P. V. Ananda Mohan

Residue Number Systems

Theory and Applications

Birkhäuser

P.V. Ananda Mohan

Residue Number Systems

Theory and Applications

Birkhäuser

P.V. Ananda Mohan
R&D
CDAC
Bangalore, Karnataka
India

ISBN 978-3-319-41383-9 ISBN 978-3-319-41385-3 (eBook)
DOI 10.1007/978-3-319-41385-3

Library of Congress Control Number: 2016947081

Mathematics Subject Classification (2010): 68U99, 68W35

Printed on acid-free paper

This book is published under the trade name Birkhäuser
The registered company is Springer International Publishing AG Switzerland (www.birkhauser-science.com)

To
The Goddess of learning Saraswati
and
Shri Mahaganapathi

Preface

The design of algorithms and hardware implementation for signal processing systems has received considerable attention over the last few decades. The primary area of application was in digital computation and digital signal processing. These systems earlier used microprocessors, and, more recently, field programmable gate arrays (FPGA), graphical processing units (GPU), and application-specific integrated circuits (ASIC) have been used. The technology is evolving continuously to meet the demands of low power and/or low area and/or computation time.

Several number systems have been explored in the past such as the conventional binary number system, logarithmic number system, and residue number system (RNS), and their relative merits have been well appreciated. The residue number system was applied for digital computation in the early 1960s, and hardware was built using the technology available at that time. During the 1970s, active research in this area commenced with application in digital signal processing. The emphasis was on exploiting the power of RNS in applications where several multiplications and additions needed to be carried out efficiently using small word length processors. The research carried out was documented in an IEEE press publication in 1975. During the 1980s, there was a resurgence in this area with an emphasis on hardware that did not need ROMs. Extensive research has been carried out since 1980s and several techniques for overcoming certain bottlenecks in sign detection, scaling, comparison, and forward and reverse conversion.

A compilation of the state of the art was attempted in 2002 in a textbook, and this was followed by another book in 2007. Since 2002, several new investigations have been carried out to increase the dynamic range using more moduli, special moduli which are close to powers of two, and designs that use only combinational logic. Several new algorithms/theorems for reverse conversion, comparison, scaling, and error correction/detection have also been investigated. The number of moduli has been increased, yet the same time focusing on retaining the speed/area advantages.

It is interesting to note that in addition to application in computer arithmetic, application in digital communication systems has gained a lot of attention. Several applications in wireless communication, frequency synthesis, and realization of

transforms such as discrete cosine transform have been explored. The most interesting development has been the application of RNS in cryptography. Some of the cryptography algorithms used in authentication which need big word lengths ranging from 1024 bits to 4096 bits using RSA (Rivest Shamir Adleman) algorithm and with word lengths ranging from 160 bits to 256 bits used in elliptic curve cryptography have been realized using the residue number systems. Several applications have been in the implementation of Montgomery algorithm and implementation of pairing protocols which need thousands of modulo multiplication, addition, and reduction operations. Recent research has shown that RNS can be one of the preferred solutions for these applications, and thus it is necessary to include this topic in the study of RNS-based designs.

This book brings together various topics in the design and implementation of RNS-based systems. It should be useful for the cryptographic research community, researchers, and students in the areas of computer arithmetic and digital signal processing. It can be used for self-study, and numerical examples have been provided to assist understanding. It can also be prescribed for a one-semester course in a graduate program.

The author wishes to thank Electronics Corporation of India Limited, Bangalore, where a major part of this work was carried out, and the Centre for Development of Advanced Computing, Bangalore, where some part was carried out, for providing an outstanding R&D environment. He would like to express his gratitude to Dr. Nelaturu Sarat Chandra Babu, Executive Director, CDAC Bangalore, for his encouragement. The author also acknowledges Ramakrishna, Shiva Rama Kumar, Sridevi, Srinivas, Mahathi, and his grandchildren Baby Manognyaa and Master Abhinav for the warmth and cheer they have spread. The author wishes to thank Danielle Walker, Associate Editor, Birkhäuser Science for arranging the reviews, her patience in waiting for the final manuscript and assistance for launching the book to production. Special thanks are also to Agnes Felema. A and the Production and graphics team at SPi-Global for their most efficiently typesetting, editing and readying the book for production.

Bangalore, India P.V. Ananda Mohan
April 2015

Contents

1 Introduction 1
References 6

2 Modulo Addition and Subtraction 9
2.1 Adders for General Moduli 9
2.2 Modulo (2^n-1) Adders 12
2.3 Modulo (2^n+1) Adders 16
References 24

3 Binary to Residue Conversion 27
3.1 Binary to RNS Converters Using ROMs 27
3.2 Binary to RNS Conversion Using Periodic Property of Residues of Powers of Two 28
3.3 Forward Conversion Using Modular Exponentiation 30
3.4 Forward Conversion for Multiple Moduli Using Shared Hardware 32
3.5 Low and Chang Forward Conversion Technique for Arbitrary Moduli 34
3.6 Forward Converters for Moduli of the Type $(2^n \pm k)$ 35
3.7 Scaled Residue Computation 36
References 37

4 Modulo Multiplication and Modulo Squaring 39
4.1 Modulo Multipliers for General Moduli 39
4.2 Multipliers mod (2^n-1) 44
4.3 Multipliers mod (2^n+1) 51
4.4 Modulo Squarers 69
References 77

5 RNS to Binary Conversion 81
5.1 CRT-Based RNS to Binary Conversion 81
5.2 Mixed Radix Conversion-Based RNS to Binary Conversion . . . 90

5.3 RNS to Binary Conversion Based on New CRT-I, New CRT-II, Mixed-Radix CRT and New CRT-III 95
5.4 RNS to Binary Converters for Other Three Moduli Sets 97
5.5 RNS to Binary Converters for Four and More Moduli Sets 99
5.6 RNS to Binary Conversion Using Core Function 111
5.7 RNS to Binary Conversion Using Diagonal Function 114
5.8 Performance of Reverse Converters 117
References 128

6 Scaling, Base Extension, Sign Detection and Comparison in RNS 133
6.1 Scaling and Base Extension Techniques in RNS 133
6.2 Magnitude Comparison 153
6.3 Sign Detection 157
References 160

7 Error Detection, Correction and Fault Tolerance in RNS-Based Designs 163
7.1 Error Detection and Correction Using Redundant Moduli 163
7.2 Fault Tolerance Techniques Using TMR 173
References 174

8 Specialized Residue Number Systems 177
8.1 Quadratic Residue Number Systems 177
8.2 RNS Using Moduli of the Form r^n 179
8.3 Polynomial Residue Number Systems 184
8.4 Modulus Replication RNS 186
8.5 Logarithmic Residue Number Systems 189
References 191

9 Applications of RNS in Signal Processing 195
9.1 FIR Filters 195
9.2 RNS-Based Processors 220
9.3 RNS Applications in DFT, FFT, DCT, DWT 226
9.4 RNS Application in Communication Systems 242
References 256

10 RNS in Cryptography 263
10.1 Modulo Multiplication Using Barrett's Technique 265
10.2 Montgomery Modular Multiplication 267
10.3 RNS Montgomery Multiplication and Exponentiation 287
10.4 Montgomery Inverse 295
10.5 Elliptic Curve Cryptography Using RNS 298
10.6 Pairing Processors Using RNS 306
References 343

Index 349

Chapter 1
Introduction

Digital computation is carried out using binary number system conventionally. Processors with word lengths up to 64 bits have been quite common. It is well known that the basic operations such as addition can be carried out using variety of adders such as carry propagate adder, carry look ahead adders and parallel-prefix adders with different addition times and area requirements. Several algorithms for high-speed multiplication and division also are available and are being continuously researched with the design objectives of low power/low area/high speed. Fixed-point as well as floating-point processors are widely available. Interestingly, operations such as sign detection, magnitude comparison, and scaling are quite easy in these systems.

In applications such as cryptography there is a need for processors with word lengths ranging from 160 bits to 4096 bits. In such requirements, a need is felt for reducing the computation time by special techniques. Applications in digital signal processing also continuously look for processors for fast execution of multiply and accumulate instruction. Several alternative techniques have been investigated for speeding up multiplication and division. An example is using logarithmic number systems (LNS) for digital computation. However, using LNS, addition and subtraction are difficult.

In binary and decimal number systems, the position of each digit determines the weight. The leftmost digits have higher weights. The ratio between adjacent digits can be constant or variable. The latter is called *Mixed Radix Number System* [1]. For a given integer X, the MRS digit can be found as

$$x_i = \left\lfloor \frac{X}{\prod_{j=0}^{i-1} M_j} \right\rfloor \bmod M_i \tag{1.1a}$$

P.V. Ananda Mohan, *Residue Number Systems*, DOI 10.1007/978-3-319-41385-3_1

where $0 \leq i < n$, n is the number of digits. Note that M_j is the ratio between weights for the jth and $(j+1)$ th digit position and x mod y is the remainder obtained by dividing x with y. MRNS can represent

$$M = \prod_{j=0}^{n-1} M_j \tag{1.1b}$$

unique values. An advantage is that it is easy to perform the inverse procedure to convert the tuple of digits to the integer value:

$$X = \sum_{i=0}^{n-1} \left(x_i \prod_{j=0}^{i-1} M_j \right) \tag{1.1c}$$

Fixed-point addition is easy since it is equivalent to integer addition. Note that Q15 format often used in digital signal processing has one sign bit and fifteen fractional bits. Fixed-point multiplication shall use scaling so as to make the product in the same format as the inputs. Fixed-point addition of fractional numbers is more difficult than multiplication since both numbers must be in the same format and attention must be paid to the possibility of an overflow. The overflow can be handled by right shifting by one place and setting an exponent flag or by using double precision to provide headroom allowing growth due to overflow [2].

The floating-point number for example is represented in IEEE 754 standard as [2]

$$X = (-1)^s (1.F) \times 2^{E-127} \tag{1.2}$$

where F is the mantissa in two's-complement binary fraction represented by bits 0–22, E is the exponent in excess 127 format and $s = 0$ for positive integers and $s = 1$ for negative numbers. Note the assumed 1 preceding the mantissa and biased exponent. As an illustration, consider the floating-point number

0	1000011.	11000...00
Sign	Exponent	Mantissa

The mantissa is 0.75 and exponent is 131. Hence $X = (1.75) \times 2^{131-127} = (1.75) \times 2^4$. When floating-point numbers are added, the exponents must be made equal (known as alignment) and we need to shift right the mantissa of the smaller operand and increment the exponent till it is equal to that of the large operand. The multiplication of the properly normalized floating-point numbers $M_1 2^{E_1}$ and $M_2 2^{E_2}$ yields the product given by $M^E = (M_1 M_2) 2^{E_1 + E_2}$. The largest and smallest numbers that can be represented are $\pm 1.2 \times 10^{38}$ and $\pm 3.4 \times 10^{-38}$.

In the case of double precision [3, 4], bits 0–51 are mantissa and bits 52–62 are exponent and bit 63 is the sign bit. The offset in this case is 1023 allowing exponents from 2^{-1023} to 2^{+1024}. The largest and smallest numbers that can be represented are $\pm 1.8 \times 10^{308}$ and $\pm 2.2 \times 10^{-308}$.

In floating-point representation, errors can occur both in addition and multiplication. However, overflow is very unlikely due to the very wide dynamic range since more bits are available in the exponent. Floating-point arithmetic is more expensive and slower.

In logarithmic number system (LNS) [5], we have

$$X \rightarrow (z, s, x = \log_b |X|) \tag{1.3a}$$

where b is the base of the logarithm, z when asserted indicates that $X = 0$, s is the sign of X. In LNS, the input binary numbers are converted into logarithmic form with a mantissa and characteristic each of appropriate word length to achieve the desired accuracy. As is well known, multiplication and division are quite simple in this system needing only addition or subtraction of the given converted inputs whereas simple operations like addition, subtraction cannot be done easily. Thus in applications where frequent additions or subtractions are not required, these may be of utility. The inverse mapping from LNS to linear numbers is given as

$$X = (1 - z)(-1)^s b^x \tag{1.3b}$$

Note that the addition operation in conventional binary system $(X + Y)$ is computed in LNS noting that $X = b^x$ and $Y = b^y$ as

$$z = x + \log_b(1 + b^{y-x}) \tag{1.4a}$$

The subtraction operation $(X - Y)$ is performed as

$$z = x + \log_b(1 - b^{y-x}) \tag{1.4b}$$

The second term is obtained using an LUT whose size can be very large for $n \geq 20$ [3, 6, 7]. The multiplication, division, exponentiation and finding nth root are very simple. After the processing, the results need to be converted into binary number system.

The logarithmic system can be seen to be a special case of floating-point system where the significand (mantissa) is always 1. Hence the exponent can be a mixed number than an integer. Numbers with the same exponent are equally spaced in floating-point whereas in sign logarithm system, smaller numbers are denser [3].

LNS reduces the strength of certain arithmetic operations and the bit activity [5, 8, 9]. The reduction of strength reduces the switching capacitance. The change of base from 2 to a lesser value reduces the probability of a transition from low to high. It has been found that about two times reduction in power dissipation is possible for operations with word size 8–14 bits.

The other system that has been considered is Residue Number system [10–12] which has received considerable attention in the past few decades. We consider this topic in great detail in the next few chapters. We, however, present here a historical review on this area. The origin is attributed to the third century Chinese author Sun

Tzu (also attributed to Sun Tsu in the first century AD) in the book Suan-Ching. We reproduce the poem [11]:

We have things of which we do not know the number
If we count them by threes, the remainder is 2
If we count them by fives, the remainder is 3
If we count them by sevens, the remainder is 2
How many things are there?
The answer, 23.

Sun Tzu in First Century AD and Greek Mathematicians Nichomachus and Hsin-Tai-Wei of Ming Dynasty (1368AD-1643AD) were the first to explore Residue Number Systems. Sun Tzu has presented the formula for computing the answer which came to be known later as Chinese Remainder Theorem (CRT). This is described by Gauss in his book *Disquisitiones Arithmeticae* [12].

Interestingly, Aryabhata, an Indian mathematician in fifth century A.D., has described a technique of finding the number corresponding to two given residues corresponding to two moduli. This was named as *Aryabhata Remainder Theorem* [13–16] and is known by the Sanskrit name *Saagra-kuttaakaara* (*residual pulveriser*) which is the well-known *Mixed Radix conversion* for two moduli RNS. Extension to moduli sets with common factors has been recently described [17].

In an RNS using mutually prime integers m_1, m_2, m_3, ..., m_j as moduli, the dynamic range M is the product of the moduli, $M = m_1 \cdot m_2 \cdot m_3 \ldots m_j$. The numbers between 0 and $M-1$ can be uniquely represented by the residues. Alternatively, numbers between $-M/2$ to $\left(\frac{M}{2} - 1\right)$ when M is even and $-\left(\frac{M-1}{2}\right)$ to $\left(\frac{M-1}{2}\right)$ when M is odd can be represented. A large number can thus be represented by several smaller numbers called *residues* obtained as the remainders when the given number is divided by the moduli. Thus, instead of big word length operations, we can perform several small word length operations on these residues. The modulo addition, modulo subtraction and modulo multiplication operations can thus be performed quite efficiently.

As an illustration, using the moduli set {3, 5, 7}, any number between 0 and 104 can be uniquely represented by the residues. The number 52 corresponds to the residue set (1, 2, 3) in this moduli set. The residue is the remainder obtained by the division operation X/m_i. Evidently, the residues r_i are such that $0 \leq r_i \leq (m_i - 1)$.

The front-end of an RNS-based processor (see Figure 1.1) is a binary to RNS converter known as *forward converter* whose k output words corresponding to k moduli m_k will be processed by k parallel processors in the Residue Processor blocks to yield k output words. The last stage in the RNS-based processor converts these k words to a conventional binary number. This process known as *reverse conversion* is very important and needs to be hardware-efficient and time-efficient, since it may be often needed also to perform functions such as comparison, sign detection and scaling. The various RNS processors need smaller word length and hence the multiplication, addition and multiplications can be done faster. Of course, these are all modulo operations. The modulo processors do not have any

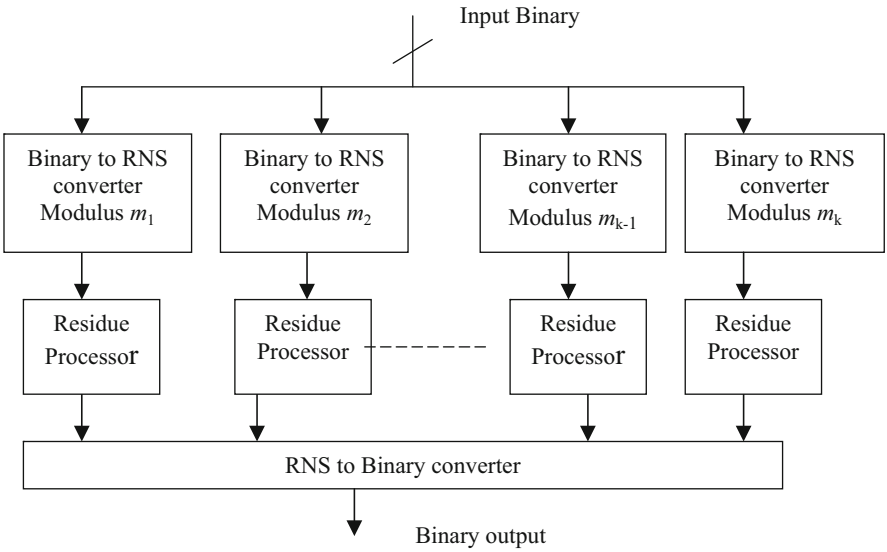

Figure 1.1 A typical RNS-based processor

inter-dependency and hence speed can be achieved for performing operations such as convolution, FIR filtering, and IIR filtering (not needing in-between scaling). The division or scaling by an arbitrary number, sign detection, and comparison are of course time-consuming in residue number systems.

Each MRS digit or RNS modulus can be represented in several ways: binary ($\lceil \log_2 M_j \rceil$ wires with binary logic), index ($\lceil \log_2 M_j \rceil$ wires with binary logic), one-hot (M_j wires with two-valued logic) [18] and M_j-ary (one wire with multi-valued logic). Binary representation is most compact in storage, but one-hot coding allows faster logic and lower power consumption. In addition to electronics, optical and quantum RNS implementations have been suggested [19, 20].

The first two books on Residue number systems appeared in 1967 [21, 22]. Several attempts have been made to build digital computers and other hardware using Residue number Systems. Fundamental work on topics like Error correction has been performed in early seventies. However, there was renewed interest in applying RNS to DSP applications in 1977. An IEEE press book collection of papers [23] focused on this area in 1986 documenting key papers in this area. There was resurgence in 1988 regarding use of special moduli sets. Since then the research interest has increased and a book appeared in 2002 [24] and another in 2007 [25]. Several topics have been addressed such as Binary to Residue conversion, Residue to binary conversion, scaling, sign detection, modulo multiplication, overflow detection, and basic operations such as addition. Since four decades, designers have been exploring the use of RNS to various applications in communication systems, such as Digital signal Processing with emphasis on low power, low area and programmability. Special RNS such as Quadratic RNS and polynomial RNS have been studied with a view to reduce computational requirements in filtering.

More recently, it is very interesting that the power of RNS has been explored to solve problems in cryptography involving very large integers of bit lengths varying from 160 bits to 4096 bits. Attempts also have been made to combine RNS with logarithmic number system known as Logarithmic RNS.

The organization of the book is as follows. In Chapter 2, the topic of modulo addition and subtraction is considered for general moduli as well powers-of-two related moduli. Several advances made in designing hardware using diminished-1arithmetic are discussed. The topic of forward conversion is considered in Chapter 3 in detail for general as well as special moduli. These use several interesting properties of residues of powers of two of the moduli. New techniques for sharing hardware for multiple moduli are also considered. In Chapter 4, modulo multiplication and modulo squaring using Booth-recoding and not using Booth-recoding is described for general moduli as well moduli of the type 2^n-1 and especially 2^n+1. Both the diminished-1 and normal representations are considered for design of multipliers mod (2^n+1). Multi-modulus architectures are also considered to share the hardware amongst various moduli. In Chapter 5, the well-investigated topic of reverse conversion for three, four, five and more number of moduli is considered. Several recently described techniques using Core function, quotient function, Mixed-Radix CRT, New CRTs, and diagonal function have been considered in addition to the well-known Mixed Radix Conversion and CRT. Area and time requirements are highlighted to serve as benchmarks for evaluating future designs. In Chapter 6, the important topics of scaling, base extension, magnitude comparison and sign detection are considered. The use of core function for scaling is also described.

In Chapter 7, we consider specialized Residue number systems such as Quadratic Residue Number systems (QRNS) and its variations. Polynomial Residue number systems and Logarithmic Residue Number systems are also considered. The topic of error detection, correction and fault tolerance has been discussed in Chapter 8. In Chapter 9, we deal with applications of RNS to FIR and IIR Filter design, communication systems, frequency synthesis, DFT and 1-D and 2-D DCT in detail. This chapter highlights the tremendous attention paid by researchers to numerous applications including CDMA, Frequency hopping, etc. Fault tolerance techniques applicable for FIR filters are also described. In Chapter 10, we cover extensively applications of RNS in cryptography perhaps for the first time in any book. Modulo multiplication and exponentiation using various techniques, modulo reduction techniques, multiplication of large operands, application to ECC and pairing protocols are covered extensively. Extensive bibliography and examples are provided in each chapter.

References

1. M.G. Arnold, The residue logarithmic number system: Theory and application, in *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH)*, Cape Cod, 27–29 June 2005, pp. 196–205
2. E.C. Ifeachor, B.W. Jervis, *Digital Signal Processing: A Practical Approach*, 2nd edn. (Pearson Education, Harlow, 2003)

3. I. Koren, *Computer Arithmetic Algorithms* (Brookside Court, Amherst, 1998)
4. S.W. Smith, *The Scientists's and Engine's Guide to Digital Signal Processing* (California Technical, San Diego, 1997). Analog Devices
5. T. Stouraitis, V. Paliouras, Considering the alternatives in low power design. IEEE Circuits Devic. **17**(4), 23–29 (2001)
6. F.J. Taylor, A 20 bit logarithmic number system processor. IEEE Trans. Comput. **C-37**, 190–199 (1988)
7. L.K. Yu, D.M. Lewis, A 30-bit integrated logarithmic number system processor. IEEE J. Solid State Circuits **26**, 1433–1440 (1991)
8. J.R. Sacha, M.J. Irwin, The logarithmic number system for strength reduction in adaptive filtering, in *Proceedings of the International Symposium on Low-power Electronics and Design (ISLPED98)*, Monterey, 10–12 Aug. 1998, pp. 256–261
9. V. Paliouras, T. Stouraitis, Low power properties of the logarithmic number system, in *15th IEEE Symposium on Computer Arithmetic*, Vail, 11–13 June 2001, pp. 229–236
10. H. Garner, The residue number system. IRE Trans. Electron. Comput. **8**, 140–147 (1959)
11. F.J. Taylor, Residue arithmetic: A tutorial with examples. IEEE Computer **17**, 50–62 (1984)
12. C.F. Gauss, *Disquisitiones Arithmeticae* (1801, English translation by Arthur A. Clarke). (Springer, New York, 1986)
13. S. Kak, Computational aspects of the Aryabhata algorithm. Indian J. Hist. Sci. **211**, 62–71 (1986)
14. W.E. Clark, *The Aryahbatiya of Aryabhata* (University of Chicago Press, Chicago, 1930)
15. K.S. Shulka, K.V. Sarma, *Aryabhateeya of Aryabhata* (Indian National Science Academy, New Delhi, 1980)
16. T.R.N. Rao, C.-H. Yang, Aryabhata remainder theorem: Relevance to public-key Crypto-algorithms. Circuits Syst. and Signal. Process. **25**(1), 1–15 (2006)
17. J.H. Yang, C.C. Chang, Aryabhata remainder theorem for Moduli with common factors and its application to information protection systems, in *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Harbin, 15–17 Aug. 2008, pp. 1379–1382
18. W.A. Chren, One-hot residue coding for low delay-power product CMOS designs. IEEE Trans. Circuits Syst. **45**, 303–313 (1998)
19. Q. Ke, M.J. Feldman, Single flux quantum circuits using the residue number system. IEEE Trans. Appl. Supercond. **5**, 2988–2991 (1995)
20. C.D. Capps et al., Optical arithmetic/logic unit based on residue arithmetic and symbolic substitution. Appl. Opt. **27**, 1682–1686 (1988)
21. N. Szabo, R. Tanaka, *Residue Arithmetic and Its Applications in Computer Technology* (McGraw Hill, New York, 1967)
22. R.W. Watson, C.W. Hastings, *Residue Arithmetic and Reliable Computer Design* (Spartan, Washington, DC, 1967)
23. M.A. Soderstrand, G.A. Jullien, W.K. Jenkins, F. Taylor (eds.), *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing* (IEEE Press, New York, 1986)
24. P.V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures* (Kluwer, Boston, 2002)
25. A.R. Omondi, B. Premkumar, *Residue Number Systems: Theory and Implementation* (Imperial College Press, London, 2007)

Chapter 2
Modulo Addition and Subtraction

In this Chapter, the basic operations of modulo addition and subtraction are considered. Both the cases of general moduli and specific moduli of the form 2^n-1 and 2^n+1 are considered in detail. The case with moduli of the form 2^n+1 can benefit from the use of diminished-1 arithmetic. Multi-operand modulo addition also is discussed.

2.1 Adders for General Moduli

The modulo addition of two operands A and B can be implemented using the architectures of Figure 2.1a and b [1, 2]. Essentially, first $A+B$ is computed and then m is subtracted from the result to find whether the result is larger than m or not. (Note that TC stands for two's complement.) Then using a 2:1 multiplexer, either $(A+B)$ or $(A+B-m)$ is selected. Thus, the computation time is that of one n-bit addition, one $(n+1)$-bit addition and delay of a multiplexer. On the other hand, in the architecture of Figure 2.2b, both $(A+B)$ and $(A+B-m)$ are computed in parallel and one of the outputs is selected using a 2:1 multiplexer depending on the sign of $(A+B-m)$. Note that a carry-save adder (CSA) stage is needed for computing $(A+B-m)$ which is followed by a carry propagate adder (CPA). Thus, the area is more than that of Figure 2.2a, but the addition time is less. The area A and computation time Δ for both the techniques can be found for n-bit operands assuming that a CPA is used as

P.V. Ananda Mohan, *Residue Number Systems*, DOI 10.1007/978-3-319-41385-3_2

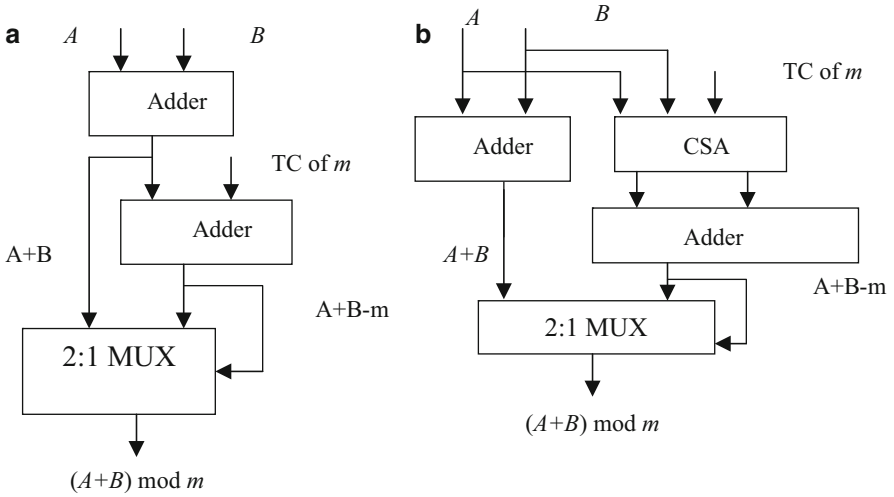

Figure 2.1 Modulo adder architectures: (**a**) sequential (**b**) parallel

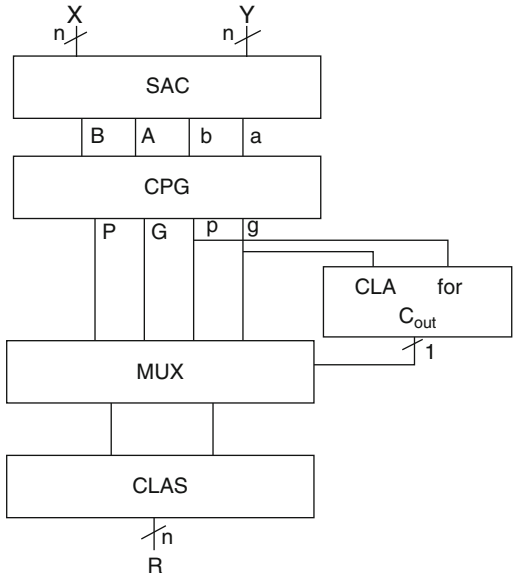

Figure 2.2 Modular adder due to Hiasat (adapted from [6] ©IEEE2002)

$$\begin{aligned} A_{cascade} &= (2n+1)A_{FA} + nA_{2:1MUX} + nA_{INV}, \quad \Delta_{cascade} = (2n+1)\Delta_{FA} + \Delta_{2:1MUX} + \Delta_{INV} \\ A_{Parallel} &= (3n+2)A_{FA} + nA_{2:1MUX} + nA_{INV}, \quad \Delta_{parallel} = (n+2)\Delta_{FA} + \Delta_{2:1MUX} + \Delta_{INV} \end{aligned} \tag{2.1}$$

where Δ_{FA}, $\Delta_{2:1MUX}$, and Δ_{INV} are the delays and A_{FA}, $A_{2:1MUX}$ and A_{INV} are the areas of a full-adder, 2:1 Multiplexer and an inverter, respectively. On the other

hand, by using VLSI adders with regular layout e.g. Brent−Kung adder [3], the area and delay requirements will be as follows:

$$\begin{aligned} &A_{cascade} = 2n(\log_2 n + 1)A_{FA} + nA_{2:1MUX} + nA_{INV}, \quad \Delta_{cascade} = 2(\log_2 n + 1)\Delta_{FA} + \Delta_{INV}, \\ &A_{Parallel} = (n + 1 + \log_2 n + \log_2(n + 1) + 2)A_{FA} + nA_{2:1MUX} + nA_{INV}, \\ &\Delta_{parallel} = ((\log_2 n + 1) + 2)\Delta_{FA} + \Delta_{2:1MUX} + \Delta_{INV} \end{aligned} \tag{2.2}$$

Subtraction is similar to the addition operation wherein $(A-B)$ and $(A-B+m)$ are computed sequentially or in parallel following architectures similar to Figure 2.1a and b.

Multi-operand modulo addition has been considered by several authors. Alia and Martinelli [4] have suggested the mod m addition of several operands using a CSA tree trying to keep the partial results at the output of each CSA stage within the range $(0, 2^n)$ by adding a proper value. The three-input addition in a CSA yields n-bit sum and carry vectors S and C. S is always in the range $\{0, 2^n\}$. The computation of $(2C+S)_m$ is carried out as $(2C+S)_m = L+H+2T_C+T_S = L+H+T+km$ where $k>0$ is an integer. Note that $L = 2(C-T_C)$ and $H = S-T_S$ were $T_S = s_{n-1}2^{n-1}$ and $T_C = c_{n-1}2^{n-1} + c_{n-2}2^{n-2}$. Thus, using s_{n-1}, c_{n-1}, c_{n-2} bits, T can be obtained using a 7:1 MUX and added to L, H. Note that L is obtained from C by one bit left shift and H is obtained as $(n-1)$-bit LSB word of S.

All the operands can be added using a CSA tree and the final result $U_F = 2C_F + S_F$ is reduced using a modular reduction unit which finds U_F, U_F-m, $U_F-2\,m$ and $U_F-3\,m$ using two CLAs and based on the sign bits of the last three words, one of the answers is selected.

Elleithi and Bayoumi [5] have presented a $\theta(1)$ algorithm for multi-operand modulo addition which needs a constant time of five steps. In this technique, the two operands A and B are written in redundant form as A_1, A_2 and B_1, B_2, respectively. The first three are added in a CSA stage which will yield sum and carry vectors. These two vectors temp1 and temp2 and B_2 are added in another CSA which will yield sum and carry vectors temp3 and temp4. In the third step, to temp3 and temp4 vectors, a correction term (2^n-m) or $2(2^n-m)$ is added in another CSA stage depending on either one or both carry bits of temp1 and temp2 are 1 to result in the sum and carry vectors temp5 and temp6. Depending on the carry bit, in the next step (2^n-m) is added to yield final result in carry save form as temp7 and temp8. There will be no overflow thereafter.

Hiasat [6] has described a modulo adder architecture based on a CSA and multiplexing the carry generate and propagate signals before being driven to the carry computation unit. In this design, the output carry is predicted that could result from computation of $A+B+Z$ where $Z = 2^n-m$. If the predicted carry is 1, an adder proceeds in computing the sum $A+B+Z$. Otherwise, it computes the sum $A+B$. Note that the calculation of Sum and Carry bits in case of bit z_i being 1 or 0 is quite simple as can be seen for both these cases:

$$s_i = a_i \oplus b_i, \quad c_{i+1} = a_i b_i \quad \text{and} \quad \hat{s}_i = \overline{a_i \oplus b_i}, \quad \hat{c}_{i+1} = a_i + b_i$$

Thus, half-adder like cells which give both the outputs are used. Note that s_i, c_{i+1}, $\hat{s}_i$, $\hat{c}_{i+1}$ serve as inputs to carry propagate and generate unit which has outputs P_i, G_i, p_i, g_i corresponding to both the cases. Based on the computation of c_{out} using a CLA, a multiplexer is used to select one of these pairs to compute all the carries and the final sum. The block diagram of this adder is shown in Figure 2.2 where SAC is sum and carry unit, CPG is carry propagate generate unit, and CLA is carry look ahead unit for computing C_{out}. Then using a MUX, either P, G or p, g are selected to be added using CLA summation unit (CLAS). The CLAS unit computes all the carries and performs the summation $P_i \oplus c_i$ to produce the output R. This design leads to lower area and delay than the designs in Refs. [1, 5].

Adders for moduli (2^n-1) and (2^n+1) have received considerable attention in literature which will be considered next.

2.2 Modulo (2^n-1) Adders

Efstathiou, Nikolos and Kalamatinos [7] have described a mod (2^n-1) adder. In this design, the carry that results from addition assuming carry input is zero is taken into account in reformulating the equations to compute the sum. Consider a mod 7 adder with inputs A and B. With the usual definition of generate and propagate signals, it can be easily seen that for a conventional adder we have

$$c_0 = G_0 + P_0 c_{-1} \tag{2.3a}$$

$$c_1 = G_1 + P_1 c_0 \tag{2.3b}$$

$$c_2 = G_2 + P_2 G_1 + P_2 P_1 g_0 \tag{2.3c}$$

Substituting c_{-1} in (2.3a) with c_2 due to the end-around carry operation of a mod (2^n-1) adder, we have

$$c_0 = G_0 + P_0 G_2 + P_0 P_2 G_1 + G_0 P_2 P_1 G_0 = G_0 + P_0 G_2 + P_0 P_2 G_1 \tag{2.4}$$

$$c_1 = G_1 + P_1 G_0 + P_1 P_0 G_2 \tag{2.5a}$$

$$c_2 = G_2 + P_2 G_1 + P_2 P_1 G_o \tag{2.5b}$$

An implementation of mod 7 adder with double representation of zero (i.e. output = 7 or zero) is shown in Figure 2.3a where $s_i = P_i \oplus c_{i-1}$. A simple modification can be carried out as shown in Figure 2.3b to realize a single zero. Note that the output can be 2^n-1, if both the inputs are complements of each other. Hence, this condition can be used by computing $P = P_0 P_1 P_2 \ldots P_{n-1}$ and modifying the equations as

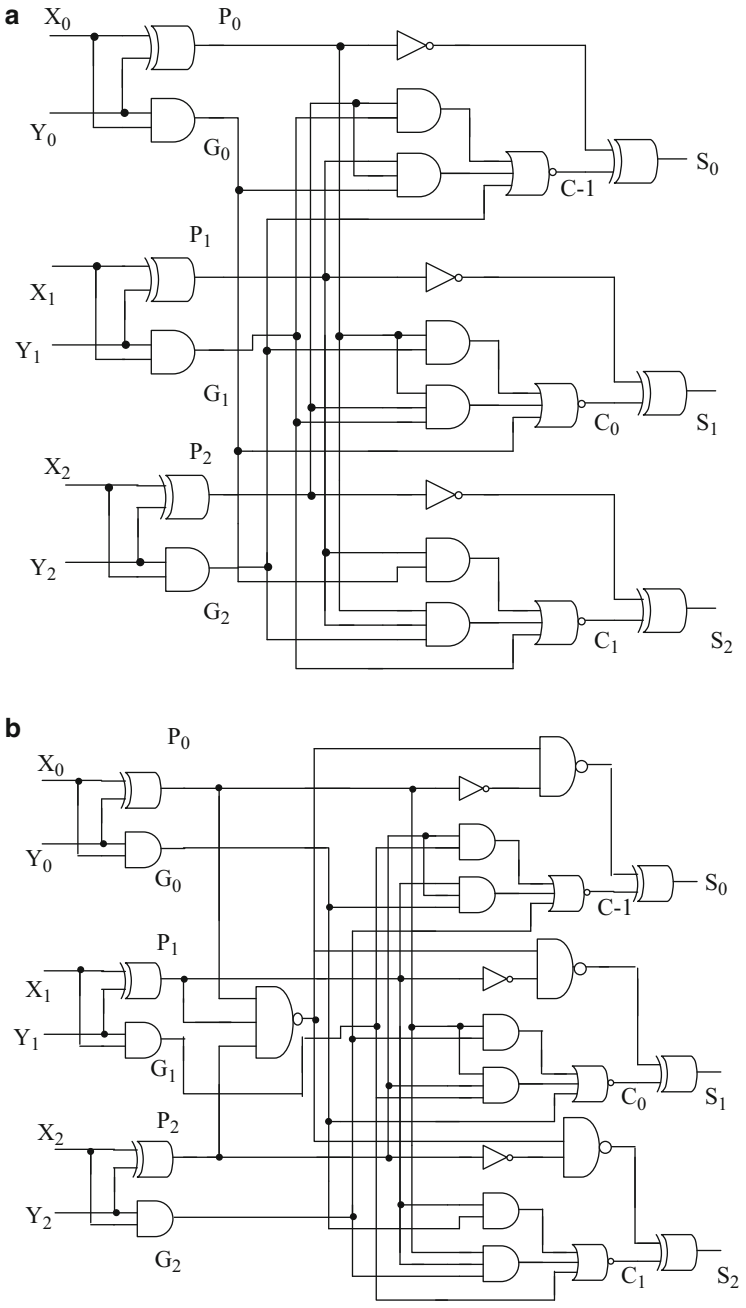

Figure 2.3 (**a**) Mod 7 adder with double representation of zero (**b**) with single representation of zero (adapted from [7] ©IEEE1994)

$$s_i = \overline{(P_i + P)} \oplus c_{i-1} \quad \text{for } 0 \leq i \leq n-1. \tag{2.6}$$

The architectures of Figure 2.3, although they are elegant, they lack regularity. Instead of using single level CLA, when the operands are large, multiple levels can also be used.

Another approach is to consider the carry propagation in binary addition as a prefix problem. Various types of parallel-prefix adders e.g. (a) Ladner−Fischer [8], (b) Kogge-Stone [9], (c) Brent−Kung [3] and (d) Knowles [10] are available in literature. Among these, type (a) requires less area but has unlimited fan out compared to type (b). But designs based on (b) are faster.

Zimmerman [11] has suggested using an additional level for adding end-around-carry for realizing a mod (2^n-1) adder (see Figure 2.4a) which needs extra hardware and more over, this carry has a large fan out thus making it slower. Kalampoukas et al. [12] have considered modulo (2^n-1) adders using parallel-prefix adders. The idea of carry recirculation at each prefix level as shown in Figure 2.4b has been employed. Here, no extra level of adders will be required, thus having minimum logic depth. In addition, the fan out requirement of the carry output is also removed. These architectures are very fast while consuming large area.

The area and delay requirements of adders can be estimated using the unit-gate model [13]. In this model, all gates are considered as a unit, whereas only exclusive-OR gate counts for two elementary gates. The model, however, ignores fan-in and fan-out. Hence, validation needs to be carried out by using static simulations. The area and delay requirements of mod (2^n-1) adder described in [12] are $3n\log n + 4n$ and $2\log n + 3$ assuming this model.

Efstathiou et al. [14] have also considered design using select-prefix blocks with the difference that the adder is divided into several small length adder blocks by proper interconnection of propagate and generate signals of the blocks. A select-prefix architecture for mod (2^n-1) adder is presented in Figure 2.5. Note that d, f and g indicate the word lengths of the three sections. It can be seen that

$$\begin{aligned} c_{in,0} &= BG_2 + BP_2BG_1 + BP_2BP_1BG_0 \\ c_{in,1} &= c_{out,0} = BG_0 + BP_0BG_2 + BP_0BP_2BG_1 \\ c_{in,2} &= c_{out,1} = BG_1 + BP_1BG_0 + BP_1BP_0BG_2 \end{aligned}$$

where BG_i and BP_i are block generate and propagate signals outputs of each block.

Tyagi [13] has given an algorithm for selecting the lengths of the various adder blocks suitably with the aim of minimization of adder delay. Note that designs based on parallel-prefix adders are fastest but are more complex. On the other hand, CLA-based adder architecture is area effective. Select prefix-architectures achieve delay closer to parallel prefix adders and have complexity close to the best adders.

Patel et al. [15] have suggested fast parallel-prefix architectures for modulo (2^n-1) addition with a single representation of zero. In these, the sum is computed with a carry in of "1". Later, a conditional decrement operation is

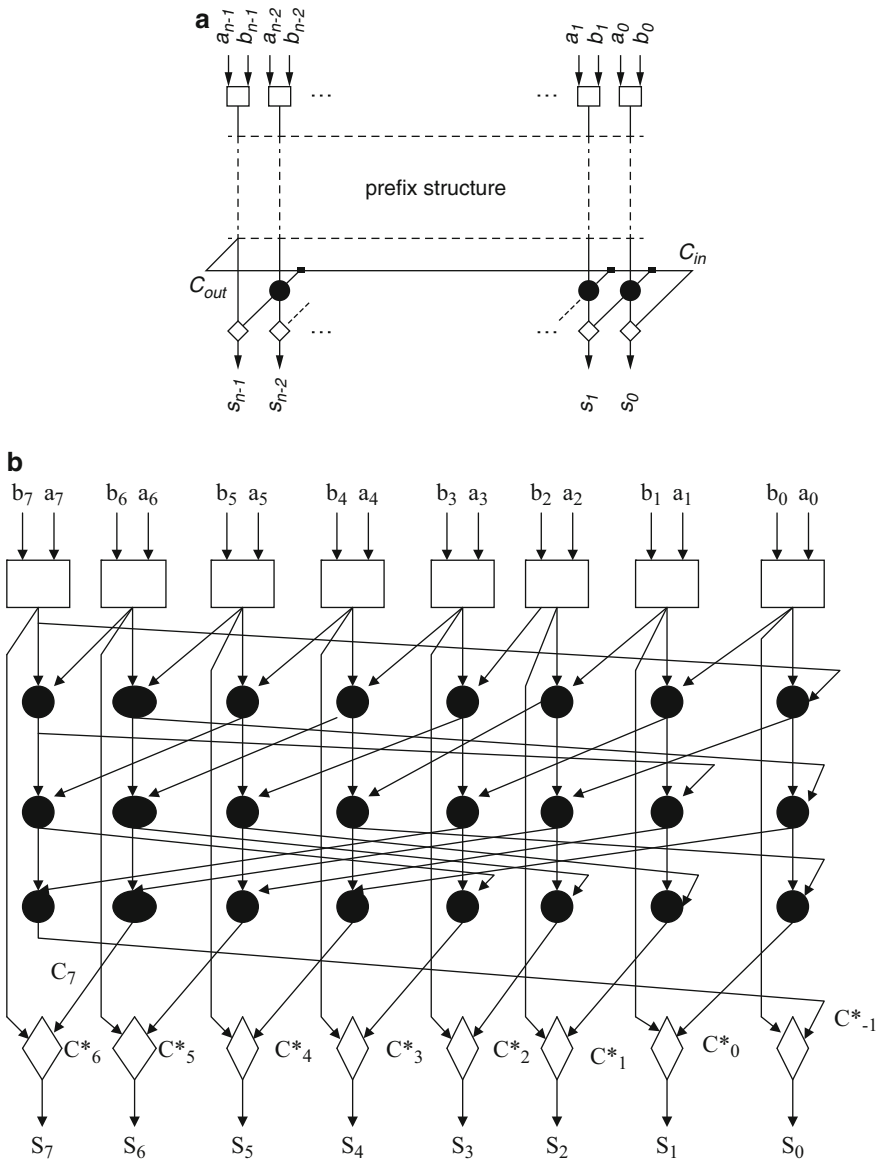

Figure 2.4 Modulo (2^n-1) adder architectures due to (**a**) Zimmermann and (**b**) modulo (2^8-1) adder due to Kalampoukas et al. ((a) adapted from [11] ©IEEE1999 and (b) adapted from [12] ©IEEE2000)

performed. However, by cyclically feeding back the carry generate and carry propagate signals at each prefix level in the adder, the authors show that significant improvement in latency is possible over existing designs.

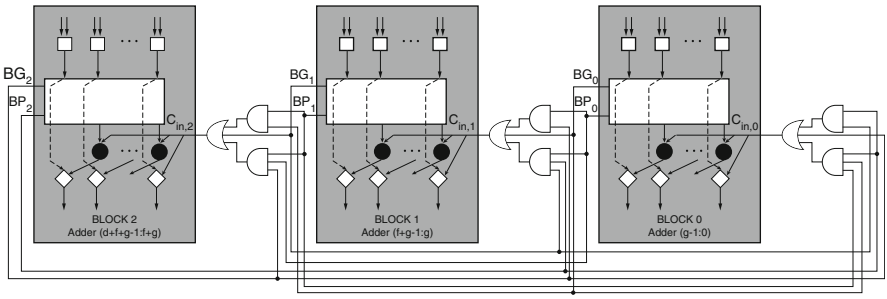

Figure 2.5 Modulo $2^{d+f+g}-1$ adder design using three blocks (adapted from [14] ©IEEE2003)

2.3 Modulo (2^n+1) Adders

Diminished-1 arithmetic is important for handling moduli of the form 2^n+1. This is because of the reason that this modulus channel needs one bit more word length than other channels using moduli 2^n and 2^n-1. A solution given by Liebowitz [16] is to represent the numbers still by n bits only. The diminished-1 number corresponding to normal number A in the range 1 to 2^n is represented as $d(A)=A-1$. If $A=0$, a separate channel with one bit which is 1 is used. Another way of representing A in diminished-1 arithmetic is (A_z, A_d) where $A_z=1$, $A_d=0$ when $A=2^n$, $A_z=0$, $A_d=A-1$ otherwise. Due to this representation, some rules need to be built to perform operations in this arithmetic which are summarized below. Following the above notation, we can derive the following properties [17]:
(a) $A+B=C$ corresponds to

$$d(A+B) \;=\; (d(A)+d(B)+1) \bmod (2^n+1) \tag{2.7}$$

(b) Similarly, we have

$$d(A-B) = \Big(d(A)+\overline{d(B)}+1\Big) \bmod (2^n+1) \tag{2.8}$$

(c) It follows further that

$$d\Big(\sum\nolimits_{k=1}^{n} A_k\Big) = (d(A_1)+d(A_2)+d(A_3)+ \quad \ldots \quad +d(A_k)+n-1) \;\bmod\; (2^n+1) \tag{2.9}$$

Next,

$$d\big(2^k A\big) \;= d(A+A+A+\ldots+A) \;=\; \big(2^k d(A)+2^k-1\big) \bmod (2^n+1).$$

or

$$2^k d(A) \;=\; \big(d\big(2^k A\big)-2^k+1\big) \bmod (2^n+1) \tag{2.10}$$

In order to simplify the notation, we denote a diminished-1 number using an asterisk e.g. $d(A) = A^* = A-1$.

Several mod (2^n+1) adders have been proposed in literature. In the case of diminished-1 numbers, mod (2^n+1) addition can be formulated as [11]

$$\begin{aligned} S-1 = S^* &= (A^*+B^*+1) \mod (2^n+1) \\ &= (A^*+B^*) \mod (2^n) \quad \text{if} \quad (A^*+B^*) \\ &\geq 2^n \quad \text{and} \quad (A^*+B^*+1) \quad \text{otherwise} \end{aligned} \tag{2.11}$$

where A^* and B^* are diminished-1 numbers and $S = A+B$. The addition of 1 can be carried out by inverting the carry bit C_{out} and adding in a parallel-prefix adder with $C_{in} = \overline{C_{out}}$ (see Figure 2.6):

$$(A^*+B^*+1) \mod (2^n+1) = \left(A^*+B^*+\overline{C_{out}}\right) \mod (2^n) \tag{2.12}$$

In the case of normal numbers as well [11], we have

$$S+1 = (A+B+1) \mod (2^n+1) = \left(A+B+\overline{C_{out}}\right) \mod (2^n) \tag{2.13}$$

where $S = A+B$ with the property that $(S+1)$ is computed. In the design of multipliers, this technique will be useful.

Note that diminished-1 adders have a problem of correctly interpreting the zero output since it may represent a valid zero (addition with a result of 1) or a real zero output (addition with a result zero) [14]. Consider the two examples of modulo

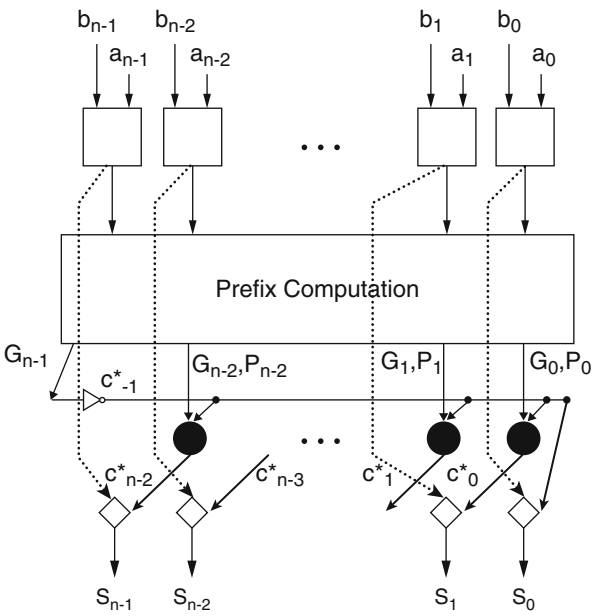

Figure 2.6 Modulo (2^n+1) adder architecture for diminished-1 arithmetic (adapted from [18] ©IEEE2002)

9 addition (a) $A=6$ and $B=4$ and (b) $C=5$ and $B=4$ using diminished-1 representation:

A^*	101	C^*	100
B^*	011	B^*	011
C_{out}	1 000	C_{out}	0 111
$\overline{C_{out}}$	0	$\overline{C_{out}}$	1
	000 Correct result		000 result indicating zero

Note that real zero occurs when the inputs are complimentary. Hence, this condition needs to be detected using logical AND of the exclusive-OR of a_i and b_i. The EXOR gates will be already present in the front-end CSA stage.

Vergos, Efstathiou and Nikolos have presented two mod (2^n+1) adder architectures [18] for diminished-1 numbers. The first one leads to CLA implementation and was derived by associating the re-entering carry equation with those producing the carries of the modulo addition similar to that for mod (2^n-1) described earlier [12]. In this architecture, both one and two level CLAs have been considered. The second architecture uses parallel-prefix adders and also was derived by re-circulation of the carries in each level of parallel-prefix structure. This architecture avoids the problem of fan-out and the additional level needed in Zimmerman's technique shown in Figure 2.6.

Efstathiou, Vergos and Nikolos [14] extended the above ideas by using select-prefix blocks which are faster than the previous ones for designing mod $(2^n \pm 1)$ adders for diminished-1 operands. Here, the lengths of the blocks can be selected appropriately as well as the number of the blocks. The derivation is similar to that for mod (2^n-1) adders with the difference that the equations contain block carry propagate, and block generate signals instead of bit level propagate and generate signals. In these, an additional level is used to add the carry after the prefix computation. A structure using two stages is presented in Figure 2.7. Note that in this case

$$c_{in,0} = (BG_1 + BP_1BG_0)'$$
$$c_{in,1} = c_{out,0} = BG_0 + BP_0BG'_1$$

These designs need lesser area than designs using parallel-prefix adders while they are slower than CLA-based designs.

Efstathiou, Vergos and Nikolos [19] have described fast parallel-prefix modulo (2^n+1) adders for two $(n+1)$-bit numbers which use two stages. The first stage computes $|X+Y+2^n-1|_{2^{n+1}}$ which has $(n+2)$ bits. If MSB of the result is zero, then 2^n+1 is added mod 2^{n+1} and the n LSBs yield the result. For computing $M = X+Y+2^n-1$, a CSA is used followed by a $(n+1)$-bit adder. The authors use parallel-prefix with fast carry increment (PPFCI) architecture and also a totally

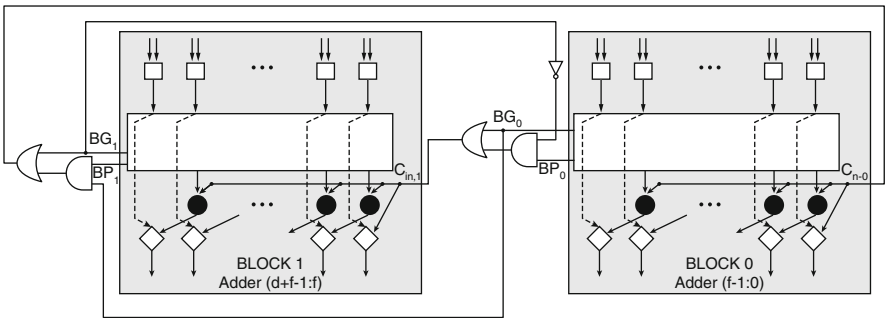

Figure 2.7 Diminished-1 modulo $(2^{d+f}+1)$ adder using two blocks (adapted from [14] ©IEEE2004)

parallel-prefix architecture. In the former, an additional stage for re-entering carry is used, whereas in the latter case, carry recirculation is done at every prefix level.

The architecture of Hiasat [6] can be extended to the case of modulus (2^n+1) in which case we have $Z=2^n-1$ and the formulae used are as follows:

$$R=|X+Y+Z|_{2^n} \quad \text{if } X+Y+Z\geq 2^{n+1} \quad \text{and} \quad R=|X+Y+Z|_{2^n}+1 \quad \text{otherwise.}$$

Note that, in this case, the added bit z_i is always 1 in all bit positions.

Vergos and Efstathiou [20] proposed an adder that caters for both weighted and diminished-1 operands. They point out that a diminished-1 adder can be used to realize a weighted adder by having a front-end inverted EAC CSA stage. Herein, $A+B$ is computed where A and B are $(n+1)$-bit numbers using a diminished-1 adder. In this design, the computation carried out is

$$|A+B|_{2^n+1}=\left||A_n+B_n+D+1|_{2^n+1}+1\right|_{2^n+1}=|Y+U+1|_{2^n+1} \tag{2.14}$$

where Y and U are the sum and carry vector outputs of a CSA stage computing A_n+B_n+D:

$$\begin{aligned} carry \quad & Y=y_{n-2}y_{n-3}.....y_o\overline{y_{n-1}} \\ sum \quad & U=u_{n-1}u_{n-2}.....u_1u_o \end{aligned}$$

where $D=2^n-4+2\overline{c_{n+1}}+\overline{s_n}$. Note that A_n, B_n are the words formed by the n-bit LSBs of A and B, respectively, and s_n, c_{n+1} are the sum and carry of addition of 1-bit words a_n and b_n. It may be seen that D is the n-bit vector $11111...1\overline{c_{n+1}s_n}$.

An example will be illustrative. Consider $n=4$ and the addition of $A=16$ and $B=11$. Evidently $a_n=1$, $b_n=0$, $A_n=0$ and $B_n=11$ and $D=01110$ yielding $(16+11)_{17}=((0+11+14+1)_{17}+1)_{17}=10$. Note that the periodic property of residues mod (2^n+1) is used. The sum of the n th bits is complimented and added to get D and a correction term is added to take into account the mod (2^n+1) operation.

The mod $(2^n + 1)$ adder for weighted representation needs a diminished-1 adder and an inverted end-around-carry stage. The full adders of this CSA stage perform $(A_n + B_n + D)$ mod $(2^n + 1)$ addition. Some of the FAs have one input "1" and can thus be simplified. The outputs of this stage Y and U are fed to a diminished-1 adder to obtain $(Y + U + 1)$ mod 2^n. The architecture is presented in Figure 2.8. It can be seen that every diminished-1 adder can be used to perform weighted binary addition using an inverted EAC CSA stage in the front-end.

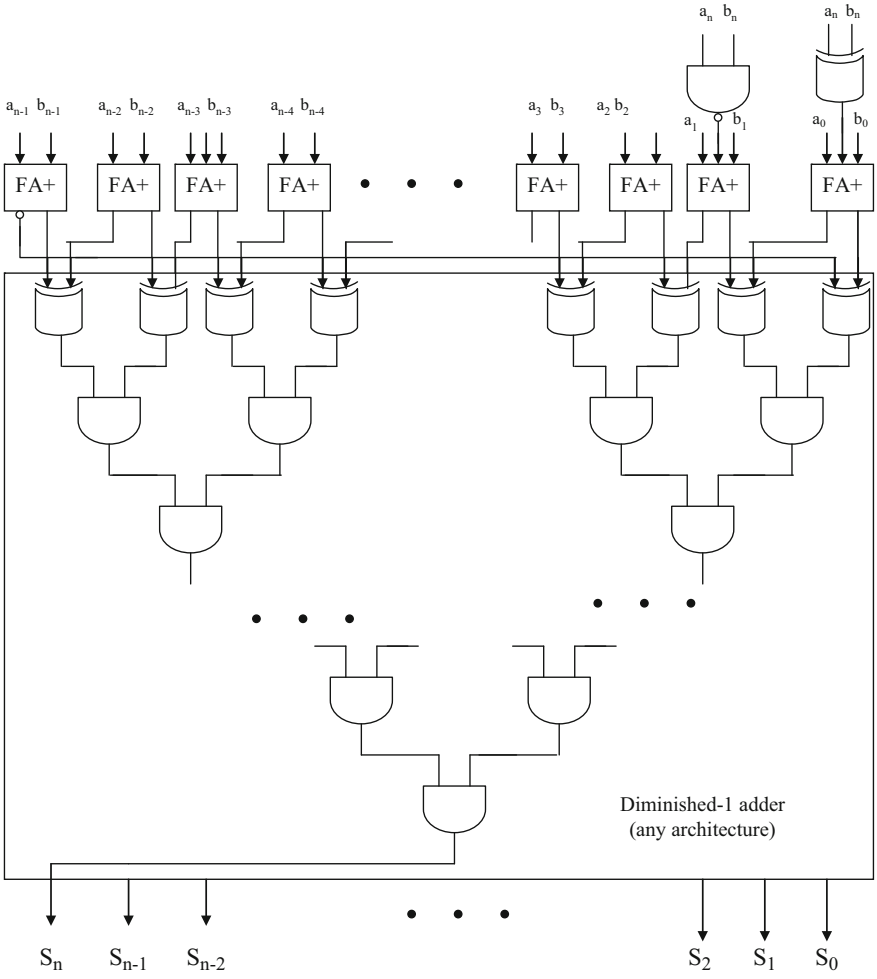

Figure 2.8 Modulo $(2^n + 1)$ adder for weighted operands built using a diminished-1 adder (adapted from [20] ©IEEE2008)

In another technique due to Vergos and Bakalis [21], first A^* and B^* are computed such that $A^*+B^*=A+B-1$ using a translator. Then, a diminished-1 adder can sum A^* and B^* such that

$$\left||A+B|_{2^n+1}\right|_{2^n} = |A^*+B^*|_{2^n} + \overline{c_{out}} \tag{2.15}$$

where c_{out} is the carry of the n-bit adder computing A^*+B^*. However, Vergos and Bakalis do not present the details of obtaining A^* and B^* using the translator. Note that in this method, the inputs are $\leq (2^n-1)$.

Lin and Sheu [22] have suggested the use of two parallel adders to find A^*+B^* and A^*+B^*+1 so that the carry of the former adder can be used to select the correct result using a multiplexer. Note that Lin and Sheu [22] have also suggested partitioning the n-bit circular carry selection (CCS) modular adder to m number of r-bit blocks similar to the select-prefix block type of design considered earlier. These need circular carry selection addition blocks and circular carry generators. Juang et al. [23] have given a corrected version of this type of mod (2^n+1) adder shown in Figure 2.9a and b. Note that this design uses a dual sum carry look ahead adder (DS-CLA). These designs are most efficient among all the mod (2^n+1) adders regarding area, time and power.

Juang et al. [24] have suggested considering $(n+1)$ bits for inputs A and B. The weighted modulo (2^n+1) sum of A and B can be expressed as

$$\begin{aligned}\left||A+B|_{2^n+1}\right|_{2^n} &= |A+B-(2^n+1)|_{2^n} \text{ if } (A+B)>2^n \\ &= |A+B-(2^n+1)|_{2^n}+1 \quad \text{otherwise}\end{aligned} \tag{2.16}$$

Thus, weighted modulo (2^n+1) addition can be obtained by subtracting the sum of A and B by (2^n+1) and using a diminished-1 adder to get the final modulo sum by making the inverted EAC as carry-in.

Denoting Y' and U' as the carry and sum vectors of the summation $A+B-(2^n+1)$, where A and B are $(n+1)$-bit words, we have

$$|A+B-(2^n+1)|_{2^n} = \left|\sum_{i=0}^{n-2}\left(2^i\left(2y_i'+u_i'\right)+2^{n-1}\left(2a_n+2b_n+a_{n-1}+b_{n-1}+1\right)\right)\right|_{2^n} \tag{2.17}$$

where

$$y_i' = a_i \vee b_i, \quad u_i' = \overline{a_i \oplus b_i}.$$

As an illustration, consider $A=16$, $B=15$ and $n=4$. We have

$$|A+B-(2^n+1)|_{2^n} = |16+15-17|_{16} = 14$$

and for $A=6$, $B=7$,

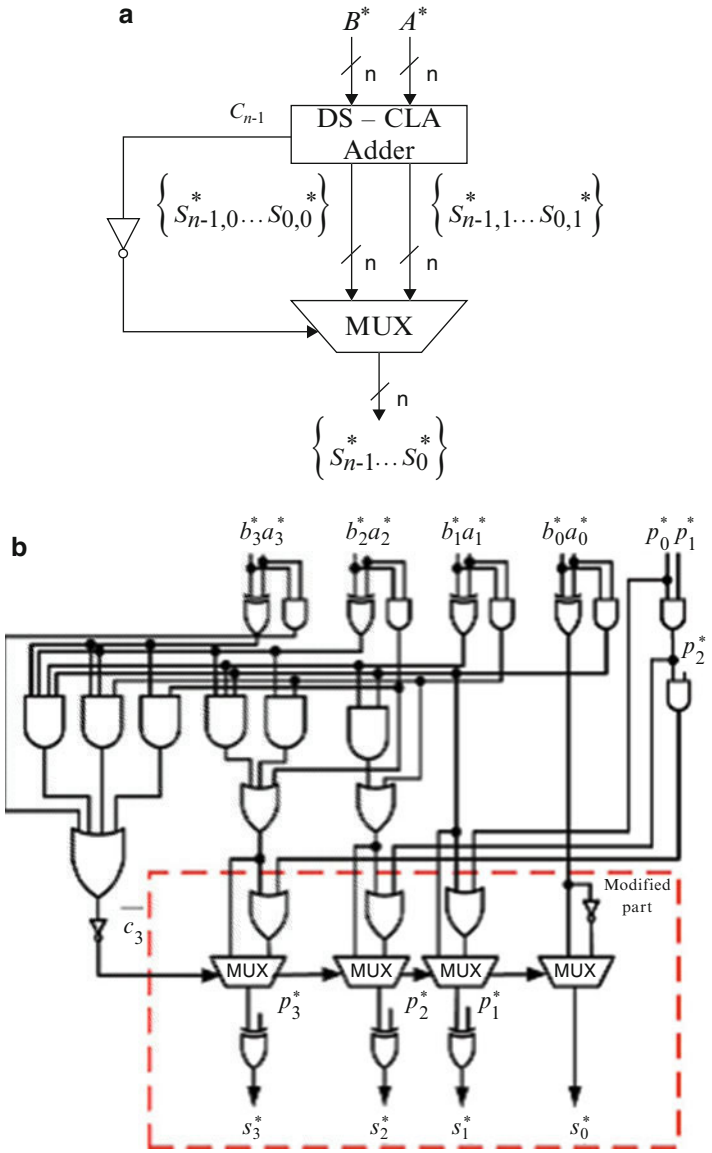

Figure 2.9 (**a**) Block diagram of CCS diminished-1 modulo $(2^n + 1)$ adder and (**b**) Logic circuit of CCS diminished-1 modulo $(2^4 + 1)$ adder ((a) adapted from [22] ©IEEE2008, (b) adapted from [23] ©IEEE2009)