $\hat{\beta} = T(\beta)$ *and* $\hat{\gamma} = T(\gamma)$.

adder *type* for the m-bit adder in Figure 6.7 (the one whose inputs are $\hat{\beta}^{(m-1)}$, $\hat{\gamma}^{(m-1)}$, and c_0). If this adder is implemented as a carry look-ahead type of adder, like for example McClellan's adder [65, Fig. 8], we presumably obtain a faster diminished–1 adder than if it is implemented as a carry ripple type of adder. However, the chip area occupied by an adder is generally larger for the carry look-ahead adder than for the carry ripple adder.

In order to get fair comparisons between the bit-parallel carry ripple-type NBC adder in Figure 5.7 and the bit-parallel adders in this section, the parallel diminished–1 adders considered here are all plain carry ripple-type adders. As mentioned in the beginning of Section 5.1, we primarily consider architectures that can be mutually compared in order to decide which form of *element representation* is most advantageous, with respect to chip area, computation time, and area-time performance. Therefore, we generally compare architectures of the same type, but with respect to different element representations, rather than try to find the most area-time efficient architecture for a certain element representation.

Hence, the bit-parallel m-bit carry ripple adder in Figure 6.7 simply consists of a row of m cascaded full adder elements. Consequently, the total size of the entire diminished–1 adder equals

$$\mathcal{C}_{\text{dimadd,1}} = m\mathcal{C}_{\text{FA}} + \mathcal{C}_{\text{CLA}} + 4\mathcal{C}_{\text{NAND/NOR}} + \mathcal{C}_{\text{OR}} = \mathcal{C}_{\text{CLA}} + 28m + 22, \tag{6.24}$$

where the complexity $\mathcal{C}_{\text{CLA}}$ of the carry look-ahead logic depends on how it is implemented. It is well known that the output carry c'_{i+1} from a full adder element in bit position i, whose input signals are $\hat{\beta}_i$, $\hat{\gamma}_i$, and c'_i, may be expressed as the Boolean function[7]

$$c'_{i+1} = g_i + p_i c'_i, \tag{6.25}$$

where $g_i = \hat{\beta}_i\hat{\gamma}_i$ and $p_i = \hat{\beta}_i + \hat{\gamma}_i$ are called the carry *generate* and *propagate* functions, respectively. For the diminished–1 adder we have $\hat{\lambda}_m = c'_m$. Therefore, by expanding (6.25) for $i = m - 1$ we get

$$\begin{aligned} \hat{\lambda}_m = c'_m \;\; = \;\; & g_{m-1} + p_{m-1}g_{m-2} + p_{m-1}p_{m-2}g_{m-3} + \cdots \\ & + p_{m-1}p_{m-2}\cdots p_1 g_0 + p_{m-1}p_{m-2}\cdots p_0 c'_0. \end{aligned} \tag{6.26}$$

However, the addend $p_{m-1}p_{m-2}\cdots p_0 c'_0$ of (6.26) can be excluded here, because for the circuit in Figure 6.7 we have $c'_0 = 0$. The resulting Boolean function can efficiently be evaluated using the carry look-ahead tree in Figure 6.8. This architecture for generating only one carry signal is also suggested by Yuan

[7] For the diminished–1 adder in Figure 6.7, we denote by c'_i the input carry signal in bit position i of the carry look-ahead circuit. We do this in order to distinguish it from the corresponding carry signal c_i of the parallel adder in the bottom part of the figure.

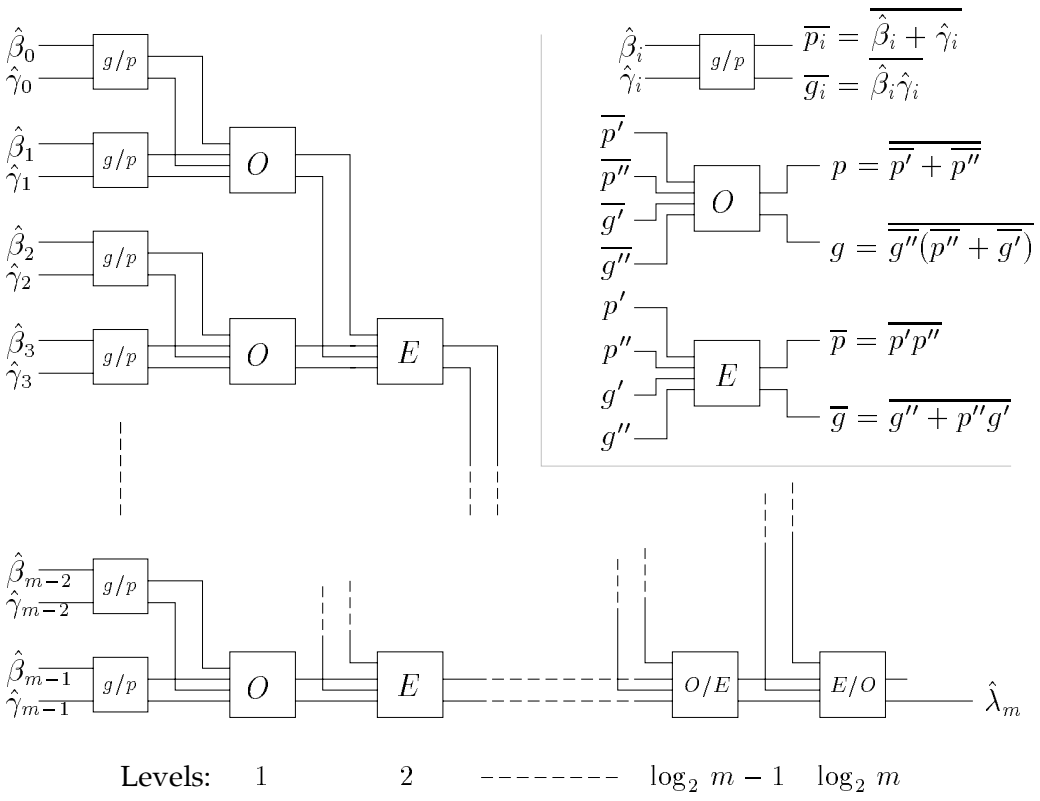


Figure 6.8: *A carry look-ahead tree that generates the carry signal $\hat{\lambda}_m = c'_m$. Each g/p cell has an inverted carry propagate and an inverted carry generate as output signals. The odd and even levels of the tree consist of the O cells and the E cells, respectively. The output functions of the three cells are displayed in the top-rightmost part of the figure.*

et al. [110, Fig. 4]. Furthermore, it is essentially a modified version of Brent and Kung's [27] well known carry look-ahead tree.

As seen in Figure 6.8, the g/p cells generate the inverted initial carry propagate and generate signals $\overline{p_i}$ and $\overline{g_i}$, respectively, for $0 \leq i \leq m-1$. Consider the tree subsequent to the array of g/p cells. This tree has $\log_2 m$ levels. By subsequently numbering the levels from 1 to $\log_2 m$ (from left to right for the tree in Figure 6.8), we deduce that the *odd* indexed levels of the tree are formed only by O cells and the *even* indexed levels are formed only by E cells. Consequently, the end cell is either an E cell or an O cell, depending on whether $\log_2 m$ is even or odd, respectively. Note that in the former case, the $\overline{g}$ output signal of the end cell must be inverted to form the desired carry signal $\hat{\lambda}_m$. Henceforth, we do not consider this extra inverter needed for even $\log_2 m$. The

Cell	Size	Input	Fan-in	Output norm. res.
g/p	8	$\hat{\beta}_i/\hat{\gamma}_i$	4	2
O	10	$\overline{p'}/\overline{g'}/\overline{g''}$	2	2
		$\overline{p''}$	4	
E	10	$p'/g'/g''$	2	2
		p''	4	

Table 6.2: *The sizes, fan-ins, and output normalised resistances of the g/p, O, and E cells of Figure 6.8. We use the cell names as subscripts of the complexity parameters, e.g. $r_{g/p} = 2$, $f_{E,p''} = 4$, and $\mathcal{C}_{E/O} = \mathcal{C}_E = \mathcal{C}_O = 10$.*

input and output signals of the g/p, O, and E cells are displayed in the top-rightmost part of Figure 6.8.

Each g/p cell consists of one NAND gate and one NOR gate. The E cell consists of one NAND gate and one combined AND–NOR gate (see Figure 6.2). The complexity parameters of the latter gate are given on page 99. Also, a schematic description of the gate is given in Figure 6.2. The O cell consists of one NOR gate and one gate which has a similar structure and the same complexity parameters as the AND–NOR gate. Recently, Wei and Thompson [112] derived an AT^2 optimal parallel carry look-ahead adder based on Brent and Kung's carry look-ahead tree. Two of their basic cells which they use to implement the parallel carry computation are equivalent to the E and O cells in Figure 6.8: Their 'black *ba*' cell [112, Fig. 3(a)] is equivalent to our E cell and their 'black *bb*' cell [112, Fig. 3(b)] is equivalent to our O cell. The sizes, the fan-ins and the output normalised resistances of the cells in Figure 6.8 are given in Table 6.2.

The binary carry look-ahead tree in Figure 6.8 comprises m g/p cells and $2m-1$ E and O cells. Hence, the size $\mathcal{C}_{\text{CLA}}$ of the tree equals

$$\mathcal{C}_{\text{CLA}} = m\mathcal{C}_{g/p} + (2m-1)\mathcal{C}_{E/O} = 8m + 10(2m-1) = 28m - 10. \qquad (6.27)$$

The values of $\mathcal{C}_{g/p}$ and $\mathcal{C}_{E/O}$ are taken from Table 6.2. By combining (6.27) and (6.24), we get the total size

$$\mathcal{C}_{\text{dimadd},1} = 56m + 12$$

of the diminished–1 adder of Figure 6.7. The fan-in f_{CLA} of the carry look-ahead tree equals $f_{g/p} = 2$. The output normalised resistance equals $r_{\text{CLA}} = r_{E/O} = 2$.

Regarding the E and O cells, because the respective p''- and $\overline{p''}$-inputs have the largest fan-ins, the CP is the path from either the $\hat{\beta}_{m-1}$ node or the $\hat{\gamma}_{m-1}$ node through the carry look-ahead tree to $\hat{\lambda}_m$ and onwards through the m-bit parallel adder to the $\hat{\varphi}_m$ output. With respect to this CP, the fan-in, the output normalised resistance, and the internal CP length equal

$$f_{\mathrm{dimadd},1} = f_{\mathrm{FA,signal}} + f_{g/p} = 8 + 4 = 12$$

$$r_{\mathrm{dimadd},1} = r_{\mathrm{NAND}} = 2$$

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,dimadd},1} &= r_{g/p} f_{O,\overline{p''}} + (\log_2 m - 1) r_{O/E} f_{E,p''/O,\overline{p''}} + r_{E/O} f_{\mathrm{NAND}} \\ &\quad + r_{\mathrm{NOR}} (f_{\mathrm{NOR}} + f_{\mathrm{FA,carry}}) + m\mathcal{L}_{\mathrm{FA,carry}} \\ &\quad + (m-1) r_{\mathrm{FA}} f_{\mathrm{FA,carry}} + (r_{\mathrm{FA}} + r_{\mathrm{NAND}}) f_{\mathrm{NAND}} \\ &= 2 \cdot 4 + (\log_2 m - 1) 2 \cdot 4 + 2 \cdot 2 + 2(2+6) + 8m \\ &\quad + (m-1) \cdot 1 \cdot 6 + (1+2) \cdot 2 \\ &= 14m + 8 \log_2 m + 20.\end{aligned}$$

A Carry Ripple Adder

The NBC adder of Figure 5.7 in Section 5.1.3 is a carry ripple type of adder. In Figure 6.9 we present an equally comparable diminished–1 carry ripple adder. We have $\hat{\lambda} = \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)}$. In the carry look-ahead adder of Figure 6.7, for case 1 (see page 109) the addend $\overline{\hat{\lambda}_m}$ is added to the sum $\hat{\lambda}^{(m-1)}$ by letting $\overline{\hat{\lambda}_m}$ be the carry input signal of the parallel adder. In contrast, the carry input signal of the parallel adder in Figure 6.9 is *always* equal to zero. Therefore, it is sufficient to use a half adder element in the least significant bit position of the adder. Furthermore, the addition of $\hat{\lambda}^{(m-1)}$ by $\overline{\hat{\lambda}_m}$ is carried out by multiplexing either the sum $\hat{\lambda}^{(m-1)}$ (for $\hat{\lambda}_m = 1$ in case 1) or $\hat{\lambda}^{(m-1)} + 1$ (for $\hat{\lambda}_m = 0$ in case 1) to the output. The addition of $\hat{\lambda}^{(m-1)}$ by 1 is carried out by the row of half adder elements in the figure, in accordance with the circuit in Figure 6.4 for code translation from the diminished–1 representation to the NBC representation. Here, because one of the inputs of the half adder element in the least significant bit position equals zero, the adder element can be simplified to an inverter (see Figure 6.9).

Let $\hat{\phi} = \hat{\lambda}^{(m-1)} + 1$, where $\hat{\lambda} = \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)}$ as before. For all three cases described in the beginning of Section 6.3.4, we introduce a Boolean function f to control which of $\hat{\phi}^{(m-1)}$ (for $f = 0$) or $\hat{\lambda}^{(m-1)}$ (for $f = 1$) should be passed to the output $\hat{\varphi}^{(m-1)}$. The most significant bit $\hat{\varphi}_m$ is generated separately. In

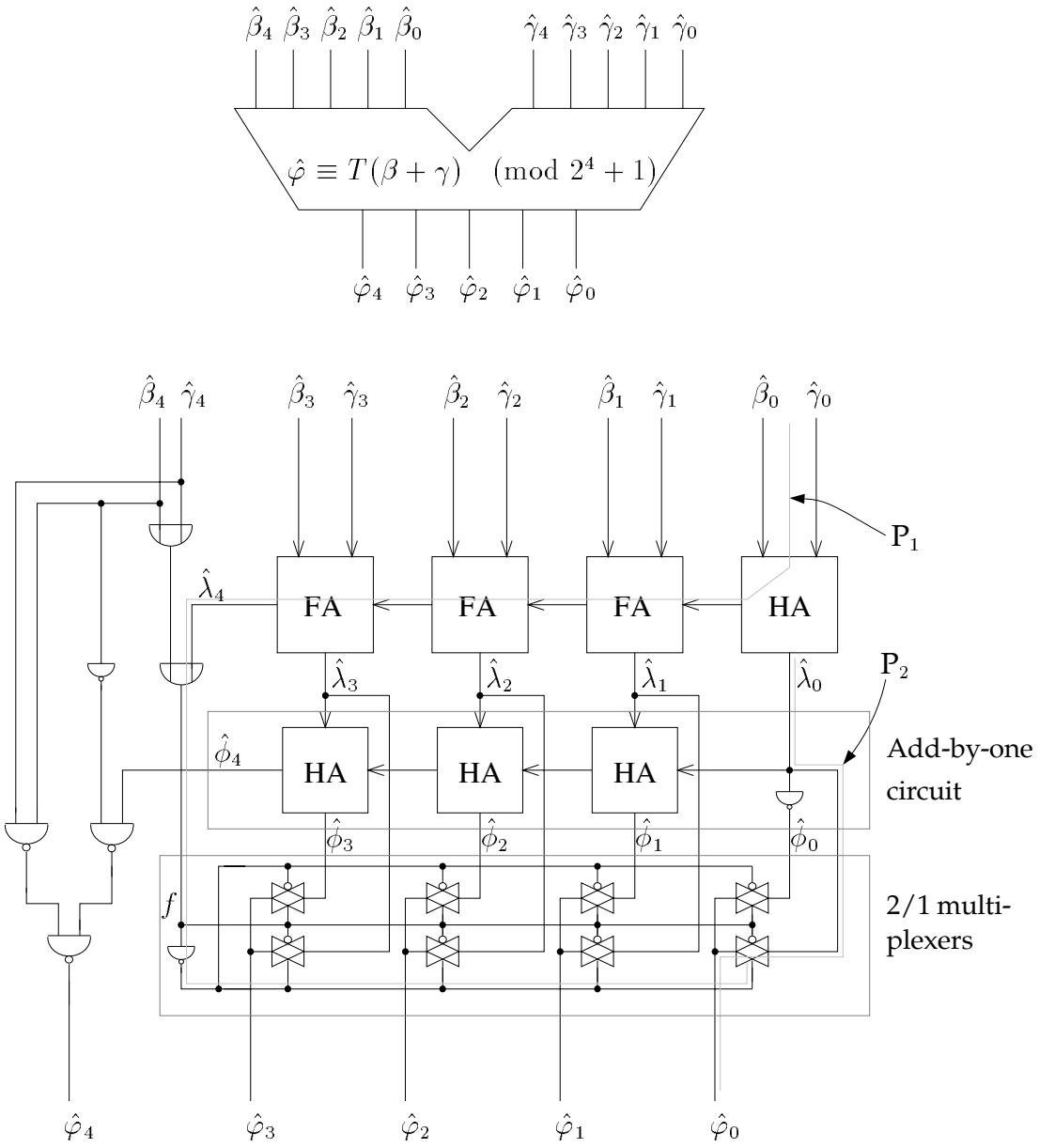


Figure 6.9: *A carry ripple architecture for diminished–1 addition modulo* 2^4+1*. The paths* P_1 *and* P_2 *form the CP through the circuit.*

Case	$\hat{\beta}_m$	$\hat{\gamma}_m$	$\hat{\lambda}_m$	$\hat{\phi}_m$	f	$\hat{\varphi}_m$
1	0	0	0	0	0	0
				1		1
			1	0	1	0
2	0/1	1/0	0	0/1	1	0
3	1	1	0	0	1	1

Table 6.3: *Properties of some variables of the addition circuit of Figure* 6.9. *The Boolean functions f and $\hat{\varphi}_m$ depend on the other variables.*

Table 6.3 we have listed the possible values of $\hat{\beta}_m$, $\hat{\gamma}_m$, $\hat{\lambda}_m$, $\hat{\phi}_m$, f, and $\hat{\varphi}_m$ for the three above-mentioned cases. By using Karnaugh maps for f and $\hat{\varphi}_m$ we obtain the minimised Boolean functions

$$\begin{aligned} f &= \hat{\beta}_m + \hat{\gamma}_m + \hat{\lambda}_m \\ \hat{\varphi}_m &= \hat{\gamma}_m\hat{\lambda}_m + \overline{\hat{\beta}_m}\hat{\phi} = \overline{\overline{\hat{\gamma}_m\hat{\lambda}_m} \cdot \overline{\hat{\beta}_m}\hat{\phi}}. \end{aligned}$$

These functions are formed by the logic gates in the leftmost part of Figure 6.9. By comparison with the adder in Figure 6.7, the size of the adder in Figure 6.9 is reduced by $6m + 10$ to

$$\mathcal{C}_{\text{dimadd},2} = (m-1)\mathcal{C}_{\text{FA}} + m\mathcal{C}_{\text{HA}} + m\mathcal{C}_{\text{MUX}} + 2\mathcal{C}_{\text{OR}} + 3\mathcal{C}_{\text{NAND}} + 3\mathcal{C}_{\text{inv}} = 50m + 2.$$

Also, as expected when comparing a carry ripple adder with a carry look-ahead adder, the internal CP length is greater for the carry ripple adder: The CP through the adder is formed by the two paths labelled P_1 and P_2 in Figure 6.9. Hence, for this adder we get the fan-in, output normalised resistance, and internal CP length

$$f_{\text{dimadd},2} = f_{\text{HA}} = 6$$

$$r_{\text{dimadd},2} = r_{\text{HA,sum}} + 1 = 3$$

$$\begin{aligned} \mathcal{L}_{\text{CP,dimadd},2} &= \mathcal{L}_{\text{HA,carry}} + r_{\text{HA,carry}} f_{\text{FA,carry}} + (m-1)\mathcal{L}_{\text{FA,carry}} \\ &\quad + (m-2) r_{\text{FA}} f_{\text{FA,carry}} + r_{\text{FA}} f_{\text{OR}} \\ &\quad + \mathcal{L}_{\text{OR}} + r_{\text{OR}}(2m + f_{\text{inv}}) + 2m r_{\text{inv}} \\ &= 4 + 6 + 8(m-1) + 6(m-2) + 2 + 4 + (2m+2) + 2m \\ &= 18m, \end{aligned}$$

respectively.

Carry Look-Ahead versus Carry Ripple Adder

In order to take all three delay parameters, i.e. the fan-in, the internal CP length, and the output normalised resistance, of each adder into account we assume as before that the adders are both preceded and followed by registers. Then, the computation times (T) of the carry look-ahead adder in Figure 6.7 and the carry ripple adder in Figure 6.9 are equal to

$$\begin{aligned}\mathcal{L}_{\text{dimadd},1} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{dimadd},1} + \mathcal{L}_{\text{CP,dimadd},1} + r_{\text{dimadd},1} f_{\text{reg}} \\ &= 22 + 2 \cdot 12 + 14m + 8 \log_2\ m + 20 + 2 \cdot 2 \\ &= 14m + 8 \log_2\ m + 70\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{\text{dimadd},2} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{dimadd},2} + \mathcal{L}_{\text{CP,dimadd},2} + r_{\text{dimadd},2} f_{\text{reg}} \\ &= 22 + 2 \cdot 6 + 18m + 3 \cdot 2 \\ &= 18m + 40, \qquad (6.28)\end{aligned}$$

respectively. Apparently, the carry look-ahead adder is faster than the carry ripple adder. By combining the size and the total CP length (including registers) of each adder, we find that the AT^2 performance of the two adders is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{dimadd},1} &\triangleq \mathcal{C}_{\text{dimadd},1}(\mathcal{L}_{\text{dimadd},1})^2 = (56m + 12)(14m + 8 \log_2\ m + 70)^2 \\ &= \mathcal{O}\left(m^3\right)\end{aligned}$$

$$\begin{aligned}\mathcal{CL}^2_{\text{dimadd},2} &\triangleq \mathcal{C}_{\text{dimadd},2}(\mathcal{L}_{\text{dimadd},2})^2 = (50m + 2)(18m + 40)^2 \\ &= \mathcal{O}\left(m^3\right),\end{aligned}$$

respectively. The sizes, total CP lengths, and $\mathcal{CL}^2$ products of the two adders are plotted versus m in Figure 6.10. We see that the values of the complexity parameters do not differ much between the adders. However, for all m the size of the carry look-ahead adder is slightly greater than the size of the carry ripple adder. For $m \geq 16$, the total CP length $\mathcal{L}_{\text{dimadd},1}$ of the carry look-ahead adder is less than the total CP length $\mathcal{L}_{\text{dimadd},2}$ of the carry ripple adder. For $m \leq 8$ we have $\mathcal{L}_{\text{dimadd},1} > \mathcal{L}_{\text{dimadd},2}$.

With respect to the AT^2 performance, the carry look-ahead adder is preferable to the carry ripple adder for $m \geq 32$, but the carry ripple adder is preferable for $m \leq 16$.

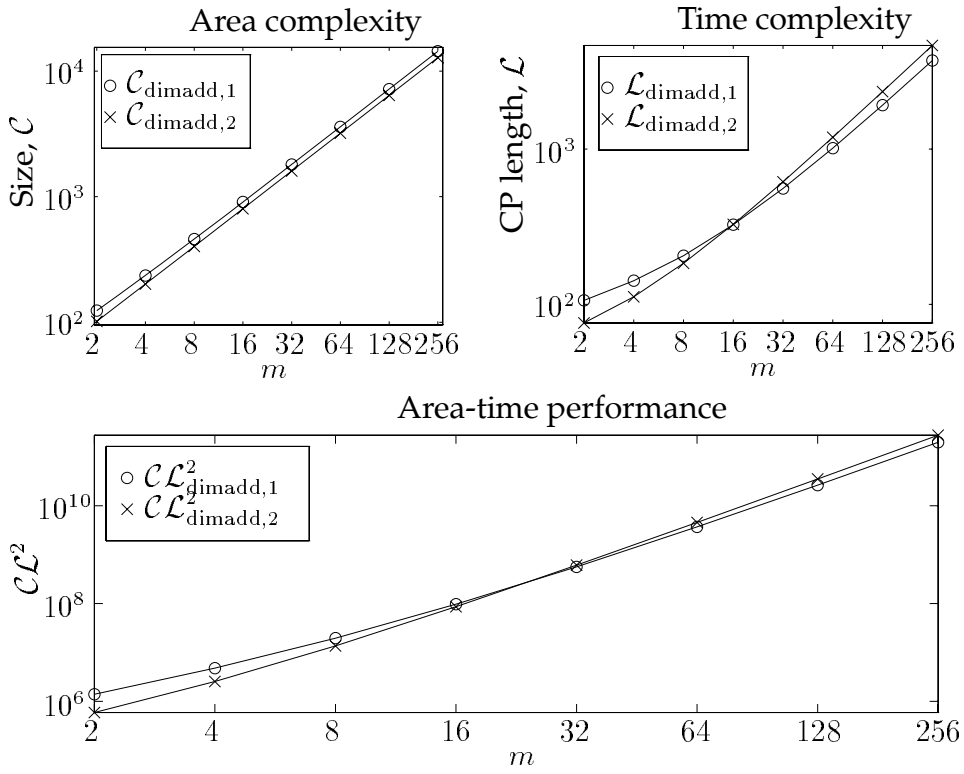


Figure 6.10: *The sizes, CP lengths, and AT^2 performances of the two diminished–1 adders. The parameters are plotted versus m for $m = 2, 4, 8, 16, 32, 64, 128, 256$.*

A Bit-Serial Adder

The chip area required to perform an arithmetic operation is usually smaller for bit-serial architectures than for bit-parallel architectures. Another advantage of bit-serial architectures is that they can be clocked with a higher clock frequency, i.e. they have a higher throughput. However, it is not certain that the *total* time required to perform an operation is smaller for a bit-serial architecture than for a bit-parallel architecture.

The bit-serial *carry-save adder* of Figure 6.11 adds the two binary coded numbers $\hat{\beta}$ and $\hat{\gamma}$. Here, we assume that the binary digits of $\hat{\beta}$ and $\hat{\gamma}$ are fed into the adder element with the least significant bits first. Each digit of the sum $\hat{\sigma} = (\ldots, \hat{\sigma}_2, \hat{\sigma}_1, \hat{\sigma}_0)_2$ is either directly stored in a shift register or first manipulated in some way, for example to form the sum $\hat{\sigma} + 1$ (according to diminished–1

addition), before it is stored. In any case, assuming that $\hat{\beta}_i$ and $\hat{\gamma}_i$ are the outputs from two (shift) registers, the internal CP length of the bit-serial adder can not be less than the length of the path from the input register through the full adder element (from the signal input to the sum output) to an output register. This minimal CP length equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,seradd,min}} &\triangleq \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} f_{\mathrm{FA,signal}} + \mathcal{L}_{\mathrm{FA,sum}} + r_{\mathrm{FA}} f_{\mathrm{reg}} \\ &= 22 + 2 \cdot 8 + 12 + 1 \cdot 2 \\ &= 52.\end{aligned}$$

It takes at least $m + 1$ clock cycles before the desired diminished–1 sum $\hat{\beta} + \hat{\gamma} + 1 \bmod 2^m + 1$ is present at the adder output, whether it is in bit-serial or bit-parallel form. Hence, the total computation time is proportional to to the length $\mathcal{L}_{\mathrm{seradd,min}}$, for which we have

$$\mathcal{L}_{\mathrm{seradd,min}} \geq (m+1)\mathcal{L}_{\mathrm{CP,seradd,min}} = 52(m+1).$$

This length is also greater than the total CP lengths $\mathcal{L}_{\mathrm{dimadd},1}$ and $\mathcal{L}_{\mathrm{dimadd},2}$ of the carry look-ahead adder and the carry ripple adder, respectively. We assert that from an AT^2 performance point of view, when comparing the bit-serial adder with the carry look-ahead and carry ripple adders, the bit serial adder is not competitive. The bit-serial adder is not further considered here.

Other Adders

In addition to the adders described above, we would like to mention some other diminished–1 adders that have been presented in the literature. In this thesis, we do not consider the complexity or performance of these adders.

Firstly, because we let the adder block of the diminished–1 adder in Figure 6.7 be a carry ripple adder, the complete adder architecture is not a true carry look-ahead adder. McClellan [65, Fig. 8], however, implements this adder part of the circuit using length-4 arithmetic logic units and carry look-ahead logic blocks.

Secondly, Towers et al. [101, Fig. 10] and Pajayakrit [71, Fig. 3.4] propose a true carry look-ahead diminished–1 adder that is based on McClellan's adder [65, Fig. 8]. They forward the generate and propagate signals obtained in the carry look-ahead block (see Figure 6.7) to an array of 4-bit carry look-ahead units, which in turn is followed by an array of XOR gates forming the sum output. The resulting adder is implemented in nMOS technology. The authors state that their carry look-ahead scheme "seemed to be the best in terms of area and speed".

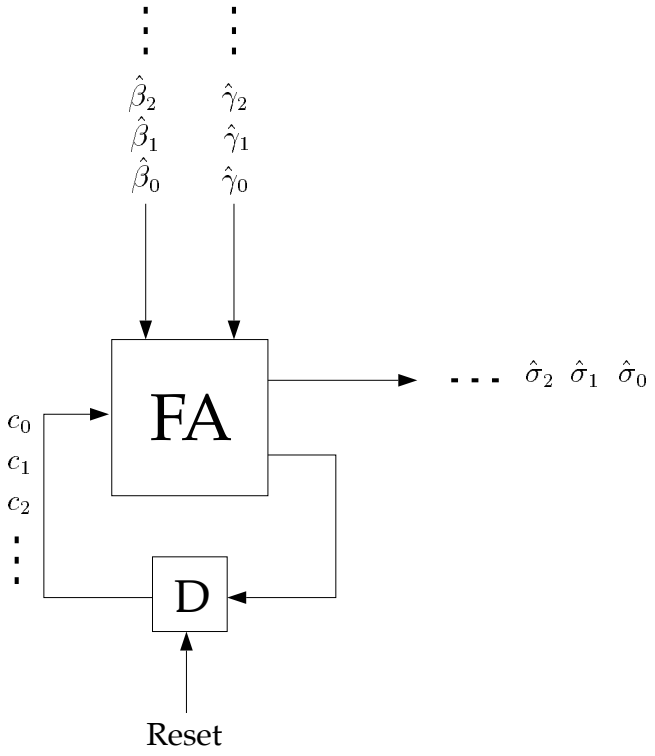


Figure 6.11: *The adder part of a diminished–1 bit-serial adder.*

Thirdly, Morikawa et al. [68] have implemented a three-input diminished–1 adder, i.e. an adder that adds three numbers at a time. Their adder is based on the three-input carry-save adder presented by Hwang [52, Ch. 4.2]. Recently, Benaissa et al. [13, Fig. 5] presented a VLSI design of a Fermat number transform using three-input adders. Benaissa's adder is an improved version of Morikawa's adder.

Subtraction

In the end of Section 5.1.3 we wrote that subtraction is simplest carried out by first negating the subtrahend and then adding the result to the minuend. This should also be the most straightforward procedure for diminished–1 subtraction. Thus, subtraction can be performed using the negation architecture in Figure 6.6 and any of the two-input adders described in the present section.

6.3.5 Multiplication by Powers of 2

Multiplication by 2

Multiplication by two is simply performed in VLSI when using the diminished–1 representation. By letting $\gamma = \beta$ in (6.22), we get

$$\begin{aligned}
\hat{\varphi} &\triangleq T(2\beta) \equiv 2\hat{\beta} + 1 \\
&= \left(2\hat{\beta}_m + \hat{\beta}_{m-1}\right) 2^m + 2\hat{\beta}^{(m-2)} + 1 \equiv 2\hat{\beta}^{(m-2)} - 2\hat{\beta}_m + \overline{\hat{\beta}_{m-1}} \\
&= \begin{cases} 2 \cdot 0 - 2 \cdot 1 + 1 = -1 \equiv 2^m = \hat{\beta}; & \text{if } \hat{\beta} = 2^m \text{ (i.e. if } \beta = 0) \\ 2\hat{\beta}^{(m-2)} + \overline{\hat{\beta}_{m-1}}; & \text{if } 0 \leq \hat{\beta} \leq 2^m - 1 \end{cases} \\
&= (\hat{\varphi}_m, \hat{\varphi}_{m-1}, \ldots, \hat{\varphi}_1, \hat{\varphi}_0)_2 \pmod{2^m + 1},
\end{aligned}$$

where $\hat{\varphi}_m = \hat{\beta}_m$ and $\hat{\varphi}_i = \hat{\beta}_{i-1}$ for $i = 1, 2, \ldots, m-1$ holds for all elements $\hat{\beta} \in \mathbb{Z}_{2^m+1}$. For $\hat{\beta} = 2^m$ we get $\hat{\varphi}_0 = 0$ and for $0 \leq \hat{\beta} \leq 2^m - 1$ we get $\hat{\varphi}_0 = \overline{\hat{\beta}_{m-1}}$. Hence, the binary digits of $\hat{\varphi}$ are formed as

$$\begin{cases} \hat{\varphi}_0 = \overline{\hat{\beta}_m} \cdot \overline{\hat{\beta}_{m-1}} = \overline{\hat{\beta}_m + \hat{\beta}_{m-1}} \\ \hat{\varphi}_i = \hat{\beta}_{i-1}; \quad 1 \leq i \leq m-1 \\ \hat{\varphi}_m = \hat{\beta}_m \end{cases}$$

Figure 6.12 shows an architecture for multiplication by 2. The CP of this simple architecture is the path from the $\hat{\beta}_m$-input node to the $\hat{\varphi}_0$-output node. With respect to this CP, the size of the circuit, its fan-in, and its output normalised resistance equal

$$\begin{aligned}
\mathcal{C}_{\text{dimmult2}} &= \mathcal{C}_{\text{NOR}} = 4 \\
f_{\text{dimmult2}} &= n_m + f_{\text{NOR}} = n_m + 4 \\
r_{\text{dimmult2}} &= r_{\text{NOR}} = 2,
\end{aligned}$$

respectively, where n_m is the circuit fan-out with respect to the $\hat{\varphi}_m$-output node. There is no internal stage. If n_m is (much) greater than 2, the circuit performance can be improved by connecting the $\hat{\varphi}_m$-output to a simple driver (two cascaded inverters). Then, $n_m = f_{\text{inv}} = 2$ and thus the fan-in f_{dimmult2} equals 6.

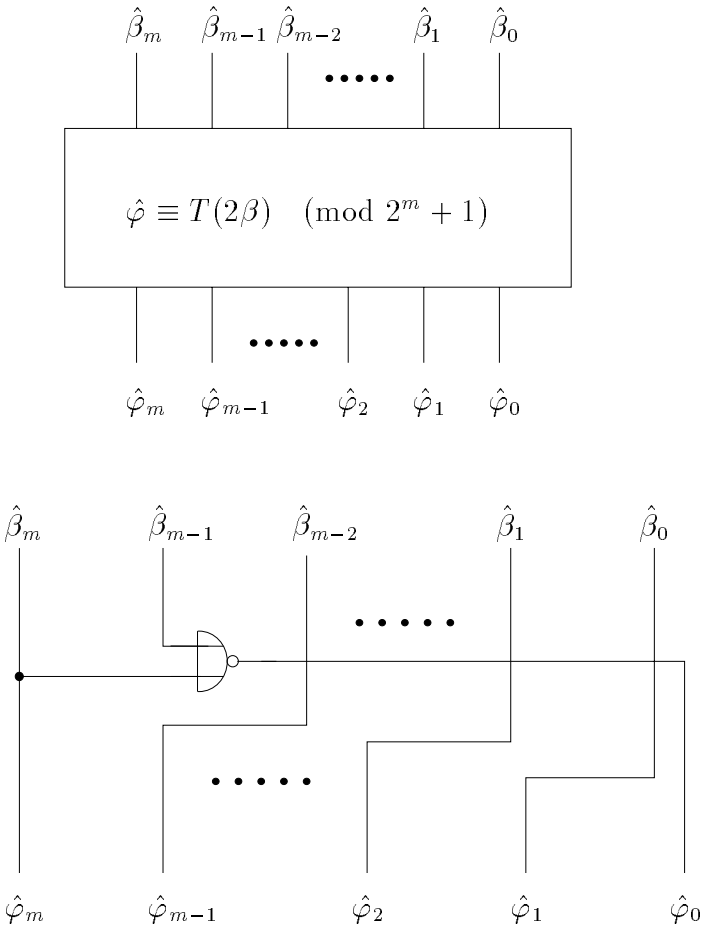


Figure 6.12: *Diminished–1 multiplication by 2 modulo* $2^m + 1$

The total CP length of the multiplication-by-2 circuit in Figure 6.12, including registers[8], equals

$$\begin{aligned} \mathcal{L}_{\text{dimmult2}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{dimmult2}} + r_{\text{dimmult2}} f_{\text{reg}} \\ &= 22 + 2(2+4) + 4 \cdot 2 = 42. \end{aligned}$$

Hence, the area-time performance is proportional to the product

$$\mathcal{C}\mathcal{L}^2_{\text{dimmult2}} = 4 \cdot 42^2 = 7056.$$

[8]Thus, we have $n_m = r_{\text{reg}} = 2$.

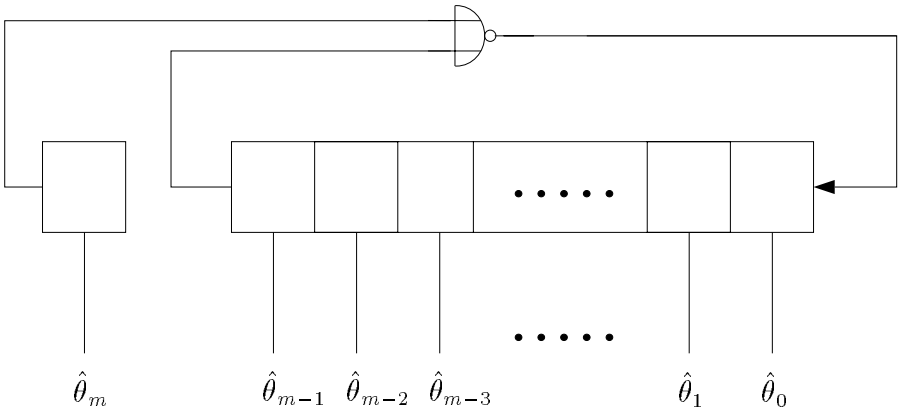


Figure 6.13: *A feedback shift register for repeated multiplication by 2.*

Multiplication by 2^n

Some computations, like for example bit-serial/parallel multiplication, involve repeated multiplication by 2. Repeated multiplication by 2 may be conveniently implemented as a feedback shift register with a NOR gate in the feedback loop, as shown in Figure 6.13. This circuit is based on the circuit in Figure 6.12.

The feedback shift register is initially loaded with $\hat{\beta}$. After k clock cycles, the contents of the register (including the single register element holding the most significant bit) equals $T(2^k\beta) \triangleq \hat{\theta}(k) = (\hat{\theta}_m, \hat{\theta}_{m-1}, \ldots, \hat{\theta}_1, \hat{\theta}_0)_2 \pmod{2^m+1}$. As concluded in Section 5.1.4 (page 80), for $t = \log_2 m$ only the $t+1$ least significant bits of the exponent n have to be considered when computing $T(2^n\beta)$.[9] Hence, at most $k = n^{(t)}$ clock cycles are required to compute $T(2^n\beta)$ using the circuit in Figure 6.13.

Furthermore, in Section 5.1.4 we also concluded from (5.8) that it is enough to first compute $\hat{\theta}(n^{(t-1)}) \equiv T(2^{n^{(t-1)}}\beta) \pmod{2^m+1}$ and then, if and only if $n_t = 1$, negate $\hat{\theta}(n^{(t-1)})$ to obtain the desired result.[10] Diminished–1 negation is performed by inverting the m least significant bits of the register contents $\hat{\theta}(n^{(t-1)})$. If n_t equals zero or $\hat{\theta}_m$ equals one, the negation does not take place. Let $\hat{\varphi} \equiv T(2^n\beta) \pmod{2^m+1}$. The binary digit $\hat{\varphi}_i$ is obtained from the Karnaugh map in Figure 6.14 as the Boolean function

$$\hat{\varphi}_i = \overline{\hat{\theta}_i}\,\overline{\hat{\theta}_m} n_t + \hat{\theta}_i(\hat{\theta}_m + \overline{n_t}) = \overline{\hat{\theta}_m} n_t \oplus \hat{\theta}_i, \tag{6.29}$$

[9] Because $\mathrm{ord}_{2^m+1} 2 = 2m = 2^{t+1}$ it is enough to consider $n^{(t)} = n \bmod 2m$.

[10] By (5.8) we get $T(2^n\beta) \equiv T(2^{n^{(t-1)}}) \pmod{2^m+1}$ if $n_t = 0$ and $T(2^n\beta) \equiv T(-2^{n^{(t-1)}}) \pmod{2^m+1}$ if $n_t = 1$.

$\hat{\theta}_m n_t$

$\hat{\theta}_i$	00	01	11	10
0	0	1	0	0
1	1	0	X	X

$\hat{\varphi}_i$

Figure 6.14: *Karnaugh map for the output bit $\hat{\varphi}_i$ of $\hat{\varphi}$ for $0 \leq i \leq m-1$.*
X = *"don't care".*

where $\hat{\theta}_i$ is the contents in bit position i of the feedback register. Figure 6.15 shows an architecture that performs the operation $\hat{\varphi} \equiv T(2^n\beta) \pmod{2^m+1}$ using repeated multiplication by 2 according to the above procedure. The control logic is not included in the figure. According to the Karnaugh map in Figure 6.14, $\hat{\varphi}_i$ can also be formed by other Boolean functions, depending on which values are assigned to the "don't cares". However, the function in (6.29) results in the most efficient realisation (the array of XOR gates), with respect to the area-time performance.

The size of the architecture in Figure 6.15 equals

$$\begin{aligned} \mathcal{C}_{\text{seq,mult2n}} &= (m+1)\mathcal{C}_{\text{reg}} + m\mathcal{C}_{\text{XOR}} + \mathcal{C}_{\text{NOR}} + \mathcal{C}_{\text{AND}} + \mathcal{C}_{\text{inv}} \\ &= 28m - 4. \end{aligned}$$

The internal CP during the shift operation is the feedback path P_1 from the register holding $\hat{\theta}_{m-1}$ through the NOR gate to the register in the least significant bit position. This path has length

$$\mathcal{L}_{\text{CP,seq,mult2n}} = \mathcal{L}_{\text{reg}} + r_{\text{reg}}(f_{\text{XOR}} + f_{\text{NOR}}) + r_{\text{NOR}} f_{\text{reg}} = 38.$$

During an initial clock cycle, $\hat{\theta}$ is loaded into the shift register. After the $n^{(t-1)}$ subsequent clock cycles, the shift register contains the diminished–1 integer $T(2^{n^{(t-1)}})$. An extra clock cycle is then required to shift this result through the array of XOR gates to the output. Assuming that $\hat{\varphi}$ is directly stored in a register, the length of this final output path (which is named P_3 in Figure 6.15) equals

$$\mathcal{L}_{\text{P3}} = \mathcal{L}_{\text{reg}} + r_{\text{reg}}(f_{\text{XOR}} + f_{\text{NOR}}) + \mathcal{L}_{\text{XOR}} + r_{\text{XOR}} f_{\text{reg}} = 40,$$

which is slightly greater than the length $\mathcal{L}_{\text{CP,seq,mult2n}}$ of the internal critical path P_1. Therefore, by letting the clock interval be proportional to $\mathcal{L}_{\text{P3}}$, the time

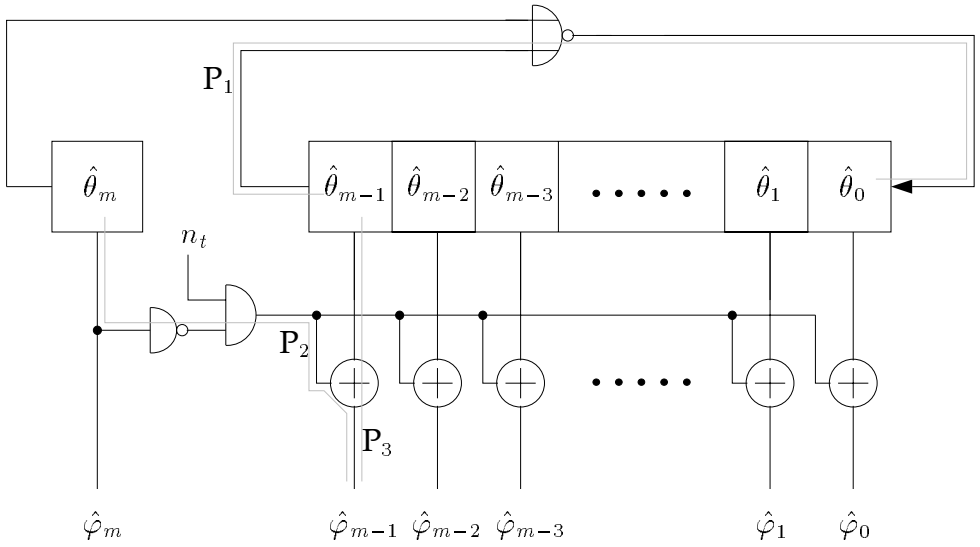


Figure 6.15: *An architecture for diminished–1 multiplication by a power of 2.*

T required to perform the multiplication by 2^n modulo $2^m + 1$ is proportional to

$$\mathcal{L}_{\text{seq,mult2n}} = (n^{(t-1)} + 2)\mathcal{L}_{\text{P3}} = 40(n^{(t-1)} + 2).$$

Because $0 \leq n^{(t-1)} \leq 2^t - 1 = m - 1$, the *maximum* multiplication time is proportional to $40(m + 1)$. When the circuit in Figure 6.15 is followed by a register, the length $\mathcal{L}_{\text{P2}}$ of path P_2 of the figure equals

$$\begin{aligned}\mathcal{L}_{\text{P2}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}}(f_{\text{reg}} + f_{\text{inv}}) + r_{\text{inv}} f_{\text{AND}} + \mathcal{L}_{\text{AND}} \\ &\quad + r_{\text{AND}} \cdot m f_{\text{XOR}} + \mathcal{L}_{\text{XOR}} + r_{\text{XOR}} f_{\text{reg}} \\ &= 4m + 42.\end{aligned}$$

Therefore, for $\mathcal{L}_{\text{seq,mult2n}} < \mathcal{L}_{\text{P2}}$, i.e. for $n^{(t-1)} < \lceil (4m+42)/40 - 2 \rceil = \lceil m/10 \rceil - 1$, the computation time T is proportional to $\mathcal{L}_{\text{P2}}$ and hence, for $n^{(t-1)} \geq \lceil m/10 \rceil - 1$ the computation time is proportional to $\mathcal{L}_{\text{seq,mult2n}}$. The AT^2 performance of the circuit is proportional to the product

$$\mathcal{C}\mathcal{L}^2_{\text{seq,mult2n}} \triangleq$$

$$= \begin{cases} \mathcal{C}_{\text{seq,mult2n}}(\mathcal{L}_{\text{P2}})^2 = (28m - 4)(4m + 42)^2 \\ \qquad = \mathcal{O}(m^3); \\ \qquad \text{for } 0 \leq n^{(t-1)} < \lceil m/10 \rceil - 1 \\ \\ \mathcal{C}_{\text{seq,mult2n}}(\mathcal{L}_{\text{seq,mult2n}})^2 = (28m - 4)(40n^{(t-1)} + 80)^2 \\ \qquad = \mathcal{O}\left(m(n^{(t-1)})^2\right); \\ \qquad \text{for } \lceil m/10 \rceil - 1 \leq n^{(t-1)} \leq m - 1 \end{cases}$$

Note that for a *nonzero* integer β, its corresponding diminished–1 integer $\hat{\beta}$ is an element of $\mathbb{Z}_{2^m}$, i.e. we have $\hat{\beta}_m = 0$. Because the odd Fermat number 2^m+1 is not divisible by 2, for $n \in \mathbb{N}$ every integer $2^n\beta \bmod 2^m+1$ is also a nonzero integer. Thus, for all such nonzero numbers β we get $\hat{\varphi}_m = 0$, where $\hat{\varphi} \equiv T(2^n\beta) \pmod{2^m+1}$, and hence the NOR gate in the feedback path can be replaced by an inverter. Also, for $0 \le i \le m-1$, each output bit $\hat{\varphi}_i$ of the circuit is then formed by the Boolean function $\hat{\varphi}_i = n_t \oplus \hat{\theta}_i$,. The procedure for repeated multiplication by 2 using only a feedback shift register with an inverter in the feedback loop was originally described by Leibowitz [58].

Multiplication Using a Modified Barrel Shifter

In 1982, Truong et al. [103] proposed a method of computing multiplication by a power of two using a modified Barrel shifter. To the author's knowledge, Truong's multiplication method (or modified versions of the method) is used by most people when implementing diminished–1 multiplication by powers of two. For example, Pajayakrit [71, Ch. 3.6] and Towers et al. [101, Sec. 11.1.3] propose an nMOS architecture which comprises *two* modified Barrel (circular) shifters. This architecture works for both negative and positive exponents n, in order to be applicable in the computation of the *inverse* Fermat number transform as well as the forward transform.

One of the shifters is a diminished–1 left shifter, which is used for positive exponents. The second diminished–1 shifter is used when the exponent is negative. This shifter shifts the input to the right. The shifters are controlled by a decoder. Figure 6.16 shows a block diagram for such a multiplier over $\mathbb{Z}_{2^m+1}$. The signal $ctrl$ in the figure controls which of the shifters is to be activated.

If the exponent is always nonnegative (or nonpositive), i.e. we have $\hat{\varphi} \equiv T(2^{n^{(t)}}\beta) \pmod{2^m+1}$ (or $\hat{\varphi} \equiv T(2^{-n^{(t)}}\beta) \pmod{2^m+1}$), then it is sufficient to use only *one* modified Barrel shifter. Figure 6.17(a) shows a block diagram of Truong's modified Barrel (left) shifter together with a decoder. The decoder has $t+1$ inputs and, consequently, $2^{t+1} = 2m$ outputs. Hence, the size of the shifter is $\mathcal{O}(m \cdot 2m)$. The modified Barrel shifter differs from an ordinary Barrel shifter only in the wirings of its transistors. An architecture of a transmission-gate based modified Barrel shifter over $\mathbb{Z}_{2^4+1}$ is presented in a paper by Shakaff et al. [90, Fig. 6].

In Figure 6.17(b), we present a block diagram of a multiplier for which the size of the decoder is half the size of the decoder in Truong's architecture. The decoder has the t-bit NBC number $n^{(t-1)}$ as its input and it has $2^t = m$ outputs. Therefore, the size of the Barrel shifter is $\mathcal{O}(m \cdot m)$, i.e. half the size of the shifter needed in Truong's architecture. The output of the reduced-size shifter

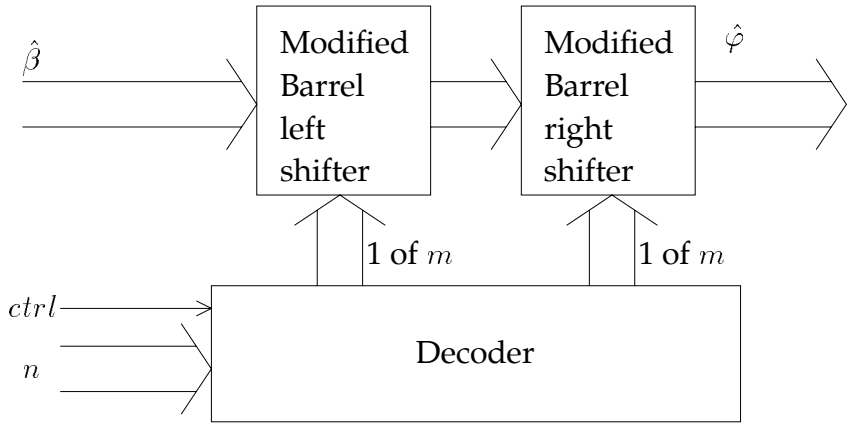


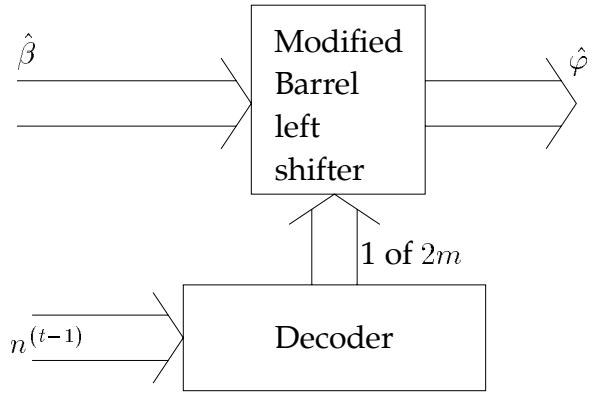
Figure 6.16: *A diminished–1 multiplier of a power of two modulo $2^m + 1$ (from [71, Fig. 3.8] but using our notations). The signal $ctrl$ controls whether the input β is to be shifted to the left (for a positive exponent) or to the right (for a negative exponent). The output $\hat{\varphi}$ equals $\hat{\varphi} = T(2^{\pm n}\beta)$, where $n \in \mathbb{N}$.*

equals $\hat{\lambda} \equiv T(2^{n^{(t-1)}}\beta) \pmod{2^m + 1}$ (we assume that the exponent is positive). If n_t equals one, $\hat{\lambda}^{(t-1)}$ is inverted to form the desired result $\hat{\varphi} \equiv T(2^n\beta) \equiv T(-2^{n^{(t-1)}}\beta) \pmod{2^m + 1}$. If n_t equals zero, $\hat{\lambda}$ is passed unchanged to the output. The inversion is carried out by a row of XOR gates, as in Figure 6.15.

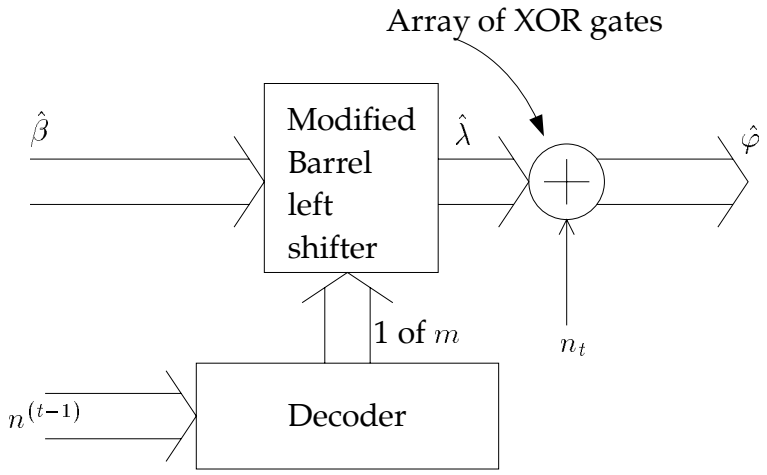
The decoder can be implemented in several ways, see for instance Weste and Eshraghian [113, Ch. 8.3.1.1.3]. The choice of implementation may for example be governed by speed requirements, power dissipation constraints, and chip size constraints. In this section, we do not give any further details about the complexities and performances of the Barrel-shifter type of architectures in Figures 6.16 and 6.17.

6.3.6 General Multiplication

Several algorithms and architectures for diminished–1 general multiplication have appeared in the literature. To our knowledge, only bit-serial/parallel and bit-parallel architectures are suggested by the originators of these architectures.



(a)



(b)

Figure 6.17: *Diminished–1 multiplication of a positive power of two modulo* $2^m + 1$*, using* one *shifter.* (a) *Truong's modified Barrel shifter of size* $m \times 2m$ *bits. The output equals* $\hat{\varphi} \equiv T(2^n \beta) \pmod{2^m + 1}$. (b) *A modified Barrel shifter of size* $m \times m$ *bits, followed by an array of XOR gates. For the output* $\hat{\varphi}^{(m-1)}$ *we have* $\hat{\varphi}_i = \hat{\lambda}_i \oplus n_t$*, where* $0 \leq i \leq m-1$ *and* $\hat{\lambda} \equiv T(2^{n^{(t-1)}} \beta) \pmod{2^m + 1}$.

6.3.6.1 Bit-Parallel Architectures

Leibowitz [58] was the first to present some procedures for general multiplication. He briefly considers three procedures,[11] which are based on how the multiplicand and the multiplier are represented:

1. The multiplicand and the multiplier are both diminished–1 numbers.
2. The multiplicand and the multiplier are both NBC numbers.
3. The multiplicand is an NBC number and the multiplier is a diminished–1 number, or vice versa.

The third multiplication procedure is generally used only for bit-serial/parallel multiplication. In the literature, we have not found any architecture for bit-*parallel* diminished–1 general multiplication that is based on such a procedure. In this section we give an analytical description of the above multiplication procedures 1 and 2, beginning with the first procedure. For nonzero factors β and γ, i.e for the diminished–1 numbers $0 \leq \hat{\beta}, \hat{\gamma} \leq 2^m - 1$, the product in (6.10) can be written as

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + (\hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} + 1) - 1 \\ &= \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\sigma} - 1 \\ &= 2^m \sum_{i=0}^{m-1} \hat{\theta}_{m+i} 2^i + \hat{\theta}^{(m-1)} - 1 \\ &\equiv -\hat{\lambda} + \hat{\theta}^{(m-1)} - 1 \pmod{2^m + 1}, \end{aligned} \tag{6.30}$$

where $\hat{\sigma} \triangleq \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} + 1 = \hat{\beta}^{(m-1)} \oplus \hat{\gamma}^{(m-1)} \pmod{2^m + 1}$ and where $\hat{\theta} = \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\sigma}$ is a $2m$-bit NBC integer and $\hat{\lambda} = \sum_{i=0}^{m-1} \hat{\theta}_{m+i} 2^i$ is an m-bit NBC integer. Therefore, we have $-\hat{\lambda} - 1 \equiv (2^m - 1) - \hat{\lambda}^{(m-1)} + 1 = \overline{\hat{\lambda}^{(m-1)}} + 1 \pmod{2^m + 1}$, where $\overline{\hat{\lambda}^{(m-1)}}$ is the one's complement of $\hat{\lambda}^{(m-1)}$ and hence (6.30) can be expressed as

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \hat{\theta}^{(m-1)} + \overline{\hat{\lambda}^{(m-1)}} + 1 \\ &= \hat{\theta}^{(m-1)} \oplus \overline{\hat{\lambda}^{(m-1)}} \pmod{2^m + 1}. \end{aligned} \tag{6.31}$$

Hence, the above procedure 1 involves one ordinary $(m \times m)$-bit general multiplication, one ordinary $2m$-bit addition, and two diminished–1 additions (see Example 12 in Leibowitz' paper [58]).

[11] Leibowitz did not formulate any *algorithm* for diminished–1 general multiplication. He only sketched the main steps of the proposed multiplication procedures and presented two examples.

Sunder's Parallel Multiplier

An $(m \times m)$-bit multiplication can be carried out using a conventional (Braun) bit-parallel multiplier array, see for instance Weste and Eshraghian [113, Ch. 8.2.7.1 (Figures 8.36 and 8.37)] or Hwang [52, Ch. 6.1 (Fig. 6.3)]. Sunder et al. [97] recently proposed an architecture for diminished–1 general multiplication based on the above procedure (Equation (6.30). They modified the conventional array multiplier such that the addition of $\hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)}$ by the *non-reduced* sum $\hat{\sigma}$ is performed in the multiplier array. Thus, following the notations above, the output of the array is the $(2m+1)$-bit NBC integer

$$\begin{aligned}\hat{\theta} &= \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} + 1 \\ &= 2^m \sum_{i=0}^{m} \hat{\theta}_{m+i} 2^i + \hat{\theta}^{(m-1)},\end{aligned}$$

rather than just the $2m$-bit NBC integer $\hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)}$. Here, we consequently let $\hat{\lambda} = \sum_{i=0}^{m} \hat{\theta}_{m+i} 2^i$ (compare this $(m+1)$-bit $\hat{\lambda}$ with the m-bit $\hat{\lambda}$ used in (6.30) and (6.31)). By (5.5) we get $-\hat{\lambda} \equiv \overline{\hat{\lambda}} + 3 \pmod{2^m + 1}$, which implies that $T(\beta \cdot \gamma)$ can be written as

$$\begin{aligned}T(\beta \cdot \gamma) &\equiv \hat{\theta} - 1 \equiv \hat{\theta}^{(m-1)} + (\overline{\hat{\lambda}} + 1) + 1 \\ &= \hat{\theta}^{(m-1)} \oplus (\overline{\hat{\lambda}} + 1) \pmod{2^m + 1}.\end{aligned}$$

The addend $\overline{\hat{\lambda}} + 1$ can be obtained using the procedure for diminished–1 negation. Figure 6.18 shows the modified multiplier array. The addends $\hat{\beta}^{(m-1)}$ and $\hat{\gamma}^{(m-1)}$ are added to the sum of partial products in the first row of the array. The addition by one is carried by the rightmost column of half adder elements.[12]

A block diagram of Sunder's bit-parallel diminished–1 pipelined multiplier is shown in Figure 6.19. The output from the diminished–1 adder of the architecture is the sum $\hat{\varphi} \equiv \hat{\theta}^{(m-1)} \oplus (\overline{\hat{\lambda}} + 1) \pmod{2^m + 1}$, where $\hat{\theta}$ and $\hat{\lambda}$ are defined as above. The desired product $T(\beta \cdot \gamma)$ equals $\hat{\varphi}$ only for nonzero inputs, i.e. for $\beta, \gamma \neq 0$. When either β or γ (or both) equals zero, we have $\hat{\beta}_m = 1$ ($\hat{\beta} = 2^m$) and $\hat{\gamma}_m = 1$ ($\hat{\gamma} = 2^m$), respectively, and $T(\beta \cdot \gamma) \equiv 2^m \pmod{2^m + 1}$. However, when only *one* of β and γ equals zero, the adder output $\hat{\varphi}$ of Figure 6.19 is nonzero. The correct output is formed by a row of inverters and NOR gates, see Figure 2 in Sunder's paper [97]. Sunder names this circuit the *output controller*.

For $0 \leq i \leq m-1$, the output bit in position i can be expressed as the Boolean function $\overline{(\hat{\beta}_m + \hat{\gamma}_m) + \overline{\hat{\varphi}_i}}$, i.e. its value can be generated using one inverter and

[12] Compare with the equivalent add-by-one circuit in Figure 6.4.

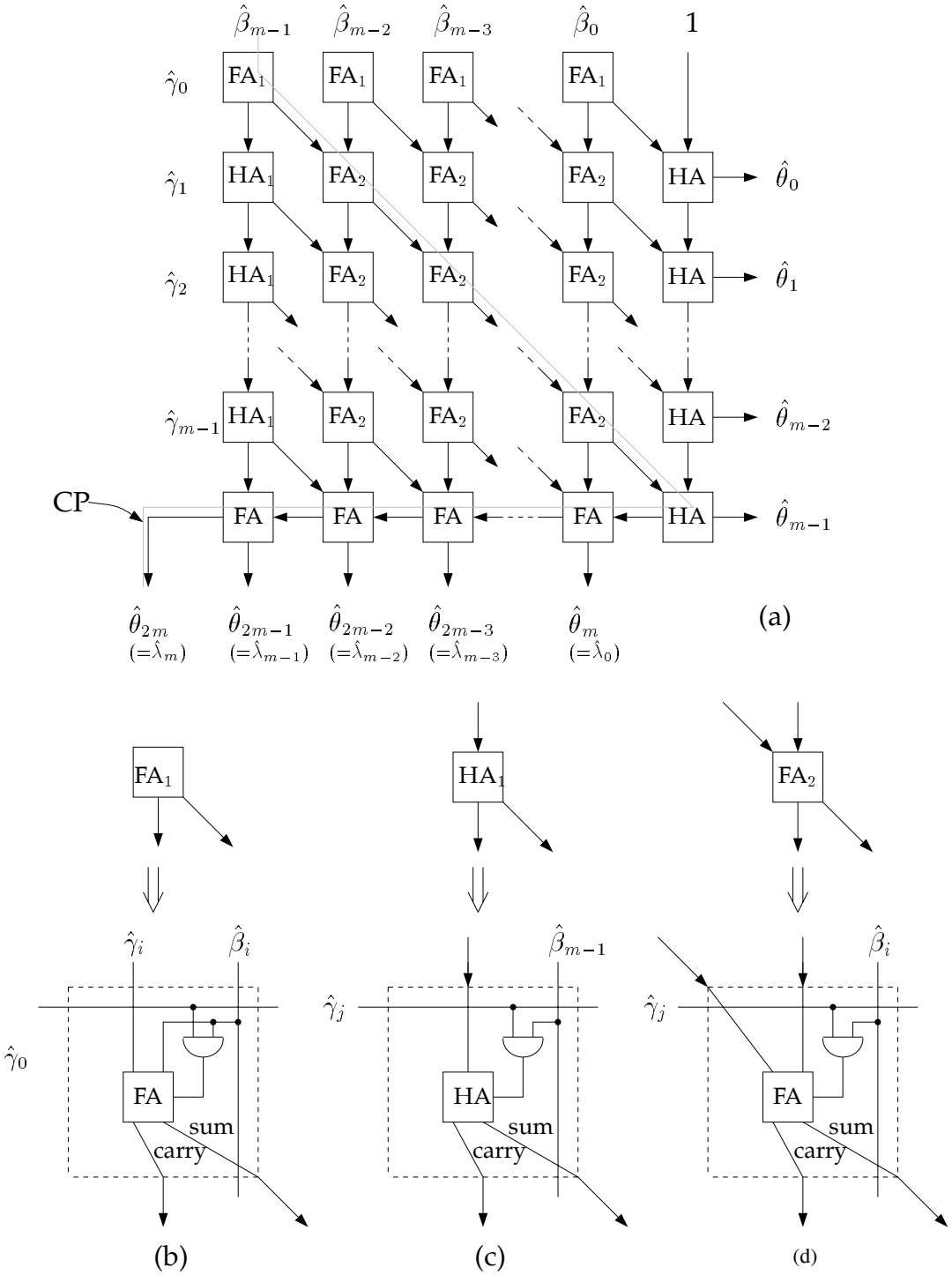


Figure 6.18: *A modified* $(m \times m)$*-bit multiplier (from Sunder et al. [97, Fig. 1]).*
(a) *The multiplier array. The dotted line is the CP through the array.*
(b), (c), *and* (d)*: The* FA_1, HA_1, *and* FA_2 *cells, respectively.*

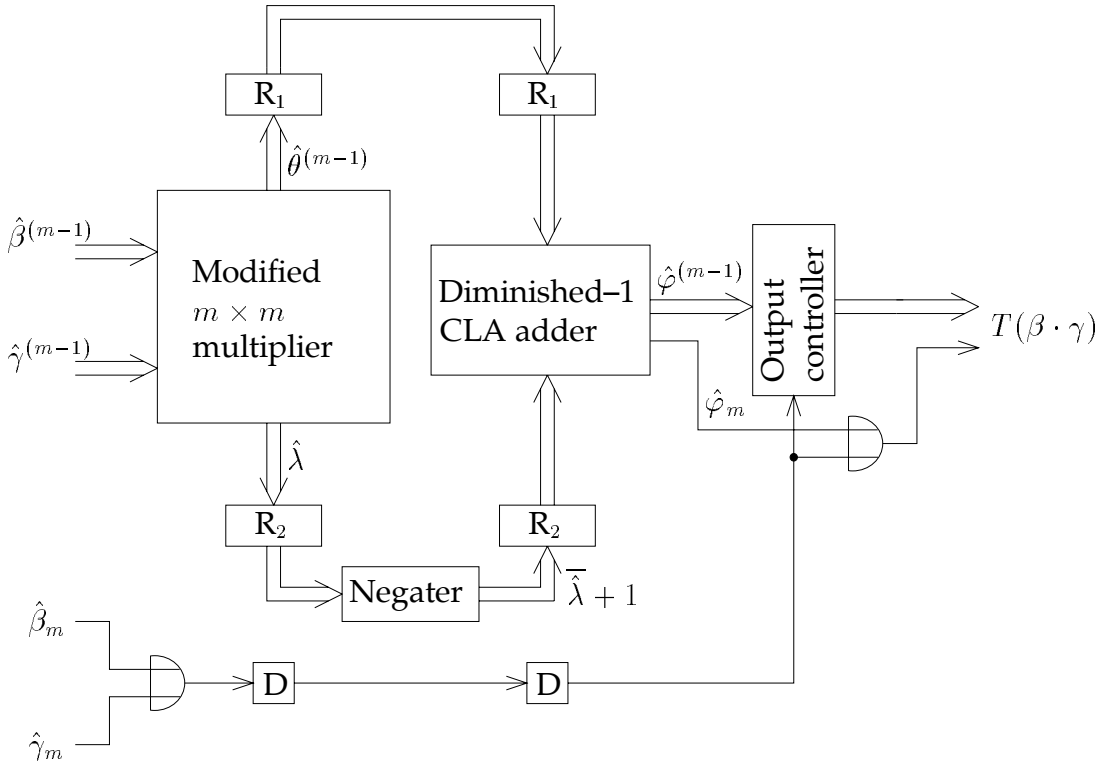


Figure 6.19: *A block diagram of a modified $(m \times m)$-bit diminished–1 multiplier over $\mathbb{Z}_{2^m+1}$ (essentially from Sunder et al. [97, Fig. 4]).*

one NOR gate. The Boolean function $\hat{\beta}_m + \hat{\gamma}_m$ is evaluated separately. Sunder assigns the value of this function to the most significant bit of the output $T(\beta \cdot \gamma)$. However, when the modulus $2^m + 1$ is composite, there exist products of nonzero integers of $\mathbb{Z}_{2^m+1}$ that are congruent to zero modulo $2^m + 1$. If such a situation occurs, the most significant bit of the product $T(\beta \cdot \gamma)$ should *not* be formed by the Boolean function $\hat{\beta}_m + \hat{\gamma}_m$. The correct Boolean function is $\hat{\beta}_m + \hat{\gamma}_m + \hat{\varphi}_m$, i.e. only an extra OR gate is needed to form the true output (see Figure 6.19).

Let $\mathcal{C}_{\text{Sunder,array}}$ denote the size of Sunder's $(m \times m)$-bit array multiplier of Figure 6.18. Then we have

$$\begin{aligned} \mathcal{C}_{\text{Sunder,array}} &= (m-1)^2 \mathcal{C}_{\text{FA2}} + (m-1)\mathcal{C}_{\text{HA1}} + m(\mathcal{C}_{\text{FA1}} + \mathcal{C}_{\text{FA}} + \mathcal{C}_{\text{HA}}) \\ &= 34m^2 + 36m + 10, \end{aligned}$$

where $\mathcal{C}_{\text{FA1}} = \mathcal{C}_{\text{FA}} + \mathcal{C}_{\text{AND}} = 34$, $\mathcal{C}_{\text{FA2}} = \mathcal{C}_{\text{FA1}} = 34$, $\mathcal{C}_{\text{HA1}} = \mathcal{C}_{\text{HA}} + \mathcal{C}_{\text{AND}} = 24$, $\mathcal{C}_{\text{FA}} = 28$, and $\mathcal{C}_{\text{HA}} = 18$ are the sizes of the FA_1, FA_2, HA_1, FA, and HA cells,

respectively. The registers labelled R_1 in Figure 6.19 are m-bit parallel registers and the ones labelled R_2 are $(m+1)$-bit parallel registers. The D cells are D flip-flops (single-bit registers). Hence, the total chip area A occupied by the bit-parallel multiplier in Figure 6.19 is proportional to its size

$$\begin{aligned}\mathcal{C}_{\text{Sunder,mult}} &= \mathcal{C}_{\text{Sunder,array}} + (2m + 2(m+1) + 2)\mathcal{C}_{\text{reg}} \\ &\quad + \mathcal{C}_{\text{dimneg}} + \mathcal{C}_{\text{dimadd,1}} + \mathcal{C}_{\text{out,ctrl}} + 2\mathcal{C}_{\text{OR}} \\ &= 34m^2 + 166m + 98,\end{aligned}$$

where $\mathcal{C}_{\text{out,ctrl}} = m(\mathcal{C}_{\text{NOR}} + \mathcal{C}_{\text{inv}}) = 6m$ is the size of the output controller. We assume that the diminished–1 adder in Figure 6.19 is the carry look-ahead adder of Figure 6.7.

The CP through Sunder's array multiplier is the dotted path in Figure 6.18 from the $\hat{\beta}_{m-1}$-input node to the $\hat{\theta}_{2m}$-output node. With the inputs taken directly from registers, the minimum clock cycle time of the complete multiplier in Figure 6.19 is proportional to the length

$$\begin{aligned}\mathcal{L}_{\text{CP,Sunder,array}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}}(mf_{\text{AND}} + f_{\text{FA,carry}}) + \mathcal{L}_{\text{AND}} + r_{\text{AND}} f_{\text{FA,signal}} \\ &\quad + m\mathcal{L}_{\text{FA,sum}} + (m-1)r_{\text{FA}} f_{\text{FA,carry}} + r_{\text{FA}} f_{\text{HA}} + \mathcal{L}_{\text{HA,carry}} \\ &\quad + r_{\text{HA,carry}} f_{\text{FA,carry}} + m\mathcal{L}_{\text{FA,carry}} + (m-1)r_{\text{FA}} f_{\text{FA,carry}} \\ &\quad + r_{\text{FA}} f_{\text{reg}} \\ &= 36m + 52.\end{aligned}$$

of this path.

Remark: The carry input fan-in $f_{\text{FA,carry}}$ of a full adder element (the one in Figure 4.10) is less than its signal input fan-in $f_{\text{FA,signal}}$. Therefore, in order to minimise the overall propagation delay, we assume that the sum output of the full adder element in each FA_2 cell is fed to the carry input of the full adder element in the subsequent FA_2 cell. Then the CP passes through the full adder element of each FA_2 cell from the carry input to the sum output. The smallest propagation delay through a FA_1 cell is obtained when the $\hat{\beta}_i$-input signal is connected to the carry input of the full adder element.

The desired product $T(\beta \cdot \gamma)$ is obtained in an output register after three clock cycles. Hence, the total multiplication time T is proportional to

$$\mathcal{L}_{\text{Sunder,mult}} = 3\mathcal{L}_{\text{CP,Sunder,array}} = 108m + 156,$$

which implies that the area-time performance AT^2 is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{Sunder,mult}} &\triangleq \mathcal{C}_{\text{Sunder,mult}}(\mathcal{L}_{\text{Sunder,mult}})^2 \\ &= (34m^2 + 116m + 98)(108m + 156)^2 = \mathcal{O}\left(m^4\right).\end{aligned}$$

Ashur's Parallel Multiplier

Quite recently, Ashur et al. [10] presented an architecture for bit-parallel diminished–1 multiplication that is based on Sunder's architecture in Figure 6.19. They obtain a smaller area-time performance for their architecture inter alia by including the negation step in the array multiplier. Below, we analytically describe how Ashur's algorithm works: For nonzero NBC factors β and γ, i.e for the diminished–1 numbers $0 \leq \hat{\beta},\ \hat{\gamma} \leq 2^m - 1$, the general multiplication $T(\beta \cdot \gamma) \equiv \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} \pmod{2^m + 1}$ can be expanded as

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} + 2^m(2^m - 1) - 2^m(2^m - 1) \\ &= 2^m \sum_{i=0}^{m} \hat{\vartheta}_{m+i} 2^i + \hat{\vartheta}^{(m-1)} - 2^m(2^m - 1) \\ &\equiv -\hat{\delta} + \hat{\vartheta}^{(m-1)} - 2 \pmod{2^m + 1}, \end{aligned} \tag{6.32}$$

where

$$\hat{\vartheta} = \hat{\beta}^{(m-1)}\hat{\gamma}^{(m-1)} + \hat{\gamma}^{(m-1)} + 2^m(2^m - 1) \tag{6.33}$$

is a $(2m+1)$-bit NBC integer and $\hat{\delta} = \sum_{i=0}^{m-1} \hat{\theta}_{m+i} 2^i$ is an $(m+1)$-bit NBC integer. Again, by (5.5) we get $-\hat{\delta} \equiv \overline{\hat{\delta}} + 3 \pmod{2^m + 1}$. Using this congruence and the congruence $2^m(2^m - 1) \equiv -(-1 - 1) = 2 \pmod{2^m + 1}$, (6.32) can be written as

$$T(\beta \cdot \gamma) \equiv \hat{\vartheta}^{(m-1)} + \overline{\hat{\delta}} + 1. \tag{6.34}$$

Figure 6.20 shows a block diagram of Ashur's diminished–1 multiplier. Ashur et al. have modified Sunder's array multiplier (the one in Figure 6.18) in the following way: The rightmost column of half adder elements, which performs an addition by one, is excluded. Instead, the addition by $2^m(2^m - 1)$ in (6.33) is carried out by exchanging the half adder elements in the leftmost column for full adder elements (i.e. exchange the HA_1 cells for FA_2 cells) and let each redundant full adder input be equal to one. The so far grounded input of the leftmost full adder element in the bottom row of Sunder's array multiplier should now also be equal to one.

Furthermore, the resulting m full adder elements in the bottom row of the array forms an m-bit carry ripple adder. The output of this adder is the $(m+1)$-bit integer $\hat{\delta}$, which is defined above. The output $\hat{\delta}$ can be formed by the sum $\hat{\delta} = \hat{\mu} + \hat{\eta}$, where in turn the m-bit integer $\hat{\mu} = (\hat{\mu}_{m-1},\ \hat{\mu}_{m-2},\ \hat{\mu}_{m-3}, \ldots,\ \hat{\mu}_0)_2$ is formed by the carry outputs and the m-bit integer $\hat{\eta} = (\hat{\eta}_m,\ \hat{\eta}_{m-1},\ \hat{\eta}_{m-2}, \ldots,\ \hat{\eta}_1)_2$ is formed by the sum outputs of the row of full adder elements prior to the carry ripple adder (see Figure 6.20). Hence, (6.34) can be further expanded as

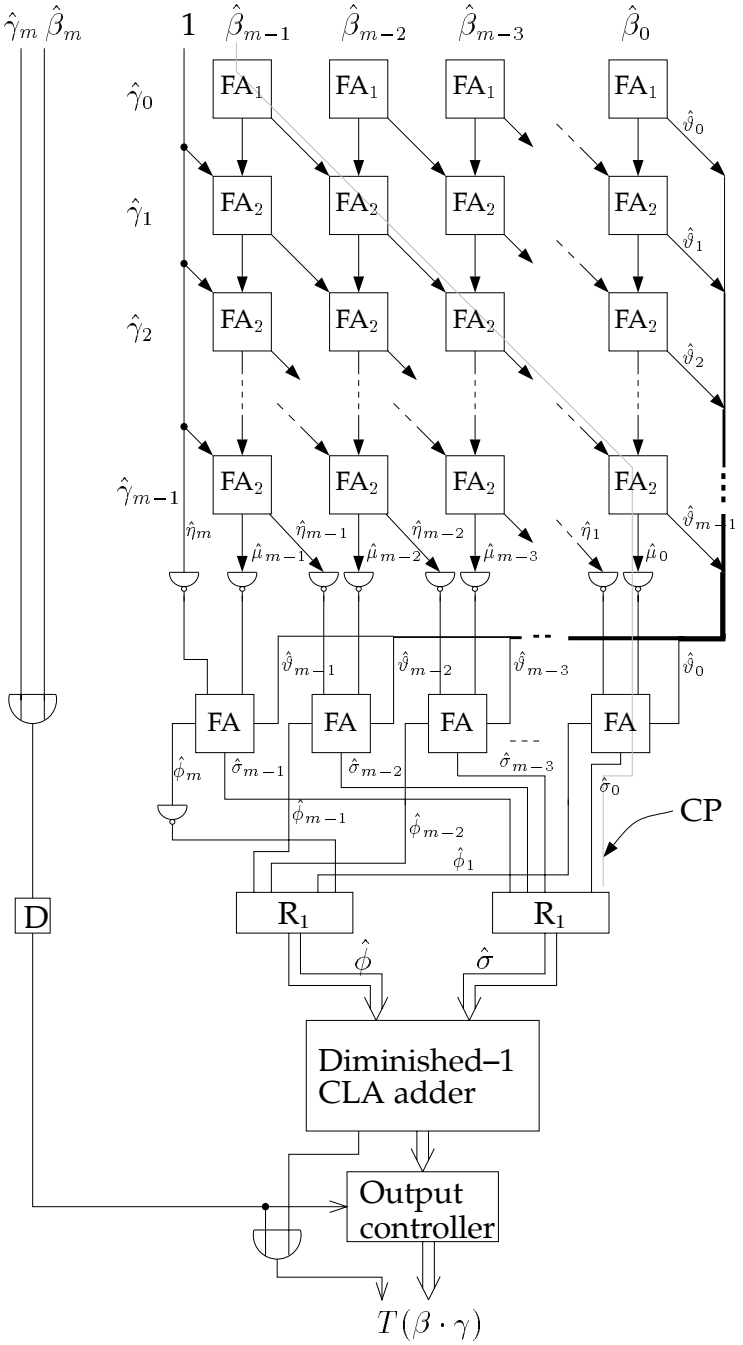


Figure 6.20: *A block diagram of a modified $m \times m$ diminished–1 multiplier over $\mathbb{Z}_{2^m+1}$, based on Sunder's multiplier (from Ashur et al. [10, Fig. 1]). The dotted line is the CP through the array multiplier.*

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \hat{\vartheta}^{(m-1)} + \overline{\hat{\mu} + \hat{\eta}} + 1 \\ &= \hat{\vartheta}^{(m-1)} + (2^{m+1} - 1) - (\hat{\mu} + \hat{\eta}) + 1 \\ &= \hat{\vartheta}^{(m-1)} + (2^m - 1) - \hat{\mu} + (2^m - 1) - \hat{\eta} + 2 \\ &= \hat{\vartheta}^{(m-1)} + \overline{\hat{\mu}} + \overline{\hat{\eta}} + 2 \pmod{2^m + 1}. \end{aligned} \tag{6.35}$$

The addends $\overline{\hat{\mu}}$ and $\overline{\hat{\eta}}$ are formed by the row of inverters below the array multiplier in Figure 6.20. *Carry-save adders* are preferably used when more than two numbers are to be added together. For example, array multipliers (like the ones described in the present section) generally comprise rows of carry-save adders that perform the summation of the partial products. The final addition is performed using a carry ripple or carry look-ahead adder.

Ashur et al. efficiently adds the m-bit addends $\hat{\vartheta}^{(m-1)}$, $\overline{\hat{\mu}}$, and $\overline{\hat{\eta}}$ in (6.35) by using a carry save adder. These three addends are the inputs of the carry-save adder which is subsequent to the row of inverters in Figure 6.20. Let $\hat{\rho} = \hat{\vartheta}^{(m-1)} + \overline{\hat{\mu}} + \overline{\hat{\eta}} = \hat{c} + \hat{\sigma}$, where the $(m+1)$-bit integer $\hat{c} = \sum_{i=1}^{m} \hat{c}_i 2^i \equiv \sum_{i=1}^{m-1} \hat{c}_i 2^i - \hat{c}_m \pmod{2^m + 1}$ and the m-bit integer $\hat{\sigma} = \sum_{i=0}^{m-1} \hat{\sigma}_i 2^i$ are formed by the carry outputs $\hat{c}_i$ and sum outputs $\hat{\sigma}_i$, respectively, of the carry-save adder. Hence, (6.35) can be written as the diminished–1 sum

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \hat{\rho} + 2 \pmod{2^m + 1} = \hat{c} + \hat{\sigma} + 2 \\ &\equiv \sum_{i=1}^{m-1} \hat{c}_i 2^i + (1 - \hat{c}_m) + \hat{\sigma} + 1 \\ &\equiv \hat{\phi} \oplus \hat{\sigma} \pmod{2^m + 1}, \end{aligned} \tag{6.36}$$

where $\hat{\phi} = \sum_{i=1}^{m-1} \hat{c}_i 2^i + \overline{\hat{c}_m}$ is an m bit integer, i.e. we have

$$\hat{\phi} = \sum_{i=0}^{m-1} \hat{\phi}_i 2^i, \text{ where } \begin{cases} \hat{\phi}_0 = \overline{\hat{c}_m}; \\ \hat{\phi}_i = \hat{c}_i; \qquad \text{for } 1 \le i \le m-1 \end{cases}$$

The addition by $\overline{\hat{c}_m}$ is thus carried out by inverting the most significant carry output $\hat{c}_m$ of the carry-save adder and feeding it into the vacant least significant bit position of the register that holds $\hat{\phi}$. For consistency, we have introduced our notations for Ashur's multiplier in Figure 6.20. As for Sunder's multiplier in Figure 6.19, we have modified the output controller in order to obtain the correct output when $\beta, \gamma \mid (2^m + 1)$ and $T(\beta \cdot \gamma) \equiv 2^m \pmod{2^m + 1}$ (i.e. when $\beta\gamma \equiv 0 \pmod{2^m + 1}$).

The chip area A occupied by Ashur's multiplier is proportional to its size

$$\begin{aligned} \mathcal{C}_{\text{Ashur,mult}} &= m(\mathcal{C}_{\text{FA1}} + (m-1)\mathcal{C}_{\text{FA2}} + 2\mathcal{C}_{\text{inv}} + \mathcal{C}_{\text{FA}}) + (2m+1)\mathcal{C}_{\text{reg}} \\ &\quad + \mathcal{C}_{\text{dimadd,1}} + \mathcal{C}_{\text{out,ctrl}} + 2\mathcal{C}_{\text{OR}} + \mathcal{C}_{\text{inv}} \\ &= 34m^2 + 94m + 26 \end{aligned}$$

The CP through the array multiplier (see the dotted line in Figure 6.20) determines the maximum clock frequency. The maximum clock frequency is inversely proportional to the CP length[13]

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,Ashur,array}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}}(m f_{\mathrm{AND}} + f_{\mathrm{FA,carry}}) + \mathcal{L}_{\mathrm{AND}} + r_{\mathrm{AND}} f_{\mathrm{FA,signal}} \\ &\quad + (m-1)(\mathcal{L}_{\mathrm{FA,sum}} + r_{\mathrm{FA}} f_{\mathrm{FA,carry}}) + \mathcal{L}_{\mathrm{FA,carry}} \\ &\quad + r_{\mathrm{FA}} f_{\mathrm{inv}} + r_{\mathrm{inv}} f_{\mathrm{FA,signal}} + \mathcal{L}_{\mathrm{FA,sum}} + r_{\mathrm{FA}} f_{\mathrm{reg}} \\ &= 22m + 60\end{aligned}$$

Because Ashur's multiplier computes the product $T(\beta \cdot \gamma)$ in only two clock cycles, the total computation time T is proportional to

$$\mathcal{L}_{\mathrm{Ashur,mult}} = 2\mathcal{L}_{\mathrm{CP,Ashur,array}} = 44m + 120$$

and the AT^2 performance is proportional to the product

$$\begin{aligned}\mathcal{C}\mathcal{L}^2_{\mathrm{Ashur,mult}} &\triangleq \mathcal{C}_{\mathrm{Ashur,mult}}(\mathcal{L}_{\mathrm{Ashur,mult}})^2 \\ &= (34m^2 + 94m + 26)(44m + 120)^2 = \mathcal{O}\left(m^4\right).\end{aligned}$$

Benaissa's Parallel Multiplier

Regarding the second of Leibowitz' multiplication procedures (see page 130), the diminished–1 product $T(\beta \cdot \gamma)$ can be written as

$$\begin{aligned}T(\beta \cdot \gamma) &\equiv \beta\gamma - 1 \\ &= 2^m \sum_{i=0}^{m} \theta_{m+i} 2^i + \theta^{(m-1)} - 1 \\ &\equiv \theta^{(m-1)} - \lambda - 1 \pmod{2^m + 1},\end{aligned}$$

where $\theta = \beta\gamma$ is a $(2m+1)$-bit NBC integer and $\lambda = \sum_{i=0}^{m} \theta_{m+i} 2^i$ is an $(m+1)$-bit NBC integer.[14] By (5.5) we get $-\lambda \equiv \overline{\lambda} + 3 \pmod{2^m + 1}$ and therefore $T(\beta \cdot \gamma)$ can be formed by the diminished–1 sum

$$\begin{aligned}T(\beta \cdot \gamma) &\equiv \theta^{(m-1)} + (\overline{\lambda} + 1) + 1 \\ &= \theta^{(m-1)} \oplus (\overline{\lambda} + 1) \pmod{2^m + 1},\end{aligned}$$

where $\overline{\lambda} + 1$ equals diminished–1 negation of λ.

Benaissa et al. [11] have implemented a bit-parallel multiplier which is based on this multiplication procedure. Figure 6.21 shows a block diagram of their

[13] The CP starts with the output path of a register.

[14] Leibowitz [58], however, erroneously stated that λ is an m-bit NBC integer. The procedure described in his article gives an incorrect answer for $\beta = \gamma = 2^m$.

pipelined multiplier. The translation blocks translate the inputs $\hat{\beta}$ and $\hat{\gamma}$ to β and γ, respectively, which are multiplied in the array multiplier. The realisation of the translation blocks are shown in Figure 6.4 of Section 6.3.1. Benaissa et al. use a standard $(m+1) \times (m+1)$-bit square-version array multiplier, see Benaissa et al. [11, Fig. 6] or Weste and Eshraghian [113, Fig. 8.37], which comprises m^2-1 full adder elements, $m+1$ half adder elements, and $(m+1)^2$ AND gates. Hence, the size of the this array multiplier equals

$$\begin{aligned} \mathcal{C}_{\text{Benaissa,array}} &= (m^2-1)\mathcal{C}_{\text{FA}} + (m+1)\mathcal{C}_{\text{HA}} + (m+1)^2\mathcal{C}_{\text{AND}} \\ &= 34m^2 + 30m - 4, \end{aligned}$$

which is slightly less than the size $\mathcal{C}_{\text{Sunder,array}}$ of Sunder's array multiplier in Figure 6.18. Using the same types of m-bit parallel registers R_1 and $(m+1)$-bit registers R_2 and the same type of carry look-ahead adder as in Figure 6.19, the size of the complete multiplier of Figure 6.21 equals

$$\begin{aligned} \mathcal{C}_{\text{Benaissa,p-mult}} &= \mathcal{C}_{\text{Benaissa,array}} + (2m + 4(m+1))\mathcal{C}_{\text{reg}} \\ &\quad + \mathcal{C}_{\text{dimneg}} + \mathcal{C}_{\text{dimadd,1}} + 2\mathcal{C}_{\text{Dim2NBC}} + 2\mathcal{C}_{\text{OR}} \\ &= 34m^2 + 222m + 88. \end{aligned}$$

The CP of the array multiplier is similar to the CP of Sunder's array multiplier in Figure 6.18. It runs from a register output into the AND gate in the top-leftmost position of the array and then diagonally through the array of full adder elements and finally to the left along the bottom carry-chain row to the register holding the most significant output bit θ_{2m}. The length of this CP equals

$$\begin{aligned} \mathcal{L}_{\text{CP,Benaissa,array}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} \cdot (m+1) f_{\text{AND}} + \mathcal{L}_{\text{AND}} + r_{\text{AND}} f_{\text{HA}} \\ &\quad + \mathcal{L}_{\text{HA,sum}} + r_{\text{HA,sum}} f_{\text{FA,carry}} + (m-1)\mathcal{L}_{\text{FA,sum}} \\ &\quad + (m-2) r_{\text{FA}} f_{\text{FA,carry}} + r_{\text{FA}} f_{\text{HA}} + \mathcal{L}_{\text{HA,carry}} \\ &\quad + r_{\text{HA,carry}} f_{\text{FA,carry}} + (m-2)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}} f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}} f_{\text{reg}} \\ &= 36m + 28. \end{aligned}$$

The product $T(\beta \cdot \gamma)$ is obtained in the output register subsequent to the diminished–1 adder after four clock cycles. Hence, the time T required to multiply using Benaissa's array multiplier architecture is proportional to

$$\mathcal{L}_{\text{Benaissa,p-mult}} = 4\mathcal{L}_{\text{CP,Benaissa,array}} = 144m + 112,$$

which means that the AT^2 performance of the multiplier is proportional to the product

$$\begin{aligned} \mathcal{CL}^2_{\text{Benaissa,p-mult}} &\triangleq \mathcal{C}_{\text{Benaissa,p-mult}}(\mathcal{L}_{\text{Benaissa,p-mult}})^2 \\ &= (34m^2 + 222m + 88)(144m + 112)^2 = \mathcal{O}\left(m^4\right). \end{aligned}$$

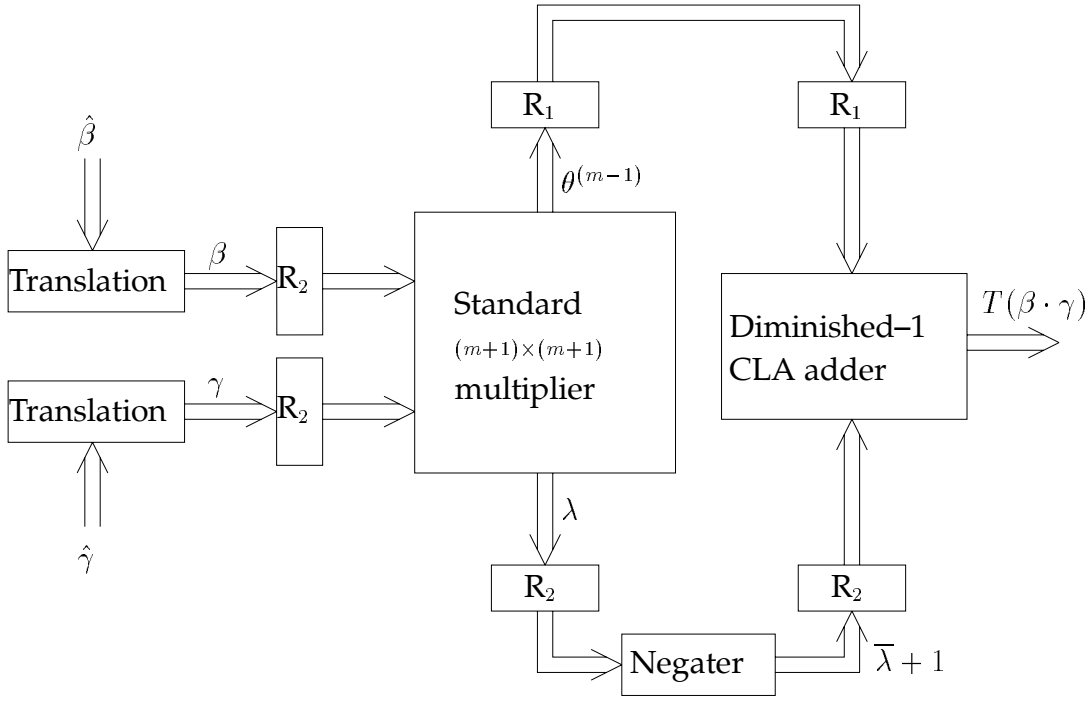


Figure 6.21: *A block diagram of Benaissa's [11, Fig. 4] diminished–1 pipelined array multiplier.*

Remark: If the multiplier (or the multiplicand) is available as an NBC number β (or γ), one of the translation circuits in Figure 6.21 can be excluded. This reduces the total size $\mathcal{C}_{\text{Benaissa,p-mult}}$. If *both* the multiplier and the multiplicand are NBC numbers, the translation part and the two input registers (R_2) can be excluded. Consequently, the initial clock cycle is then excluded. This reduces the total computation time as well as the total circuit size. Note also that a simple additional modification of the multiplier (in Figure 6.21), makes it applicable for general multiplication with respect to the NBC symbol representation.

6.3.6.2 Bit-Serial/Parallel Multipliers

Probably the most frequently used diminished–1 multiplier is the bit-serial/parallel multiplier.[15] In general, serial/parallel multipliers are known to oc-

[15]The multiplication scheme adopted is often referred to as the iterative shift-and-add technique.

cupy less chip area than the corresponding parallel multipliers, but to the cost of a poorer time performance.

Several algorithms for serial/parallel diminished–1 multiplication have appeared in the literature. They mainly differ in how the multiplicand and the multiplier are represented and which initial values have to be computed. The registers needed in the corresponding architectures are loaded with the initially computed values.

Chang's Serial/Parallel Multiplier

Chang et al. [32] were among the first to publish a VLSI implementation of a serial/parallel diminished–1 general multiplier. It is based on a diminished–1 representation of the multiplicand and an NBC representation of the multiplier. Let $\hat{\beta}$ and $\hat{\gamma}$ be the multiplicand and the multiplier, respectively, in their diminished–1 form of representation. The multiplication algorithm is valid only for $\beta,\ \gamma \neq 0$, i.e. for $\hat{\beta},\ \hat{\gamma} \neq 2^m$. Situations where either β or γ (or both) equals zero are handled separately. The algorithm of Chang et al. is based on the following expansion of $T(\beta \cdot \gamma)$ (here, we use our notations):

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \beta\gamma - 1 \equiv \beta \sum_{i=0}^{m} \gamma_i 2^i - 1 \\ &= \sum_{i=0}^{m} \gamma_i (2^i \beta - 1) + \sum_{i=0}^{m} \gamma_i \ - 1 + m - m \\ &\equiv \left(\sum_{i=0}^{m} \gamma_i T(2^i \beta) + m \right) + (2^m - 1 - D) + 1 \\ &= \left(\bigoplus_{i=0}^{m} \gamma_i T(2^i \beta) \right) \oplus \overline{D} \quad \pmod{2^m + 1}, \end{aligned} \tag{6.37}$$

where $D \triangleq m - \sum_{i=0}^{m} \gamma_i$ and where $\bigoplus$ and $\oplus$ denote diminished–1 addition. Because $\gamma \in \mathbb{Z}_{2^m+1}$ we get $D \in \mathbb{Z}_{m+1}$, which is represented as an m-bit NBC integer.

Chang et al. [32, Fig. 1] presented a simple architecture which computes (6.37) using an recursive shift-and-add technique, where the modulus reduction is simultaneously carried out during each recursion. Thus, (6.37) can be expressed on the recursive form

$$P(i+1) \equiv P(i) \oplus \gamma_i T(2^i \beta) \pmod{2^m + 1}; \qquad \text{for } 0 \leq i \leq m,$$

where $P(0) = \overline{D}$. For $i = m$ we then get $P(m+1) = T(\beta \cdot \gamma)$. Chang et al. present a slightly modified algorithm to compute the desired product

$T(\beta\cdot\gamma)$ in $m+2$ clock cycles. The algorithm, however, suffers principally from two drawbacks. Firstly, the multiplier γ needs to be translated from its diminished–1 representation to its NBC representation. Secondly, an initial computation of $\overline{D}$ must be performed before the shift-and-add procedure can begin.

A simplified block diagram of a general diminished–1 general multiplier, based on Chang's multiplier and an MC68000 microprocessor, can be found in Shakaff's PhD. thesis [89, Fig. 3.21(a)]. Shakaff concludes that the main disadvantage of the above multiplication procedure is the need to compute the initial value $\overline{D}$. He instead proposes the multiplier by Benaissa et al. [12] as a competitive alternative. Benaissa's multiplier, which needs no precomputations, is presented below.

Benaissa's Serial/Parallel Multipliers

For the diminished–1 representation, the parameters k and l in (6.1) are equal to 1 and -1, respectively. Hence, the diminished–1 form of the general multiplication formula in (6.13) is

$$T(\beta\cdot\gamma)\equiv\sum_{i=0}^{m}(\gamma_i T(2^i\beta)-\overline{\gamma_i})\equiv\sum_{i=0}^{m}(\gamma_i T(2^i\beta)+\overline{\gamma_i}2^m)\pmod{2^m+1}. \quad (6.38)$$

The multiplication algorithm suggested by Benaissa et al. [12] is based on this congruence. In their formulation of $T(\beta\cdot\gamma)$ they have omitted the term $\overline{\gamma_i}2^m$. They express the product as $\sum\gamma_i T(2^i\beta)$ (see [12, Eq. (6)]) and only mention that for $\gamma_i=0$, the addend is set equal to the diminished–1 zero (i.e. the integer 2^m). The correct expression for $T(\beta\cdot\gamma)$, however, is given in (6.38). The congruence (6.38) can be expressed on the recursive form

$$P(i+1)\equiv P(i)\oplus(\gamma_i T(2^i\beta)+\overline{\gamma_i}2^m)\pmod{2^m+1};\qquad \text{for } 0\le i\le m,$$

where the initial value $P(0)$ equals 2^m (diminished–1 zero). Moreover, for $i=m$ we have $P(m+1)=T(\beta\cdot\gamma)$.

A block diagram of Benaissa's multiplier is shown in Figure 6.22. The control signals are not shown in the figure. The registers R_1, R_2, and R_3 are all $m+1$ bits wide.

During an initial clock cycle, $\hat{\beta}$ is loaded into register R_1 and the translated integer $\gamma\equiv\hat{\gamma}+1\pmod{2^m+1}$ is loaded into register R_2. Also, the $(m+1)$-bit integer 2^m is loaded into R_3. After the subsequent clock cycles, R_1 contains $T(2\beta)$, $T(2^2\beta)$, $T(2^3\beta)$, etc. We assume that the output $T(\beta\cdot\gamma)$ is directly stored in an $(m+1)$-bit parallel register. The CP is the dotted path in Figure 6.22, from the output of the shift register R_2 through on an AND gate and

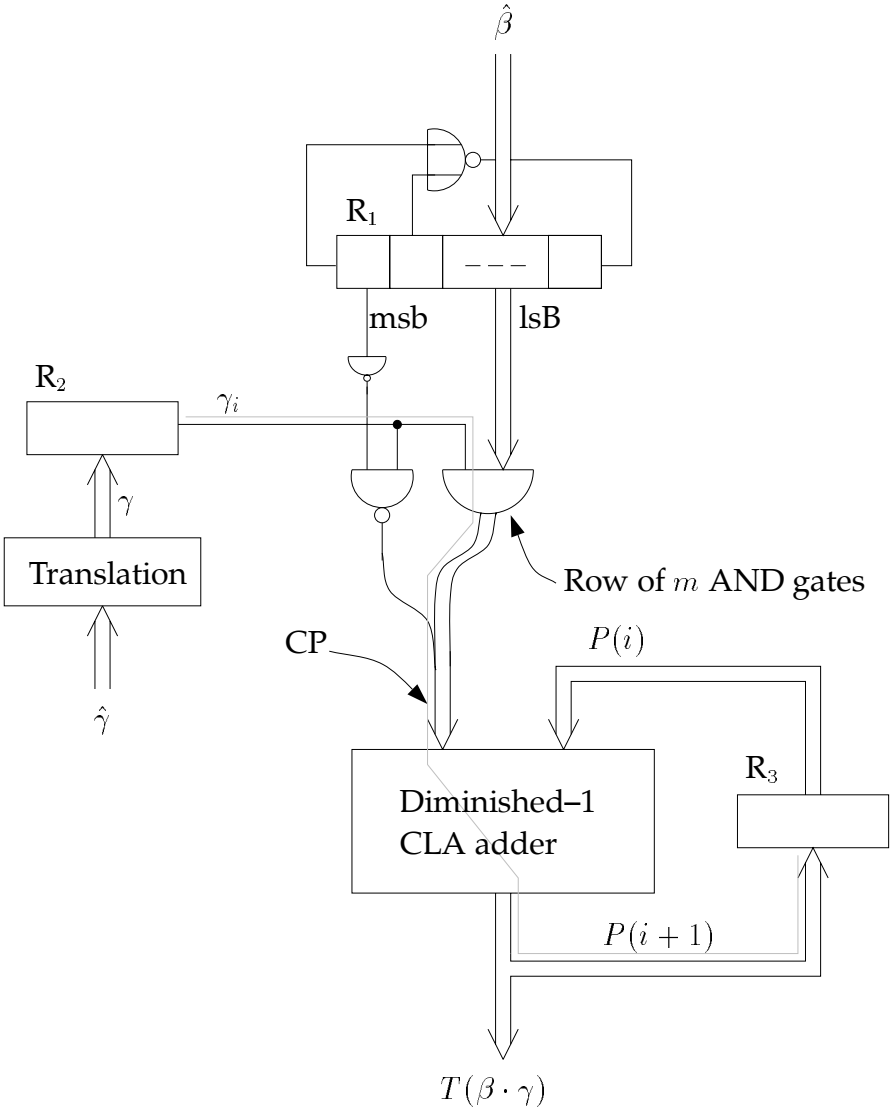


Figure 6.22: *A block diagram of Benaissa's diminished–1 serial/parallel multiplier (essentially from Benaissa et al. [12, Fig. 1]).*

the carry look-ahead adder to the input of the parallel register R_3. The length $\mathcal{L}_{\mathrm{CP,Benaissa,s/p-mult}}$ of this path equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,Benaissa,s/p-mult}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} \cdot (m+1) f_{\mathrm{AND}} + \mathcal{L}_{\mathrm{AND}} + r_{\mathrm{AND}} f_{\mathrm{dimadd,1}} \\ &\quad + \mathcal{L}_{\mathrm{dimadd,1}} + r_{\mathrm{dimadd,1}} \cdot 2 f_{\mathrm{reg}} \\ &= 18m + 8 \log_2 m + 120.\end{aligned}$$

The initial clock cycle is followed by $m+1$ clock cycles, during which the partial products are computed and recursively added together. Hence, the total computation time T is proportional to

$$\begin{aligned}\mathcal{L}_{\mathrm{Benaissa,s/p-mult}} &= (m+2)\mathcal{L}_{\mathrm{CP,Benaissa,s/p-mult}} \\ &= 18m^2 + 8m \log_2 m + 156m + 16 \log_2 m + 240.\end{aligned}$$

The desired product $P(i+1) = T(\beta \cdot \gamma)$ is shifted into an output parallel register during the final clock cycle. In order to make a fair comparison with the parallel diminished–1 multipliers described above, we again assume that the diminished–1 adder in Figure 6.22 is the carry look-ahead adder of Figure 6.7. Then, the chip area A occupied by the multiplier in Figure 6.22 is proportional to its size

$$\begin{aligned}\mathcal{C}_{\mathrm{Benaissa,s/p-mult}} &= 3(m+1)\mathcal{C}_{\mathrm{reg}} + \mathcal{C}_{\mathrm{Dim2NBC}} + m\mathcal{C}_{\mathrm{AND}} \\ &\quad + 2\mathcal{C}_{\mathrm{NAND/NOR}} + \mathcal{C}_{\mathrm{inv}} + \mathcal{C}_{\mathrm{dimadd,1}} \\ &= 128m + 72.\end{aligned}$$

Hence, the area-time performance AT^2 of the multiplier is proportional to

$$\begin{aligned}\mathcal{C}\mathcal{L}^2_{\mathrm{Benaissa,s/p-mult}} &\triangleq \mathcal{C}_{\mathrm{Benaissa,s/p-mult}} (\mathcal{L}_{\mathrm{Benaissa,s/p-mult}})^2 \\ &= (128m + 72) \\ &\quad \cdot (18m^2 + 8m \log_2 m + 156m + 16 \log_2 m + 240)^2 \\ &= \mathcal{O}\left(m^5\right).\end{aligned}$$

If the multiplier in the multiplication operation is available as an NBC number γ, the translation circuit in Figure 6.22 can be excluded. This reduces the total size of the multiplier architecture, but the computation time is not changed.

In their paper, Benaissa et al. [12, Ch. 3.2] also describes a procedure for diminished–1 multiplication which is a slight modification of the above procedure. The procedure is based on (6.38), but it uses $\hat{\gamma}$ as multiplier instead of γ. This eliminates the need for the code translation from $\hat{\gamma}$ to $\gamma \equiv \hat{\gamma} + 1 \pmod{2^m + 1}$. For *nonzero* γ we can write $(\gamma_m,\ \gamma_{m-1},\ \gamma_{m-2}, \ldots,\ \gamma_2,\ \gamma_1,\ \gamma_0)_2 = (0,\ \hat{\gamma}_{m-1},\ \hat{\gamma}_{m-2},$

$\ldots, \hat{\gamma}_2, \hat{\gamma}_1, \hat{\gamma}_0 + 1)_2$. Benaissa et al. state that γ can be replaced by $\hat{\gamma}$ in (6.38) by letting the least significant bit of the multiplier take on the value $\hat{\gamma}_0 + 1$. Actually, this can easily be analytically formulated by expanding (6.38) in the following way:

$$\begin{aligned} T(\beta \cdot \gamma) &\equiv \sum_{i=0}^{m} (\gamma_i T(2^i \beta) + \overline{\gamma_i} 2^m) \\ &= \sum_{i=1}^{m-1} (\hat{\gamma}_i T(2^i \beta) + \overline{\hat{\gamma}_i} 2^m) \quad + (\hat{\gamma}_0 + 1) T(\beta) + (1 - (\hat{\gamma}_0 + 1)) 2^m + 1 \\ &= \sum_{i=0}^{m-1} (\hat{\gamma}_i T(2^i \beta) + \overline{\hat{\gamma}_i} 2^m) \quad + T(\beta) + 1 \\ &= \sum_{i=0}^{m-1} (\hat{\gamma}_i T(2^i \beta) + \overline{\hat{\gamma}_i} 2^m) \oplus T(\beta) \qquad (\mathrm{mod}\ 2^m + 1), \end{aligned} \tag{6.39}$$

which can be expressed on the recursive form

$$P(i+1) \equiv P(i) \oplus (\hat{\gamma}_i T(2^i \beta) + \overline{\hat{\gamma}_i} 2^m) \pmod{2^m + 1}; \qquad \text{for } 0 \leq i \leq m-1,$$

where $P(0) = T(\beta) = \hat{\beta}$. We thus have $P(m) = T(\beta \cdot \gamma)$. Figure 6.23 shows the modified multiplier by Benaissa et al. It is a modified version of the multiplier in Figure 6.22. During an initial clock cycle, $\hat{\gamma}$ is loaded into register R_2 and $\hat{\beta}$ is loaded into both register R_1 and R_3. After the subsequent m clock cycles, register R_3 will contain the product $P(m) = T(\beta \cdot \gamma)$. An additional clock pulse shifts the product to an output register. If $\beta = 0$ ($\hat{\beta}_m = 1$) or $\gamma = 0$ ($\hat{\gamma}_m = 1$), the output controller sets the correct output $T(\beta \cdot \gamma) = 2^m$ (see page 131)[16].

The chip area A occupied by Benaissa's modified multiplier is proportional to its size

$$\begin{aligned} \mathcal{C}_{\mathrm{Benaissa,s/p-mult,2}} &= 3(m+1)\mathcal{C}_{\mathrm{reg}} + m\mathcal{C}_{\mathrm{AND}} + 2\mathcal{C}_{\mathrm{NAND/NOR}} + 2\mathcal{C}_{\mathrm{OR}} \\ &\quad + \mathcal{C}_{\mathrm{inv}} + \mathcal{C}_{\mathrm{dimadd,1}} + \mathcal{C}_{\mathrm{out,ctrl}} \\ &= 116m + 82. \end{aligned}$$

The CP of the multiplier is marked by the dotted line in Figure 6.23. It only slightly differs from the CP of the multiplier in Figure 6.22. The length of the CP equals

$$\begin{aligned} \mathcal{L}_{\mathrm{CP,Benaissa,s/p-mult,2}} &= \mathcal{L}_{\mathrm{CP,Benaissa,s/p-mult}} - r_{\mathrm{dimadd,1}} f_{\mathrm{reg}} \\ &= 18m + 8\log_2 m + 116. \end{aligned}$$

[16] Benaissa et al. [12, Fig. 3] use a row of AND gates instead of a row of inverters and NOR gates.

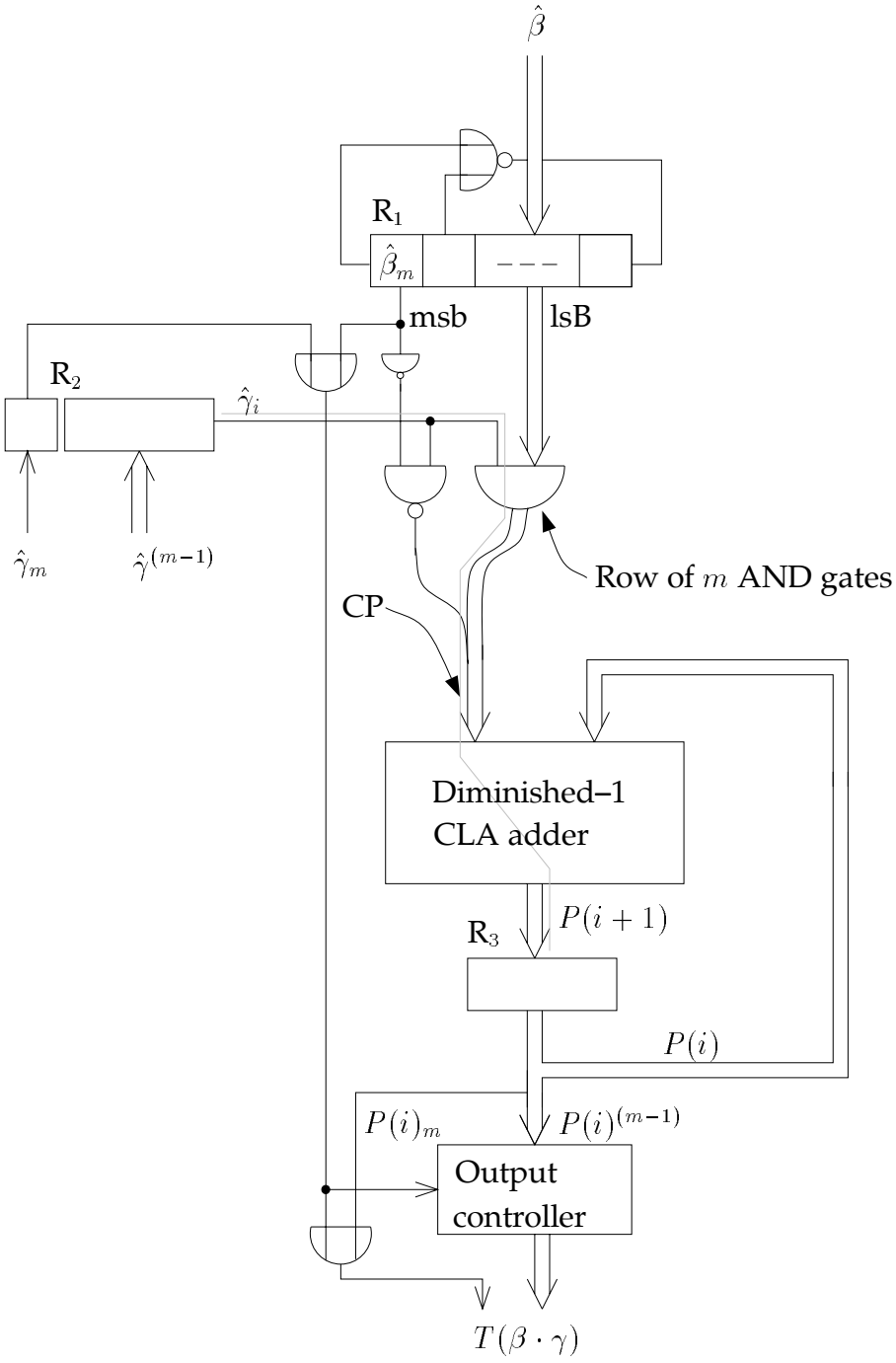


Figure 6.23: *A block diagram of Benaissa's [12, Fig. 3] modified diminished–1 serial/parallel multiplier.*

As described above, the modified multiplier needs $m+2$ clock cycles to compute a product, i.e. the same number of clock cycles as was required for the non-modified multiplier. Hence, the computation time T is proportional to

$$\begin{aligned}\mathcal{L}_{\text{Benaissa,s/p-mult,2}} &= (m+2)\mathcal{L}_{\text{CP,Benaissa,s/p-mult,2}} \\ &= 18m^2 + 8m\log_2 m + 152m + 16\log_2 m + 232\end{aligned}$$

and the product AT^2 is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{Benaissa,s/p-mult,2}} &\triangleq \mathcal{C}_{\text{Benaissa,s/p-mult,2}}(\mathcal{L}_{\text{Benaissa,s/p-mult,2}})^2 \\ &= (116m+82) \\ &\quad \cdot(18m^2 + 8m\log_2 m + 152m + 16\log_2 m + 232)^2 \\ &= \mathcal{O}\left(m^5\right).\end{aligned}$$

Shyu's Serial/Parallel Multiplier

The final serial/parallel diminished–1 multiplier to be considered here is the one suggested by Shyu et al. [92]. Theorem 1 in their paper says that diminished–1 multiplication can be calculated in $\mathbb{Z}_{2^m+1}$ as follows:[17]

$$T(\beta\cdot\gamma) \equiv \left(\sum_{i=0}^{m} T(\hat{\gamma}_i 2^i \beta)\right) \oplus \hat{\beta} \pmod{2^m+1}. \tag{6.40}$$

This congruence can be written on the recursive form

$$P(i+1) \equiv P(i) \oplus T(\hat{\gamma}_i 2^i \beta) \pmod{2^m+1}; \qquad \text{for } 0 \le i \le m,$$

where $P(0) = \hat{\beta}$ and for which we have $P(m+1) = T(\beta\cdot\gamma)$.

Note that this equation can also be derived from the more general expression in (6.11) by letting $k = -l$, which is the case for the diminished–1 representation ($(k,\, l) = (1,\, -1)$). Because(6.39) and (6.40) are quite similar, the two associated architectures have about the same structure. Figure 6.24 shows a modified version of Shyu's [92, Fig. 1] multiplier. It is based on the serial/parallel multiplier proposed by Chang et al. [32]. In Figure 6.24, we have exchanged most of Shyu's nMOS pass transistors for transmission gates. We have also modified their multiplier such that the output product is correct also when either of the diminished–1 operands (or both) are equal to 2^m. Such a situation is handled inter alia by the "output controller" circuit.

The multiplication algorithm works as follows: During an initial clock pulse, the $(m+1)$-bit registers A and D are loaded with $\hat{\beta}$ and $\hat{\beta}^{(m-1)}$, respectively,

[17] Here, we use our notations.

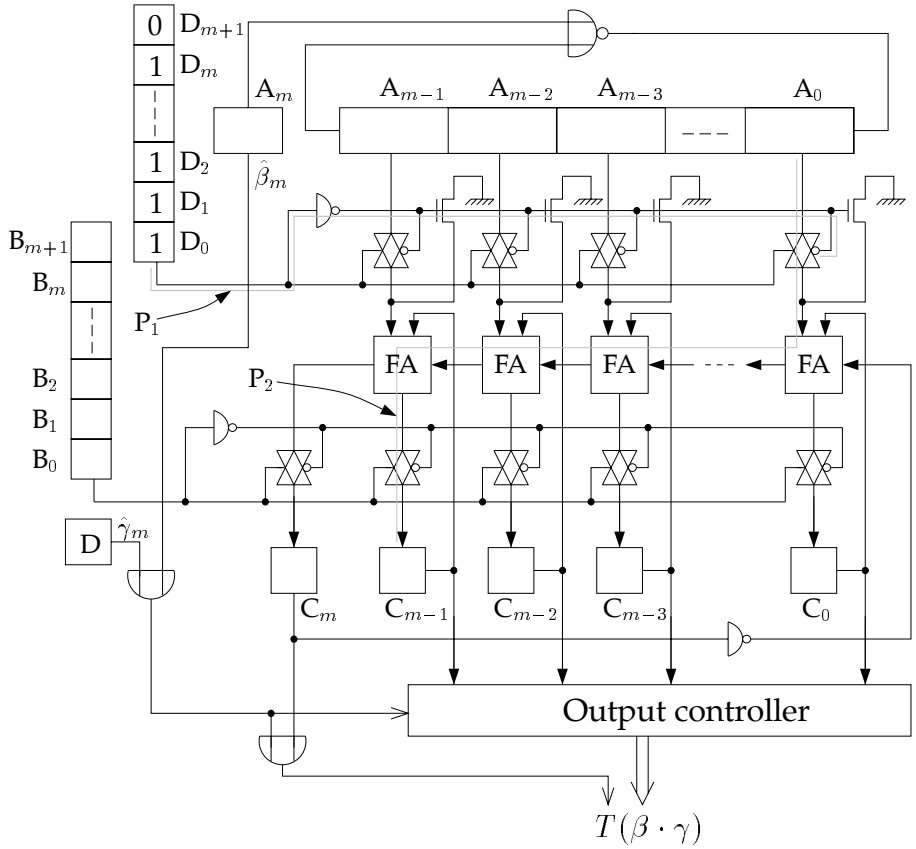


Figure 6.24: *A modified version of the multiplier proposed by Shyu et al. [92, Fig. 1]. The paths* P_1 *and* P_2 *form the CP during one clock cycle.*

and the $(m+2)$-bit registers B and C are loaded with $2^{m+1} + \hat{\gamma}$ and $2^{m+1} - 1$, respectively. Also, the single D flip-flop is loaded with $\hat{\gamma}_m$. After the subsequent $m + 3$ clock cycles, the product $T(\beta \cdot \gamma)$ has been shifted through the output controller and into an $(m + 1)$-bit parallel register. This output register is not shown in Figure 6.24. The multiplication process is described more in detail by Shyu et al. [92] and, to some extent, by Chang et al. [32].

The size of the multiplier in Figure 6.24 equals

$$\begin{aligned} \mathcal{C}_{\text{Shyu,mult}} &= (2(m+2) + 2(m+1) + 1)\mathcal{C}_{\text{reg}} + m\mathcal{C}_{\text{FA}} + \mathcal{C}_{\text{NAND/NOR}} \\ &\quad + 2\mathcal{C}_{\text{OR}} + 2\mathcal{C}_{\text{inv}} + \mathcal{C}_{\text{out,ctrl}} + (2m+1)\mathcal{C}_{\text{TG}} + m \\ &= 103m + 134. \end{aligned}$$

The CP is formed by the dotted paths P_1 and P_2 in the figure. The CP length equals

$$\begin{aligned}\mathcal{L}_{\text{CP,Shyu,mult}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}}(f_{\text{inv}} + mf_{\text{TG}}) + r_{\text{inv}} \cdot m(f_{\text{TG}} + 1) \\ &\quad + (r_{\text{reg}} + 1)f_{\text{FA,signal}} + m\mathcal{L}_{\text{FA,carry}} + (m-1)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}}f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + (r_{\text{FA}} + 1)f_{\text{reg}} \\ &= 18m + 52\end{aligned}$$

Because the desired diminished–1 product is shifted into the output register during the last of a total of $m + 4$ clock cycles, the computation time T is proportional to

$$\begin{aligned}\mathcal{L}_{\text{Shyu,mult}} &= (m+4)\mathcal{L}_{\text{CP,Shyu,mult}} \\ &= 18m^2 + 124m + 208.\end{aligned}$$

Hence, the area-time performance AT^2 is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{Shyu,mult}} &\triangleq \mathcal{C}_{\text{Shyu,mult}}(\mathcal{L}_{\text{Shyu,mult}})^2 \\ &= (103m + 134)(18m^2 + 124m + 208)^2 = \mathcal{O}\left(m^5\right).\end{aligned}$$

It is possible to obtain yet another algorithm for diminished–1 serial/parallel multiplication, based on the general expression in (6.12). For $k = 1$, (6.12) changes to

$$T(\beta \cdot \gamma) \equiv \sum_{i=0}^{m} T(\gamma_i 2^i \beta) \pmod{2^m + 1}. \tag{6.41}$$

This formula is also derived by Shyu et al. [92, Theorem 2]. An architecture for multiplication based on (6.41) is suitably used when the multiplier (γ) is represented on NBC form. The architecture in Figure 6.24 may be modified to be based on (6.41). However, such an architecture is not considered here.

6.3.6.3 Comparisons

In Table 6.4 we have listed the sizes and the total CP lengths of the diminished–1 general multipliers presented in the thesis. It is clear that the multiplier proposed by Ashur et al. [10] has the smallest size and CP length among the bit-parallel architectures. It is also clear that the multiplier proposed by Shyu et al. [92] has the smallest size and CP length among the bit-serial/parallel architectures.

The sizes, total CP lengths, and AT^2 performances of Ashur's and Shyu's multipliers are plotted versus m, for $m = 2^t$; $1 \leq t \leq 8$, in Figure 6.25.

Multiplier type	Subscript name	Figure	Size $\mathcal{C}$	Total CP length $\mathcal{L}$
Bit-parallel	Sunder,mult	6.19	$34m^2 + 166m + 98$	$108m + 156$
	Ashur,mult	6.20	$34m^2 + 94m + 26$	$44m + 120$
	Benaissa,p-mult	6.21	$34m^2 + 222m + 88$	$144m + 112$
Bit-serial/parallel	Benaissa,s/p-mult	6.22	$128m + 72$	$18m^2 + 8m\log_2 m + 156m + 16\log_2 m + 240$
	Benaissa,s/p-mult,2	6.23	$116m + 82$	$18m^2 + 8m\log_2 m + 152m + 16\log_2 m + 232$
	Shyu,mult	6.24	$103m + 134$	$18m^2 + 124m + 208$

Table 6.4: *The sizes and total CP lengths of the diminished–1 multiplier architectures. The product $\mathcal{CL}^2$ is $\mathcal{O}(m^4)$ for the bit-parallel architectures and $\mathcal{O}(m^5)$ for the bit-serial/parallel architectures.*

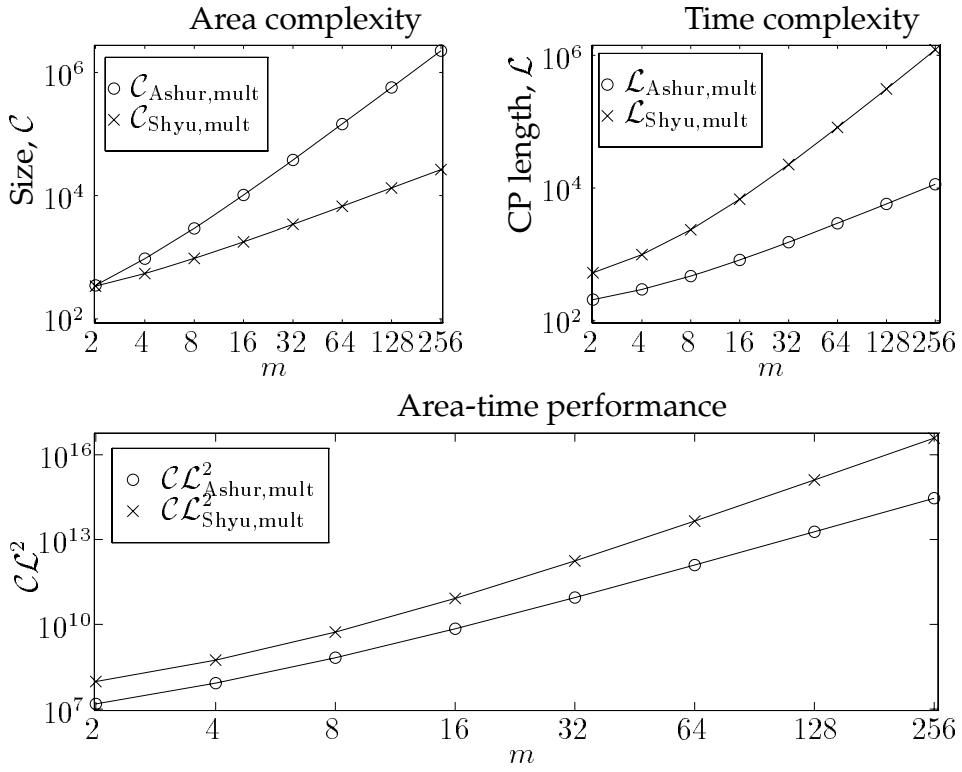


Figure 6.25: *The sizes, total CP lengths, and AT^2 performances of Ashur's and Shyu's diminished–1 multipliers, see Figures* 6.20 *and* 6.24*, respectively. The parameters are plotted versus $m = 2^t$ for $1 \leq t \leq 8$.*

From the figure we conclude that, for all m, the size of Ashur's multiplier is greater than the size of Shyu's multiplier. On the other hand, with respect both to their time performance and their AT^2 performance, Ashur's multiplier is preferable to Shyu's multiplier.

All in all, we conclude that the sizes, the total CP lengths, and the AT^2 performances of the bit-parallel multipliers are $\mathcal{O}(m^2)$, $\mathcal{O}(m)$, and $\mathcal{O}(m^4)$, respectively, while the corresponding parameters of the bit-serial multipliers are $\mathcal{O}(m)$, $\mathcal{O}(m^2)$, and $\mathcal{O}(m^5)$, respectively. The choice of architecture for general multiplication in $\mathbb{Z}_{2^m+1}$ is further discussed in Section 8.1.5.

6.3.7 Exponentiation of the Transform Kernel

For the diminished–1 element representation, the linear coordinate transformation parameters k and l in (6.1) are equal to 1 and -1, respectively. Hence, by (6.14) we get

$$\begin{aligned} T(\gamma^n) &\equiv (T(\gamma)+1)^n - 1 \\ &= \gamma^n - 1 \pmod{2^m+1}, \end{aligned}$$

which is also directly obtained from the general NBC-to-diminished–1 formula $T(\gamma) \equiv \gamma - 1 \pmod{2^m+1}$. It seems as if the diminished–1 representation does not provide a procedure for performing exponentiation in $\mathbb{Z}_{2^m+1}$ which is computationally simpler than the procedures for performing exponentiation with respect to the normal binary coded element representation. Therefore, for the computation of $\gamma^n \bmod 2^m+1$ we refer to the exponentiation procedures described in Section 5.1.6 of the previous chapter. When the modulus 2^m+1 is prime, there are some properties of the *prime field* $\mathbb{Z}_{2^m+1}$ which can be applied such that exponentiation modulo 2^m+1 can be performed in a simplified way. This is further discussed in Section 7.2.1.

6.4 Summary

The complexity and performance parameters of the architectures considered in this chapter are listed in Table 6.5. Regarding the parameters for the architectures for general diminished–1 multiplication, we refer to Table 6.4.

Operation	Figure	Subscript name	Size $\mathcal{C}$	Fan-in f	Internal CP length $\mathcal{L}_{\text{CP}}$	
NBC to dim.–1 transl.	6.2	NBC2Dim	$4m\log_2 m + 8m - 6$	8	$4m + 8\log_2 m$	
Dim.–1 to NBC transl.	6.4	Dim2NBC	$18m + 2$	2	$10m - 2$	
Negation	6.6	dimneg	$4m$	$n_m + 2m$	—	
Addition (carry l.-a.)	6.7	dimadd,1	$56m + 12$	12	$14m + 8\log_2 m + 20$	$\cdots$
Addition (carry-r.)	6.9	dimadd,2	$50m + 2$	6	$18m$	
Multiplication by 2	6.12	dimmult2	4	$n_m + 4$	—	
Multiplication by 2^n	6.15	seq,mult2n	$28m - 4$	—	38 (or 40)	
General multiplication	See Table 6.4					

	Norm. output res. r_o	Total CP length $\mathcal{L}$ (including registers)	Area-time perf. $\mathcal{CL}^2$
	1	$4m + 8\log_2 m + 40$	$\mathcal{O}(m^3 \log_2 m)$
	2	$10m + 28$	$\mathcal{O}(m^3)$
	2	$4m + 30$	$\mathcal{O}(m^3)$
$\cdots$	2	$14m + 8\log_2 m + 70$	$\mathcal{O}(m^3)$
	3	$18m + 40$	$\mathcal{O}(m^3)$
	2	42	7056
	—	$40(n^{(t-1)} + 2)$ (or $4m + 42$)	$\mathcal{O}(m^3)$
	$\longleftarrow$	$\longleftarrow$	

Table 6.5: *Complexity parameters of the architectures in the present chapter.*

Chapter 7

The Polar Representation

In Chapter 6, we considered the diminished–1 representation of the elements in Fermat integer quotient rings $\mathbb{Z}_{2^m+1}$. The code translation $T(\gamma) \equiv \gamma - 1 \pmod{2^m + 1}$, where γ is an NBC integer of $\mathbb{Z}_{2^m+1}$ and $T(\gamma)$ is the diminished–1 representation of γ, belongs to the set of *linear* coordinate transformations given by (6.1).

There exist many forms of *nonlinear* coordinate transformations, i.e. mappings P from the NBC representation of integers $\gamma \in \mathbb{Z}_{2^m+1}$ to $P(\gamma)$, where P is a nonlinear function of γ. In this chapter we investigate the properties of one such form of representation, namely the *polar representation*.[1] A restriction of the polar representation, however, is that it is only applicable in finite fields.

7.1 Introduction

For $m = 1, 2, 4, 8, 16$ the Fermat number $2^m + 1$ is prime and hence the integer quotient ring $\mathbb{Z}_{2^m+1}$ is a field. Let α be a primitive element of a prime field $\mathbb{Z}_p$, i.e. an element of $\mathbb{Z}_p$ with maximum order $p-1$. It is well known [60, Th. 1.15, Th. 2.8] that the multiplicative group of nonzero elements of $\mathbb{Z}_p$ can be formed by the cyclic group $\{\alpha^0, \alpha^1, \ldots, \alpha^{p-3}, \alpha^{p-2}\}$.[2] Let the symbol $*$ be defined by

[1] In the literature, the polar representation is sometimes referred to as the *index representation*, see for example Niederreiter [60, Ch. 10.1] and Rosen [84, Ch. 8.4].

[2] Actually, for *any* finite field, its multiplicative group can be formed by its powers of a primitive element.

the equation $\alpha^* = 0$. Then, any integer of $\mathbb{Z}_p$ can be expressed as some power of α.

Definition 7.1 *Consider the Fermat prime fields $\mathbb{Z}_{2^m+1}$; $m = 1, 2, 4, 8, 16$. In the* polar representation *of $\mathbb{Z}_{2^m+1}$, each element (integer) $\gamma \in \mathbb{Z}_{2^m+1}$ is represented by its associated power $P(\gamma)$ of a primitive element α of the field.*
Accordingly, we have

$$\gamma \equiv \alpha^{P(\gamma)} \pmod{2^m + 1}, \tag{7.1}$$

where $\begin{cases} P(\gamma) \in \mathbb{Z}_{2^m}; & \gamma \neq 0 \\ P(0) = * \end{cases}$.
An element in the polar representation is referred to as a polar *element.*

In the diminished–1 representation, the zero element is represented by the integer $T(0) = 2^m$, which we called the *zero indicator*, see Section 6.2. We suitably use the integer 2^m as a zero indicator also in the polar representation, i.e. we have $P(0) = * \triangleq 2^m$. Similar to the diminished–1 representation, by letting all integers $P(\gamma)$ be $(m+1)$-bit normal binary coded integers, the zero representative $P(0)$ is the only integer $P(\gamma)$ for which its most significant bit equals one. Situations where one of the operands in an arithmetic operation is the zero element are handled separately. For nonzero integers γ we have $P(\gamma) \in \mathbb{Z}_{2^m}$ and the order of the primitive element α modulo $2^m + 1$ equals 2^m. Consequently, *for nonzero integers γ we can use an m-bit binary arithmetic modulo 2^m for the associated exponents $P(\gamma)$ of α.*

The general properties of arithmetic operations in finite fields, with respect to the polar representation, are well known. However, the particular properties of arithmetic operations in *Fermat prime fields*, with respect to the polar representation have not been studied before. An investigation of such properties is carried out in this chapter. Henceforth, we generally refer to $\mathbb{Z}_{2^m+1}$ as a Fermat *prime field*.

7.2 Arithmetic Operations

Occasionally, we have denoted diminished–1 elements $T(\gamma)$ by $\hat{\gamma}$. In the polar representation we conveniently use the same kind of notation, i.e. the $(m+1)$-bit polar integer $P(\gamma)$ is denoted by the normal binary coded integer $\hat{\gamma} = 2^m\hat{\gamma}_m + 2^{m-1}\hat{\gamma}_{m-1} + \cdots + 2\hat{\gamma}_1 + \hat{\gamma}_0$.

In the present section we describe the arithmetic operations involved in the computation of the Fermat number transform with respect to the polar representation. Later, in Section 7.6, we also consider VLSI architectures for some of these arithmetic operations.

7.2.1 Discrete Exponentiation

The code translation from a polar number $P(\gamma)$ to its corresponding normal binary coded number γ is carried out using a discrete exponentiation modulo $2^m + 1$, as given by (7.1). In Section 5.1.6 we considered some procedures for *general* exponentiation modulo $2^m + 1$. The integer $\gamma \equiv \alpha^{P(\gamma)} \pmod{2^m + 1}$ may be computed using any of those procedures. For example, by using the well known binary method, which is briefly described in Section 5.1.6, exponentiation can be performed using $m-1$ squarings and at most $m-1$ multiplications modulo $2^m + 1$. By performing a squaring as a general multiplication, at most $2(m-1)$ general multiplications modulo 2^m+1 of normal binary coded numbers are required to compute γ from $P(\gamma)$ in (7.1).

In Section 7.4 we consider a new procedure [5] for discrete exponentiation in Fermat prime fields using some properties of Zech's logarithms.

7.2.2 The Discrete Logarithm

By taking the α-logarithm of both sides of (7.1) we get the congruence

$$P(\gamma) \equiv \log_\alpha \gamma \pmod{2^m}, \tag{7.2}$$

which is called the *discrete logarithm* to the base α modulo 2^m. The problem of computing (7.2) is generally known as the discrete logarithm problem. In general, it is quite hard to compute the discrete logarithm in a large prime field $\mathbb{Z}_p$. Several algorithms suggested in the literature require $\mathcal{O}\left(\sqrt{p}\right)$ multiplications to compute the logarithm.

The Pohlig-Hellman Algorithm

In 1978, Pohlig and Hellman [72] presented an algorithm for computing the discrete logarithm in $\mathbb{Z}_p$ which only requires $\mathcal{O}\left(\log^2 p\right)$ multiplications modulo p. In particular, for Fermat primes $p = 2^m + 1$ their algorithm computes (7.2) by recursively determining the binary digits $\hat{\gamma}_i$ of $T(\gamma) = \hat{\gamma} = (\hat{\gamma}_{m-1}, \hat{\gamma}_{m-2}, \hat{\gamma}_{m-3}, \ldots, \hat{\gamma}_0)_2$ such that $\gamma \equiv \alpha^{\hat{\gamma}} \pmod{2^m + 1}$ holds. The algorithm, which

is based on the fact that the order of the primitive element α modulo 2^m+1 equals 2^m, works as follows (see [72, Sec. III]):

The least significant bit $\hat{\gamma}_0$ of $\hat{\gamma}$ is determined by raising the nonzero integer γ to the 2^{m-1}th power and identifying whether the result equals 1 or -1. Let $\gamma(0) \stackrel{\triangle}{=} \gamma$ and $\delta(0) \equiv \alpha^{-1} \pmod{2^m+1}$. Then we have

$$\begin{aligned} \gamma(0)^{2^{m-1}} &\equiv (\alpha^{\hat{\gamma}})^{2^{m-1}} = \alpha^{(\hat{\gamma}_{m-1}2^{m-2}+\hat{\gamma}_{m-2}2^{m-3}+\cdots+\hat{\gamma}_1)2^m+\hat{\gamma}_0 2^{m-1}} \equiv (\alpha^{2^{m-1}})^{\hat{\gamma}_0} \\ &\equiv (-1)^{\hat{\gamma}_0} \equiv \begin{cases} 1 \pmod{2^m+1}; & \text{if } \hat{\gamma}_0 = 0 \\ -1 \pmod{2^m+1}; & \text{if } \hat{\gamma}_0 = 1 \end{cases}. \end{aligned} \tag{7.3}$$

Only $m-1$ squarings are required to compute $\gamma(0)^{2^{m-1}} \bmod 2^m+1$. The digit $\hat{\gamma}_0$ is set to either 0 or 1, depending on whether $\gamma(0)^{2^{m-1}} \bmod 2^m+1$ is evaluated to 1 or -1, respectively. Now, let $\gamma(1) \equiv \gamma(0) \cdot \delta(0)^{\hat{\gamma}_0} \pmod{2^m+1}$. The digit $\hat{\gamma}_1$ can be determined in the same way as above from the congruence

$$\begin{aligned} \gamma(1)^{2^{m-2}} &\equiv \alpha^{(\hat{\gamma}_{m-1}2^{m-3}+\hat{\gamma}_{m-2}2^{m-4}+\cdots+\hat{\gamma}_2)2^m+\hat{\gamma}_1 2^{m-1}} \\ &\equiv (-1)^{\hat{\gamma}_1} \equiv \begin{cases} 1 \pmod{2^m+1}; & \text{if } \hat{\gamma}_1 = 0 \\ -1 \pmod{2^m+1}; & \text{if } \hat{\gamma}_1 = 1 \end{cases}, \end{aligned} \tag{7.4}$$

which can be computed using $m-2$ squarings. Next, compute $\delta(1) \equiv \delta(0)^2 \pmod{2^m+1}$ and $\gamma(2) \equiv \gamma(1) \cdot \delta(1)^{\hat{\gamma}_1} \pmod{2^m+1}$ and determine $\hat{\gamma}_2$ from $\gamma(2)^{2^{m-1}} \bmod 2^m+1$, etc., until the most significant bit $\hat{\gamma}_{m-1}$ has been determined.

In order to determine the digit $\hat{\gamma}_i;\ 2 \le i \le m-1$, one squaring is required to compute $\delta(i-1) \equiv \delta(i-2)^2 \pmod{2^m+1}$, one multiplication is required to compute the product $\gamma(i) \equiv \gamma(i-1) \cdot \delta(i-1)^{\hat{\gamma}_1} \pmod{2^m+1}$, and $m-i-1$ squarings are required to compute $\gamma(i)^{m-i-1} \bmod 2^m+1$. Hence, if $\delta(0)$ is precomputed and squarings are performed as multiplications, the algorithm requires approximately $2m + \sum_{i=0}^{m-1} i = m(m+3)/2$ general multiplications modulo 2^m+1. Assuming that each multiplication can be carried out using at most m additions, the Pohlig-Hellman algorithm requires at most $m^2(m+3)/2$ additions modulo 2^m+1.

New Algorithms

In 1993, we [5] presented a new algorithm for computing the discrete logarithm in Fermat prime fields $\mathbb{Z}_{2^m+1}$. The algorithm, which is based on some properties of *Zech's logarithms* in Fermat prime fields, requires at most $2^m/2m$ additions modulo 2^m+1.

Recently, we [7] proposed another algorithm for computing the discrete logarithm, which in turn is based on the algorithm in [5]. By using a look-up table

of size $2^m/2m \times m$ bits, we show how to compute the logarithm using at most $2m-1$ binary shifts (rotations), one table look-up, and one addition and one simplified multiplication modulo $2m$.

Our two algorithms are thoroughly described in Sections 7.4 and 7.5, respectively.

7.2.3 Modulus Reduction

Modulus reduction in the polar representation is a very simple operation. When γ is nonzero, the least positive residue of $\hat{\gamma}$ modulo 2^m equals $\hat{\gamma}^{(m-1)}$, which is instantaneously obtained from $\hat{\gamma}$. When γ is congruent to zero modulo 2^m+1 we have $\hat{\gamma}=2^m$.

Because we use an $(m+1)$-bit normal binary coded representation of the exponents $\hat{\gamma}$ of α, there are only two cases that have to be considered:

Exponent		Reduced exponent (mod 2^m)
$\hat{\gamma} \neq 2^m$	$\Longrightarrow$	$P(\gamma \neq 0) = \hat{\gamma}^{(m-1)}$
$\hat{\gamma} = 2^m$	$\Longrightarrow$	$P(0) = * = 2^m$

7.2.4 Negation

Like modulus reduction, negation is also simply carried out in the polar representation. Because the order of the primitive element α modulo 2^m+1 equals 2^m we have $\alpha^{2^{m-1}} \equiv -1 \pmod{2^m+1}$. For $\hat{\gamma} = P(\gamma)$ and $\hat{\varphi} = P(-\gamma)$ we therefore get

$$\varphi \equiv -\gamma \equiv \alpha^{2^{m-1}+\hat{\gamma} \pmod{2^m}} \equiv \alpha^{\hat{\varphi}} \pmod{2^m+1}.$$

Hence, because in the polar representation all arithmetic operations are carried out in the exponent of α, the polar element $P(-\gamma)$ is obtained from the congruence

$$P(-\gamma) = \hat{\varphi} \equiv 2^{m-1} + \hat{\gamma} \pmod{2^m}, \tag{7.5}$$

which, for $0 \leq \hat{\gamma} \leq 2^m - 1$, we expand as

$$\begin{aligned} \hat{\varphi} &\equiv 2^{m-1} + \hat{\gamma}_{m-1}(2^m - 2^{m-1}) + \hat{\gamma}^{(m-2)} \\ &= \hat{\gamma}_{m-1}2^m + (1-\hat{\gamma}_{m-1})2^{m-1} + \hat{\gamma}^{(m-2)} \\ &\equiv \overline{\hat{\gamma}_{m-1}}2^{m-1} + \hat{\gamma}^{(m-2)} \pmod{2^m}. \end{aligned} \tag{7.6}$$

For $\hat{\gamma} = 2^m$, i.e for $\gamma = 0$, we let $\hat{\varphi} = \hat{\gamma} = 2^m$. In Section 7.6.3 we consider a VLSI architecture for negation based on (7.6).

7.2.5 Addition and Subtraction

Addition

When considering addition in the polar representation we need the following definition (see for example Conway [34, Ch. 6]).

Definition 7.2 Zech's logarithm[3] *of the polar element $\hat{\theta}$ is denoted by $Z(\hat{\theta})$ and defined by the congruence*

$$1 + \alpha^{\hat{\theta}} \equiv \alpha^{Z(\hat{\theta})} \pmod{2^m + 1}.$$

For *nonzero* $\beta, \gamma \in \mathbb{Z}_{2^m+1}$, let $\hat{\beta} = P(\beta)$ and $\hat{\gamma} = P(\gamma)$. The function evaluated when performing addition in the polar representation is found in the exponent of α in the congruence

$$\begin{aligned} \varphi &\equiv \beta + \gamma \equiv \alpha^{\hat{\beta}}(1 + \alpha^{\hat{\gamma}-\hat{\beta}}) \\ &\equiv \alpha^{\hat{\beta}+Z(\hat{\gamma}-\hat{\beta})} \pmod{2^m} \equiv \alpha^{\hat{\varphi}} \pmod{2^m + 1}, \end{aligned} \tag{7.7}$$

i.e. we have

$$P(\beta + \gamma) = \hat{\varphi} \equiv \hat{\beta} + Z\left(\hat{\gamma} - \hat{\beta}\right) \pmod{2^m}, \tag{7.8}$$

where Z is Zech's logarithm. Using the congruence $-\hat{\beta} \equiv (2^m - 1) - \hat{\beta} + 1 = \overline{\hat{\beta}^{(m-1)}} + 1 \pmod{2^m}$, where $\overline{\hat{\beta}^{(m-1)}}$ is the one's complement of $\hat{\beta}^{(m-1)}$, we rewrite (7.8) as

$$\hat{\varphi} \equiv \hat{\beta}^{(m-1)} + Z\left(\hat{\gamma} + \overline{\hat{\beta}^{(m-1)}} + 1\right) \pmod{2^m}. \tag{7.9}$$

Hence, according to (7.9), addition in the polar representation may be carried out using two additions and one discrete logarithm modulo 2^m. The direct computation of $Z(\hat{\theta})$, as expressed by the congruence

$$Z(\hat{\theta}) \equiv \log_\alpha(1 + \alpha^{\hat{\theta}}) \pmod{2^m}, \tag{7.10}$$

requires one discrete exponentiation and one addition modulo $2^m + 1$ followed by one discrete logarithm modulo 2^m, which makes it quite an intricate function. Some researchers, like for instance Conway [34], Imamura [53], and Huber [51] have considered different methods of computing Zech's logarithms

[3] Zech's logarithm is also referred to as *Jacobi's logarithm* [60, Exc. 2.8].

in $GF(p^n)$ in a simplified way. In particular, the researchers consider fields of characteristic $p = 2$. In order to speed up the computation of $Z(\hat{\theta})$, the mentioned methods all involve the use of look-up tables.

Remark: The particular properties of Zech's logarithms in Fermat prime fields $\mathbb{Z}_{2^m+1}$ are investigated in Section 7.3. The main purpose of the investigation is to find an area-time efficient way of computing Zech's logarithms in $\mathbb{Z}_{2^m+1}$.

The case when either of the addends β and γ (or both) equals zero is handled separately. For $\varphi \equiv \beta + \gamma \pmod{2^m+1}$, where $\beta = 0$ or $\gamma = 0$, we can simply do the following:

$$\begin{aligned} &\text{If } \beta = 0 \ (\hat{\beta} = 2^m), \quad \text{then let } \varphi = \gamma, \text{ i.e. let } \hat{\varphi} = \hat{\gamma} \\ &\text{If } \gamma = 0 \ (\hat{\gamma} = 2^m), \quad \text{then let } \varphi = \beta, \text{ i.e. let } \hat{\varphi} = \hat{\beta} \end{aligned}$$

Subtraction

The polar integer $P(\beta - \lambda)$, for which β and λ are nonzero integers of $\mathbb{Z}_{2^m+1}$, can be derived by letting $\gamma = -\lambda$ in (7.7). Then, by (7.8) we get

$$P(\beta - \lambda) \equiv \hat{\beta} + Z\left(P(-\lambda) - \hat{\beta}\right) \pmod{2^m}. \tag{7.11}$$

Consequently, subtraction in the polar representation can be carried out in a conventional way as a (polar) negation followed by a (polar) addition.

7.2.6 General Multiplication

For nonzero β and γ, the product $\varphi \equiv \beta\gamma \pmod{2^m+1}$ can be expanded as

$$\varphi \equiv \beta\gamma \equiv \alpha^{\hat{\beta}+\hat{\gamma} \pmod{2^m}} \equiv \alpha^{\hat{\varphi}} \pmod{2^m+1}. \tag{7.12}$$

By this congruence we get

$$P(\beta\gamma) = \hat{\varphi} \equiv \hat{\beta} + \hat{\gamma} \pmod{2^m}, \tag{7.13}$$

which is a well known property of the polar representation; general multiplication in a finite field $GF(p^n)$ turns into addition modulo $p^n - 1$ when using a polar representation. When either of the factors β and γ (or both) equals zero, $P(\beta\gamma)$ is set to $P(0) = 2^m$.

7.2.7 Multiplication by Powers of ω

The computation of the Fermat number transform of length N involves multiplication by powers of the transform kernel ω of order N. Let $\beta \equiv \omega^{n \bmod N} \pmod{2^m+1}$, where $P(\omega) = \hat{\omega}$. Then, by (7.12) and (7.13) it follows that

$$P(\omega^n \gamma) \equiv \hat{\gamma} + (n \bmod N)\hat{\omega} \pmod{2^m}. \tag{7.14}$$

Multiplication by 2^n

The Fermat number transforms most commonly used are the ones of lengths $N = 2m$ and $4m$, with transform kernels $\omega = 2$ and $\omega = \sqrt{2} = 2^{\frac{3m}{4}} + 2^{\frac{5m}{4}}$, respectively. The main reason is that, with respect to the diminished–1 and the NBC representations of the integers of $\mathbb{Z}_{2^m+1}$, multiplication by powers of ω can then be carried out as binary shifts (rotations) (see Sections 2.3.2, 5.1.4, and 6.3.5).

Multiplication by powers of two can be carried out in a simple way in the polar representation as well. By the congruence $\alpha^{2^{m-1}} \equiv -1 \equiv 2^m \equiv (1+\alpha^0)^m \equiv \alpha^{mZ(0)} \pmod{2^m+1}$, where α is a primitive element of $\mathbb{Z}_{2^m+1}$, we get

$$mZ(0) \equiv 2^{m-1} \pmod{2^m}. \tag{7.15}$$

Because m is a power of two; $m = 2^t$; $t = 0, 1, 2, 3, 4$, by Theorem 3.4 of Rosen [84] we can rewrite (7.15) as

$$Z(0) \equiv 2^c \pmod{2^{c+1}}, \tag{7.16}$$

where c is defined by the equation

$$2^c = \frac{2^m}{2m}. \tag{7.17}$$

Consequently, for some integer k we can write

$$Z(0) = k2^{c+1} + 2^c = \hat{\delta}2^c, \tag{7.18}$$

where $\hat{\delta} = 2k+1$. Because $\hat{\delta}$ is an odd $(t+1)$-bit normal binary coded integer, where $t = \log_2 m$, it follows that $k \in \mathbb{Z}_m$. Thus, depending of the primitive element α chosen, the corresponding Zech's logarithm of zero is of the form given in (7.18) for *some* $k = 0, 1, 2, \ldots, m-1$.

Theorem 7.1 *For* each $k \in \mathbb{Z}_m$ *there exist* 2^c *primitive elements* α *of* $\mathbb{Z}_{2^m+1}$ *such that the equality* $Z(0) = (2k+1)2^c$ *holds.*

Theorem 7.1 may also be formulated as follows. The primitive elements of $\mathbb{Z}_{2^m+1}$ can be partitioned into m sets, each comprising 2^c elements, such that the primitive elements in each set all have the same Zech's logarithm of zero on the form given by (7.18).

Proof: Let α and $\tilde{\alpha}$ be two primitive elements of $\mathbb{Z}_{2^m+1}$. By Corollary 8.4.1 of Rosen [84] we know that α^u is a primitive element of $\mathbb{Z}_{2^m+1}$ if and only if $\gcd(u, 2^m) = 1$, which is true for all *odd* integers u. Hence, for some integer $r \in \mathbb{Z}_{2^{m-1}}$, $\tilde{\alpha}$ can be written on the form $\tilde{\alpha} \equiv \alpha^{2r+1} \pmod{2^m+1}$. By (7.10) we have $Z(0) \equiv \log_\alpha(1+\alpha^0) \pmod{2^m}$. Suppose that $\tilde{\alpha}$ has the same Zech's logarithm of zero as α, i.e. suppose $Z(0) \equiv \log_{\tilde{\alpha}}(1+\tilde{\alpha}^0) \pmod{2^m}$. Then it follows that

$$\alpha^{Z(0)} \equiv \tilde{\alpha}^{Z(0)} \equiv \alpha^{(2r+1)Z(0)} \pmod{2^m+1}$$

and hence we have $Z(0) \equiv (2r+1)Z(0) \pmod{2^m}$, where $Z(0) = \hat{\delta}2^c$. Because $2^m/\gcd(2^c, 2^m) = 2m$, by [84, Th. 3.4] it follows that $\hat{\delta} \equiv (2r+1)\hat{\delta} \pmod{2m}$. Consequently, α and $\tilde{\alpha} \equiv \alpha^{(2r+1)} \pmod{2^m+1}$ have the same Zech's logarithm of zero only if $2r \equiv 0 \pmod{2m}$, or equivalently if m is a divisor of r.

From the above reasoning we conclude that there exist exactly m Zech's logarithms of zero on the form given by (7.18). By [84, Th. 8.5] we know that there are $\phi(\phi(2^m+1)) = 2^{m-1}$ primitive roots of $\mathbb{Z}_{2^m+1}$. Hence, these primitive elements can be partitioned into m sets of $2^{m-1}/m = 2^c$ elements, which all have the same Zech's logarithm of zero. □

For $\omega = 2$ we get $\hat{\omega} = Z(0)$ and $N = 2m$ in (7.14). By Theorem 7.1, there exist 2^c primitive elements for which the associated Zech's logarithms of zero are all equal to 2^c ($\hat{\delta} = 1$ in (7.18)). Consequently, by appropriately choosing such an α, multiplication by a power of two can be computed in the polar representation as in (7.14), with $\omega \equiv \alpha^{2^c} \pmod{2^m}$, i.e. we get

$$P(2^n\gamma) \equiv \hat{\gamma} + (n \bmod 2m)2^c \pmod{2^m}. \tag{7.19}$$

Let $\hat{\theta} = (n \bmod 2m)2^c = n^{(t)}2^c$. This binary coded integer may be computed as $c = m-(t+1)$ binary shifts of $n^{(t)}$. However, because the factor $n^{(t)}$ is a $(t+1)$-bit NBC integer, no reduction modulo 2^m is needed for the m-bit NBC integer $n^{(t)}2^c$. The shifts can therefore be carried out instantaneously and hence the evaluation of (7.19) only requires one addition. Furthermore, since $\hat{\theta}^{(c-1)} = 0$, i.e. the c least significant bits of $\hat{\theta}$ are zero, this addition modulo 2^m simplifies to a $(t+1)$-bit addition of $n^{(t)}$ by the $t+1$ most significant bits of $\hat{\gamma}$. The sum is reduced modulo 2^{t+1}. This computational procedure is generalised and further explained below.

An Optimal Choice of ω

The main results in the remainder of the present section (7.2.7) can also be found in [6]. Because the transform length N is a power of two, i.e. we have $N = 2^b$ for $0 \leq b \leq m$, we can write (7.14) as

$$P(\omega^n \gamma) \equiv \hat{\gamma} + (n \bmod 2^b)\hat{\omega} \equiv \hat{\gamma} + n^{(b-1)}\hat{\omega} \pmod{2^m}. \tag{7.20}$$

By choosing an appropriate kernel ω, it is possible to compute $P(\omega^n \gamma)$ with a complexity that is smaller than the complexity of performing one general multiplication followed by one addition modulo 2^m, i.e. according to the direct computation of (7.20). We showed above that for some bases α and for $\omega = 2$, $P(2^n \gamma)$ can conveniently be computed using only one addition modulo $2m = 2^{t+1}$. That simple way of computing $P(2^n \gamma)$ is actually a special case of a general procedure for computing $P(\omega^n \gamma)$ which only requires one addition modulo N for *all* possible transform lengths $N = 2^b$ in $\mathbb{Z}_{2^m+1}$.

Theorem 7.2 *Let* $\omega \equiv \alpha^{2^{m-b}} \pmod{2^m+1}$ *and* $P(\omega^n \gamma) \equiv \hat{\gamma} + n^{(b-1)}\hat{\omega} \pmod{2^m}$, *where* α *is a primitive element of the prime field* $\mathbb{Z}_{2^m+1}$, n *is a nonnegative integer, and* $0 \leq b \leq m$. *Then the order of* ω *modulo* 2^m+1 *equals* 2^b *and* $P(\omega^n \gamma)$ *can be computed using only one* b*-bit addition modulo* 2^b.

The choice of $\omega \equiv \alpha^{2^{m-b}} \pmod{2^m+1}$ as the kernel of a Fermat number transform of length 2^b was also considered in Section 2.3.2 (page 15). In the proof of Theorem 7.2 we need the following notation.

Definition 7.3 *Let* $\hat{\beta}$ *be an* m*-bit polar integer. By* $\hat{\beta}_{(i)}$ *we denote the NBC integer which is formed by the* $m - i$ *most significant bits of* $\hat{\beta}$ *such that, for* $1 \leq i \leq m$, $\hat{\beta}$ *can be written on the form* $\hat{\beta} = \hat{\beta}_{(i)} 2^i + \hat{\beta}^{(i-1)}$.

Proof: (Theorem 7.2) The order of the primitive element α modulo the prime $2^m + 1$ equals $\phi(2^m + 1) = 2^m$. By Theorem 8.4 of Rosen [84], for $0 \leq b \leq m$ the order of $\alpha^{2^{m-b}}$ modulo $2^m + 1$ equals $2^m / \gcd(2^m, 2^{m-b}) = 2^b$. Hence, the element

$$\omega \equiv \alpha^{2^{m-b}} \pmod{2^m + 1}$$

can be used as the kernel of a Fermat number transform of length $N = 2^b$. Similar to the notation used above, let[4] $\hat{\theta} = n^{(b-1)}\hat{\omega}$, where we now have $\hat{\omega} = P(\omega) = 2^{m-b}$. Thus, using this definition of $\hat{\theta}$ we can write

$$P(\omega^n \gamma) \equiv \hat{\gamma} + \hat{\theta} \pmod{2^m}, \tag{7.21}$$

[4] For $b = t + 1$ we get $N = 2^b = 2^{t+1} = 2m$, which is the order of the transform kernel $\omega = 2$ used above.

where $\hat{\theta} = n^{(b-1)}2^{m-b}$. Because $n^{(b-1)}$ is a b-bit NBC integer, $\hat{\theta}$ is an m-bit NBC integer for which $\hat{\theta}^{(m-b-1)} = 0$, i.e. the $m-b$ least significant bits of $\hat{\theta}$ equal zero. Hence, $P(\omega^n\gamma)$ in (7.21) can be computed as a b-bit addition modulo 2^b. By Definition 7.3 we can write $\hat{\gamma} = \hat{\gamma}_{(m-b)}2^{m-b} + \hat{\gamma}^{(m-b-1)}$ and

$$P(\omega^n\gamma) \triangleq \hat{\varphi} = \hat{\varphi}_{(m-b)}2^{m-b} + \hat{\varphi}^{(m-b-1)}, \tag{7.22}$$

where

$$\begin{cases} \hat{\varphi}_{(m-b)} & \equiv & \hat{\gamma}_{(m-b)} + n^{(b-1)} \pmod{2^b} \\ \hat{\varphi}^{(m-b-1)} & = & \hat{\gamma}^{(m-b-1)} \end{cases} . \tag{7.23}$$

Obviously, $P(\omega^n\gamma)$ can be computed using only one b-bit addition of $\hat{\gamma}_{(m-b)}$ and $n^{(b-1)}$ modulo 2^b. □

Theorem 7.2 leads immediately to the following corollary.

Corollary 7.1 *Consider a Fermat number transform in the prime field $\mathbb{Z}_{2^m+1}$ and of arbitrary transform length $N = 2^b$, such that $0 \leq b \leq m$. By letting $\omega \equiv \alpha^{2^{m-b}} \pmod{2^m+1}$ be the kernel of the transform, in the polar representation each transform multiplication by a power of ω can be computed using only one addition modulo 2^b.*

Proof: The proof follows directly from Theorem 7.2. □

7.3 Zech's Logarithm

In Section 7.2.5 we saw that polar addition involves the evaluation of a Zech's logarithm. The computational complexity of polar addition depends heavily on the complexity of computing the Zech logarithm. Zech's logarithms are defined in Definition 7.2 (page 160). In this section we investigate some properties of Zech's logarithm over Fermat prime fields, with the purpose of finding an (area-time) efficient way of computing the logarithm.

We mentioned in Section 7.2.5 that some researchers have considered different methods of computing Zech's logarithms in finite fields, in particular fields of characteristic two. To the authors knowledge, their methods all involve the use of a look-up table. See for example Huber's [51] technique for computing Zech's logarithm in $GF(2^n)$. He uses a restricted set of elements which, together with their Zech's logarithms, are stored in a look-up table. Arbitrary Zech's logarithms in the field can then be computed by using this table and

some properties of Zech's logarithms in fields of characteristic two. However, some of the properties used by Huber for computing Zech's logarithms do not apply to prime fields. In this chapter, we present new methods of computing Zech's logarithms which only apply to Fermat prime fields.

In Appendix C, we investigate some special properties of Zech's logarithms in Fermat prime fields. Using these properties we show that the integers of $\mathbb{Z}_{2^m} \setminus \{2^{m-1}\}$ can be partitioned into $(2^m + 2)/6$ subsets of six integers each,[5] such that the Zech logarithm of any integer of a subset can be computed from any of the other integers of the set and its Zech's logarithm. Consequently, a method of computing Zech's logarithm could be the following:
Select one integer $\hat{\beta}$ from each subset and store the associated Zech's logarithms $Z(\hat{\beta})$ in a look-up table (of size $(2^m + 2)/6 \times m$ bits). Given $\hat{\gamma}$, suppose we want to compute $Z(\hat{\gamma})$.

1. The first step is to find which subset contains $\hat{\gamma}$. We know that the Zech logarithm $Z(\hat{\beta})$ of one of the integers $\hat{\beta}$ of this subset is stored in the table. Thus, the first step of the method is to find $\hat{\beta}$ and then obtain $Z(\hat{\beta})$ from the look-up table.
2. The remaining integers of the subset are subsequently computed using the equations in Theorem C.2 until $\hat{\gamma}$ is found.
3. The desired logarithm $Z(\hat{\gamma})$ is computed using the appropriate equation in Theorem C.1.

In step 2, at most one addition modulo 2^m is required to compute, from $\hat{\beta}$ and $Z(\hat{\beta})$, an arbitrary integer of the subset. In step 3, $Z(\hat{\gamma})$ can also be computed using at most one addition modulo 2^m.

There is, however, a major drawback of this procedure for computing Zech's logarithms. We have not yet discovered a straightforward way of finding a simple connection between an arbitrary integer of $\mathbb{Z}_{2^m} \setminus \{2^{m-1}\}$ and the associated subset to which it belongs. Thus, in the above step 1 we are not able to find the subset which contains $\hat{\gamma}$, or equivalently, find the associated integer $\hat{\beta}$ in the table, without searching the whole table. Therefore, the above procedure is not further considered in this section. The properties of the integers of the mentioned subsets (and their Zech's logarithms) are thoroughly investigated in Appendix C.

In the following section we consider properties of Zech's logarithms which lead to procedures for computing Zech's logarithms, either with or without

[5] However, one of the subsets only contains three integers.

the use of look-up tables. When using look-up tables, the tables required are smaller than the table of size $(2^m+2)/6 \times m$ bits used in the above-mentioned procedure. The main contents of Sections 7.4 and 7.5 has recently been presented by the author in [5] and [7].

7.4 Properties of the $\mathcal{D}_m$ Matrix

Definition 7.4 *Let $\hat{\gamma}$ be a polar integer, i.e. $\hat{\gamma} \in \mathbb{Z}_{2^m} \cup \{*\}$. We define the jth Zech logarithm of $\hat{\gamma}$ as*

$$Z^{\{j\}}(\hat{\gamma}) = Z\left(Z^{\{j-1\}}(\hat{\gamma})\right); \qquad j \in \mathbb{Z},$$

where $Z^{\{0\}}(\hat{\gamma}) = \hat{\gamma}$.

From Definition 7.2 of Zech's logarithm in Fermat prime fields we have (see also (C.1) and (C.2) in Appendix C)

$$\begin{aligned} Z(2^{m-1}) &\equiv * \pmod{2^m} \\ Z(*) &\equiv 0 \pmod{2^m}, \end{aligned}$$

which, together with the fact that there exists an integer of $\mathbb{Z}_{2^m}$ whose Zech logarithm equals 2^{m-1}, implies $Z^{\{j+k(2^m+1)\}}(\hat{\gamma}) = Z^{\{j\}}(\hat{\gamma})$. Hence, we have

$$Z^{\{j\}}(\hat{\gamma}) = Z^{\{j \bmod 2^m+1\}}(\hat{\gamma}).$$

In Section 7.2.7 we saw that $Z(0)$, the Zech logarithm of zero, is involved in the computation of multiplication by powers of two. As seen below, we obtain several interesting properties of Zech's logarithms in Fermat prime fields which are related to $Z(0)$. Henceforth, each Zech's logarithm $Z(\hat{\gamma})$ in $\mathbb{Z}_{2^m+1}$ is generally considered as a jth Zech logarithm of zero, for some j. In Figure 7.1, we visualise the sequence of jth Zech's logarithms by drawing lines from $Z^{\{j\}}(0)$ to $Z^{\{j+1\}}(0)$ for $j = 0, 1, 2, 3, \ldots, 2^4$. From the figure we can derive some special properties of Zech's logarithms in $\mathbb{Z}_{2^m+1}$. This is further discussed in Appendix C.

Theorem 7.3 *Let α be a primitive element of the Fermat prime field $\mathbb{Z}_{2^m+1}$ and let $Z^{\{j\}}(0)$ be the jth Zech logarithm of zero. Also, for $i, k \in \mathbb{Z}$, let*

$$\begin{aligned} a_i &\equiv 2^i(a_0+1) - 1 \\ &\equiv 2a_{i-1} + 1 \pmod{2^m+1}; \quad a_0 \in \mathbb{Z}_{2^m} \qquad (7.24) \\ d_k &\equiv \alpha^k - 1 \\ &\equiv \alpha d_{k-1} + \alpha - 1 \pmod{2^m+1}. \qquad (7.25) \end{aligned}$$

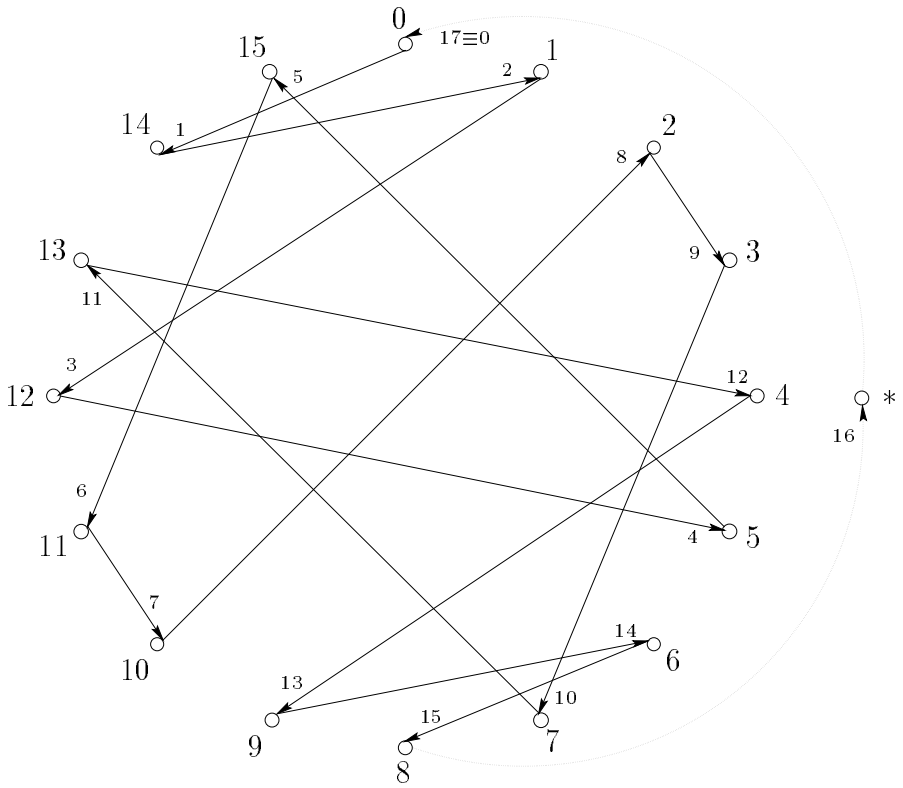


Figure 7.1: *The sequence of Zech's logarithms* $Z^{\{2^m+1\}}(0) \equiv Z^{\{0\}}(0) \equiv Z(*) \equiv 0$, $Z^{\{1\}}(0) = Z(0) = 14$, $Z^{\{2\}}(0) \equiv Z(14) \equiv 1$, $Z^{\{3\}}(0) \equiv Z(1) \equiv 12, \ldots$, $Z^{\{2^m\}}(0) \equiv Z(2^{m-1}) \equiv * \pmod{2^m}$, *for* $m = 4$.

Then we have

$$\alpha^{Z^{\{j\}}(0)} \equiv j+1 \pmod{2^m+1} \tag{7.26}$$

$$\begin{aligned} Z^{\{j\}}(0) &\equiv Z^{\{a_0\}}(0) + iZ(0) \\ &\equiv Z^{\{a_{i-1}\}}(0) + Z(0) \pmod{2^m} \end{aligned} \tag{7.27}$$

$$Z^{\{d_k\}}(0) \equiv k \pmod{2^m} \tag{7.28}$$

$$mZ(0) \equiv 2^{m-1} \pmod{2^m}. \tag{7.29}$$

Proof: The expansions of a_i and d_k in (7.24) and (7.25), respectively, follow easily from the definitions of a_i and d_k.

- Equation (7.26): By Definitions 7.2 and 7.4 we get

$$\alpha^{Z\{j\}(0)} \equiv \underbrace{1+1+\cdots+1}_{j} + \alpha^{Z\{0\}(0)} \equiv j+1 \pmod{2^m+1}.$$

- Equation (7.27): By combining (7.26) and (7.24) and using the congruence $2 \equiv \alpha^{Z(0)} \pmod{2^m+1}$ we get

$$\begin{aligned} \alpha^{Z\{a_i\}(0)} &\equiv a_i + 1 \\ &\equiv \begin{cases} (2^i(a_0+1)-1)+1 \equiv \alpha^{Z\{a_0\}(0)+iZ(0)} \pmod{2^m+1} \\ (2a_{i-1}+1)+1 \equiv \alpha^{Z\{a_{i-1}\}(0)+Z(0)} \pmod{2^m+1} \end{cases}, \end{aligned}$$

  from which we get (7.27).

- Equation (7.28): By letting $j = d_k$ in (7.26) we get

$$\alpha^{Z\{d_k\}(0)} \equiv d_k - 1 \equiv \alpha^k - 1 + 1 = \alpha^k \pmod{2^m+1},$$

  from which we get (7.28).

- Equation (7.29) was obtained on page 162 (see the congruence leading to (7.15). It is repeated here only for the sake of completeness.

□

The recursive part of (7.24) is on the same form as diminished–1 multiplication by two. Therefore, a_i can very simply be obtained from a_{i-1} using an m-bit feedback shift register with an inverter in the feedback loop (see page 127 – the last paragraph concerning multiplication by 2 – in Section 6.3.5).

Theorem 7.4 *Let $a_i \equiv 2^i(a_0+1)-1 \pmod{2^m+1}$, where $i \in \mathbb{Z}$ and $a_0 \in \mathbb{Z}_{2^m}$. Then, the sequence $\ldots, a_{i-1}, a_i, a_{i+1}, \ldots$ is cyclic with period $2m$, i.e. we have*

$$a_i \equiv a_{i \bmod 2m} \pmod{2^m+1}.$$

For $a_0 = 2^m$ and $i \in \mathbb{Z}$, we have $a_i \equiv a_0 \pmod{2^m+1}$.

Proof: The cyclic property of the sequence $\ldots, a_{i-1}, a_i, a_{i+1}, \ldots$ follows simply from the fact that the order of 2 modulo 2^m+1 equals $2m$. By letting $a_i \equiv a_0 \pmod{2^m+1}$ in (7.24) we get $a_0 \equiv 2^i(a_0+1)-1 \pmod{2^m+1}$, which implies

$$(2^i-1)(a_0+1) \equiv 0 \pmod{2^m+1}.$$

This congruence has the solutions $2^i \equiv 1 \pmod{2^m+1}$, i.e. $i \equiv 0 \pmod{2m}$, and $a_0 \equiv -1 \equiv 2^m \pmod{2^m+1}$. However, because we have $a_0 \in \mathbb{Z}_{2^m}$, the only valid solution is $i \equiv 0 \pmod{2m}$. Consequently,

$$\begin{cases} a_i \not\equiv a_0 \pmod{2^m+1}; & \text{for } 2m \nmid i \\ a_i \equiv a_0 \pmod{2^m+1}; & \text{for } 2m \mid i \end{cases}$$

□

For $k \in \mathbb{Z}$ we have $\alpha^k \not\equiv 0 \pmod{2^m+1}$ which, by (7.25), implies $d_k \not\equiv -1 \equiv 2^m \pmod{2^m+1}$ and thus $d_k \in \mathbb{Z}_{2^m}$. By Theorem 7.4 it follows that when representing each Zech's logarithm in $\mathbb{Z}_{2^m+1}$ on the form given by (7.28), i.e. as some d_kth Zech logarithm of zero, the set $\left\{Z^{\{d_k\}}(0) : k \in \mathbb{Z}_{2^m}\right\}$ can be partitioned into $2^c = 2^m/2m$ distinct cyclic groups. Each subgroup can be generated using (7.24) and (7.27) and with the knowledge of only $Z(0)$ and $Z^{\{a_0\}}(0)$, for some integer a_0 associated with the group.

The sequence $d_0, d_1, d_2, , \ldots, d_{2^m-1}$ of indices can be arranged to form a matrix $\mathcal{D}_m$ of size $2^c \times 2m$, which we define [5, Sec. 2] as

$$\mathcal{D}_m \triangleq \begin{pmatrix} d_0 & d_{2^c} & d_{2*2^c} & \cdots & d_{(2m-1)2^c} \\ d_1 & d_{1+2^c} & d_{1+2*2^c} & \cdots & d_{1+(2m-1)2^c} \\ d_2 & d_{2+2^c} & d_{2+2*2^c} & \cdots & d_{2+(2m-1)2^c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{2^c-1} & d_{2*2^c-1} & d_{3*2^c-1} & \cdots & d_{2^m-1} \end{pmatrix}, \tag{7.30}$$

where $2^c = 2^m/2m$ (see (7.17)). Thus, the matrix $\mathcal{D}_m$ is formed such that, by writing k on the form $k = \rho + \kappa 2^c$, the element d_k is in row ρ and column κ of $\mathcal{D}_m$, where $0 \le \rho \le 2^c - 1$ and $0 \le \kappa \le 2m - 1$. Because k is an m-bit NBC integer, the NBC integer ρ is in turn formed by the $c = m-t-1$ least significant bits and the NBC integer κ is formed by the $t+1$ most significant bits of k.

Theorem 7.5 *The set of $2m$ elements in row ρ of $\mathcal{D}_m$ equals the cyclic group $\{2^i(a_0+1)-1 \pmod{2^m+1} : i = 0, 1, \ldots, 2m-1\}$, where a_0 is any element of the row.*

The theorem is simply proved using the following lemma.

Lemma 7.1 *Denote by $a_i|_a$ the integer $a_i \equiv 2^i(a_0+1)-1 \pmod{2^m+1}$, where $a_0 = a$. Let $\hat{\sigma}$ be the multiplicative inverse of $\hat{\delta}$ modulo $2m$, where $\hat{\delta}$ is defined by the*

Zech logarithm $Z(0) = \hat{\delta}2^c$. *Also, let* $r \in \mathbb{Z}_{2m}$ *and* $d_k \equiv a_i|_{a_0} \pmod{2^m+1}$ *for some* k, i, *and* a_0. *Then we have*

$$d_{k+r2^c} \equiv a_{i+r\hat{\sigma}}|_{a_0} \pmod{2^m+1}. \tag{7.31}$$

Proof: It follows from (7.25) that

$$d_{k+r2^c} \equiv \alpha^{k+r2^c} - 1 = (d_k+1)(d_{r2^c}+1) - 1 \pmod{2^m+1}.$$

From the congruence $\hat{\delta}\hat{\sigma} \equiv 1 \pmod{2m}$ it follows that $2m \mid (\hat{\delta}\hat{\sigma}-1)$ and hence $2^c \cdot 2m = 2^m \mid (2^c\hat{\delta}\hat{\sigma} - 2^c) = (\hat{\sigma}Z(0) - 2^c)$, which implies $2^c \equiv \hat{\sigma}Z(0) \pmod{2^m}$. Hence

$$d_{r2^c} \equiv \alpha^{r2^c} - 1 \equiv \alpha^{r\hat{\sigma}Z(0)} - 1 \equiv 2^{r\hat{\sigma}} - 1 = a_{r\hat{\sigma}}|_0 \pmod{2^m+1},$$

and consequently

$$\begin{aligned} d_{k+r2^c} &\equiv (a_i|_{a_0}+1)(a_{r\hat{\sigma}}|_0+1) - 1 = 2^{i+r\hat{\sigma}}(a_0+1) - 1 \\ &\equiv a_{i+r\hat{\sigma}}|_{a_0} \pmod{2^m+1}. \end{aligned}$$

□

In the following proof of Theorem 7.5 we consider row ρ of $\mathcal{D}_m$, which we denote by $\mathcal{D}_{m,\rho}$.

Proof: (Theorem 7.5) Let d_k, where $k = \rho + \kappa 2^c$, be the integer in position κ of $\mathcal{D}_{m,\rho}$. Then we can write

$$\begin{aligned} \mathcal{D}_{m,\rho} &= (\, d_\rho \quad d_{\rho+2^c} \quad \cdots \quad d_{\rho+(\kappa-1)2^c} \quad d_{\rho+\kappa 2^c} \quad d_{\rho+(\kappa+1)2^c} \quad \cdots \quad d_{\rho+(2m-1)2^c}\,) \\ &= (\, d_{k-\kappa 2^c} \quad d_{k-(\kappa-1)2^c} \quad \cdots \quad d_{k-2^c} \quad d_k \quad d_{k+2^c} \quad \cdots \quad d_{k-(\kappa+1)2^c}\,). \end{aligned}$$

Let $d_k \equiv a_i|_{a_0} \pmod{2^m+1}$ for some $i \in \mathbb{Z}_{2m}$ and $a_0 \in \mathbb{Z}_{2^m}$. Because $\gcd(\hat{\sigma}, 2m) = 1$, the set $\{a_{i+r\hat{\sigma}}|_{a_0};\ r \equiv 0, 1, \ldots, 2m-1 \pmod{2m}\}$ forms the cyclic group of order $2m$ which contains $a_i|_{a_0}$. Hence, by Lemma 7.1 (Equation (7.31)), $\mathcal{D}_{m,\rho}$ can be written on the form

$$(\, a_{i-\kappa\hat{\sigma}}|_{a_0} \quad a_{i-(\kappa-1)\hat{\sigma}}|_{a_0} \quad \cdots \quad a_{i-\hat{\sigma}}|_{a_0} \quad a_i|_{a_0} \quad a_{i+\hat{\sigma}}|_{a_0} \quad \cdots \quad a_{i-(\kappa+1)\hat{\sigma}}|_{a_0}\,),$$

where a_0 is the integer in column $\kappa - \hat{\delta}i \bmod 2m$. (Since $a_i|_{a_0}$ is in column κ, the integer a_0 is in column $\kappa + r \bmod 2m$, where r is obtained from the congruence $i + r\hat{\sigma} \equiv 0 \pmod{2m}$. Thus, we have $r \equiv -\hat{\sigma}^{-1}i \equiv -\hat{\delta}i \pmod{2m}$.) □

Henceforth, for each row of $\mathcal{D}_m$, we generally let a_0 be the element in the first (leftmost) position of the row. Thus, for the row vector $\mathcal{D}_{m,\rho}$ we have $a_0 = d_\rho$.

In particular, in the first row $\mathcal{D}_{m,0}$ of $\mathcal{D}_m$ we get $a_0 = d_0 \equiv 0 = (000\cdots000)_2 \pmod{2^m+1}$. By (7.24), the remaining elements of the row are $a_1|_0 = (000\cdots001)_2$, $a_2|_0 = (000\cdots011)_2$, ..., $a_{m-1}|_0 = (011\cdots111)_2$, $a_m|_0 = (111\cdots111)_2$, $a_{m+1}|_0 = (111\cdots110)_2$, $a_{m+2}|_0 = (111\cdots100)_2$, ..., $a_{2m-1}|_0 = (100\cdots000)_2$. Hence, *for $0 \leq i \leq m-1$, the NBC integer $a_i|_0$ is formed by a block of $m-i$ zeros followed by a block of i ones. For $m \leq i \leq 2m-1$, $a_i|_0$ is formed by a block of $2m-i$ ones followed by a block of $i-m$ zeros*. Let d be an arbitrary NBC integer of $\mathcal{D}_{m,0}$. Consequently, for $a_i|_0 = e$, the subscript i is simply obtained as

$$i = \overline{e_0}m + n_d, \tag{7.32}$$

where e_0 is the least significant bit of e and n_e is the number of bits of e which are equal to e_0. For example, for $m = 8$ we have $e = (00111111)_2 = a_i|_0$, where $i = 0 \cdot 8 + 6 = 6$ and $e = (10000000)_2 = a_i|_0$, where $i = 1 \cdot 8 + 7 = 15$.

So far, we have not said much about the multiplicative inverse $\hat{\sigma}$ of $\hat{\delta}$ modulo $2m$. In Table 7.1 we have listed the parameters $Z(0)$, $\hat{\delta}$, and $\hat{\sigma}$ for $m = 2, 4, 8, 16$, with respect to the primitive element $\alpha = 3$. By definition, we have $\hat{\sigma} \equiv \hat{\delta}^{-1} \pmod{2m}$, where $\hat{\delta}$ is defined by $Z(0) = \hat{\delta}2^c \equiv \log_\alpha 2 \pmod{2^m}$. Regarding $\hat{\delta}$ we have observed the following property.

Observation 7.1 *For $m = 4, 8, 16$ and $\alpha = 3$ we can write $\hat{\delta}$ on the form*

$$\hat{\delta} \equiv m + 11 \pmod{2m}. \tag{7.33}$$

For $m = 2$ we simply have $\hat{\delta} = \hat{\sigma} = 3$. We have not been able to show whether the fact that Observation 7.1 holds can be derived from the definition of $\hat{\delta}$ or if it is just some kind of coincidence. Anyhow, a consequence of (7.33) of Observation 7.1 is the following theorem.

Theorem 7.6 *For $m = 4, 8, 16$, when the primitive element α equals 3, the multiplicative inverse $\hat{\sigma}$ of $\hat{\delta}$ modulo $2m$ can be written on the form*

$$\hat{\sigma} \equiv m + 3 \pmod{2m}. \tag{7.34}$$

Proof: By Observation 7.1, for $m = 4, 8, 16$ and $\alpha = 3$ we have $\hat{\delta} \equiv m + 11 \pmod{2m}$. The congruence

$$\hat{\delta}(m+11) \equiv (m+11)(m+3) = 2m\left(\frac{m}{2} + 7 + \frac{16}{m}\right) + 1 \equiv 1 \pmod{2m}$$

holds for $m = (2,)\, 4, 8, 16$. Hence, the multiplicative inverse of $\hat{\delta} \equiv m + 11$ modulo $2m$ equals $m + 3$. □

m	$Z(0)$	$\hat{\delta}$	$\hat{\sigma}$
2	3	3	3
4	14	7	7
8	48	3	11
16	55296	27	19

Table 7.1: *The parameters $Z(0)$, $\hat{\delta}$, and $\hat{\sigma}$ for $m = 2, 4, 8, 16$ when the primitive element α equals 3.*

7.4.1 Discrete Exponentiation

The properties of the matrix $\mathcal{D}_m$ derived in the previous section can be utilised to perform exponentiation and compute the discrete logarithm.

Theorem 7.7 *Let $P(\gamma) = \hat{\gamma} \in \mathbb{Z}_{2^m}$ be on the form $\hat{\gamma} = \rho + \kappa 2^c$, where ρ and κ are c-bit and $(m-c)$-bit NBC integers, respectively. Then, the discrete exponentiation $\gamma \equiv \alpha^{\hat{\gamma}} \pmod{2^m+1}$ can be performed by first deriving the integer $d_{\hat{\gamma}}$ in position (ρ, κ) of $\mathcal{D}_m$ and then computing $\gamma = d_{\hat{\gamma}} + 1$.*

Proof: For $\hat{\gamma} \in \mathbb{Z}_{2^m}$, γ is a nonzero integer of $\mathbb{Z}_{2^m+1}$. By (7.1), (7.25) and (7.28) we then have

$$\alpha^{\hat{\gamma}} \equiv 1 + d_{\hat{\gamma}} = \gamma \pmod{2^m+1}, \tag{7.35}$$

where $\hat{\gamma} \equiv Z^{\{d_{\hat{\gamma}}\}}(0) \pmod{2^m}$. From the definition of the matrix $\mathcal{D}_m$ in (7.30) we know that for a given $\hat{\gamma} = \rho + \kappa 2^c$, the associated integer $d_{\hat{\gamma}}$ is located in row ρ and column κ of $\mathcal{D}_m$. After finding this integer $d_{\hat{\gamma}}$ we get, from (7.35), $\gamma = d_{\hat{\gamma}} + 1$. Because $0 \leq d_{\hat{\gamma}} \leq 2^m - 1$ we have $d_{\hat{\gamma}} + 1 \in \mathbb{Z}_{2^m+1}$, i.e. no modulus reduction is needed when computing γ from $d_{\hat{\gamma}}$. □

The computational complexity of performing discrete exponentiation according to the procedure described in Theorem 7.7 mainly depends on the complexity of obtaining $d_{\hat{\gamma}}$ from $\hat{\gamma} = \rho + \kappa 2^c$.

The discrete exponentiation $\gamma \equiv \alpha^{\hat{\gamma}} \pmod{2^m+1}$ can be computed in the following way:

1. The first step is to compute $d_{\kappa 2^c}$. By letting $a_0 = d_0 = 0$ it follows, from Lemma 7.1 (Equation (7.31)), that $d_{\kappa 2^c} \equiv a_{\kappa\hat{\sigma}}\,|_0 \pmod{2^m+1}$. Let $i \equiv \kappa\hat{\sigma} \pmod{2m}$. The NBC integer $a_i\,|_0$ is preferably computed in either of the following two ways:

(a) In the paragraph subsequent to the proof of Theorem 7.5, we described how the elements of $\mathcal{D}_{m,0}$, i.e. the top row of $\mathcal{D}_m$, are formed. Consequently, if $0 \leq i \leq m-1$ we let $a_i\,|_0 = \left(0^{(m-i)}\,1^{(i)}\right)_2$ and if $m \leq i \leq 2m-1$ we let $a_i\,|_0 = \left(1^{(2m-i)}\,0^{(i-m)}\right)_2$.

(b) As mentioned in the paragraph subsequent to the proof of Theorem 7.3, $a_i\,|_0$ can simply be recursively computed in i clock cycles using a feedback shift register of length m and with an inverter in the feedback loop. This computational procedure is based on the recursive form of a_i in (7.24).

2. The second step is to recursively compute $d_{\hat{\gamma}} = d_{\rho+\kappa 2^c}$ from $d_{\kappa 2^c} = a_i\,|_0$. Thus, we compute $d_{1+\kappa 2^c}$ from $d_{\kappa 2^c}$, $d_{2+\kappa 2^c}$ from $d_{1+\kappa 2^c}$, etc., until, after ρ steps, $d_{\hat{\gamma}} = d_{\rho+\kappa 2^c}$ is computed from $d_{\rho-1+\kappa 2^c}$. In each computation step, d_k is computed from d_{k-1} using the recursive congruence $d_k \equiv \alpha d_{k-1} + \alpha - 1 \pmod{2^m+1}$ in (7.25).

3. In the third step we compute the desired result $\gamma = d_{\hat{\gamma}} + 1$.

In Figure 7.2 we illustrate which parts of the matrix $\mathcal{D}_m$ are associated with the above Steps 1 and 2 of the procedure for performing discrete exponentiation. The complexity of computing the recursive congruence $d_k \equiv \alpha d_{k-1} + \alpha - 1 \pmod{2^m+1}$ strongly depends on which primitive element α is chosen. By Theorem 2.5 of Section 2.3.2, the integer 3 is a primitive element of every Fermat prime field $\mathbb{Z}_{2^m+1}$; $m \geq 2$. If the primitive element α equals 3, we get

$$\begin{aligned} d_k &\equiv 3d_{k-1} + 2 = (2d_{k-1} + 1) + d_{k-1} + 1 \\ &\equiv (2d_{k-1} + 1) \oplus d_{k-1} \pmod{2^m+1}, \end{aligned} \tag{7.36}$$

where $2d_{k-1} + 1 \pmod{2^m+1}$ is equivalent to diminished–1 multiplication by 2 and where $\oplus$ denotes diminished–1 addition. Figure 7.8 in Section 7.6.1 show how to compute (7.36) using a feedback parallel adder.

Remark: Step 2 (computations along a column) may be carried out prior to Step 1(b) (computations along a row) as follows: Firstly, d_ρ is recursively computed from d_0 as in Step 2 using (7.25) (which for $\alpha = 3$ is equivalent to (7.36)). Secondly, by letting $a_0 = d_\rho$, the integer $d_{\hat{\gamma}} \equiv a_i\,|_0 \pmod{2^m+1}$ is recursively computed as in Step 1(b) using (7.24).

Theorem 7.8 *Let the matrix $\mathcal{D}_m$ be written on the form*

$$\mathcal{D}_m = \left(\mathcal{D}_m^{(1)} \,\middle|\, \mathcal{D}_m^{(2)}\right),$$

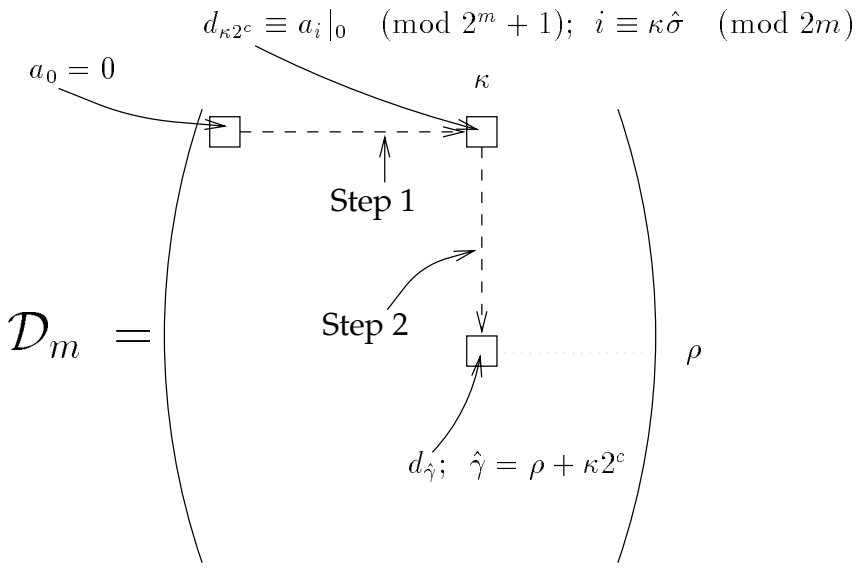


Figure 7.2: *The computation steps for performing discrete exponentiation using properties of the matrix $\mathcal{D}_m$. Equations* (7.24) *and* (7.36) *are used in Step 1 and 2, respectively.*

where $\mathcal{D}_m^{(1)}$ and $\mathcal{D}_m^{(2)}$ are formed respectively by the m first and m last columns of $\mathcal{D}_m$. Also, let $\overline{\mathcal{D}_m^{(1)}}$ denote the matrix obtained when exchanging each (m-bit) integer of $\mathcal{D}_m^{(1)}$ for its one's complement. Then

$$\mathcal{D}_m^{(2)} = \overline{\mathcal{D}_m^{(1)}}.$$

In the proof of Theorem 7.8 we use the following properties: Because the multiplicative inverse $\hat{\delta}$ of $\hat{\sigma}$ modulo $2m$ is odd, it follows that $\hat{\sigma}$ is also odd, i.e. we have $\hat{\sigma} = 2d + 1$ for some nonnegative integer d. Hence, by Theorem 7.4 and (7.24) we get

$$\begin{aligned} a_{i+m\hat{\sigma}}|_{a_0} &\equiv a_{i+2m\cdot d+m}|_{a_0} \equiv a_{i+m \bmod 2m}|_{a_0} \\ &\equiv 2^{i+m}(a_0+1) - 1 \equiv 2^m - 1 - (2^i(a_0+1) - 1) \\ &\equiv \overline{a_i}|_{a_0} \pmod{2^m+1}, \end{aligned} \tag{7.37}$$

where $\overline{a_i}|_{a_0}$ is the one's complement of the m-bit integer $a_i|_{a_0}$, for any $a_0 \in \mathbb{Z}_{2^m}$.

Proof: (Theorem 7.8) Let $a_i|_{a_0}$ be the m-bit integer in an arbitrary position of $\mathcal{D}_m^{(1)}$. Then, by (7.31) in Lemma 7.1 and the definition of $\mathcal{D}_m^{(2)}$, $a_{i+m\hat{\sigma}}|_{a_0}$ is the m-bit integer in the corresponding position of $\mathcal{D}_m^{(2)}$. Because the congruence in (7.37) holds for every integer $a_i|_{a_0}$ of $\mathcal{D}_m^{(1)}$, we have $\mathcal{D}_m^{(2)} = \overline{\mathcal{D}_m^{(1)}}$. □

As it is described on page 174, the computation of $a_i|_0$ from $a_0 = 0$ in Step 1(b) requires at most $2m - 1$ clock cycles. One binary shift (rotation) is performed during each clock cycle. We have $i \equiv \kappa\hat{\sigma} \pmod{2m}$, which implies $i \leq 2m-1$. In consequence of Theorem 7.8, when $i \geq m$, $a_i|_0$ can be obtained in at most $m-1$ clock cycles: If $i \geq m$, let $i = j + m$ for some integer $j \in \mathbb{Z}_m$. From (7.37), after j clock cycles we then get $a_i|_0 \equiv \overline{a_j}\,|_{a_0} \pmod{2^m + 1}$.

Another way of reducing the number of shifts required in Step 1(b) is the following: Let $l = 2m - i$. Because the sequence $\{a_i|_{a_0}\}_{i\in\mathbb{Z}}$ is cyclic with period $2m$, we have $a_i|_{a_0} \equiv a_{2m-l}|_{a_0} \equiv a_{-l}|_{a_0} \pmod{2^m + 1}$. Hence, for $0 \leq i \bmod 2m \leq m$, $a_i|_{a_0}$ can be obtained by rotating the m-bit NBC integer $a_0|_{a_0}$ i bits to the left (in the increased significant bits direction). As before, there is an inverter in the feedback loop. For $m + 1 \leq i \bmod 2m \leq 2m - 1$, which implies $1 \leq l \bmod 2m \leq m - 1$, $a_i|_{a_0} \equiv a_{-l}|_{a_0} \pmod{2^m + 1}$ is obtained by rotating $a_0|_{a_0}$ l bits in the opposite direction (to the right). The bits in the feedback loop are inverted. This procedure requires either two feedback shift registers or just one register which can rotate its contents in both directions. In any case, the maximum number of shifts is m.

We preferably state the computational complexity of an algorithm in terms of the number of additions required to perform the algorithm. Here, all additions are carried out modulo $2^m + 1$. We assume that the i binary shifts of Step 1(b) (where i can be maximised to $m - 1$) can be carried out as fast as one addition. With $\alpha = 3$, the above Step 2 requires at most $2^c = 2^m/2m$ additions and about half as many additions in average. Step 3 can be carried out using a simplified adder.[6]

Consequently, the algorithm for performing discrete exponentiation described in this section can be performed using at most $2^c + 2 = 2^m/2m + 2$ additions (modulo $2^m + 1$). The algorithm requires about $2^{c-1} + 2 = 2^m/4m + 2$ additions modulo $2^m + 1$ in average.

As mentioned in Section 7.2.1, the binary method for discrete exponentiation requires at most $2(m - 1)$ multiplications. Assuming that a binary multiplication is computed using at most m additions, the binary method requires at most $2m(m-1)$ additions. The average number of additions required is about $m(m - 1 + m/2) = m(3m - 2)/2$.

[6] In Figure 6.4 of Section 6.3.1, we see that addition by one can simply be carried out using a row of m cascaded half adder elements.

7.4.2 The Discrete Logarithm

Theorem 7.9 *Let γ be a nonzero integer of $\mathbb{Z}_{2^m+1}$ and $d_{\hat{\gamma}} = \gamma - 1$, where $\hat{\gamma} = P(\gamma)$. Then, the discrete logarithm $\hat{\gamma} \equiv \log_\alpha \gamma \pmod{2^m}$ can be obtained by first finding the position (ρ, κ) in $\mathcal{D}_m$ where $d_{\hat{\gamma}}$ is located and then forming $\hat{\gamma}$ as $\hat{\gamma} = \rho + \kappa 2^c$.*

Proof: For $1 \leq \gamma \leq 2^m + 1$, the integer $d_{\hat{\gamma}} = \gamma - 1$ is an element of $\mathbb{Z}_{2^m}$. Then, by (7.1), (7.25) and (7.28) we have

$$\alpha^{\hat{\gamma}} \equiv 1 + d_{\hat{\gamma}} = \gamma \pmod{2^m + 1},$$

where $\hat{\gamma} \equiv Z^{\{d_{\hat{\gamma}}\}}(0) = \rho + \kappa 2^c \pmod{2^m}$. Also, from the definition of the matrix $\mathcal{D}_m$ in (7.30) we know that the integer $d_{\hat{\gamma}}$, which is associated with $\hat{\gamma}$, is located in row ρ and column κ of $\mathcal{D}_m$. Hence, by finding the position (ρ, κ) we directly obtain the desired discrete logarithm $\hat{\gamma}$. □

By Theorem 7.9, the problem of computing the discrete logarithm $\hat{\gamma} \equiv \log_\alpha \gamma \pmod{2^m}$ is equivalent to the problem of finding the position (ρ, κ) in $\mathcal{D}_m$ where $d_{\hat{\gamma}}$ is located. One way of finding this position is to compute $d_{\hat{\gamma}+1}$, $d_{\hat{\gamma}+2}$, $d_{\hat{\gamma}+3}$, etc., using (7.36)[7] until, for some i, the integer $d_{(\kappa+1)2^c} \equiv a_i|_0 \pmod{2^m + 1}$ of the top row $\mathcal{D}_{m,0}$ of $\mathcal{D}_m$ is obtained. As described on page 172, each NBC integer of $\mathcal{D}_{m,0}$ is formed either by a block of zeros followed by a block of ones or vice versa. Hence, for $j = 1, 2, 3, \ldots$, the recursive computation of $d_{\hat{\gamma}+j}$ from $d_{\hat{\gamma}+j-1}$ progress until such a binary word is detected. By (7.32), the subscript i of $a_i|_0$ equals

$$i = \overline{e_0} m + n_e,$$

where e_0 is the least significant bit of $a_i|_0$ and n_e is the number of bits in $a_i|_0$ which are equal to e_0. Because the integer $d_{(\kappa+1)2^c} \equiv a_i|_0 \pmod{2^m + 1}$ is in column $\kappa + 1$ of $\mathcal{D}_m$, it follows from Lemma 7.1 (Equation (7.31)) that $i \equiv (\kappa + 1)\hat{\sigma} \pmod{2m}$. Consequently, $d_{\hat{\gamma}}$ is in column

$$\kappa \equiv i\hat{\sigma}^{-1} - 1 \equiv i\hat{\delta} - 1 \pmod{2m}$$

of $\mathcal{D}_m$. The row position ρ is obtained from the number of recursions. This gives the desired discrete logarithm $\hat{\gamma} = \rho + \kappa 2^c$.

The above procedure for computing the discrete logarithm $\hat{\gamma} = P(\gamma)$ is summarised in the following algorithm.

1. Let $d = \gamma - 1$ and $j = 2^c$.

[7] Or in general Equation (7.25). However, as in Section 7.4.1, by choosing $\alpha = 3$ the recursive congruence in (7.25) changes to (7.36).

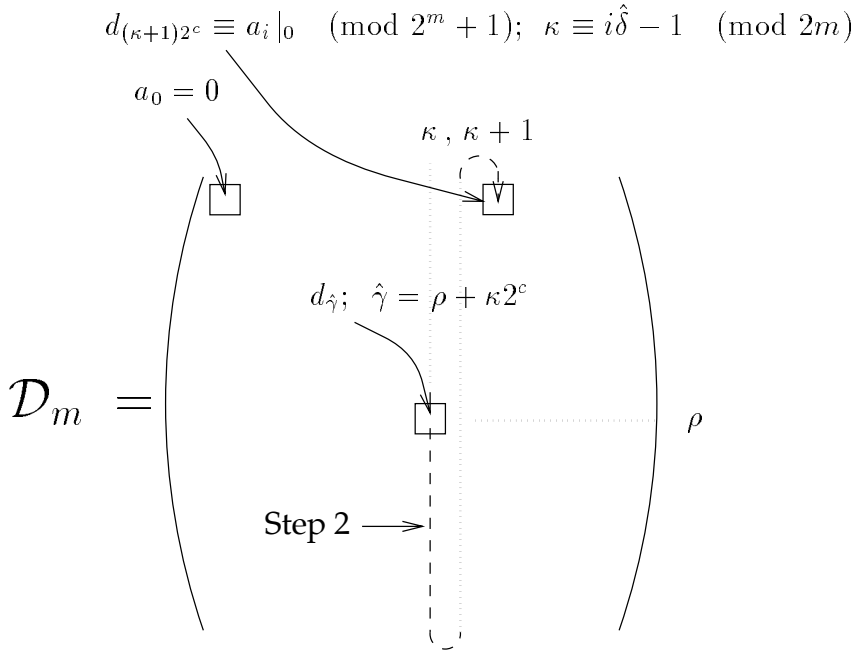


Figure 7.3: *The procedure for computing the discrete logarithm using properties of the matrix $\mathcal{D}_m$. Equation* (7.36) *is used in Step 2.*

2. If $d \in \mathcal{D}_{m,0}$, goto Step 3.
   Otherwise, let $j = j - 1$, compute the next d using the recursive congruence in (7.36) (i.e. $d_{\text{next}} = (2d + 1) \oplus d \mod 2^m + 1$), and goto Step 2.

3. Let $\rho \equiv j \pmod{2^c}$.
   Compute the subscript $i = \overline{e_0}m + n_e$ of $a_i|_0 = d$.
   Also, let $\kappa \equiv i\hat{\delta} - 1 \pmod{2m}$.
   Then we have $\hat{\gamma} = \rho + \kappa 2^c$.

In Figure 7.3 we illustrate which elements of the $\mathcal{D}_m$ matrix are the computed in the above algorithm.

The initial value of d is computed in Step 1 using one (simplified) addition. In Step 2 we assume that the computation of d and the checking whether d is an element of $\mathcal{D}_{m,0}$ are concurrent operations.[8] Then, in total at most $2^c - 1$ additions modulo $2^m + 1$ are required in Step 2. Finally, assuming that the computational complexity of the derivation of κ in Step 3 equals the complexity

[8] This may be possible only if d_{next} is computed using a carry ripple diminished–1 adder.

of performing one addition modulo $2^m + 1$, the complete algorithm presented above for computing the discrete logarithm requires at most $2^c+1 = 2^m/2m+1$ additions modulo $2^m + 1$. About $2^m/4m + 1$ additions are required on average.

7.4.3 Zech's Logarithm

Theorem 7.10 *Let $\hat{\theta} = \rho_{\hat{\theta}} + \kappa_{\hat{\theta}} 2^c$ for some integers $\rho_{\hat{\theta}}$ and $\kappa_{\hat{\theta}}$. The Zech logarithm of $\hat{\theta}$ can be obtained by first finding the integer $d_{\hat{\theta}}$, which is in position $(\rho_{\hat{\theta}},\ \kappa_{\hat{\theta}})$ of $\mathcal{D}_m$, and then finding the position $(\rho_{\hat{\gamma}},\ \kappa_{\hat{\gamma}})$ of $\mathcal{D}_m$ where $d_{\hat{\gamma}} = d_{\hat{\theta}} + 1$ is located. Then we have $Z(\hat{\theta}) = \rho_{\hat{\gamma}} + \kappa_{\hat{\gamma}} 2^c$.*

Proof: By taking Zech's logarithm on both sides of (7.28) and letting $k = \hat{\theta}$ we get

$$Z(\hat{\theta}) \equiv Z^{\{d_{\hat{\theta}}+1\}}(0) \pmod{2^m}. \tag{7.38}$$

Let $d_{\hat{\gamma}} = d_{\hat{\theta}} + 1$. Then, again by (7.28) it follows that $Z^{\{d_{\hat{\theta}}+1\}}(0)$ in (7.38) equals the subscript $\hat{\gamma} = \rho_{\hat{\gamma}} + \kappa_{\hat{\gamma}} 2^c$ of $d_{\hat{\gamma}}$. Consequently, we have $Z(\hat{\theta}) = \rho_{\hat{\gamma}} + \kappa_{\hat{\gamma}} 2^c$ where, by the definition of the matrix $\mathcal{D}_m$ in (7.30), $(\rho_{\hat{\gamma}},\ \kappa_{\hat{\gamma}})$ is the position in $\mathcal{D}_m$ where $d_{\hat{\gamma}} = d_{\hat{\theta}}+1$ is located. Again by the definition of $\mathcal{D}_m$, for $\hat{\theta} = \rho_{\hat{\theta}}+\kappa_{\hat{\theta}} 2^c$, $d_{\hat{\theta}}$ is the integer located in position $(\rho_{\hat{\theta}},\ \kappa_{\hat{\theta}})$ of $\mathcal{D}_m$. □

From Theorem 7.10 it follows that, using properties of the matrix $\mathcal{D}_m$, we need both the procedure in Section 7.4.1 for discrete exponentiation and the procedure in Section 7.4.2 for the discrete logarithm in order to compute Zech's logarithms. This should be compared with the direct computation of the Zech logarithm, as expressed in (7.10), which also requires one discrete exponentiation and one discrete logarithm (and one addition by one) over $\mathbb{Z}_{2^m+1}$.

Consequently, in any case we need one discrete exponentiation, one addition by one, and one discrete logarithm in order to compute a Zech logarithm. The number of additions modulo $2^m + 1$ required for performing discrete exponentiation and computing the discrete logarithm are given in the end of Sections 7.4.1 and 7.4.2. In Table 7.2 we have listed these complexity numbers together with the resulting number of additions required for computing Zech's logarithm. For comparison, we have also listed the number of additions required when using the binary method for performing exponentiation and Pohlig-Hellman's algorithm for computing the discrete logarithm. These algorithms are described in Sections 7.2.1 (and 7.4.1) and 7.2.2.

In Figure 7.4 we have plotted the number of additions modulo $2^m + 1$ required to perform discrete exponentiation ("Exp") and compute the discrete loga-

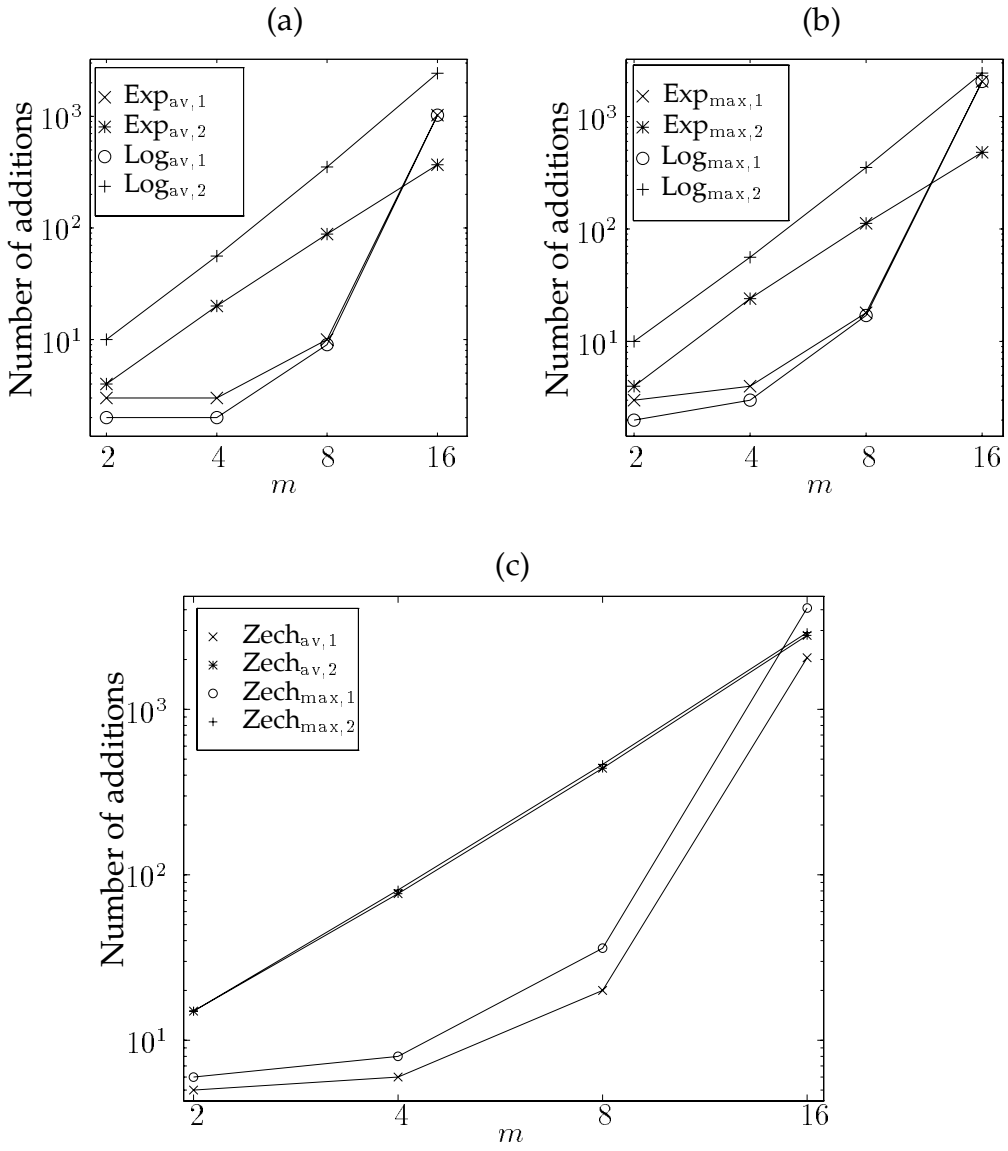


Figure 7.4: *The number of additions modulo $2^m + 1$ required to perform discrete exponentiation and compute the discrete logarithm and Zech's logarithm, with respect to different algorithms (see Table 7.2). The functions are plotted versus m for $m = 2, 4, 8, 16$.*

Operation	Algorithm	Average	Maximum
Discrete expon.	Algorithm in Sec. 7.4.1	$\frac{2^m}{4m} + 2$	$\frac{2^m}{2m} + 2$
	The binary method	$\frac{m(3m-2)}{2}$	$2m(m-1)$
Discrete logarithm	Algorithm in Sec. 7.4.2	$\frac{2^m}{4m} + 1$	$\frac{2^m}{2m} + 1$
	Pohlig-Hellman (P-H)	$\frac{m^2(m+3)}{2}$	$\frac{m^2(m+3)}{2}$
Zech's logarithm	Alg.s in Sec.s 7.4.1 & 7.4.2	$\frac{2^m}{2m} + 4$	$\frac{2^m}{m} + 4$
	Binary method & P-H	$\frac{m^3+6m^2-2m+2}{2}$	$\frac{m^3+7m^2-4m+2}{2}$

Table 7.2: *The average and maximum number of additions modulo $2^m + 1$ required to perform discrete exponentiation and compute the discrete logarithm and the Zech logarithm, with respect to the algorithms in Sections* 7.4.1 *and* 7.4.2 *and with respect to the binary method for exponentiation and Pohlig-Hellman's algorithm for computing the discrete logarithm.*

rithm ("Log") and Zech's logarithm ("Zech"). The functions plotted are the ones in Table 7.2. The subscript "1" refers to the algorithms in Sections 7.4.1 and 7.4.2 for performing exponentiation and computing the discrete logarithm, respectively. The subscript "2" refers to the binary method for exponentiation and to Pohlig-Hellman's algorithm for computing the discrete logarithm.

Figure 7.4(a) shows the average ("av") number of additions required in the exponentiation and discrete logarithm algorithms. Figure 7.4(b) shows the maximum ("max") number of additions required in the algorithms. We see that in general, the algorithms presented in Sections 7.4.1 and 7.4.2 are superior to the binary method and Pohlig-Hellman's algorithm, respectively. However, for $m = 16$ the binary method requires less additions than the algorithm in Section 7.4.1.

Figure 7.4(c) shows both the average and maximum number of additions required to compute Zech's logarithm when using ("1") the algorithms in Sections 7.4.1 and 7.4.2 and ("2") the binary method and Pohlig-Hellman's algorithm. Again, we conclude that in general the least number of additions are required when using the algorithms in Sections 7.4.1 and 7.4.2. However, because for the number of additions required by the binary method is relatively small $m = 16$ (see the previous paragraph), the maximum number of additions required by our algorithms for $m = 16$ is greater than the number of

additions required by the binary method together with Pohlig-Hellman's algorithm.

7.5 The Mirror Sequence $\mathcal{M}_m$

From Figure 7.4 we conclude that, for $m \leq 8$, the computational complexities (in terms of the required number of additions) of performing discrete exponentiation and computing the discrete logarithm and the Zech logarithm using the algorithms proposed in the previous section are relatively low. For $m = 16$, however, no significant reduction of the computational complexities are made, compared with conventional algorithms.

The number of additions required when performing a discrete exponentiation and computing a discrete logarithm is at most about $2^c = 2^m/2m$ for each operation. These additions derive in both cases from the recursive computation of d_k from d_{k-1} for some k. The mentioned additions along some column of the matrix $\mathcal{D}_m$ can be avoided by using two look-up tables – one for exponentiation and one for the discrete logarithm.

7.5.1 Discrete Exponentiation Using a Look-Up Table

When using a look-up table, we can define an algorithm for performing discrete exponentiation, based on the algorithm proposed in Section 7.4.1 (see page 174), in the following way.

The table used has size $2^c \times m$ bits and it contains the integers from the leftmost column of the matrix $\mathcal{D}_m$: For $0 \leq \rho \leq 2^c - 1$, location ρ of the table contains d_ρ. For $\hat{\gamma} = \rho + \kappa 2^c \in \mathbb{Z}_{2^m}$, the integer $\gamma = 1 + d_{\hat{\gamma}} \equiv \alpha^{\hat{\gamma}} \pmod{2^m + 1}$ can be computed in the following way:

1. Let $a_0 = d_\rho$, where d_ρ is obtained from the look-up table at location ρ.
2. By Lemma 7.1 (Equation (7.31)) we have $d_{\hat{\gamma}} = d_{\rho+\kappa 2^c} \equiv a_i|_{d_\rho} \pmod{2^m + 1}$,where $i \equiv \kappa\hat{\sigma} \pmod{2m}$. Thus, by loading an m-bit feedback shift register (which has an inverter in the feedback loop) with d_ρ and rotating the contents of the register i steps, the resulting contents of the register equals $d_{\hat{\gamma}}$.
3. The desired result $\gamma = d_{\hat{\gamma}} + 1$ is obtained simply by adding a one to $d_{\hat{\gamma}}$.

As described in Section 7.4.1, it is possible to bound the maximum number of required shifts in the above Step 2 to $m-1$. Consequently, *we can perform a discrete exponentiation using one table look-up followed by at most $m-1$ binary shifts and an addition by one*. Hence, the complexity of performing a discrete exponentiation can be considerably reduced, compared with the computational complexity obtained when using the procedure described in Section 7.4.1. This holds particularly for exponentiation in $\mathbb{Z}_{2^{16}+1}$, i.e. for $m = 16$. The reduced computational complexity is achieved to the cost of the look-up table of size $2^m/2m \times m$ bits.

7.5.2 The Discrete Logarithm Using a Look-Up Table

When computing a discrete logarithm $\hat{\gamma} \equiv \log_\alpha \gamma \pmod{2^m}$ using the algorithm proposed in Section 7.4.2, the most demanding part of the algorithm is the procedure for finding the row ρ of the matrix $\mathcal{D}_m$ in which $d_{\hat{\gamma}}$ is located. Depending on γ, this procedures may require up to $2^c = 2^m/2m - 1$ additions modulo $2^m + 1$. However, these additions can be avoided by using a look-up table.

The look-up table considered here contains a number of m-bit subscripts $\tilde{k}$ of $d_{\tilde{k}}$. For each c-bit integer $\tilde{\rho} \in \mathbb{Z}_{2^c}$, there is at least one $(m-c)$-bit integer $\tilde{\kappa} \in \mathbb{Z}_{2^{m-c}}$ such that $\tilde{k} = \tilde{\rho} + \tilde{\kappa}2^c$ is stored in table. In other words, there is *at least one* integer $d_{\tilde{k}}$ in each row ($\mathcal{D}_{m,\tilde{\rho}}$) of $\mathcal{D}_m$ for which its subscript $\tilde{k}$ is stored in the table.

Notation 7.1 *We denote by $\tilde{\Pi}$ the set of integers $d_{\tilde{k}}$ whose respective subscripts $\tilde{k}$ form the contents of the look-up table used for computing the discrete logarithm.*

A table of minimum size, i.e. whose size equals $2^c \times m$ bits, where $c = m - \log_2 m - 1$, is formed in such a way that its associated set $\tilde{\Pi}$ only contains one element from each row of $\mathcal{D}_m$. Hence, for each row of $\mathcal{D}_m$ we preferably would like to find *one* suitable such element $d_{\tilde{k}}$ that in a simple way maps to a unique entry of the look-up table. This table, which performs a one-to-one mapping from $d_{\tilde{k}}$ to $\tilde{k}$, is in a sense some kind of inverse table of the above table used for performing discrete exponentiation. However, each integer d_ρ stored in the table for exponentiation originates from some row position ρ in the leftmost column of $\mathcal{D}_m$, while the various integers $d_{\tilde{k}}$ (for which $\tilde{k}$ is stored in the table) used here may originate from an *arbitrary* column position in $\mathcal{D}_m$.

The discrete logarithm $\hat{\gamma} = \rho + \kappa 2^c$ of a nonzero integer $\gamma \equiv \alpha^{\hat{\gamma}} \pmod{2^m + 1}$ can be computed using the look-up table of subscripts $\tilde{k}$ in the following way:

We know that the integer $d_{\hat{\gamma}} = \gamma - 1$ is in column κ and row ρ of $\mathcal{D}_m$. Starting from the integer $a_0 = d_{\hat{\gamma}}$, we use (7.24) to successively calculate $a_1|_{d_{\hat{\gamma}}}$, $a_2|_{d_{\hat{\gamma}}}$, . . ., etc., until after i successions we obtain an integer $a_i|_{d_{\hat{\gamma}}}$ which is an element of $\tilde{\Pi}$. The desired logarithm $\hat{\gamma} = \rho + \kappa 2^c$ can now be formed using the associated table output $\tilde{k} = \tilde{\rho} + \tilde{\kappa} 2^c$, which is the subscript of $d_{\tilde{k}} = a_i|_{d_{\hat{\gamma}}}$. Because $d_{\hat{\gamma}}$ and $d_{\tilde{k}}$ are in the same row of $\mathcal{D}_m$ we get $\rho = \tilde{\rho}$. We therefore have

$$d_{\tilde{k}} \equiv d_{\hat{\gamma}+j2^c} = d_{\rho+(\kappa+j)2^c} \pmod{2^m + 1},$$

for some integer j, and hence $\tilde{\kappa} \equiv \kappa + j \pmod{2m}$, i.e. $\kappa \equiv \tilde{\kappa} - j \pmod{2m}$. Because we also have $d_{\tilde{k}} = a_i|_{d_{\hat{\gamma}}}$ it follows by (7.31) that $i \equiv j\hat{\sigma} \pmod{2m}$, i.e. $j \equiv i\hat{\delta} \pmod{2m}$, where $\hat{\delta}$ is the multiplicative inverse of $\hat{\sigma}$ modulo $2m$. Consequently, we obtain $\kappa \equiv \tilde{\kappa} - i\hat{\delta} \pmod{2m}$.

Thus, the above procedure for computing the discrete logarithm $\hat{\gamma} \equiv \log_\alpha \gamma \pmod{2^m}$ using a look-up table can be summarised in the following algorithm.

1. Let $a_0 = \gamma - 1$ and $i = 0$.

2. If $a_i|_{a_0} \in \tilde{\Pi}$, goto Step 3.
   Otherwise, let $i_{\text{next}} = i + 1$, compute $a_i|_{a_0}$ from $a_{i-1}|_{a_0}$ using (7.24), and goto Step 2.

3. Perform the mapping from $d_{\tilde{k}} = a_i|_{a_0}$ to a table address and read $\tilde{k} = \tilde{\rho} + \tilde{\kappa}2^c$ from the look-up table.

4. Let $\rho = \tilde{\rho}$ and compute $\kappa \equiv \tilde{\kappa} - i\hat{\delta} \pmod{2m}$.
   Then we have $\hat{\gamma} = \rho + \kappa 2^c$.

When using this algorithm to compute the discrete logarithm we need an addition by one (Step 1), at most $2m - 1$ binary shifts (Step 2), one table look-up (Step 3), and one multiplication by $\hat{\delta}$ and one addition modulo $2m$ (Step 4). Hence,the *computational* complexity of computing the discrete logarithm using the above algorithm is considerably reduced compared with the computational complexity of the corresponding algorithm described in Section 7.4.2. A similar reduction in complexity was obtained for exponentiation in Section 7.5.1, again to the cost of a look-up table of size at least[9] $2^c \times m$ bits.

In Step 2, we also need to check (at most $2m - 1$ times) whether $a_i|_{a_0}$ is an element of $\tilde{\Pi}$. The complexity of such a check and the complexity of obtaining the table address from $d_{\tilde{k}}$ depend strongly on the binary representation of the integers of $\tilde{\Pi}$. The ideal situation would of course be if the integers of $\tilde{\Pi}$ were consecutive numbers. The problem of finding a suitable set $\tilde{\Pi}$ is considered in the following sections.

7.5.3 The Mirror Properties of $\mathcal{M}_m$

The set $\tilde{\Pi}$ of integers $d_{\tilde{k}}$ from the matrix $\mathcal{D}_m$ was introduced in Notation 7.1 of the previous section. In the remainder of Section 7.5 we consider the problem of finding a suitable set $\tilde{\Pi}$ such that the elements of $\tilde{\Pi}$ can be analytically described in a simple way and such that we obtain a straightforward mapping from each $d_{\tilde{k}}$ to its associated table entry. The forming of the set $\tilde{\Pi}$ is based on the properties of an integer sequence $\mathcal{M}_m$:

[9] For exponentiation, the size of the table used is *exactly* $2^c \times m$ bits.

Definition 7.5 *Let $j \in \mathbb{Z}_{2^m}$ and let μ_j be equal to the number of the row of $\mathcal{D}_m$ which contains the integer j. We refer to the sequence*

$$\mathcal{M}_m = \mu_0, \mu_1, \ldots, \mu_j, \ldots, \mu_{2^m-1},$$

which has length 2^m, as the mirror sequence *associated with $\mathcal{D}_m$.*

From the definition follows that if $j = d_k$ for some $k = \rho + \kappa 2^c$ then $\mu_j = \rho$. Consequently, each row number $\rho \in \mathbb{Z}_{2^c}$ of $\mathcal{D}_m$ appears $2m$ times in $\mathcal{M}_m$. The following theorem describes the main connection between the subscripts of the $2m$ integers μ of $\mathcal{M}_m$ which are all located in the same row of $\mathcal{D}_m$.

Theorem 7.11 *Let, for some $j \in \mathbb{Z}_{2^m}$, the integer μ_j be an element of the sequence $\mathcal{M}_m$. Then, for any $i \in \mathbb{Z}$, we have*

$$\mu_{a_i|_j} = \mu_j.$$

Proof: For $a_0 = j \in \mathbb{Z}_{2^m}$ and $\rho \in \mathbb{Z}_{2^c}$, it follows by Theorem 7.5 that the $2m$ integers $j, a_1|_j, a_2|_j, \ldots a_{2m-1}|_j$ form the set of all elements in one of the rows $\mathcal{D}_{m,\rho}$ of $\mathcal{D}_m$. Consequently, from Definition 7.5 it follows that any two elements μ_g and μ_h with subscripts $g, h \in \mathcal{D}_{m,\rho}$ are equal. □

Corollary 7.2 *For $j \in \mathbb{Z}_{2^m}$, we have*

$$\mu_{2^m-1-j} = \mu_j \tag{7.39}$$

$$\mu_{2^{m-1}-1-j} = \mu_{2j} \tag{7.40}$$

Proof: The equalities follow by choosing some appropriate subscripts i in Theorem 7.11 and then using (7.24).

- For $i = m$ and $a_0 = j$ we get
  $a_m|_j \equiv 2^m(j+1) - 1 \equiv 2^m - 1 - j \pmod{2^m+1}$
  and hence we have $\mu_{2^m-1-j} = \mu_j$.
- For $i = m-1$ and $a_0 = 2j$ we get
  $a_{m-1}|_{2j} \equiv 2^{m-1}(2j+1) - 1 \equiv 2^{m-1} - 1 - j \pmod{2^m+1}$
  and hence we have $\mu_{2^{m-1}-1-j} = \mu_{2j}$.

□

By (7.39) of Corollary 7.2 it follows that the contents of the second half (the elements in positions 2^{m-1} to $2^m - 1$) of the sequence $\mathcal{M}_m$ is a mirror image of its first half (the elements in positions 0 to $2^{m-1} - 1$). Furthermore, (7.40) reveals another kind of distributed "mirror" property of $\mathcal{M}_m$: It follows from (7.40) that the sequence $\mu_0, \mu_2, \mu_4, \mu_6, \ldots, \mu_{2^m-2}$ (i.e. we take every *second* element of $\mathcal{M}_m$, starting from μ_0) equals the sequence $\mu_{2^{m-1}-1}, \mu_{2^{m-1}-2}, \mu_{2^{m-1}-3}, \mu_{2^{m-1}-4}, \ldots, \mu_0$ (i.e. we take every *consecutive* element of $\mathcal{M}_m$, starting from $\mu_{2^{m-1}-1}$ and going in the opposite direction). It is mainly because of these and other similar properties of $\mathcal{M}_m$ that we refer to $\mathcal{M}_m$ as a *mirror* sequence.

In Figure 7.5 we show the structure of the first half $\mu_0, \mu_1, \mu_2, \ldots, \mu_{2^7-1}$ of the sequence $\mathcal{M}_8$, in which each row number appears $m = 8$ times.[10] We have plotted the row numbers versus their respective positions in $\mathcal{M}_8$ in the form of checkerboard plots. The 128 cell columns of the two checkerboard plots are associated with the 128 first positions (0 to 127) of $\mathcal{M}_8$. In each column (position) there is only one cell that is black. The row number ρ associated with the black cell in column p equals the element μ_p of $\mathcal{M}_m$. For example, the black cells in columns 16 and 17 are located in the cell rows which are numbered 8 and 2, respectively. This implies $\mu_{16} = 8$ and $\mu_{17} = 2$.

Figures 7.5(a) and (b) differ only in the ordering of the cell rows. In Figure 7.5(a), where the rows appear in a natural increasing order, the checkerboard plot does not seem to reveal any particular structure. In Figure 7.5(b), however, it is quite easy to identify the mirror properties of $\mathcal{M}_m$. The ordering of the rows of the checkerboard plot in Figure 7.5(b) is based on the following rule:

1. Consider the positions $0, 1, 2, \ldots, 2^{m-1} - 1$ in $\mathcal{M}_m$.
2. Pick row 0 (zero) as the first row.
   The positions $0, 1, 3, 7, \ldots, 2^{m-1}-1$ in the first half of $\mathcal{M}_m$ which contain zero are now ruled out.
3. Select the row number which is contained in the foremost position of $\mathcal{M}_m$ that is *not* ruled out. This row is the next row of the checkerboard.
4. All positions in $\mathcal{M}_m$ which contain the last selected row number are now ruled out. Repeat from Step 3 until all rows have been selected.

By following this rule, the checkerboard plots of $\mathcal{M}_4$, $\mathcal{M}_8$, and $\mathcal{M}_{16}$ are all of the same type as the plot in Figure 7.5. In fact, it shows that the sequences $\mathcal{M}_4$, $\mathcal{M}_8$, and $\mathcal{M}_{16}$ are special cases of a more general class of mirror sequences. The

[10] This follows from the first "mirror" property (Equation (7.39)).

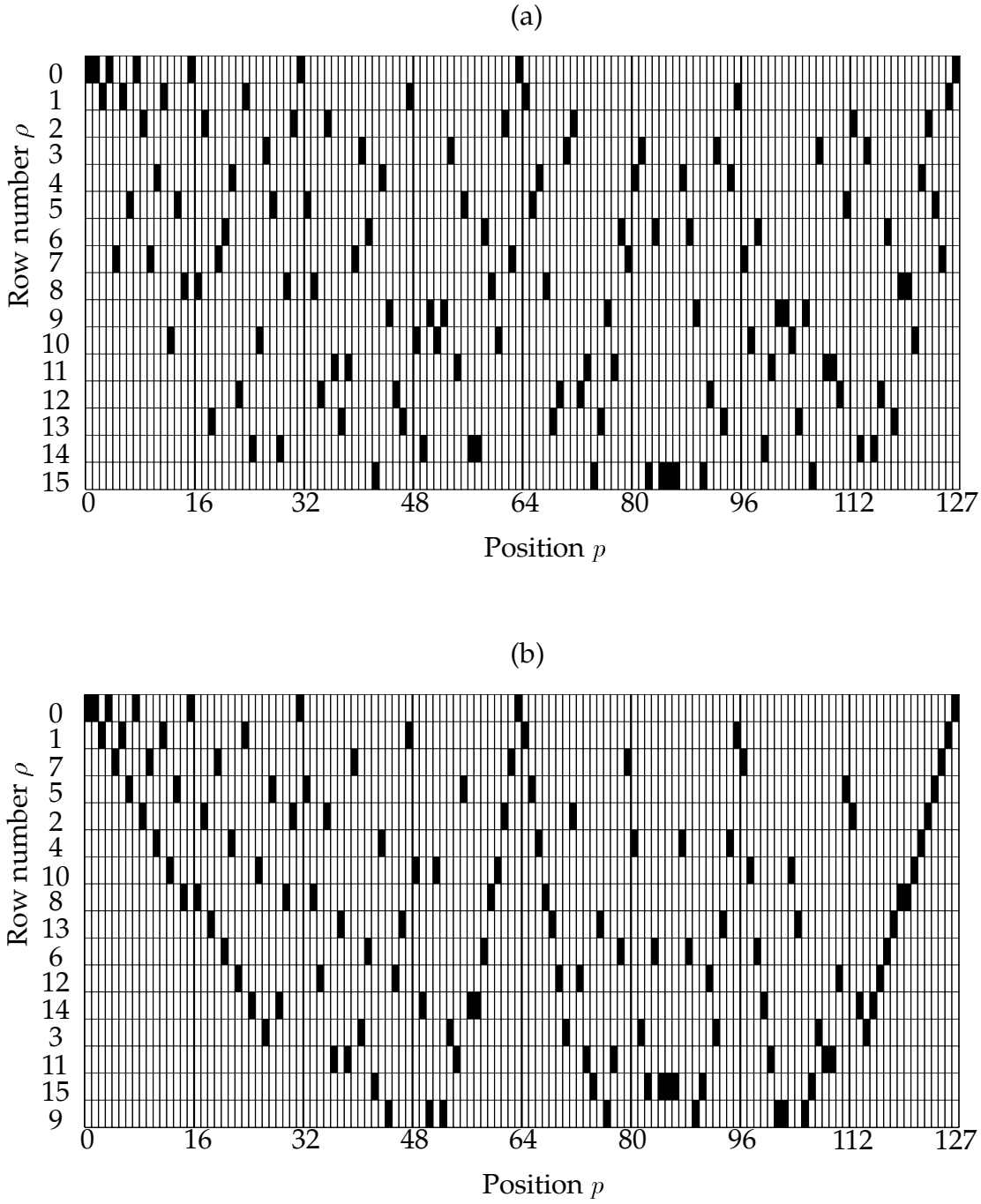


Figure 7.5: *Checkerboard plots of the first half of the sequence* $\mathcal{M}_8$.

sequences in this class all have checkerboard plots of the same type as the one in Figure 7.5(b). Also, the general class, which is not further considered here, contains one such sequence of length 2^m for *every* positive integer m.

7.5.4 Finding the Unique Distinct Positions in $\mathcal{M}_m$

From the definition of the mirror sequence $\mathcal{M}_m$ we have the following:

The problem of finding one unique matrix element in each row of $\mathcal{D}_m$, as described in Section 7.5.2, is equivalent to finding 2^c positions in $\mathcal{M}_m$ such that the contents in these positions are the 2^c distinct row numbers of $\mathcal{D}_m$ which form the set $\tilde{\Pi}$.

This set $\tilde{\Pi}$ of 2^c distinct row numbers of $\mathcal{D}_m$ was introduced in Notation 7.1 (page 184). Note that for $m = 1$ and $m = 2$, we have $2^c = 2^m/2m = 1$, which means that $\mathcal{D}_1$ and $\mathcal{D}_2$ are row vectors (i.e. $\mathcal{M}_1$ and $\mathcal{M}_2$ only contain the integer zero). Therefore, the results in the remainder of Section 7.5 is valid only for $m \geq 4$. When deriving the set $\tilde{\Pi}$ we need the following mapping:

Definition 7.6 *Let $P_0 = \{p_0, \ldots, p_{n-1}\}$ be a set of n integers $p_j \in \mathbb{Z}_{2^m}$ for $j = 0, 1, 2, \ldots, n-1$. The mapping $f^{(i)}(P_0) = P_i$ is defined as*

$$f^{(i)} : P_0 = \{p_0, \ldots, p_{n-1}\} \longrightarrow P_i = 2^i (P_0 + 1) - 1 = \left\{ a_i \,|_{p_0}, \ldots, a_i \,|_{p_{n-1}} \right\},$$

where $a_i \,|_{p_j} \equiv 2^i (p_j + 1) - 1 \pmod{2^m + 1}$.

Theorem 7.12 *Let $P_i = f^{(i)}(P_0)$ an let $\mathcal{M}_{P_0}$ denote the set $\left\{ \mu_{p_0}, \ldots, \mu_{p_{n-1}} \right\}$ of elements from the sequence $\mathcal{M}_m$. Then*

$$\mathcal{M}_{P_i} = \mathcal{M}_{P_0},$$

where $\mathcal{M}_{P_i} = \left\{ \mu_{a_i \,|_{p_0}}, \ldots, \mu_{a_i \,|_{p_{n-1}}} \right\}$.

Proof: By Theorem 7.5, the integers p_j and $a_i \,|_{p_j}$ are located in the same row of the matrix $\mathcal{D}_m$. Therefore, from Definition 7.5 it follows that μ_{p_j} equals $\mu_{a_i \,|_{p_j}}$. Hence, for $j = 0, 1, 2, \ldots, n-1$, we have $\mathcal{M}_{P_0} = \left\{ \mu_{p_0}, \ldots, \mu_{p_{n-1}} \right\} = \left\{ \mu_{a_i \,|_{p_0}}, \ldots, \mu_{a_i \,|_{p_{n-1}}} \right\} = \mathcal{M}_{P_i}$. □

Using the above notations, the goal is to find a set $\tilde{\Pi} = \{\tilde{\pi}_0, \tilde{\pi}_1, \ldots, \tilde{\pi}_{n-1}\}$ of positions such that $\mathcal{M}_{\tilde{\Pi}} = \{0, 1, \ldots, 2^c - 1\}$ and such that the size $n \geq 2^c$ of $\tilde{\Pi}$ is preferably equal to 2^c.

Definition 7.7 *The ordered set* $\{s_0,\, s_0+t,\, s_0+2t,\, s_0+3t, \ldots,\, s_{n-1}\}$ *of integers from* $\mathbb{Z}_{2^m}$ *is denoted by* $\{s_0, \ldots,\, s_{n-1}\}_t$.

Lemma 7.2 *Every integer of the set* $\{0,\, 1, \ldots,\, 2^{m-2}-1\} = \mathbb{Z}_{2^{m-2}}$ *maps, by* $f^{(i)}$, *into the set* $\Pi \triangleq \{2^{m-3}, \ldots,\, 2^{m-2}-1\}$ *of size* 2^{m-3}.

Proof: First, we have $f^{(m-2)}(0) = a_{m-2}\,|_0 \equiv 2^{m-2}(0+1)-1 = 2^{m-2}-1 \pmod{2^m+1}$, which is an element of Π. For $i = 0,\, 1,\, 2, \ldots,\, m-3$, we then apply the mapping $f^{(m-3-i)}$ on the set $P_0 = \{2^i, \ldots,\, 2^{i+1}-1\}$, which gives the set $P_{m-3-i} = 2^{m-3-i}\,(P_0+1)-1 \equiv \{2^{m-3}+2^{m-3-i}-1, \ldots,\, 2^{m-2}-1\}_{2^{m-3-i}} \pmod{2^m+1}$. It is obvious that $P_{m-3-i} \in \Pi$. □

Theorem 7.13 *Let* $\mathcal{M}_\Pi \triangleq \{\mu_{2^{m-3}},\, \mu_{2^{m-3}+1}, \ldots,\, \mu_{2^{m-2}-1}\}$ *be the set of row numbers which is associated with the set* $\Pi = \{2^{m-3},\, 2^{m-3}+1, \ldots,\, 2^{m-2}-1\}$ *of* 2^{m-3} *positions in* $\mathcal{M}_m$. *Then, we have*

$$\mathcal{M}_\Pi = \{0,\, 1, \ldots,\, 2^c-1\} = \mathbb{Z}_{2^c}.$$

Proof: We prove the theorem by showing that all positions $\lambda \in \mathbb{Z}_{2^m}$ map, via $f^{(i)}$, into the set Π. We know by (7.39) that the second half of $\mathcal{M}_m$ is a mirror image of its first half. This implies (see the proof of Corollary 7.2)

$$f^{(m)}\left(\{2^{m-1}, \ldots,\, 2^m-1\}\right) \equiv \{0, \ldots,\, 2^{m-1}-1\} \pmod{2^m+1}.$$

From Lemma 7.2 we get that all positions in $\{0, \ldots,\, 2^{m-2}-1\}$ map into Π. Hence, for every $\lambda \in \{0, \ldots,\, 2^{m-2}-1\} \bigcup \{2^{m-1}, \ldots,\, 2^m-1\}$, the row number μ_λ is contained in $\mathcal{M}_\Pi$.

Now, only the 2^{m-2} positions of $\Lambda \triangleq \{2^{m-2}, \ldots,\, 2^{m-1}-1\}$ remain. We partition this set into $m-1$ disjoint subsets Λ_i in the following way. Let $\Lambda = \bigcup_{i=0}^{m-2} \Lambda_i$, such that

$$\Lambda_i \triangleq \begin{cases} 2^{i+1}\,\{2^{m-3-i}, \ldots,\, 2^{m-2-i}-1\} + \Theta_i; & \text{for } 0 \le i \le m-3 \\ \Theta_{m-2}; & \text{for } i = m-2 \end{cases},$$

where

$$\Theta_i = 1 + \sum_{n=0}^{i-1}\left((-2)^{n+1}+2^n\right) = \frac{(3+2(-1)^i)2^i-2}{3} = \begin{cases} \dfrac{5\cdot 2^i-2}{3}; & \text{if } i \text{ is even} \\ \dfrac{2^i-2}{3}; & \text{if } i \text{ is odd} \end{cases}.$$

For $i = 0, 1, \ldots, m-3$, the set Λ_i contains 2^{m-3-i} elements. The set Λ_{m-2} only contains one element. For $0 \leq i \leq m-3$, we have

$$\begin{aligned} f^{(i(m-1)-1)}(\Lambda_i) &\equiv 2^{i(m-1)-1}\left(2^{i+1}\left\{2^{m-3-i}, \ldots, 2^{m-2-i}-1\right\} + \Theta_i + 1\right) - 1 \\ &\equiv (2^m)^i \left\{2^{m-3-i}, \ldots, 2^{m-2-i}-1\right\} \\ &\quad + (2^m)^{i+1} \cdot \frac{(3+2(-1)^i)2^{m-1} + 2^{m-1-i}}{3} - 1 \pmod{2^m+1}. \end{aligned}$$

If i is even, we get

$$\begin{aligned} f^{(i(m-1)-1)}(\Lambda_i) &\equiv \left\{2^{m-3-i}, \ldots, 2^{m-2-i}-1\right\} - \frac{5\cdot 2^{m-1} + 2^{m-1-i}}{3} + 2^m \\ &\equiv \left\{\frac{2^{m-1}-2^{m-3-i}}{3}, \ldots, \frac{2^{m-1}+2^{m-2-i}}{3} - 1\right\} \pmod{2^m+1}. \end{aligned}$$

It can easily be checked that this set of integers is a subset of Π.
If i is odd, we get

$$\begin{aligned} f^{(i(m-1)-1)}(\Lambda_i) &\equiv -\left\{2^{m-3-i}, \ldots, 2^{m-2-i}-1\right\} + \frac{2^{m-1}+2^{m-1-i}}{3} - 1 \\ &\equiv \left\{\frac{2^{m-1}-2^{m-2-i}}{3}, \ldots, \frac{2^{m-1}+2^{m-3-i}}{3} - 1\right\} \pmod{2^m+1}, \end{aligned}$$

which is also a subset of Π.
Furthermore, for $i = m-2$, the mapping $f^{(i(m-1)-1)}(\Lambda_i)$ equals

$$\begin{aligned} f^{((m-2)(m-1)-1)}(\Lambda_{m-2}) &\equiv 2^{(m-2)(m-1)-1}\left(\frac{5\cdot 2^{m-2}-2}{3} + 1\right) - 1 \\ &\equiv -2\cdot\frac{5\cdot 2^{m-2}+1}{3} + \frac{6\cdot 2^{m-1}}{3} \\ &\equiv \frac{2^{m-1}-2}{3} \pmod{2^m+1}, \end{aligned}$$

which is an integer of Π. Hence, we have $f^{(i(m-1)-1)}(\Lambda_i) \in \Pi$ for $0 \leq i \leq m-2$, which means that for every $\lambda \in \Lambda = \bigcup_{i=0}^{m-2} \Lambda_i$, the row number μ_λ is contained in $\mathcal{M}_\Pi$. □

The conception of the partitioning of Λ into the disjoint subsets Λ_i in the above proof may at first be difficult to grasp. We prefer not to go into detail here

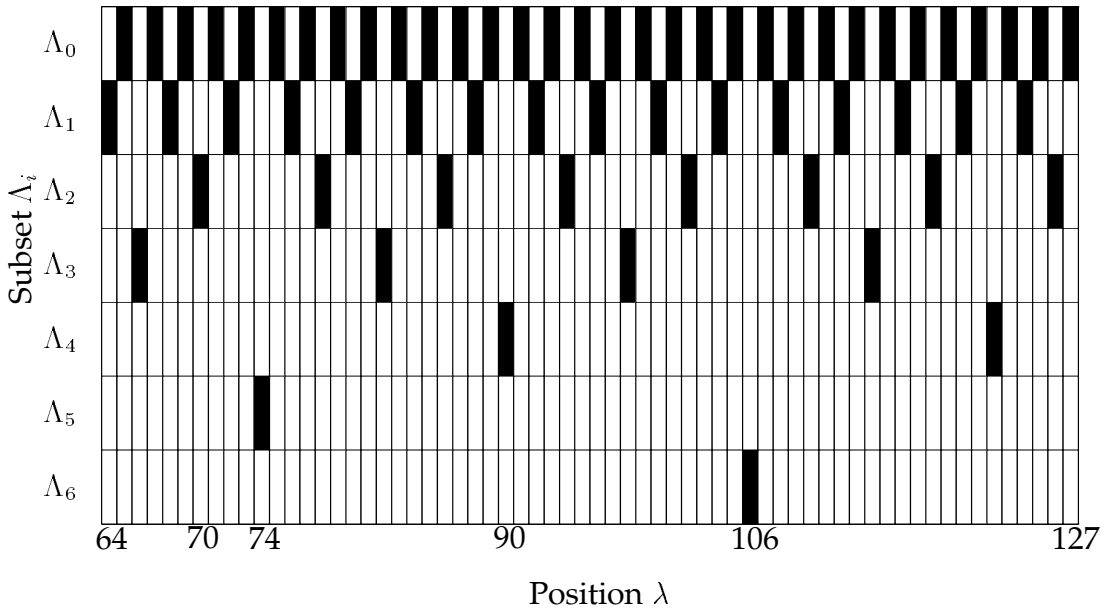


Figure 7.6: *Checkerboard plots of the contents* $\{64, 65, \ldots, 127\}$ *of* $\Lambda = \bigcup_{i=0}^{m-2} \Lambda_i$ *for* $m = 8$.

about the forming of these subsets. However, the checkerboard plot of the contents of Λ in Figure 7.6 may give some insight into the partitioning of Λ. The black cells in a cell row indicate which positions are contained in the subset Λ_i which is associated with that particular cell row. For example, we see that Λ_4 is formed by the positions 90 and 122.

Definition 7.8 *If an element appears several times in a set, we say that the extra elements are* redundant. *By the* relative redundancy *of a set we mean the ratio of the redundant elements to the total number of elements in the set.*

For $m \geq 4$, the relative redundancy of the set $\mathcal{M}_\Pi$ equals $(2^{m-3} - 2^c)/2^{m-3} = (m-4)/m$. Then, for $m = 4$ the relative redundancy equals zero, which means that $\Pi = \{2^{4-3}, \ldots, 2^{4-2} - 1\} = \{2, 3\}$ is such a set $\tilde{\Pi}$ of unique positions that we are looking for. The integers $2 = (0010)_2$ and $3 = (0011)_2$ of $\Pi = \tilde{\Pi}$ are the only 4-bit NBC integers whose three most significant bits equal $(001)_2$. Therefore, in Step 2 on page 185, the checking whether $a_i|_{a_0}$ is an integer of $\tilde{\Pi}$ is performed by checking whether the three most significant bits of $a_i|_{a_0}$ equals $(001)_2$. In Step 3 on the same page, the least significant bit of $a_i|_{a_0}$ can be used to address the look-up table. If $a_i|_{a_0} = d_{\tilde{k}_0} \triangleq 2$, the subscript $\tilde{k}_0$ is read from table location 0 and if $a_i|_{a_0} = d_{\tilde{k}_1} \triangleq 3$, the subscript $\tilde{k}_1$ is read from table location 1.

For $m = 8$ and $m = 16$ the relative redundancy is 50% and 75% respectively. It is desirable to reduce these redundancies even more. We therefore further reduce the set $\mathcal{M}_\Pi$.

Theorem 7.14 *Let* $\Pi_{24} = \Pi_2 \cup \Pi_4$, *where*

$$\begin{aligned}\Pi_2 &\triangleq \left\{\frac{2(2^{m-2}-1)}{3}, \ldots, 3\cdot 2^{m-4}-1\right\}\\ \Pi_4 &\triangleq \left\{\frac{5\cdot 2^{m-3}-1}{3}, \ldots, 2^{m-2}-1\right\}\end{aligned}$$

are disjoint subsets of Π. *Also, let* $\mathcal{M}_{\Pi_{24}}$ *be the set of* $2^{m-4}+1$ *integers of* $\mathcal{M}_m$ *in the positions given by the elements of* Π_{24}. *Then, for* $m = 8$ *and* $m = 16$, *we have*

$$\mathcal{M}_{\Pi_{24}} = \{0, 1, \ldots, 2^c - 1\} = \mathbb{Z}_{2^c}.$$

Proof: Let $\Pi = \bigcup_{i=1}^{4} \Pi_i$, where

$$\begin{aligned}\Pi_1 &\triangleq \left\{2^{m-3}, \ldots, \frac{2(2^{m-2}-1)}{3}-1\right\}\\ \Pi_3 &\triangleq \left\{3\cdot 2^{m-4}, \ldots, \frac{5\cdot 2^{m-3}-1)}{3}-1\right\}\end{aligned}$$

and where Π_2 and Π_4 are defined in the theorem. The equality $\mathcal{M}_{\Pi_{24}} = \mathbb{Z}_{2^c}$ in Theorem 7.14 holds if (and only if) the integers in the sets Π_1 and Π_3 map, by $f^{(i)}$, into Π_2 and/or Π_4 for some i. Let $\Pi_3 = \bigcup_{j=1}^{3} \Pi_{3,j}$, where

$$\begin{aligned}\Pi_{3,1} &\triangleq 3\cdot 2^{m-4} + \left\{0, \ldots, \frac{2(2^{m-6}-1)}{3}-1\right\}\\ \Pi_{3,2} &\triangleq 3\cdot 2^{m-4} + \left\{\frac{2(2^{m-6}-1)}{3}, \ldots, 2^{m-6}-1\right\}\\ \Pi_{3,3} &\triangleq 3\cdot 2^{m-4} + \left\{2^{m-6}, \ldots, \frac{2^{m-4}-1}{3}-1\right\}\end{aligned}$$

are disjoint subsets of Π_3. Then, using the function $f^{(m+2)}$, we get

$$\begin{aligned}f^{(m+2)}(\Pi_{3,1}) &\equiv \left\{\frac{5\cdot 2^{m-3}-1}{3}+3, \ldots, 2^{m-2}-4\right\}_4 \pmod{2^m+1}\\ f^{(m+2)}(\Pi_{3,2}) &\equiv \left\{3\cdot 2^{m-4}, \ldots, \frac{5\cdot 2^{m-3}-1}{3}-1\right\}_4 \pmod{2^m+1}\\ f^{(m+2)}(\Pi_{3,3}) &\equiv \left\{\frac{2(2^{m-2}-1)}{3}+2, \ldots, 3\cdot 2^{m-4}-4\right\}_4 \pmod{2^m+1},\end{aligned}$$

where clearly $f^{(m+2)}(\Pi_{3,1})$ is a subset of Π_4, $f^{(m+2)}(\Pi_{3,2})$ is a subset of Π_3, and $f^{(m+2)}(\Pi_{3,3})$ is a subset of Π_2. We again partition $f^{(m+2)}(\Pi_{3,2})$ into three disjoint subsets, say $\Pi_{3,2,1}$, $\Pi_{3,2,2}$, and $\Pi_{3,2,3}$, in the same way as we did with Π_3. Then, we get $f^{(m+2)}(\Pi_{3,2,1}) \subset \Pi_4$, $f^{(m+2)}(\Pi_{3,2,2}) \subset \Pi_3$, and $f^{(m+2)}(\Pi_{3,2,3}) \subset \Pi_2$. This process of partitioning $\Pi_{3,2,2,\ldots,2} \subset \Pi_3$ into three disjoint subsets which map (by $f^{(m+2)}$) into Π_4, Π_3, and Π_2, respectively, is repeated until only two integers remain. One of these integers map into Π_2 and the other integer maps into Π_4. *Hence, for every integer $\lambda \in \Pi_3$ there is a positive integer i such that $f^{(i(m+2))}(\lambda) \in \Pi_{24}$.*

In order to show that the set Π_1 can be mapped into $\Pi_{24} = \Pi \cup \Pi_4$ we partition it into a number of disjoint subsets which map into Π_{24} in different ways. This approach is used both in the proof of Theorem 7.13 and in the above proof that Π_3 maps into Π_{24}. It shows that Π_1 can be partitioned into $m-4$ disjoint subsets, say $\Pi_{1,1}$, $\Pi_{1,2}$, ..., $\Pi_{1,m-4}$, such that

$$f^{(j+2)}(\Pi_{1,j}) \in \Pi_{24} \cup \Pi_3,$$

where $1 \leq j \leq m-4$. We have not yet been able to express the subsets $\Pi_{1,j}$ analytically, but they can easily be obtained as follows.[11]

1. Let $j = 0$ and $P_0 = f^{(2)}(\Pi_1)$.
2. Let $j = j+1$ and $P_j \equiv 2P_{j-1} + 1 \pmod{2^m+1}$.
3. Let $\Pi_{1,j}$ be equal to the set of integers in P_j which are also contained in $\Pi_{24} \cup \Pi_3$.
   Let $P_j = P_j \setminus \Pi_{1,j}$.
4. If $P_j \neq \emptyset$ (i.e. if $j < m-4$), goto Step 2.
   Otherwise, stop.

Hence, for every integer $\lambda \in \Pi_1$ there is an integer $i; 3 \leq i \leq m-2$ such that $f^{(i)}(\lambda) \in \Pi_{24} \cup \Pi_3$. As shown above, the integers of Π_3 in turn map into Π_{24}. Consequently, *every* integer of $\Pi_1 \cup \Pi_3$ maps into Π_{24}, which means that $\mathcal{M}_{\Pi_{24}} = \mathcal{M}_{\Pi} = \mathbb{Z}_{2^m}$. □

Because Π_2 contains $(2^{m-4}+2)/3$ elements and Π_4 contains $(2^{m-3}+1)/3$ elements, their union $\Pi_{24} = \Pi_2 \cup \Pi_4$ contains $(2^{m-4}+2)/3 + (2^{m-3}+1)/3 = 2^{m-4}+1$ elements. Therefore, the relative redundancy of the set $\mathcal{M}_{\Pi_{24}}$ equals

$$\frac{2^{m-4}+1-2^m/2m}{2^{m-4}+1} = \frac{m-8}{m} + \frac{16}{m(2^{m-3}+2)} = \begin{cases} 1/17; & \text{for } m=8 \\ 0.5+1/8194; & \text{for } m=16 \end{cases}.$$

[11] For example, for $m = 8$ we have $\Pi_{1,1} = \{37, 38, 39\}$, $\Pi_{1,2} = \{34, 35\}$, $\Pi_{1,3} = \{33, 41\}$, and $\Pi_{1,4} = \{32, 36, 40\}$.

In order to store $2^{m-4}+1$ integers of $\mathbb{Z}_{2^m}$, we need a table of size $2^{m-3}\times m$ bits. Unfortunately, almost half of the locations in such a table would not be used. In the next section we show that one of the elements in the set $\mathcal{M}_{\Pi_{24}}$ is dispensable. This property makes it possible to reduce the size of the table needed to $2^{m-4}\times m$ bits.

7.5.5 Addressing the Look-Up Table for Discrete Logarithm

The sets $\Pi_2=\left\{\frac{2(2^{m-2}-1)}{3},\ldots,\,3\cdot 2^{m-4}-1\right\}$ and $\Pi_4=\left\{\frac{5\cdot 2^{m-3}-1}{3},\ldots,\,2^{m-2}-1\right\}$ can be viewed as row vectors of m-bit NBC integers:

$$\Pi_2=\left\{\begin{array}{c}0010\,1010\cdots 010\\0010\,1010\cdots 011\\\vdots\\0010\,1111\cdots 110\\0010\,1111\cdots 111\end{array}\right\}\qquad \Pi_4=\left\{\begin{array}{c}0011\,0101\cdots 101\\0011\,0101\cdots 110\\\vdots\\0011\,1111\cdots 110\\0011\,1111\cdots 111\end{array}\right\}$$

We see that the four most significant bits in every NBC integer of Π_2 and Π_4 are equal to $(0010)_2$ and $(0011)_2$, respectively. Hence, arbitrary integers $p_2\in\Pi_2$ and $p_4\in\Pi_4$ can be written on the forms $p_2=2^{m-3}+p_2^{(m-5)}$ and $p_4=2^{m-3}+2^{m-4}+p_4^{(m-5)}$, where

$$\begin{aligned}p_2^{(m-5)}\in\tilde{\Pi}_2 &\triangleq \Pi_2-2^{m-3}=\left\{\frac{2(2^{m-4}-1)}{3},\ldots,\,2^{m-4}-1\right\}\\ p_4^{(m-5)}\in\tilde{\Pi}_4 &\triangleq \Pi_4-2^{m-3}-2^{m-4}=\left\{\frac{2^{m-4}-1}{3},\ldots,\,2^{m-4}-1\right\}\end{aligned}$$

are $(m-4)$-bit NBC integers. Let $\overline{\tilde{\Pi}}_2$ denote the set formed by the one's complements of the $(m-4)$-bit NBC integers of $\tilde{\Pi}_2+1 \mod 2^{m-4}$, i.e. we have

$$\begin{aligned}\overline{\tilde{\Pi}}_2 &\triangleq 2^{m-4}-1-(\tilde{\Pi}_2+1)=\left\{\frac{2^{m-4}-1}{3}-1,\ldots,\,0,\,-1\right\}_{-1}\\ &\equiv \left\{0,\ldots,\,\frac{2^{m-4}-1}{3}-1\right\}\cup\{2^{m-4}-1\}\pmod{2^{m-4}}.\end{aligned}$$

Hence, we have $\overline{\tilde{\Pi}}_2\cup\tilde{\Pi}_4=\{0,\ldots,\,2^{m-4}-1\}$ with only one redundant element – the integer $2^{m-4}-1$ appears twice in the union set $\overline{\tilde{\Pi}}_2\cup\tilde{\Pi}_4$. Fortunately, we can allow this overlap. For any $p_2\in\Pi_2$, let $q_2=p_2^{(m-5)}+1$. Then, we have $\overline{q_2^{(m-5)}}=2^{m-4}-1-q_2^{(m-5)}\in\overline{\tilde{\Pi}}_2$. Using (7.24) we can write $p_4=2^{m-2}-1$ on

the form $p_4 = 2^{m-2}(0+1) - 1 \equiv a_{m-2}|_{a_0} \pmod{2^m+1}$, where $a_0 = d_0 = 0$. Hence, we have $p_4 = d_{\tilde{k}_4}$ for some $\tilde{k}_4 = \tilde{\rho}_4 + \tilde{\kappa}_4 2^c$, where $\tilde{\rho}_4 = 0$ (i.e. p_4 is in row zero (the top row) of the matrix $\mathcal{D}_m$). By (7.31) in Lemma 7.1 we also have

$$p_4 = d_{\tilde{k}_4} \equiv a_{\tilde{\kappa}_4\hat{\sigma}}|_0 \pmod{2^m+1},$$

which consequently implies $\tilde{\kappa}_4\hat{\sigma} \equiv m-2 \pmod{2m}$. From this congruence we get $\tilde{\kappa}_4 \equiv (m-2)\hat{\delta} \equiv m-22 \pmod{2m}$. Hence, for $p_4 = d_{\tilde{k}_4} = 2^{m-2}-1$, the subscript $\tilde{k}_4 \equiv (m+22)2^c \pmod{2^m}$ is stored in location $p_4^{(m-5)} = 2^{m-4}-1$ of the look-up table.

Now, consider the integer $p_2 = 3 \cdot 2^{m-4} - 1 = 2^{m-4}(2+1) - 1 \equiv a_{m-4}|_{a_0} \pmod{2^m+1}$, where $a_0 = d_1 = 2$, and which maps to the same table entry as p_4 (note that here we generally use the primitive element $\alpha = 3$ when computing d_k in (7.25)). Thus, p_2 is in row $\tilde{\rho}_2 = 1$ of $\mathcal{D}_m$. Again, for $\tilde{k}_2 = \tilde{\rho}_2 + \tilde{\kappa}_2 2^c$, we have

$$p_2 = d_{\tilde{k}_2} \equiv a_{\tilde{\kappa}_2\hat{\sigma}}|_2 \pmod{2^m+1}$$

by (7.31). This gives $\tilde{\kappa}_2\hat{\sigma} \equiv m-4 \pmod{2m}$, from which we get $\tilde{\kappa}_2 \equiv (m-4)\hat{\delta} \equiv m-44 \pmod{2m}$. For $m = 8$ and $m = 16$, which are the cases considered here, we can write $\tilde{\kappa}_2 \equiv m-12 \equiv m+20 \pmod{2m}$ and $\tilde{\kappa}_4 \equiv m+10 \pmod{2m}$. Hence, with $\tilde{\kappa}_4$ being the least nonnegative residue of $m+10$ modulo $2m$, we get $\tilde{\kappa}_2 = 2m-2-\tilde{\kappa}_4$, which means that $\tilde{\kappa}_2$ *can be obtained from* $\tilde{\kappa}_4$ *by inverting its* $\log_2 m$ *most significant bits*. Also, $\tilde{\rho}_2 = 1$ can be obtained from $\tilde{\rho}_4 = 0$ by inverting its least significant bit.

Consequently, by storing the subscript $\tilde{k}_4$ of $d_{\tilde{k}_4} = p_4 = 2^{m-2} - 1$ in the table, the subscript $\tilde{k}_2$ of $d_{\tilde{k}_2} = p_2 = 3 \cdot 2^{m-4} - 1$ (which maps to the same table entry as p_4) can be obtained simply by inverting $\log_2 m + 1$ of the table output bits. Each inversion may be implemented as a 2-input XOR gate, with one of its inputs coming from the table output and the other input coming from the last carry of the $(m-4)$-bit addition $q_2 = \tilde{p}_2 + 1$. This carry is high only when $\tilde{p}_2 = 2^{m-4}-1$, i.e when $p_2 = 3 \cdot 2^{m-4} - 1$, which therefore is the only case when the table output is changed (from $d_{\tilde{k}_4}$ to $d_{\tilde{k}_2}$) by the XOR gates. This is further discussed in Section 7.6.2.

From the above reasoning we conclude that the $2^{m-4}+1$ elements of $\Pi_{24} = \Pi_2 \cup \Pi_4$ can be mapped onto the entries of a look-up table (memory) of size $2^{m-4} \times m$ bits as follows:

- Each position $p_4 \in \Pi_4$ maps to table entry $p_4^{(m-5)}$. The table output is the m-bit NBC subscript $\tilde{k}_4$ of $d_{\tilde{k}_4} = p_4$.

- Each position $p_2 \in \Pi_2$ maps to table entry $\overline{q_2^{(m-5)}}$, i.e. the one's complement of $q_2^{(m-5)}$, where $q_2 = p_2^{(m-5)} + 1$. The table output is the m-bit NBC subscript $\tilde{k}_2$ of $d_{\tilde{k}_2} = p_2$. However, if $p_2 = 3 \cdot 2^{m-4}$, the table output is the subscript $\tilde{k}_4$ of $d_{\tilde{k}_4} = 2^{m-2} - 1$. This table output is modified to the desired value by a simple circuit.

Because we can handle the problem when $3 \cdot 2^{m-4} - 1$ and $2^{m-2} - 1$ both map to the same table entry, the actual relative redundancy of the set $\mathcal{M}_{\Pi_{24}}$ is exactly 0% for $m = 8$ and 50% for $m = 16$. In the latter case, for $m = 16$, it is possible to reduce the set Π_{24} even further. However, we have not yet succeeded in reducing it by half to 2^{m-5} elements. Such a set would have 0% relative redundancy. If the number of elements in the reduced set obtained from Π_{24} is greater than 2^{m-5}, we still need a look-up table (memory) of size $2^{m-4} \times m$ bits. Therefore, for $m = 8,\ 16$ we let $\tilde{\Pi} = \Pi_{24}$, where $\tilde{\Pi}$ is the set introduced in Notation 7.1 in Section 7.5.2. Note that for $m = 4$, we let $\tilde{\Pi} = \Pi$ (see the paragraph subsequent to Definition 7.8 in Section 7.5.4).

7.6 Architectures for Arithmetic Operations

In this section we propose VLSI architectures for most of the arithmetic operations considered in Sections 7.2 – 7.5. The sizes, fan-ins, internal CP delays, and output normalised resistances of the basic building blocks in the architectures are given in Chapter 4. Note that these complexity parameters are summarised in Table 4.2.

7.6.1 Discrete Exponentiation

An Architecture for Computing $a_i|_{a_0}$

The respective algorithms in Sections 7.4.1 and 7.5.1 for performing discrete exponentiation both involve the computation of $a_i|_{a_0}$ from some a_0 using a feedback shift register of length m (see Step 1(b) on page 174 and Step 2 on page 183). Such a shift register is shown in Figure 7.7. Generally, the circuit can be used to compute $a_i|_{a_0}$, which is defined in (7.24), from an arbitrary $a_0 \in \mathbb{Z}_{2^m}$ by loading it with a_0 and shifting (rotating) the register contents i steps to the left. The size of the circuit equals

$$\mathcal{C}_{\text{ai}} = m\mathcal{C}_{\text{reg}} + \mathcal{C}_{\text{inv}} = 16m + 2$$

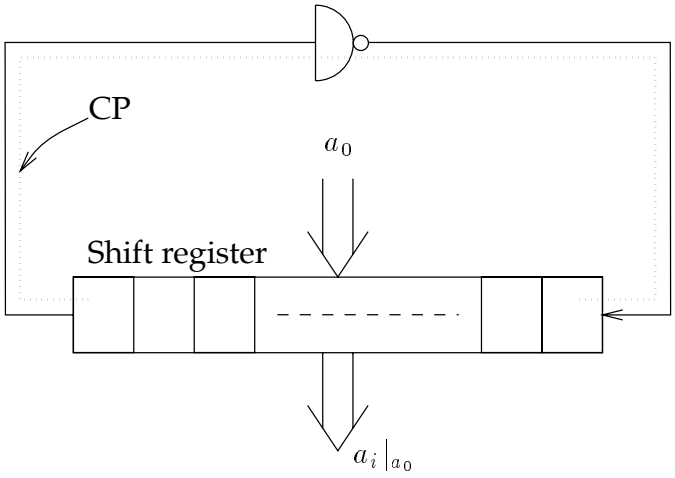


Figure 7.7: *Recursive computation of $a_i|_{a_0}$ from a_0 using a feedback shift register of length m.*

and the CP, which runs from the output of the register element in the most significant bit position to the input of the element in its least significant bit position, equals

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,ai}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}}(f_{\mathrm{inv}} + f_{\mathrm{next}}) + r_{\mathrm{inv}} f_{\mathrm{reg}} \\ &= 28 + 2f_{\mathrm{next}},\end{aligned}$$

where f_{next} is the fan-in (seen from the most significant bit position of the register) of the circuit subsequent to the register. For example, if the subsequent circuit is another register, we get $f_{\mathrm{next}} = f_{\mathrm{reg}} = 2$ and thus $\mathcal{L}_{\mathrm{CP,ai}} = 32$. Assuming that an initial clock cycle is required to load the shift register with a_0, the register contains the desired result $a_i|_{a_0}$ after i additional clock cycles. Thus, the total computation time T is proportional to

$$\mathcal{L}_{\mathrm{ai}} = (i+1)\mathcal{L}_{\mathrm{CP,ai}}, \tag{7.41}$$

where $i \leq 2m - 1$.

An Architecture for Computing d_k

When performing discrete exponentiation using the algorithm in Section 7.4.1, we need to recursively compute d_k from d_{k-1} (see Step 2 on page 174). The architecture in Figure 7.8 for computing d_k is based on (7.36), i.e.

$$d_k \equiv (2d_{k-1} + 1) + d_{k-1} + 1 \pmod{2^m + 1},$$

which was obtained from (7.25) by letting $\alpha = 3$. The addend $2d_{k-1} + 1 \bmod 2^m + 1$, which equals $a_1|_{d_{k-1}}$, is obtained simply by inverting the most significant bit (the wire labelled "msb" in the figure) of d_{k-1} and modifying the feedback wirings.[12] Let $d_k \equiv \sigma + 1 \pmod{2^m + 1}$, where $\sigma \triangleq (2d_{k-1} + 1) + d_{k-1} = 3d_{k-1} + 1$ is the sum of the addend $2d_{k-1} + 1$ and the augend d_{k-1} in Figure 7.8. If $\sigma \geq 2^m$, i.e. if $\sigma_m = 1$, we have $d_k \equiv 2^m + \sigma^{(m-1)} + 1 \equiv \sigma^{(m-1)} \pmod{2^m + 1}$. In this case we set the first carry signal, which in the figure is denoted by c_0, of the adder equal to zero. If $\sigma < 2^m$, we have $d_k \equiv \sigma^{(m-1)} + 1 \pmod{2^m + 1}$. Here, we set c_0 equal to one.

The carry signal c_0 can be generated using for example a comparator. From the inequality $\sigma = 3d_{k-1}+1 \geq 2^m$ it follows that $d_{k-1} \geq (2^m-1)/3 = (01010\cdots01)_2$. A comparator can for example be implemented using a chain of full adder elements, where the carry out of the most significant bit position indicates whether one of the addends is greater than the other (see Weste and Eshraghian [113, Fig. 8.26]). In our case, where one of the comparator addends is always $(2^m - 1)/3$, the comparator simplifies to a chain of $m - 1$ alternating OR and AND gates.[13] With respect to the comparator propagation delay, this carry ripple type of comparator is preferably used together with a parallel carry ripple adder: By inserting the comparator *prior* to the register in Figure 7.8 and by modifying its output circuitry, *the resulting comparator do not have any effect on the CP length of the total circuit*.

If $d_{k-1} \geq (2^m - 1)/3$, the comparator output equals 1, otherwise it equals 0. Note that we do *not* refer to the output of the modified comparator (see the figure). The first carry c_0 of the adder equals the inverse of the comparator output. In the architecture in Figure 7.8, this inversion is realised by exchanging the output OR gate of the comparator for a NOR gate. In the figure, the resulting NOR gate is moved outside the comparator.

With the m-bit parallel adder in Figure 7.8 being a standard carry ripple adder, the size of the complete circuit equals

$$\begin{aligned} \mathcal{C}_{\text{dk}} &= m\mathcal{C}_{\text{FA}} + (m+1)\mathcal{C}_{\text{reg}} + \mathcal{C}_{\text{comp}} + \mathcal{C}_{\text{NOR}} + \mathcal{C}_{\text{inv}} \\ &= 50m + 10, \end{aligned}$$

where $\mathcal{C}_{\text{comp}} = (m-2)\mathcal{C}_{\text{AND/OR}} = 6(m-2)$ is the size of the modified comparator. The CP through the circuit is the path from the output of the register element in the least significant bit position along the carry chain of the parallel adder to the input of the register element in the most significant bit position. This path, which is marked by the dotted line in Figure 7.8, has length

[12] The procedure is based on the architecture in Figure 7.7.

[13] The resulting chain of gates both starts and ends with an OR gate.

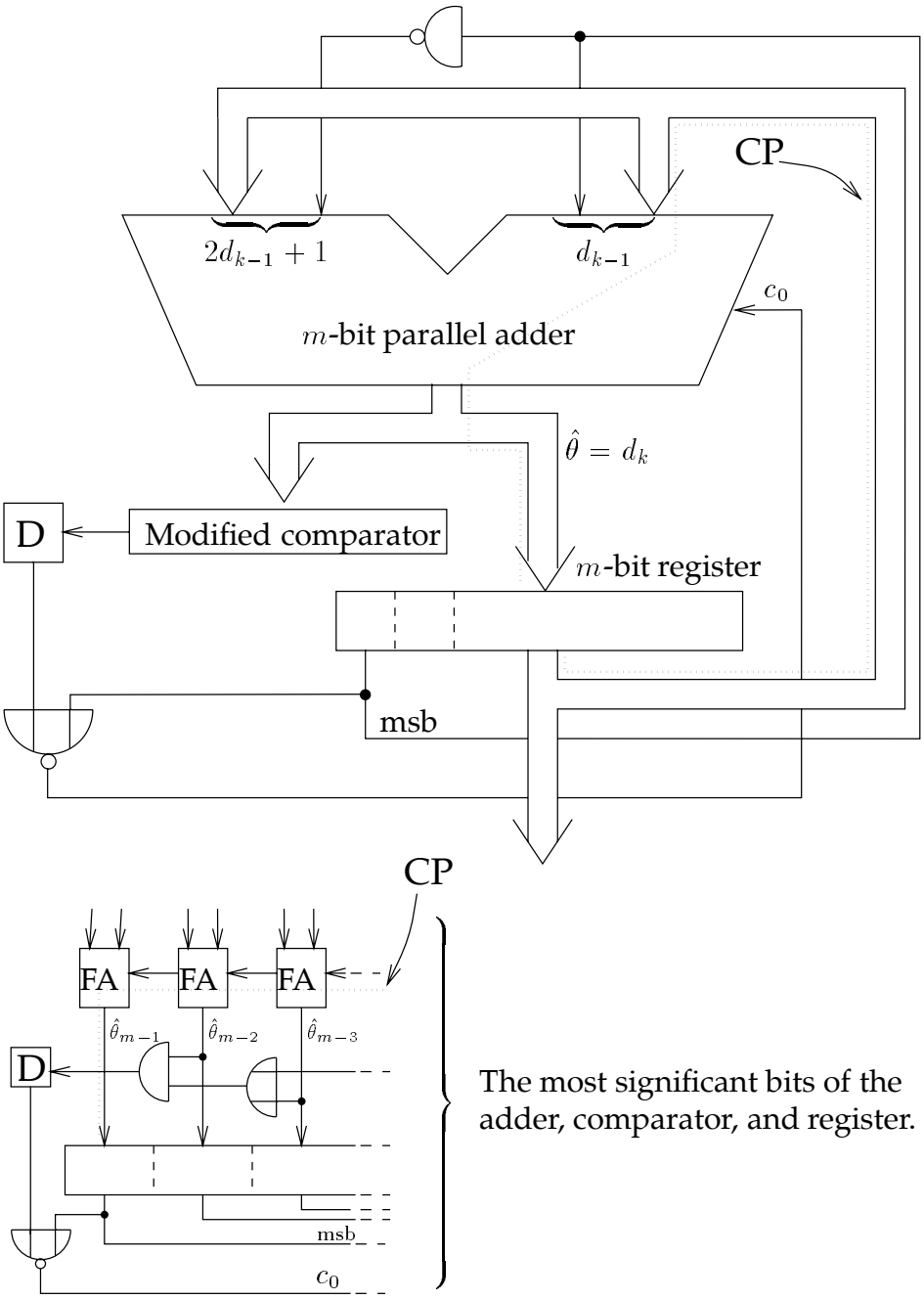


Figure 7.8: *Recursive computation of d_k from d_{k-1} when $\alpha = 3$. In order to conveniently label the sum output bits, say as $\hat{\theta}_{m-1}$, $\hat{\theta}_{m-2}$, $\hat{\theta}_{m-3}, \ldots, \hat{\theta}_0$, we only temporarily define $\hat{\theta} \triangleq d_k$ in this figure.*

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,dk}} &= \mathcal{L}_{\mathrm{reg}} + r_{\mathrm{reg}} \cdot 2f_{\mathrm{FA,signal}} + (m-1)(\mathcal{L}_{\mathrm{FA,carry}} + r_{\mathrm{FA}} f_{\mathrm{FA,carry}}) \\ &\quad + \mathcal{L}_{\mathrm{FA,sum}} + r_{\mathrm{FA}} f_{\mathrm{reg}} \\ &= 14m + 54.\end{aligned}$$

Hence, after the register has been loaded with some integer $d_k \in \mathbb{Z}_{2^m}$, the time T needed for each recursive computation of d_{k+1}, d_{k+2}, d_{k+3}, etc. is proportional to $\mathcal{L}_{\mathrm{CP,dk}}$. Note that in order to obtain the correct carry signal c_0 for these computations, *the initial integer d_k must also be fed to the modified comparator*. If not, the D flip-flop in Figure 7.8 may not be properly initiated.

The Look-Up Table

The algorithm in Section 7.5.1 for performing discrete exponentiation involves the process of reading integers from a look-up table. Such a look-up table is suitably implemented as a semiconductor memory. A look-up table of size $2^u \times 2^w$ bits is usually implemented as a memory of size $2^a \times 2^b$ bits (2^a rows and 2^b columns), where $a + b = u + w$ and $b \geq w$. Figure 7.9 shows the block diagram of a typical random-access memory (RAM) of size $2^a \times 2^b$ bits.

When reading from the memory, a of the u address lines select one of the 2^a rows of the memory array. The remaining $u - a$ address lines select 2^w of the 2^b columns. The contents of the 2^w corresponding memory cells in the accessed row are detected and multiplexed to the data output. A memory cell and the sense amplifier connected to the bit and $\overline{bit}$ lines of that cell is shown in Figure 7.10. The memory cell considered is a standard six-transistor static RAM cell. The sense amplifier is used to sense the state of the memory cell.

The size of the $(2^a \times 2^b)$-bit memory cell array equals $6 \cdot 2^{a+b}$. Using a NOR-type row decoder [44, Fig. 9.10-2] and a standard column tree decoder [44, Fig. 9.10-3], the size of these (line) decoders together with the sense amplifiers is roughly in the order of $a2^a + b2^b$. Hence, using six-transistor memory cells, the total chip area occupied by the $(2^a \times 2^b)$-bit RAM in Figure 7.9 is proportional to the size

$$\mathcal{C}_{\mathrm{RAM}} = 6 \cdot 2^{a+b} + a2^a + b2^b. \tag{7.42}$$

In order to minimise the area complexity of the address decoding, the memory array is usually organised as a square array, i.e. we have $a = b$ (if $a + b$ is even). Note that we do not consider the chip area occupied by the address bus or the word lines and data lines of the memory.

The critical path associated with the process of reading from the memory can be separated into two main paths. The first path runs from the address input

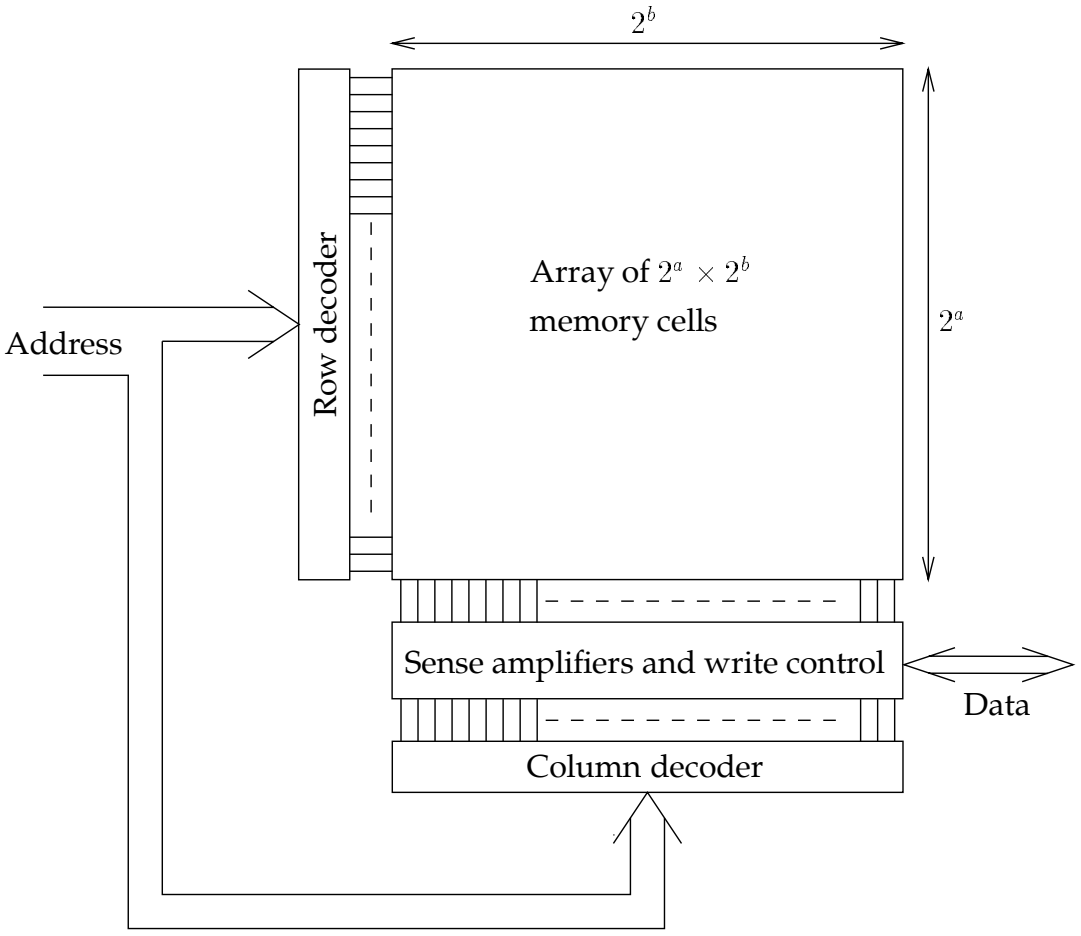


Figure 7.9: *A block diagram of a typical random-access memory.*

through the row decoder and along one of the word lines in the memory array. The second path runs from inside an accessed memory cell along a data bit line, through a sense amplifier, to the data output. The memory access time is dominated by the time required to fully charge the word line plus the time required to sense the state of an accessed memory cell. In order to minimise the length of the first path, i.e. to speed up the charging of the word line, a column of drivers is usually inserted between the row decoder and the memory array. The chip area occupied by these drivers is neglected here.[14]

The delay of a stage with capacitive load C_{L}, which is driven by an optimised driver, is proportional to $\log_2 (C_{\text{L}}/C_g)$, where C_g is the (minimum size) transis-

[14] An optimised driver on a word line is formed by a number of cascaded inverters of *increasing size*. The total area occupied by the column of such drivers is actually greater than the row decoder area, but it is less than the area occupied by the memory cell array.

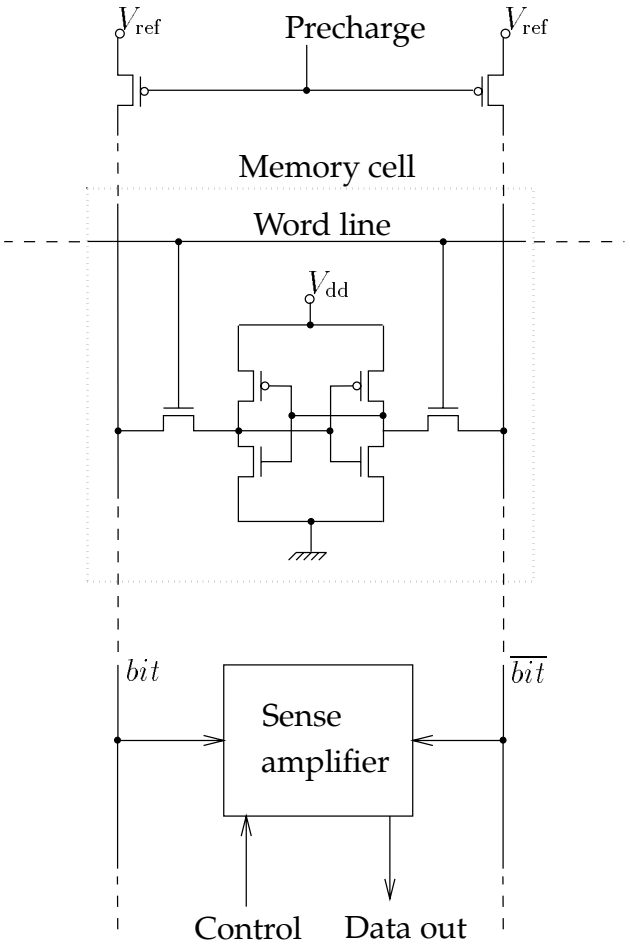


Figure 7.10: *A memory cell and the sense amplifier for detecting the memory cell contents.*

tor gate capacitance. We refer to Mead an Conway [66, Sec. 1.5]. When using the six-transistor static memory cell in Figure 7.10, the total capacitance C_{L} at each word line (not counting the wire capacitance) equals $2^b \cdot 2C_g$, where 2^b is the number of memory cells in one row of the memory array. Hence, the word line delay is proportional to $\log_2 2^{b+1} = b + 1$, which implies that the length of the first part of the critical path is about $\mathcal{L}_1 = (b + 1)L_1$, where L_1 is some constant.

Prior to the driving of the word line, all bit and $\overline{bit}$ lines of the memory array are precharged to some suitable potential. When a word line opens the memory cells in a row, the potentials on the bit and $\overline{bit}$ lines start changing. The resulting difference in potential is either positive or negative, depending on the

data stored in the cell. When the difference in potential between the lines has reached some specified voltage level ΔV, the memory state can be detected by the (differential) sense amplifier. There exist various sense amplifiers, see for example Bakoglu [14, Ch. 4.9] and Annaratone [8, Ch. 6.4.3].

By properly precharging the bit and $\overline{bit}$ lines and choosing a suitable type of sense amplifier, the memory cell contents can be detected very quickly. Let t_{cell} denote the time needed for a memory cell to induce the potential difference ΔV between the bit and $\overline{bit}$ lines and let t_{sense} denote the sense amplifier delay time. Then, the delay associated with the second part of the memory critical path equals $t_{\text{detect}} = t_{\text{cell}} + t_{\text{sense}}$. It can be shown that t_{detect} can be minimised to be approximately proportional to $\log_2 (C_{\text{bit}}/C_g)$, where C_{bit} is the total capacitive load at a bit line and C_g is the transistor gate capacitance. We refer to Svensson et al. [98], McCarroll et al. [64], and Mohsen and Mead [67].

Let C_d denote the drain capacitance of a CMOS transistor. For a $(2^a \times 2^b)$-bit memory, where each memory cell has a transistor drain connected to the bit line (and another transistor drain connected to the $\overline{bit}$ line), we get[15] $C_{\text{bit}} \approx 2^a C_d$. Assuming that the drain capacitance is approximately equal to the gate capacitance C_g, we get

$$t_{\text{detect}} \propto \log_2 2^a = a.$$

Then, the length of the second part of the critical path, which has delay time t_{detect}, is about $\mathcal{L}_2 = a L_2$, where L_2 is some constant.

Hence, the access time of the $(2^a \times 2^b)$-bit memory in Figure 7.9 is proportional to the length of its critical path, which in turn is approximately equal to

$$\mathcal{L}_{\text{RAM}} = \mathcal{L}_1 + \mathcal{L}_2 = (b+1)L_1 + a L_2. \tag{7.43}$$

The look-up table used in the algorithm described in Section 7.5.1 has size $2^c \times m$ bits, where $c = m - t - 1$ and $m = 2^t$. Using the above notations, we have $u = c$, $w = \log_2 m = t$, and thus $a + b = c + t = m - 1$. Because $m - 1$ is odd, the memory array associated with the table can not be square. Instead, we let $a = b + 1$ (or alternatively $a = b - 1$) which implies $a = m/2$ and $b = m/2 - 1$. Then, the size $\mathcal{C}_{\text{exp,tab}}$ of the memory in which the $(2^c \times m)$-bit look-up table is stored is approximately equal to $\mathcal{C}_{\text{RAM}}\,|_{(a,b)=(m/2,m/2-1)}$, i.e. we get

$$\mathcal{C}_{\text{exp,tab}} \approx (12 \cdot 2^{m/2} + 3m - 2)2^{m/2-2} \approx 3 \cdot 2^m. \tag{7.44}$$

The critical path through the memory equals

$$\mathcal{L}_{\text{exp,tab}} \approx \mathcal{L}_{\text{RAM}}\,|_{(a,b)=(m/2,m/2-1)} = m L_{\text{exp,tab}}, \tag{7.45}$$

where $L_{\text{exp,tab}} = (L_1 + L_2)/2$ is some constant.

[15] We do not include the wire capacitance.

Remark: Note that the memory size $\mathcal{C}_{\text{RAM}}$ and the length $\mathcal{L}_{\text{RAM}}$ of the critical path trough the memory in Figure 7.9 are *approximate* reflections of the true chip area occupied by the memory and its true access time, respectively.

7.6.2 The Discrete Logarithm

In Section 7.4.2, the algorithm for computing the discrete logarithm without using any table involves the recursive computation of d_k from d_{k-1} (see Step 2 on page 178). An architecture for this computation was considered in Section 7.6.1.

An architecture for computing $a_i|_{a_0}$ is needed in the algorithm described in Section 7.5.1 (Step 2 on page 185). Such an architecture is also considered in Section 7.6.1.

In Section 7.5.5, page 196, we describe how to correct an erroneous look-up table output by letting each of the $\log_2 m$ most significant bits and the least significant bit of the NBC output integer pass through an XOR gate. Figure 7.11 shows how the table output is modified by the XOR gates. When the control signal $ctrl$ equals 1, each XOR gates inverts its signal taken from the table. For $ctrl = 0$, the XOR gates do not change the table output bits. Let p_2 and q_2 be defined as in Section 7.5.5. The erroneous table mapping occurs for $p_2 = 3 \cdot 2^{m-4} - 1$, which is the *only* case where the carry out, say c_2, from the most significant bit position of the sum $q_2 = p_2^{(m-5)} + 1$ equals one (1).

Let $p \in \Pi_{24}$ be an integer which maps to an entry of the look-up table. Then, $p_{m-4} = 0$ if $p \in \Pi_2$ and $p_{m-4} = 1$ if $p \in \Pi_4$. Hence, the control signal can be formed by the Boolean function

$$ctrl = c_2 \cdot \overline{p_{m-4}} = \overline{\overline{c_2} + p_{m-4}}.$$

Note that if we define $c_2 = 0$ whenever $p \notin \Pi_2$, we simply get $ctrl = c_2$.

For $m \geq 8$, the above-mentioned look-up table used when computing the discrete logarithm has size $2^{m-4} \times m$ bits, see the end of Section 7.5.5. We do not consider the simple look-up tables used when $m = 2$ and $m = 4$. When $\log_2 m$ is even (i.e. when $m = 16$) we let $a = b = (m - 4 + \log_2 m)/2$ in (7.42) and (7.43) and when $\log_2 m$ is odd (i.e. when $m = 8$) we let $(a, b) = (b + 1, (m - 5 + \log_2 m)/2)$. Then, the size of the memory which realises the $(2^{m-4} \times m)$-bit table and the length of the critical path through that memory

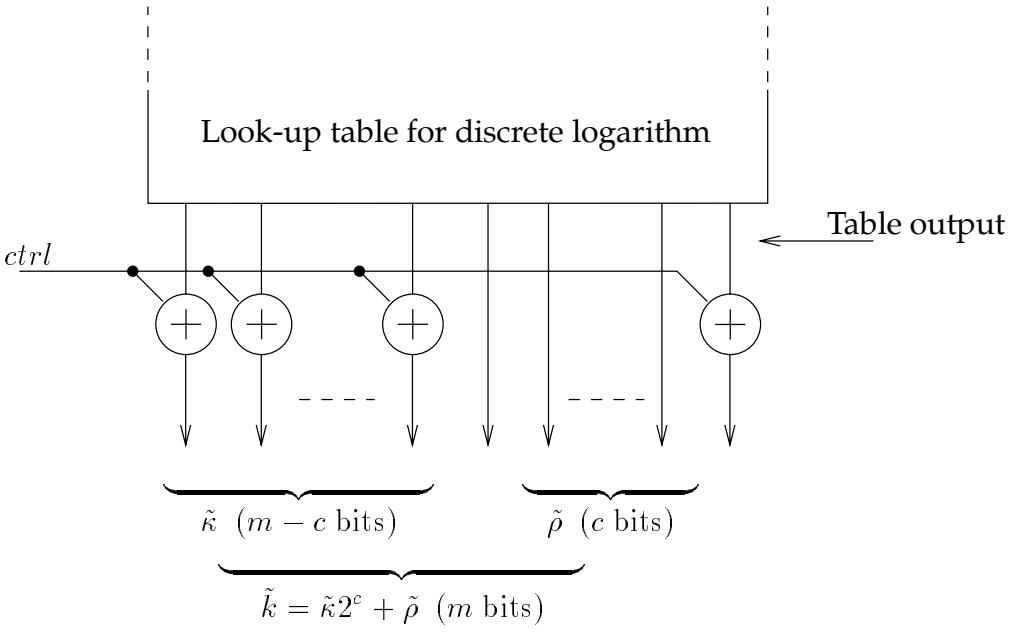


Figure 7.11: *Correcting the one case of erroneous output from the table used when computing the discrete logarithm.*

are equal to

$$\mathcal{C}_{\log,\mathrm{tab}} \approx \mathcal{C}_{\mathrm{RAM}} \approx 3m \cdot 2^{m-3} \tag{7.46}$$

$$\mathcal{L}_{\log,\mathrm{tab}} \approx \mathcal{L}_{\mathrm{RAM}} \approx (m - 4 + \log_2 m) L_{\log,\mathrm{tab}}\,, \tag{7.47}$$

respectively, where $L_{\log,\mathrm{tab}}$ is some constant.

7.6.3 Negation

From (7.6) we have $\hat{\varphi} = P(-\gamma) = \overline{\hat{\gamma}_{m-1}} 2^{m-1} + \hat{\gamma}^{(m-2)} \pmod{2^m}$, where γ is a nonzero integer of $\mathbb{Z}_{2^m+1}$. Thus, for $\hat{\gamma} \in \mathbb{Z}_{2^m}$ (i.e. for $\hat{\gamma}_m = 0$), $\hat{\varphi}$ is obtained from $\hat{\gamma}$ simply by inverting the digit $\hat{\gamma}_{m-1}$. If $\gamma = 0$ we let $\hat{\varphi} = \hat{\gamma} = P(0) = 2^m$. Consequently we get

$$\begin{aligned} \hat{\varphi}_m &= \hat{\gamma}_m \\ \hat{\varphi}_{m-1} &= \overline{\hat{\gamma}_m} \cdot \overline{\hat{\gamma}_{m-1}} = \overline{\hat{\gamma}_m + \hat{\gamma}_{m-1}} \\ \hat{\varphi}^{(m-2)} &= \hat{\gamma}^{(m-2)}. \end{aligned}$$

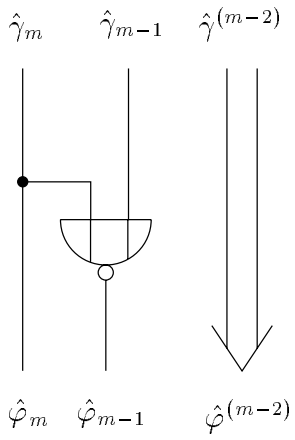


Figure 7.12: *Negation in the polar representation.*

Figure 7.12 shows an architecture for performing negation with respect to the polar representation. The CP through the not so complicated circuit runs from the most significant bit input to the output of the NOR gate. The size, fan-in, and output normalised resistance of the circuit equal

$$\begin{aligned} \mathcal{C}_{\text{polneg}} &= \mathcal{C}_{\text{NOR}} = 4 \\ f_{\text{polneg}} &= n_{\text{polneg}} + f_{\text{NOR}} = n_{\text{polneg}} + 2 \\ r_{\text{polneg}} &= r_{\text{NOR}} = 2, \end{aligned}$$

respectively, where n_{polneg} is the fan-out of the circuit, with respect to the $\hat{\varphi}_m$-output node. Assuming that the input $\hat{\gamma}$ is obtained from a parallel register and the output $\hat{\varphi}$ is also stored in a register,[16] the time required for performing negation in the polar representation is proportional to the length

$$\mathcal{L}_{\text{polneg}} = \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{polneg}} + r_{\text{polneg}} f_{\text{reg}} = 34$$

and hence the area-time performance of the architecture is proportional to

$$\mathcal{C}\mathcal{L}^2_{\text{polneg}} \triangleq \mathcal{C}_{\text{polneg}} (\mathcal{L}_{\text{polneg}})^2 = 4624.$$

[16] Like we did in Chapters 5 and 6.

7.6.4 Addition

For nonzero $\beta, \gamma \in \mathbb{Z}_{2^m+1}$, i.e. for $\hat{\beta}, \hat{\gamma} \in \mathbb{Z}_{2^m}$, the following congruence was given in (7.9):

$$\hat{\varphi} \equiv P(\beta+\gamma) \equiv \hat{\beta} + Z\left(\hat{\gamma} + \overline{\hat{\beta}^{(m-1)}} + 1\right) \pmod{2^m}, \tag{7.48}$$

where $\overline{\hat{\beta}^{(m-1)}} = 2^m - 1 - \hat{\beta}^{(m-1)}$ is the one's complement of $\hat{\beta}^{(m-1)}$. With respect to area complexity, this congruence is preferably computed in two clock cycles, using an m-bit feedback parallel adder. Let $\hat{\theta}$ denote the sum output of the adder. During the first clock interval we compute the Zech logarithm $Z\left(\hat{\gamma}^{(m-1)} + \overline{\hat{\beta}^{(m-1)}} + 1\right)$. The two adder inputs are $\hat{\gamma}^{(m-1)}$ and $\overline{\hat{\beta}^{(m-1)}}$ and the first carry input signal equals 1. The m-bit sum $\hat{\theta} \equiv \hat{\gamma}^{(m-1)} + \overline{\hat{\beta}^{(m-1)}} + 1 \pmod{2^m}$ is the input of a circuit which outputs the m-bit Zech logarithm $Z(\hat{\theta})$. However, because we have defined $Z(2^{m-1}) = * \triangleq 2^m$, the Zech's logarithm circuit will generate an erroneous output when the input $\hat{\theta}$ equals 2^{m-1}. This situation is handled by defining an additional output signal, say Z_{ind}, which indicates whether the logarithm output of the circuit is the correct Zech logarithm of its input $\hat{\theta}$:

$$\begin{array}{lll} \hat{\theta} \neq 2^{m-1} & \Longrightarrow & \text{Correct output } Z(\hat{\theta}), \quad Z_{\text{ind}} = 0 \\ \hat{\theta} = 2^{m-1} & \Longrightarrow & \text{Incorrect output } Z(\hat{\theta}), \quad Z_{\text{ind}} = 1 \end{array} .$$

During the second clock interval, the *new* adder output $\hat{\theta}$ equals the sum $\hat{\beta}^{(m-1)} + Z(\hat{\theta}) \bmod 2^m$, i.e. $\hat{\theta}_{\text{second}} \equiv \hat{\beta}^{(m-1)} + Z(\hat{\theta}_{\text{first}})$. Thus, the adder input signals are $\hat{\beta}^{(m-1)}$ and $Z(\hat{\theta})$ and the carry input signal equals 0. If $Z_{\text{ind}} = 0$, the desired sum $\hat{\varphi}$ equals the adder output $\hat{\theta}$. If $Z_{\text{ind}} = 1$, we let $\hat{\varphi} = * = 2^m$. If both β and γ are zero, i.e. if $\hat{\beta} = \hat{\gamma} = 2^m$, we also let $\hat{\varphi} = 2^m$. If only β (or γ) equals zero, we let $\hat{\varphi} = P(0+\gamma) = \hat{\gamma}$ (or $\hat{\varphi} = \hat{\beta}$).

Figure 7.13 shows an architecture for "polar" addition, which is based on the procedure described above. The m-bit parallel registers R_1 and R_2 are initially loaded with $\hat{\beta}^{(m-1)}$ and $\hat{\gamma}^{(m-1)}$, respectively, and the D flip-flops D_1, D_2, and D_3 are loaded with 1, $\hat{\beta}_m$, and $\hat{\gamma}_m$, respectively. The number of parallel wires in every signal bus ("$\Longrightarrow$") in the figure equals m. Consequently, the inverter which has the m-bit contents of register R_1 as its input signal is actually a *row* of m ordinary one-bit inverters.

The desired output signal $\hat{\varphi}$ is formed by the output controller circuit in the bottom-leftmost part of Figure 7.13. Table 7.3 shows which output is generated for different values of $\hat{\beta}_m$, $\hat{\gamma}_m$, and Z_{ind}. Based on this table, we form the two Karnaugh maps in Figure 7.14 for $\hat{\varphi}_m$ and $\hat{\varphi}_i$, where $0 \leq i \leq m-1$.

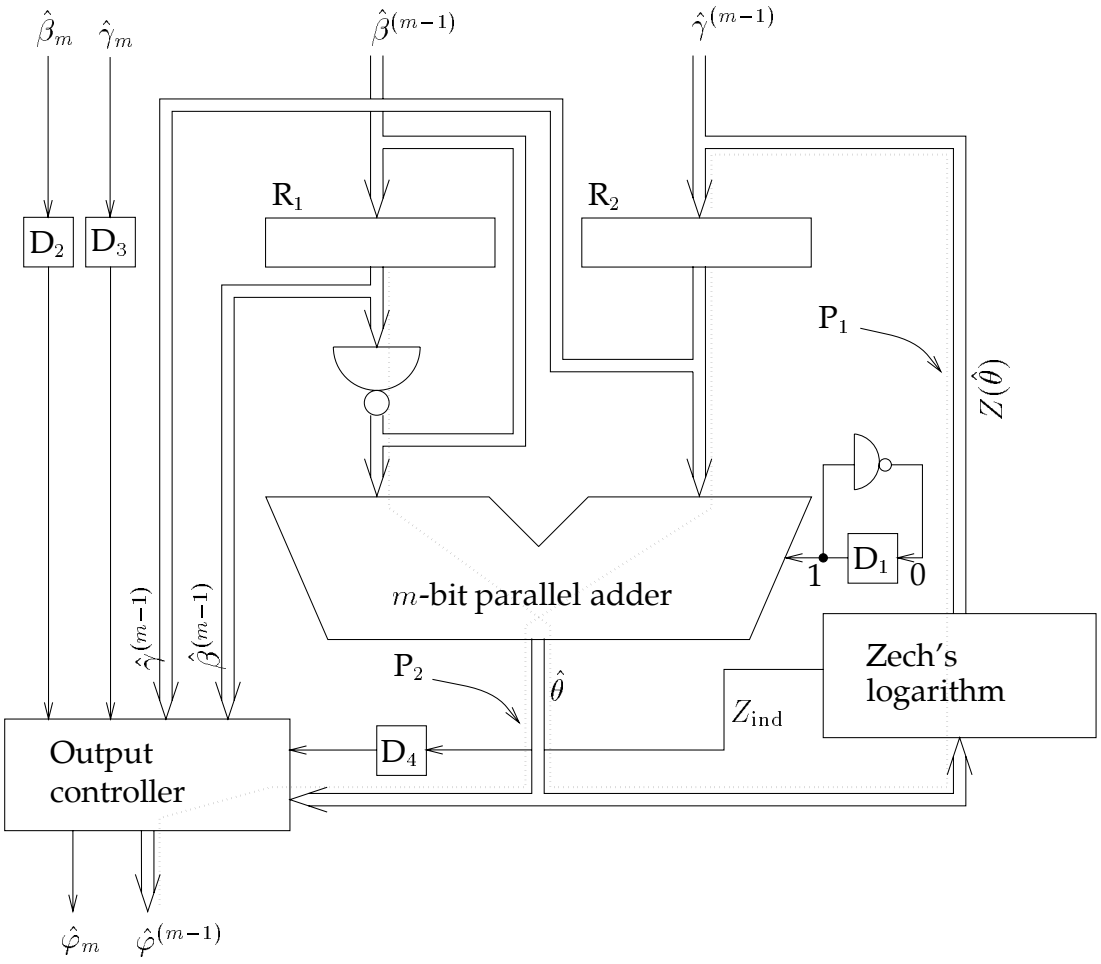


Figure 7.13: *An architecture for addition using the polar representation. The arrangement of the output controller circuit is shown in Figure 7.16.*

$\hat{\beta}_m$	$\hat{\gamma}_m$	Z_{ind}	$\hat{\varphi}_m$	$\hat{\varphi}^{(m-1)}$
1	1	0	$\hat{\gamma}_m = \hat{\beta}_m = 1$	0
1	0	0/1	$\hat{\gamma}_m = 0$	$\hat{\gamma}^{(m-1)}$
0	1	0/1	$\hat{\beta}_m = 0$	$\hat{\beta}^{(m-1)}$
0	0	1	1	0
		0	0	$\hat{\theta}^{(m-1)}$

Table 7.3: *The output* $\hat{\varphi} = \hat{\varphi}_m 2^m + \hat{\varphi}^{(m-1)}$ *for the various values of* $\hat{\beta}_m$, $\hat{\gamma}_m$, *and* Z_{ind}.

$\hat{\varphi}_m$

Z_{ind} \ $\hat{\beta}_m\hat{\gamma}_m$	00	01	11	10
0	0	0	1	0
1	1	0	X	0

$\hat{\varphi}_i;\ 0 \le i \le m-1$

Z_{ind} \ $\hat{\beta}_m\hat{\gamma}_m$	00	01	11	10
0	$\hat{\theta}_i$	$\hat{\beta}_i$	0	$\hat{\gamma}_i$
1	0	$\hat{\beta}_i$	X	$\hat{\gamma}_i$

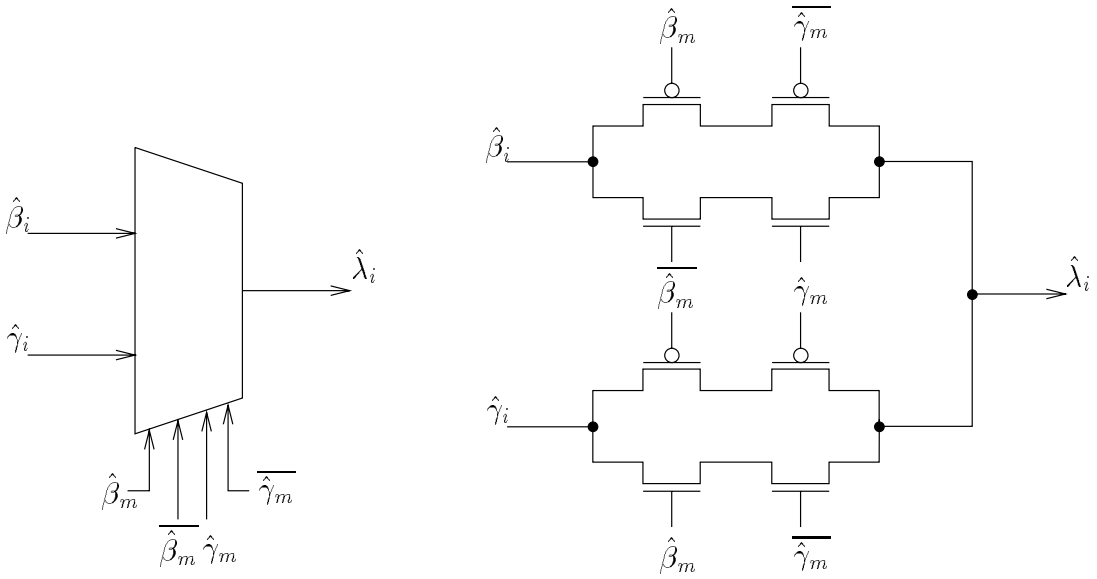
Figure 7.14: *Karnaugh maps for* $\hat{\varphi}_m$ *and* $\hat{\varphi}_i$, *where* $0 \le i \le m-1$. X = *"don't care"*.

From these maps, we obtain the Boolean functions

$$\hat{\varphi}_m = \overline{\hat{\beta}_m} \cdot \overline{\hat{\gamma}_m} \cdot Z_{\text{ind}} + \hat{\beta}_m\hat{\gamma}_m = \overline{\overline{(\hat{\beta}_m + \hat{\gamma}_m)Z_{\text{ind}}} \cdot \overline{\hat{\beta}_m\hat{\gamma}_m}} \tag{7.49}$$

$$\hat{\varphi}_i = \overline{\hat{\beta}_m} \cdot \overline{\hat{\gamma}_m} \cdot \overline{Z_{\text{ind}}} \cdot \hat{\theta}_i + \hat{\lambda}_i = \overline{\overline{(\hat{\beta}_m + \hat{\gamma}_m)} \cdot \overline{(Z_{\text{ind}} + \hat{\theta}_i)} \cdot \overline{\hat{\lambda}_i}}, \tag{7.50}$$

for $0 \le i \le m-1$ and where $\hat{\lambda}_i = \hat{\beta}_i\overline{\hat{\beta}_m}\hat{\gamma}_m + \hat{\gamma}_i\hat{\beta}_m\overline{\hat{\gamma}_m}$. The Boolean function $\hat{\lambda}_i$ can simply be generated using the reduced four-input multiplexer shown in Figure 7.15. This multiplexer lets either $\hat{\beta}_i$ or $\hat{\gamma}_i$ pass to the output $\hat{\lambda}_i$, depending on whether $(\hat{\beta}_m,\ \hat{\gamma}_m)$ equals $(0,\ 1)$ or $(1,\ 0)$, respectively. According to the Boolean function for $\hat{\lambda}_i$, when $(\hat{\beta}_m,\ \hat{\gamma}_m)$ equals $(0,\ 0)$ or $(1,\ 1)$, we should have $\hat{\lambda}_i = 0$. *Therefore, in order to always get the correct output, each output node* $\hat{\lambda}_i$ *of the reduced multiplexer should be discharged (i.e. we set the logical level equal to zero) before the control signals* $\hat{\gamma}_m$ *and* $\hat{\beta}_m$ *and their inverses are present at the multiplexer inputs*. The circuitry for doing this, however, is not considered here.



(a) (b)

Figure 7.15: *A reduced four-input multiplexer. (a) Symbolic description. (b) Schematic description.*

Figure 7.16 shows the structure of the output controller in Figure 7.13. For $0 \leq i \leq m$, the gates of the circuit generate the binary digits $\hat{\varphi}_i$ of $\hat{\varphi} = P(\beta + \gamma)$ where, depending on i, $\hat{\varphi}_i$ is given either by (7.49) or (7.50). The size $\mathcal{C}_{\rm ctrl}$ of the output controller in Figure 7.16 equals

$$\begin{aligned} \mathcal{C}_{\rm ctrl} &= m\mathcal{C}_{\rm RMUX} + (m+2)\mathcal{C}_{\rm reg} + (3m+4)\mathcal{C}_{\rm NAND/NOR} \\ &\quad + (2m+2)\mathcal{C}_{\rm inv} \\ &= 40m + 20, \end{aligned}$$

where $\mathcal{C}_{\rm RMUX} = 8$ is the size of one reduced (four-input) multiplexer. The fan-in, internal delay, and output normalised resistance of the output controller, with respect to the dotted path P_2 in the figure, equal

$$\begin{aligned} f_{\rm ctrl} &= f_{\rm inv} = 2 \\ \mathcal{L}_{\rm ctrl} &= r_{\rm inv} f_{\rm NOR} + r_{\rm NOR} f_{\rm NAND} + r_{\rm NAND} f_{\rm NAND} = 10 \\ r_{\rm ctrl} &= r_{\rm NAND} = 2, \end{aligned}$$

respectively. The chip area A occupied by the entire "polar" adder in Figure 7.13 is proportional to its size

$$\mathcal{C}_{\rm poladd,1} = \mathcal{C}_{\rm Zech} + \mathcal{C}_{m\rm add} + (2m+4)\mathcal{C}_{\rm reg} + (m+1)\mathcal{C}_{\rm inv} + \mathcal{C}_{\rm ctrl}$$

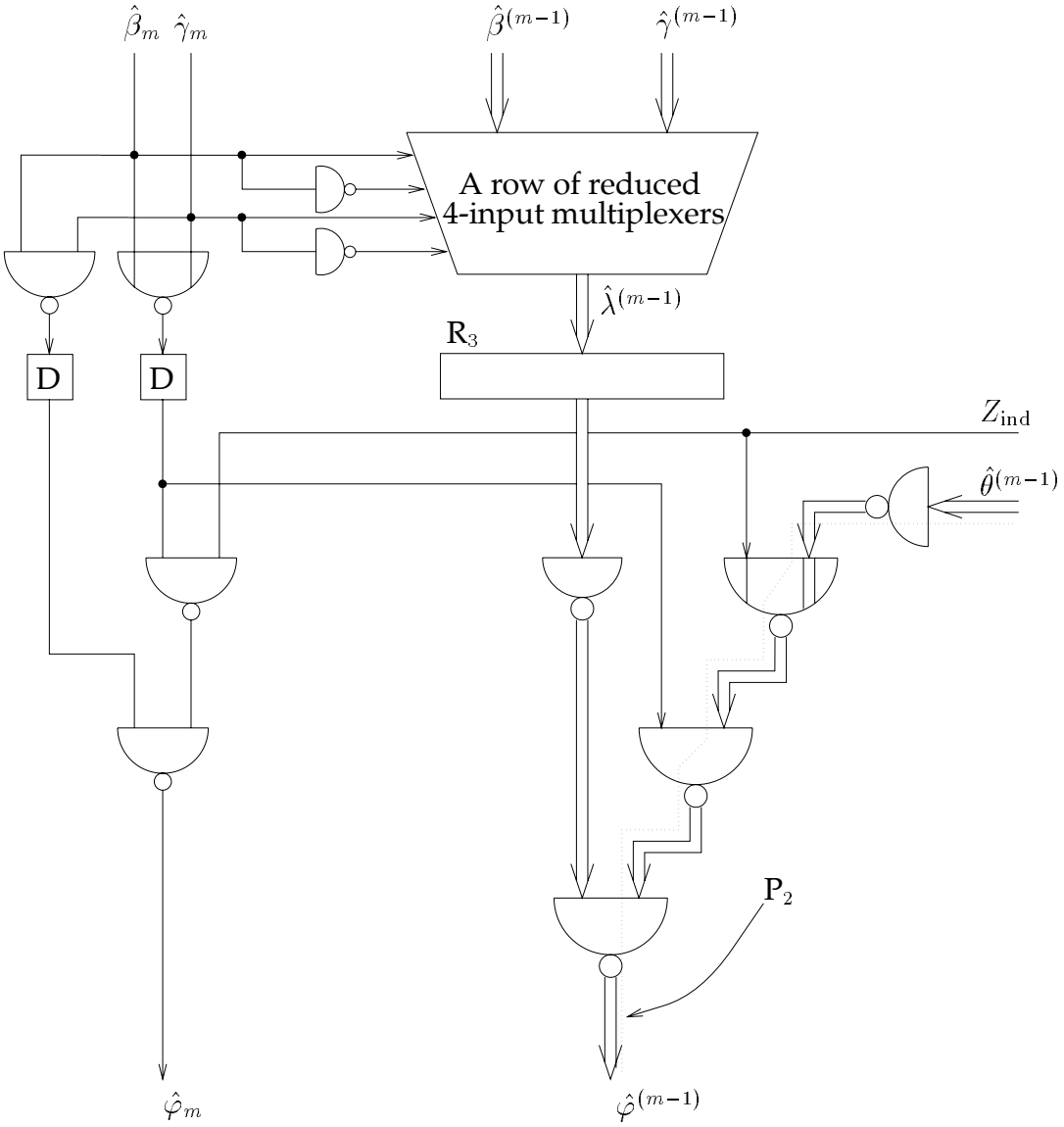


Figure 7.16: *The output controller of Figure* 7.13.

$$= \mathcal{C}_{\text{Zech}} + \mathcal{C}_{m\text{add}} + 74m + 86,$$

where $\mathcal{C}_{m\text{add}}$ is the size of the m-bit parallel adder and $\mathcal{C}_{\text{Zech}}$ is the size of the Zech's logarithm circuit in the bottom-rightmost part of the figure. The Zech logarithm can be computed using one discrete exponentiation and one discrete logarithm, which both can be efficiently computed using look-up tables, see Sections 7.5.1 and 7.5.2. The size of such a Zech's logarithm circuit is dominated by the sizes of the look-up tables. For $m \geq 8$ we thus have[17]

$$\mathcal{C}_{\text{Zech}} \approx \mathcal{C}_{\text{exp,tab}} + \mathcal{C}_{\text{log,tab}} \approx 3 \cdot 2^m + 3m \cdot 2^{m-3}, \tag{7.51}$$

where $\mathcal{C}_{\text{exp,tab}}$ and $\mathcal{C}_{\text{log,tab}}$ are given by (7.44) and (7.46), respectively. Assuming that the parallel adder in Figure 7.13 is an ordinary carry ripple adder, consisting of m full adder elements, we have $\mathcal{C}_{m\text{add}} = m\mathcal{C}_{\text{FA}} = 28m$ and thus

$$\mathcal{C}_{\text{poladd,1}} \approx 3 \cdot 2^m + 3m \cdot 2^{m-3} + 102m.$$

The total CP of the "polar" adder architecture is formed by the two dotted paths P_1 and P2 in the figure, where P_1 is the CP during the first clock interval and P_2 is the CP during the second clock interval of the computation. The length of path P_1, which runs from the output of register R_1 through one inverter, along the carry chain of the carry ripple adder, and through the Zech's logarithm circuit to the input of register R_2, equals

$$\begin{aligned}\mathcal{L}_{\text{P1}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{inv}} + r_{\text{inv}}(f_{\text{reg}} + f_{\text{FA,signal}}) + (m-1)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}} f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}}(f_{\text{ctrl}} + f_{\text{Zech}}) + \mathcal{L}_{\text{Zech}} + r_{\text{Zech}} f_{\text{reg}} \\ &= \mathcal{L}_{\text{Zech}} + f_{\text{Zech}} + 2r_{\text{Zech}} + 14m + 46,\end{aligned}$$

where $\mathcal{L}_{\text{Zech}}$, f_{Zech}, and r_{Zech} are the internal CP length, the fan-in, and the output normalised resistance of the Zech's logarithm circuit. Note that when the path P_1 and P_2 are active we have $(\hat{\beta}_m,\ \hat{\gamma}_m) = (0,\ 0)$, which means that the reduced multiplexers of the output controller are switched off. Hence, the output controller does not affect the output stages of register R_1 and R_2.

Assuming that the sum $\hat{\varphi}$ is stored in a parallel register, the length of path P_2, which runs from the output of register R_2 through the carry ripple adder and the output controller, equals

$$\begin{aligned}\mathcal{L}_{\text{P2}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{FA,signal}} + (m-1)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}} f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}}(f_{\text{ctrl}} + f_{\text{Zech}}) + \mathcal{L}_{\text{ctrl}} + r_{\text{ctrl}} f_{\text{reg}} \\ &= 14m + 52 + f_{\text{Zech}}.\end{aligned}$$

[17] The simpler cases for which $m \leq 4$ should be handled separately.

If either $\hat{\beta}_m$ or $\hat{\gamma}_m$ (or both) equals one, the m-bit NBC integer $\hat{\lambda} = (\hat{\lambda}_{m-1}, \hat{\lambda}_{m-2}, \ldots, \hat{\lambda}_0)_2$ is loaded into register R_3 of the output controller in the *beginning* of the first clock cycle. Then, in the *beginning* of the second clock cycle, the output controller sets $\hat{\varphi}^{(m-1)} = \hat{\lambda}^{(m-1)}$. Hence, if we do not include the time needed for the initiation of the registers (and D flip-flops), the time T required to perform an addition using the polar representation is proportional to[18]

$$\mathcal{L}_{\text{poladd},1} \triangleq \mathcal{L}_{\text{P}1} + \mathcal{L}_{\text{P}2} = \mathcal{L}_{\text{Zech}} + 2f_{\text{Zech}} + 2r_{\text{Zech}} + 28m + 98.$$

As indicated above, a Zech's logarithm can be computed using two look-up tables – one for exponentiation and one for the discrete logarithm. The algorithms for performing discrete exponentiation and computing the discrete logarithm are described in Sections 7.5.1 and 7.5.2, respectively. We conclude that the *worst-case* time for computing a Zech's logarithm using these algorithms is approximately proportional to the length

$$\begin{aligned}\mathcal{L}_{\text{Zech}}|_{\max} &= \mathcal{L}_{\text{exp,tab}} + \mathcal{L}_{\text{log,tab}} + 2\mathcal{L}_{\text{ai}}|_{\max} \\ &\approx mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 128m,\end{aligned}$$

where $\mathcal{L}_{\text{exp,tab}}$ and $\mathcal{L}_{\text{log,tab}}$ are given in (7.45) and (7.47), respectively, and $\mathcal{L}_{\text{ai}}|_{\max} = 64m$ is the maximum value of $\mathcal{L}_{\text{ai}}$ given in (7.41). *Note that* $\mathcal{L}_{\text{ai}}|_{\max}$ *can quite simply be reduced to* $32m$, for example by using an architecture for computing a_i which allows shifting to the right as well as to the left. With $\mathcal{L}_{\text{Zech}} \approx \mathcal{L}_{\text{Zech}}|_{\max}$ and by assuming $f_{\text{Zech}} = 2$ and $r_{\text{Zech}} = 1$, we get

$$\mathcal{L}_{\text{poladd},1} \approx mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 156m + 104. \tag{7.52}$$

The area-time product AT^2 performance of the adder architecture in Figure 7.13 is proportional to

$$\mathcal{CL}^2_{\text{poladd},1} \triangleq \mathcal{C}_{\text{poladd},1}(\mathcal{L}_{\text{poladd},1})^2.$$

Remark: The two clock intervals associated with the respective CP length P_1 and P_2 are *not* equally long.

An Alternative Adder

Let β and γ be *nonzero* integers of the prime field $\mathbb{Z}_{2^m+1}$. Using the congruences $\beta \equiv 1 + d_{\hat{\beta}} \pmod{2^m+1}$ and $\gamma \equiv 1 + d_{\hat{\gamma}} \pmod{2^m+1}$ (see (7.35) in the proof of Theorem 7.7), addition in $\mathbb{Z}_{2^m+1}$ can be expressed as

$$\begin{aligned}\varphi &\equiv \beta + \gamma \equiv 1 + d_{\hat{\beta}} + 1 + d_{\hat{\gamma}} \equiv d_{\hat{\beta}} \oplus d_{\hat{\gamma}} + 1 \\ &\equiv d_{\hat{\varphi}} + 1 \equiv \alpha^{\hat{\varphi}} \pmod{2^m+1},\end{aligned} \tag{7.53}$$

[18] The time for needed for initiating the registers is negligible in comparison with the first and second clock cycle times.

where

$$d_{\hat{\varphi}} \equiv d_{\hat{\beta}} \oplus d_{\hat{\gamma}} \equiv \alpha^{\hat{\varphi}} - 1 \pmod{2^m + 1}$$

and where $\oplus$ denotes diminished–1 addition. Based on (7.53), we can perform polar addition in the following way: First, we compute $d_{\hat{\beta}}$ and $d_{\hat{\gamma}}$ from $\hat{\beta}$ and $\hat{\gamma}$, respectively. This can be done either using direct computations (see the algorithm in Section 7.4.1) or using a look-up table and some binary shifts (see the algorithm in Section 7.5.1). Then, the sum $d_{\hat{\varphi}} \equiv d_{\hat{\beta}} \oplus d_{\hat{\gamma}} \pmod{2^m + 1}$ is formed by the output of a diminished–1 adder (see Section 6.3.4) with $d_{\hat{\beta}}$ and $d_{\hat{\gamma}}$ as its input signals. The desired result $\hat{\varphi} = P(\beta + \gamma)$ is obtained from $d_{\hat{\varphi}}$ either by using direct computations (see the algorithm in Section 7.4.2) or using a look-up table and essentially some binary shifts (see the algorithm in Section 7.5.2).

Figure 7.17 shows a block diagram of a polar adder which is based on the above addition procedure. The adder in the figure is modified to work also for zero addends ($\beta = 0$ and/or $\gamma = 0$). Let

$$\hat{\theta} \equiv (\hat{\beta}_m 2^m + d_{\hat{\beta}^{(m-1)}}) \oplus (\hat{\beta}_m 2^m + d_{\hat{\beta}^{(m-1)}}) \pmod{2^m + 1},$$

where $d_{\hat{\beta}^{(m-1)}}$ and $d_{\hat{\beta}^{(m-1)}}$ are obtained from $\hat{\beta}^{(m-1)}$ and $\hat{\gamma}^{(m-1)}$, respectively, be the output of the diminished–1 adder in Figure 7.17. Then, we have $\hat{\varphi}_m = \hat{\theta}_m$ and $\hat{\varphi}^{(m-1)}$ is obtained from $d_{\hat{\varphi}^{(m-1)}} = \hat{\theta}^{(m-1)}$. This can be understood from the following three special cases:

1. If $\beta, \gamma \neq 0$, i.e. $(\hat{\beta}_m, \hat{\gamma}_m) = (0, 0)$, $0 \leq \hat{\beta}^{(m-1)}, \hat{\gamma}^{(m-1)} \leq 2^m - 1$:

   (a) If $\hat{\theta} = 2^m$, which occurs if $\beta + \gamma \equiv 0 \pmod{2^m + 1}$, then $\hat{\varphi} = * = 2^m$. Thus, we set $\hat{\varphi}_m = \hat{\theta}_m = 1$. Also, $\hat{\varphi}^{(m-1)} = 0$ is obtained from $d_{\hat{\varphi}^{(m-1)}} = \hat{\theta}^{(m-1)} = 0$.

   (b) If $0 \leq \hat{\theta} \leq 2^m - 1$, which occurs if $\varphi \equiv \beta + \gamma \not\equiv 0 \pmod{2^m + 1}$, then we set $\hat{\varphi}_m = \hat{\theta}_m = 0$. Also, $\hat{\varphi}^{(m-1)} \in \mathbb{Z}_{2^m}$ is obtained from $d_{\hat{\varphi}^{(m-1)}} = \hat{\theta}^{(m-1)} \in \mathbb{Z}_{2^m}$.

2. If $\beta \neq 0$ and $\gamma = 0$, i.e. $(\hat{\beta}_m, \hat{\gamma}_m) = (0, 1)$, $0 \leq \hat{\beta}^{(m-1)} \leq 2^m - 1$, and $\hat{\gamma}^{(m-1)} = 0$ (or $\beta = 0$ and $\gamma \neq 0$, i.e. $(\hat{\beta}_m, \hat{\gamma}_m) = (1, 0)$, $\hat{\beta}^{(m-1)} = 0$, and $0 \leq \hat{\gamma}^{(m-1)} \leq 2^m - 1$):
   Then $\hat{\theta} = d_{\hat{\beta}^{(m-1)}}$ (or $\hat{\theta} = d_{\hat{\gamma}^{(m-1)}}$) and thus, we get $\hat{\varphi}_m = \hat{\theta}_m = 0$. Also, $\hat{\varphi}^{(m-1)} = \hat{\beta}$ (or $\hat{\varphi}^{(m-1)} = \hat{\gamma}$) is obtained from $d_{\hat{\varphi}^{(m-1)}} = \hat{\theta}^{(m-1)}$.

3. If $\beta, \gamma = 0$, i.e. $(\hat{\beta}_m, \hat{\gamma}_m) = (1, 1)$, and $\hat{\beta}^{(m-1)}, \hat{\gamma}^{(m-1)} = 0$:
   Then $d_{\hat{\beta}^{(m-1)}} = d_{\hat{\gamma}^{(m-1)}} = 0$, which means that $\hat{\theta} = 2^m$. We have $\varphi \equiv$

$\beta + \gamma = 0 \pmod{2^m + 1}$ and thus $\hat{\varphi}_m = \hat{\theta}_m = 1$. Also, $\hat{\varphi}^{(m-1)} = 0$ is obtained from $d_{\hat{\varphi}^{(m-1)}} = \hat{\theta}^{(m-1)} = 0$. Hence, the output $\hat{\varphi}$ equals $* = 2^m$.

The polar adder in Figure 7.17 works as follows: The computation procedure is split into two parts (clock cycles). During the first clock cycle, $d_{\hat{\gamma}^{(m-1)}}$ is computed from the input $\hat{\gamma}^{(m-1)}$ and is stored, together with $\hat{\gamma}_m$, in the register. During the second clock cycle, $d_{\hat{\beta}^{(m-1)}}$ is first computed from the input $\hat{\beta}^{(m-1)}$ and the addends $\hat{\beta}_m 2^m + d_{\hat{\beta}^{(m-1)}}$ and $\hat{\gamma}_m 2^m + d_{\hat{\gamma}^{(m-1)}}$ appears at the inputs of the diminished-1 adder. Then, the desired sum $\hat{\varphi} = P(\beta + \gamma)$ is computed from the adder output $\hat{\theta}$.

Assuming that the two translation circuits in Figure 7.17 are realised using look-up tables, as described above, the sizes of the input and output translation circuits are approximately equal to $\mathcal{C}_{\text{exp,tab}} \approx 3 \cdot 2^m$ and $\mathcal{C}_{\text{log,tab}} \approx 3m \cdot 2^{m-3}$, respectively.[19] The parameters $\mathcal{C}_{\text{exp,tab}}$ and $\mathcal{C}_{\text{log,tab}}$ are given by (7.44) and (7.46), respectively. Using the carry ripple adder in Figure 6.9, which has size $\mathcal{C}_{\text{dimadd,2}} = 50m + 2$, the total size of the polar adder in Figure 7.17 equals[20]

$$\begin{aligned} \mathcal{C}_{\text{poladd,2}} &\approx \mathcal{C}_{\text{exp,tab}} + \mathcal{C}_{\text{log,tab}} + \mathcal{C}_{\text{dimadd,2}} + (m+1)\mathcal{C}_{\text{reg}} \\ &\approx 3 \cdot 2^m + 3m \cdot 2^{m-3} + 66m, \end{aligned}$$

where $\mathcal{C}_{\text{reg}} = 16$. Hence, the size of the polar adder in Figure 7.17 is *less* than the size of the polar adder in Figure 7.13. Also, the overall structure of the former architecture is simpler than the structure of the latter one.

The total critical path through the circuit in Figure 7.17 is formed by the paths P_1 and P_2, which correspond to the critical paths associated with the first and second clock cycles, respectively. The length $\mathcal{L}_{\text{P}_1}$ of path P_1 is approximately equal to $\mathcal{L}_{\text{exp,tab}} + \mathcal{L}_{\text{ai}} \,|_{\text{max}} = m(L_{\text{exp,tab}} + 64)$ and the length $\mathcal{L}_{\text{P}_2}$ of path P_2 is approximately equal to $\mathcal{L}_{\text{exp,tab}} + \mathcal{L}_{\text{dimadd,2}} + \mathcal{L}_{\text{log,tab}} + 2\mathcal{L}_{\text{ai}} \,|_{\text{max}} \approx mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 146m$ (see (7.45), (6.28), (7.47), and (7.41)). Hence, the total time required to perform polar addition, using the architecture in Figure 7.17, is approximately proportional to the length

$$\begin{aligned} \mathcal{L}_{\text{poladd,2}} &= \mathcal{L}_{\text{P}_1} + \mathcal{L}_{\text{P}_2} \\ &\approx 2mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 210m, \end{aligned}$$

where $L_{\text{exp,tab}}$ and $L_{\text{log,tab}}$ are some constants, By comparing $\mathcal{L}_{\text{poladd,2}}$ with $\mathcal{L}_{\text{poladd,1}}$, which is given in (7.52), we conclude that the polar adder in Figure 7.13 is faster than the adder in Figure 7.17.

[19] Thus, the sum of the sizes of the two translation circuits is equal to $\mathcal{C}_{\text{Zech}}$ (see (7.51)).

[20] Note that here we only consider the cases $m = 8$ and $m = 16$.

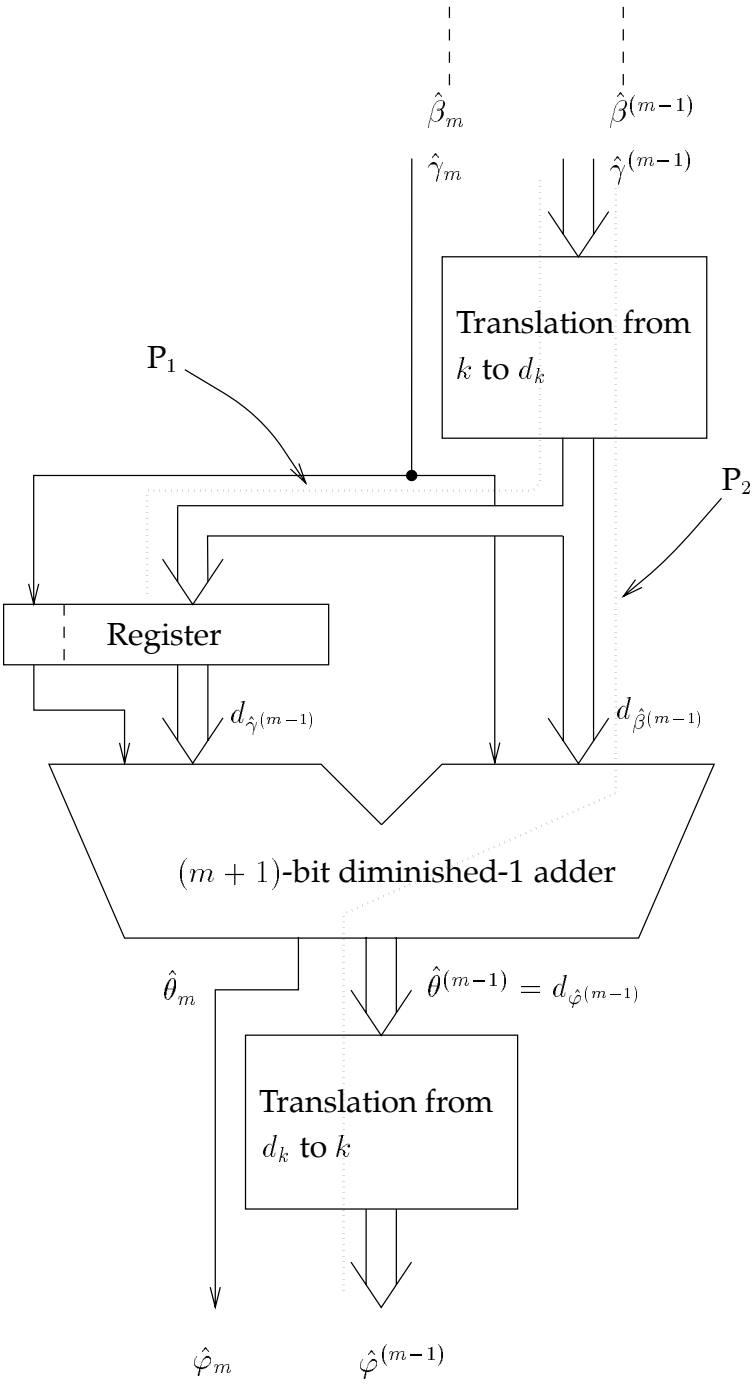


Figure 7.17: *The block diagram of an alternative polar adder. The paths P_1 and P_2 form the CP through the circuit. We have k, $d_k \in \mathbb{Z}_{2^m}$.*

The AT^2 performance of the polar adder in Figure 7.17 is proportional to the product

$$\mathcal{CL}^2_{\mathrm{poladd},2} \stackrel{\triangle}{=} \mathcal{C}_{\mathrm{poladd},2}(\mathcal{L}_{\mathrm{poladd},2})^2.$$

Remark: The two clock intervals associated with the respective CP length P_1 and P_2 are *not* equally long.

Subtraction

By (7.11) and (7.6) we get

$$P(\beta-\gamma) \equiv \hat{\beta} + Z\left(P(-\gamma) + \overline{\hat{\beta}^{(m-1)}} + 1\right) \pmod{2^m}, \tag{7.54}$$

where $P(-\gamma) \equiv \overline{\hat{\gamma}_{m-1}}2^{m-1} + \hat{\gamma}^{(m-2)} \pmod{2^m}$. Hence, by comparing (7.54) with (7.48), we conclude that polar subtraction can be carried out by using the adder architecture in Figure 7.13 but with the input bit $\hat{\gamma}_{m-1}$ exchanged for its one's complement $\overline{\hat{\gamma}_{m-1}}$. An XOR gate can be used to control whether the input bit $\hat{\gamma}_{m-1}$ of $\hat{\gamma}$ is to be inverted (when subtracting) or unchanged (when adding).

Polar subtraction can also be carried out by using a modified version of the polar adder in Figure 7.17. Similar to (7.53), we can write

$$\begin{aligned} \lambda &\equiv \beta - \gamma \equiv 1 + d_{\hat{\beta}} - (1 + d_{\hat{\gamma}}) \equiv d_{\hat{\beta}} + 2^m - 1 - d_{\hat{\gamma}} + 2 \\ &\equiv d_{\hat{\beta}} \oplus \overline{d_{\hat{\gamma}}} + 1 \equiv d_{\hat{\lambda}} + 1 \equiv \alpha^{\hat{\lambda}} \pmod{2^m+1}, \end{aligned} \tag{7.55}$$

where $\overline{d_{\hat{\gamma}}}$ is the one's complement of the m-bit normal binary coded integer $d_{\hat{\gamma}}$ and $d_{\hat{\lambda}} \equiv d_{\hat{\beta}} \oplus \overline{d_{\hat{\gamma}}} \equiv \alpha^{\hat{\lambda}} - 1 \pmod{2^m+1}$. A row of m XOR gates can be placed at the output of the k-to-d_k translation circuit in Figure 7.17 to control whether the output is to be inverted (when subtracting) or unchanged (when adding).

7.6.5 General multiplication

From Section 7.2.6, we get

$$\hat{\varphi} \triangleq P(\beta\gamma) \equiv \begin{cases} \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} \pmod{2^m}; & \text{if } \hat{\beta}_m, \hat{\gamma}_m = 0 \\ (=) * = 2^m; & \text{if } \hat{\beta}_m = 1 \text{ and/or } \hat{\gamma}_m = 1 \end{cases}, \tag{7.56}$$

which we compute as follows. Let $\hat{\theta} \triangleq \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} \pmod{2^m}$. Then, by (7.56) and for $0 \le i \le m-1$, each output bit $\hat{\varphi}_i$ of $\hat{\varphi}$ can be written as the Boolean function

$$\hat{\varphi}_i = \hat{\theta}_i \tau, \tag{7.57}$$

where $\tau = \overline{\hat{\beta}_m} \cdot \overline{\hat{\gamma}_m} = \overline{\hat{\beta}_m + \hat{\gamma}_m}$ indicates whether $\hat{\varphi} \equiv \hat{\beta}^{(m-1)} + \hat{\gamma}^{(m-1)} \pmod{2^m}$ ($\Rightarrow \tau = 1$) or $\hat{\varphi} = 2^m$ ($\Rightarrow \tau = 0$). The most significant bit $\hat{\varphi}_m$ of $\hat{\varphi}$ equals $\overline{\tau}$.

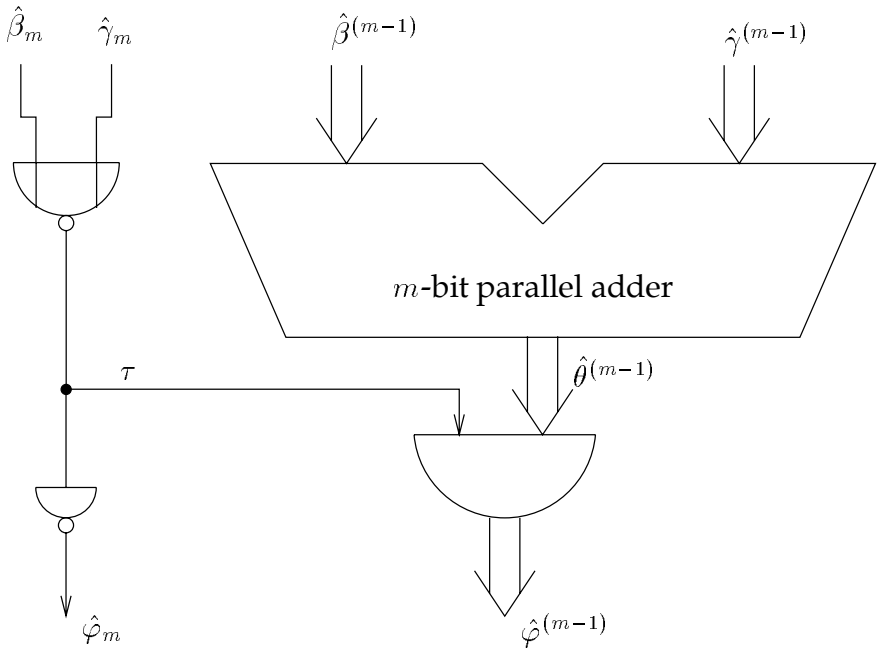


Figure 7.18: *A bit-parallel architecture for general multiplication using the polar representation.*

A Bit-Parallel Architecture

An architecture for general "polar" multiplication, which is based on the above procedure, is shown in Figure 7.18. The AND gate in the figure represents a row of m AND gates, each which, for some $i = 0, 1, \ldots, m-1$, generates the output bit $\hat{\varphi}_i$ according to the Boolean function in (7.57).

Assuming that the parallel adder in Figure 7.18 is an ordinary carry ripple adder, which comprises m full adder elements, the chip area A occupied by the general multiplier architecture is proportional to its size

$$\begin{aligned} \mathcal{C}_{\text{polmult,par}} &= m\mathcal{C}_{\text{FA}} + m\mathcal{C}_{\text{AND}} + \mathcal{C}_{\text{NOR}} + \mathcal{C}_{\text{inv}} \\ &= 34m + 6. \end{aligned}$$

The length of the internal CP, which is the path from the least significant bit input of the parallel adder, through the chain of full adder elements to the output of the AND gate in bit position $m-1$. The length of this CP equals

$$\begin{aligned} \mathcal{L}_{\text{CP,polmult,par}} &= (m-1)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}} f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}} f_{\text{AND}} + \mathcal{L}_{\text{AND}} \\ &= 14m + 4. \end{aligned}$$

The fan-in and the output normalised resistance of the architecture are equal to

$$\begin{aligned} f_{\text{polmult,par}} &= f_{\text{FA,signal}} = 8 \\ r_{\text{polmult,par}} &= r_{\text{AND}} = 1, \end{aligned}$$

respectively. Assuming as before that the circuit inputs are obtained directly from some parallel registers and the output $\hat{\varphi}$ is directly stored in a parallel register, the total computation time T is proportional to

$$\begin{aligned} \mathcal{L}_{\text{polmult,par}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{polmult,par}} + \mathcal{L}_{\text{CP,polmult,par}} + r_{\text{polmult,par}} f_{\text{reg}} \\ &= 14m + 44. \end{aligned}$$

Hence, the area-time product AT^2 is proportional to

$$\begin{aligned} \mathcal{C}\mathcal{L}^2_{\text{polmult,par}} &\triangleq \mathcal{C}_{\text{polmult,par}} (\mathcal{L}_{\text{polmult,par}})^2 \\ &= (34m + 6)(14m + 44)^2 \end{aligned}$$

A Bit-Serial Architecture

In the beginning of Chapter 4 we stated that, depending on the modulus, bit-serial architectures are often impracticable for arithmetic operations in integer quotient rings. However, when using the polar representation of the integers of $\mathbb{Z}_{2^m+1}$, all computations are carried out modulo 2^m. Because the reduction modulo 2^m can be carried out instantaneously, bit-serial architectures may be competitive for some arithmetic operations.

In Figure 7.19 we show a bit-serial architecture for general "polar" multiplication, which is based on the parallel multiplier in Figure 7.18. The chip area A occupied by this multiplier is proportional to its size

$$\begin{aligned} \mathcal{C}_{\text{polmult,ser}} &= \mathcal{C}_{\text{FA}} + (2m + 3)\mathcal{C}_{\text{reg}} + \mathcal{C}_{\text{AND}} + \mathcal{C}_{\text{NOR}} + \mathcal{C}_{\text{inv}} \\ &= 32m + 88. \end{aligned}$$

During an initial clock cycle, the m-bit shift registers R_1 and R_2 are loaded with $\hat{\beta}^{(m-1)}$ and $\hat{\gamma}^{(m-1)}$, respectively, and the D flip-flops D_1, D_2, and D_3 are loaded with $\hat{\beta}_m$, $\hat{\gamma}_m$, and zero, respectively. Then, during the m subsequent clock cycles, the digits $\hat{\varphi}_0, \hat{\varphi}_1, \ldots, \hat{\varphi}_{m-1}$ are shifted into the feedback shift register R_2. Hence, after a total of $m + 1$ clock cycles, the result $\hat{\varphi}^{(m-1)}$ is contained in register R_2.

The CP is the dotted path from the serial output of R_1 (or R_2) to the serial input of R_2. The length of this path equals

$$\begin{aligned} \mathcal{L}_{\text{CP,polmult,ser}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{FA,signal}} + \mathcal{L}_{\text{FA,sum}} + r_{\text{FA}} f_{\text{AND}} + \mathcal{L}_{\text{AND}} + r_{\text{AND}} f_{\text{reg}} \\ &= 58, \end{aligned}$$

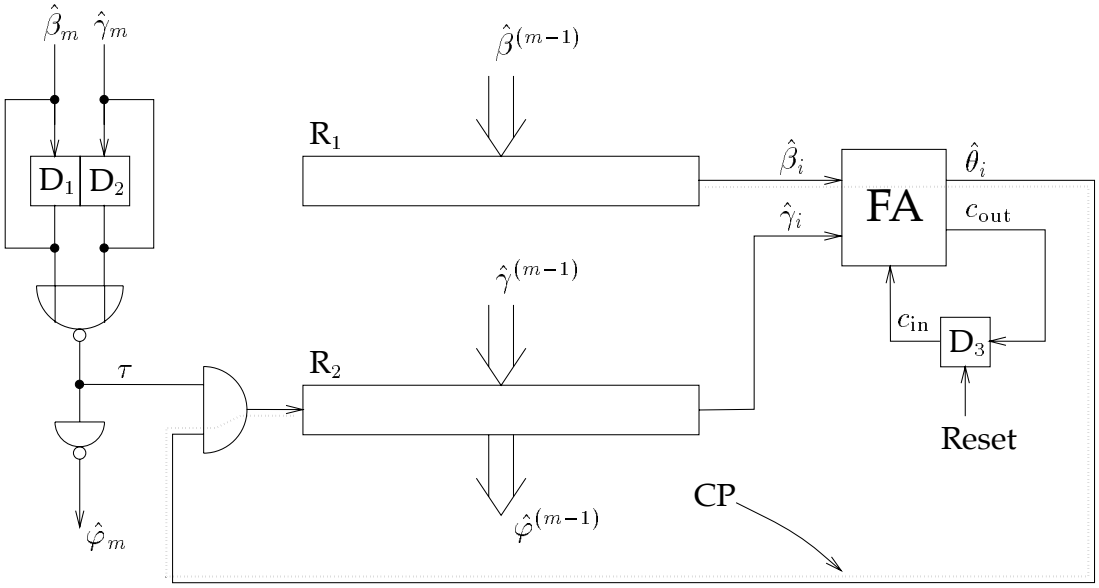


Figure 7.19: *A bit-serial architecture for general multiplication using the polar representation.*

which implies that the total computation time T is proportional to

$$\mathcal{L}_{\text{polmult,ser}} = (m+1)\mathcal{L}_{\text{CP,polmult,ser}} = 58(m+1)$$

and the AT^2 performance is proportional to

$$\begin{aligned} \mathcal{CL}^2_{\text{polmult,ser}} &\triangleq \mathcal{C}_{\text{polmult,ser}}(\mathcal{L}_{\text{polmult,ser}})^2 \\ &= (32m+88)(58(m+1))^2. \end{aligned}$$

In Figure 7.20 we have plotted the parameters $\mathcal{C}_{\text{polmult,par}}$, $\mathcal{C}_{\text{polmult,ser}}$, $\mathcal{L}_{\text{polmult,par}}$, $\mathcal{L}_{\text{polmult,ser}}$, $\mathcal{CL}^2_{\text{polmult,par}}$, and $\mathcal{CL}^2_{\text{polmult,ser}}$ versus m for $m = 2, 4, 8, 16$. Obviously, the bit-parallel architecture in Figure 7.18 is superior to the bit-serial architecture in Figure 7.19 with respect to both chip area and computation time and, consequently, also with respect to area-time performance. Note, however, that the size (area) of the bit-parallel architecture becomes greater than the size of the bit-serial architecture if the input and output registers are included in the size parameter $\mathcal{C}_{\text{polmult,par}}$.

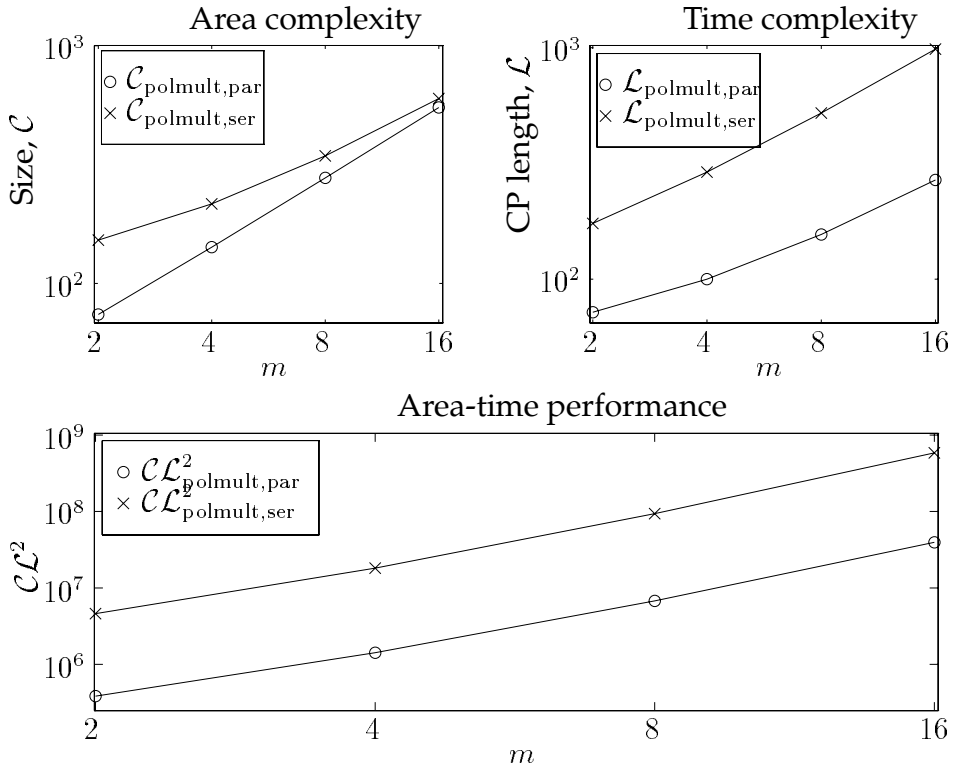


Figure 7.20: *The sizes $\mathcal{C}$, lengths $\mathcal{L}$, and AT^2 performances $\mathcal{C}\mathcal{L}^2$ of the bit-parallel and the bit-serial polar multipliers in Figure* 7.18 *and Figure* 7.19*, respectively. The parameters are plotted versus m for $m = 2, 4, 8, 16$.*

7.6.6 Multiplication by powers of ω

One of the major attributes of the polar representation of the elements of $\mathbb{Z}_{2^m+1}$ follows from Corollary 7.1: When computing a Fermat number transform of length $N = 2^b$ using the transform kernel $\omega \equiv \alpha^{2^{m-b}} \pmod{2^m+1}$, each multiplication by ω can be performed as one b-bit addition modulo 2^b.
Let γ be a nonzero integer of $\mathbb{Z}_{2^m+1}$ and $0 \leq b \leq m$. By Definition 7.3, the polar integer $P(\gamma) = \hat{\gamma} \in \mathbb{Z}_{2^m}$ can be written on the form $\hat{\gamma} = \hat{\gamma}_{(m-b)}2^{m-b} + \hat{\gamma}^{(m-b-1)}$, where $\hat{\gamma}_{(m-b)}$ is formed by the b most significant bits and $\hat{\gamma}^{(m-b-1)}$ is formed by the $m-b$ least significant bits of $\hat{\gamma}$. By (7.21), (7.22) and (7.23), the polar representation of the product $\varphi \equiv \gamma\omega^n \equiv \gamma\alpha^{n2^{m-b}} \pmod{2^m+1}$ equals

$$\begin{aligned} P(\gamma\omega^n) = \hat{\varphi} &= \hat{\varphi}_{(m-b)}2^{m-b} + \hat{\varphi}^{(m-b-1)} \\ &\equiv \hat{\gamma} + \hat{\theta} \pmod{2^m}, \end{aligned}$$

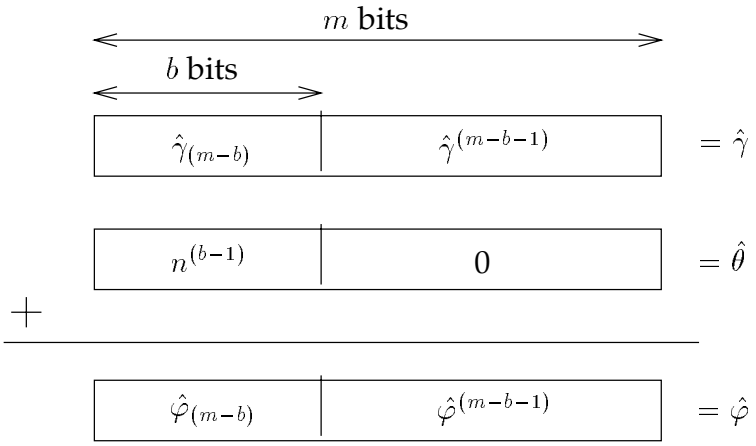


Figure 7.21: *Computation of* $\hat{\varphi} \equiv \hat{\gamma} + \hat{\theta} \pmod{2^m}$, *where* $\hat{\theta} = n^{(b-1)}2^{m-b}$.

where

$$\begin{cases} \hat{\varphi}_{(m-b)} & \equiv & \hat{\gamma}_{(m-b)} + \hat{\theta}_{(m-b)} \pmod{2^b} \\ \hat{\varphi}^{(m-b-1)} & = & \hat{\gamma}^{(m-b-1)} \end{cases},$$

where in turn we have $\hat{\theta}_{(m-b)} = n^{(b-1)}$. Obviously, $P(\omega^n\gamma)$ can be computed using only one b-bit addition of $\hat{\gamma}_{(m-b)}$ and $n^{(b-1)}$ modulo 2^b. This is illustrated in Figure 7.21.

7.6.6.1 Fixed Architectures

An architecture which computes $\hat{\varphi} = P(\omega^n\gamma)$, for some *fixed* $b \in [0, m]$, is shown in Figure 7.22. Let $\hat{\lambda} \equiv \hat{\gamma}_{(m-b)} + n^{(b-1)} \pmod{2^b}$ denote the b-bit output of the parallel adder in the figure. Each digit of $\hat{\lambda}$ and each digit of $\hat{\gamma}^{(m-b-1)}$ is forwarded to one of the inputs of a two-input AND gate. The one's complement $\overline{\hat{\gamma}_m}$ of $\hat{\gamma}_m$ is the second input of each AND gate. If $\hat{\gamma} \in \mathbb{Z}_{2^m}$, i.e. if $\hat{\gamma} = 0$, the desired product $\hat{\varphi} \equiv (\hat{\gamma}_{(m-b)} + n^{(b-1)} \bmod 2^b)2^{m-b} + \hat{\gamma}^{(m-b-1)}$ will be present at the circuit output. If $\hat{\gamma} = * = 2^m$, i.e. if $\hat{\gamma} = 1$, the output $\hat{\varphi}^{(m-1)}$ is set equal to zero by the row of AND gates. We always have $\hat{\varphi}_m = \hat{\gamma}_m$.

If the b-bit parallel adder in Figure 7.22 is an ordinary carry ripple adder, the size of the architecture in the figure equals

$$\mathcal{C}_{\text{mult},\omega,\text{par}} = b\mathcal{C}_{\text{FA}} + m\mathcal{C}_{\text{AND}} + \mathcal{C}_{\text{inv}} = 6m + 28b + 2,$$

where $0 \leq b \leq m$. The internal CP of the architecture, which is indicated by the dotted path in the figure, has length

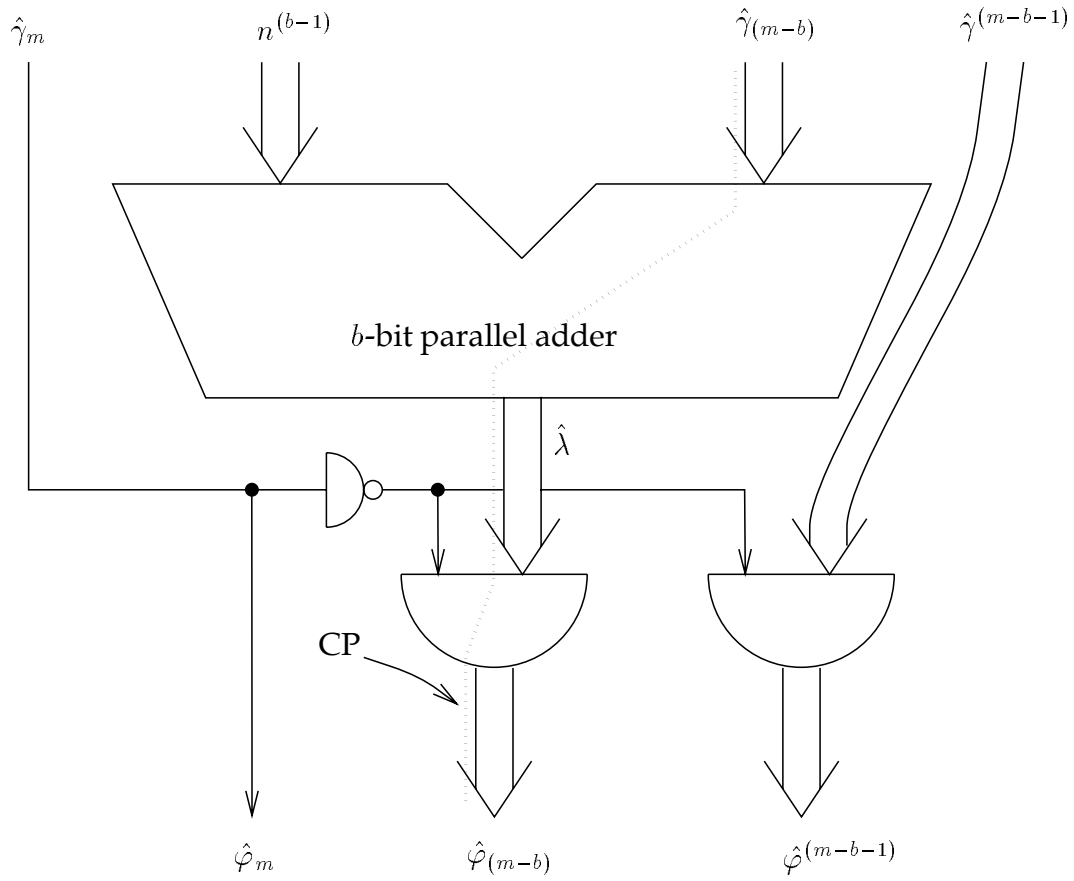


Figure 7.22: *A bit-parallel architecture for computing polar multiplication by powers of ω; $\hat{\varphi} = P(\gamma\omega^n) = \hat{\varphi}_m 2^m + \hat{\varphi}_{(m-b)} 2^{m-b} + \hat{\varphi}^{(m-b-1)}$, where $\mathrm{ord}_{2^m+1}(\omega) = 2^b$ for some fixed $b \in [0, m]$. The output circuitry is formed by a row of $b + (m - b) = m$ two-input AND gates.*

$$\begin{aligned}\mathcal{L}_{\mathrm{CP,mult},\omega,\mathrm{par}} &= (b-1)(\mathcal{L}_{\mathrm{FA,carry}} + r_{\mathrm{FA}} f_{\mathrm{FA,carry}}) + \mathcal{L}_{\mathrm{FA,sum}} + r_{\mathrm{FA}} f_{\mathrm{AND}} + \mathcal{L}_{\mathrm{AND}} \\ &= 14b + 4.\end{aligned}$$

The fan-in and the output normalised resistance , with respect to this CP, are equal to

$$\begin{aligned}f_{\mathrm{mult},\omega,\mathrm{par}} &= f_{\mathrm{FA,signal}} = 8 \\ r_{\mathrm{mult},\omega,\mathrm{par}} &= r_{\mathrm{AND}} = 1,\end{aligned}$$

respectively. With the CP both starting and ending in a register, the total computation time of the the architecture in Figure 7.22 is proportional to

$$\begin{aligned}\mathcal{L}_{\text{mult},\omega,\text{par}} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{mult},\omega,\text{par}} + \mathcal{L}_{\text{CP},\text{mult},\omega,\text{par}} + r_{\text{mult},\omega,\text{par}} f_{\text{reg}} \\ &= 14b + 44\end{aligned}$$

and hence, the area-time performance is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{mult},\omega,\text{par}} &\triangleq \mathcal{C}_{\text{mult},\omega,\text{par}} (\mathcal{L}_{\text{mult},\omega,\text{par}})^2 \\ &= (6m + 28b + 2)(14b + 44)^2.\end{aligned}$$

Note that for $b = m$ we have $\mathcal{L}_{\text{mult},\omega,\text{par}} = \mathcal{L}_{\text{polmult,par}}$ and $\mathcal{C}_{\text{mult},\omega,\text{par}} \approx \mathcal{C}_{\text{polmult,par}}$. For $b < m$ we have $\mathcal{L}_{\text{mult},\omega,\text{par}} < \mathcal{L}_{\text{polmult,par}}$ and $\mathcal{C}_{\text{mult},\omega,\text{par}} < \mathcal{C}_{\text{polmult,par}}$.

A bit-serial architecture for polar multiplication by powers of ω, can be designed in a rather straightforward manner. It is derived from the bit-parallel architecture in Figure 7.22 in the same way as the bit-serial general multiplier in Figure 7.19 was derived from the bit-parallel multiplier in Figure 7.18. Such an architecture would be rather similar to the universal bit-serial architecture in Figure 7.24, which is described below. Therefore, it is not considered here.

7.6.6.2 Universal Architectures

A bit-serial/parallel architecture

So far, all architectures considered in the present chapter, except the one in Figure 7.22, can be used when computing the Fermat number transform of length $N = 2^b$ in $\mathbb{Z}_{2^m+1}$ for some given $m = 2, 4, 8, 16$. The circuit in Figure 7.22 can only be used for some *fixed* $b \in [0, m]$. The bit-serial/parallel architecture in Figure 7.23, however, is a *universal* circuit for multiplication by powers of ω, i.e. it is applicable for *all* possible transform lengths $N = 2^b$, where $b \in [0, m]$. The circuit works as follows.

- During an initial clock cycle, the parallel register R_1 is loaded with $n^{(b-1)}$, shift register R_4 is loaded with $\hat{\gamma}^{(m-1)}$, and the D flip-flop is loaded with $\hat{\gamma}_m$. All registers in the architecture are m bits wide.
- During the following b clock cycles, the transmission gates subsequent to the parallel adder are all closed and the b-bit NBC integer $\hat{\gamma}_{(m-b)}$ is shifted into both register R_2 and R_3. The signal S (*S*hift enable) is a control signal that either enables ($S = 1$) or disables ($S = 0$) the shifting of the contents of the shift registers. Consequently, during the b shifts just mentioned, we have $S = 1$. Each clock interval is proportional to the length

$$\mathcal{L}_{\text{P1}} = \mathcal{L}_{\text{reg}} + r_{\text{reg}} \cdot 2 f_{\text{reg}} = 30$$

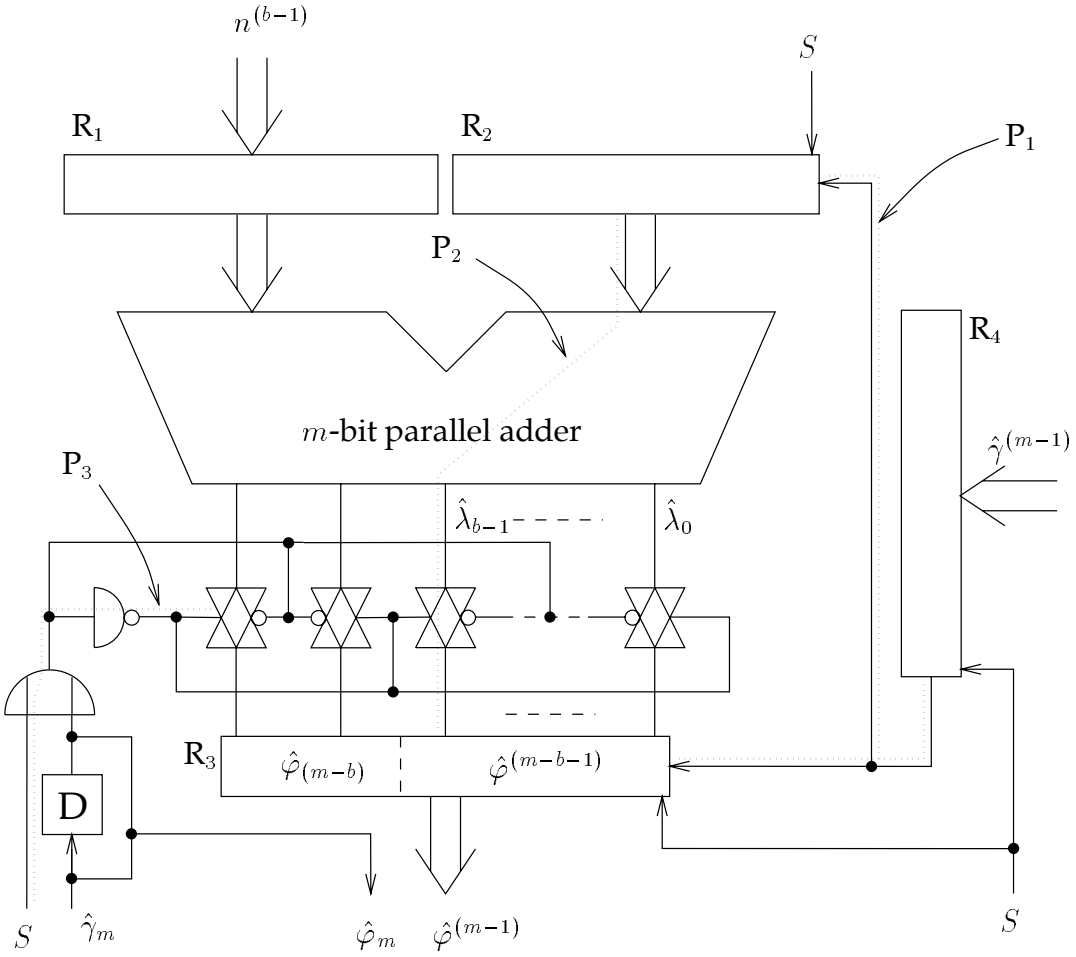


Figure 7.23: *A universal bit-serial/parallel architecture for polar multiplication by powers of ω; $\hat{\varphi} = P(\gamma\omega^n)$, where $\mathrm{ord}_{2^m+1}(\omega) = 2^b$ for any $b \in [0,\, m]$.*

of the dotted path P_1 in Figure 7.23.

- After the b shifts, the control signal S is set to 0 (zero). Let $\hat{\lambda}$ denote the NBC integer which is formed by the b least significant output bits of the parallel (carry ripple) adder. Then we have $\hat{\lambda} \equiv \hat{\gamma}_{(m-b)} + n^{(b-1)} \pmod{2^b}$. The $m - b$ most significant output bits of the adder are redundant.

  1. If $\hat{\gamma}_m = 1$, i.e. if $\gamma = 0$, the transmission gates subsequent to the adder remain closed, so that the contents $\hat{\gamma}_{(m-b)} = (0,\, 0, \ldots, 0)_2$ in the b least significant bit positions of R_3 remain unchanged.

2. If $\hat{\gamma}_m = 0$, i.e. if $\gamma \neq 0$, the transmission gates are open[21] and the register R_3 is loaded with the adder output $\hat{\lambda}$.

The time needed to compute $\hat{\lambda} = \hat{\varphi}_{(m-b)}$ and load it into the b least significant bit positions of R_3 is proportional to the length[22]

$$\begin{aligned}\mathcal{L}_{P2} &= \mathcal{L}_{\text{reg}} + r_{\text{reg}} f_{\text{FA,signal}} + (b-1)(\mathcal{L}_{\text{FA,carry}} + r_{\text{FA}} f_{\text{FA,carry}}) \\ &\quad + \mathcal{L}_{\text{FA,sum}} + (r_{\text{FA}} + 1) f_{\text{reg}} \\ &= 14b + 40\end{aligned}$$

of the dotted path P_2 in the figure. The maximum computation time is obtained for $b = m$, i.e. we have $\mathcal{L}_{P2,\max} \triangleq \max \mathcal{L}_{P2} = 14m + 40$.

- Next, the control signal S is set to 1 (one). This transition closes the transmission gates (if they were open) and enables shifting of the shift register contents. For $\hat{\gamma}_m = 0$, the time to close the transmission gates is proportional to the length

$$\begin{aligned}\mathcal{L}_{P3} &= r_S(f_{OR} + 3f_{S,\text{reg}}) + \mathcal{L}_{OR} + r_{OR}(f_{\text{inv}} + m) + r_{\text{inv}} \cdot m \\ &= r_S(2 + 3f_{S,\text{reg}}) + 2m + 6,\end{aligned}$$

where r_S is the normalised resistance from the S input node of the OR gate to the supply voltage source and $f_{S,\text{reg}}$ is the fan-in of the shift registers, with respect to the control input signal S. By assuming $r_S = 2$ and $f_{S,\text{reg}} = 2$, we get $\mathcal{L}_{P3} = 2m + 22$.

- Finally, during $m - b$ clock cycles, $\hat{\varphi}^{(m-b-1)} = \hat{\gamma}^{(m-b-1)}$ is shifted from R_4 into the $m - b$ least significant bit positions of R_3 while $\hat{\varphi}_{(m-b)} = \hat{\lambda}$ is shifted up to the b most significant bit positions of R_3.

We assume that the registers can be initialised during one cycle of the shift register clock. Then, the total time needed to perform a polar multiplication by a power of ω, using the universal architecture in Figure 7.23, is proportional to

$$\begin{aligned}\mathcal{L}_{\text{univ,mult},\omega,\text{par}} &= (b+1)\mathcal{L}_{P1} + \mathcal{L}_{P2} + \mathcal{L}_{P3} + (m-b)\mathcal{L}_{P1} \\ &= 32m + 14b + 92\end{aligned}$$

[21] The transmission gates have opened before the digit $\hat{\lambda}_{b-1}$ of $\hat{\lambda}$ is present at the adder output.

[22] We assume that, for all $b \leq m$, the length of path P_2 is always greater than the length of path P_3. This is true in virtually all cases.

which, for $b = m$, equals $\mathcal{L}_{\text{univ,mult},\omega,\text{par,max}} = 46m + 92$. The desired result $\hat{\varphi} = P(\hat{\gamma}\omega^n)$ is present at the output of register R_3 (and the D flip-flop). The size of the universal architecture equals

$$\begin{aligned}\mathcal{C}_{\text{univ,mult},\omega,\text{par}} &= m\mathcal{C}_{\text{FA}} + (4m+1)\mathcal{C}_{\text{reg}} + m\mathcal{C}_{\text{TG}} + \mathcal{C}_{\text{OR}} + \mathcal{C}_{\text{inv}} \\ &= 94m + 24,\end{aligned}$$

which implies that its area-time performance is proportional to

$$\begin{aligned}\mathcal{CL}^2_{\text{univ,mult},\omega,\text{par}} &\triangleq \mathcal{C}_{\text{univ,mult},\omega,\text{par}}(\mathcal{L}_{\text{univ,mult},\omega,\text{par}})^2 \\ &= (94m + 24)(32m + 14b + 92)^2.\end{aligned}$$

A bit-serial architecture

In Figure 7.24 we show a universal bit-serial architecture for polar multiplication by powers of ω, which is based on the bit-serial/parallel architecture in Figure 7.23. Also, it is quite similar to the bit-serial architecture for general multiplication, see Figure 7.19. The size of the architecture in Figure 7.24, in which the shift registers R_1 and R_2 are m bits wide, equals

$$\begin{aligned}\mathcal{C}_{\text{univ,mult},\omega,\text{ser}} &= \mathcal{C}_{\text{FA}} + (2m+2)\mathcal{C}_{\text{reg}} + \mathcal{C}_{\text{AND}} + 2\mathcal{C}_{\text{inv}} + \mathcal{C}_{\text{TG}} + 1 \\ &= 32m + 73.\end{aligned}$$

The control signal S is the same signal for shift enabling/disabling that is used in the above universal bit-serial/parallel architecture. During an initial clock cycle, S is set to zero, the registers R_1 and R_2 are loaded with $n^{(b-1)}$ and $\hat{\gamma}^{m-1}$, respectively, and the D flip-flops D_1 and D_2 are loaded with $\hat{\gamma}_m$ and zero, respectively. During the following $m - b$ clock cycles, $\hat{\varphi}^{(m-b-1)}$ is shifted into the most significant bit positions of shift register R_2, i.e. we simply perform an $(m-b)$-bit rotation of the contents of R_2. Then, S is set to one (this is done during one clock cycle) and if $\hat{\gamma}_m = 0$, the b-bit sum $\hat{\lambda} \equiv \hat{\gamma}_{(m-b)} + n^{(b-1)} \pmod{2^b}$ is shifted into register R_2. If $\hat{\gamma}_m = 1$, only zeros are shifted into the register ($\hat{\varphi} = P(0 \cdot \omega^n) = P(0) = 2^m \Rightarrow \hat{\varphi}^{(m-1)} = 0$).

The CP of the multiplier is the dotted path from the serial output of register R_1 to the serial input of register R_2 in Figure 7.24. The clock cycle time is proportional to the length

$$\begin{aligned}\mathcal{L}_{\text{CP,univ,mult},\omega,\text{ser}} &= \mathcal{L}_{\text{reg}} + (r_{\text{reg}} + 1)f_{\text{FA,signal}} + \mathcal{L}_{\text{FA,sum}} \\ &\quad + r_{\text{FA}} f_{\text{AND}} + \mathcal{L}_{\text{AND}} + r_{\text{AND}} f_{\text{reg}} \\ &= 66\end{aligned}$$

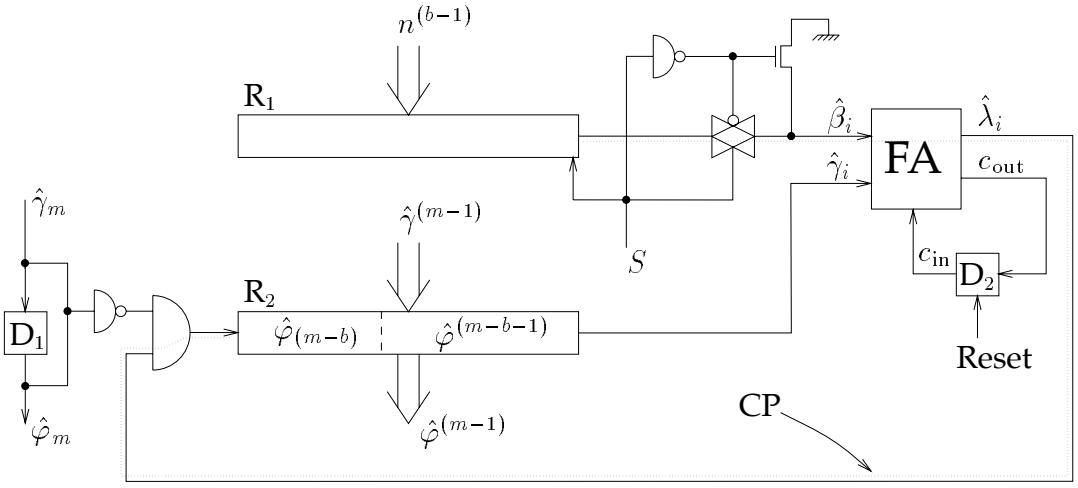


Figure 7.24: *A universal bit-serial architecture for polar multiplication by powers of* ω; $\hat{\varphi} = P(\gamma\omega^n)$, *where* $\text{ord}_{2^m+1}(\omega) = 2^b$ *for any* $b \in [0,\, m]$.

of the CP. Because the desired product $\hat{\varphi} = P(\gamma\omega)$ is obtained in register R_2 after $1 + (m - b) + 1 + b = m + 2$ clock cycles, the total computation time is proportional to

$$\mathcal{L}_{\text{univ,mult},\omega,\text{ser}} = (m+2)\mathcal{L}_{\text{CP,polmult,ser}} = 66(m+2),$$

which implies that the AT^2 performance of the bit-serial architecture is proportional to

$$\begin{aligned} \mathcal{CL}^2_{\text{univ,mult},\omega,\text{ser}} &\triangleq \mathcal{C}_{\text{univ,mult},\omega,\text{ser}}(\mathcal{L}_{\text{univ,mult},\omega,\text{ser}})^2 \\ &= (32m+73)(66(m+2))^2. \end{aligned}$$

The area and time complexities and the area-time performances of the above universal bit-serial/parallel and strictly bit-serial architectures are plotted versus m in Figure 7.25. Note that for $\mathcal{L}_{\text{univ,mult},\omega,\text{par}}$ and $\mathcal{CL}^2_{\text{univ,mult},\omega,\text{par}}$ we have actually set $b = m$, i.e we have plotted $\mathcal{L}_{\text{univ,mult},\omega,\text{par,max}}$ and $\mathcal{CL}^2_{\text{univ,mult},\omega,\text{par,max}}$. As expected, for all $m = 2,\, 4,\, 8,\, 16$, the size of the bit-serial/parallel architecture is greater than the size of the bit-serial architecture, while we have the opposite relation when considering their respective computation time. With respect to their area-time performance, the bit-serial/parallel architecture is preferable to the bit-serial architecture for $m = 2,\, 4$ with $b \leq m$ and for $m = 8,\, 16$ with $b \leq m/2 + 1$. The bit-serial architecture is preferable to the bit-serial/parallel architecture for $m = 8,\, 16$ with $b \geq m/2 + 2$. Note, however, that the difference in area-time performance of the two architectures is relatively insignificant.

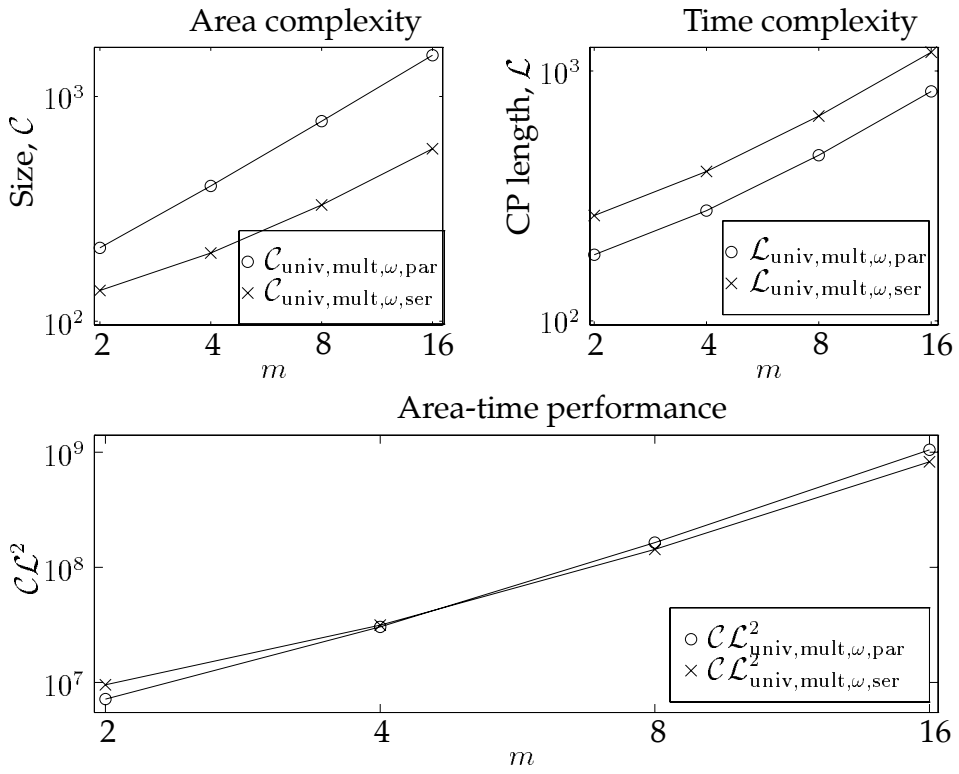


Figure 7.25: *The sizes $\mathcal{C}$, lengths $\mathcal{L}$, and AT^2 performances $\mathcal{C}\mathcal{L}^2$ of the universal bit-serial/parallel and bit-serial architectures in Figure* 7.23 *(for $b = m$) and Figure* 7.24*, respectively. The parameters are plotted versus m for $m = 2,\ 4,\ 8,\ 16$.*

Remark: For simplicity, we have assumed that the shift registers in Figures 7.23 and 7.24, with shift enable control signal S, have the same area and time complexities as the other registers considered in the thesis.

7.7 Summary

In Sections 5.2 and 6.4 we summarised the complexity and performance parameters of the architectures considered in the respective chapters. In Table 7.4, we have summarised the corresponding parameters for the architectures considered in the *present* chapter.

Operation	Figure	Subscript name	Size $\mathcal{C}$	Fan-in f	Int. CP length $\mathcal{L}_{\text{CP}}$	
Computing $a_i\|_{a_0}$	7.7	ai	$16m+2$	—	$28+2f_{\text{next}}$	
Computing d_k	7.8	dk	$50m+10$	—	$14m+54$	
Negation	7.12	polneg	4	$n_{\text{polneg}}+2$	—	
Addition	7.13	poladd,1	$\approx 3\cdot 2^m+3m\cdot 2^{m-3}+102m$	—	—	
Addition	7.17	poladd,2	$\approx 3\cdot 2^m+3m\cdot 2^{m-3}+66m$	—	—	⋯
General multipl.	7.18	polmult,par	$34m+6$	8	$14m+4$	
General multipl.	7.19	polmult,ser	$32m+88$	—	58	
Multiplication by ω^n	7.22	mult,ω,par	$6m+28b+2$	8	$14b+4$	
Univ. multipl. by ω^n	7.23	univ,mult,ω,par	$94m+24$	—	—	
Univ. multipl. by ω^n	7.24	univ,mult,ω,ser	$32m+73$	—	66	

	Norm. output res. r_o	Total CP length $\mathcal{L}$ (including registers)	Area-time perf. $\mathcal{CL}^2$
	—	$(i+1)\mathcal{L}_{\text{CP}}$	—
	—	—	—
	2	34	4624
	—	$\approx mL_{\text{exp,tab}}+(m-4+\log_2 m)L_{\text{log,tab}}+156m+104$	—
⋯	—	$\approx 2mL_{\text{exp,tab}}+(m-4+\log_2 m)L_{\text{log,tab}}+210m$	—
	1	$14m+44$	$\mathcal{O}(m^3)$
	—	$58m+58$	$\mathcal{O}(m^3)$
	1	$14b+44$	$\mathcal{O}(mb^2)$
	—	$32m+14b+92$	$\mathcal{O}(m^3)$
	—	$66m+132$	$\mathcal{O}(m^3)$

Table 7.4: *Complexity parameters of the architectures considered in the present chapter.*

Chapter 8

Comparisons Between Element Representations

The purpose of this chapter is to make brief comparisons between the element representations in Chapters 5, 6, and 7, i.e. the normal binary coded (NBC), the diminished–1, and the polar representation, respectively. We compare the respective VLSI architectures for arithmetic operations which are considered in these chapters.

8.1 Arithmetic Operations

Only the measure $\mathcal{C}$ of area complexity, the measure $\mathcal{L}$ of time complexity, and the measure $\mathcal{C}\mathcal{L}^2$ of combined area-time performance of each architecture are considered here. Regarding the parameter $\mathcal{L}$, we generally only consider the *total* CP length (which is proportional to the total computation time) and not the *internal* CP length (which for a bit-serial architecture is proportional to the clock cycle time). For detailed characterisation of the architectures, we refer to the mentioned Chapters 5, 6, and 7. In particular, see Tables 5.1, 6.5, and 7.4 in the respective Summary sections 5.2, 6.4, and 7.7.

8.1.1 Modulus Reduction

One of the main advantages of the polar representation is that modulus reduction is an instantaneous operation: The residue of the normal binary coded

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC			
• CR type	$22m - 6$	$16m + 38$	$\mathcal{O}(m^3)$
• CLA type	$4m \log_2 m + 14m$	$6m + 8 \log_2 m + 48$	$\mathcal{O}(m^3 \log_2 m)$
Diminished–1	As in the NBC case		
Polar	0	0	0

Table 8.1: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for modulus reduction, with respect to element representation. "CR" = carry ripple, "CLA" = carry look-ahead.*

integer $\hat{\gamma} \in \mathbb{Z}$ modulo 2^m equals $\hat{\gamma}^{(m-1)} = (\hat{\gamma}_{m-1}, \hat{\gamma}_{m-2}, \ldots, \hat{\gamma}_0)_2$. In both the diminished–1 and the polar representation, the integer 2^m is used as a representative of zero, which means that we can use an m-bit arithmetic for the nonzero integers of $\mathbb{Z}_{2^m+1}$.

In Section 5.1.1 (see for example Figure 5.3) we concluded that, with respect to the area-time performance AT^2 (and the time performance), the carry look-ahead type modulus reduction architecture is preferable to the carry ripple type architecture. From the $\mathcal{C}\mathcal{L}^2$ parameters in the rightmost column of Table 8.1 one may conclude that the carry ripple type architecture (with $\mathcal{C}\mathcal{L}^2 = \mathcal{O}(m^3)$) is preferable to the carry look-ahead type architecture (for which $\mathcal{C}\mathcal{L}^2 = \mathcal{O}(m^3 \log_2 m)$). However, the product $\mathcal{C}\mathcal{L}^2$ is smaller for the former architecture, compared to the latter one, only for very large m; $m > 2^{35}$.

Anyhow, as seen in Chapters 5, 6, and 7, in most circuits performing arithmetic operations, the modulus reduction part of the operation is preferably incorporated into each separate arithmetic operation.

8.1.2 Code Translation

The code translation from the NBC to the diminished–1 representation is simply carried out as a subtraction by one modulo $2^m + 1$. As seen in Table 8.2, the area-time product $\mathcal{C}\mathcal{L}^2$ is slightly less for the reverse translation (addition by one modulo $2^m + 1$). The code translation from the NBC to the polar representation and it reverse code translation involves the computation of the discrete logarithm and discrete exponentiation, respectively. In Sections 7.4 and

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC	—		
Diminished–1			
• NBC to dim.–1	$4m \log_2\ m + 8m - 6$	$4m + 8 \log_2\ m + 40$	$\mathcal{O}\left(m^3 \log_2\ m\right)$
• Dim.–1 to NBC	$18m + 2$	$10m + 28$	$\mathcal{O}\left(m^3\right)$
Polar			
• NBC to polar	One discrete logarithm		
• Polar to NBC	One discrete exponentiation		

Table 8.2: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for code translation, with respect to element representation.*

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC	$20m - 22$	$30m + 20$	$\mathcal{O}\left(m^3\right)$
Diminished–1	$4m$	$4m + 30$	$\mathcal{O}\left(m^3\right)$
Polar	4	34	4624

Table 8.3: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for negation, with respect to element representation.*

7.5, we showed how to compute the discrete logarithm and perform discrete exponentiation either without (Sec. 7.4) or with (Sec. 7.5) the use of look-up tables.

It is obvious that both the area complexities and the time performances of the code translations to and from the diminished–1 representation are less than the corresponding complexities of the code translations to and from the polar representation. Regarding the area and time complexities of the discrete logarithm and discrete exponentiation, we refer to Sections 7.6.1 and 7.6.2.

8.1.3 Negation

Table 8.3 shows some complexity parameters related to the architectures for negation using the NBC, diminished–1, and polar representations. The pa-

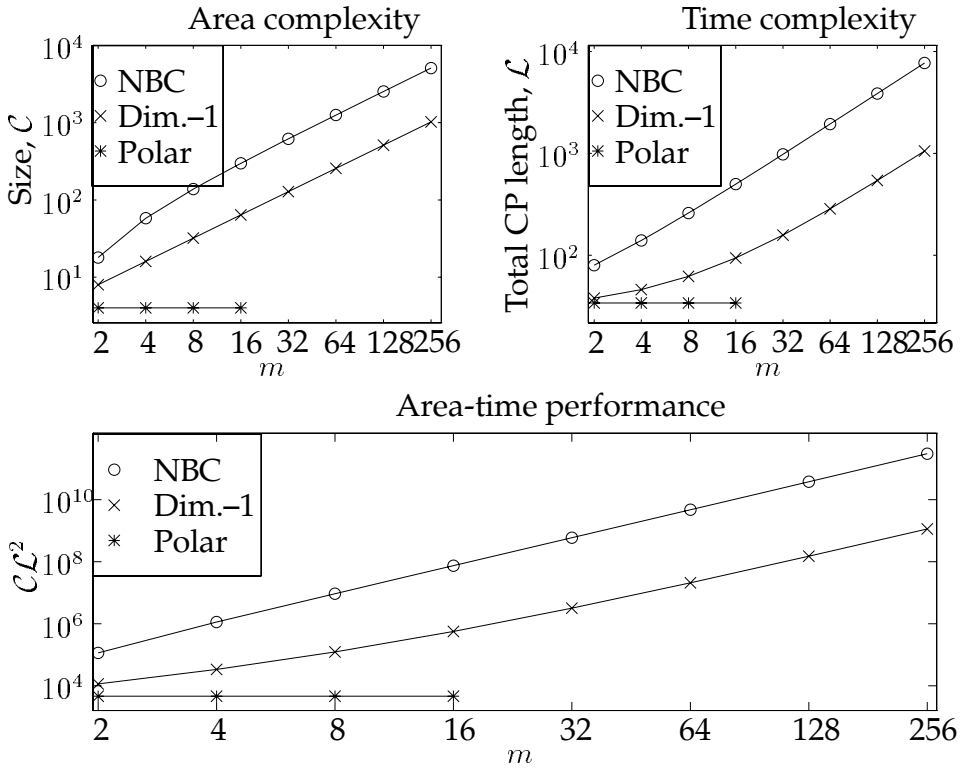


Figure 8.1: *Plots of the complexity parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ for negation when using the NBC, the diminished–1, or the polar representation. The parameters are obtained from Table 8.3.*

rameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ are plotted versus m in Figure 8.1. With respect to each of these parameters, it is clear that diminished–1 negation is generally less complex than NBC negation. In Fermat prime fields, i.e. for $m = 1, 2, 4, 8, 16$, negation in the polar representation is in turn less complex than negation in the diminished–1 representation.

8.1.4 Addition

As seen in Table 8.4, the complexity and the performance of performing addition in $\mathbb{Z}_{2^m+1}$ are approximately the same when using the NBC representation as when using the diminished–1 representation. For a comparison between

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC (carry r.)	$50m + 4$	$20m + 58$	$\mathcal{O}(m^3)$
Diminished–1			
• Carry ripple	$50m + 2$	$18m + 40$	$\mathcal{O}(m^3)$
• Carry l.-a.	$56m + 12$	$14m + 8\log_2 m + 70$	$\mathcal{O}(m^3)$
Polar			
• Figure 7.13	$\approx 3 \cdot 2^m + 3m2^{m-3} + 102m$	$\approx mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 156m + 104$	—
• Figure 7.17	$\approx 3 \cdot 2^m + 3m2^{m-3} + 66m$	$\approx 2mL_{\text{exp,tab}} + (m - 4 + \log_2 m)L_{\text{log,tab}} + 210m$	—

Table 8.4: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for addition, with respect to element representation. The sizes and CP lengths for polar addition is valid for $m = 8$ and $m = 16$.*

the carry ripple-type and the carry look-ahead-type diminished–1 adders, we refer to Section 6.3.4.

One of the main disadvantages of the polar representation derives from the fact that each polar addition involves the computation of one Zech's logarithm (see Figure 7.13), or essentially two discrete exponentiations and one discrete logarithm (see Figure 7.17). We have considered realisations of these operations which involve look-up tables. As seen in Table 8.4, polar addition is a much more complex operation than for example diminished–1 addition. Note, however, that in order to get a correct/fair comparison between the polar representation and the diminished–1 (or NBC) representation, polar addition should be compared with diminished–1 (or NBC) general multiplication and polar general multiplication should be compared with diminished–1 (or NBC) addition. This is further discussed in Section 8.1.7.

Remark: The complexity parameters for polar addition in Table 8.4 are approximate estimations. When using the delay model described in Section 4.2, we have not been able to determine the values of the constants $L_{\text{exp,tab}}$ and $L_{\text{log,tab}}$.

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC (s/p)	$4m \log_2 m + 121m + 57$	$24m^2 + 132m + 180$	$\mathcal{O}(m^5 \log_2 m)$
Diminished–1			
• Ashur's par	$34m^2 + 94m + 26$	$44m + 120$	$\mathcal{O}(m^4)$
• Shyu's s/p	$103m + 134$	$18m^2 + 124m + 208$	$\mathcal{O}(m^5)$
Polar			
• Bit-parallel	$34m + 6$	$14m + 44$	$\mathcal{O}(m^3)$
• Bit-serial	$32m + 88$	$58m + 58$	$\mathcal{O}(m^3)$

Table 8.5: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for general multiplication, with respect to element representation. "s/p" = bit-serial/parallel. "par" = bit-parallel.*

8.1.5 General Multiplication

The sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time products $\mathcal{C}\mathcal{L}^2$ of the architectures for general multiplication considered in Chapters 5, 6, and 7 are listed in Table 8.5. In Chapter 6 we considered six different diminished–1 general multipliers, of which three are bit-serial and the other three are bit-serial/parallel multipliers. The sizes and total CP lengths of these multipliers were summarised in Table 6.4 in the end of Section 6.3.6. Also, the complexity parameters of the best bit-parallel multiplier (Ashur's) and the best bit-serial/parallel multiplier (Shyu's) were plotted versus m in Figure 6.25. Among the diminished–1 multipliers, only these two are considered in Table 8.5.

In Figure 8.2, we have plotted the parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ of the NBC bit-serial/parallel multiplier, Ashur's diminished–1 bit-parallel multiplier, and our polar bit-parallel multiplier. For $m \geq 4$, the complexity parameters of Shyu's multiplier are all slightly less than the corresponding complexity parameters of the NBC multiplier. Therefore, Shyu's multiplier is not considered in Figure 8.2. We see that the AT^2 performance of Ashur's multiplier is less than the AT^2 performance of the NBC multiplier. In Fermat prime fields, the polar multiplier is in turn superior to the other multipliers.

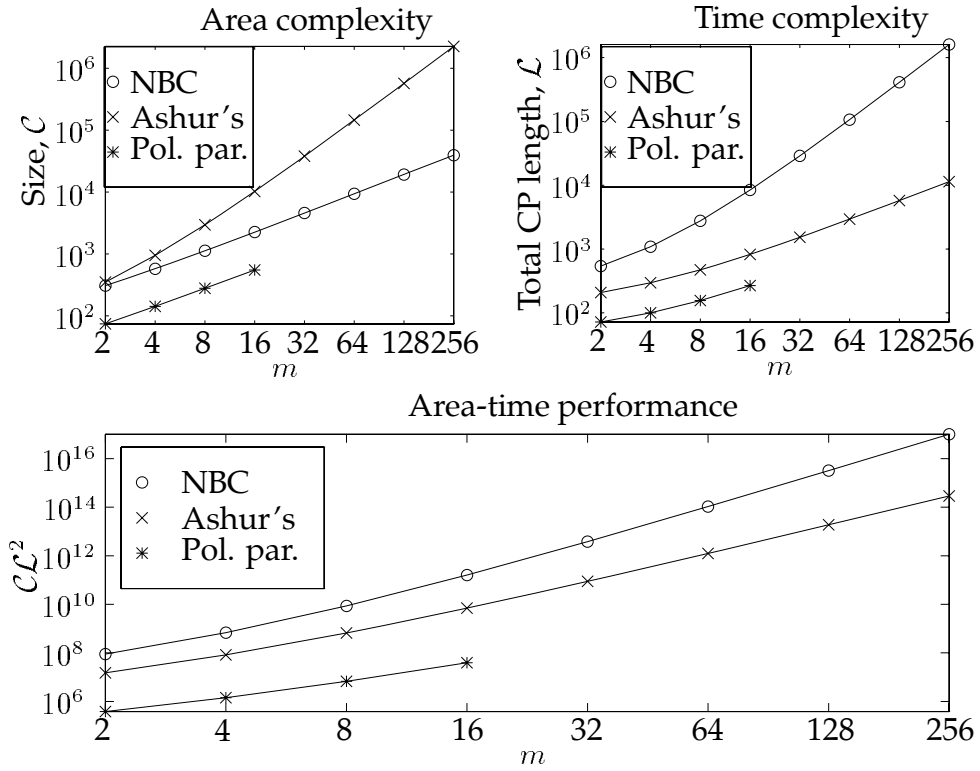


Figure 8.2: *Plots of the complexity parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ for general multiplication with respect to the NBC, the diminished–1, or the polar representation. The diminished–1 multiplier is Ashur's bit-parallel multiplier and the polar multiplier is the bit-parallel one. The parameters are obtained from Table* 8.5.

8.1.6 Multiplication by Powers of ω

Multiplication by 2^n

Multiplications by powers of two typically occur when computing Fermat number transforms of lengths $N = 2m$ and $N = 4m$ using the NBC or the diminished–1 representation. Then, the transform kernels most often used are $\omega = 2$ (for $N = 2m$) and $\omega = \sqrt{2}$ (for $N = 4m$), see Section 2.3.2. In Table 8.6 we have listed some complexity parameters of architectures for multiplication by 2^n; $n \in \mathbb{Z}$, with respect to the NBC, the diminished–1, and the polar representation. The parameters of the architecture for polar multiplication (the bottom

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC	$4m \log_2 m + 33m + 15$	$\leq 20m^2 + 16m \log_2 m + 148m$	$\mathcal{O}(m^5 \log_2 m)$
Diminished–1	$28m - 4$	$\leq 40m + 40$	$\mathcal{O}(m^3)$
Polar	$6m + 28 \log_2 m + 30$	$14 \log_2 m + 58$	$\mathcal{O}(m \log_2^2 m)$

Table 8.6: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for multiplication by 2^n, with respect to element representation.*

row in the table) are obtained by letting[1] $b = \log_2 2m = \log_2 m + 1$ in the corresponding parameters of the fixed architecture for polar multiplication by ω^n in Table 8.7.

The parameters in Table 8.6 are plotted versus m in Figure 8.3. The complexity and performance of the architecture for the NBC representation are relatively high for all m. The architecture for the diminished–1 representation is generally superior to the other architectures. However, for $m = 2, 4, 8, 16$, the architecture for the polar representation has the smallest time complexity and the smallest area-time performance. Hence, whenever applicable, the architecture for polar multiplication by powers of two is preferable to the other architectures performing the same operation.

Multiplication by Powers of $\omega = \alpha^{2^{m-b}}$

When using the diminished–1 representation (or the NBC representation), the Fermat number transform is generally known to be applicable only for some small transform lengths, because then the transform multiplications by powers of the transform kernel can be carried out using only binary shifts (rotations) (we mentioned above the kernels $\omega = 2$ and $\omega = \sqrt{2}$, for which we get the transform lengths $N = 2m$ and $N = 4m$). The restriction to relatively small transform lengths, however, is still adequate in Fermat integer quotient rings where the modulus $2^m + 1$ is *composite*, because in such rings the maximum possible transform length *is* relatively small, in comparison with the modulus. In the Fermat prime fields $\mathbb{Z}_{2^8+1}$ and $\mathbb{Z}_{2^{16}+1}$, however, i.e. where the modulus $2^m + 1$ is *prime*, there exist transforms of much greater lengths than $2m$ and $4m$:

[1] The equality follows from the fact that the order of $\omega = 2$ modulo $2^m + 1$ equals $N = 2^b = 2m$.

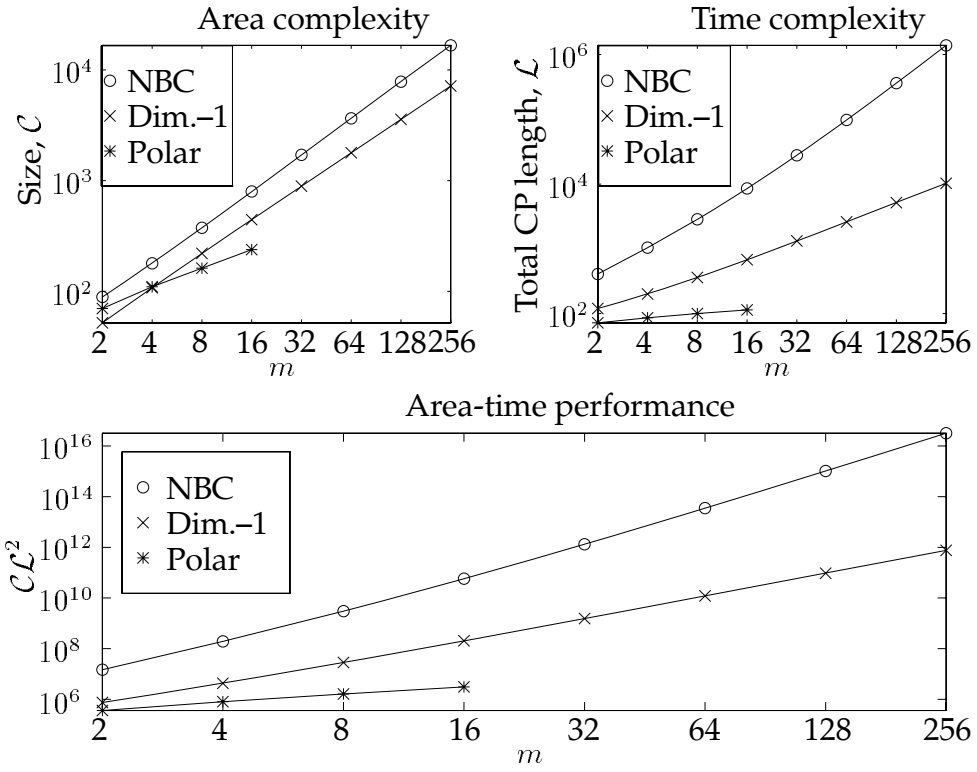


Figure 8.3: *Plots of the complexity parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ for multiplication by 2^n when using the NBC, the diminished–1, or the polar representation. The parameters are obtained from Table 8.6.*

We know that for $m = 1, 2, 4, 8, 16$, there exist Fermat number transforms of length $N = 2^b$ in $\mathbb{Z}_{2^m+1}$, where $0 \leq b \leq m$.

Using the NBC representation or the diminished–1 representation, when computing a transform of *arbitrary* length $N = 2^b$, each nontrivial multiplication by a power of the transform kernel ω of order N modulo $2^m + 1$ must generally be carried out as a general multiplication. The powers of ω which appear in the computation of each transform may be precomputed and stored in a memory. If not, they can be obtained using general exponentiations.

In Chapter 7 we showed how to compute multiplications by arbitrary powers of the transform kernel $\omega \equiv \alpha^{2^{m-b}} \pmod{2^m+1}$ of arbitrary order 2^b; $0 \leq b \leq m$ modulo $2^m + 1$, using one simplified addition in the polar representation. The complexity parameters of the architectures for polar multiplication by a

Form of repr.	Size $\mathcal{C}$	Total CP length $\mathcal{L}$	$\mathcal{C}\mathcal{L}^2$
NBC	General (exponentiation and) multiplication needed		
Diminished–1	General (exponentiation and) multiplication needed		
Polar			
• Fixed	$38m + 44b + 34$	$28b + 88$	$\mathcal{O}(mb^2)$
• Universal			
– Serial/parallel	$94m + 24$	$32m + 14b + 92$	$\mathcal{O}(m^3)$
– Serial	$32m + 73$	$66m + 132$	$\mathcal{O}(m^3)$

Table 8.7: *Sizes $\mathcal{C}$, total CP lengths $\mathcal{L}$, and area-time performances $\mathcal{C}\mathcal{L}^2$ of the architectures for multiplication by $\omega^n \equiv \alpha^{n \cdot 2^{m-b}} \pmod{2^m + 1}$, with respect to element representation.*

power of $\alpha^{2^{m-b}} \bmod 2^m + 1$, is listed in Table 8.7. These parameters are also plotted versus m in Figure 8.4. Some of the parameters are plotted twice in the figure. For each such pair of curves, the upper curve is an upper bound (for $b = m$) and the lower curve is a lower bound (for $b = 1$) on the parameter in question.

For all $b \in [0, m]$, the fixed architecture is superior to the two universal architectures. However, the universal bit-serial architecture has the smallest size among the architecture. Note that the complexity and performance of these three architectures are less than the complexity and performance of the architectures for NBC and diminished–1 general multiplication.

8.1.7 Butterfly Computations

In Section 2.3.3, we considered some algorithms for computing the Fermat number transform. In each of these algorithms, the transform computation is subdivided into a number of *butterfly* computations. For example, in the well known radix-2 decimation-in-time and decimation-in-frequency algorithms, which are described in Section 2.3.3, a Fermat number transform of length $N = 2^b$ is obtained by computing $(N/2) \log_2 N$ basic butterflies. Each butterfly, which performs a transform of length two, involves one negation, two additions, and one multiplication by some power of the transform kernel ω. We refer to Figures 2.1 and 2.2.

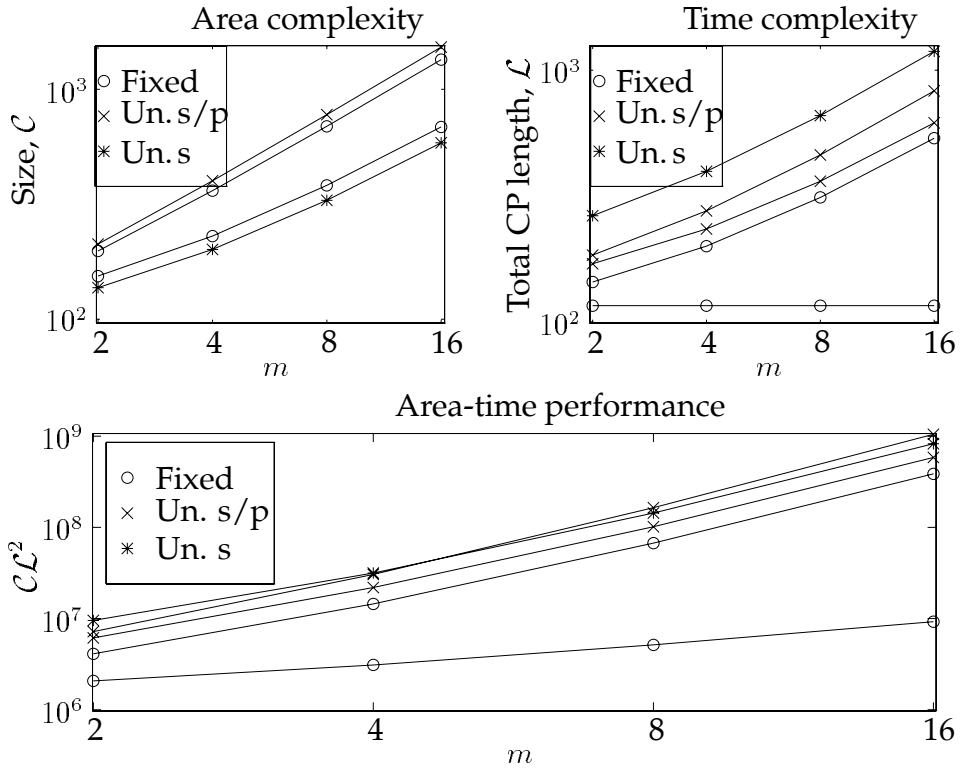


Figure 8.4: *Plots of the complexity parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{CL}^2$ for multiplication by $\omega^n \equiv \alpha^{n \cdot 2^{m-b}} \pmod{2^m + 1}$ when using the polar representation. The parameters are obtained from Table 8.7. "Fixed" = the fixed bit-parallel architecture. "Un. s/p" = the universal bit-serial/parallel architecture. "Un. s" = the universal bit-serial architecture.*

Next, we consider gross estimations of the total size and the total critical path length of such a butterfly, with respect to the normal binary coded representation, the diminished–1 representation, and the polar representation. When using the normal binary coded and the diminished–1 representations, we assume that we have two adders in parallel. We use the following complexity parameters:

• The Normal Binary Coded Representation:

$$\text{Negation:} \quad \begin{cases} \mathcal{C}_{\text{neg}} = 20m - 22 \\ \mathcal{L}_{\text{neg}} = 30m + 20 \end{cases}$$

$$\text{Addition:} \quad \begin{cases} \mathcal{C}_{\text{add}} = 50m + 4 \\ \mathcal{L}_{\text{add}} = 20m + 58 \end{cases}$$

$$\text{General multiplication:} \quad \begin{cases} \mathcal{C}_{\text{mult}} = 4m \log_2 m + 121m + 57 \\ \mathcal{L}_{\text{mult}} = 24m^2 + 132m + 180 \end{cases}$$

$$\textbf{Total complexity:} \quad \begin{cases} \mathcal{C}_{\text{butt,NBC}} &= \mathcal{C}_{\text{neg}} + 2\mathcal{C}_{\text{add}} + \mathcal{C}_{\text{mult}} \\ &= 4m \log_2 m + 241m + 43 \\ \mathcal{L}_{\text{butt,NBC}} &= \mathcal{L}_{\text{neg}} + \mathcal{L}_{\text{add}} + \mathcal{L}_{\text{mult}} \\ &= 24m^2 + 182m + 258 \end{cases}$$

• The Diminished–1 Representation:

$$\text{Negation:} \quad \begin{cases} \mathcal{C}_{\text{dimneg}} = 4m \\ \mathcal{L}_{\text{dimneg}} = 4m + 30 \end{cases}$$

$$\text{Addition:} \quad \begin{cases} \mathcal{C}_{\text{dimadd,2}} = 50m + 2 \\ \mathcal{L}_{\text{dimadd,2}} = 18m + 40 \end{cases}$$

$$\text{General multiplication:} \quad \begin{cases} \mathcal{C}_{\text{Ashur,mult}} = 34m^2 + 94m + 26 \\ \mathcal{L}_{\text{Ashur,mult}} = 44m + 120 \end{cases}$$

$$\textbf{Total complexity:} \quad \begin{cases} \mathcal{C}_{\text{butt,dim}} &= \mathcal{C}_{\text{dimneg}} + 2\mathcal{C}_{\text{dimadd,2}} + \mathcal{C}_{\text{Ashur,mult}} \\ &= 34m^2 + 198m + 30 \\ \mathcal{L}_{\text{butt,dim}} &= \mathcal{L}_{\text{dimneg}} + \mathcal{L}_{\text{dimadd,2}} + \mathcal{L}_{\text{Ashur,mult}} \\ &= 66m + 190 \end{cases}$$

When using the polar representation, the two butterfly additions can not be computed exactly in parallel, because then we would get a memory access conflict. Therefore, the total critical path runs through the negater and the two adders of the decimation-in-frequency butterfly in Figures 2.2. Multiplication by a power of the transform kernel is carried out during the computation of the second addition. If the decimation-in-time butterfly is used, the path through the multiplier must also be added to the total critical path length.

Hence, for the polar representation (and when using decimation-in-frequency butterflies), we use the following complexity parameters:

• The Polar Representation:

$$\text{Negation:} \quad \begin{cases} \mathcal{C}_{\text{polneg}} = 4 \\ \mathcal{L}_{\text{polneg}} = 34 \end{cases}$$

$$\text{Addition:} \quad \begin{cases} \mathcal{C}_{\text{poladd,1}} \approx 3 \cdot 2^m + 3m \cdot 2^{m-3} + 102m \\ \mathcal{L}_{\text{poladd,1}} \approx m L_{\text{exp,tab}} + (m - 4 + \log_2\, m) L_{\text{log,tab}} \\ \qquad\qquad + 156m + 104 \end{cases}$$

$$\text{Multiplication by } \omega^n\text{:} \quad \begin{cases} \mathcal{C}_{\text{mult},\omega,\text{par}} = 6m + 28b + 2 \\ \mathcal{L}_{\text{mult},\omega,\text{par}} = 14b + 44 \end{cases}$$

$$\textbf{Total complexity:} \quad \begin{cases} \mathcal{C}_{\text{butt,polar}} &= \mathcal{C}_{\text{polneg}} + \mathcal{C}_{\text{poladd,1}} + \mathcal{C}_{\text{mult},\omega,\text{par}} \\ &= 3 \cdot 2^m + 3m \cdot 2^{m-3} + 108m + 28b + 6 \\ \mathcal{L}_{\text{butt,polar}} &< \mathcal{L}_{\text{polneg}} + 2\mathcal{C}_{\text{poladd,1}} \\ &\approx 2m L_{\text{exp,tab}} + 2(m - 4 + \log_2\, m) L_{\text{log,tab}} \\ &\quad + 312m + 242 \end{cases}$$

The butterfly complexity parameters $\mathcal{C}_{\text{butt,NBC}}$, $\mathcal{L}_{\text{butt,NBC}}$, $\mathcal{C}_{\text{butt,dim}}$, $\mathcal{L}_{\text{butt,dim}}$, $\mathcal{C}_{\text{butt,polar}}$, and $\mathcal{L}_{\text{butt,polar}}$ are plotted versus m in Figure 8.5. For $\mathcal{C}_{\text{butt,polar}}$ and $\mathcal{L}_{\text{butt,polar}}$ we have set maximum $b = m$ and $L_{\text{exp,tab}} = L_{\text{log,tab}} = 1$, respectively. These complexity parameters, however, do not change significantly for other (reasonable) values of b, $L_{\text{exp,tab}}$, and $L_{\text{log,tab}}$.

From Figure 8.5 we conclude that, for all m, the diminished–1 representation is superior to the normal binary coded representation. Regarding the polar representation, the complexity parameters $\mathcal{C}_{\text{butt,polar}}$ and $\mathcal{L}_{\text{butt,polar}}$ should be taken with a pinch of salt. The reason for this is the inaccuracies of the modelled areas and access times of the memories used to perform discrete exponentiation and compute discrete logarithms. In Sections 7.6.1 and 7.6.2, we only derived approximate estimations of the parameters $\mathcal{C}_{\text{exp,tab}}$, $\mathcal{L}_{\text{exp,tab}}$, $\mathcal{C}_{\text{log,tab}}$, and $\mathcal{L}_{\text{log,tab}}$. We can obtain more accurate estimations of these parameters by considering *all* parts of the memory architecture in Figure 7.9. Such a complex modelling, however, is not considered in this thesis.

As mentioned earlier in the thesis, the main disadvantage of the polar representation is the relatively large chip area required when implementing polar addition on the form which uses look-up tables. Still, some other nice properties of the polar representation may make up for this disadvantage. For

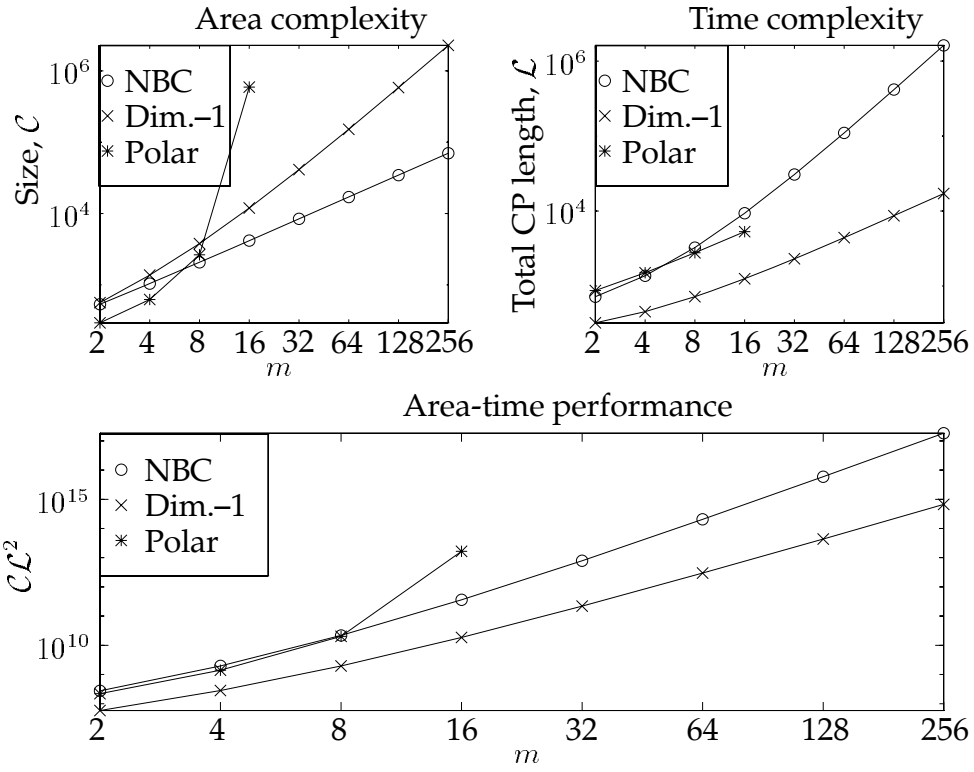


Figure 8.5: *Plots of the complexity parameters $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{C}\mathcal{L}^2$ for the complete decimation-in-frequency butterfly when using the NBC, the diminished–1, or the polar representation.*

example, we have proposed universal architectures for multiplication by powers of the transform kernel with favourable sizes and critical path lengths, see Sections 7.6.6 and 8.1.6. Any of these universal architectures can be used in the computation of a Fermat number transform of *arbitrary* allowed length in a Fermat prime field.

8.2 Other element representations

We have focused on the normal binary coded, the diminished–1, and the polar representation. A few alternative ways of representing the (binary coded) integers of Fermat integer quotient rings $\mathbb{Z}_{2^m+1}$ have been suggested in the lit-

erature. For example, Agrawal and Rao [3][2] describes an $(m + 1)$-bit binary coded representation which uses one of the bits as a *zero indicator*. However, none of these forms of representation have been considered in this thesis.

[2] See also references [6] and [7] in their paper.

Chapter 9

Conclusions

The arithmetic operations considered in this thesis are essentially modulus reduction, code translation, negation, addition, subtraction, general multiplication, and multiplication by powers of the Fermat number transform kernel. All operations are carried out in Fermat integer quotient rings. The properties of these operations were thoroughly investigated with respect to the normal binary coded representation, the diminished–1 representation, and the polar representation of the binary coded integers of Fermat integer quotient rings. The polar representation is applicable only when the Fermat number modulus is prime.

Based on a linear switch-level RC model for CMOS transistors we derived area and time complexities and combined area$\times$time^2 performances of the various architectures for the above arithmetic operations. The architectures were mutually compared with respect to these measures of complexity and performance. To the authors knowledge, such a comparison has not been carried out before.

Regarding the normal binary coded representation, we found that the area$\times$ time^2 performance of some of the architectures considered was relatively poor. This derives mainly from the relatively complex circuitry for performing the modulus reduction of the corresponding arithmetic operations. In some architectures, the modulus reduction part of the circuit represented a rather large part of the complete architecture.

With respect to the area and time complexities and the area$\times$time^2 performance, we established the superiority of the diminished–1 representation over

the normal binary coded representation. We also came to the general conclusion that, mainly from a computational complexity point of view, the diminished–1 representation *is* in fact the one most efficient in the class of element representations that can be expressed as a linear elementary function of the normal binary coded representation.

Using properties of Zech's logarithms, we derived an algorithm for efficiently computing the discrete logarithm in Fermat prime fields, principally using only a number of recursive diminished–1 additions. We also derived an algorithm for performing discrete exponentiation using only a number of recursive diminished–1 additions and some binary shifts. Based on these algorithms, we then derived computational procedures for computing the discrete logarithm and performing discrete exponentiation using look-up tables of appropriate sizes (one table for each operation). Each resulting algorithm principally only involves a number of binary shifts and a table look-up. Hence, the complexity of computing the discrete logarithm and performing discrete exponentiation was significantly reduced, to the cost of two look-up tables.

One of the main advantages of the polar representation concerns the complexity of performing multiplication by powers of the transform kernel. We proved that, for *every* possible transform length $N = 2^b;\ 0 \leq b \leq m$, the polar representation provides a suitable choice of the transform kernel for which multiplication by powers of the transform kernel can be carried out using only one addition modulo 2^b. We also designed universal architectures (one bit-serial/parallel and one bit-serial) for performing such multiplications. Thus, any of these universal architectures can be used in the computation of a Fermat number transform of *arbitrary* allowed length in a Fermat prime field.

Appendix A

Proofs of Some Theorems

In this Appendix we present proofs of some theorems of the thesis. The proofs themselves may not be of central importance for the results of the thesis, but they are included mainly because they have great number theoretic significance in the context of the thesis.

A.1 Proof of Theorem 2.1

The outline of the proof is essentially the same as the outline of the proof by Agarwal and Burrus in [2, Th. 1]. The theorem is equivalent to

Theorem A.1 *There exists an invertible NTT of length N in $\mathbb{Z}_q$ if and only if $N \mid (p_i - 1)$ for every prime p_i that divides q.*

Proof: According to Euler's theorem (*"if q is a positive integer and ω is relatively prime to q, then $\omega^{\phi(q)} \equiv 1 \pmod{q}$"*), the order N of the transform kernel ω modulo q must divide $\phi(q)$ where ϕ denotes Euler's *totient function* (see for example Rosen, [84, Ch. 5.3]). It can be shown that for such an integer q with prime-power factorisation $q = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$, the totient function is

$$\phi(q) = p_1^{n_1-1}(p_1 - 1)p_2^{n_2-1}(p_2 - 1) \cdots p_k^{n_k-1}(p_k - 1).$$

Hence, we get

$$N \mid p_1^{n_1-1}(p_1 - 1)p_2^{n_2-1}(p_2 - 1) \cdots p_k^{n_k-1}(p_k - 1).$$

However, by the congruence $\omega^N \equiv 1 \pmod{q}$ we get $q \mid (\omega^N - 1)$, and hence $p_i^{n_i} \mid (\omega^N - 1)$, i.e.

$$\omega^N \equiv 1 \pmod{p_i^{n_i}}$$

for every factor $p_i^{n_i}$ of q. Then, by Euler's theorem, we get

$$N \mid \phi(p_i^{n_i}) = p_i^{n_i - 1}(p_i - 1). \tag{A.1}$$

In order for the inverse transform to exist, N^{-1} must exist in the ring. The congruence $N \cdot N^{-1} \equiv 1 \pmod{q}$ implies that N and q must be relatively prime, which means that no prime factor p_i of q can be a factor of N. Therefore (A.1) reduces to

$$N \mid (p_i - 1),$$

for $i = 1, 2, \ldots, k$, which can also be written as

$$N \mid \gcd(p_1 - 1, p_2 - 1, \ldots, p_k - 1).$$

Conversely, if $N \mid (p_i - 1)$ we know, by Theorem 8.8 of [84], that there are $\phi(N)$ incongruent integers with order N modulo p_i. For $p_i = 2$ we get the solution $N = 1$ and the theorem becomes trivial. For odd primes p_i, let α_i be an integer with $\gcd(\alpha_i, p_i) = 1$ such that $\operatorname{ord}_{p_i} \alpha_i = p_i - 1$. Then, each nonzero integer of $\mathbb{Z}_{p_i}$ is congruent to some power of α_i modulo p_i [84, Th. 8.3]. For such an integer $\beta_i = \alpha_i^{r_i}$ with $\operatorname{ord}_{p_i} \beta_i = N$ and some positive integer r_i, it follows from [84, Th. 8.4] that

$$N = \frac{p_i - 1}{\gcd(p_i - 1, r_i)}.$$

By Theorems 8.9 and 8.10 of [84] we know that if $\operatorname{ord}_{p_i} \alpha_i = \phi(p_i)$, the order of α_i modulo $p_i^{n_i}$ is $\phi(p_i^{n_i}) = (p_i - 1)p_i^{n_i - 1}$ for all positive integers n_i.

From the above reasoning we get

$$\alpha_i^{(p_i-1)p_i^{n_i-1}} = \alpha_i^{\frac{N}{N}(p_i-1)p_i^{n_i-1}} = \left(\alpha_i^{\gcd(p_i-1,r_i)\cdot p_i^{n_i-1}}\right)^N \equiv 1 \pmod{p_i^{n_i}},$$

and consequently we can choose

$$\omega_i = \alpha_i^{\gcd(p_i-1,r_i)\cdot p_i^{n_i-1}}$$

as an integer with order N modulo $p_i^{n_i}$. By the Chinese reminder theorem [84, Th. 3.12] we can find a unique solution ω modulo $q = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ such that

$$\omega \equiv \omega_i \pmod{p_i^{n_i}}$$

for distinct primes p_i and $i = 1, 2, \ldots, k$. Also, the order of ω modulo q is N. Because $\gcd(N, p_i) = 1$ we have $\gcd(N, q) = 1$ and thus there exists an inverse

of N modulo q. Hence, there exists an invertible NTT of length N in $\mathbb{Z}_q$ for which $N \mid (p_i - 1)$ for every prime factor p_i of q.

□

A.2 Proof of Theorem 2.3

In most number theory books the author leaves the proof of Theorem 2.3 as an exercise for the reader. In this section we present our solution to this exercise.

The proof involves the concept of *quadratic residues*.

Definition A.1 *An integer a which is relatively prime to a positive integer q is said to be a* quadratic residue *modulo q if there is an integer x such that the congruence $x^2 \equiv a \pmod{q}$ has a solution. If the congruence has no solution, we say that a is a* quadratic nonresidue *modulo q.*

The Legendre symbol $\left(\frac{a}{p}\right)$ is frequently used to indicate whether an integer a, not divisible by the odd prime p, is a quadratic residue modulo p:

$$\left(\frac{a}{p}\right) \triangleq \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic nonresidue modulo } p \end{cases}. \tag{A.2}$$

Euler's criterion is useful when *deciding* whether an integer is a quadratic residue modulo a prime:

Lemma A.1 *If p is an odd prime and a is a positive integer not divisible by p, then*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}. \tag{A.3}$$

Proof: See the proof of Theorem 9.2 of Rosen in [84]. □

Now, we are ready for the proof of Theorem 2.3, which is equivalent to

Theorem A.2 *Every prime divisor of the Fermat number $F_t = 2^{2^t} + 1$, where $t \geq 2$, is on the form $k \cdot 2^{t+2} + 1$, for some natural number k.*

Proof: For every Fermat number F_t we have $2^{2\cdot 2^t} \equiv (-1)^2 = 1 \pmod{F_t}$, which implies $F_t \mid (2^{2^{t+1}} - 1)$. Therefore, for every prime divisor p of F_t, we get $p \mid (2^{2^{t+1}} - 1)$ or equivalently

$$2^{2^{t+1}} \equiv 1 \pmod{p}. \tag{A.4}$$

Also, by Euler's theorem we have $2^{p-1} \equiv 1 \pmod{p}$ and therefore $2^{t+1} \mid (p-1)$, which means that p is of the form $p = k' \cdot 2^{t+1} + 1$ for some positive integer k'. For $t \geq 2$ we see that $p = k' \cdot 2^{t-2} \cdot 2^3 + 1$ is congruent to 1 modulo 8.

By Proposition A.17(ii) of Stewart [95], 2 is a quadratic residue modulo p, i.e. $\left(\frac{2}{p}\right) = 1$, and thus, from Euler's criterion (Equation (A.3)) we get

$$2^{\frac{p-1}{2}} \equiv 1 \pmod{p}. \tag{A.5}$$

Hence, from (A.4) and (A.5), we see that

$$2^{t+1} \mid \frac{p-1}{2},$$

which implies that p is on the form $p = 2 \cdot 2^{t+1} + 1 = 2^{t+2} + 1$.

□

A.3 Proof of Theorem 2.5

The Legendre symbol, which was defined in (A.2), can be used to check whether an integer is primitive or not. It follows from Euler's criterion (Equation (A.3)), together with the definition of primitive elements, that a primitive element in $\mathbb{Z}_{F_t}$ is a quadratic nonresidue modulo F_t.

The *quadratic reciprocity law*, which was discovered by Euler and proved by Gauss, can be of great help to calculate the Legendre symbol:

Lemma A.2 *If p and q are odd primes, then*

$$\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)(-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Proof: See for example Lang [57, pp. 76–78] or Rosen [84, Ch. 9.2]. □

We are now able to prove Theorem 2.5, which is equivalent to

Theorem A.3 *The integer 3 is a primitive element of each Fermat prime field* $\mathbb{Z}_{F_t}$ *where* $t \geq 1$.

Proof: (See for example the proof of Theorem 9.7 (Pepin's test) in the book by Rosen, [84]). Consider the primes among the Fermat numbers $F_t = 2^m + 1$; $m = 2^t$ for $t \geq 1$. The quadratic reciprocity law yields

$$\left(\frac{3}{2^m+1}\right) = \left(\frac{2^m+1}{3}\right)(-1)^{\frac{2^m+1-1}{2}\cdot\frac{3-1}{2}} = \left(\frac{2^m+1}{3}\right).$$

By Euler's criterion (A.3) we can write $\left(\frac{2^m+1}{3}\right)$ as

$$\left(\frac{2^m+1}{3}\right) \equiv (2^m+1)^{\frac{3-1}{2}} = 2^m+1 \equiv (-1)^m + 1 = 2 \equiv -1 \pmod 3,$$

and hence we have

$$\left(\frac{3}{2^m+1}\right) = -1,$$

or equivalently

$$3^{2^{m-1}} \equiv -1 \pmod{2^m+1}. \tag{A.6}$$

By Euler's theorem we know that the order of 3 modulo the prime F_t divides $F_t - 1 = 2^m$, i.e. $\mathrm{ord}_{F_t} 3 \mid 2^m$, which means that $\mathrm{ord}_{F_t} 3$ is a power of two. Furthermore, since (A.6) implies that $\mathrm{ord}_{F_t} 3 \nmid 2^{m-1}$, we consequently get $\mathrm{ord}_{F_t} 3 = 2^m$. Thus, the integer 3 is a primitive element of $\mathbb{Z}_{2^m+1}$ for $2^m + 1 \geq 5$ (whenever $2^m + 1$ is prime.

□

Appendix B

A Table of Some Primes

n	m	q	$q-1$
2	1	3	2
3	1	7	$2 \cdot 3$
3	2	5	2^2
4	2	13	$2^2 \cdot 3$
5	1	31	$2 \cdot 3 \cdot 5$
5	2	29	$2^2 \cdot 7$
5	4	17	2^4
6	2	61	$2^2 \cdot 3 \cdot 5$
7	1	127	$2 \cdot 3^2 \cdot 7$
7	4	113	$2^4 \cdot 7$
7	5	97	$2^5 \cdot 3$
8	4	241	$2^4 \cdot 3 \cdot 5$
8	6	193	$2^6 \cdot 3$
9	2	509	$2^2 \cdot 127$
9	6	449	$2^6 \cdot 7$
9	8	257	2^8
10	2	1021	$2^2 \cdot 3 \cdot 5 \cdot 17$
10	4	1009	$2^4 \cdot 3^2 \cdot 7$
10	8	769	$2^8 \cdot 3$

Table B.1: *Prime numbers of the form* $q = 2^n - 2^m + 1$ *for* $0 < m < n \leq 32$. *The table continues on the next page.*

n	m	q	$q-1$
11	5	2017	$2^5 \cdot 3^2 \cdot 7$
12	2	4093	$2^2 \cdot 3 \cdot 11 \cdot 31$
13	1	8191	$2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
13	5	8161	$2^5 \cdot 3 \cdot 5 \cdot 17$
13	8	7937	$2^8 \cdot 31$
13	9	7681	$2^9 \cdot 3 \cdot 5$
14	2	16381	$2^2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
14	4	16369	$2^4 \cdot 3 \cdot 11 \cdot 31$
14	10	15361	$2^{10} \cdot 3 \cdot 5$
14	12	12289	$2^{12} \cdot 3$
15	9	32257	$2^9 \cdot 3^2 \cdot 7$
16	4	65521	$2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
16	10	64513	$2^{10} \cdot 3^2 \cdot 7$
16	12	61441	$2^{12} \cdot 3 \cdot 5$
17	1	131071	$2 \cdot 3 \cdot 5 \cdot 17 \cdot 257$
17	5	131041	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
17	6	131009	$2^6 \cdot 23 \cdot 89$
17	8	130817	$2^8 \cdot 7 \cdot 73$
17	14	114689	$2^{14} \cdot 7$
17	16	65537	2^{16}
19	1	524287	$2 \cdot 3^3 \cdot 7 \cdot 19 \cdot 73$
19	5	524257	$2^5 \cdot 3 \cdot 43 \cdot 127$
19	9	523777	$2^9 \cdot 3 \cdot 11 \cdot 31$
19	12	520193	$2^{12} \cdot 127$
20	2	1048573	$2^2 \cdot 3^3 \cdot 7 \cdot 19 \cdot 73$
20	14	1032193	$2^{14} \cdot 3^2 \cdot 7$
20	18	786433	$2^{18} \cdot 3$
22	2	4194301	$2^2 \cdot 3 \cdot 5^2 \cdot 11 \cdot 31 \cdot 41$
23	4	8388593	$2^4 \cdot 524287$
23	13	8380417	$2^{13} \cdot 3 \cdot 11 \cdot 31$
23	17	8257537	$2^{17} \cdot 3^2 \cdot 7$
23	20	7340033	$2^{20} \cdot 7$
24	2	16777213	$2^2 \cdot 3 \cdot 23 \cdot 89 \cdot 683$
24	6	16777153	$2^6 \cdot 3^3 \cdot 7 \cdot 19 \cdot 73$
24	8	16776961	$2^8 \cdot 3 \cdot 5 \cdot 17 \cdot 257$
24	14	16760833	$2^{14} \cdot 3 \cdot 11 \cdot 31$
24	18	16515073	$2^{18} \cdot 3^2 \cdot 7$

Table B.1: *cont': Prime numbers of the form* $q = 2^n - 2^m + 1$ *for* $0 < m < n \leq 32$.
The table continues on the next page.

n	m	q	$q-1$
25	12	33550337	$2^{12} \cdot 8191$
25	14	33538049	$2^{14} \cdot 23 \cdot 89$
25	18	33292289	$2^{18} \cdot 127$
26	12	67104769	$2^{12} \cdot 3 \cdot 43 \cdot 127$
26	16	67043329	$2^{16} \cdot 3 \cdot 11 \cdot 31$
27	11	134215681	$2^{11} \cdot 3 \cdot 5 \cdot 17 \cdot 257$
27	21	132120577	$2^{21} \cdot 3^2 \cdot 7$
28	16	268369921	$2^{16} \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
29	2	536870909	$2^2 \cdot 7 \cdot 73 \cdot 262657$
29	6	536870849	$2^6 \cdot 47 \cdot 178481$
29	8	536870657	$2^8 \cdot 7^2 \cdot 127 \cdot 337$
29	9	536870401	$2^9 \cdot 3 \cdot 5^2 \cdot 11 \cdot 31 \cdot 41$
29	18	536608769	$2^{18} \cdot 23 \cdot 89$
29	26	469762049	$2^{26} \cdot 7$
30	18	1073479681	$2^{18} \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
31	1	2147483647	$2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$
31	9	2147483137	$2^9 \cdot 3 \cdot 23 \cdot 89 \cdot 683$
31	17	2147352577	$2^{17} \cdot 3 \cdot 43 \cdot 127$
31	19	2146959361	$2^{19} \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
31	24	2130706433	$2^{24} \cdot 127$
31	25	2113929217	$2^{25} \cdot 3^2 \cdot 7$
31	27	2013265921	$2^{27} \cdot 3 \cdot 5$
32	20	4293918721	$2^{20} \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
32	30	3221225473	$2^{30} \cdot 3$

Table B.1: *cont': Prime numbers of the form* $q = 2^n - 2^m + 1$ *for* $0 < m < n \leq 32$.

Appendix C

Further Properties of Zech's Logarithms

Several properties of Zech's logarithms in Fermat prime fields were considered in Chapter 7. In this appendix we present some additional properties of such logarithms. These properties may be used to derive alternative ways of computing Zech's logarithms in Fermat prime fields.

Theorem C.1 *Let $P(\gamma) = \hat{\gamma}$ be a polar representation of $\gamma \in \mathbb{Z}_{2^m+1}$. For $P(0) = *$ we have*

$$Z(2^{m-1}) \equiv * \pmod{2^m} \tag{C.1}$$

$$Z(*) \equiv 0 \pmod{2^m} \tag{C.2}$$

For nonzero γ, i.e. for $\hat{\gamma} \in \mathbb{Z}_{2^m}$, the following congruences hold.

$$Z(-\hat{\gamma}) \equiv Z(\hat{\gamma}) - \hat{\gamma} \pmod{2^m} \tag{C.3}$$

$$Z\left(Z(\hat{\gamma}) + 2^{m-1}\right) \equiv \hat{\gamma} + 2^{m-1} \pmod{2^m} \tag{C.4}$$

$$Z\left(2^{m-1} - Z(\hat{\gamma})\right) \equiv \hat{\gamma} - Z(\hat{\gamma}) \pmod{2^m} \tag{C.5}$$

$$Z\left(2^{m-1} + \hat{\gamma} - Z(\hat{\gamma})\right) \equiv -Z(\hat{\gamma}) \pmod{2^m} \tag{C.6}$$

$$Z\left(2^{m-1} - \hat{\gamma} + Z(\hat{\gamma})\right) \equiv 2^{m-1} - \hat{\gamma} \pmod{2^m} \tag{C.7}$$

Proof:

- Equation (C.1): From Definitions 7.1 and 7.2 we get $\alpha^{Z(2^{m-1})} \equiv 1+\alpha^{2^{m-1}} \equiv 1-1=0=\alpha^* \pmod{2^m+1}$ and thus $Z(2^{m-1}) \equiv * \pmod{2^m}$.
- Equation (C.2): From $\alpha^{Z(*)} \equiv 1+\alpha^* \equiv 1 \equiv \alpha^0 \pmod{2^m+1}$ we get $Z(*) \equiv 0 \pmod{2^m}$.
- Equation (C.3): Taking the discrete logarithm of the congruence $\alpha^{Z(-\hat{\gamma})} \equiv 1+\alpha^{-\hat{\gamma}} \equiv (1+\alpha^{\hat{\gamma}})\alpha^{-\hat{\gamma}} \equiv \alpha^{Z(\hat{\gamma})-\hat{\gamma}} \pmod{2^m+1}$ yields $Z(-\hat{\gamma}) \equiv Z(\hat{\gamma})-\hat{\gamma} \pmod{2^m}$.
- Equation (C.4): From the congruence $\alpha^{Z\left(Z(\hat{\gamma})+2^{m-1}\right)} \equiv 1-\alpha^{Z(\hat{\gamma})} \equiv 1-(1+\alpha^{\hat{\gamma}}) = \alpha^{\hat{\gamma}+2^{m-1}} \pmod{2^m+1}$ we get $Z\left(Z(\hat{\gamma})+2^{m-1}\right) \equiv \hat{\gamma}+2^{m-1} \pmod{2^m}$.
- Equation (C.5): By (C.3) we get $Z\left(2^{m-1}-Z(\hat{\gamma})\right) \equiv Z\left(-(Z(\hat{\gamma})+2^{m-1})\right) \equiv Z\left(Z(\hat{\gamma})+2^{m-1}\right) - Z(\hat{\gamma}) + 2^{m-1} \pmod{2^m}$. Using (C.4) we then get $Z\left(2^{m-1}-Z(\hat{\gamma})\right) \equiv \hat{\gamma}+2^{m-1}-Z(\hat{\gamma})+2^{m-1} \equiv \hat{\gamma}-Z(\hat{\gamma}) \pmod{2^m}$.
- Equation (C.6): Using (C.3) and (C.5), we can write $Z\left(2^{m-1}+\hat{\gamma}-Z(\hat{\gamma})\right) \equiv Z\left(2^{m-1}-Z(-\hat{\gamma})\right) \equiv -\hat{\gamma}-Z(-\hat{\gamma}) \equiv -\hat{\gamma}-(Z(\hat{\gamma})-\hat{\gamma}) = Z(\hat{\gamma}) \pmod{2^m}$.
- Equation (C.7): Using (C.3) and (C.4), we can write $Z\left(2^{m-1}-\hat{\gamma}+Z(\hat{\gamma})\right) \equiv Z\left(2^{m-1}+Z(-\hat{\gamma})\right) \equiv -\hat{\gamma}+2^{m-1} \pmod{2^m}$.

□

The set of all polar integers $\hat{\gamma} \in \mathbb{Z}_{2^m}$ can be partitioned into subsets such that the Zech logarithms of all integers in each subset can be computed using the knowledge of only *one* logarithm in the subset. This property is demonstrated in the following theorem

Theorem C.2 *Let $\hat{\gamma} \in \mathbb{Z}_{2^m} \setminus \{2^{m-1}\}$ be a polar integer and let f_1, f_2, f_3, f_4, and f_5 be mappings form $\mathbb{Z}_{2^m}$ to $\mathbb{Z}_{2^m}$, given by*

$$f_1(\hat{\gamma}) \equiv -\hat{\gamma} \pmod{2^m} \tag{C.8}$$

$$f_2(\hat{\gamma}) \equiv Z(\hat{\gamma})+2^{m-1} \pmod{2^m} \tag{C.9}$$

$$f_3(\hat{\gamma}) \equiv 2^{m-1}-Z(\hat{\gamma}) \pmod{2^m} \tag{C.10}$$

$$f_4(\hat{\gamma}) \equiv 2^{m-1}+\hat{\gamma}-Z(\hat{\gamma}) \pmod{2^m} \tag{C.11}$$

$$f_5(\hat{\gamma}) \equiv 2^{m-1}-\hat{\gamma}+Z(\hat{\gamma}) \pmod{2^m} \tag{C.12}$$

Let $F(\hat{\gamma})$ *be a set of polar integers defined by* $F(\hat{\gamma}) \triangleq \{\hat{\gamma}, f_1(\hat{\gamma}), f_2(\hat{\gamma}), f_3(\hat{\gamma}), f_4(\hat{\gamma}), f_5(\hat{\gamma})\}$. *Then, we have* $F(\hat{\gamma}) = F(f_1(\hat{\gamma})) = F(f_2(\hat{\gamma})) = F(f_3(\hat{\gamma})) = F(f_4(\hat{\gamma})) = F(f_5(\hat{\gamma}))$.

Proof: Let $j \in \{1, 2, 3, 4, 5\}$. Then, by (C.8), (C.9), (C.10), (C.11), and (C.12) we have

$$\begin{aligned}
f_1(f_j(\hat{\gamma})) &\equiv -f_j(\hat{\gamma}) \pmod{2^m} \\
f_2(f_j(\hat{\gamma})) &\equiv Z(f_j(\hat{\gamma})) + 2^{m-1} \pmod{2^m} \\
f_3(f_j(\hat{\gamma})) &\equiv 2^{m-1} - Z(f_j(\hat{\gamma})) \pmod{2^m} \\
f_4(f_j(\hat{\gamma})) &\equiv 2^{m-1} + f_j(\hat{\gamma}) - Z(f_j(\hat{\gamma})) \pmod{2^m} \\
f_5(f_j(\hat{\gamma})) &\equiv 2^{m-1} - f_j(\hat{\gamma}) + Z(f_j(\hat{\gamma})) \pmod{2^m},
\end{aligned}$$

respectively. Depending on j, Zech's logarithm of $f_j(\hat{\gamma})$ is given by either (C.3), (C.4), (C.5), (C.6), or (C.7). By (C.8) and (C.3) it follows that $Z(f_1(\hat{\gamma})) \equiv Z(-\hat{\gamma}) \equiv Z(\hat{\gamma}) - \hat{\gamma} \pmod{2^m}$. Therefore, for $j = 1$ we get

$$\begin{aligned}
f_1(f_1(\hat{\gamma})) &\equiv -f_1(\hat{\gamma}) \\
&\equiv \hat{\gamma} \pmod{2^m} \\
f_2(f_1(\hat{\gamma})) &\equiv Z(f_1(\hat{\gamma})) + 2^{m-1} \equiv Z(\hat{\gamma}) - \hat{\gamma} + 2^{m-1} \\
&\equiv f_5(\hat{\gamma}) \pmod{2^m} \\
f_3(f_1(\hat{\gamma})) &\equiv 2^{m-1} - Z(f_1(\hat{\gamma})) \equiv 2^{m-1} - (Z(\hat{\gamma}) - \hat{\gamma}) \\
&\equiv f_4(\hat{\gamma}) \pmod{2^m} \\
f_4(f_1(\hat{\gamma})) &\equiv 2^{m-1} + f_1(\hat{\gamma}) - Z(f_1(\hat{\gamma})) \equiv 2^{m-1} - \hat{\gamma} - (Z(\hat{\gamma}) - \hat{\gamma}) \\
&\equiv f_3(\hat{\gamma}) \pmod{2^m} \\
f_5(f_1(\hat{\gamma})) &\equiv 2^{m-1} - f_1(\hat{\gamma}) + Z(f_1(\hat{\gamma})) \equiv 2^{m-1} + \hat{\gamma} + Z(\hat{\gamma}) - \hat{\gamma} \\
&\equiv f_2(\hat{\gamma}) \pmod{2^m}
\end{aligned}$$

and thus $F(f_1(\hat{\gamma})) = \{f_1(\hat{\gamma}), f_1(f_1(\hat{\gamma})), f_2(f_1(\hat{\gamma})), f_3(f_1(\hat{\gamma})), f_4(f_1(\hat{\gamma})), f_5(f_1(\hat{\gamma}))\} = \{f_1(\hat{\gamma}), \hat{\gamma}, f_5(\hat{\gamma}), f_4(\hat{\gamma}), f_3(\hat{\gamma}), f_2(\hat{\gamma})\} = F(\hat{\gamma})$.

For $j = 2, 3, 4, 5$ and $i = 1, 2, 3, 4, 5$, the integer $f_i(f_j(\hat{\gamma}))$ can be obtained in a way similar to the above derivation of $f_i(f_1(\hat{\gamma}))$. All elements $f_i(f_j(\hat{\gamma}))$ are shown in Table C.1. For example, $f_2(f_4(\hat{\gamma}))$ is found as the element $f_3(\hat{\gamma})$ in the intersection of the f_2-row and the f_4-column. The elements in the first row of the table form the set $F(\hat{\gamma})$, the elements in the second row form the

g \ h	—	f_1	f_2	f_3	f_4	f_5	
—	$\hat{\gamma}$	$f_1(\hat{\gamma})$	$f_2(\hat{\gamma})$	$f_3(\hat{\gamma})$	$f_4(\hat{\gamma})$	$f_5(\hat{\gamma})$	$\longleftarrow F(\hat{\gamma})$
f_1	$f_1(\hat{\gamma})$	$\hat{\gamma}$	$f_3(\hat{\gamma})$	$f_2(\hat{\gamma})$	$f_5(\hat{\gamma})$	$f_4(\hat{\gamma})$	$\longleftarrow F(f_1(\hat{\gamma}))$
f_2	$f_2(\hat{\gamma})$	$f_5(\hat{\gamma})$	$\hat{\gamma}$	$f_4(\hat{\gamma})$	$f_3(\hat{\gamma})$	$f_1(\hat{\gamma})$	$\longleftarrow F(f_2(\hat{\gamma}))$
f_3	$f_3(\hat{\gamma})$	$f_4(\hat{\gamma})$	$f_1(\hat{\gamma})$	$f_5(\hat{\gamma})$	$f_2(\hat{\gamma})$	$\hat{\gamma}$	$\longleftarrow F(f_3(\hat{\gamma}))$
f_4	$f_4(\hat{\gamma})$	$f_3(\hat{\gamma})$	$f_5(\hat{\gamma})$	$f_1(\hat{\gamma})$	$\hat{\gamma}$	$f_2(\hat{\gamma})$	$\longleftarrow F(f_4(\hat{\gamma}))$
f_5	$f_5(\hat{\gamma})$	$f_2(\hat{\gamma})$	$f_4(\hat{\gamma})$	$\hat{\gamma}$	$f_1(\hat{\gamma})$	$f_3(\hat{\gamma})$	$\longleftarrow F(f_5(\hat{\gamma}))$

Table C.1: *The table shows, for $i, j = 1, 2, 3, 4, 5$, all combinations of $f_i(f_j(\hat{\gamma})) = g(h(\hat{\gamma}))$, where g and h denote f_i and f_j, respectively. The symbol '—' indicates that no mapping is carried out, i.e. if h '=' — or g '=' — we get the mapping $g(\hat{\gamma})$ or $h(\hat{\gamma})$, respectively.*

set $F(f_1(\hat{\gamma}))$, the elements in the third row form the set $F(f_2(\hat{\gamma}))$, etc. Furthermore, we see that each of the elements $\hat{\gamma}$, $f_1(\hat{\gamma})$, $f_2(\hat{\gamma})$, $f_3(\hat{\gamma})$, $f_4(\hat{\gamma})$, and $f_5(\hat{\gamma})$ only appears *once* in every row (and column) of the table. Hence, we have $F(\hat{\gamma}) = F(f_1(\hat{\gamma})) = F(f_2(\hat{\gamma})) = F(f_3(\hat{\gamma})) = F(f_4(\hat{\gamma})) = F(f_5(\hat{\gamma}))$. □

Thus, Theorem C.2 says that *given two arbitrary elements $\hat{\beta}$ and $\hat{\lambda}$ of $F(\hat{\gamma})$, the sets $F(\hat{\beta})$ and $F(\hat{\lambda})$ are equivalent*. Let $\hat{\beta} \equiv f_3(\hat{\gamma}) \pmod{2^m}$ and $\hat{\lambda} \equiv f_5(\hat{\gamma}) \pmod{2^m}$. Then, by Theorem C.2 we get $F(\hat{\gamma}) = \{\hat{\gamma}, \hat{\beta}, \hat{\lambda}\} \bigcup \{-\hat{\gamma}, -\hat{\beta}, -\hat{\lambda}\}$, where $-\hat{\gamma} \equiv f_1(\hat{\gamma}) \pmod{2^m}$, $-\hat{\beta} \equiv f_2(\hat{\gamma}) \pmod{2^m}$, and $-\hat{\lambda} \equiv f_4(\hat{\gamma}) \pmod{2^m}$. Using these notations for the elements of $F(\hat{\gamma})$, we show in Figure C.1 how these elements are related to each other, via the mappings defined in Theorem C.2. Table C.1 and Figure C.1 are equivalent descriptions of the relations between the elements of $F(\hat{\gamma})$.

It can bee seen in Figure C.1 that the paths $f_3 \leftrightarrow f_5$ associated with a set $F(\hat{\gamma})$ (for some $\hat{\gamma}$) form a pair of triples. The paths $f_3 \leftrightarrow f_5$ are marked with thicker lines in the figure. The set of all integers of $\mathbb{Z}_{2^m}$, except the integer 2^{m-1}, can be partitioned into disjoint subsets of size six, which each can be viewed as such a pair of triples. Note, however, that one of these subsets only comprises three integers: It follows from (C.8), (C.9), (C.10), (C.11), and (C.12) in Theorem C.2 that $f_1(0) = 0$, $f_4(0) = f_3(0)$, and $f_2(0) = f_5(0)$. Hence, the subset $F(0)$ is equal to $\{0, f_3(0), f_5(0)\} = \{0, 2^{m-1} - Z(0), 2^{m-1} + Z(0)\}$, which has size three (3).

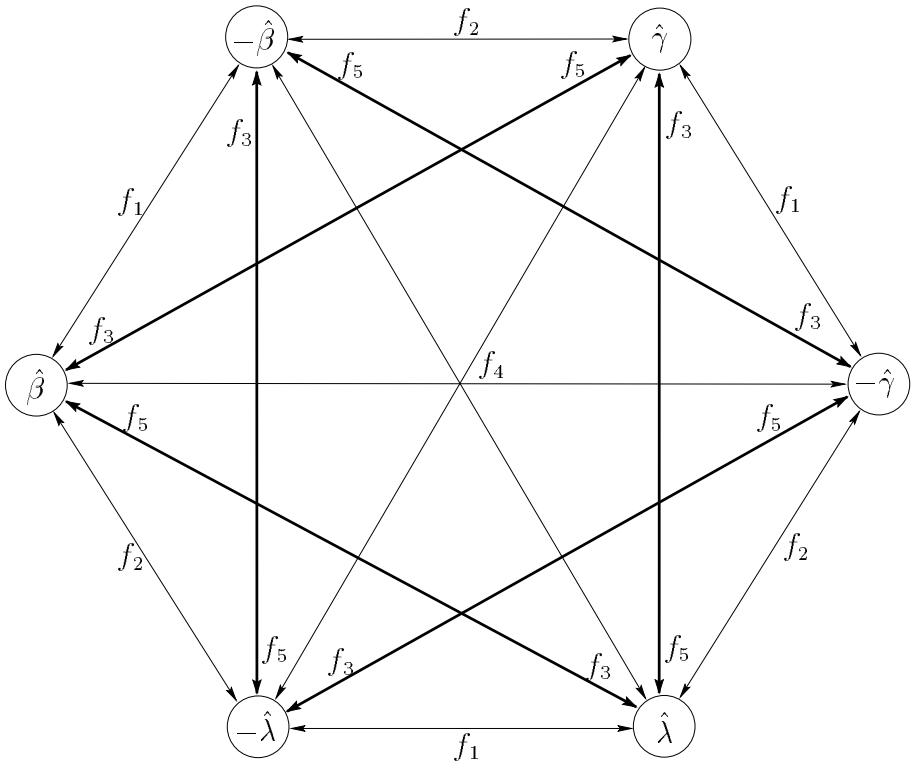


Figure C.1: *Relations between the polar integers* $\hat{\beta}$, $-\hat{\beta}$, $\hat{\gamma}$, $-\hat{\gamma}$, $\hat{\lambda}$, *and* $-\hat{\lambda}$ *modulo* 2^m, *with respect to the mappings* f_1, f_2, f_3, f_4, *and* f_5.

The set of all Zech's logarithms, except $Z(2^{m-1}) = *$, is also partitioned into corresponding subsets of size six. This is illustrated, for $m = 4$, in Figure C.2. The number of disjoint subsets of size six, as described above, equals

$$\frac{(2^m - 1)/3 - 1}{2} = \frac{2^{m-1} - 2}{3}.$$

Suppose the Zech logarithm of one integer, say $\hat{\gamma}$, from each of the above subsets (of size six) is stored in a table. Then, the Zech logarithm $Z(x)$ of an arbitrary integer $x \in \mathbb{Z}_{2^m} \setminus \{2^{m-1}\}$ can be computed in the following way:

1. Find the unique integer $\hat{\gamma}$ which is contained in $F(x)$ and whose Zech's logarithm $Z(\hat{\gamma})$ is stored in the table. The set F was defined in Theorem C.2.

2. Read $Z(\hat{\gamma})$ from the look-up table.

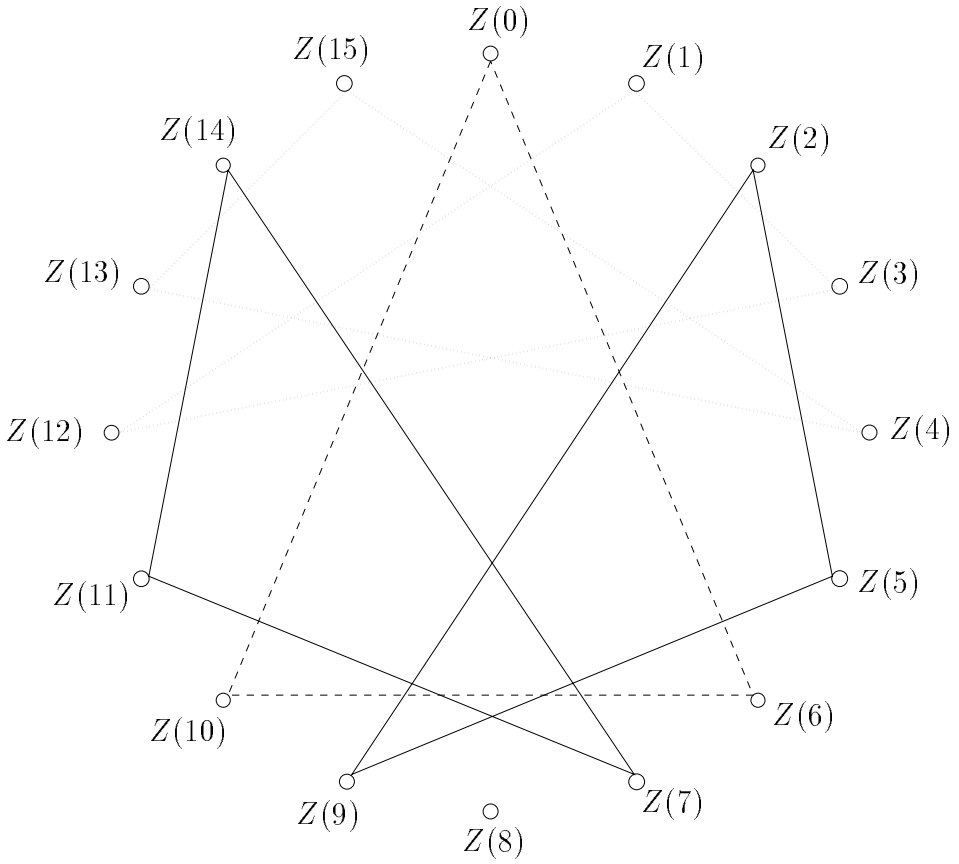


Figure C.2: *The Zech logarithms in $\mathbb{Z}_{2^4+1}$, partitioned into pairs of triples.*

3. Use the congruences in Theorem C.1 and C.2 to compute, from x, $\hat{\gamma}$, and $Z(\hat{\gamma})$, the desired logarithm $Z(x)$.

When 2^{m-1} is added to an m-bit normal binary coded integer, the sum is simply obtained by inverting the most significant bit of the integer. Therefore, apart from this simple operation, the computation of an integer $f_j(\hat{\gamma})$, or its Zech's logarithm $Z(f_j(\hat{\gamma}))$, requires at most one addition modulo 2^m. The main problem here is to carry out Step 1. We have not fully investigated how to select the integers $\hat{\gamma}$ which in a unique way should map to the entries of the look-up table. This problem is similar to the problem in Section 7.5.4 of finding the unique positions in $\mathcal{M}_m$.

We conclude this appendix by presenting two properties of the subset $F(x)$. These properties may be of help in Step 1, when trying to find the unique integer $\hat{\gamma}$ of $F(x)$. Consider the set $F(x)$ of integers, where x is an arbitrary integer in any triple, as described above. Then, by the congruences in Theorems C.1

and C.2 we straightforwardly obtain the two following properties:[1]

$$\begin{aligned} x + f_3(x) + f_5(x) &\equiv x + \left(2^{m-1} - Z(x)\right) + \left(2^{m-1} - x + Z(x)\right) \\ &\equiv 0 \pmod{2^m} \end{aligned} \tag{C.13}$$

$$\begin{aligned} Z(x) + Z(f_3(x)) + Z(f_5(x)) &\equiv Z(x) + (x - Z(x)) + \left(2^{m-1} - x\right) \\ &\equiv 2^{m-1} \pmod{2^m}. \end{aligned} \tag{C.14}$$

Remark: We have derived still more properties of Zech's logarithms in Fermat prime fields. However, these properties are not considered here.

[1] Alternatively, using the above notations, we can write $\hat{\gamma} + \hat{\beta} + \hat{\lambda} \equiv 0 \pmod{2^m}$ and $Z(\hat{\gamma}) + Z(\hat{\beta}) + Z(\hat{\lambda}) \equiv 2^{m-1} \pmod{2^m}$.

Bibliography

[1] M. Afgahi and J Yuan, "A Novel Implementation of Double-Edge Trigger Flip-Flop for High Speed CMOS Circuit", *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 8, pp. 1168–1170, August 1991.

[2] R. C. Agarwal and C. S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-22, No. 2, pp. 87–97, April 1974.

[3] D. P. Agrawal and T. R. N. Rao, "Modulo $(2^n + 1)$ arithmetic logic", *IEE Journ. Electronic Circuits and Systems*, Vol. 2, pp. 186–188, November 1978.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[5] L.-I. Alfredsson, "Properties of Zech's Logarithms over Fermat Prime Fields", *Proc. Sixth Joint Swedish-Russian International Workshop on Information Theory*, Mölle, Sweden, pp. 310–314, August 1993.

[6] L.-I. Alfredsson, "A Fast Fermat Number Transform for Long Sequences", *Proc. Seventh European Signal Processing Conference, (EUSIPCO-94)*, Edinburgh, Scotland, Vol. III, pp. 1579–1581, September 1994.

[7] L.-I. Alfredsson, "A Mirrored Integer Sequence of Length 2^m and the Discrete Logarithm in Fermat Prime Fields", *Proc. Sixth Joint*

Swedish-Russian International Workshop on Information Theory, St.-Petersburg, Russia, pp. 15–19, June 1995.

[8] M. Annaratone, *Digital CMOS Circuit Design*, Kluwer Academic Publishers, 1986.

[9] B. Arambepola and S. Choomchuay, "Algorithms and Architectures for Reed-Solomon Codes", *GEC Journal of Research*, Vol. 9, No. 3, pp. 172–184, 1992.

[10] A. S. Ashur, "Area-Time Efficient Diminished–1 Multiplier for Fermat Number Transform", *Electronic Letters*, Vol. 30, No. 20, pp. 1640–1641, September 1994.

[11] M. Benaissa, A. Bouridane, S. S. Dlay, and A. G. J. Holt, "Diminished–1 Multiplier for a Fast Convolver and Correlator Using the Fermat Number Transform", *IEE Proceedings*, Vol. 135, Pt. G, No. 5, pp. 187–193, October 1988.

[12] M. Benaissa, A. Pajayakrit, S. S. Dlay, and A. G. J. Holt, "VLSI Design for Diminished–1 Multiplication of Integers Modulo a Fermat Number", *IEE Proceedings*, Vol. 135, Pt. E, No. 3, pp. 161–164, May 1988.

[13] M. Benaissa, S. S. Dlay, and A. G. J. Holt, "CMOS VLSI Design of a High-Speed Fermat Number Transform Based Convolver/Correlator Using Three-Input Adders", *IEE Proceedings*, Vol. 138, Pt. G, No. 2, pp. 182–190, April 1991.

[14] H. B. Bakoglu, *Circuits, Interconnections, and Packing for VLSI*, Addison-Wesley, 1990.

[15] G. Bilardi, M. Pracchi, and F. P. Preparata, "A Critique and an Appraisal of VLSI Models of Computation", *VLSI Systems and Computations*, pp. 81–88, Editors: H. T. Kung, B. Sproull, and G. Steele, Springer-Verlag, 1981.

[16] R. E. Blahut, *Theory and Practice of Error Control Codes*, Adison-Wesley, 1984.

[17] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*, Adison-Wesley, 1985.

[18] I. E. Bocharova and B. D. Kudryashov, "Fast Exponentiation Based on Lempel-Ziv Algorithm", *Proc. of the Sixth Joint Swedish-Russian International Workshop on Information Theory*, Mölle, Sweden, pp. 259–263, August 1992.

[19] I. E. Bocharova and B. D. Kudryashov, "Fast Exponentiation Based on Data Compression Algorithms", *Proc. of the Seventh Joint Swedish-Russian International Workshop on Information Theory*, St.-Petersburg, Russia, pp. 36–39, June 1995.

[20] S. Boussakta and A. G. J. Holt, "Calculation of the discrete Hartley transform via the Fermat number transform using a VLSI chip", *IEE Proceedings*, Vol. 135, Pt. G, No. 3, pp. 101–103, June 1988.

[21] S. Boussakta and A. G. J. Holt, "Fast multidimensional discrete Hartley transform using Fermat number transform", *IEE Proceedings*, Vol. 135, Pt. G, No. 6, pp. 253–257, December 1988.

[22] S. Boussakta and A. G. J. Holt, "Relationship between the Fermat number transform and the Walsh-Hadamard transform", *IEE Proceedings*, Vol. 136, Pt. G, No. 4, pp. 191–204, August 1989.

[23] S. Boussakta, A. Y. Md. Shakaff, F. Marir, and A. G. J. Holt, "Number theoretic transforms of periodic structures and their applications", *IEE Proceedings*, Vol. 135, Pt. G, No. 2, pp. 83–96, April 1988.

[24] R. P. Brent, "Factorization of the Eleventh Fermat Number" (preliminary report), *Abstracts, Amer. Math. Soc.*, Vol. 10, 89T-11-73, 1989.

[25] R. P. Brent, "Parallel Algorithms for Integer Factorizations", *Number Theory and Cryptography*, London Math. Soc. Lecture Note Series, Editor: J. H. Loxton, Vol. 154, Cambridge, 1990.

[26] R. P. Brent and H. T. Kung, "The Area-Time Complexity of Binary Multiplication", *Journ. of the Ass. for Comp. Mash.*, Vol. 28, No. 3, pp. 521–534, July 1981.

[27] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Trans. on Computers*, Vol. C-31, No. 3, pp. 260–264, March 1982.

[28] J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1,\ b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers*, Contemporary Mathematics, Volume 22, American Mathematical Society, Second Edition, 1988.

[29] J. T. Butler (editor), *Multiple-valued logic in VLSI*, IEEE Computer Press Society, Los Alamitos, 1991.

[30] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473–484, April 1992.

[31] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[32] J. J. Chang, T. K. Truong, H. M. Shao, I. S. Reed, and I-S Hsu, "The VLSI Design of a Single Chip for the Multiplication of Integers Modulo a Fermat Number", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-33, No. 6, pp. 1599–1602, December 1985.

[33] P. R. Chevillat, "Transform-Domain Digital Filtering with Number Theoretic Transforms and Limited Word Lengths", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-26, No. 4, pp. 284–290, August 1978.

[34] J. H. Conway, "A Tabulation of Some Information Concerning Finite Fields", *Computers in Mathematical Research* (R. F. Churchhouse and J.-C. Herz, Editors), pp. 37–50, North-Holland, Amsterdam, 1968.

[35] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics Computation*, Vol. 19, pp. 297–301, 1965.

[36] L. E. Dickson, *History of the theory of numbers*, Vol. I, Washington D. C.: Carnegie Institute, 1919.

[37] V. S. Dimitrov, T. V. Cooklev, and B. D. Donevsky, "Generalized Fermat-Mersenne Number Theoretic Transform", *IEEE Trans. on Circuits and Systems–II: Analog and Digital Signal Processing*, Vol. 41, No. 2, pp. 1–7, February 1994.

[38] E. Dubois and A. N. Venetsanopoulos, "Number Theoretic Transforms with modulus $2^{2q} - 2^q + 1$", *Rec. 1978 IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 624–627, April 1978.

[39] E. Dubois and A. N. Venetsanopoulos, "The Generalized Discrete Fourier Transform in Rings of Algebraic Integers", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-28, No. 2, pp. 169–175, April 1980.

[40] P. Duhamel and H. Hollman, "Split-radix FFT Algorithm", *Electron. Lett.*, Vol. 20, pp. 14–16, January 1984.

[41] P. Duhamel, "Implementation of 'Split-Radix' FFT Algorithms for Complex, Real, and Real-Symmetric Data", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-34, No. 2, pp. 285–295, April 1986.

[42] H. M. Edwards, *Fermat's Last Theorem, A Genetic Introduction to Algebraic Number Theory*, Springer-Verlag, New York 1977.

[43] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifier", *Journal of Applied Physics*, Vol. 19, No. 1, pp. 55–63, January 1948.

[44] R. L. Geiger, P. E. Allen, and N. R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill Publishing Company, 1991.

[45] W. M. Gentleman and G. Sande, "Fast Fourier transforms for fun and profit", *Fall Joint Computing Conference*, AFIPS Proc., Vol. 29, pp. 563–578, 1966.

[46] D. Gollman, Y. Han, and C. J. Mitchell, "Redundant Integer Representations and Fast Exponentiation", To appear in *Designs, Codes and Cryptography*.

[47] S. W. Golomb, "Properties of the Sequence $3 \cdot 2^n + 1$", *Mathematics of Computation*, Vol. 30, N0. 135, pp. 657–663, July 1976.

[48] S. W. Golomb, I. S. Reed, and T. K. Truong, "Integer Convolutions over the Finite Field $GF(3 \cdot 2^n + 1)$", *SIAM Journal of Applied Math.*, Vol. 32, No. 2, pp. 356–365, March 1977.

[49] N. Hedenstierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 2, pp. 270-281, March 1987.

[50] I. N. Herstein, *Topics in Algebra*, Second Edition, John Wiley & Sons, 1975.

[51] K. Huber, "Some Comments on Zech's Logarithms", *IEEE Trans. on Inf. Theory*, Vol. IT-36, No. 4, pp. 946–950, July 1990.

[52] K. Hwang, *Computer Arithmetic: principles, architecture, and design*, John Wiley & Sons, 1979.

[53] K. Imamura, "A Method for Computing Addition Tables in $GF(p^n)$", *IEEE Trans. on Inf. Theory*, Vol. IT-26, No. 3, pp. 367–369, May 1980.

[54] J. Justesen, "On the Complexity of Decoding Reed-Solomon Codes", *IEEE Trans. on Inf. Theory*, Vol. IT-22, pp. 237–238, March 1976.

[55] A. Karatsuba and Y. Hofman, "Multiplication of multidigit numbers on automata" (in Russian), *Dokl. Akad. Nauk SSSR*, Vol. 145, pp. 293–294, 1962.

[56] D. E. Knuth, *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

[57] S. Lang, *Algebraic Number Theory*, Springer-Verlag, New York, 1986.

[58] L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-24, No. 5, pp. 356–359, October 1976.

[59] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, "The factorization of the ninth Fermat Number", *Math. Comp.*, Vol. 61, pp. 318–349, 1993.

[60] R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopedia of Mathematics and its Applications, Volume 20, Cambridge University Press, 1984.

[61] D. Liu, *Low Power Digital CMOS Design*, Ph.D. dissertation, No. 364, Linköping University, Linköping, Sweden 1994.

[62] K. Y. Liu, I. S. Reed, and T. K. Truong, "Fast Number-Theoretic Transforms for Digital Filtering", *Electronic Latters*, Vol. 12, No. 24, pp. 644–646, November 1976.

[63] K. Y. Liu, I. S. Reed, and T. K. Truong, "High-Radix Transforms for Reed-Solomon Codes over Fermat Primes", *IEEE Trans. on Inf. Theory*, Vol. IT-23, No. 6, pp. 776–778, November 1977.

[64] B. J. McCarroll, C. G. Sodini, and H.-S. Lee, "A High-Speed CMOS Comparator for Use in an ADC", *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 1, pp. 159–165, February 1988.

[65] J. H. McClellan, "Hardware Realization of a Fermat Number Transform", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-24, No. 3, pp. 216–225, June 1976.

[66] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[67] A. M. Mohsen and C. A. Mead, "Delay-Time Optimization for Driving and Sensing of Signals on High-Capacitance Paths of VLSI Systems", *IEEE Journal of Solid-State Circuits*, Vol. SC-14, No. 2, pp. 462–470, April 1979.

[68] Y. Morikawa, H. Hamada, and K. Nagayasu, "Hardware Realisation of High Speed Butterfly for the Maximum Length Fermat Number Transform", *Trans. IECE, Japan*, Vol. J66-D, No. 1, pp. 81–88 1983. (We have not yet been able to get a copy of this reference.)

[69] H. Murakami, I. S. Reed, and L. R. Welch, "A Transform Decoder for Reed-Solomon Codes in Multiple-User Communication Systems", *IEEE Trans. on Inf. Theory*, Vol. IT-23, No. 6, pp. 675–683, November 1977.

[70] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-4, No. 3, pp. 336–349, July 1985.

[71] A. Pajayakrit, *VLSI Architecture and Design for the Fermat Number Transform Implementation*, PhD. Thesis, Dept. of Electrical and Electronic Engineering, University of Newcastle-Upon-Tyne, United Kingdom, 1988.

[72] S. C. Pohlig and M. E. Hellman, "An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance", *IEEE Trans. on Inf. Theory*, Vol. IT-24, No. 1, pp. 106–110, January 1978.

[73] J. M. Pollard, "Implementation of Number-Theoretic Transforms", *Electronic Letters*, Vol. 12, No. 15, pp. 378–379, July 1976.

[74] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Second Edition, Macmillan Publ. Comp., 1992.

[75] J. G. Proakis, C. M. Rader, F. Ling, and C. L. Nikias, *Advanced Digital Signal Processing*, Macmillan Publ. Company, 1992.

[76] D. A. Pucknell and K. Eshraghian, *Basic VLSI Design*, Third Edition, Prentice Hall, 1994.

[77] C. M. Rader, "Discrete Convolution via Mersenne Transforms", *IEEE Trans. Comput.*, Vol. C-21, No. 12, pp. 1269–1273, December 1972.

[78] C. M. Rader, "On the application of the number theoretic methods of high-speed convolution to two-dimensional filtering", *IEEE Trans. on Circuits and Systems*, Vol. 22, p. 575, 1975.

[79] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1975.

[80] I. S. Reed, R. A. Scholtz, T. K. Truong, and L. R. Welsh, "The Fast Decoding of Reed-Solomon Codes Using Fermat Theoretic Transforms and Continued Fractions", *IEEE Trans. on Inf. Theory*, Vol. IT-24, No. 1, pp. 100–106, January 1978.

[81] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 8, pp. 300–304, June 1960.

[82] I. S. Reed, T. K. Truong, and L. R. Welch, "The Fast Decoding of Reed-Solomon Codes Using Fermat Transforms", *IEEE Trans. on Inf. Theory*, Vol. IT-24, No. 4, pp. 497–499, July 1978.

[83] R. M. Robinson, "A Report on Primes of the Form $k \cdot 2^n + 1$ and on Factors of Fermat Numbers", *Proc. of the Amer. Math. Soc.*, Vol. 9, pp. 673–681, 1958.

[84] K. H. Rosen, *Elementary Number Theory and its Applications*, Third Edition, Addison-Wesley, 1993.

[85] J. Rubinstein, P. Penfield Jr., and M. A. Horowitz, "Signal Delay in RC Tree Networks", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-2, No. 3, pp. 202–211, July 1983.

[86] A. Schönhage and V. Strassen, "Fast multiplication of integers" (in German), *Computing*, Vol. 7, pp. 281–292, 1971.

[87] W. C. Siu and A. G. Constantinides, "Very fast discrete Fourier transform, using number theoretic transform", *IEE Proceedings*, Vol. 130, Pt. G, No. 5, pp. 201–204, October 1983.

[88] W. C. Siu and A. G. Constantinides, "On the computation of discrete Fourier transform using Fermat number transform", *IEE Proceedings*, Vol. 131, Pt. F, No. 1, pp. 7–14, February 1984.

[89] A. Y. Md. Shakaff, *Practical Implementation of the Fermat Number Transform with Applications to Filtering and Image Processing*, PhD. Thesis, Dept. of Electrical and Electronic Engineering, University of Newcastle-Upon-Tyne, United Kingdom, 1988.

[90] A. Y. Md. Shakaff, A. Pajayakrit, and A. G. J. Holt, "Practical implementations of block-mode image filters using the Fermat number transform on a microprocessor based system", *IEE Proceedings*, Vol. 135, Pt. G, No. 4, pp. 141–154, August 1988.

[91] A. Shiozaki, T. K. Truong, K. M. Cheung, and I. S. Reed, "Fast Transform Decoding of Nonsystematic Reed-Solomon codes", *IEE Proceedings*, Vol. 137, Pt. E, No. 2, pp. 139–143, March 1990.

[92] H. C. Shyu, T. K. Truong, I. I. Reed, I. S Hsu, and J. J. Chang, "A New VLSI Complex Integer Multiplier Which Uses a Quadratic-Polynomial Residue System with Fermat Numbers", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. ASSP-35, No. 7, pp. 1076–1079, July 1987.

[93] W. Sierpiński, *Elementary Theory of Numbers*, Editor: A. Schinzel, Second Edition, PWN-Polish Scientific Publishers, 1988.

[94] A. N. Skodras and A. G. Constantinides, "Efficient Computation of the Split-Radix FFT", *IEE Proceedings–F*, Vol. 139, No. 1, pp. 56–60, February 1992.

[95] I. N. Stewart and D. O. Tall, *Algebraic Number Theory*, Second Edition, Chapman & Hall, 1987, Reprinted 1994.

[96] R. Sundblad and C. Svensson, "Fully Dynamic Switch-Level Simulation of CMOS Circuits", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 2, pp. 282–289, March 1987.

[97] S. Sunder, F. El-Guibali, and A. Antoniou, "Area-efficient Diminished–1 Multiplier for Fermat Number-theoretic Transform", *IEE Proceedings–G*, Vol. 140, No. 3, pp. 211–215, June 1993.

[98] C. Svensson, K. Cheng, and J. Yuan, "Decisionmaking in Fast A/D Converters", Int. Report LiTH-IFM-IS-154, Linköping University, October 1989.

[99] C. Svensson and D. Liu, "Low Power Circuit Techniques", Manuscript, 1995.

[100] C. D. Thompson, "Area-Time Complexity for VLSI", *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, pp. 81–88, 1979.

[101] P. J. Towers, A. Pajayakrit, and A. G. J. Holt, "Cascadable NMOS VLSI circuit for implementing a fast convolver using the Fermat number transform", *IEE Proceedings*, Vol. 134, Pt. G, No. 2, pp. 57–66, April 1987.

[102] T. K. Truong, J. J. Chang, I. S. Hsu, D. Y. Pei, and I. S. Reed, "Techniques for Computing the Discrete Fourier Transform Using the Quadratic Residue Fermat Number Systems", *IEEE Trans. on Computers*, Vol. C-35, No. 11, pp. 1008–1012, November 1986.

[103] T. K. Truong, I. S. Reed, C. -Yeh, and H. M. Shao, "A Parallel VLSI Architecture for a Digital Filter of Arbitrary Length Using Fermat Number Transforms", *Proceedings of IEEE International Conference on Circuits and Computers (ICCC '82)*, New York, pp. 574–578, Sept. 28 – Oct. 1, 1982.

[104] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1983.

[105] J. P. Uyemura, *Circuit Design for CMOS VLSI*, Kluwer Academic Publishers, 1992.

[106] J. P. Uyemura, *Fundamentals of MOS Digital Integrated Circuits*, Addison-Wesley, 1988.

[107] C. -Yeh, I. S. Reed, J. J. Chang, and T. K. Truong, "VLSI Design of Number Theoretic Transforms for a Fast Convolution", *Proceedings of IEEE International Conference on Computer Design: VLSI in Computing*, New York, pp. 200–203, October 1983.

[108] J. Yuan, I. Karlsson, and C. Svensson, "A True Single-Phase-Clock Dynamic CMOS Circuit Technique", *IEEE Journal of Solid-State Circuits*, Vol. 22, No. 5, pp. 899–901, October 1987.

[109] J. Yuan and C. Svensson, "CMOS Circuit Speed Optimization Based on Switch Level Simulation", *Proceedings of 1988 IEEE International Symposium on Circuits and Systems*, Espoo, Finland, Vol. 3, pp. 2109–2112, June 1988.

[110] J. Yuan, C. Svensson, and P. Larsson, "New Domino Logic Precharged by Clock and Data", *Electronic Letters*, Vol. 29, No. 25, pp. 2188–2189, December 1993.

[111] L. Wanhammar, B. Sikström, *DSP Integrated Circuits*, Dept. of EE, Linköping University, Sweden, 1990.

[112] B. W. Y. Wei and C. D. Thompson, "Area-Time Optimal Adder Design", *IEEE Trans. on Computers*, Vol. C-39, No. 5, pp. 666–675, May 1990.

[113] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Second Edition, Addison-Wesley Publ. Comp., 1993.

[114] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[115] D. Zuras, "More on Squaring and Multiplying Large Integers", *IEEE Trans. on Computers*, Vol. 43, No. 8, pp. 899–908, August 1994.

Linköping Studies in Science and Technology
Dissertations, Data Transmission

V. Ramamoorthy: *Speech coding based on a composite-Gaussian source model.*
Dissertation No. 60, 1981.

Jan-Erik Stjernvall: *A study of distortion measures for source coding.*
Dissertation No. 68, 1981.

Jens Zander: *Distributed access algorithms for a class of multi-access channels.*
Dissertation No. 123, 1985.

Edoardo Mastrovito: *VLSI architectures for computations in Galois fields.*
Dissertation No. 242, 1991.

Shakir Abdul-Jabbar: *Disjunctive codes for the multiple access OR-channel.*
Dissertation No. 254, 1991.

Youzhi Xu: *Contributions to the decoding of Reed-Solomon and related codes.*
Dissertation No. 257, 1991.

Tommy Pedersen: *Performance aspects of concatenated codes.*
Dissertation No. 245, 1992.

Jan Nilsson: *On hard and soft decoding of block codes.*
Dissertation No. 333, 1994.

Per-Olof Anderson: *Superimposed codes for the Euclidean channel.*
Dissertation No. 342, 1994.

Per Larsson: *Codes for correction of localized errors.*
Dissertation No. 374, 1995.

Eva Englund: *Codes with unequal error protection.*
Dissertation No. 412, 1995.

Ralf Kötter: *On algebraic decoding of algebraic-geometric and cyclic codes.*
Dissertation No. 419, 1996.