INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction..

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

University of Alberta

DESIGN AND IMPLEMENTATION OF DIGIT-SERIAL ONLINE MULTIPLY-ACCUMULATE ARITHMETIC OPERATIONS

by

William Natter ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Department of Electrical and Computer Engineering

Edmonton, Alberta
spring 2001

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisitions et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-60479-9

Canada

University of Alberta

Library Release Form

Name of Author: William Natter

Title of Thesis: Design and Implementation of Digit-Serial Online Multiply-Accumulate Arithmetic Operations

Degree: Master of Science

Year this Degree Granted: 2001

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

William Natter

William Natter
CEB 238
University of Alberta
Edmonton, AB
Canada, T6G 2G7

Date: January 23, 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Design and Implementation of Digit-Serial Online Multiply-Accumulate Arithmetic Operations** submitted by William Natter in partial fulfillment of the requirements for the degree of **Master of Science**

[illegible] . . . .
Dr. T. Chen (Committee chair)

[illegible] . . . . .
Dr. B. Nowrouzian (Supervisor)

[illegible] . . . .
Dr. W. Pedrycz

L. Stewart . . . .
Dr. L. Stewart

Date: January 26, 2001

To my wife, and both our families.

Abstract

This thesis is concerned with the combination of the online and digit-serial arithmetic techniques for the design, development, and hardware implementation of algorithms for multiplication and multiply-accumulate arithmetic operations. The online technique processes digital signals as generated and consumed by current practical analog-to-digital and digital-to-analog converters. The digit-serial technique permits a trade-off between speed and area in a corresponding hardware implementation, and is extended to dynamically changing wordlengths (with small hardware overhead). Multiplication and multiply-accumulate operations are performed as successive additions of partial operation updates, justifying the use of (redundant) ordinary signed-digit number systems where the addition architecture delays can be made independent of the wordlengths of the inputs. Emphasis is placed on the signed-binary number system, as it is closest to the current practical number systems (signed-magnitude and two's complement). Relationships between number systems are established to subsequently exploit their addition scheme similarities and allow the determination of the corresponding fastest and smallest hardware implementations for the signed-binary and binary carry-save number systems. A generic online algorithm for multiply-accumulate operation is developed so as to allow its modification into an algorithm for inner product by the mere change of the computation of a single variable (a partial operation update), the other variables being operation-independent. Consequently, considerable design time savings are achieved by sharing the same core element for numerous different arithmetic operations. The feasibility of a re-pipelined online digit-serial signed-binary multiplication algorithm is established by employing the IEEE 754 SB RNE rounding technique, and compares it to an existing re-pipelined least-significant-digit- (LSD-) first digit-serial two's complement multiplication algorithm employing the same rounding technique. A simulation of the corresponding FPGA hardware implementation confirms the correct functionality of the algorithm. Parameterized gate-level area and delay estimates of corresponding ASIC hardware implementations are given. Moreover, an online bit-parallel signed-binary algorithm for multiply-accumulate operation employing a novel signed-binary to minimally redundant base-4 recoding technique, the IEEE 754 SB RNE rounding technique, and a novel overflow detection and correction technique is developed. The resulting algorithm is subsequently compared to an existing LSD-first bit-parallel signed-binary algorithm for multiply-accumulate operation employing an existing signed-binary to minimally redundant base-4 recoding technique and the IEEE 754 SB RNE rounding technique. A simulation of FPGA hardware implementation again confirms the correct functionality of the algorithm.

Acknowledgements

First, let me thank my wife, our families, and our close friends for supporting me during the journey coming to an end with this thesis, in particular for proof-reading part of it.

Neil definitely deserves my humble gratitude for accepting to proof-read the thesis when he had so much to do.

The writing of this thesis would not have been possible without NSERC and Micronet grants, obtained through the hard work of numerous students in the research group. In particular, Vishwas Rao has initiated the work on which this thesis is based.

Last, but far from being least, I would like to acknowledge the help, support, and guidance of my supervisor, Dr Nowrouzian.

Contents

1 Introduction **1**

1.1 Digital Signal Processing 1

1.2 Arithmetic for Digital Signal Processing 3

1.3 Data Processing Techniques 5

1.3.1 The Digit-Serial Arithmetic Technique 6

1.3.2 Online Arithmetic Technique 7

1.4 Open Problems 8

1.4.1 Online Processing 8

1.4.2 Constant-Delay Addition 9

1.4.3 Digit-Serial Online Operations 9

1.5 Overview of the Thesis 10

2 Theoretical Background for High-Speed Digit-Serial Online Arithmetic Operations **12**

2.1 Introduction 12

2.2 Fixed-Point Arithmetic 13

2.2.1 Introduction 13

2.2.2 Number Representation and Number Systems 13

2.2.3 Similarities Between Addition Schemes in Generalized Signed-Digit Number Systems 16

2.3 Data Processing Techniques 20

2.3.1 Digit-Serial Arithmetic Technique 20

2.3.2 Online Arithmetic Technique 30

2.4 High-Speed Signed-Binary Addition 34
2.4.1 Redundant Binary Addition Schemes 35
2.4.2 Characterization and Equivalence of Redundant Binary Addition Schemes 38
2.4.3 Bit-Serial and Digit-Serial Signed-Binary Limited-Carry Addition Architectures 43
2.5 Chapter Summary 47

3 Theoretical Background for Online Signed-Digit Multiplication and Multiply-Accumulate Operations 48
3.1 Introduction 48
3.2 Proposed Algorithm for Signed-Digit Online MAC Operation 49
3.2.1 Nomenclature 49
3.2.2 Signed-Digit Online MAC Algorithm 51
3.2.3 Determination of the Parameters of the Algorithm 54
3.3 Algorithms and Building Blocks for High-Speed Signed-Binary Multiplication Architectures 59
3.3.1 Single-Digit Multiplier and Digit Clearing Unit 59
3.3.2 Signed-Binary to Minimally Redundant Base-4 Recoding Technique 61
3.4 Rounding and Overflow Handling in Online Arithmetic Operations 70
3.4.1 Rounding of Signed-Binary Words 70
3.4.2 Overflow Handling in Online Arithmetic Operations 79
3.5 Chapter Summary 88

4 Architecture for Online Signed-Digit Digit-Serial Multiplication 90
4.1 Introduction 90
4.2 Algorithm for Bit-Serial Signed-Binary Online Multiplication 91
4.3 Extension to a Digit-Serial Signed-Binary Online Multiply-and-Round Algorithm 95
4.4 Architecture for Digit-Serial Signed-Binary Online Multiplication and Rounding 99
4.4.1 Multiplicand Selection Unit 101

4.4.2 Online Multiplication Units 101
4.4.3 Online Rounding Units 103
4.5 Computer Simulation Results and Performance Comparison 104
4.5.1 Simulation Results 104
4.5.2 Throughput Parameterization and Improvement Via Re-Pipelining . 106
4.5.3 Area Parameterization 109
4.5.4 Performance Comparisons 110
4.6 Conclusion 112

5 Architecture for Online Signed-Binary Bit-Parallel Multiply-Accumulate Operation 114

5.1 Introduction 114
5.2 Bit-Parallel Signed-Binary MAC Algorithm Employing SB to MRB4 Recoding 115
5.3 Architecture for MAC Operation Employing Signed-Binary Multiplier Recoding 122
5.3.1 Signed-Binary to Minimally Redundant Base-4 Recoders 122
5.3.2 Online Multiplication Units 123
5.3.3 Rounding and Overflow Correction Unit 124
5.4 Computer Simulation Results and Performance Comparison 124
5.4.1 Simulation Results 125
5.4.2 Throughput Parameterization 127
5.4.3 Area Parameterization 127
5.4.4 Comparison 128
5.5 Conclusion 130

6 Conclusion 131

6.1 Review of Material Presented 131
6.2 Proposed Areas of Future Work 132
6.3 Concluding Remarks 134

Bibliography 135

A Systematic Enumeration of Redundant Binary Addition Schemes **139**

A.1 Determination of Set Cardinalities 139

A.2 Systematic Determination of the Sets S_{c_1}, S_α, S_{c_2}, and S_β 141

A.2.1 Case 1: $|S_{c_1}| = 3$ and $|S_\alpha| = 2$ 142

A.2.2 Case 2: $|S_{c_1}| = 2$ and $|S_\alpha| = 3$ 143

A.3 Characterization of Redundant Binary Number Addition Schemes 144

A.4 Equivalence of Redundant Binary Number Addition Schemes 145

A.5 Conclusion 150

List of Figures

1.1 Typical DSP System 2

2.1 4-Bit LSD-First Bit-Serial Binary Adder 26

2.2 Bit Number in a 4-Bit Bit-Serial Data Stream 27

2.3 4-Bit LSD-First Digit-Serial Binary Adder ($D = 3$) 28

2.4 Bit Number in a 4-Bit Digit-Serial Data Stream ($D = 3$) 28

2.5 Limited Carry Adder Unit 38

2.6 Architecture for Bit-Serial Online Limited-Carry Addition 44

2.7 Architecture for Digit-Serial Online Limited-Carry Addition 47

3.1 Bound on P_ρ as a function of δ and i_c for $\beta = 2$, $\gamma = 4$ and $l_a = 1$ 56

3.2 Three-Level Signed-Binary to Radix-4 Digit Set Conversion 66

4.1 Architecture for Digit-Serial Online Multiply-and-Round Operation 99

4.2 Architecture of an Online Multiplication Unit 102

4.3 Architecture of an Online Rounding Unit 103

4.4 Simulation Results for $D = 2$ and $W_a = W_b = 8$ 105

4.5 Simulation Results for $D = 3$ and $W_a = W_b = 8$ 106

4.6 Re-Pipelined Architecture for MAC Operation and IEEE 754 RNE Rounding, with $W_a = 5$ and $D = 4$ 108

4.7 Throughput Versus Digit-Size of Proposed and Existing Digit-Serial Multiplication Architectures 111

4.8 Log of Efficiency Versus Digit-Size of Proposed and Existing Digit-Serial Multiplication Architectures 112

5.1 Architecture for Bit-Parallel Online Multiply-Accumulate Operation Employing SB to MRB4 Recoding 123

5.2 Simulation Results for $W_a = W_b = 8$ 126

5.3 Throughput Versus Wordlength of Proposed and Existing Bit-Parallel MAC Architectures 129

5.4 Efficiency Versus Wordlength of Proposed and Existing Bit-Parallel MAC Architectures 129

List of Tables

1.1 Classification of Signals 2

2.1 Set Cardinalities in Redundant-Binary Addition Schemes 39

2.2 Parametrized Digit Sets of Redundant Binary Addition Schemes 40

2.3 Coding the Set Values 42

2.4 Code for c_1 and c_1' 45

3.1 Signed-Binary Encoding 67

3.2 Conventions for MRB4 Digit Encoding 68

3.3 Low-Precision Rounding of SB Numbers 77

3.4 Conversion of OVF_ρ, $POVF_\rho$, and S_ρ into ϵ_ρ 87

4.1 Test Vectors for Digit-Serial Multiply-Round Operation in Decimal 105

4.2 Test Vectors for Digit-Serial Multiply-Round Operation in Signed-Binary . 105

4.3 Hexadecimal to Signed-Binary Conversion Table 106

4.4 Area Requirement for the Multiplication Architecture 110

5.1 Test Vectors for Parallel MAC Operation in Decimal 125

5.2 Test Vectors for Parallel MAC Operation in Signed-Binary 126

5.3 List of Delays of Architectural Units 127

5.4 List of Areas of Units Constituent in the MAC Architecture 128

List of Symbols

A/D	analog-to-digital
ASIC	application-specific integrated circuit
BCS	binary carry-save
BS	bit-serial
BSD	balanced (ordinary) signed-digit
D/A	digital-to-analog converter
DFG	data-flow graph
DS	digit-serial
DSP	digital signal processing/processor
FA	full adder
FPGA	field-programmable gate-array
GSD	generalized signed-digit
GSDNS	generalized signed-digit number system
IEEE	Institute of Electrical and Electronics Engineers
LSD	least significant digit
MAC	multiply-accumulate (arithmetic operation)
MRB4	minimally redundant base-4
MSD	most significant digit
OSD	ordinary (balanced) signed-digit

OSDNS	ordinary signed-digit number system
RNE	round to nearest/even
RNU	round to nearest/upper
SB	signed-binary
TC	two's complement
USD	unbalanced signed-digit
USDNS	unbalanced signed-digit number system
VHDL	Very large scale integrated circuit Hardware Description Language
A	multiplicand word
a_i	multiplicand digit of index i
B	multiplier word
B_ρ	partially formed multiplier word at iteration ρ
b_i	multiplier digit of index i
C	addend word
C_ρ	partially formed addend word at iteration ρ
c_i	addend digit of index i
l_x	attainable upper limit/ bound of a representation
P_ρ	off-line partial operation update word at iteration ρ
$\widetilde{P}_\rho$	online partial operation result word at iteration ρ
$\overline{\widetilde{P}}_\rho$	truncated (to its γ MSDs) online partial operation result word at iteration ρ
$\widetilde{p}_{\rho,i}$	online partial operation result digit of index i at iteration ρ
R_ρ	off-line operation result word at iteration ρ
$\widetilde{R}_\rho$	online operation result word at iteration ρ
$\widetilde{r}_i$	online partial operation result digit of index i

W_x	wordlength of X
β	radix/base of a number representation/system
β^{-i}	weight of a digit of index i
γ	internal wordlength (of $\overline{\overline{P}}_\rho$)
Δ	offset parameter
δ, δ_{op}	latency of an online operation
ϵ_ρ	online error at iteration ρ
η	redundancy index (OSD number system)
ρ	iteration number
$+$	addition
$\mid$	logic OR operation
$\dot{+}$	addition of sets
$\dot{\times}$	multiplication of a set by a scalar
$\oplus$	XOR logic operation
$\frown$	AND logic operation
$\overline{\cdot}$	NOT logic operation
$\mathbf{N}$	set of positive integers
$\mathbb{Q}$	set of rational
$\mathbb{R}$	set of reals
$\mathcal{R}$	equivalence relationship between additions
$\mathbf{Z}$	set of signed integers
$\mathbf{Z}^-$	set of negative integers
$\mathbf{Z}^+$	set of positive integers

Chapter 1

Introduction

1.1 Digital Signal Processing

Most living creatures need to communicate in order to survive. Signals, which are sounds, gestures, or objects, convey the necessary pieces of information to perform communication.

The first man-made devices using electrical signals to convey information are the telegraph (invented by Morse in 1844), and the telephone (invented by Bell in 1876). The fundamental difference between telegraph and telephone is the type of signals they handle. To send a message by telegraph, one needs to write it on a sheet of paper. Then, a telegrapher translates every letter into Morse code. A letter is represented by a sequence of two symbols, which is called a binary representation of the letter. Then, the symbols are transmitted in sequence over a wire as short and long impulses of electricity. The message is decoded on the other side by another telegrapher and handed over to the recipient of the message or forwarded to another place if necessary. By contrast, to send a message by telephone one speaks in a mouthpiece which translates the air vibrations into electrical signals. Of course, these signals are not a binary representation of the message. The electrical signals propagate through a wire, and are translated back to air vibrations through an ear-piece so that the receiver can hear what was said on the other end.

In electrical engineering, a signal is represented as an amplitude that varies as a function of time. In other words, the domain of a signal is an interval of time, and its range an interval of amplitude. An interval is discrete if it contains only a finite number of values (as opposed to a continuous interval). Therefore, there are four classes of signals, as listed in Table 1.1. Discrete-time discrete-amplitude signals are frequently referred to as digital,

Table 1.1: Classification of Signals

	Continuous Amplitude	Discrete Amplitude
Continuous Time	temperature	street light
Discrete Time	precipitations	paycheque

whereas continuous-time continuous-amplitude signals are referred to as analog.

In nature, signals are usually analog, whereas man-made machines usually generate digital signals. Digital signals are handled by digital signal processors. A main important practical advantage of digital processors is their programmability, permitting various operations to be performed by using the same processor, and their cost-effectiveness. Fortunately, Nyquist determined that if certain conditions apply, analog signals can be digitized, i.e. represented using corresponding digital signals, and recovered perfectly. Several factors have contributed to the widespread use of the digital processing of analog and digital data, often called digital signal processing (DSP). First, the inventions of microprocessors and application-specific integrated circuits (ASICs) have permitted the custom design of digital signal processors. Second, the ever-decreasing size of silicon-based transistors has made possible the doubling of the speed of digital circuits every 18 months, and the reduction of their cost, the silicon area of a digital circuit largely determining its cost.

A typical DSP system is made of analog-to-digital (A/D) and digital-to-analog (D/A) converters, and of a digital signal processor, as shown in Figure 1.1. The A/D and D/A

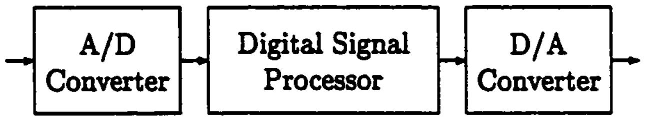


Figure 1.1: Typical DSP System

converters perform the same kind of operation as the translation of a message into their Morse code in the telegraph. The digital signal processor performs arithmetic operations, which most often comprise numerous additions and multiplications. The number of calculations to perform thereby determines the maximum speed at which the DSP system can operate. Moreover, the area of the digital signal processor is a dominant component in the cost of the DSP system. Both these factors, speed and area, depend greatly on how the addition and multiplication operations are implemented.

Of particular interest is an arithmetic operation capable of multiplying two numbers, and adding the result to a third number. Such an operation can be performed by a multiply-add or by a multiply-accumulate (MAC) arithmetic operation. The multiply-add arithmetic operation calculates the result and rounds it, whereas the MAC arithmetic operation calculates the full-precision result.

Example 1 *Let us multiply 0.01 by 0.02. The result is obtained as 0.0002 in a MAC arithmetic operation, whereas it is rounded to 0.00 in a multiply-add arithmetic operation.*

In this way, the use of MAC arithmetic operations is preferable in digital signal processors which are sensitive to calculation accuracy.

Consider a digital signal processor employing only addition and multiplication operations. A MAC modularization technique was developed in (Rao, 1996) to modify such a processor in order for it to use exclusively MAC arithmetic operations. Therefore, in a situation where area is the dominant factor, it is sufficient to build a digital signal processor having a single MAC arithmetic operator, and to perform multiplex all the operations onto that operator (c.f. the Motorola DSP56002).

A typical digital signal processor using addition, multiplication, and unit-delays is called a digital filter. Its primary objective is to selectively attenuate or boost certain frequency components of a signal. This is usually performed by adding weighted present and past input and past output samples, where the weights are constant. This corresponds to an inner product of a vector of constant weights by a vector of delayed input and output samples, which can be translated to a number of MAC arithmetic operations.

1.2 Arithmetic for Digital Signal Processing

Digital signals are sampled and quantized analog signals, and are represented at each sample time instant by a number. Many different number systems can be used, where the algebraic value of a number is represented by a succession of digits referred to as a word. The mapping from word to algebraic value can be performed by assigning each digit a weight. In this way, the algebraic value is obtained by summing the digit values multiplied by their corresponding weights. In most number systems, the weights are expressed as consecutive powers of a radix which will be denoted by β (also referred to as the base). Therefore, most number systems are characterized by their digit set and radix.

Example 2 *The radix of the decimal number system is 10, and the digit set is* $\{0,1,2,3,4,5,6,7,8,9\}$. *Consider the decimal number 5, which can be represented by the decimal word* 005_{10}, *where* $005_{10} = 0 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$. *Consider now the binary number system: it has a radix 2, and a digit set* $\{0,1\}$. *The decimal number 5 can now be represented in the binary number system by* 101_2, *where* $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 005_{10}$.

Radix- and digit set-based number systems can be classified into two categories, namely, the fixed-point and floating-point number systems. In a fixed-point number system, the weights have fixed values, i.e. the largest weight can always be made equal to, for example, β^{-1}. The digit related to the smallest weight is called the least significant digit (LSD), and the digit related to the largest weight is called the most significant digit (MSD). Moreover, the number of digits constituting the word is called the wordlength. In a floating-point number system, the word is split into two components, namely, the fractional and exponent parts f and e, respectively. The fractional part is a fixed-point representation of a number N, and the algebraic value of the desired number is obtained by multiplying N by β^e, which is referred to as scaling up by β^e. The increased range of a floating-point representation compared to that of a fixed-point representation is obtained at the expense of additional precision digits. Consequently, floating-point-based calculations are more prone to accuracy-related errors than fixed-point-based calculations. The fixed-point binary number system is the dominant number system in computers, mainly because its digit set has only two elements, logic 0 and logic 1.

This thesis is concerned with ordinary signed-digit (OSD) number systems, which are radix-β fixed-point number systems having a digit set $\{-\eta, \ldots, 0, \ldots, \eta\}$ such that $\left\lceil \frac{\beta}{2} \right\rceil \leq \eta \leq \beta$ (Avizienis, 1961; Parhami, 1990). These number systems are called redundant, because a given algebraic value may have several different representations.

Example 3 *The signed-binary (SB) number system has a radix 2, and a digit set* $\{\bar{1}, 0, 1\}$, *where* $\bar{1}$ *represents* -1. *The decimal number 3 can then be represented either by* 011_2, *where* $0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10}$, *by* $10\bar{1}_2$, *where* $1 \times 2^2 + 0 \times 2^1 - 1 \times 2^0 = 3_{10}$, *or even by* $1\bar{1}1_2$, *where* $1 \times 2^2 - 1 \times 2^1 + 1 \times 2^0 = 3_{10}$.

In high-speed arithmetic operations, the time required to perform an operation, i.e. the delay, must be minimized. Consider the addition of the words A and B of equal wordlength, yielding a sum S. If A, B, and S are words expressed in a non-redundant number system, then the delay to obtain S is, at best, proportional to the logarithm of the common wordlengths of A and B (Kornerup, 1994). This is mainly due to the fact that the value of the MSD of S depends on the value of the LSDs of A and B, thereby requiring the propagation of a carry along the full length of A and B. However, if the number system is redundant, then the delay can be made independent of the wordlength of A and B, because the value of the MSD is no longer dependent upon the values of all the digits of A and B (Avizienis, 1961; Parhami, 1990; Kornerup, 1994). Redundant number system hardware implementations exhibit larger areas relative to those of non-redundant number systems for equivalent operations, mainly because their digit sets contain more values. As a result, the absolute areas of redundant number system hardware implementations have represented a hindrance in the past. This hindrance has been overcome with the advent of sub-micron technologies.

In other number systems, such as in the logarithmic, residue, and rational number systems (Hwang, 1979), the mapping between the word and its algebraic value takes on different forms, but these forms are beyond the scope of this thesis.

1.3 Data Processing Techniques

A digital signal must be transmitted as a word from an A/D converter to a digital signal processor, and from the digital signal processor to a D/A converter. At a given time instant, one, several, or all digits of a word can be transmitted, i.e. in a bit-serial, digit-serial, or bit-parallel fashion, respectively. Of course, the bit-serial and bit-parallel fashions are subsumed by the digit-serial arithmetic technique. However, the digit-serial arithmetic technique can be derived from the bit-serial arithmetic technique.

Moreover, a bit-serial data stream can be processed, (a) the LSD first, which is the conventional arithmetic technique, or (b) the MSD first, which is the online arithmetic technique. The digit-serial and online arithmetic techniques are described in the following.

1.3.1 The Digit-Serial Arithmetic Technique

Let us consider an architecture having input data streams consisting of successions of words having a common wordlength W. By definition, the bit-serial arithmetic technique only requires to process one bit at a time (as opposed to W for the bit-parallel arithmetic technique). In addition, the number of wires required for transmission increases from 1 to W. Consequently, the bit-serial technique results in smaller processing units (which is area-effective), but the bit-parallel technique permits the design of faster architectures. The speed of an architecture is measured by its throughput, i.e. *the number of samples it can process per time instant.*

The main principal trade-off in the design of electronic systems involves achieving the highest execution speed at the smallest area and the lowest power. The digit-serial arithmetic technique proposes that the given architecture may process D digits of each of the input data streams per time instant, where the digit-size D may or may not be a divisor of W. In this way, a balance can be struck between the area efficiency of bit-serial systems and the speed efficiency of bit-parallel systems. It is important to note that the bit-serial and bit-parallel arithmetic techniques are special cases of the digit-serial arithmetic for a digit-size of 1 and W, respectively.

Any bit-serial architecture can be transformed into its unique digit-serial counterpart of digit-size D (Parhi, 1991). Then, the bit-serial data stream is sectioned into sets of D consecutive digits. Of course, digits from two consecutive words may be present in that set. Therefore, an arithmetic operation on digits coming from two different words/numbers may occur at a given time instant in a digit-serial architecture, but, of course, must not interfere.

1.3.2 Online Arithmetic Technique

Current practical A/D and D/A converters generate and consume digits one by one, from the MSD first to the LSD last. However, in the conventional arithmetic techniques, the carry propagates from the LSD first to the MSD last. Therefore, unless operations can be performed the MSD first, delays due to changes in the flow of digits occur. The online arithmetic technique performs arithmetic operations digit by digit, the MSD first. The result of such an operation must be expressed in a redundant number system[1] (Owens, 1983). Therefore, online arithmetic operations have been seldom used because of the large area in the corresponding hardware implementations.

The online arithmetic technique finds its roots in 1961, when Avizienis introduced the notions of signed-digit arithmetic and constant-delay addition (Avizienis, 1961). In 1977, Irwin introduced online algorithms for several arithmetic operations (Irwin, 1977). In the past two decades, the increase in transistor density has permitted the very large scale integration of corresponding architectures to take place. As a result, online algorithms have gained plenty of interest. The most popular online algorithms perform arithmetic operations iteratively, the MSD first.

Let us consider an algorithm performing an arithmetic operation. The inputs are referred to as the operands, and the output as the result. Formally speaking, the *online* property can be defined in terms of the input to, or the output from, an arithmetic operation (Owens, 1981). The algorithm is online with respect to its inputs when at a given iteration ρ, the first $\rho + k_{in}$ MSDs of the operands are required for calculation, where k_{in} is a small constant. Similarly, the algorithm is online with respect to its output when, at a given iteration ρ, the first $\rho - k_{out}$ MSDs of the result have been generated by the algorithm (Owens, 1981), where k_{out} is again a small constant. Finally, the algorithm is online when it is both online with respect to its inputs and with respect to its output. The constant $k_{in} + k_{out}$ is referred to as the *latency* of the online algorithm. The latency represents the number of iterations elapsing between the arrival time for the input MSD and the departure time for the result digit having the same weight as the input MSD. It is important to note that, in general, the

[1]If one assumes that the result word is expressed in a non-redundant number system, then the MSD cannot be output at the first iteration, because it depends on the values of the LSDs of the inputs.

weight of the MSD of the result is larger than the weight of the MSD of the input, resulting in a non-zero latency.

The flow of digits in online arithmetic operations corresponds to that in current practical A/D and D/A converters (unlike the conventional LSD-first arithmetic operations). The resulting advantages are that, (a) the delay due to a change in the flow of digits is eliminated, and (b) the LSDs can be discarded when full-precision computation is not required. Then, the result must be expressed in a redundant number system. If redundant representations are used both for the input words and the output words, then addition within online arithmetic processors can be performed in constant time.

The hardware implementations of online arithmetic processors require large areas, because redundant number systems require larger boolean functions which are more difficult to develop (Carter and Robertson, 1990; Ercegovac and Lang, 1990; Chow and Robertson, 1978); given the rapid decrease in transistor size, the area becomes less important.

In online arithmetic operations, the propagation of the carry toward the MSD must be stopped. This is accomplished by introducing a latency between the weight of the online input operand digit(s) and that of the output result digit. As a result, online algorithms result in slower parallel architectures than LSD-first algorithms for a given number system.

1.4 Open Problems

1.4.1 Online Processing

Numerous online algorithms performing arithmetic operations, including addition, subtraction, multiplication, division, and multiply-accumulate arithmetic operation (MAC arithmetic operation), have been developed (Irwin, 1977; Owens, 1981; Irwin and Owens, 1987; Guyot and Kusumaputri, 1991; Brackert et al., 1989; Sips and Lin, 1990; Lapointe et al., 1993; McQuillan and McCanny, 1995). The development of such algorithms is commonly based on the function approximation, the immediate evaluation, or the recursion-based approaches. The function approximation is employed for the approximation of complicated functions as polynomials. Two E-model approaches have been described, in (Ercegovac, 1984) and in (Sips and Lin, 1990). In (Ercegovac, 1984), a function value is calculated via an equivalent system of linear equations, for which efficient online architectures exist. In (Sips and Lin, 1990), an exact function value is fetched from a table by using the partially known inputs, and an output digit is estimated by taking into account the previous output digits. These two approaches can be employed for any function. The recursion-based approach can be employed for any polynomial which includes addition, MAC operation, and inner product (Irwin, 1977; Ercegovac, 1984; Sips and Lin, 1990).

So far, none of these techniques have separated the mechanism of the online process from the calculation of the function under study. In particular, finding an internal mechanism common to all recursion-based online algorithms remains an open problem for arithmetic operations. The main advantage of such a mechanism would be its applicability to the calculation of several different operations: in the industry, a corresponding hardware implementation would be common to a number of arithmetic operations, and design re-use could be applied, saving substantial design time and capital.

1.4.2 Constant-Delay Addition

In a redundant number system with large radix value, arithmetic operations require large boolean functions which are difficult to optimize for hardware implementation. As a result, the redundant binary system remains the most suitable for hardware implementation. Multiplications and MAC operations using redundant-binary number systems are implemented as nested additions of redundant-binary numbers. Architectures for constant-delay hardware implementations of redundant-binary addition have been reported in (Chow and Robertson, 1978; Parhami, 1988; Thornton, 1997). A systematic enumeration of such architectures for redundant-binary addition has not been undertaken. Such an approach may result in novel, small, and high-speed hardware implementations. Moreover, interrelationships between the existing developments of architectures for redundant binary addition may be discovered, saving design time.

1.4.3 Digit-Serial Online Operations

The hardware implementations of architectures for online arithmetic operations often result in large areas, implying high cost. Therefore, despite their outstanding features (including MSD-first processing, and low latency), online arithmetic operations are seldom used in practical applications. However, the digit-serial arithmetic technique, which permits a trade-off between the speed and area of an architecture for arithmetic operation, has been applied to some online operations only for the special case of the digit-size being a divisor of the wordlength (Irwin and Owens, 1988). The general digit-serial technique has not to been applied yet to the development of architectures for online MAC arithmetic operations.

1.5 Overview of the Thesis

The purpose of this thesis is twofold, namely, (a) to introduce the necessary background for the development of architectures for digit-serial online signed-digit arithmetic operations, and (b) to develop digit-serial and digit-parallel purely signed-digit multiply-accumulate operations.

Chapter 2 is concerned with an introduction to the mathematical framework necessary for digit-serial online fixed-point arithmetic operations, with a thorough description of limited-carry addition architectures. Discussions concerning number systems and the digit-serial and online arithmetic techniques are given. A simplification of the existing digit-serial unfolding algorithm is provided along with an introduction of a new dynamically changing wordlength technique. A systematic enumeration of architectures permitting the constant-delay addition of redundant binary numbers is presented. This is required for the design of high-speed signed-binary MAC operations as nested additions.

Chapter 3 introduces the necessary background for the development of industry-standard multiplication architectures. The development of bit-serial online MAC arithmetic operations is first discussed in detail, leading to the description of the recursion-based online mechanism, which can be used for the calculation of any affine function. Then, description of single-digit signed-binary multipliers is given, followed by the development of a novel technique for signed-binary to minimally redundant base-4 conversion. Such a conversion permits the design of faster and smaller architectures for MAC arithmetic operations. Also, an online signed-binary algorithm is given for IEEE 754 round-to-nearest/even, together with an online algorithm for signed-digit overflow handling.

Chapter 4 develops an algorithm for digit-serial online multiply-and-round arithmetic operation for general digit-size and input wordlength values. The algorithm employs the

IEEE 754 RNE industry standard. An architecture for subsequent FPGA or ASIC hardware implementation is given, which is re-pipelined for throughput maximization and proven functionally correct through simulation. The throughput and efficiency (throughput per unit area) performances of this architecture are compared unfavorably to those of an existing LSD-first digit-serial two's complement multiply-and-round operation employing signed-binary intermediate partial products. However, the architecture is shown to be viable for high-speed applications.

In Chapter 4, an algorithm is developed for signed-binary parallel online MAC arithmetic operation employing signed-binary to minimally redundant base-4 multiplier conversion, IEEE 754 RNE rounding, and overflow detection and correction. A corresponding architecture for subsequent FPGA or ASIC hardware implementation is given, and proven functionally correct through simulation. The throughput and efficiency figures are compared unfavorably to those of an existing architecture for signed-binary parallel LSD-first MAC arithmetic operation employing signed-binary to minimally redundant base-4 multiplier conversion and IEEE 754 RNE rounding.

Chapter 2

Theoretical Background for High-Speed Digit-Serial Online Arithmetic Operations

2.1 Introduction

Presently, digital signal processing finds numerous applications in many areas, such as virtual image synthesis, data transmission and reception, and database management. A digital signal is seen as time-dependent data, and is represented as a sequence of numbers (frequently referred to as samples), where each sample is represented as a sequence of digits arranged in a given format. Consequently, digital signal processing requires many arithmetic operations, predominantly additions and multiplications. The performance of an arithmetic operation depends heavily on the choice of the digit sets, how many digits to process at a time, and in which order the digits are processed.

The present chapter provides the necessary background for the design and development of architectures for online arithmetic operations in general, and for the multiplication operation in particular. In Section 2.2, the corresponding fixed-point number representations are discussed with an emphasis on generalized signed-digit number representations and their properties. Then, the digit-serial and online arithmetic techniques are introduced in Section 2.3. An improved digit-serial unfolding algorithm and an example of an online algorithm are also provided in that section. In addition, a new digit-serial unfolding technique for architectures performing operations where the wordlength of the input changes dynamically is proposed. Finally, in Section 2.4, limited-carry addition schemes that yield architectures whose delays are independent of the lengths of their inputs are explored. A characterization of these schemes permits one to link the design and development of several kinds of addition architectures, so as to reduce their design and development time.

2.2 Fixed-Point Arithmetic

2.2.1 Introduction

Arithmetic operations are the building blocks of digital signal processors. This section is concerned with a discussion of the impact of generalized signed-digit number representation, (a) on the range of permissible digital data, and (b) on the area and speed of the corresponding DSP hardware architecture.

2.2.2 Number Representation and Number Systems

Definition 4 *Number representation: consider a number N that belongs to a set S (e.g. $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{Q}$). A representation of N consists of*

1. *a digit set D (e.g. $\{0, \ldots, 9\}$),*
2. *an integer length L,*
3. *a sequence $n_i\rceil_{i_{\min} \le i \le i_{\max}}$ of digits that belong to D, where $i_{\max} - i_{\min} + 1 = L$, and*
4. *a mapping M defined in accordance with*

$$M : d_i\rceil_{i_{\min} \le i \le i_{\max}} \in D^L \mapsto s \in S$$

Then, N is represented by the sequence $n_i\rceil_{i_{\min} \le i \le i_{\max}}$ through the mapping M if

$$M\left(n_i\rceil_{i_{\min} \le i \le i_{\max}}\right) = N \tag{2.1}$$

holds.

The digits n_i are constrained to a certain digit set D. For example, the decimal representation of N requires that n_i belongs to the set $\{0, 1, \ldots, 9\}$. The sequence of digits is commonly referred to as a word.

A system in which number representations share the same mapping is called a number system. Number systems can be classified into two categories, namely complete and incomplete number systems. In a complete number system, any number from S can be represented by using the digit set D and the mapping M (Kornerup, 1994). This thesis is concerned with complete number systems only.

Fixed- and Floating-Point Number Representations

Usually, the digits of a number representation are assigned weights, denoted by the sequence $w_i|_{i_{\min}\leq i\leq i_{\max}}$, and the mapping is the obtained as the inner product of the digit and the weight sequences in accordance with

$$M\left(n_i|_{i_{\min}\leq i\leq i_{\max}}\right) = \sum_{i=i_{\min}}^{i_{\max}} n_i w_i. \tag{2.2}$$

Most often, such a representation is either referred to as fixed-point or as floating-point. Certain representation mappings do not use weights in this way (e.g. the logarithmic, residue, and rational number systems (Hwang, 1979)).

In a fixed-point representation, the weights are calculated in accordance with

$$w_i = \beta^{-i} \quad \forall i \in \{i_{\min}, \ldots, i_{\max}\}, \tag{2.3}$$

where β represents the radix (or base). In this thesis, a digit with index i is always associated with a weight β^{-i}. The digit of smallest weight is referred to as the least-significant digit, or LSD, and the digit of largest weight is referred to as the most-significant digit, or MSD.

In a floating-point representation, the number N is represented by a fixed-point mantissa m and a fixed-point exponent e such that $N = m\beta^e$. Part of the word is therefore reserved for m, and the other part is reserved for e.

This thesis is only concerned with fixed-point number representations. An important feature of such a representation is that a fixed-point number can always be multiplied by a power of the radix (scaled up) to yield an integer. A number system using weights in this way can thus be characterized by the radix β and the digit set D. The decimal number system ($\beta = 10$ and $D = \{0, \ldots, 9\}$) is the one humans use to learn to count and add, and the binary number system ($\beta = 2$ and $D = \{0, \ldots, 1\}$) is used by digital computers and other digital electronic devices.

Example 5 *The radix of the decimal number system is 10, and the digit set is* $\{0, \ldots, 9\}$. *Consider the decimal number 25, which can represented by the decimal word* 025_{10}, *where* $025_{10} = 0 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$. *Consider now the binary number system: it has a radix 2, and a digit set* $\{0, \ldots, 1\}$. *The decimal number 5 can now be represented in the binary number system by* 11001_2, *where* $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 025_{10}$.

Redundant Number Systems

A number system is said to be redundant when a given algebraic value can have several representations (Avizienis, 1961; Parhami, 1990; Kornerup, 1994).

Example 6 *The signed-binary (SB) number system has a radix 2, and a digit set* $\{\bar{1}, 0, 1\}$, *where* $\bar{1}$ *represents* -1. *The decimal number 3 can then be represented either by* 011_2, *where* $0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10}$, *or by* $10\bar{1}_2$, *where* $1 \times 2^2 + 0 \times 2^1 - 1 \times 2^0 = 3_{10}$.

The key point with redundant number systems rests with the addition of two redundant words. In fact, when using redundant number systems it is possible to develop constant-delay architectures for addition In fact, if the result is expressed in a redundant number system, it is possible to stop the carry propagation, leading to constant-delay architectures for addition (Avizienis, 1961; Chow and Robertson, 1978; Parhami, 1990; Kornerup, 1994; Rao, 1996; Thornton, 1997).

Let us describe the conventional addition. One adds two digits at a given digit position, yielding a sum digit for that digit position and a carry. Then, one adds the carry to the digits of the next higher digit position, yielding a sum digit and a new carry, and so on. Let us now describe constant-delay addition in redundant number systems. At any digit position, the two digits are combined to yield a carry for the next higher digit position, regardless of the value of the carry generated at the previous lower digit position. Then, the remaining value is combined with the incoming carry, yielding a sum digit. This process is also referred to as weight-transfer decomposition. As a consequence, the carry is absorbed immediately, and does not ripple along the length of the input words, justifying the term *constant-delay*. Specific number representations have been developed to stop the carry propagation at fixed digit position intervals (Phatak and Koren, 1994).

For some number systems, two consecutive weight-transfer decompositions are required for addition as their digit set does not allow sufficient redundancy in the sum word. In this way, the addition process can still be referred to as either constant-delay, or limited-carry, since the carry is absorbed after two digit positions instead of one. This is the case for the signed-binary number system (Avizienis, 1961; Parhami, 1990; Kornerup, 1994). The performances of the architectures and corresponding hardware implementations developed in this thesis rely greatly on this constant-delay property.

2.2.3 Similarities Between Addition Schemes in Generalized Signed-Digit Number Systems

Generalized signed-digit (GSD) number systems were formally introduced in (Parhami, 1990). They are radix-β number systems having a digit set of the form $\{-\eta_-, \ldots, \eta_+\}$, where $\eta_- \geq 0$, where $\eta_+ \geq 0$, and where $\eta_+ + \eta_- + 1 > \beta$. These number systems are redundant, and were proven to allow constant-delay addition (Parhami, 1990). In the following, a relationship between GSD number systems is introduced to prove that their addition mechanisms are very similar. The exploitation of these similarities results in substantial design time savings by using architecture re-use.

Relationships Between GSD Number Systems

A subset of GSD number systems, ordinary signed-digit (OSD) number systems, was also introduced in (Parhami, 1990). Their digit set is balanced, i.e. $\eta_+ = \eta_- \stackrel{\Delta}{=} \eta$, leading to the added advantage that if x belongs to the digit set $\{-\eta, -\eta+1, \ldots, \eta-1, \eta\}$, then $-x$ belongs to the same digit set. Unless otherwise stated, this thesis is concerned with OSD number systems only. Often, η is constrained as follows

$$\left\lceil \frac{\beta}{2} \right\rceil \leq \eta \leq \beta - 1, \tag{2.4}$$

where the lower bound is required in order for the corresponding number system to be complete and redundant (Kornerup, 1994), whereas the upper bound is required for the representation of 0 in the corresponding number system to be unique.

Let us introduce unbalanced signed-digit (USD) number systems, where one can choose $\eta_+ = \eta_- + 1 \stackrel{\Delta}{=} \eta$. One could have equivalently chosen $\eta_- = \eta_+ + 1$. Similarly, one can constrain η as follows

$$\left\lfloor \frac{\beta}{2} \right\rfloor + 1 \leq \eta \leq \beta - 1. \tag{2.5}$$

As shown in the following, any GSD number system can be related to a unique OSD/USD number system, and thus inherits the same addition mechanisms. This property is at the heart of various techniques for limiting the carry propagation in arithmetic operations (referred to as "tricks of the trade" in (Kornerup, 1994)).

Let us define the addition of two sets in accordance with

$$S_1 \dot{+} S_2 = \{a \text{ such that } a = a_1 + a_2, \text{ where } a_1 \in S_1 \text{ and } a_2 \in S_2\}, \tag{2.6}$$

and the multiplication of a set by a scalar in accordance with

$$n \dot{\times} S = \{y \text{ such that } y = n \times x, \text{ where } x \in S\}. \tag{2.7}$$

Let us give an example:

$$\{1,4\} \dot{+} \{0,3\} = \{1,4,7\}, \tag{2.8}$$

and

$$3 \dot{\times} \{1,4\} = \{3,12\}. \tag{2.9}$$

The relationship $\mathcal{R}$ between two GSD number systems can now be defined as follows:

Definition 7 *Denote by $T_{\epsilon,\Delta}$ a digit set transformation such that*

$$T_{\epsilon,\Delta} \quad : \quad D \in \Sigma \quad \mapsto \quad (\epsilon \dot{\times} D \dot{+} \{\Delta\}) \in \Sigma,$$

where Σ represents $\{D = \{-\eta_-, \ldots, \eta_+\} | (\eta_-, \eta_+) \in \mathbb{N}^{*2}\}$. Then, consider a radix-β GSD number system NS of digit set $D = \{-\eta_-, \ldots, \eta_+\}$, and another radix-$\beta$ GSD number system NS' of digit set $D' = \{-\eta'_-, \ldots, \eta'_+\}$ such that $\eta'_+ + \eta'_- + 1 = \eta_+ + \eta_- + 1$. NS is related through $\mathcal{R}$ to NS' if and only if there exist $\epsilon \in \{-1, 1\}$ and $\Delta \in \mathbb{Z}$ such that $D' = T_{\epsilon,\Delta}(D)$. This relationship is equivalently denoted by $NS\mathcal{R}NS'$.

It can be noted that the reverse transformation $T^{-1}_{\epsilon,\Delta}$ exists, and that

$$T^{-1}_{\epsilon,\Delta} = T_{\epsilon,-\epsilon\Delta}. \tag{2.10}$$

It can be proven that $\mathcal{R}$ is an equivalence relationship, i.e. that $\mathcal{R}$ is reflexive, symmetric, and transitive.

Let us denote the cardinality of a set S by $|S|$. There are two types of GSD number systems: those for which $|D|$ is odd, and those for which $|D|$ is even (referred to as odd and even GSD number systems, respectively).

Theorem 8 *Any odd GSD number system is related through $\mathcal{R}$ to a unique OSD number system, and any even GSD number system is related through $\mathcal{R}$ to a unique USD number system.*

Proof. The proof of the theorem consists of two parts, the first for odd GSD number systems, and the second for even GSD number systems.

- Let us consider an odd GSD number system $GSDNS$ of digit set $D = \{-\eta_-, \dots, \eta_+\}$. A necessary condition for an OSD number system $OSDNS$ of digit set $D' = \{-\eta, \dots, \eta\}$ to be related through $\mathcal{R}$ to $GSDNS$ is expressed by

$$|T_{\epsilon,\Delta}(D)| = |D'|, \tag{2.11}$$

which implies that

$$\eta + \eta + 1 = \eta_+ + \eta_- + 1 \quad \Leftrightarrow \quad \eta = \frac{\eta_- + \eta_+}{2}, \tag{2.12}$$

with η being a positive integer (because $GSDNS$ is odd). Therefore, only one unique OSD number system can be related to $GSDNS$. Let us prove that $GSDNS$ is related through $\mathcal{R}$ to $OSDNS$. If one chooses $\epsilon = -1$, then $\Delta = \eta_+ - \eta$ leading to

$$\begin{aligned} T_{\epsilon,\Delta}(D) &= -\{-\eta_-, \dots, \eta_+\} + \{\eta_+ - \eta\} & (2.13)\\ &= \{-\eta_+, \dots, \eta_-\} + \{\eta_+ - \eta\} & (2.14)\\ &= \{0, \dots, \eta_+ + \eta_-\} + \{-\eta\} & (2.15)\\ &= \{0, \dots, 2\eta\} + \{-\eta\} & (2.16)\\ &= \{-\eta, \dots, \eta\} & (2.17)\\ &= D'. & (2.18) \end{aligned}$$

Therefore, $GSDNS \ \mathcal{R} \ OSDNS$. Similarly, one can prove that $\epsilon = 1$ and $\Delta = \eta_- - \eta$ implying that $GSDNS \ \mathcal{R} \ OSDNS$. Consequently, $GSDNS \ \mathcal{R} \ OSDNS$, where $OSDNS$ is unique.

- Let us consider an even GSD number system $GSDNS$ of digit set $D = \{-\eta_-, \ldots, \eta_+\}$. Similar to the previous discussion, a necessary condition for a USD number system $USDNS$ of digit set $D' = \{-\eta+1, \ldots, \eta\}$ to be related by $\mathcal{R}$ to $GSDNS$ is expressed by

$$\eta + \eta - 1 + 1 = \eta_+ + \eta_- + 1 \quad \Leftrightarrow \quad \eta = \frac{\eta_- + \eta_+ + 1}{2}, \tag{2.19}$$

with η being a positive integer (because $GSDNS$ is even). Therefore, only one unique USD number system can be related to $GSDNS$. Let us prove that $GSDNS$ is related through $\mathcal{R}$ to $USDNS$. If one chooses $\epsilon = -1$, then $\Delta = \eta_+ - \eta + 1$ leading to

$$\begin{aligned} T_{\epsilon,\Delta}(D) &= -\{-\eta_-, \ldots, \eta_+\} + \{\eta_+ - \eta + 1\} && (2.20)\\ &= \{-\eta_+, \ldots, \eta_-\} + \{\eta_+ - \eta + 1\} && (2.21)\\ &= \{0, \ldots, \eta_+ + \eta_-\} + \{-\eta + 1\} && (2.22)\\ &= \{0, \ldots, 2\eta - 1\} + \{-\eta + 1\} && (2.23)\\ &= \{-\eta + 1, \ldots, \eta\} && (2.24)\\ &= D'. && (2.25) \end{aligned}$$

Therefore, $GSDNS \ \mathcal{R} \ USDNS$. Similarly, one can prove that $\epsilon = 1$ and $\Delta = \eta_- - \eta + 1$ implying that $GSDNS \ \mathcal{R} \ USDNS$. Consequently, $GSDNS \ \mathcal{R} \ USDNS$, where $USDNS$ is unique.

The above two bulleted points establish the proof. ∎

Theorem 8 implies that by assigning the same hardware code to an element of D and to its counterpart in D' via $T_{\epsilon,\Delta}$, the addition of two numbers expressed in these two different number systems can be performed by using the same architecture. Moreover, all the GSD number systems of a given class shares the addition mechanisms of a unique corresponding OSD or USD number system. Therefore, a lot of design time can be saved by considering these similarities in the internal addition mechanisms in GSD number systems.

An application of this theorem will be given in Section 2.4, where addition schemes will be characterized in the binary case leading to architectural similarities. Numerous studies of the properties of OSD number systems can now be applied to odd GSD number systems (Chow and Robertson, 1978; Irwin and Owens, 1987; Irwin and Owens, 1988; Parhami, 1988; Srinivas and Parhi, 1983; Thornton, 1997).

2.3 Data Processing Techniques

This section is concerned with the presentation of the digit-serial and the online arithmetic techniques as data processing methods. The former technique permits the processing of "several" digits at a time, from one to the full wordlength. An extension of the digit-serial technique to a dynamically changing wordlength situation is also introduced. The latter technique permits the processing of data the MSD first. An introduction to the approaches to the development of resulting arithmetic operations is given, followed by an example.

2.3.1 Digit-Serial Arithmetic Technique

The bit-serial arithmetic technique processes a digital signal one digit at a time, whereas the bit-parallel arithmetic technique processes it one word of length W at a time. The bit-serial technique is area-efficient because only one wire and one single-digit arithmetic unit are required to process the input data. Conversely, a bit-parallel transmission requires W wires and W single-digit arithmetic units, but allows the design of faster architectures by introducing as much concurrency between the internal operations as possible.

The digit-serial arithmetic technique processes words D digits per time instant, where the digit-size D may or may not be a divisor of W. The bit-serial and bit-parallel arithmetic are special cases of the digit-serial arithmetic technique for digit-sizes of 1 and W, respectively. Therefore, this technique allows a trade-off between the area-efficiency of bit-serial systems and the time-efficiency of bit-parallel systems by adjusting the parameter D.

Digit-Serial Unfolding Algorithm

Any bit-serial architecture can be represented by combinatorial units, unit-delays, switches, and wires. This representation can be translated into a data-flow graph, which is a directed graph whose nodes represent combinatorial units, and whose arcs represent either communication involving a non-negative integer number of delays or zero-delay communication at specific time instances. The digit-serial unfolding algorithm (Parhi, 1991) transforms a data-flow graph DFG of a bit-serial architecture into its corresponding unfolded data-flow graph $UDFG$ of digit-size D, processing D bits of the original bit-serial stream at a time, where D may or may not be a divisor of W.

An arc is denoted by $U \rightarrow V$, where node U represents its source, and where node V represents its destination. If the arc introduces i delays, then the result of node U obtained at time instant n_0 is used at time instant $n_0 + i$ in node V. If the arc provides zero-delay communication at a specific time instance, then the time instance is given by $Ww + u$, where W represents the wordlength of the bit-serial stream, where u belongs to the set $\{0, 1, \ldots, W-1\}$, and where w represents the word number in the bit-serial stream.

Let us denote L as the least common multiple of W and D. The digit-serial unfolding algorithm below was presented in (Parhi, 1991).

Algorithm 9 *Digit-Serial Unfolding Algorithm*

```
Step 1. For each node $U$ in DFG, draw $D$ nodes in UDFG, and label them $U_0, \ldots, U_{D-1}$.
Step 2. For each arc $U \rightarrow V$ in DFG having 0 delay, draw the arcs $U_q \rightarrow V_q$ with 0 delay for all $q \in \{0, \ldots, D-1\}$.
Step 3. For each arc $U \rightarrow V$ in DFG having $i$ delays,
    Step 3a). If $0 < i < D$, draw the arcs $U_{D-i+q} \rightarrow V_q$ with 1 delay for all $q \in \{0, 1, \ldots, i-1\}$, and draw the arcs $U_{q-i} \rightarrow V_q$ with 0 delay for $q \in \{i, i+1, \ldots, D-1\}$.
    Step 3b). If $i \geq D$, draw the arcs $U_{q-i+D\lceil \frac{q-i}{D} \rceil} \rightarrow V_q$ with $\left\lceil \frac{i-q}{D} \right\rceil$ delay(s) for all $q \in \{0, 1, \ldots, D-1\}$.
Step 4. For each switch $U \rightarrow S$ having a switching instance $Wl + u$, calculate the bit-serial switching instance as $Ll + u + wW$, where $w \in \{0, \ldots, \frac{L}{W} - 1\}$, and where the corresponding digit-serial switching instance $U_{(u+wW) \bmod D}$ are calculated as $\frac{L}{D}l + \lfloor \frac{u+wW}{D} \rfloor$, $\forall w \in \{0, \ldots, \frac{L}{W} - 1\}$.
```

Then, the following algorithm is a modification of Algorithm 9.

Algorithm 10 *Modified Digit-Serial Unfolding Algorithm*

Step 1'. *For each node U in DFG, draw D nodes in UDFG, and label them $U_0, \ldots, U_{D-1}$.*

Step 2'. *For each arc $U \to V$ in DFG having i delays, draw the arcs $U_{(q-i) \bmod D} \to V_q$ with $\left\lceil \frac{i-q}{D} \right\rceil$ delay(s) for all $q \in \{0, \ldots, D-1\}$.*

Step 3'. *For each switch $U \to S$ having a switching instance $Wl + u$, calculate the bit-serial switching instance as $Ll + u + wW$, where $w \in \{0, \ldots, \frac{L}{W} - 1\}$, and where the corresponding digit-serial switching instance $U_{(u+wW) \bmod D}$ are calculated as $\frac{L}{D} l + \left\lfloor \frac{u+wW}{D} \right\rfloor$, $\forall w \in \{0, \ldots, \frac{L}{W} - 1\}$.*

Theorem 11 *Algorithm 9 and Algorithm 10 perform the same data-flow graph unfolding operation.*

Proof. One can remark that Steps 1' and 3' of Algorithm 10 correspond exactly to Steps 1 and 4 of Algorithm 9, respectively. Therefore, one has to prove that Step 2' in Algorithm 10 performs the operations of Steps 2, 3(a) and 3(b) in Algorithm 9. This is achieved by successively considering the cases $i = 0$, $0 < i < D$, and $i \geq D$.

- Let us assume that $i = 0$, as in Step 2. One can readily observe that for q in $\{0, \ldots, D-1\}$,

$$(q - i) \bmod D = q \quad \text{and} \quad \left\lceil \frac{i-q}{D} \right\rceil = 0 \tag{2.26}$$

  hold. Therefore, for all q in $\{0, \ldots, D-1\}$, one arc $U_q \to V_q$ is drawn with 0 delays, which is the definition of Step 2: Step 2' and Step 2 are equivalent.

- Let us assume that $0 < i < D$, as in Step 3(a). Then, q belongs either to $\{0, \ldots, i-1\}$ or to $\{i, \ldots, D-1\}$.

  - Let us consider $q \in \{0, \ldots, i-1\}$, which implies that

$$(q - i) \bmod D = (D - i + q) \bmod D, \tag{2.27}$$

    or equivalently

$$(q - i) \bmod D = D - i + q, \tag{2.28}$$

    because

$$D - i \leq D - i + q \leq D - 1, \tag{2.29}$$

where $D - i > 0$. Moreover,

$$\left\lceil \frac{1}{D} \right\rceil \leq \left\lceil \frac{i-q}{D} \right\rceil \leq \left\lceil \frac{i}{D} \right\rceil \tag{2.30}$$

holds, which yields

$$\left\lceil \frac{i-q}{D} \right\rceil = 1. \tag{2.31}$$

In this case, Step 2' draws the arcs $U_{D-i+q} \rightarrow V_q$ with 1 delay for all q in $[0, \ldots, i-1]$. Therefore, for $0 < i < D$ and $q \in [0, i-1]$, Step 2' and Step 3(a) are equivalent.

– Let us consider $q \in \{i, \ldots, D-1\}$, which implies that

$$(q - i) \bmod D = q - i. \tag{2.32}$$

Moreover,

$$\left\lceil \frac{i-D+1}{D} \right\rceil \leq \left\lceil \frac{i-q}{D} \right\rceil \leq \left\lceil \frac{0}{D} \right\rceil \tag{2.33}$$

holds, yielding

$$\left\lceil \frac{i-q}{D} \right\rceil = 0, \tag{2.34}$$

because

$$\left\lceil \frac{i-D+1}{D} \right\rceil = \left\lceil \frac{i+1}{D} \right\rceil - 1, \tag{2.35}$$

where $\lceil \frac{i+1}{D} \rceil = 1$. In this case, Step 2' draws the arcs $U_{q-i} \rightarrow V_q$ with 0 delay for all q in $\{i, \ldots, D-1\}$. Therefore, for $0 < i < D$, Step 2' and Step 3(a) are also equivalent when $q \in \{i, \ldots, D-1\}$.

From these two points, $0 < i < D$ implies that Step 3(a) and Step 2' are equivalent.

- Let us now assume that $i \geqslant D$. Firstly, one has to prove that Step 2' and Step 3(b) create the same arcs. By recalling the definition of the modulo operation,

$$a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor, \tag{2.36}$$

one can write

$$(q-i) \bmod D = q - i - D\left\lfloor \frac{q-i}{D} \right\rfloor, \tag{2.37}$$

which becomes

$$(q-i) \bmod D = q - i + D\left\lceil \frac{i-q}{D} \right\rceil. \tag{2.38}$$

Therefore, Step 2' creates the arcs $U_{q-i+D\lceil\frac{i-q}{D}\rceil} \to V_q$ for $q \in \{0, \ldots, D-1\}$, which is the definition of the creation of the arcs in Step 3(b). Secondly, each arc created by using Step 2' must be shown to have the same number of delays than if created by Step 3(b), which holds immediately. Therefore, for $i \geqslant D$, Step 2' and Step 3(b) are equivalent.

These above three bulleted cases establish the proof. ■

As a result of Theorem 11, digit-serial unfolding can be performed by using Algorithm 10, thereby avoiding the multiple delay-based cases for the instantiation of arcs.

Theorem 12 *One can write Step 2' equivalently as follows: for each arc $U \to V$ in DFG having i delays, draw the arcs $U_{q'} \to V_{(q'+i)\bmod D}$ with $\left\lfloor\frac{q'+i}{D}\right\rfloor$ delay(s) for all $q' \in \{0, \ldots, D-1\}$.*

Proof. The proof is established by proving that, given i in $\mathbf{N}$, and given q' in $\{0, \ldots, D-1\}$, any arc $U_{q'} \to V_{(q'+i)\bmod D}$ with $\left\lfloor\frac{q'+i}{D}\right\rfloor$ delay(s) is the same arc as $U_{(q-i)\bmod D} \to V_q$ with $\left\lceil\frac{i-q}{D}\right\rceil = \left\lfloor\frac{q'+i}{D}\right\rfloor$ delay(s), where $q = (q'+i) \bmod D$.

Given i in $\mathbf{N}$, and given q' in $\{0, \ldots, D-1\}$, defining

$$q = (q'+i) \bmod D \tag{2.39}$$

implies that

$$\begin{align} (q-i) \bmod D &= ((q'+i) \bmod D - i) \bmod D \tag{2.40} \\ &= (q'+i-i) \bmod D \tag{2.41} \\ &= q' \bmod D \tag{2.42} \\ &= q', \tag{2.43} \end{align}$$

because $q' \in \{0,\ldots,D-1\}$. Of course, $q \in \{0,\ldots,D-1\}$ by definition. Therefore, $U_{q'} \rightarrow V_{(q'+i)\mathrm{mod}D}$ is the same arc as $U_{(q-i)\mathrm{mod}D} \rightarrow V_q$ when q is defined in accordance with Eqn. 2.39.

One has to prove that if i belongs to $\mathbf{N}$, and if q' belongs to $\{0,\ldots,D-1\}$, then

$$\left\lceil \frac{i-q}{D} \right\rceil = \left\lfloor \frac{q'+i}{D} \right\rfloor . \tag{2.44}$$

Firstly, by definition of the modulo operation, one can write

$$q = q' + i - D \left\lfloor \frac{q'+i}{D} \right\rfloor . \tag{2.45}$$

From Eqn. 2.43 and by modulo operation,

$$q' = q - i - D \left\lfloor \frac{q-i}{D} \right\rfloor . \tag{2.46}$$

By adding Eqns. 2.45 and 2.46, and by recalling that $-\lfloor a \rfloor = \lceil -a \rceil$, one obtains

$$q' + q = q + q' - i + i + D \left(\left\lceil \frac{i-q}{D} \right\rceil - \left\lfloor \frac{q'+i}{D} \right\rfloor \right) , \tag{2.47}$$

which readily leads to

$$\left\lfloor \frac{q'+i}{D} \right\rfloor = \left\lceil \frac{i-q}{D} \right\rceil . \tag{2.48}$$

In this way, it has been established that given i in $\mathbf{N}$, drawing the arcs $U_{q'} \rightarrow V_{(q'+i)\mathrm{mod}D}$ with $\left\lfloor \frac{q'+i}{D} \right\rfloor$ delay(s) for all q' in $\{0,\ldots,D-1\}$ is equivalent to drawing the arcs $U_{(q-i)\mathrm{mod}D} \rightarrow V_q$ with $\left\lceil \frac{i-q}{D} \right\rceil$ delay(s) for all q in $\{0,\ldots,D-1\}$. ■

Theorem 12 allows the replacement in the modified algorithm of Step 2' by another equivalent step, which takes the source node number as the index of instantiation instead of the destination node number.

By varying the digit-size in the application of the digit-serial unfolding algorithm, which translates bit-serial architectures into digit-serial architectures, one can strike a balance between the area and the speed of a given hardware architecture. A modified digit-serial unfolding algorithm allows one to perform the instantiation of the registers of an architecture in a single step without the need for testing the number of delays on the current arc. Moreover, the same step in the modified algorithm can be performed by considering either the source nodes or the destination nodes.

Application of the Digit-Serial Architecture Unfolding

Any bit-serial architecture can be translated into its unique digit-serial conterpart of digit-size D by applying the digit-serial unfolding algorithm presented in (Parhi, 1991). In the following, this algorithm is applied to the example of bit-serial unsigned binary addition.

Consider two digital signals a and b represented by binary words of wordlength 4, where a binary digit is referred to as a bit. The bits are transmitted bit-serially, the LSD first, and the words are concatenated in time. The addition of the words in the two data streams can be performed by the full-adder (FA) architecture shown in Figure 2.1. There is only one

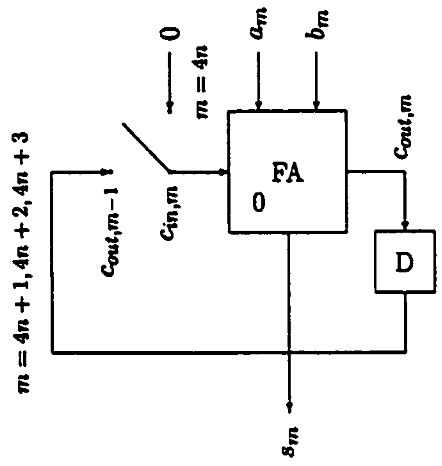


Figure 2.1: 4-Bit LSD-First Bit-Serial Binary Adder

FA unit, which is numbered 0. At each bit-serial time instant m, the incoming bits a_m and b_m are added together with the input carry $c_{in,m}$, and a sum bit s_m and a corresponding output carry is generated in accordance with

$$2c_{out,m} + s_m = a_m + b_m + c_{in,m} \tag{2.49}$$

The output carry is then stored in a unit-delay register D for use in the next bit-serial time instant.

Most of the time, i.e. for $m \in \{4n+1, 4n+2, 4n+3\}$, the switch equates the current input carry value $c_{in,m}$ with the output carry at the previous bit-serial time instant, $c_{out,m-1}$. Otherwise, i.e. for $m = 4n$, the switch resets the input carry to 0. In other words, the addition of two words begins at every bit-serial time instant m of the form $m = 4n$ in the bit-serial architecture. The operation performed by the switch is illustrated by the time-line shown in Figure 2.2. The bit number in the A or B data stream fed to the FA unit number 0

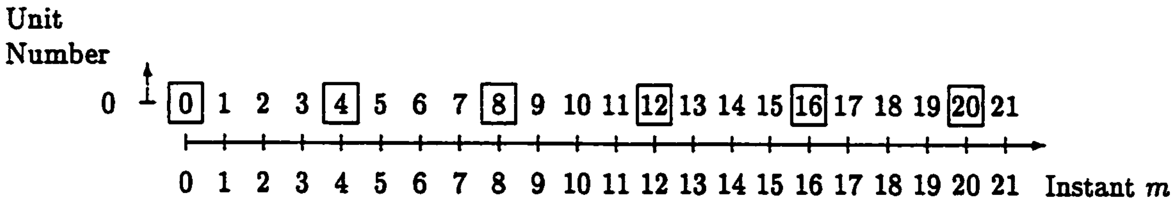


Figure 2.2: Bit Number in a 4-Bit Bit-Serial Data Stream

is given at any bit-serial time instant m. The bit-serial time instant corresponds exactly to the bit number in the bit stream as the words are transmitted one bit per bit-serial time instant. A boxed number indicates the bit number of the LSD of a word, and is equal to the bit-serial time instant at which the switch must reset the carry-in of the FA unit number 0. In a corresponding hardware implementation, the reset switch is controlled by a signal $Ctrl_0$.

The bit-serial architecture can be transformed into its digit-serial counterpart with a digit-size of $D = 3$, by using the digit-serial unfolding algorithm by Parhi (Parhi, 1991), as shown in Figure 2.3. One can observe that the unfolded architecture has been obtained as a cascade of FA units with resettable input carry, numbered from 0 to $D - 1$. In a corresponding hardware implementation, the reset switches are controlled by corresponding signals $Ctrl_0$, $Ctrl_1$, ... , $Ctrl_{D-1}$.

It is important to note that m' represents the digit-serial time instant (as opposed to m for the bit-serial time instant). This architecture fetches $D = 3$ consecutive bits in the bit-serial data stream to process them in parallel at a given digit-serial time instant m', as illustrated in the time-line shown in Figure 2.4. At a given digit-serial time instant m', the FA unit number 0 is fed with the bit number $3m'+0$, while the FA unit number 1 is fed with the bit number $3m' + 1$, and the FA unit 2 ($= D - 1$) is fed with the bit number $3m' + 2$. It can now be observed that the reset switches of the FA units 0, 1, and 2 have to reset the input carries of the same FA units at digit-serial time instants of the form $4n'$, $4n' + 1$, and $4n' + 2$, respectively. Therefore, no input carry has to be reset when $m' = 4n' + 3$.

One can observe that the bit-serial data stream is sectioned into sets of D consecutive digits. Of course, digits belonging to two consecutive samples may be present in those

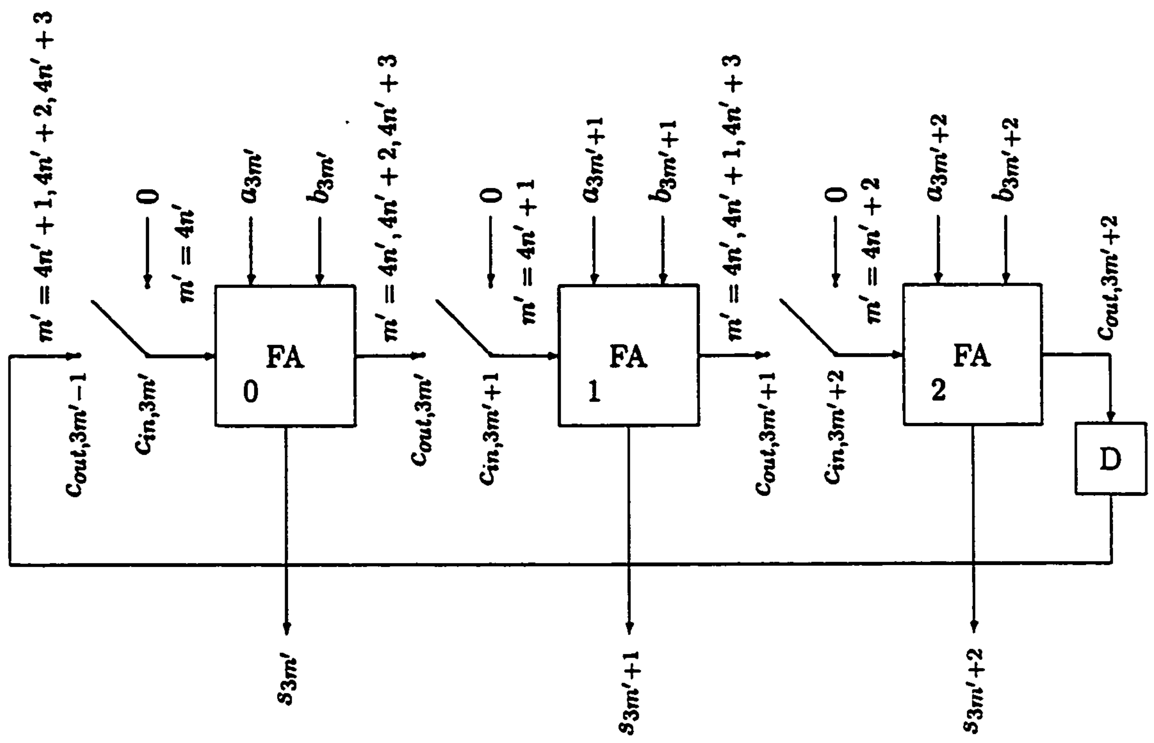
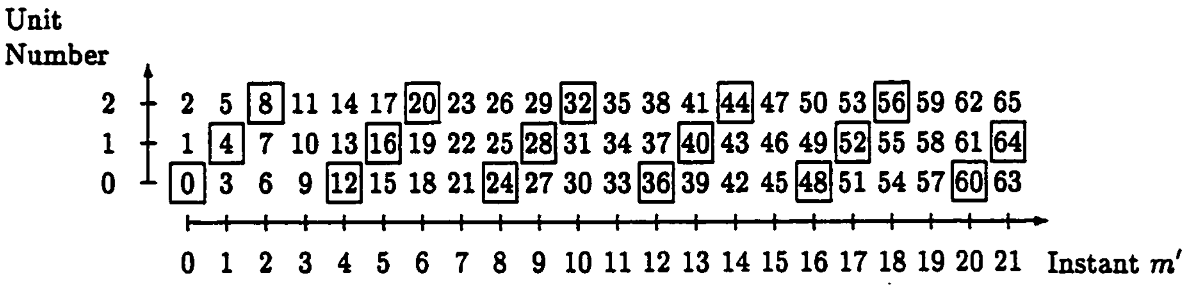


Figure 2.3: 4-Bit LSD-First Digit-Serial Binary Adder ($D = 3$)



Unit Number
2: 2 5 $\boxed{8}$ 11 14 17 $\boxed{20}$ 23 26 29 $\boxed{32}$ 35 38 41 $\boxed{44}$ 47 50 53 $\boxed{56}$ 59 62 65
1: 1 $\boxed{4}$ 7 10 13 $\boxed{16}$ 19 22 25 $\boxed{28}$ 31 34 37 $\boxed{40}$ 43 46 49 $\boxed{52}$ 55 58 61 $\boxed{64}$
0: $\boxed{0}$ 3 6 9 $\boxed{12}$ 15 18 21 $\boxed{24}$ 27 30 33 $\boxed{36}$ 39 42 45 $\boxed{48}$ 51 54 57 $\boxed{60}$ 63
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Instant m'

Figure 2.4: Bit Number in a 4-Bit Digit-Serial Data Stream ($D = 3$)

sets, e.g. samples 0 and 1 at the digit-serial time instant $m' = 1$ (digits 3, 4, and 5) in Figure 2.4. Therefore, the main function of the reset switches is to isolate these two consecutive additions being processed at the same digit-serial time instant in the same addition architecture.

Dynamically Changing Wordlength Technique

Consider the digit-serial unfolding example shown previously in this section. Let us assume that the addend and augend words have a dynamically changing wordlength, i.e. that W is a function of the word number $word \geq 0$. Then, it is possible to calculate the active-high control signals $Ctrl_i$ at each digit-serial time instant in such a manner as to permit addition, as shown in Algorithm 13.

Algorithm 13 *Calculation of Control Signals for Dynamically Changing Wordlength*

```
Step 1: Set $word \leftarrow 0$;
Step 2: Set $c \leftarrow 0$;
Step 3: For $i$ between 0 and $D-1$,
            If $i = c$, then
              • Set $Ctrl_i \leftarrow 1$;
              • Set $c \leftarrow c + W_{word}$;
              • Set $word \leftarrow word + 1$;
            Else
              • Set $Ctrl_i \leftarrow 0$;
            End
        End
Step 4: Set $c \leftarrow c - D$;
Step 5: Go to Step 3
```

A resulting hardware implementation thus calculates the addition of an addend to an augend of time-evolving wordlength while still using only D bit-serial cells.

It is important to note that the above algorithm is valid only if the minimum wordlength value is larger than the digit-size D. Otherwise, more than two $Ctrl_i$ signals may be active at a given time instant. Moreover, the above algorithm can be used for any control signal that must be active at a time instance Wn, where $n \in \mathbf{N}$.

Let us consider a control signal that must be active c_{offset} bit-serial time instants after the start of the operation, i.e. at bit-serial time instances $Wn + c_{offset}$, where $n \in \mathbf{N}$, and where $c_{offset} \in \mathbf{Z}$. Then, one can generalize the above algorithm by replacing Step 2 by the step shown below.

```
Step 2 new: If $c_{offset} < 0$, then
        • Set $c \leftarrow W_0 + c_{offset}$;
        • Set word $\leftarrow 1$;
    Else
        • Set $c \leftarrow c_{offset}$;
        • Set word $\leftarrow 0$;
    End
```

The main benefit of a resulting digit-serial cell is that it accepts inputs of dynamically changing wordlengths. As a result, coarser results using shorter wordlengths can be calculated faster for a fixed hardware requirement. For example, if the wordlength drops by half, then the throughput is increased by a factor of 2.

Let us describe the digit-serial dynamically changing wordlength unfolding technique for directed flow graphs representing arbitrary architectures. First, one executes steps 1 and 2 of Algorithm 10. Then, for every switch, the control signal active at the bit-serial time instance $Wn + c_{offset}$ It is imperative that the bit-serial cell accepts inputs of dynamically changing wordlengths. The main disadvantage of this technique is that one must ensure that the wordlength truncation does not affect the functionality of the whole system.

2.3.2 Online Arithmetic Technique

In conventional arithmetic operations, the carries naturally ripple toward the MSDs of the result. The main practical disadvantages of conventional arithmetic operations are, (a) the required change in the flow of digits from the MSD (imposed by A/D and D/A converters) to the LSD first, and (b) the large latency due to the calculation in LSD-first arithmetic. In order to avoid these problems, efforts have been made in the past two decades to develop online operations processing digits the MSD first, which exhibit small latencies. An introduction to the main approaches to the development of online arithmetic operations will be given, followed by the development of an example to illustrate the technique.

Approaches to the Development of Online Arithmetic Operations

A number of online arithmetic architecutres have been developed in the past, ranging from the traditional addition, subtraction, multiplication, and division (Trivedi and Ercegovac, 1977; Irwin, 1977; Owens, 1980; Chow, 1980; Gorji-Sinaki and Ercegovac, 1981; Irwin and Owens, 1987; Irwin and Owens, 1988; Guyot et al., 1989; Perlee and Casasent, 1989; Ercegovac and Lang, 1990; Privat, 1990; Balsara et al., 1991; Sips and Lin, 1990; Guyot and Kusumaputri, 1991; Srinivas and Parhi, 1983) to the more complex trigonometric, exponential, or filtering operations (Owens, 1981; McQuillan and McCanny, 1995; Fernando and Ercegovac, 1992; Lapointe et al., 1993; Lin and Sips, 1990; McNally et al., 1990; Brackert et al., 1989). Online arithmetic operations have been developed by employing either a recursion-based method or an evaluation-based method.

The recursion-based method was developed first (Trivedi and Ercegovac, 1977; Irwin, 1977) and yields iterative and recursive algorithms. A review of such algorithms was presented in (Ercegovac, 1984). Many of the initial recursive online algorithms were obtained by successively transforming a conventional (LSD-first) algorithm to accept input digits the MSD first, and then introducing additional steps in order for the algorithm to generate the output digits in an online fashion (Trivedi and Ercegovac, 1977; Irwin, 1977; Owens, 1980; Chow, 1980; Gorji-Sinaki and Ercegovac, 1981). In this way, conventional algorithms based on continued sum/products for the evaluation of general functions (e.g. sine, logarithm, or multiplicative inverse) have been transformed into online algorithms using the aforementioned procedure (Owens, 1981). However, most of the recent recursive online algorithms were obtained by directly considering the online flow of the input and output digits (Trivedi and Ercegovac, 1977; Chow, 1980; Gorji-Sinaki and Ercegovac, 1981; Guyot et al., 1989; Privat, 1990; Guyot and Kusumaputri, 1991; Srinivas and Parhi, 1983; McQuillan and McCanny, 1995; Fernando and Ercegovac, 1992; Lapointe et al., 1993; Lin and Sips, 1990; McNally et al., 1990; Brackert et al., 1989).

The recursion-based method yields algorithms which share a common iterative sequence. At a given iteration, a partial operation update is added to a scaled online error formed at the previous iteration, where the scaling factor is equal to the number system radix. Then, the result is rounded to an integer called the online result digit. Finally, an online error is calculated for use in the next iteration.

Let us recall that the online result MSD is the first online result digit to be calculated. If no attention is paid to the magnitude of the scaled online error and that of the partial operation update, the radix-β representation of the result of their addition may require MSDs of weights that are larger than that of the online result digit to be outputted. This problem is equivalent to stopping the carry propagation of the cumulative sum of the off-line partial updates, facilitated by scaling down the partial operation update by a constant factor, and by using a redundant representation for the online result. Unfortunately, by scaling down the partial operation update, one increases the latency of the online arithmetic operation (c.f. Section 1.3.2). The determination of the minimum latency depends on the online arithmetic operation, as shown by many independent studies (Duprat et al., 1989; Sips and Lin, 1990). In fact, the latency corresponds to the number of MSDs exceeding the input representation format (referred to as guard digits), and is thus determined both by the precision of online result digit estimation and by the range of the result of the operation itself.

More recently, the evaluation-based method was developed with the intention of finding the minimum latency of any arithmetic operation (Sips and Lin, 1990). Algorithms for fixed-point arithmetic operation obtained by using this method all share a common procedure and are all iterative. At a given iteration, the off-line iterative result is calculated as accurately as the precision of the input words permits, and an online result digit is calculated by taking into account that result as well as the previous online result digits. Of course, the hardware implementations of the exact result calculation are impractical, i.e. they are too slow or too large (Sips and Lin, 1990). Therefore, in this thesis, efforts are concentrated on the recursion-based approach.

Application: Online Algorithm for Signed-Binary Addition

Let us consider the addition of an addend A to an augend B resulting in a sum R in accordance with

$$R = A + B. \tag{2.50}$$

The inputs A and B, referred to as the operands, are signed-binary number representations in accordance with

$$A = \sum_{i=0}^{W-1} a_i 2^{-i}; \quad a_i \in \{-1, 0, 1\} \tag{2.51}$$

and

$$B = \sum_{i=0}^{W-1} b_i 2^{-i}; \quad b_i \in \{-1, 0, 1\}, \tag{2.52}$$

where W represents the wordlength of A and B. The MSDs of A and B are a_0 and b_0, respectively. As the addition must be online with respect to the operands, the inputs to the algorithm at iteration ρ $(0 \leq \rho \leq W-1)$ are a_ρ and b_ρ, thus leading to the formation of

$$A_\rho = \sum_{i=0}^{\rho} a_i 2^{-i} \text{ and } B_\rho = \sum_{i=0}^{\rho} b_i 2^{-i} \tag{2.53}$$

which represent the partially formed operands A_ρ and B_ρ.

The sum digits $\tilde{r}_{\rho-\delta}$ are similarly generated in an online fashion, thus leading to the formation of the signed-binary representation

$$\tilde{R}_\rho = \sum_{i=-\delta}^{\rho-\delta} \tilde{r}_i 2^{-i}, \tag{2.54}$$

where δ represents the latency of the online addition algorithm to be discussed later. The signed-binary representation $\tilde{R}_\rho$ is an increasingly more accurate approximation of the sum R. The difference $R - \tilde{R}_\rho$ has two origins, namely, the absence of knowledge of all the digits of accuracy of the operands, and the online error which is due to the online digit selection process.

The following algorithm performs the addition of A and B in an online fashion: at a given iteration ρ, a_ρ and b_ρ are input, and the online sum digit $\tilde{r}_{\rho-\delta}$ is generated.

Algorithm 14 *Online Ordinary Signed-Digit Addition*

```
Step 1: Set $\rho \leftarrow 0$ and $\epsilon_{-1} \leftarrow 0$,
Step 2: Calculate $P_\rho \leftarrow (a_\rho + b_\rho)\beta^{-\delta}$,
Step 3: Calculate $\tilde{P}_\rho \leftarrow \epsilon_{\rho-1}\beta + P_\rho$,
Step 4: Round $\tilde{P}_\rho$ to $\tilde{r}_{\rho-\delta}$,
Step 5: Calculate $\epsilon_\rho \leftarrow \tilde{P}_\rho - \tilde{r}_{\rho-\delta}$,
Step 6: Calculate $\rho \leftarrow \rho + 1$,
Step 7: If $\rho < W$, then go to Step 2, else done.
```

In the above algorithm, the online error and the iteration number are initialized to 0. At a given iteration ρ, an off-line partial addition update P_ρ is calculated, by using the operand digits a_ρ and b_ρ. Then, P_ρ is combined to the online error $\epsilon_{\rho-1}$ generated at the previous iteration, yielding an online partial addition update $\widetilde{P}_\rho$. Next, $\widetilde{P}_\rho$ is rounded to the online sum digit $\widetilde{r}_{\rho-\delta}$, giving rise to an online error ϵ_ρ. This rounding operation can be performed on a small number of the MSDs of $\widetilde{P}_\rho$, increasing the speed of a corresponding hardware implementation. The iteration number is then incremented, and Steps 2 to 6 are performed until $\rho = W - 1$. It can be shown that at the end of iteration $\rho = W - 1$, the online iterative sum $\widetilde{R}_{W-1}$ represents the MSDs, and the online error ϵ_{W-1} represents the LSDs of R.

The error between the exact sum and the online sum is caused by the absence of knowledge of $A - A_\rho$ and $B - B_\rho$, and by the online error ϵ_ρ. In the algorithm, the former error is reduced at each iteration by taking into account the off-line partial update P_ρ. The latter error changes at each iteration, and must be kept small enough so that the addition of P_ρ to the scaled online error does not affect the online result digits outputted in previous iterations. This problem is further complicated by performing the rounding operation for online result digit estimation on a small number of MSDs of $\widetilde{P}_\rho$, but can be circumvented by increasing the latency. As a result, the range of P_ρ becomes smaller, reducing the range of $\widetilde{P}_\rho$ (as desired). However, the latency should be kept small in order to reduce the wordlength of the result. For a given arithmetic operation, one can calculate a minimum bound on the latency. Then, this bound can be used to determine the optimal trade-off between the delay of a corresponding implementation, which largely depends on the rounding operation (McQuillan and McCanny, 1995) and the latency of the operation.

2.4 High-Speed Signed-Binary Addition

This section is concerned with an introduction to limited-carry addition schemes in redundant binary number systems, and with the development of corresponding bit-serial and digit-serial online addition architectures. In particular, the limited-carry redundant binary addition schemes will be observed to share a common bit-parallel architecture. A systematic enumeration of all the corresponding addition schemes will extract and highlight their similar behaviors. As a result, one will be able to employ an existing architecture for addition in a given number system as an addition architecture in a different number system.

2.4.1 Redundant Binary Addition Schemes

The addition of two signed-binary (SB) words can be achieved in constant time, independently of the wordlengths of the inputs (Avizienis, 1961). The implementation of such adders was investigated in (Chow and Robertson, 1978) by analyzing all possible digit encodings and by deriving the corresponding boolean equations, yielding efficient architectures. Then, a recoding approach was taken in (Parhami, 1988; Kornerup, 1994) for the development of SB adders. Recently, in (Thornton, 1997), two weight-transfer digit decompositions were applied to perform the addition of SB numbers, where the binary code of the result always features an even number of 1's, referred to as inherent parity. Such a property can be used for embedded correct functionality testing.

In the following, it is shown that all the above approaches are variations of the same approach, resulting in similar architectures. Therefore, one will characterize the redundant binary addition schemes that yield these constant-delay architectures by using two parameters, one for the redundant binary digit-set, and one for the addition scheme itself. In this way, it is shown that the similarity between redundant binary addition schemes can indeed be exploited to map two different schemes to the same architecture through appropriate digit encoding. Due to the length of the proofs, the theorems and lemmas are stated, but the proofs are presented in appendix.

Carry-Free and Limited-Carry Addition

Addition can be performed in constant time when a redundant number representation is used (Avizienis, 1961; Parhami, 1990). The corresponding addition schemes can be qualified as either carry-free or limited-carry. In a carry-free addition scheme, the digits of equal weight of the addend and augend words are added in parallel, and the result digit is decomposed into weight and transfer digits, where the transfer digit bears the next higher weight. Then, the weight and transfer digits are added (recomposed), yielding the digits of the sum in the desired number representation. The transfer digit, also called a carry, has thus been absorbed by the recomposition, resulting in an absence of carry propagation.

Both the decomposition and recomposition can be performed in constant time by one level of hardware cells. This addition scheme is referred to as weight-transfer decomposition (Parhami, 1990; Rao and Nowrouzian, 1999), and as digit set conversion (Kornerup, 1994). It can be observed that the cardinality of the digit set of the result representation is smaller than that of the input representation. In a limited-carry addition scheme, two such conversions are performed successively in order to perform the constant-delay addition, resulting in a three-level architecture.

It can be observed that in carry-free addition, a change in an input digit of weight β^{-i} only propagates to the output sum digit of weight β^{-i}, or to that of weight β^{-i+1}. Consequently, in limited-carry addition, a change in an input digit of weight β^{-i} may propagate to the result digits of weights β^{-i}, β^{-i+1}, and β^{-i+2}.

Addition in Redundant Binary Number Systems

The redundant binary number systems form three distinct radix-2 GSD number systems having a digit-set of cardinality 3. These digit-sets are $\{0, 1, 2\}$, $\{-1, 0, 1\}$, and $\{-2, -1, 0\}$ for the binary carry-save (Parhami, 1990; Kornerup, 1994), the SB (Avizienis, 1961) and the negative binary carry-save number systems. The redundant binary number systems require limited-carry addition schemes because they do not feature enough representation redundancy (Parhami, 1990). The binary carry-save number system is complete (Kornerup, 1994) for $\mathbf{N}^+$, the SB number system is complete for $\mathbf{N}$, and the negative carry-save number system is complete for $\mathbf{N}^-$. Of course, it can be observed that the negative binary carry-save number system does not find practical applications since it can only yield representations for negative numbers.

The constant-delay property of addition in the redundant binary number systems has triggered their widespread use within binary multipliers. As an example, the multipliers introduced in (Wallace, 1964; Dadda, 1976; Larsson and Nicol-Chris, 1996) use the carry-save number system. Similarly, the SB number system has been widely used for the design of non-redundant as well as redundant number multipliers (Lapointe et al., 1993; Rao and Nowrouzian, 1997). The similarities of the addition schemes obtained here are valid for all redundant binary number systems, but are expected to be exploited only with the SB and binary carry-save number systems.

Generic Architecture and Corresponding Scheme for Redundant Binary Addition

Let us consider the result S of the sum of an addend A to an augend B in accordance with

$$S = A + B. \tag{2.55}$$

The operands A and B, and the result S are assumed to be represented in the same redundant binary number system having a contiguous digit set $\{\gamma-1, \gamma, \gamma+1\}$, where $\gamma \in \{\bar{1}, 0, 1\}$ so that the corresponding number system is complete (Kornerup, 1994). Therefore, by recalling the limited-carry property of the addition schemes under consideration, one can write

$$A = \sum_{i=0}^{W-1} a_i 2^{-i}; \quad a_i \in \{\gamma - 1, \gamma, \gamma + 1\} \tag{2.56}$$

$$B = \sum_{i=0}^{W-1} b_i 2^{-i}; \quad b_i \in \{\gamma - 1, \gamma, \gamma + 1\} \tag{2.57}$$

and

$$S = \sum_{i=-2}^{W-1} s_i 2^{-i}; \quad s_i \in \{\gamma - 1, \gamma, \gamma + 1\} \tag{2.58}$$

where W represents the wordlength of A and B. In the following, all the limited-carry addition schemes which can calculate R are enumerated by determining all the possible sets for the weight and transfer digits for each conversion.

Since any limited-carry addition scheme can be implemented by using three levels of hardware cells, one can always group them as shown in Figure 2.5. This three-level architecture will be employed for all addition schemes. The variables c_1, c_1', c_2, and c_2', are referred to as the weight digits, and α and β as the transfer digits of the addition scheme. The addition scheme corresponding to Figure 2.5 is expressed in accordance with

$$\alpha + c_1' = a + b, \tag{2.59}$$

$$\beta + c_2' = \alpha + 2c_1, \tag{2.60}$$

and

$$4s = \beta + 2c_2, \tag{2.61}$$

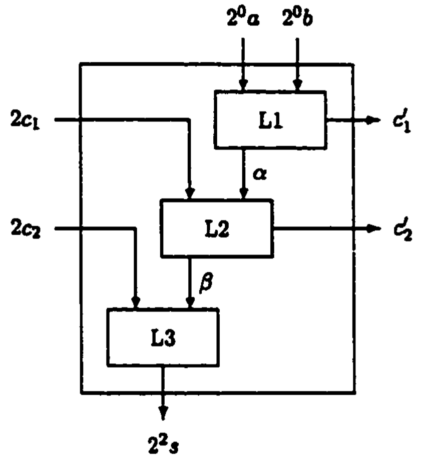


Figure 2.5: Limited Carry Adder Unit

respectively, as shown in Figure 2.5. It is necessary that the carries c_1 and c_2 be given a weight 2. In this way, one integrates in the successive additions the fact that they are generated by a similar architecture processing digits of next higher power of two than a and b.

For the sake of uniformity, any variable x is assumed to belong to a set denoted by S_x. Of course, both c_1 and c'_1 belong to the same set S_{c_1} to allow the above architecture to be cascaded to form a bit-parallel addition architecture. In such a bit-parallel architecture, the single-digit architectures can be manipulated to use α and β as carries (transfer digits) and c_1 and c_2 as internal weight digits, changing the apparent carry propagation from MSD-first to LSD-first. Similarly, c_2 and c'_2 belong to the same set S_{c_2}. It is important to note that the architecture in Figure 2.5 corresponds to a particular case of online addition, where the latency is equal to 2, and where the online error is obtained as a sum of two components, c'_1 and c'_2. Therefore, by decomposing the online error, it is possible to obtain a constant-delay online addition.

2.4.2 Characterization and Equivalence of Redundant Binary Addition Schemes

In the following, it is shown that the hardware implementation of the architecture shown in Figure 2.5 can perform several addition schemes employing different number systems. This proof is obtained by first characterizing the addition schemes by using two parameters, followed by defining an equivalence relationship between them, and by finally observing the resulting similarities and exploiting them for the desired purpose.

Characterization of Redundant Binary Addition Schemes

If one denotes the cardinality of a set S by $|S|$, then one can readily observe that, by assumption,

$$|S_a| = 3,\ |S_b| = 3, \text{ and } |S_s| = 3. \tag{2.62}$$

Moreover, the calculation of the left-hand sides of Eqns. 2.59—–2.61 in terms of their corresponding right-hand sides are only possible if the following constraints hold:

$$5 = \left|S_a \dot{+} S_b\right| \leqslant \left|S_{c_1} \dot{+} S_\alpha\right|, \tag{2.63}$$

$$\left|S_\alpha \dot{+} 2S_{c_1}\right| \leqslant \left|S_{c_2} \dot{+} S_\beta\right|, \tag{2.64}$$

and

$$\left|S_\beta \dot{+} 2S_{c_2}\right| \leqslant 4\,|S_s| = 3. \tag{2.65}$$

It can be shown that only two sets of cardinality values are allowed by Eqns. 2.63–2.65, as shown in Table 2.1 (see Appendix A for a proof). In both cases, it is possible to determine

Table 2.1: Set Cardinalities in Redundant-Binary Addition Schemes

Set	S_a	S_b	S_{c_1}	S_α	S_{c_2}	S_β	S_s
Cardinality	3	3	2	3	2	2	3
Cardinality	3	3	3	2	2	2	3

the values of the elements of the sets as functions of γ and of an offset parameter Δ as given in Table 2.2 (again, see Appendix A for a proof).

Theorem 15 *Given any redundant binary addition scheme, there exists $\gamma \in \{-1, 0, 1\}$ and $\Delta \in \mathbb{R}$ such that $S_a = S_b = S_s = \{\gamma-1, \gamma, \gamma+1\}$, and that the weight and transfer digit-sets of the addition scheme are as summarized in case 1, 2, or 3 of Table 2.2.*

Proof. The proof is given in Appendix A. ■

The above theorem proves that there is an infinite number of redundant binary addition schemes, but that they are characterized by only two parameters, one related to the digit

Table 2.2: Parametrized Digit Sets of Redundant Binary Addition Schemes

Set	No. 1	No. 2	No. 3
S	$\{\gamma-1,\gamma,\gamma+1\}$	$\{\gamma-1,\gamma,\gamma+1\}$	$\{\gamma-1,\gamma,\gamma+1\}$
S_{c_1}	$S\dot{+}\{\Delta\}$	$\{\gamma-1,\gamma\}\dot{+}\{\Delta\}$	$\{\gamma,\gamma+1\}\dot{+}\{\Delta\}$
S_α	$\{\gamma-1,\gamma+1\}\dot{+}\{-\Delta\}$	$\{\gamma-1,\gamma+1,\gamma+3\}\dot{+}\{-\Delta\}$	$\{\gamma-3,\gamma-1,\gamma+1\}\dot{+}\{-\Delta\}$
S_{c_2}	$\{\gamma-1,\gamma+1\}\dot{+}\{-\Delta\}$	$\{\gamma-1,\gamma+1\}\dot{+}\{-\Delta\}$	$\{\gamma-1,\gamma+1\}\dot{+}\{-\Delta\}$
S_β	$2\{\gamma-1,\gamma+1\}\dot{+}2\{\Delta\}$	$2\{\gamma-1,\gamma+1\}\dot{+}2\{\Delta\}$	$2\{\gamma-1,\gamma+1\}\dot{+}2\{\Delta\}$

set of the number system (γ), and the other related to the addition scheme itself (Δ). It is important to note that the result does not depend on Δ, because it is introduced by the first conversion and is cancelled by the second.

Equivalence of Redundant Binary Addition Schemes

Definition 16 *Relationship between addition schemes: consider the redundant binary addition schemes A with parameters (γ,Δ), and A' with parameters (γ',Δ'). Then, A and A' are related through $\mathcal{R}$ if and only if there exists ϵ in $\{\bar{1},1\}$ such that*

$$S'_a = \epsilon(S_a \dot{+} \{-\gamma\}) \dot{+} \{\gamma'\}, \tag{2.66}$$

$$S'_b = \epsilon(S_b \dot{+} \{-\gamma\}) \dot{+} \{\gamma'\}, \tag{2.67}$$

$$S'_s = \epsilon(S_s \dot{+} \{-\gamma\}) \dot{+} \{\gamma'\}, \tag{2.68}$$

$$S'_{c_1} = \epsilon(S_{c_1} \dot{+} \{-(\gamma+\Delta)\}) \dot{+} \{\gamma'+\Delta'\}, \tag{2.69}$$

$$S'_{c_2} = \epsilon(S_{c_2} \dot{+} \{-(\gamma-\Delta)\}) \dot{+} \{\gamma'-\Delta'\}, \tag{2.70}$$

$$S'_\alpha = \epsilon(S_\alpha \dot{+} \{-(\gamma-\Delta)\}) \dot{+} \{\gamma'-\Delta'\}, \tag{2.71}$$

and

$$S'_\beta = \epsilon(S_\beta \dot{+} \{-2(\gamma+\Delta)\}) \dot{+} \{2(\gamma'+\Delta')\}. \tag{2.72}$$

This is equivalently denoted by

$$A\,\mathcal{R}\,A'. \tag{2.73}$$

In Appendix A, a lemma is given which shows that $\mathcal{R}$ is an equivalence relationship. It can be readily observed that a necessary condition for two addition schemes to be related through $\mathcal{R}$ is given by

$$|S'_v| = |S_v| \quad \forall v \in \{c_1, \alpha\} \tag{2.74}$$

It can also be shown that the condition is also sufficient, thus leading to the following theorem.

Theorem 17 *There are two equivalence classes for $\mathcal{R}$ in the proposed space of addition schemes: C_{32} for which $|S_{c_1}| = 3$ and $|S_\alpha| = 2$, and C_{23} for which $|S_{c_1}| = 2$ and $|S_\alpha| = 3$. In other words, any redundant binary addition scheme is either related to any member of class C_{32}, or to any member of class C_{23}.*

Proof. The proof is established in Appendix A. ■

Several implementations of schemes of class C_{32} have been implemented in the past (Chow and Robertson, 1978; Thornton, 1997). Two addition schemes of class C_{23} have been implemented in (Thornton, 1997). As a consequence of Theorem 17, one can state the following:

Theorem 18 *Let A and A' be two two-level redundant binary addition schemes such that*

$$A \,\mathcal{R}\, A'. \tag{2.75}$$

If there exists a circuit that implements A, then the same circuit can be used to perform A'.

Proof. The proof is established in Appendix A. ■

Theorem 18 is very important as it states that a hardware implementation of a given addition scheme can be re-used for another equivalent addition scheme. It is crucial to note that this equivalence can be obtained from one redundant binary representation to another, i.e. a binary carry-save adder and a signed-binary adder can be translated to the same architecture.

Let us now give the example of a circuit for addition scheme A with parameters $(\gamma, \Delta) = (0, \bar{1})$ and addition scheme A' with parameters $(\gamma, \Delta) = (0, 0)$. By applying Theorem 17, one obtains that $A\mathcal{R}A'$. The numbers contained in the various sets are coded as shown in Table 2.3.

Table 2.3: Coding the Set Values

Addition	$\{\bar{1},0,1\}$	S_{c_1}	S_α	S_{c_2}	S_β
A	$\bar{1} \to 00$	$\bar{2} \to 0$	$0 \to 00$	$0 \to 0$	$\bar{4} \to 0$
	$0 \to (01,10)$		$2 \to (01,10)$		
	$1 \to 11$	$\bar{1} \to 1$	$4 \to 11$	$2 \to 1$	$0 \to 1$
A'	$\bar{1} \to 11$	$0 \to 1$	$\bar{3} \to 11$	$\bar{1} \to 1$	$\bar{2} \to 1$
	$0 \to (01,10)$		$\bar{1} \to (01,10)$		
	$1 \to 00$	$1 \to 0$	$1 \to 00$	$1 \to 0$	$2 \to 0$

The codes for the elements of the sets of A are arbitrarily chosen. Every element of a set of A' is given the same code as its corresponding element of a set of A by using the appropriate transformation. Then, the conversion tables used for A are given below

$a+b$	α	c_1'
$\bar{2}$	0	$\bar{2}$
$\bar{1}$	0	$\bar{1}$
0	2	$\bar{2}$
1	2	$\bar{1}$
2	4	$\bar{2}$

$\alpha + 2 \times c_1$	β	c_2'
$\bar{4}$	$\bar{4}$	0
$\bar{2}$	$\bar{4}$	2
0	0	0
2	0	2

and the conversion tables used for A' are given below

$a+b$	α	c_1'
$\bar{2}$	$\bar{3}$	1
$\bar{1}$	$\bar{1}$	0
0	$\bar{1}$	1
1	1	0
2	1	1

$\alpha + 2 \times c_1$	β	c_2'
$\bar{3}$	$\bar{2}$	$\bar{1}$
$\bar{1}$	$\bar{2}$	1
1	2	$\bar{1}$
3	2	1

By replacing the digits by their binary codes, one obtains the same conversion tables, the same Karnaugh maps, and the same circuit.

A Limited-Carry Adder: The 4:2 Compressor

The 4:2 compressor, among many other counters, was introduced in (Dadda, 1976). Plenty of research has been carried out to use and optimize the 4:2 compressor (Law et al., 1999; Hagihara et al., 1998; Shim and Kim, 1997; Goto et al., 1997; Pillai et al., 1996; Larsson and Nicol-Chris, 1996; Kanie et al., 1994). Its primary application involved the reduction of trees of binary numbers, arising in the multiplication of binary numbers. A 4:2 compressor adds four bits of weight 1 with a carry-in of weight 1 and generates one bit of weight 1 and one bit of weight 2 and a carry-out of weight 2. Its main feature is that the carry-in has no influence on the carry-out, permitting a cascade of 4:2 compressors to perform addition in constant time (Dadda, 1976). Let us regroup the input bits into two sets of two, to form two binary carry-save digits. Moreover, let us transmit the output bit of weight 2 as a carry-out to the 4:2 compressor processing the bits of the next higher power of two. The second carry-in can thus be output directly, and combined with the existing output bit to form a binary carry-save sum digit. As a result, the 4:2 compressor has been transformed into a LSD-first limited-carry binary carry-save adder. Therefore, the 4:2 compressor is also a SB adder.

The systematic development of SB adders for any given SB digit encoding has been investigated (Chow and Robertson, 1978). In that article, it was proven that a SB adder corresponding to the 4:2 compressor is one of the three best SB adders in terms of delay and area. Consequently, a 4:2 compressor hardware implementation proposed recently was chosen for subsequent hardware implementation of the limited-carry SB adders (Kanie et al., 1994).

2.4.3 Bit-Serial and Digit-Serial Signed-Binary Limited-Carry Addition Architectures

Consider the online limited-carry addition of an addend word A to an augend word B given by

$$A = \sum_{i=1}^{W} a_i 2^{-i} \quad \text{and} \quad B = \sum_{i=1}^{W} b_i 2^{-i}; \quad (a_i, b_i) \in \{-1, 0, 1\}^2, \tag{2.76}$$

where the digits a_i and b_i are inputted at time instant i. The resulting sum S is of the form

$$S = \sum_{i=-1}^{W} s_i 2^{-i}; \quad s_i \in \{-1, 0, 1\}. \tag{2.77}$$

A corresponding bit-serial architecture is shown in Figure 2.6. This architecture implements Algorithm 14 (page 33) for bit-serial online addition. The result digit $\tilde{r}_{\rho-\delta}$ corresponds to the sum digit s, the online error ϵ_ρ corresponds to $c_2' + c_1'$, but the partial update P_ρ cannot be identified directly. This architecture has two modes of operation, namely, addition

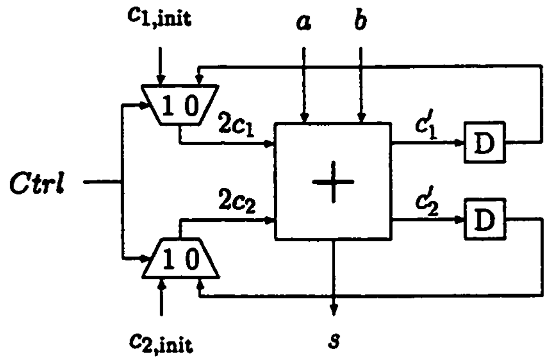


Figure 2.6: Architecture for Bit-Serial Online Limited-Carry Addition

initialization and continuing addition. During the addition initialization mode, one assumes that the digits of A and B have been set to zero for all the past time instants. The *Ctrl* signal is set to 1 to select the resulting hardwired initialization carry values. The MSDs a_1 of A and b_1 of B are added to the initialization carry values $c_{1,\text{init}}$ and $c_{2,\text{init}}$, yielding the MSD s_{-1} of S, in addition to the carries that are stored for use in the next time instant. During the continuing addition mode, the digits a_i and b_i are added to the carries c_1 and c_2 formed at the previous time instant, yielding the digit s_{i-2} and the carries c'_1 and c'_2 that are stored for use in the next time instant. This implies a latency of 2 for the online addition (c.f. Section 1.3.2).

It can be observed that the initialization mode occurs every W time instants. Therefore, the *Ctrl* signal is equal to 1 for the (discrete) time instant n being divisible by W, i.e. when

$$n \bmod W \equiv 0. \tag{2.78}$$

It is important to note that the carries formed at the previous time instant are discarded in the addition initialization mode. By padding both the addend and the augend with two zeros as their LSDs, all the digits of accuracy of the sum will be made available, and the carries will be set to their initialization values (Natter and Nowrouzian, 1999). Consequently, the switches are not required. Therefore, the cost of padding two zeros as the LSDs of both the addend and the augend has to be compared to the cost of using switches.

The above adder originates from a very efficient 4:2 compressor developed in (Kanie et al., 1994), where the modifications introduced solely consist of wire re-routings. The SB input digit a is represented by a bit pair $({}^1a, {}^2a)$ so that

$$a = a_1 + a_2 - 1; \quad {}^1a, {}^2a \in \{0, 1\}. \tag{2.79}$$

The SB input digit b is similarly represented by a bit pair $({}^1b, {}^2b)$ so that

$$b = b_1 + b_2 - 1; \quad {}^1b, {}^2b \in \{0, 1\}. \tag{2.80}$$

This SB digit encoding was presented in (Lapointe et al., 1993). As in Figure 2.5, a and b are combined to yield $\alpha \in \{-2, 0\}$, and the output carry $c'_1 \in \{0, 1, 2\}$ so that Eqn. 2.59 (page 37) holds. One represents α by a single bit ${}^1\alpha$ in accordance with

$$\alpha = 2({}^1\alpha - 1); \quad {}^1\alpha \in \{0, 1\}, \tag{2.81}$$

and c'_1 by a pair of bits $({}^1c'_1, {}^2c'_1)$ as per Table 2.4. The $|$, $\oplus$, $\frown$, and $\overline{\cdot}$ symbols represent

Table 2.4: Code for c_1 and c'_1

Digit	Code $({}^1c_1\ {}^2c_1)$
0	0 0
1	1 0 or 1 1
2	0 1

the logic OR, XOR, AND, and NOT operations, respectively. The above bits representing α and c'_1 are calculated in accordance with

$$\begin{aligned} {}^1\alpha &= \overline{{}^1a \oplus {}^2a} \frown {}^1a \,|\, ({}^1a \oplus {}^2a) \frown {}^1b, && (2.82) \\ {}^1c'_1 &= ({}^1a \oplus {}^2a) \oplus ({}^1b \oplus {}^2b), \text{ and} && (2.83) \\ {}^2c'_1 &= b_2, && (2.84) \end{aligned}$$

respectively. Then, α and c_1 are combined to yield $\beta \in \{0, 4\}$ and $c'_2 \in \{-2, 0\}$, as given by Eqn. 2.60. The representation of c_1 by a bit pair $({}^1c_1, {}^2c_1)$ is identical to that of c'_1. One represents β by a single bit ${}^1\beta$ so that

$$\beta = 4\,{}^1\beta; \quad {}^1\beta \in \{0, 1\} \tag{2.85}$$

and c_2' by a single bit ${}^1c_2'$ so that

$$c_2' = 2({}^1c_2' - 1); \quad {}^1c_2' \in \{0,1\}. \tag{2.86}$$

These bits are calculated in accordance with

$${}^1\beta = \overline{{}^1c_1} \frown {}^2c_1 \,|\, {}^1c_1 \frown {}^1\alpha \tag{2.87}$$

and

$${}^1c_2' = {}^1\alpha \oplus {}^1c_1, \tag{2.88}$$

respectively. Finally, β and c_2 are combined to yield the SB output sum digit s in accordance with Eqn. 2.61. One represents s by a bit pair $({}^1s, {}^2s)$ so that

$$s = {}^1s + {}^2s - 1; \quad {}^1s, {}^2s \in \{0,1\}, \tag{2.89}$$

and c_2 by a single bit ${}^1c_2$ similarly to c_2'. Subsequently, the sum digit s is calculated by using

$${}^1s = {}^1c_2 \text{ and } {}^2s = {}^1\beta. \tag{2.90}$$

From Eqns. 2.82–2.90 and Table 2.4, one can obtain the following bit representations of the initialization carry values

$$({}^1c_{1,\text{init}}, {}^2c_{1,\text{init}}) = (0,1) \text{ and } {}^1c_{2,\text{init}} = 0. \tag{2.91}$$

Digit-Serial Architecture

By using Algorithm 10, one obtains a generic digit-serial architecture of digit-size D for signed-binary limited-carry addition, as shown in Figure 2.7. At each time instant, D input digits are made available in both the addend A and augend B streams at each time instant, resulting in the generation of D corresponding digits that are outputted by the sum S stream. The signal $Ctrl_i$ is set when the (discrete) time instant n satisfies

$$(n \times D + i) \bmod W \equiv 0, \tag{2.92}$$

where i corresponds to the single-digit addition instance number.

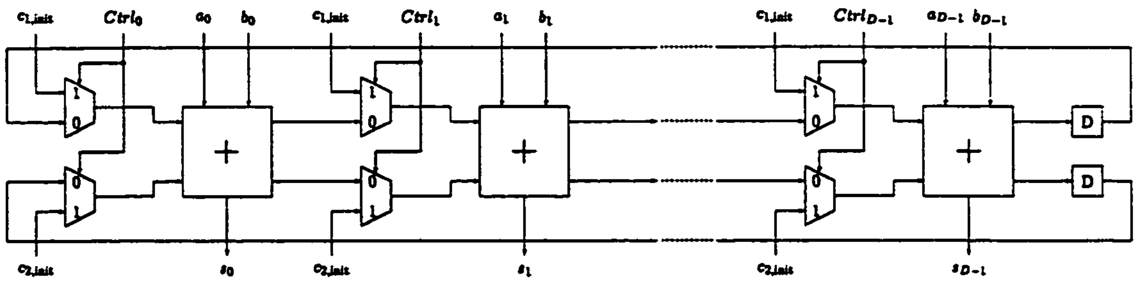


Figure 2.7: Architecture for Digit-Serial Online Limited-Carry Addition

2.5 Chapter Summary

In this chapter, all redundant binary addition schemes have been found and characterized. Four of these schemes were previously reported in (Thornton, 1997). All schemes belong to two distinct classes, and a theorem has been given stating that the hardware implementation of one of these addition schemes can be used for any addition scheme belonging to the same class. The importance of this theorem lies in the facts that, (a) only two classes of three-level redundant binary addition schemes exist, and (b) all different implementations of one addition scheme can be used for all other members of the same class as they share the same internal mechanisms. This implies that an addition scheme with input and output digit sets $\{0, 1, 2\}$ can be implemented with the same circuit as an addition scheme with input and output digit sets $\{-1, 0, 1\}$, or with input and output digit sets $\{-2, -1, 0\}$.

This chapter has also introduced the necessary background for digit-serial online operations. The digit-serial unfolding algorithm has been simplified by merging two steps into one. Moreover, the digit-serial technique has been extended to the case of a dynamically changing wordlengths. Similarities between number systems have been identified, which can be exploited for design time savings, in particular for redundant binary number systems, where the well-documented 4:2 compressor has been proven to be a limited-carry signed-binary adder. Finally, the proposed techniques have been applied to the development of a digit-serial online limited-carry addition architecture.

Chapter 3

Theoretical Background for Online Signed-Digit Multiplication and Multiply-Accumulate Operations

3.1 Introduction

With the advent of the communication era, digital signal processing is rapidly gaining popularity, mainly because of the flexibility and cost-efficiency of corresponding hardware implementations. The multiplication and multiply-accumulate operations are essential for digital signal processors. In such an operation, the result R is obtained in accordance with

$$R = AB + C, \tag{3.1}$$

where A, B, and C represent the multiplicand, multiplier, and addend, respectively[1].

This chapter is concerned with an introduction to the theoretical background for the development of architectures for online signed-digit multiplication and multiply-accumulate operations. The discussions begin with the development of a general algorithm for bit-serial online signed-digit multiply-accumulate operation (Section 3.2). The salient feature of the proposed algorithm is to have the operation depend on one single variable only, namely the partial update, while all the other variables are operation-independent. In this way, the resulting algorithm can perform any operation expressed as a sum of partial operation updates in a bit-serial online fashion. Then, in Section 3.3, an architecture unit for the multiplication of a signed-binary word by a binary and a signed-binary digit is presented together with a novel signed-binary to minimally redundant base-4 recoding technique. The

[1] A multiplication corresponds to the special case of a MAC operation with $C = 0$.

proposed recoding technique permits a speed-efficient reduction of the number of partial updates, substantially increasing the processing speed of a corresponding architecture. As the format of the inputs to such arithmetic operations cannot be naturally preserved for their outputs, the discussions continue in Section 3.3 with a description of the existing techniques for the rounding of signed-binary numbers and the overflow handling in online operations. In particular, algorithms are given for parallel low-precision rounding of a signed-binary word and for online signed-binary rounding—compliant with the IEEE 754 rounding to nearest/even standard.

3.2 Proposed Algorithm for Signed-Digit Online MAC Operation

In this section, a generalized recursion-based algorithm for signed-digit online MAC operation is developed, where the multiplicand is assumed to be known at the outset, and where the multiplier and addend are made available in an online fashion. The resulting algorithm obtains an increasingly more accurate online MAC result as digits of decreasing weight of the multiplier and addend are made available. For the sake of generality, the redundancy indices of the MAC operation operands (i.e. the multiplicand, the multiplier, and the addend) are all assumed to be different. A strategy for the determination of the various algorithm parameters is presented. This strategy relies on a bound on the latency of the resulting algorithm together with a bound on the difference between the indices of the most significant digits of the multiplier and addend.

3.2.1 Nomenclature

It is assumed that the MAC operation operands (A, B, C, and the final MAC result R) are represented in a fixed-point radix-β ordinary signed-digit (OSD) number system

(Parhami, 1990) in accordance with

$$A = \sum_{i=i_a}^{W_a+i_a-1} a_i\beta^{-i}; \quad a_i \in \{-\eta_a, \ldots, \eta_a\}, \tag{3.2}$$

$$B = \sum_{i=i_b}^{W_b+i_b-1} b_i\beta^{-i}; \quad b_i \in \{-\eta_b, \ldots, \eta_b\}, \tag{3.3}$$

$$C = \sum_{i=i_c}^{W_c+i_c-1} c_i\beta^{-i}; \quad c_i \in \{-\eta_c, \ldots, \eta_c\}, \tag{3.4}$$

and

$$R = \sum_{i=i_r}^{W_r+i_r-1} r_i\beta^{-i}; \quad r_i \in \{-\eta_r, \ldots, \eta_r\}. \tag{3.5}$$

Here W_a, W_b, W_c, and W_r represent the wordlengths of A, B, C, and R, respectively, i_a, i_b, i_c, and i_r represent the indices of their MSDs, and $\left\lceil \frac{\beta}{2} \right\rceil \leq \eta_a, \eta_b, \eta_c, \eta_r \leq \beta - 1$ represent their redundancy indices. An estimation of a number X is denoted by $\widetilde{X}$, and its truncation by $\overline{X}$. The MSDs of A and B can be assumed to have the same weight β^{-1} as made possible by proper scaling in Eqn. 3.1, implying that i_a and i_b are equal to 1. Consequently, the parameter i_c indicates the relative position of the MSDs of AB and C (Natter and Nowrouzian, 2000a).

For an OSD representation of A, l_a can be calculated in accordance with

$$l_a = \frac{\eta_a}{\beta - 1}(1 - \beta^{-W_a}). \tag{3.6}$$

Since $\eta_a \leq \beta - 1$, one has

$$l_a < 1. \tag{3.7}$$

Therefore, the above representations restrict the ranges of the multiplicand A and multiplier B so as to permit one to represent AB in the same way as A (after rounding and overflow processing).

An online operation is typically performed in an iterative manner, where the online input digits are available one by one, the MSD first. By taking into account the online mode of the arrival of the multiplier and addend,

$$B_\rho = \sum_{i=1}^{\rho} b_i\beta^{-i} \text{ and } C_\rho = \sum_{i=i_c}^{i_c+\rho} c_i\beta^{-i} \tag{3.8}$$

represent the partially formed B and C at iteration ρ. The corresponding off-line iterative MAC result is defined as

$$R_\rho = AB_\rho + C_\rho. \tag{3.9}$$

The MAC result digits are assumed to be available online, forming an online iterative MAC result $\tilde{R}_\rho$ at iteration ρ as given by

$$\tilde{R}_\rho = \tilde{R}_{\rho-1} + \tilde{r}_{\rho-\delta}\beta^{\delta-\rho}, \tag{3.10}$$

where δ represents the latency of the MAC operation. The off-line iterative MAC result can be recursively defined so that

$$R_\rho = R_{\rho-1} + P_\rho\beta^{\delta-\rho}, \tag{3.11}$$

where P_ρ represents an off-line partial MAC update (as determined in terms of the newly arriving b_ρ and $c_{i_c+\rho}$ digits) at iteration ρ. By padding B or C with zeros, one can set $W_b = W_c \equiv W$, with $\rho \in \{1, \ldots, W\}$.

As formed by the accumulation of off-line partial MAC updates of large wordlength, R_ρ has more digits of accuracy than $\tilde{R}_\rho$. In this way, $\tilde{R}_\rho$ approximates R_ρ with some online error ϵ_ρ such that

$$R_\rho = \tilde{R}_\rho + \epsilon_\rho\beta^{\delta-\rho}. \tag{3.12}$$

The iterative MAC result R_ρ can be expressed as a function of $\tilde{R}_{\rho-1}$ in accordance with

$$R_\rho = \tilde{R}_{\rho-1} + \tilde{P}_\rho\beta^{\delta-\rho} \tag{3.13}$$

where $\tilde{P}_\rho$ represents an online partial MAC update. In the following, recursion equations are developed, (a) to obtain $\tilde{P}_\rho$ in terms of the off-line partial MAC update P_ρ and the scaled online error $\epsilon_{\rho-1}\beta$, and (b) to determine $\tilde{r}_{\rho-\delta}$ in terms of online partial MAC update $\tilde{P}_\rho$ and the online error ϵ_ρ.

3.2.2 Signed-Digit Online MAC Algorithm

The iterative MAC result R_ρ is given by

$$R_\rho = R_{\rho-1} + Ab_\rho\beta^{-\rho} + c_{i_c+\rho}\beta^{-(i_c+\rho)}. \tag{3.14}$$

Therefore, $P_\rho = \left(Ab_\rho + c_{i_c+\rho}\beta^{-i_c}\right)\beta^{-\delta}$ is an off-line partial MAC update value determined by the newly arriving b_ρ and $c_{i_c+\rho}$ digits of the multiplier B and addend C, respectively. The bound on P_ρ is independent of ρ as it is scaled by the same weight as $\widetilde{r}_{\rho-\delta}$, and can thus be used as a signal in a corresponding hardware implementation.

If Eqn. 3.12 holds at iteration $\rho - 1$, then expanding $R_{\rho-1}$ in Eqn. 3.11 leads to

$$R_\rho = \widetilde{R}_{\rho-1} + (\epsilon_{\rho-1}\beta + P_\rho)\,\beta^{\delta-\rho}. \tag{3.15}$$

The bound on ϵ_ρ is also independent of ρ. By substituting $\widetilde{R}_\rho$ from Eqn. 3.10 into Eqn. 3.12, one obtains

$$R_\rho = \widetilde{R}_{\rho-1} + (\widetilde{r}_{\rho-\delta} + \epsilon_\rho)\beta^{\delta-\rho}. \tag{3.16}$$

Finally, by invoking Eqns. 3.15, 3.16, and 3.13, one arrives at the recursion relationships

$$\widetilde{P}_\rho = \epsilon_{\rho-1}\beta + P_\rho, \tag{3.17}$$

and

$$\widetilde{r}_{\rho-\delta} + \epsilon_\rho = \widetilde{P}_\rho. \tag{3.18}$$

The online MAC result digit $\widetilde{r}_{\rho-\delta}$ is an integer estimate of $\widetilde{P}_\rho$ to within an online error ϵ_ρ. Rounding $\widetilde{P}_\rho$ can be restricted to rounding $\epsilon_{\rho-1}\beta$, as in (McQuillan and McCanny, 1995). The recursion relationships in Eqns. 3.17 and 3.18 can be recast into the following algorithm for purely signed-digit online MAC operation.

Algorithm 19 *Online Ordinary Signed-Digit MAC Operation*

```
Step 1 Set $\rho \leftarrow 1$ and $\epsilon_0 \leftarrow 0$,
Step 2 Read $b_\rho$ and $c_{i_c+\rho}$,
Step 3 Calculate $P_\rho \leftarrow \left(Ab_\rho + c_{i_c+\rho}\beta^{-i_c}\right)\beta^{-\delta}$,
Step 4 Calculate $\widetilde{P}_\rho \leftarrow \epsilon_{\rho-1}\beta + P_\rho$,
Step 5 Round $\widetilde{P}_\rho$ to $\widetilde{r}_{\rho-\delta}$,
Step 6 Calculate $\epsilon_\rho \leftarrow \widetilde{P}_\rho - \widetilde{r}_{\rho-\delta}$,
Step 7 Write $\widetilde{r}_{\rho-\delta}$,
Step 8 Calculate $\rho \leftarrow \rho + 1$,
Step 9 If $\rho < W + 1$, then go to Step 2, else done end if.
```

In the above algorithm, after the initialization of the online error ϵ_0, the multiplier B and addend C are consumed in an online fashion. At a given iteration ρ ($1 \leq \rho \leq W$), an off-line partial MAC result P_ρ is formed by using the most recent online digits of B and C. Then, one adds the scaled error $\epsilon_{\rho-1}\beta$ generated at the previous iteration to the off-line partial MAC update P_ρ, and obtains an online partial MAC update $\widetilde{P}_\rho$. Next, an integer online MAC result digit $\widetilde{r}_{\rho-\delta}$ is obtained by rounding $\widetilde{P}_\rho$, giving rise to an online error ϵ_ρ (stored for use in the next iteration).

In Step 5, one can use a truncation of $\widetilde{P}_\rho$ to its first γ MSDs to perform the rounding operation, where $1 \leq \gamma \leq W_{\widetilde{p}}$ is called the internal wordlength, and where $W_{\widetilde{p}}$ represents the wordlength of the online partial result. In the following, the number system radix β, the redundancy indices η_b, η_c, η_r, $\eta_{\widetilde{p}}$, the multiplicand bound l_a, the wordlengths W_a, W, the internal wordlength γ, and the relative position of the MSDs of AB and C, i_c, are taken into account as user-specified MAC arithmetic operation design parameters for the design of a corresponding architecture, where $\eta_{\widetilde{p}}$ represents the redundancy index of the OSD number $\widetilde{P}_\rho$, and where l_a represents the bound on A such that $|A| \leq l_a$. Moreover, P_ρ, ϵ_ρ, and $\widetilde{P}_\rho$ are taken into account as the variables of the MAC arithmetic operation.

At the last iteration $\rho = W$, one has $B_W = B$ and $C_W = C$. Therefore,

$$\widetilde{R}_W + \epsilon_W \beta^{\delta - W} = R_W \equiv R. \tag{3.19}$$

In this way, $\widetilde{R}_W$ represents the MSDs, and ϵ_W represents the least significant digits (LSDs) of R (as desired).

The correct functionality of the above algorithm has been verified by numerous Matlab simulations. It should be pointed out that the algorithm can be modified to perform other operations, such as inner products (Muller, 1994).

The internal wordlength γ should be kept small, because the delay of the FPGA hardware implementation of the estimator in Step 5 is proportional to $\log_\beta(\gamma)$ (look-ahead operation), or to γ (ripple-carry operation). Moreover, a low latency δ allows for the consumption of the result digits by another MAC architecture after only a small fixed delay (McQuillan and McCanny, 1995; Lapointe et al., 1993). Finally, negative values for i_c have been shown to make possible a reduction in the delay of the hardware implementation of the single-digit MAC architecture (Natter and Nowrouzian, 2000a). The same holds if i_c is larger than or equal to W_a: if $i_c = W_a$, the addend digit can be padded as the LSD of the online error, instead of a zero; otherwise, if $i_c > W_a$, the addend digit can be padded as the LSD of Ab_ρ.

3.2.3 Determination of the Parameters of the Algorithm

In this subsection, an interrelationship between γ, δ, and i_c is derived, and lower bounds on δ and i_c are obtained, (a) to place in evidence the trade-off between these user-specified parameters for an actual optimal hardware implementation, and (b) to obtain a procedure for the determination of the numerous design parameters (β, η_b, η_c, η_r, $\eta_{\overline{p}}$, W_a, l_a, γ, δ, and i_c). It is assumed that the number system radix β has been fixed at the outset, implying that the redundancy indices are constrained to the range $\left\{\left\lceil\frac{\beta}{2}\right\rceil, \ldots, \beta - 1\right\}$. By taking into account the fact that the redundancy indices can be chosen independently of each other, one can optimize the latency of nested MAC operations for subsequent hardware implementation. Therefore, it is further assumed that the redundancy indices have also been fixed.

At a given iteration ρ, the rounding of $\widetilde{P}_\rho$ to a single-digit integer $\widetilde{r}_{\rho-\delta}$ in Step 5 restricts the weight of the MSDs of $\widetilde{P}_\rho$ and $\epsilon_{\rho-1}$ to β^0. Let l_ϵ and l_p represent the bounds on ϵ_ρ and P_ρ, respectively, so that

$$|\epsilon_\rho| \leq l_\epsilon \quad \text{and} \quad |P_\rho| \leq l_p \tag{3.20}$$

both hold, where equality can be obtained for some ϵ_ρ and P_ρ. Then, for $\epsilon_{\rho-1}\beta + P_\rho$ to be representable by $\widetilde{r}_{\rho-\delta} + \epsilon_\rho$, the bound on $\widetilde{P}_\rho$ before rounding (c.f. Eqn. 3.17) must be tighter than the bound on $\widetilde{P}_\rho$ after rounding (c.f. Eqn. 3.18), i.e.

$$l_\epsilon\beta + l_p \leq \eta_r + l_\epsilon. \tag{3.21}$$

If Eqn. 3.21 is not satisfied, one may find an off-line partial update P_ρ and an online error $\epsilon_{\rho-1}$ such that their sum, $\widetilde{P}_\rho$, cannot be represented as an online result digit and a new online error ϵ_ρ that can satisfy the desired upper bounds.

Let us establish the values of the bounds l_ϵ and l_p by introducing an estimation function whose main operation involves rounding the real OSD number $\widetilde{P}_\rho$ to the closest single-digit integer $\tilde{r}_{\rho-\delta}$. Consider the OSD number $\widetilde{P}_\rho$ defined in accordance with

$$\widetilde{P}_\rho = \sum_{i=0}^{W_{\tilde{p}}-1} \tilde{p}_{\rho,i}\beta^{-i}; \quad \tilde{p}_{\rho,i} \in \{-\eta_{\tilde{p}}, \ldots, \eta_{\tilde{p}}\}, \tag{3.22}$$

where $W_{\tilde{p}}$ represents the wordlength of $\widetilde{P}_\rho$, and where $\eta_{\tilde{p}}$ represents the redundancy index of $\widetilde{P}_\rho$. In order to reduce the delay of a corresponding hardware implementation of the estimation function (which depends on the logarithm of the wordlength of $\widetilde{P}_\rho$), one may consider a truncation of $\widetilde{P}_\rho$ to its γ MSDs as given by

$$\overline{\widetilde{P}}_\rho = \sum_{i=0}^{\gamma-1} \tilde{p}_{\rho,i}\beta^{-i}, \tag{3.23}$$

to estimate $\tilde{r}_{\rho-\delta}$. The advantage of this truncation is to permit a constant-delay estimation in a corresponding hardware architecture (Irwin, 1977), but it increases l_ϵ, and, consequently, increases the latency δ. Then, the bound on the online error can be calculated by observing that the inequalities

$$|\epsilon_\rho| \le \left|\widetilde{P}_\rho - \tilde{r}_{\rho-\delta}\right| \tag{3.24}$$

$$\le \left|\widetilde{P}_\rho - \overline{\widetilde{P}}_\rho\right| + \left|\overline{\widetilde{P}}_\rho - \tilde{r}_{\rho-\delta}\right| \tag{3.25}$$

hold (the corresponding equalities may be obtained for some values of the online partial result). Next, one can calculate the upper bound on the online error in accordance with

$$l_\epsilon = \frac{1}{2} + \frac{\eta_{\tilde{p}}}{\beta - 1}\left(\beta^{1-\gamma} - \beta^{1-W_{\tilde{p}}}\right), \tag{3.26}$$

where $\frac{1}{2}$ is the usual real-to-integer rounding error, and where the other term is the bound on the truncated part of $\widetilde{P}_\rho$. The term $-\frac{\eta_{\tilde{p}}}{\beta-1}\beta^{1-W_{\tilde{p}}}$ becomes important when the wordlength of the online partial result becomes small (e.g. in addition $A = 1$ and $C = 0$), because it reduces l_ϵ substantially, permitting the increase of the bound on the off-line partial update. However, for practical multiply-accumulation operations, $W_{\tilde{p}}$ is large enough to warrant ignoring this term. Once β and η_r have been determined, one should minimize l_ϵ so as to avoid over-constraining l_p. When γ decreases, l_ϵ increases beyond a typical real-to-integer rounding error of $\frac{1}{2}$, leading to an increase in the latency of the MAC arithmetic operation.

Eqn. 3.21 implies an interrelationship between the bounds on ϵ_ρ and P_ρ which will be exploited in the proposed MAC algorithm to derive a lower bound on the latency δ, with the lower bound being dependent on the characteristics of the input operands A, B, and C. The following bounds on δ and i_c can be derived only after l_p has been determined.

Let $|P_\rho|_{\max}$ represent the maximum value of $|P_\rho|$. From the definition of P_ρ, one can show that

$$|P_\rho|_{\max} = \left(l_a\eta_b + \eta_c\beta^{-i_c}\right)\beta^{-\delta} \triangleq l_p. \tag{3.27}$$

For an OSD representation of A, l_a has been calculated in Subsection 3.2.1. For number systems other than OSD, one can replace l_a by the appropriate upper bound, e.g. $l_a = 2 - 2^{1-W_a}$ for SB numbers as represented in (McQuillan and McCanny, 1995). Finding the maximum l_p when the internal wordlength, the upper bound on the multiplicand, and the redundancy indices are given yields the smallest latency δ. For purely signed-binary MAC operations, all redundancy indices are equal to 1, and $l_a = 1 - 2^{-W_a}$. A plot of the valid bounds on P_ρ with respect to δ and i_c is obtained as shown in Figure 3.1 for $\gamma = 4$. The two

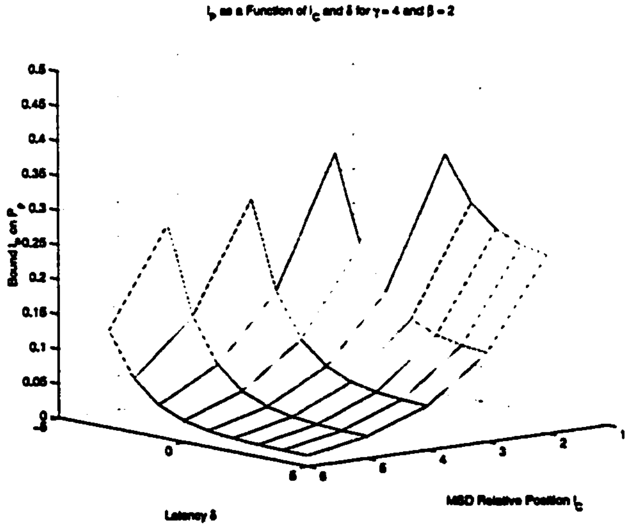


Figure 3.1: Bound on P_ρ as a function of δ and i_c for $\beta = 2$, $\gamma = 4$ and $l_a = 1$

solutions for which l_p in Figure 3.1 is maximum are obtained as tuples $(\delta, i_c) = (3, -1)$ and $(\delta, i_c) = (2, 1)$. Similarly, for $\gamma = 3$, the maximum l_p is obtained by the tuple $(\delta, i_c) = (3, 0)$. No improvement on l_p is obtained when using $\gamma = 5$ and $\gamma = 6$. This is due to the facts that l_p is given by

$$l_p = 2^{-\delta} - 2^{-W_a-\delta} + 2^{-i_c-\delta}, \tag{3.28}$$

and that l_p must always be smaller than $1 - l_\epsilon$ (c.f. Eqn. 3.21), which in turn implies

$$1 - l_\epsilon < 0.5 \tag{3.29}$$

since $l_\epsilon > \frac{1}{2}$ (unless $\gamma = W_{\bar{p}}$, which is impractical for a large $W_{\bar{p}}$). Therefore, it can be shown that the valid value of l_p nearest to 0.5 is 0.375, if the term $-2^{-W_a-\delta}$ is considered as small.

By substituting Eqns. 3.27 and 3.26 in Eqn. 3.21, one obtains

$$\left(l_a\eta_b + \eta_c\beta^{-i_c}\right)\beta^{-\delta} \le \eta_r + \frac{1-\beta}{2} - \eta_{\bar{p}}\left(\beta^{1-\gamma} - \beta^{1-W_{\bar{p}}}\right). \tag{3.30}$$

Let us restrict i_c to be smaller than W_a [2]. In this way, the wordlength of the online partial result can be calculated as

$$W_{\bar{p}} = W_a + \delta + 1 \tag{3.31}$$

by identifying the weight of its LSD. By manipulating Eqn. 3.30, one obtains the following lower bound on the latency

$$\delta_{\min} = \left\lceil \log_\beta \left(\frac{l_a\eta_b + \eta_c\beta^{-W_a} - \eta_{\bar{p}}\beta^{-W_a}}{\eta_r - \frac{\beta-1}{2} - \eta_{\bar{p}}\beta^{1-\gamma}} \right) \right\rceil, \tag{3.32}$$

satisfying

$$\delta \ge \delta_{\min}, \tag{3.33}$$

which is in agreement with that in (Brackert et al., 1989) for $\eta_b = \eta_c = \eta_r$, $i_c = 0$. Similarly, it is in agreement with that in (McQuillan and McCanny, 1995) for $\beta = 2$ (implying $\eta_b = \eta_c = \eta_{\bar{p}} = \eta_r = 1$), $\gamma = 4$, $i_c = 1$, and $l_a = 1$ (TC) or $l_a = 2 - 2^{1-W_a}$ (SB).

It is important to note that the digit of weight 1 in the proposed representations (Eqns. 3.2 through 3.5) must be equal to 0, whereas it can be non-zero in the representations presented in (McQuillan and McCanny, 1995). Therefore, the resulting latency values from the two representation types cannot be compared directly, because the representations of the inputs to and the outputs from the MAC operations are not the same. However, if the addend C is equal to zero, then one can compare the latency values indirectly by scaling

[2]Otherwise, the number of iterations ($= W$) required by the algorithm to calculate the MAC result must be increased because of the unnecessary padding of B and C with too many zeros.

the proposed result by 2 to obtain the result presented in (McQuillan and McCanny, 1995). Subsequently, one can show that the two results match exactly, thus proving that the latency is definitely a relative measurement. It can be concluded that the latency value depends on the representations of the inputs to and the outputs from the MAC operation.

When the bound on $Ab_\rho + c_{i_c+\rho}\beta^{-i_c}$ increases, the latency also increases to force l_p to satisfy Eqn. 3.21. Similarly, from Eqn. 3.30, one can obtain this lower bound on i_c

$$i_{c_{\min}} = \left\lceil \log_\beta \left(\frac{\eta_c}{\eta_r - \frac{\beta-1}{2} - \eta_{\tilde{p}}(\beta^{1-\gamma} - \beta^{-W_a-\delta}) - l_a\eta_b\beta^{-\delta}} \right) \right\rceil - \delta. \tag{3.34}$$

The complexity of choosing δ and i_c in Eqns. 3.32 and 3.34 can be reduced by using the same number representation for A, B, C, and $\widetilde{R}$, i.e. by setting $\eta_a = \eta_b = \eta_c = \eta_r \equiv \eta$, also permitting the consumption of $\widetilde{R}$ as a multiplicand, multiplier or addend in another online MAC architecture (Natter and Nowrouzian, 2000a). The following strategy can be applied for choosing appropriate values for the parameters in Eqns. 3.32 and 3.34.

(a) Choose the number system radix β,

(b) Choose the redundancy indices, η_b, η_c, η_r, and $\eta_{\tilde{p}}$,

(c) Choose the wordlength W_a of the multiplicand, and the bound l_a on A,

(d) Choose the internal wordlength γ, and calculate $\delta_{\min}$ using Eqn. 3.32,

(e) Choose the latency $\delta \geq \delta_{\min}$, and calculate $i_{C_{\min}}$ using Eqn. 3.34, and

(f) Choose the relative position of the MSDs of AB and C i_c such that $W_a \geq i_c \geq i_{C_{\min}}$.

As an example, for a SB online MAC arithmetic operation, $\beta = 2$, $l_a = 1$, and $\eta_b = \eta_c = \eta_{\tilde{p}} = \eta_r = 1$. Consequently, all the parameters in (a), (b), and (c) above are fixed except for W_a and γ. Let us find valid values for δ and i_c for various values of γ for $W_a = 5$. For $\gamma = 2$, no $\delta_{\min}$ and $i_{C_{\min}}$ can be found. For $\gamma = 3$, $\delta_{\min} = 3$ and the maximum valid l_p bound of 0.25 (represented by $l_{P_{\max}}$) is obtained for $i_{C_{\min}} = 0$. Similarly for $\gamma = 4$, one can find $\delta_{\min} = 2$: if one chooses $\delta = 2$, then $i_{C_{\min}} = 1$ and $l_{P_{\max}} = 0.375$, and if one chooses $\delta = 3$, then $i_{C_{\min}} = -1$ and $l_{P_{\max}} = 0.375$. No improvement on $l_{P_{\max}}$ is obtained when using larger values for γ for the same reason as mentioned previously.

The parameter i_c introduced in the proposed algorithm gives rise to additional flexibility in a corresponding hardware implementation, and may lead to the discovery of novel faster

MAC architectures. The MAC algorithm can be generalized for the determination of all affine functions as such functions can be evaluated by a bounded online operator (Muller, 1994). This generalization would be obtained by generating a different off-line partial update, implying a different lower bound on the latency, but keeping the general bounds on l_p and l_ϵ as fixed.

3.3 Algorithms and Building Blocks for High-Speed Signed-Binary Multiplication Architectures

Multiplication is obtained as a successive addition of partial products, where a partial product is equal to the appropriately scaled result of the multiplication of the multiplicand word by a single multiplier digit. An important widespread technique to increase the speed of multiplication involves the reduction of the number of partial products, usually obtained by encoding the multiplier into a different representation. In addition, the double-precision representation of a multiplication result must be approximated by a single-precision result, thereby requiring rounding and overflow correction operations. This section is concerned with the discussion of a conventional single-digit SB multiplier and a digit clearing unit, and with the development of an algorithm for recoding a signed-binary (SB) representation into a minimally redundant base-4 (MRB4) representation having $\{-2, -1, 0, 1, 2\}$ as its digit set in an attempt to improve the area-time efficiency of the resulting hardware multiplier architectures.

3.3.1 Single-Digit Multiplier and Digit Clearing Unit

In an iterative online algorithm for SB multiplication or SB multiply-accumulate operation, the multiplication of a SB multiplicand by a SB multiplier digit is required to form the off-line partial update. However, it is sufficient to consider the multiplication of a SB multiplicand digit by a SB multiplier digit, and to repeat the operation on every digit of the SB multiplicand.

Consider two SB digits $(a, b) \in \{-1, 0, 1\}^2$ represented by the bits $({}^1a, {}^2a) \in \{0, 1\}^2$ and $({}^1b, {}^2b) \in \{0, 1\}^2$ in accordance with

$$a = {}^1a + {}^2a - 1, \quad \text{and} \quad b = {}^1b + {}^2b - 1, \tag{3.35}$$

as in (Lapointe et al., 1993). Then, the product ab belongs to the set $\{-1, 0, 1\}$, and can be represented by a single result digit r as given by

$$r = {}^1r + {}^2r - 1, \quad ({}^1r, {}^2r) \in \{0, 1\}^2 \tag{3.36}$$

where the bits ${}^1r$ and ${}^2r$ are calculated in accordance with

$${}^1r = (\overline{{}^2a} \frown \overline{{}^1b}) \,|\, ({}^1a \frown {}^1b), \tag{3.37}$$

and

$${}^2r = (\overline{{}^1a} \frown \overline{{}^2b}) \,|\, ({}^2a \frown {}^2b), \tag{3.38}$$

where $\frown$, $|$, and $\bar{}$ represent the AND, OR, and NOT logic operations, respectively.

The correct functionality of these equations can be verified by exhaustive enumeration.

Digit Clearing Unit

In an iterative online algorithm, the online error has to be cleared at appropriate time instants. As the proposed architectures will employ SB words, the clearing operation is obtained by setting each SB digit a of a given sample to algebraic zero when a clearing control bit b is equal to logic 0. Consider the SB digit $a \in \{-1, 0, 1\}$, represented by the bits $({}^1a, {}^2a) \in \{0, 1\}^2$ in accordance with

$$a = {}^1a + {}^2a - 1, \tag{3.39}$$

and consider the control bit $Ctrl \in \{0, 1\}$. Then, the SB clearing unit result digit r belongs to the set $\{-1, 0, 1\}$, and can be represented in accordance with

$$r = {}^1r + {}^2r - 1, \quad ({}^1r, {}^2r) \in \{0, 1\}^2 \tag{3.40}$$

where the bits ${}^1r$ and ${}^2r$ are given by

$${}^1r = {}^1a \frown Ctrl \quad \text{and} \quad {}^2r = {}^2a \,|\, \overline{Ctrl}. \tag{3.41}$$

As a result, if $Ctrl = 1$, then $r = a$. Otherwise, $Ctrl = 0$ implying ${}^1r = 0$ and ${}^2r = 1$, which corresponds to $r = 0$, as desired.

3.3.2 Signed-Binary to Minimally Redundant Base-4 Recoding Technique

The well-known modified-Booth recoding technique was developed by MacSorley for the area-time efficient multiplication of two's complement (TC) numbers. This recoding technique recasts a TC multiplier into a corresponding minimally redundant base-4 (MRB4) representation having $\{-2,-1,0,1,2\}$ as its digit set by 3-bit overlapped-scanning. A corresponding 5-digit overlapped-scanning technique was developed for parallel signed-binary (SB) to MRB4 multiplier recoding (Rao and Nowrouzian, 1999). By using a base-4 multiplier representation, both techniques reduce the number of partial products by half. By avoiding the digit values 3 and -3 in the MRB4 representation, the modified-Booth recoding technique reduces the partial product calculation delay to a constant, whereas the 5-digit overlapped technique mostly reduces the area and delay by a constant factor. A 6-digit overlapped-scanning technique is presented in the following, which performs a parallel SB to MRB4 number recoding by using two successive conversions which are similar to addition. The proposed scanning technique is characterized by three important practical features. Firstly, as for the two previous techniques, the base-4 multiplier representation reduces the number of partial products by half, increasing the multiplication speed by a factor of 2. Secondly, as the 5-digit technique, the scanning technique is available to SB numbers. Thirdly, the conversions being similar to addition, a corresponding partial product formation architecture employing this multiplier recoding technique exhibits a reduction by half in the area obtained when employing the 5-digit recoding technique, at the expense of a small increase of the corresponding delay. The proposed technique adapts the existing binary carry-save (BCS) to MRB4 representation recoding technique proposed in (Kornerup, 1994) to SB representations by exploiting the similarities between the BCS and SB number representations.

Development of an Algorithm for SB to MRB4 Recoding

Let B represent a SB number of wordlength W given by

$$B = \sum_{i=1}^{W} b_i 2^{-i}; \quad b_i \in \{-1, 0, 1\}. \tag{3.42}$$

Let us assume that W is even in accordance with

$$W = 2W', \tag{3.43}$$

which can be obtained by padding a zero as the most or least significant digit of B, as deemed appropriate.

The 6-digit overlapped scanning technique amounts to recoding Eqn. 3.42 into a MRB4 representation B''' having $\{-2, -1, 0, 1, 2\}$ as its digit set in accordance with

$$B''' = \sum_{j=0}^{W'} b_j''' 4^{-j}. \tag{3.44}$$

This recoding consists of three phases, namely, (a) recasting the representation B into a maximally redundant base-4 representation B' having $\{-3, -2, -1, 0, 1, 2, 3\}$ as its digit-set, (b) recasting the representation B' into a base-4 representation B'' having $\{-1, 0, 1, 2, 3\}$ as its digit-set, and (c) recasting the representation B'' into the MRB4 representation B'''.

Firstly, let us recast B into a maximally redundant base-4 representation B' given by

$$B' = \sum_{j=1}^{W'} b_j' 4^{-j}. \tag{3.45}$$

By regrouping the odd and even terms of B in Eqn. 3.42, one obtains

$$B = \sum_{j=1}^{W'} (2b_{2j-1} + b_{2j}) 4^{-j}, \tag{3.46}$$

where $b_i = 0$ for $i < 1$ and $i > W$. Let b_j' be defined in accordance with

$$b_j' = 2b_{2j-1} + b_{2j}, \tag{3.47}$$

for all j in $\{1, \ldots, W'\}$. Then,

$$b_j' \in \{-3, -2, -1, 0, 1, 2, 3\}, \tag{3.48}$$

and

$$B' = B, \tag{3.49}$$

as desired.

Secondly, let us recast Eqn. 3.45 into a base-4 representation B'' given by

$$B'' = \sum_{j=0}^{W'} b''_j 4^{-j}, \tag{3.50}$$

where b''_j belongs to the set $\{-1, 0, 1, 2, 3\}$. This is achieved through a decomposition of the digits b'_j into transfer digits s_{j-1} and weight digits v_j in accordance with

$$4s_{j-1} + v_j = b'_j, \tag{3.51}$$

subject to

$$s_{j-1} \in \{-1, 0\} \tag{3.52}$$

and

$$v_j \in \{0, 1, 2, 3\}. \tag{3.53}$$

Then, by taking into account the constraints in Eqns. 3.52 and 3.53, it can be shown that Eqn. 3.51 possesses a unique solution as given by

$$s_{j-1} = \left\lfloor \frac{b'_j}{4} \right\rfloor, \tag{3.54}$$

and

$$v_j = \mathrm{mod}(b'_j, 4), \tag{3.55}$$

where $b'_j = 0$ for $j < 1$ and $j > W'$. Consequently, one can observe that

$$v_0 = 0 \quad \text{and} \quad s_{W'} = 0. \tag{3.56}$$

In addition, by making use of Eqns. 3.51 and 3.56, Eqn. 3.45 can be written as

$$B' = \sum_{j=0}^{W'} (s_j + v_j) 4^{-j}. \tag{3.57}$$

By letting the digits b''_j be defined in accordance with

$$b''_j = s_j + v_j, \tag{3.58}$$

it follows that

$$b''_j \in \{-1, 0, 1, 2, 3\}, \tag{3.59}$$

and that

$$B'' = B', \tag{3.60}$$

as desired.

Thirdly, let us recast Eqn. 3.50 into a MRB4 representation B''' given by

$$B''' = \sum_{j=0}^{W'} b_j''' 4^{-j}, \tag{3.61}$$

where b_j''' belongs to the set $\{-2, -1, 0, 1, 2\}$. This is achieved through a decomposition of the digits b_j'' into transfer digits t_{j-1} and weight digits w_j in accordance with

$$4t_{j-1} + w_j = b_j'', \tag{3.62}$$

subject to

$$t_{j-1} \in \{0, 1\} \tag{3.63}$$

and

$$w_j \in \{-2, -1, 0, 1\}. \tag{3.64}$$

Then, by taking into account the constraints in Eqns. 3.63 and 3.64, it can be shown that Eqn. 3.62 possesses a unique solution as given by

$$t_{j-1} = \left\lfloor \frac{b_j'' + 2}{4} \right\rfloor, \tag{3.65}$$

and

$$w_j = \mathrm{mod}(b_j'' + 2, 4) - 2, \tag{3.66}$$

where $b_j'' = 0$ for $j < 0$ and $j > W'$. Consequently, by making use of Eqn. 3.56, one can observe that

$$t_{-1} = 0 \quad \text{and} \quad t_{W'} = 0. \tag{3.67}$$

Moreover, by making use of Eqns. 3.62 and 3.67, Eqn. 3.50 can be written as

$$B'' = \sum_{j=0}^{W'} (t_j + w_j) 4^{-j}. \tag{3.68}$$

By defining the digits b_j''' in accordance with

$$b_j''' = t_j + w_j, \tag{3.69}$$

it follows that

$$b_j''' \in \{-2, -1, 0, 1, 2\}, \tag{3.70}$$

and that

$$B''' = B'', \tag{3.71}$$

as desired. Therefore, by making use of Eqns. 3.49, 3.60, 3.71, and 3.70 it can be concluded that B''' is a MRB4 representation corresponding to the SB representation B.

The above steps can be combined into a 6-digit overlapped-scanning technique as follows. Let a 6-digit overlapped scanning of the sets of digits

$$\langle b_{2j-1},\ b_{2j},\ b_{2j+1},\ b_{2j+2},\ b_{2j+3},\ b_{2j+4} \rangle \tag{3.72}$$

be performed successively for j in $\{0, \ldots, W'\}$ to calculate

$$b_j''' = \ \bmod\left(b_j' + \left\lfloor \frac{b_{j+1}'}{4} \right\rfloor + 2, 4\right) - 2 + \left\lfloor \frac{\bmod(b_{j+1}', 4) + \left\lfloor \frac{b_{j+2}'}{4} \right\rfloor + 2}{4} \right\rfloor, \tag{3.73}$$

where b_j' represents a base-4 digit of B' and is defined in accordance with

$$b_j' = 2b_{2j-1} + b_{2j}, \tag{3.74}$$

and where the digits b_i are assumed to be equal to 0 for $i > W$ and $i < 1$. By defining the word B''' in accordance with Eqn. 3.44, one obtains a MRB4 representation of the SB representation B.

The proposed recoding technique can also be viewed as successive digit set conversion schemes, as shown in Figure 3.2. This digit set conversion scheme was developed by using the technique presented in (Kornerup, 1994), where it was also shown that the digit set conversion shown in Figure 3.2 cannot be performed by using fewer levels.

The proposed recoding technique can also be regarded as recasting a maximally redundant base-4 representation, B', into a minimally redundant one, B'''. The digits 3 and -3

$$\begin{tikzcd}
 & 2^1\{-1,0,1\}+2^0\{-1,0,1\} \arrow[d] & \\
L_1 & b'_j \in \{-3,-2,-1,0,1,2,3\} \arrow[d] \arrow[dl] & \\
4s_{j-1} \in \{-4,0\} & v_j \in \{0,1,2,3\} \arrow[d] & s_j \in \{-1,0\} \arrow[dl] \\
L_2 & b''_j \in \{-1,0,1,2,3\} \arrow[d] \arrow[dl] & \\
4t_{j-1} \in \{0,4\} & w_j \in \{-2,-1,0,1\} \arrow[d] & t_j \in \{0,1\} \arrow[dl] \\
L_3 & b'''_j \in \{-2,-1,0,1,2\} &
\end{tikzcd}$$

Figure 3.2: Three-Level Signed-Binary to Radix-4 Digit Set Conversion

are thus avoided in the recoded number representation. Multiplication by -2, -1, 0, 1, and 2 can be obtained in parallel by appropriate hardwired shift, zeroing, and negation. However, multiplication of A by 3 is the result of $2A + A$, which necessitates the addition of two words. Therefore, the multiplication of a SB multiplicand word A of wordlength W_a by a SB multiplier word B of wordlength W usually requires W additions to obtain the desired result, whereas recoding the multiplier B into a minimally redundant base-4 representation implies the need for only $\lceil \frac{W}{2} \rceil + 1$ additions.

By recoding the SB multiplier into a corresponding MRB4 representation, about fifty percent of the additions are saved in a LSD-first SB multiplication algorithm, increasing the multiplication speed by a factor of 2. It is important to note that higher base recoding techniques exist, but that the additional gains in speed over the proposed recoding technique are small (Kornerup, 1994).

The following pseudo-code implements the proposed recoding technique.

Algorithm 20 *SB to MRB4 Recoding Algorithm*

```
input:  $B$
output:  $B'$
begin
  read $B$;
  set $b_i = 0$ for $i < 0$ and $i > W - 1$;
  set $B' = 0$;
  for $j = 0$ to $W'$;
  begin
    select $b_{2j-1}$, $b_{2j}$, $b_{2j+1}$, $b_{2j+2}$, $b_{2j+3}$, $b_{2j+4}$,
    compute $b'_j$ using Eqn. 3.73;
    set $B' = B' + b'_j 4^{-j}$;
  end
  write $B'$;
end
```

This algorithm can be exemplified by the recoding of the 16-digit SB word $0\bar{1}1110\bar{1}011\bar{1}101\bar{1}0$ into the 9-digit base-4 word $00\bar{1}2\bar{1}\bar{1}\bar{1}1\bar{2}$, where $\bar{1}$ denotes -1, and where $\bar{2}$ denotes -2.

Implementation of a SB to MRB4 Recoder

In a hardware implementation of a multiplication algorithm employing an SB to MRB4 recoded multiplier, $\left\lceil \frac{W_B}{2} \right\rceil + 1$ recoders are required, and, since a digit value multiplies an entire word, $W_A \left(\left\lceil \frac{W_B}{2} \right\rceil + 1 \right)$ corresponding SB digit by MRB4 digit multipliers are required. Therefore, by carefully choosing the encoding used for the outputs of the MRB4 recoder, one can minimize the area and delay of a corresponding SB digit by MRB4 digit multiplier.

The SB encoding given in (Lapointe et al., 1993) is chosen for the input SB digits, as shown in Table 3.1. If a SB digit b_i of weight 2^{-i} is encoded in accordance with Table 3.1

Table 3.1: Signed-Binary Encoding

Binary Code	SB digit
0 0	-1
0 1 and 1 0	0
1 1	1

by two bits, ${}^1b_i$ and ${}^2b_i$ respectively, then

$$b_i = {}^1b_i + {}^2b_i - 1 \tag{3.75}$$

holds.

Let us determine the encoding of the digits s_j, v_j, t_j, w_j, and b_j'''. Given their respective digit sets, one can code these variables in accordance with

$$s_j = {}^1s_j - 1 \tag{3.76}$$

$$v_j = 2\,{}^1v_j + {}^2v_j \tag{3.77}$$

$$t_j = {}^1t_j \tag{3.78}$$

$$w_j = 2({}^1w_j - 1) + {}^2w_j \tag{3.79}$$

and

$$b_j''' = (2\overline{negate_j} - 1) \times (\overline{shift_j} + 1) \times (zero_j) \tag{3.80}$$

where ${}^1s_j$, ${}^1v_j$, ${}^2v_j$, ${}^1t_j$, ${}^1w_j$, ${}^2w_j$, $negate_j$, $shift_j$, and $zero_j$ are bits, and thus belong to $\{0,1\}$.

The bit $shift_j$ indicates a multiplication by 1 or 2, the bit $zero_j$ indicates a multiplication by 0 or 1, and the bit $negate_j$ indicates a multiplication by 1 or -1, as listed in Table 3.2. It can be readily observed that the proposed encoding leads to the desired digit

Table 3.2: Conventions for MRB4 Digit Encoding

$shift_j$	
code	action
0	×2
1	×1

$zero_j$	
code	action
0	×0
1	×1

$negate_j$	
code	action
0	×1
1	×(−1)

sets.

The implementation of the first level is directed by Eqn. 3.51, which can be translated into

$$4\,{}^1s_{j-1} + 2\,{}^1v_j + {}^2v_j = 2\,{}^1a_{2j-1} + 2^2a_{2j-1} + {}^1a_{2j} + {}^2a_{2j} + 1, \tag{3.81}$$

which is a binary addition.

The implementation of the second level is directed by Eqn. 3.62, which can be translated into

$$4\,{}^1t_{j-1} + 2\,{}^1w_j + {}^2w_j = 2\,{}^1v_j + {}^2v_j + {}^1s_j + 1, \tag{3.82}$$

which is also a binary addition.

The implementation of the third level is directed by Eqn. 3.69 in accordance with

$$(2\overline{negate_j} - 1) \times (\overline{shift_j} + 1) \times (zero_j) = 2({}^1w_j - 1) + {}^2w_j + {}^1t_j, \tag{3.83}$$

which can be obtained by using

$$shift_j = {}^2s_j \oplus {}^1t_j, \tag{3.84}$$

$$zero_j = {}^1s_j \frown ({}^2s_j \,|\, {}^1t_j) \,|\, \overline{{}^1s_j} \frown (\overline{{}^2s_j \frown {}^1t_j}), \tag{3.85}$$

and

$$negate_j = \overline{{}^1s_j}, \tag{3.86}$$

where $\frown$, $|$, and $\overline{\ }$ represent the logical AND, OR, and NOT operations, respectively.

The resulting MRB4 recoder requires a 15 gate-equivalent area, assuming the area of a multiplexer is the same as that of a gate, and assuming the area of NOT gates can be neglected. A subsequent ASIC hardware implementation is therefore more area-efficient than that given in (Rao, 1996), for which 25 gate-equivalents are needed. Let us denote by t_G and t_{MUX} the delay through a gate and a multiplexer, respectively. The delays required to obtain the $shift_j$, $zero_j$, and $negate_j$ signals are $4t_G + 2t_{MUX}$, $4t_G + 3t_{MUX}$, and $3t_G + 2t_{MUX}$, respectively. If t_{MUX} is assumed to be equal to t_G, then the delay of an ASIC hardware implementation is only larger than that proposed in (Rao, 1996) by one gate delay.

Implementation of a mixed SB/MRB4 Digit Multiplier

A mixed SB/MRB4 multiplier takes three digits as inputs: two consecutive digits a_{i-1} and a_i of the SB multiplicand word A, having weights 2^{-i+1} and 2^{-i}, respectively, and one digit b_j''' of the recoded MRB4 multiplier word B''', represented by the signals $shift_j$, $zero_j$, and $negate_j$.

The SB result of the multiplication of A by b_j''' is denoted by R, where the bits representing the digit r_{i+2j} of weight 2^{-i-2j} can be expressed in terms of the inputs in accordance with

$$ {}^1r_{i+2j} = \left(\left({}^1a_i \frown shift_j \,|\, {}^1a_{i-1} \frown \overline{shift_j}\right) \frown zero_j \,|\, 0 \frown \overline{zero_j}\right) \oplus negate_j, \tag{3.87} $$

and

$$ {}^2r_{i+2j} = \left(\left({}^2a_i \frown shift_j \,|\, {}^2a_{i-1} \frown \overline{shift_j}\right) \frown zero_j \,|\, 1 \frown \overline{zero_j}\right) \oplus negate_j. \tag{3.88} $$

A corresponding ASIC hardware implementation requires a 6 gate-equivalent area, and its delay is the maximum of $t_G + 2t_{MUX}$ and $2t_G + t_{MUX}$. This area is much smaller than the 12 gate-equivalent area required in (Rao, 1996).

3.4 Rounding and Overflow Handling in Online Arithmetic Operations

The result R of an arithmetic operation very often belongs to a range that is different from that of the input(s). If R must be fed to the same unit as an input, then the problem of fitting the result into the desired input format comes into play. In more down-to-earth terms, the weights of some LSDs are too small, and those of some MSDs are too large, and R has to be re-represented in the desired format by taking all these digits into account.

The format fitting problem can be subdivided into two problems of different nature, referred to as rounding and overflow handling. Rounding, dealt with in Subsection 3.4.1, finds the closest number having no LSD of lower weight than desired. Subsection 3.4.2 is concerned with overflow handling, which determines whether the value of R belongs to the desired range, and corrects the representation accordingly. If the magnitude of R is too large (overflow), then the new representation corresponds to the largest (positive or negative, as appropriate) valid value: this is referred to as saturating the representation of R.

3.4.1 Rounding of Signed-Binary Words

The wordlength of the result R of the multiplication of a multiplicand of wordlength W by a multiplier of worlength W is at least larger than $2W$, as can be observed when calculating

$0.1 \times 0.1 = 0.01$. If this result is to be used as an input to another operation requiring the same format as the input multiplier and multiplicand, then R has to be rounded. The IEEE 754 standard includes a definition of the rounding-to-nearest/up (RNU) and the rounding-to-nearest/even (RNE) techniques (Santoro et al., 1989; Rao, 1996). In the following, a short description of the RNU and RNE rounding techniques will be given. Then, an algorithm for online IEEE 754 SB RNE rounding will be presented. Finally, a specific low-precision rounding operation is developed, which is required in online arithmetic operations for estimation of the online result digit.

IEEE 754 RNU and RNE Rounding Techniques

In both the RNU and the RNE rounding techniques, the exact result R of a multiplication must be rounded to a lower-precision word R_{rnd} which minimizes the absolute distance $d(R, R_{rnd})$ given by

$$d(R, R_{rnd}) = |R - R_{rnd}| \,. \tag{3.89}$$

By scaling R and R_{rnd} appropriately, one can assume that R_{rnd} is an integer, and thus that

$$d(R, R_{new}) \leq \frac{1}{2} \tag{3.90}$$

holds. The RNU standard stipulates that R_{rnd} must be equal to the integer nearest to R, and in the case of a tie (e.g. -1.5, -0.5, 0.5, or 1.5), R_{rnd} must be equal to the lowest integer larger than R (Santoro et al., 1989; Rao, 1996). The RNE standard differs from the RNU standard only in case of a tie: R_{rnd} must then be equal to the even integer nearest to R (Santoro et al., 1989; Rao, 1996).

Algorithms for IEEE 754 RNU and RNE rounding of SB numbers have been developed in (Rao, 1996) and will be explained briefly. The exact result R is decomposed into its most significant word R_{msw} and its least significant word R_{lsw} in accordance with

$$R = R_{msw} + R_{lsw}. \tag{3.91}$$

Then, one can observe that R_{lsw} is given by

$$R_{lsw} = \sum_{i=1}^{W_{lsw}} r_i 2^{-i}, \tag{3.92}$$

where W_{lsw} represents the wordlength of R_{lsw}. In this way, R_{lsw} consists of a sum of a digit r_1 with a least significant word $\widehat{R}_{lsw}$ in accordance with

$$R_{lsw} = r_1 2^{-1} + \widehat{R}_{lsw}. \tag{3.93}$$

One can observe that R_{lsw} represents a fractional number that belongs to the set $(-1, 1)$. Similarly, R_{msw} is given by

$$R_{msw} = \sum_{i=-W_{msw}+1}^{0} r_i 2^{-i}, \tag{3.94}$$

where W_{msw} represents the wordlength of R_{msw}. Again, R_{msw} consists of a sum of a digit r_0 with a most significant word $\widehat{R}_{msw}$ in accordance with

$$R_{msw} = r_0 2^0 + \widehat{R}_{msw}. \tag{3.95}$$

Finally, by considering the sign and magnitude of both r_1 and $\widehat{R}_{lsw}$, it is possible to determine whether R_{lsw} belongs to one of the five sets $[-1+2^{-W_{lsw}}, -0.5)$, $\{-0.5\}$, $(-0.5, 0.5)$, $\{0.5\}$, and $(0.5, 1-2^{-W_{lsw}}]$. Therefore, an algorithm parsing the 9 cases (3 for r_1, followed by 3 for $\widehat{R}_{lsw}$) is sufficient to perform the RNU rounding technique for a SB number. Similarly, the RNE rounding technique can be performed by an algorithm that takes into account the sign and magnitude of both r_1 and $\widehat{R}_{lsw}$, as well as the magnitude (and not the sign) of r_0. These rounding algorithms generate a *round* digit which can be added to R_{msw} for correction. Tables providing the *round* digit values for all the possible cases, and the corresponding algorithms have been presented in (Rao, 1996).

In a corresponding hardware implementation, the sign and magnitude of a word are represented by *sign* and *sticky* bits, respectively, by using the following conventions

$$\widehat{R}_{lsw} \geq 0 \quad \Leftrightarrow \quad sign = 0 \tag{3.96}$$

$$\widehat{R}_{lsw} < 0 \quad \Leftrightarrow \quad sign = 1, \tag{3.97}$$

and

$$\widehat{R}_{lsw} = 0 \quad \Leftrightarrow \quad sticky = 0 \tag{3.98}$$

$$\widehat{R}_{lsw} \neq 0 \quad \Leftrightarrow \quad sticky = 1, \tag{3.99}$$

as in (Rao, 1996). Then, the *round* digit to add to R_{msw} for correction is represented by two bits in accordance with

$$round = {}^1round + {}^2round - 1; \quad ({}^1round, {}^2round) \in \{0,1\}^2. \tag{3.100}$$

Similarly, r_0 and r_1 are both represented by two bits in accordance with

$$r_0 = {}^1r_0 + {}^2r_0 - 1; \quad ({}^1r_0, {}^2r_0) \in \{0,1\}^2, \tag{3.101}$$

and

$$r_1 = {}^1r_1 + {}^2r_1 - 1; \quad ({}^1r_1, {}^2r_1) \in \{0,1\}^2. \tag{3.102}$$

Next, *round* is calculated by using the following boolean equations

$$^1round = {}^1r_1 \frown {}^2r_1.(\overline{sticky} \frown \overline{^1r_0 \oplus ^2r_0} \,|\, sticky \frown \overline{sign}) \tag{3.103}$$

$$^2round = \overline{\overline{^1r_1} \,|\, \overline{^2r_1} \frown (\overline{sticky} \frown \overline{^1r_0 \oplus ^2r_0} \,|\, sticky \frown sign)}, \tag{3.104}$$

where $\frown$, $|$, $\oplus$, and $\bar{\ }$ denote the AND, OR, XOR, and NOT logic operations, respectively.

Online Algorithm for IEEE 754 SB RNE Rounding

Consider an iterative online algorithm for arithmetic operation with latency δ_{op} and effective wordlength W_{eff}[3]. The symbols relating to the arithmetic and the rounding operations are marked by using "op" and "rnd" subscripts, respectively. Then, one must recall Eqn. 3.12 (page 51), where the online operation result $\tilde{R}_{op,\rho}$ is combined with the online operation error $\epsilon_{op,\rho}$ to yield the off-line operation result $R_{op,\rho}$ in accordance with

$$R_{op,\rho} = \tilde{R}_{op,\rho} + \epsilon_{op,\rho}\beta^{\delta_{op}-\rho}. \tag{3.105}$$

In this way, there must exist an iteration number ρ_0 such that $\tilde{R}_{op,\rho_0}$ represents $R_{op,msw}$, and $\epsilon_{\rho_0}\beta^{\delta_{op}-\rho_0}$ represents $R_{op,lsw}$. One can calculate the desired round digit at any iteration, and clear it, unless $\rho = \rho_0$. Of course, the online error of the online arithmetic operation has to be cleared at iteration $\rho_0 + 1$. Otherwise, online arithmetic operation result digits of lower significance will be generated, although they have been taken into account in the

[3]The effective wordlength is defined as the wordlength augmented by the number of padded zero digits needed to match the wordlength of the online rounded operation result.

round digit already. At any given iteration ρ, the online arithmetic operation result digit $\tilde{r}_{op,\rho-\delta_{op}}$ is added to a round digit $round_{\rho-1-\delta_{op}}$. This digit is either equal to the round digit *round* obtained by performing the SB RNE algorithm on ϵ_{op,ρ_0} at iteration ρ_0, or to 0 otherwise. Consequently, it is sufficient to modify the online addition Algorithm 14 (page 33) so that it incorporates the calculation of the round digit before its addition to the online result digit. The resulting algorithm performs the desired SB RNE rounding of $\tilde{R}_{op}$ in an online fashion.

Algorithm 21 *Online IEEE 754 SB RNE Rounding*

```
Step 1: Set $\rho \leftarrow 2$ and $\epsilon_{rnd,1} \leftarrow 0$,
Step 2: Set $r_0 \leftarrow \tilde{r}_{\rho-\delta_{op}}$, and $R_{lsw} \leftarrow \epsilon_{op,\rho}$
Step 3: Calculate $rnd_{\rho_0-\delta_{op}}$ by performing the SB RNE rounding algorithm on $r_0$ and
        $R_{lsw}$;
Step 4: If $\rho = \rho_0 + 1$, then calculate $P_{rnd,\rho} \leftarrow \tilde{r}_{rnd,\rho-1-\delta_{op}} 2^{-\delta_{rnd}}$, else calculate $P_{rnd,\rho} \leftarrow$
        $\left(\tilde{r}_{rnd,\rho-1-\delta_{op}} + rnd_{\rho-1-\delta_{op}}\right) 2^{-\delta_{rnd}}$,
Step 5: Calculate $\tilde{P}_{rnd,\rho} \leftarrow 2\epsilon_{rnd,\rho-1} + P_{rnd,\rho}$,
Step 6: Round $\tilde{P}_{rnd,\rho}$ to $\tilde{r}_{rnd,\rho-1-\delta_{op}-\delta_{rnd}}$,
Step 7: Calculate $\epsilon_{rnd,\rho} \leftarrow \tilde{P}_{rnd,\rho} - \tilde{r}_{rnd,\rho-1-\delta_{op}-\delta_{rnd}}$,
Step 8: Calculate $\rho \leftarrow \rho + 1$,
Step 9: Store $\epsilon_{rnd,\rho}$,
Step 10: If $\rho < W_{eff} + 2$, then go to Step 4, else done.
```

In the above algorithm, the online error ρ starts at 2 to account for the unit-delay between the arrival of the first input digit to the arithmetic operation and the departure of the corresponding output result digit. If the arithmetic operation is the addition (c.f. Subsection 2.4, page 34), then the SB input digits a_1 and b_1 are consumed at iteration 1, the SB sum digit r_{-1} is generated at iteration 1, and made available to the rounding operation at iteration 2. This delay results in latching the operation result digit in a corresponding hardware implementation. The online error (corresponding to the carries c_1 and c_2 of a limited-carry addition, c.f. Subsection 2.4.3) is initialized to 0. Then, at a given iteration ρ, a round digit $rnd_{\rho-\delta_{op}}$ is determined in accordance with the SB RNE algorithm presented in (Rao, 1996). Next, the online arithmetic operation result digit $\tilde{r}_{op,\rho-1-\delta_{op}}$ is added to the round digit if the iteration is equal to $\rho_0 + 1$, and 0 otherwise. The resulting off-line partial update $P_{rnd,\rho}$ is combined with the scaled online error $2\epsilon_{rnd,\rho-1}$ calculated at the previous iteration, yielding the online partial rounding result $\widetilde{P}_{rnd,\rho}$. Finally, the online partial rounding result is approximated to the online rounded result digit $\widetilde{r}_{rnd,\rho-1-\delta_{op}-\delta_{rnd}}$, thus forming an online error $\epsilon_{rnd,\rho}$ (the carries c_1' and c_2' of a limited-carry addition). The rounding operation is finished when the iteration number is larger than the effective wordlength of the input plus one digit.

The latency of the proposed rounding operation is equal to that of addition, i.e.

$$\delta_{rnd,\min} = 2. \tag{3.106}$$

It is important to observe that

$$\epsilon_{rnd,W_{b,eff}+1} = 0. \tag{3.107}$$

This is due to the fact that δ_{rnd} online result operation digits equal to zero have been introduced to flush the online rounded operation digits. As a result, the online rounding error does not need clearing when operations are processed one after another (Natter and Nowrouzian, 1999).

Low-Precision Rounding for Calculation of a SB Online Result Digit

At any given iteration ρ of an online algorithm, one must round an online partial result $\widetilde{P}_\rho$ to an online result digit $\widetilde{r}_{\rho-\delta}$. This rounding technique differs greatly from other rounding techniques such as the IEEE 754 standard RNU and RNE rounding techniques described in (Santoro et al., 1989; Rao, 1996). First, $\widetilde{P}_\rho$ is truncated to its first γ MSDs before rounding, where γ is referred to as the internal wordlength of the online operation. Second, the output of the rounding operation is an integer that must belong to a restricted output result digit set. If the result is represented in a SB format, then the digit set is $\{-1, 0, 1\}$.

The first difference results in a degradation of the closeness of the rounded result to the exact result, which may, in turn, cause true overflow in the online operation. In other words, the wordlength of the online partial result may prove to be too large, yielding more than one online result digit, and thus requiring the change of an online result digit that has already been generated at a previous iteration. However, a careful determination of the internal wordlength together with the latency of the operation, which are commonly referred to as the online algorithm parameters, solves this problem.

After carefully determining the online algorithm parameters, the second difference can only cause unnecessary (but correctable) overflow, which can solely occur with redundant representations. Of course, $\widetilde{r}_{\rho-\delta}$ must represent the nearest integer to $\widetilde{P}_{\rho}$. However, in the case of a tie, it is mandatory to choose the digit value closest to 0 if the online result digit risks to be chosen out of the valid digit set.

It is important to note that the above rounding principle for online algorithms is applicable to any radix-β OSD number system. An algorithm causing the unnecessary overflow can be found in (Owens, 1980), where the online digit $r_{\rho-\delta}$ is given by

$$r_{\rho-\delta} = \text{sign}\left(\widetilde{P}_{\rho}\right) \times \min\left(\left\lceil\left|\widetilde{P}_{\rho}\right|\right\rceil, \eta\right), \tag{3.108}$$

where η represents the redundancy index of the OSD number representation of the online result, where sign(.) represents the sign of its argument, where min(.,.) represents the minimum value of its arguments, where $|.|$ represents the magnitude of its argument, and where $\lceil . \rceil$ represents the result of the RNU algorithm on its positive argument.

In case of a SB number system, it is desired to obtain an algorithm which rounds a SB number to the nearest integer, but in the case of a tie, chooses the integer closest to 0. In this way, an online partial result equal to 1.5 is rounded to 1, which is an allowed SB digit value. One can use the decomposition of Eqn. 3.91 to determine the desired rounding procedure.

Let us distinguish between three cases corresponding to the three possible values of r_1:

- If $r_1 = -1$, then two sub-cases need consideration:
  - If $\widehat{R}_{lsw} < 0$, then $R_{lsw} < -0.5$, and one must choose $round = -1$.
  - If $\widehat{R}_{lsw} \geq 0$, then $R_{lsw} \geq -0.5$, and one must choose $round = 0$.
- If $r_1 = 0$, then one must always choose $round = 0$.
- If $r_1 = 1$, then two sub-cases need consideration:
  - If $\widehat{R}_{lsw} > 0$, then $R_{lsw} > 0.5$, and one must choose $round = 1$.

– If $\widehat{R}_{lsw} \leq 0$, then $R_{lsw} \leq -0.5$, and one must choose $round = 0$.

These results are summarized in Table 3.3.

Table 3.3: Low-Precision Rounding of SB Numbers

rounded	$r_1 = -1$		$r_1 = 0$	$r_1 = 1$	
number	$\widehat{R}_{lsw} < 0$	$\widehat{R}_{lsw} \geq 0$		$\widehat{R}_{lsw} \leq 0$	$\widehat{R}_{lsw} > 0$
R_{rnd}	$R_{msw} - 1$	R_{msw}	R_{msw}	R_{msw}	$R_{msw} + 1$

The above table can be implemented by generating a round digit *round*, represented algebraically in accordance with

$$round = {}^1round + {}^2round - 1, \tag{3.109}$$

where the two bits ${}^1round$ and ${}^2round$ can be calculated by the following boolean functions

$${}^1round = {}^1r_1 \frown {}^2r_1 \frown \overline{sign}, \tag{3.110}$$

$${}^2round = \overline{sign \frown sticky \frown \overline{{}^1r_1 \,|\, {}^2r_1}}, \tag{3.111}$$

where *sign* and *sticky* are as described for the IEEE 754 RNE rounding technique, and where r_1 is represented algebraically in accordance with

$$r_1 = {}^1r_1 + {}^2r_1 - 1. \tag{3.112}$$

Then, the *round* digit is added to R_{msw} to yield R_{rnd}. In order to avoid the generation of a non-desired carry, this addition must be obtained by letting the carry propagate through the two digit positions of highest weight.

Let us develop a SB carry-propagate adder combining only one input digit a with a carry-in c_{in}, and generating a sum digit s and a carry-out digit c_{out}. The SB digit a and the SB carry-in c_{in} can be expressed as follows

$$a = {}^1a + {}^2a - 1 \quad \text{and} \quad c_{in} = {}^1c_{in} + {}^2c_{in} - 1, \tag{3.113}$$

where $({}^1a, {}^2a) \in \{0, 1\}$, and where $({}^1c_{in}, {}^2c_{in}) \in \{0, 1\}$. Then, addition occurs in accordance with

$$2c_{out} + s = a + c_{in}, \tag{3.114}$$

where $c_{out} \in \{-1,0,1\}$ and $s \in \{-1,0,1\}$ can be expressed as follows

$$s =^1 s +^2 s - 1 \quad \text{and} \quad c_{out} =^1 c_{out} +^2 c_{out} - 1, \tag{3.115}$$

where $({}^1s, {}^2s) \in \{0,1\}$, and where $({}^1c_{out}, {}^2c_{out}) \in \{0,1\}$. The c_{out} digit must always be equal to 0, unless both a and c_{in} are different from 0, thereby avoiding the possible writing of 1 as $2-1$ or -1 as $-2+1$. The corresponding boolean equations are as follows

$${}^1c_{out} =^1 a \tag{3.116}$$

$${}^2c_{out} =^2 a \frown \overline{{}^1a} \,|\, \overline{{}^1a} \frown ({}^1b \,|\, {}^2b) \,|\, {}^2a \frown {}^1b \frown {}^2b \tag{3.117}$$

$${}^1s =^2 a \frown {}^2b \,|\, {}^1a \frown {}^1b \tag{3.118}$$

and

$${}^2s = \overline{{}^1a \,|\, {}^2a \,|\, ({}^1b \oplus^2 b)} \,|\, {}^2a \frown (\overline{{}^1b} \frown {}^1a \,|\, {}^1b \frown \overline{{}^2b}) \,|\, {}^2b \frown (\overline{{}^1a} \frown {}^1b \,|\, {}^1a \frown \overline{{}^2a}), \tag{3.119}$$

where $\frown$, $|$, $\oplus$ and $\bar{\ }$ denote the AND, OR, XOR, and NOT logic operations, respectively.

Finally, the proposed low-precision rounding must also generate a least significant word $R_{lsw,new}$ that, when added to R_{rnd}, yields R again in accordance with

$$R = R_{rnd} + R_{lsw,new}, \tag{3.120}$$

where R is given by Eqn. 3.91. In other words,

$$R = R_{msw} + round + R_{lsw,new}, \tag{3.121}$$

must hold. In this way, it is sufficient to find a digit $r_{1,new}$ satisfying

$$2round + r_{1,new} = r_1. \tag{3.122}$$

If *round* equals 0, then no correction is necessary. Otherwise, one can observe that r_1 is always exactly opposite to *round*. Therefore, the correction of the digit r_1 consists of multiplying it by -1 when *round* is different from zero. This is obtained by setting

$${}^1r_{1,new} =^1 r_1 \oplus ({}^1round \oplus^2 round) \quad \text{and} \quad {}^2r_{1,new} =^2 r_1 \oplus ({}^1round \oplus^2 round), \tag{3.123}$$

where

$$r_{1,new} =^1 r_{1,new} +^2 r_{1,new} - 1 \tag{3.124}$$

holds.

An extension of the proposed rounding technique to larger radices and redundancy indices consists of obtaining the online result digit in accordance with

$$r_{\rho-\delta} = \text{sign}\left(\widetilde{P}_\rho\right) \times \left(\left\lceil \left|\widetilde{P}_\rho\right| + 0.5 \right\rceil - 1\right). \tag{3.125}$$

In this way, only a RNU algorithm is required for estimation of the online result digit, as opposed to a RNU algorithm and a operation calculating the minimum value of its arguments. The complexity of a corresponding hardware implementation is thus lowered. Since the estimation function is the bottleneck of any architecture for online arithmetic operation, this reduction in complexity has important practical advantages. It can be observed that the "even" event in the RNE standard becomes irrelevant in odd bases, because the decimal value 0.5 cannot be obtained exactly.

3.4.2 Overflow Handling in Online Arithmetic Operations

The result $\widetilde{R}$ of an online arithmetic operation with latency δ is given by

$$\widetilde{R} = \sum_{i=1-\delta}^{W} \widetilde{r}_i \beta^{-i}; \quad \widetilde{r}_i \in \{-\eta, -\eta+1, \ldots, \eta\}, \tag{3.126}$$

but is desired to be represented in accordance with

$$\widetilde{R}_{des} = \sum_{i=1}^{W} \widetilde{r}_{des,i} \beta^{-i}; \quad \widetilde{r}_{des,i} \in \{-\eta, -\eta+1, \ldots, \eta\}, \tag{3.127}$$

such that $\widetilde{R}_{des} = \widetilde{R}$, where W represents the wordlength of the desired format, where β represents the radix, and where $\left\lceil \frac{\beta}{2} \right\rceil \leq \eta \leq \beta - 1$ represents the redundancy index. The first representation must thus be corrected in order to fit the desired format of the second representation. The extra MSDs of the first representation may induce too large a value for $\widetilde{R}$ to be corrected. This situation is called a true overflow (Timmermann and Hosticka, 1993; Rao, 1996). If these extra MSDs represent a non-zero value, and if $\widetilde{R}$ can still be represented in the desired format, one refers to the situation as a correctable overflow (Timmermann and Hosticka, 1993; Rao, 1996). Finally, there is no overflow when these extra MSDs represent 0.

Rules to determine and handle overflow situations have been developed in the signed-binary case in (Woods et al., 1993; Timmermann and Hosticka, 1993; Lapointe et al., 1993; Rao, 1996). They are extended to the case of an OSD number system in the following.

Necessary Condition for OSD Number Correctability

Let us decompose $\widetilde{R}$ into its most significant word $\widetilde{R}_{msw}$ and its least significant word $\widetilde{R}_{lsw}$ in accordance with

$$\widetilde{R} = \widetilde{R}_{msw} + \widetilde{R}_{lsw}, \tag{3.128}$$

where

$$\widetilde{R}_{msw} = \sum_{i=1-\delta}^{0} \widetilde{r}_i \beta^{-i}, \tag{3.129}$$

and where

$$\widetilde{R}_{lsw} = \sum_{i=1}^{W} \widetilde{r}_i \beta^{-i}. \tag{3.130}$$

In this way, $\widetilde{R}_{msw}$ represents the overflow number, and $\widetilde{R}_{lsw}$ represents the truncation of $\widetilde{R}$ to the desired format.

Theorem 22 *If $\widetilde{R}_{msw}$ represents a correctable overflow, then $\widetilde{R}_{msw}$ can take on any of the following three forms: $0\ldots0\bar{1}\eta_0\ldots\eta_0$, $0\ldots0$, or $0\ldots01\bar{\eta_0}\ldots\bar{\eta_0}$, where $\eta_0 = \beta - 1$, and where $\bar{\eta_0}$ represents $-\eta_0$.*

Proof. The proof of this theorem can be established after proving the following three lemmas. ∎

Lemma 23 *If $\widetilde{R}$ is represented as in Eqn. 3.126, then*

$$\left|\widetilde{R}\right| \leq \eta \frac{1-\beta^{-W}}{\beta - 1} \tag{3.131}$$

is a necessary and sufficient condition for $\widetilde{R}$ to be representable as given in Eqn. 3.127.

Proof. If $\widetilde{R}$ can be represented in the desired format of Eqn. 3.127, then Eqn. 3.131 holds, and is thus a necessary condition.

Conversely, if $\tilde{R}$ is represented in the typical format of Eqn. 3.126, and if Eqn. 3.131 holds, then $\tilde{R}\beta^W$ is an integer whose range belongs to the representable range of the format given in Eqn. 3.127. Since an OSD number system is complete, $\tilde{R}$ can be represented in the desired format. ■

Lemma 24 *If $\tilde{R}_{msw}$ represents a correctable overflow, then the corresponding value belongs to the set* $\{-1, 0, 1\}$.

Proof. Let us assume that the (integer) value of $\tilde{R}_{msw}$ does not belong to the set $\{-1, 0, 1\}$ and that $\tilde{R}_{msw}$ represents a correctable overflow. Then, one can observe that $\tilde{R}_{lsw}$, by itself, satisfies Eqn. 3.131. Next, by calculating the extreme values of $\tilde{R}$, one obtains

$$\tilde{R} \leq 2 - \eta\frac{1-\beta^{-W}}{\beta-1} \tag{3.132}$$

and

$$\tilde{R} \geq -2 + \eta\frac{1-\beta^{-W}}{\beta-1}. \tag{3.133}$$

Since $\beta \geq 2$ and $\eta \leq \beta - 1$, it follows that

$$\eta\frac{1-\beta^{-W}}{\beta-1} < 1. \tag{3.134}$$

Consequently,

$$\left|\tilde{R}_{lsw}\right| > 1, \tag{3.135}$$

contradicting the fact that $\tilde{R}$ represents a correctable overflow, and completing the proof. ■

Lemma 24 is an extension of when $\tilde{R}_{msw}$ contains one or two SB digits and the operation is addition (Timmermann and Hosticka, 1993; Rao, 1996).

Lemma 25 *Let $\eta_0 = \beta - 1$ and $\overline{\eta_0} = -\eta_0$. An integer representation of* 0 *in any OSD number system is of the form* $00\ldots0$. *If $\eta = \eta_0$, then integer representations of* -1, *and* 1 *in an OSD number system of redundancy index η are of the form* $00\ldots0\bar{1}\eta_0\eta_0\ldots\eta_0$, *and* $00\ldots01\overline{\eta_0}\,\overline{\eta_0}\ldots\overline{\eta_0}$, *respectively. If $\eta < \eta_0$, then the corresponding OSD integer number representations of* -1 *and* 1 *become* $00\ldots0\bar{1}$, *and* $00\ldots01$, *respectively.*

Proof. The representation of the value 0 in any OSD number system is of the form $00\ldots0$. To prove this, consider the representation $\widetilde{R}_0$ of 0 given by

$$\widetilde{R}_0 = \sum_{i=1-\delta}^{0} \widetilde{r}_{0,i}\beta^{-i}; \quad \widetilde{r}_{0,i} \in \{-\eta, -\eta+1, \ldots, \eta\}, \tag{3.136}$$

where β represents the radix of the number system, and where $\left\lceil \frac{\beta}{2} \right\rceil \le \eta \le \beta - 1$ represents the redundancy index of the number system. Let us assume that $\widetilde{r}_{0,i_0}$ is different from 0, where i_0 belongs to the set $\{1-\delta, \ldots, 0\}$. Of course, the integer part of $\widetilde{R}_0\beta^{i_0}$ forms its most significant word, and must be a correctable overflow. Otherwise, it can be readily observed that $\widetilde{R}_0$ cannot be equal to 0. Then, by applying Lemma 24 to the integer part of $\widetilde{R}_0\beta^{i_0}$, the corresponding value must belong to the set $\{-1, 0, 1\}$. However, the magnitude of the fractional part of $\widetilde{R}_0\beta^{i_0}$ is strictly smaller than 1 (c.f. Lemma 23 and $\eta \le \beta - 1$). Therefore, $\widetilde{R}_0$ cannot be equal to zero, establishing the proof.

Let us consider a SB representation $\widetilde{R}_1$ of 1 given by

$$\widetilde{R}_1 = \sum_{i=1-\delta}^{0} \widetilde{r}_{1,i}\beta^{-i}; \quad \widetilde{r}_{1,i} \in \{-\eta, -\eta+1, \ldots, \eta\}. \tag{3.137}$$

Then, consider the index $1 - \delta \le i_0 \le 0$ of the non-zero digit of largest weight. By choosing i_1 in $\{i_0, i_0+1, \ldots, 0\}$ and by applying Lemma 24, one has

$$\beta^{i_1} \sum_{i=i_0}^{i_1} \widetilde{r}_{1,i}\beta^{-i} \in \{-1, 0, 1\}. \tag{3.138}$$

By observing that $\widetilde{R}_1 = 1$ and by taking $i_1 = i_0$, it can be readily concluded that $\widetilde{r}_{1,i_0} = 1$.

It is very important to observe that if $\eta < \beta - 1$, then $i_0 = 0$. The proof is established by assuming that $i_0 < 0$ and by observing that $i_0 \le 0$ must always hold. In this situation, the smallest value $\widetilde{R}_{msw,\min}$ that can be represented by $\widetilde{R}_{msw}$ is given by

$$\widetilde{R}_{msw,\min} = \beta^{-i_0} + \sum_{i=i_0+1}^{0} -\eta\beta^{-i}, \tag{3.139}$$

which can be recast into

$$\widetilde{R}_{msw,\min} = \beta^{-i_0} - \eta\frac{\beta^{-i_0-1} - 1}{\beta - 1}. \tag{3.140}$$

Since $\eta < \beta - 1$, $\widetilde{R}_{msw,\min} > 1$, and $\widetilde{R}_{msw}$ cannot be equal to 1, leading to a contradiction.

Next, let us prove by recurrence on i_1 that when $\eta = \beta - 1$, the following mathematical proposition is true for $i_0 + 1 \le i_1 \le 0$

$$P_{i_1} : \forall i \in \{i_0 + 1, i_0 + 2, \ldots, i_1\}, \quad \widetilde{r}_{1,i} = -(\beta - 1). \tag{3.141}$$

- If $i_1 = i_0 + 1$, then

  $$\beta^{i_1} \sum_{i=i_0}^{i_1} \widetilde{r}_{1,i}\beta^{-i} = 1, \tag{3.142}$$

  which can be written as

  $$\widetilde{r}_{1,i_0}\beta + \widetilde{r}_{1,i_0+1} = 1. \tag{3.143}$$

  Since $\widetilde{r}_{1,i_0} = 1$, one can obtain $\widetilde{r}_{1,i_0+1}$, satisfying proposition P_{i_1}.

- If P_{i_1} is valid for $i_0 + 1 \le i_1 \le -1$, then let us prove that P_{i_1+1} is also valid. In this way, it is sufficient to prove that $\widetilde{r}_{1,i_1+1} = -(\beta - 1)$. One can replace i_1 by $i_1 + 1$ in Eqn. 3.142 to obtain

  $$\beta^{i_1+1} \sum_{i=i_0}^{i_1+1} \widetilde{r}_{1,i}\beta^{-i} = 1, \tag{3.144}$$

  which, since P_{i_1} is valid, becomes

  $$\beta + \widetilde{r}_{1,i_1+1} = 1, \tag{3.145}$$

  and restricts the value of $\widetilde{r}_{1,i_1+1}$ as necessary.

This establishes the proof by recurrence.

The proof for the representation of -1 is obtained by observing that $-1 = 1 \times (-1)$, and that any OSD number system has a balanced digit set. ■

The proof of Theorem 22 can now be established.

Proof. If $\widetilde{R}_{msw}$ represents a correctable overflow, then Lemma 24 implies that $\widetilde{R}_{msw}$ represents -1, 0, or 1, and Lemma 25 further determines the representation of $\widetilde{R}_{msw}$, establishing the proof. ■

Online Algorithm for Overflow Handling in OSD Number Systems

The algorithm developed in this subsection extends the existing work on algorithms and architectures for overflow detection and correction in signed-binary representations (Woods et al., 1993; Fernando and Ercegovac, 1992; Timmermann and Hosticka, 1993; Lapointe et al., 1993; McQuillan and McCanny, 1995; Rao, 1996). Moreover, it is shown that the online detection and correction of an overflow in the most significant part of an OSD representation is simplified by the use of Theorem 22.

One must recast the representation of $\widetilde{R}$ as given by Eqn. 3.126 into the desired format of Eqn. 3.127. Then, at a given iteration $1 \leq \rho \leq W + \delta$ of the algorithm for online arithmetic operation of latency δ, the online operation result $\widetilde{R}_\rho$ is defined in accordance with

$$\widetilde{R}_\rho = \sum_{i=1}^{\rho} \widetilde{r}_{i-\delta} \beta^{\delta-i}. \tag{3.146}$$

Next, $\widetilde{R}_\rho$ must be recast into an online corrected result $\widetilde{R}_{des,\rho}$ as given by

$$\widetilde{R}_{des,\rho} = \sum_{i=1}^{\rho} \widetilde{r}_{des,i-\delta} \beta^{\delta-i}. \tag{3.147}$$

The online error between the online corrected result and the online operation result is defined in accordance with

$$\epsilon_\rho = \widetilde{R}_\rho - \widetilde{R}_{des,\rho}. \tag{3.148}$$

From Theorem 22, ϵ_ρ can only take on the following five values: $-OVF$, -1, 0, 1, OVF, where OVF and $-OVF$ denote positive and negative overflow values, respectively. In the case of an overflow, the digits of $\widetilde{R}_{des,\rho-\delta}$ must be saturated to η or $-\eta$, depending on the sign of the overflow. A saturation function $sat(x, y)$ can be defined in accordance with

$$sat(x, y) = \max(\min(x, y), -y), \tag{3.149}$$

where min(.,.) and max(.,.) represent the minimum and maximum values of their arguments, respectively. This function will be used for the determination of the online corrected result digit. An absolute value of ϵ_ρ equal to 1 indicates a potential overflow, which may turn into a true overflow, or into no overflow at all (Timmermann and Hosticka, 1993; Rao, 1996).

Similar to the online MAC algorithm, an online partial correction result $\widetilde{P}_\rho$ is calculated as follows

$$\widetilde{P}_\rho = \epsilon_{\rho-1}\beta + \widetilde{r}_{\rho-\delta}, \tag{3.150}$$

and must be translated into a corrected digit $\widetilde{r}_{des,\rho-\delta}$ such that

$$\epsilon_\rho + \widetilde{r}_{des,\rho-\delta} = \widetilde{P}_\rho \tag{3.151}$$

holds, unless $\epsilon_{\rho-1}$ indicates an overflow. If $\epsilon_{\rho-1}$ equals OVF or $-OVF$, ϵ_ρ must be set to the same positive or negative overflow, and $\widetilde{r}_{des,\rho-\delta}$ must be set to η or $-\eta$, in accordance with the sign of the overflow. It is important to observe that if $\epsilon_{\rho-1} = 0$, then $\widetilde{r}_{des,\rho-\delta} = \widetilde{r}_{\rho-\delta}$, and no correction is necessary. The case $|\epsilon_{\rho-1}| = 1$ must be explained in more detail.

Let us observe that if $\epsilon_{\rho-1} = 1$, then the situation is that of a potential overflow. If $\widetilde{P}_\rho = \eta + 1$, then one can only choose $\epsilon_\rho = 1$ and $\widetilde{r}_{des,\rho-\delta} = \eta$ in order for Eqn. 3.151 to hold. However, if $\widetilde{P}_\rho > \eta + 1$, then it is not possible to find any valid combination of online error and online corrected result digit for Eqn. 3.151 to hold. Therefore, one must set $\widetilde{r}_{des,\rho-\delta} = \eta$ and $\epsilon_\rho = OVF$. Finally, if $\widetilde{P}_\rho < \eta + 1$, it is always possible to find $\epsilon_\rho = 0$ and $\widetilde{r}_{des,\rho-\delta} = \widetilde{P}_{\rho-\delta}$ so that Eqn. 3.151 holds. It is thus possible to combine the cases $\widetilde{P}_\rho = \eta + 1$ and $\widetilde{P}_\rho < \eta + 1$ together. Moreover, it can be observed that if a potential overflow becomes a true overflow, then the previously generated MSDs have saturated to their correct values. It is very important to note that the calculation of the online corrected result digit corresponds to that of $\widetilde{P}_\rho$ saturated to η. If $\epsilon_{\rho-1} = -1$, then it is sufficient to invert the signs of the above calculations.

Since $\widetilde{r}_{des,\rho-\delta}$ must be set to 0 for $0 \le \rho \le \delta$, one can distinguish between two different phases in the online overflow handling process. The first phase corresponds to $0 \le \rho \le \delta$, while the second phase corresponds to $\delta + 1 \le \rho \le W + \delta$. In the first phase, one constrains the desired online result digit $\widetilde{r}_{des,\rho-\delta}$ to be equal to 0, further imposing that $\epsilon_\rho = \widetilde{P}_\rho$. As a consequence, an overflow value must be generated as soon as $\widetilde{P}_\rho$ reaches a value outside the digit set $\{-1, 0, 1\}$. If the first non-zero online operation result digit does not belong to the set $\{-1, 0, 1\}$, then overflow occurs, (c.f. Theorem 22). In the case of a potential overflow, true overflow occurs if $\widetilde{r}_{\rho-\delta}$ is different from $\beta - 1$ or $1 - \beta$ (whichever is appropriate), (c.f.

Theorem 22). In the second phase, the desired online result digit is not constrained to be equal to 0, but true, potential, or no overflow situations can be considered as mentioned previously.

Algorithm 26 *Online Overflow Handling for OSD Representations*

```
Step 1 Set $\rho \leftarrow 1$, and $\epsilon_0 \leftarrow 0$,
Step 2 Read $\tilde{r}_{\rho-\delta}$,
Step 3 Calculate $\tilde{P}_\rho \leftarrow \tilde{r}_{\rho-\delta} + \epsilon_{\rho-1}\beta$
Step 4 If $\rho \leq \delta$, then set $\tilde{r}_{des,\rho-\delta} \leftarrow 0$ else set $\tilde{r}_{des,\rho-\delta} \leftarrow sat(\tilde{P}_\rho, \eta)$ end
Step 5 If $(\tilde{P}_\rho - \tilde{r}_{des,\rho-\delta}) > 1$ (true if $\epsilon_{\rho-1} = OVF$), then set $\epsilon_\rho \leftarrow OVF$ else if $(\tilde{P}_\rho -$
       $\tilde{r}_{des,\rho-\delta}) < -1$ (true if $\epsilon_{\rho-1} = -OVF$), then set $\epsilon_\rho \leftarrow -OVF$ else set $\epsilon_\rho \leftarrow$
       $\tilde{P}_\rho - \tilde{r}_{des,\rho-\delta}$ end
Step 6 Set $\rho \leftarrow \rho + 1$,
Step 7 If $\rho < W + 1$, then go to Step 2, else done end if.
```

In the above algorithm, each digit of the online operation result is read one by one, the MSD first. Its value is then combined with that of a scaled online error in an online partial result $\tilde{P}_\rho$, where the scaled online error may indicate an overflow. A digit $\tilde{r}_{des,\rho-\delta}$ of equal weight to that of the input online operation result digit is either set to 0 for the first δ MSDs, or obtained by saturating the online partial result value to $-\eta$ and η, as required for the desired OSD format. Saturation occurs when overflow is indicated by an overflowed online error, or in a potential overflow situation. The resulting online error is determined as the subtraction of the online partial result $\tilde{P}_\rho$, but where an absolute value of $\tilde{P}_\rho$ strictly larger than 1 yields an overflowed online error. Finally, the iteration number is updated.

As one can observe, the online error takes on 5 different values, namely $-OVF$, -1, 0, 1, and OVF. Of course, -1 and 1 correspond to a situation of potential overflow. As a result, three bits are required to represent the online error: OVF indicates an overflow by a logic 1, $POVF$ indicates a potential overflow by a logic 1, and S indicates a strictly negative potential or true overflow by a 1. Then, a table for the conversion of OVF_ρ, $POVF_\rho$, and S_ρ into ϵ_ρ is given in Table 3.4. Next, ϵ_ρ is obtained recursively in accordance with

Table 3.4: Conversion of OVF_ρ, $POVF_\rho$, and S_ρ into ϵ_ρ

OVF_ρ	$POVF_\rho$	S_ρ	ϵ_ρ
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	−1
1	0	0	OVF
1	0	1	$-OVF$
1	1	0	OVF
1	1	1	$-OVF$

$$OVF_\rho \leftarrow \left(\left|\tilde{P}_\rho - \tilde{r}_{des,\rho-delta}\right| > 1\right) \tag{3.152}$$

$$POVF_\rho \leftarrow \left(\left|\tilde{P}_\rho - \tilde{r}_{des,\rho-delta}\right| = 1\right) \tag{3.153}$$

$$S_\rho \leftarrow S_{\rho-1} \mid \left(\overline{S_{\rho-1}} \frown \overline{(OVF_{\rho-1} \mid POVF_{\rho-1})} \frown (\tilde{r}_{\rho-\delta} < 0)\right), \tag{3.154}$$

where $\bar{\ }$, $\frown$, and $\mid$ represent the NOT, AND, and OR logic operations, respectively.

Overflow Situations in Online Arithmetic Operations

The type of overflow handled by the proposed algorithm assumes that the online operation result is accurate. However, this may not be the case in an algorithm for MAC operation. If the result is fed back as the addend, then the value of the accumulation may overflow its fixed-point representation. This results in a feedback of the accumulation value as an initialization value of the online error that exceeds the upper bound on the online error. This situation often occurs in digital filters incorporating online arithmetic operations (Brackert et al., 1989; Ercegovac and Lang, 1989; Fernando and Ercegovac, 1992; Lapointe et al., 1993; McQuillan and McCanny, 1995).

It is important to note that two different operations make use of the redundancy of the representation of their result, namely constant-delay addition and online processing, thus both may generate correctable overflow. As a result, the online partial result may itself require overflow correction. This is due to the fact that the addition of the online error with the off-line partial update is obtained by a constant-delay adder.

It has been found that algorithms can be developed to correct an overflow if a result $\widetilde{R}$ represented in accordance with

$$\widetilde{R} = \sum_{i=-1}^{W-1} \widetilde{r}_i 2^{-i}; \quad \widetilde{r}_i \in \{-1, 0, 1\} \tag{3.155}$$

is known to belong to $[-1, 1]$ or $[-1.5, 1.5]$, where W represents the wordlength of the corrected result (Timmermann and Hosticka, 1993). If one assumes that $\widetilde{R}$ will belong to the range $[-1.5, 1.5]$, being the result of the addition of a SB addend A of wordlength W represented in accordance with

$$A = \sum_{i=0}^{W-1} a_i 2^{-i}; \quad a_i \in \{-1, 0, 1\} \tag{3.156}$$

with a SB augend B represented in accordance with

$$B = \sum_{i=2}^{W-1} b_i 2^{-i}; \quad b_i \in \{-1, 0, 1\}, \tag{3.157}$$

then it is possible to avoid generating the correctable overflow altogether by restoring the carry propagation during the addition of the two first MSDs of the result $\widetilde{r}_0$ and $\widetilde{r}_1$. This is due to the facts that, (a) during a carry propagation, a carry can only be created if the absolute value of the sum of the input digits (whose weight is assumed to be 1) exceeds 2, (b) true overflow ($\widetilde{r}_{-1} = \widetilde{r}_0$ both equal to 1 or -1) cannot occur because of the restriction of the range of $\widetilde{R}$, and (c) potential overflow ($\widetilde{r}_{-1}$ equal to -1 or 1 and $r_0 = 0$) cannot occur. It can be readily shown that (a) and (b) are valid. To show the validity of (c), one can consider six cases, three of which are negatives of the other three. Then, $\widetilde{r}_1$ is either equal to 1 or to 0, and $\widetilde{R}$ cannot belong to the desired range, or $\widetilde{r}_1$ equals -1, which cannot occur by forcing the carry to propagate.

3.5 Chapter Summary

In this chapter, a general algorithm has been developed for bit-serial online signed-digit multiply-accumulate operation. Its predominant salient feature consists of novel operation-independent variables, which permit the development of a bit-serial online algorithm for any operation as long as it can be expressed as a sum of partial operation updates.

A novel signed-binary to minimally redundant base-4 recoding technique has been introduced, which permits a more speed-efficient reduction of the number of partial updates, substantially increasing the processing speed of a corresponding architecture. Digits appear in excess in both the least and the most significant parts of the results of such online operations. Therefore, algorithms have been presented for online signed-binary IEEE 754 standard for rounding to nearest/even and online signed-digit overflow handling.

One MAC operation has been developed: the multiplicand A is known at the outset, the multiplier B and the addend C are made available in an online fashion. By using a multiplicand known at the outset, one reserves the possibility to increase the wordlength of B without increasing the hardware requirements. Moreover, the wordlength of A can be increased as desired, since the delay of a corresponding architecture does not depend on it.

If A were consumed in an online fashion, then an incremental multiplication technique would be required (c.f. (Irwin, 1977)). In this situation, the partial MAC update is formed as the addition of two words of wordlengths linearly increasing with the iteration number. It is important to note that, although these wordlengths increase, the delay of a corresponding implementation is still a constant (limited-carry addition). Therefore, since the hardware requirements cannot be infinite, a maximum wordlength for A and B must be set.

One MAC operation has been developed: A is known at the outset, B and C online. By using a multiplicand known at the outset, one reserves the possibility to increase the wordlength of B without increasing the hardware requirements. Moreover, the wordlength of the multiplicand A can be increased as desired, since the delay of a corresponding architecture does not depend on it.

If A were input in an online fashion, then the incremented multiplication technique would be required, c.f. Irwin. In this situation, the partial MAC update is formed as the addition of two words of linearly increasing wordlengths. It important to note that, although these wordlengths increase, the delay of a corresponding implementation is still a constant (limited-carry addition). Therefore, since the hardware requirements cannot be infinite, a maximum wordlength for A AND B must be set.

Chapter 4

Architecture for Online Signed-Digit Digit-Serial Multiplication

4.1 Introduction

Digital signal processing algorithms often require the accumulation of numerous multiplication results. A multiply-accumulate (MAC) operation can efficiently perform such operations by returning a non-rounded intermediate result R in accordance with

$$R = AB + C, \tag{4.1}$$

where A represents the multiplicand, B the multiplier, and C the addend. Then, the intermediate result R is fed back as the addend C for accumulation.

In Chapter 2, the digit-serial and online arithmetic techniques were described, and efficient addition architectures were developed. Chapter 3 discussed the IEEE 754 SB RNE rounding technique. This chapter is concerned with, (a) the development of an algorithm for digit-serial purely signed-binary online multiplication employing the IEEE 754 RNE rounding technique, (b) the development of a corresponding architecture for subsequent FPGA hardware implementation, and (c) the comparison of the gate-level speed and area performances of the proposed architecture to existing similar architectures, illustrating a combination of the digit-serial and online arithmetic techniques.

Section 4.2 introduces the bit-serial purely signed-binary online multiplication and the bit-serial online IEEE 754 SB RNE rounding algorithms in a data stream context. Section 4.3 combines the above bit-serial online algorithms into a single digit-serial algorithm for purely signed-binary digit-serial online multiply-and-round operation. Section 4.4 describes a corresponding architecture for subsequent FPGA hardware implementation. Finally, Section 4.5 presents simulation results to demonstrate that the resulting architecture is functionally correct, discusses a throughput improvement via re-pipelining (Rao, 1996), and compares the resulting architecture to existing similar architectures by measuring their respective throughputs and efficiencies[1] using speed and area gate-level estimates.

4.2 Algorithm for Bit-Serial Signed-Binary Online Multiplication

This section is concerned with the development of an algorithm for bit-serial signed-binary online multiplication. A data stream-based framework is used, which will permit the transformation of the bit-serial algorithm into a corresponding digit-serial algorithm without the need for an architecture or for a digit-serial unfolding algorithm.

Consider a stream $\mathcal{A}$ of SB multiplicands of wordlength W_a whose digits are made available in parallel, i.e. all digits of a word are available at a given time instant. Similarly, consider a stream $\mathcal{B}$ of SB multiplier words of wordlength W_b and effective wordlength $W_{b,eff}$ (c.f. Subsection 3.4.1) whose digits are made available in a bit-serial online fashion, i.e. digit-by-digit, the MSD first. These two data streams will be combined to obtain the output stream $\tilde{\mathcal{R}}_{rnd}$ of IEEE 754 SB RNE-rounded multiplication results of effective wordlength $W_{b,eff}$. The online multiplication operation has a latency δ_{mul}, and the online rounding operation has a latency δ_{rnd}.

The bit-parallel stream of SB multiplicand words $\mathcal{A}$ of wordlength W_a is represented by W_a sequences ${a_{i,n}}_{n\geq 0}$, where i belongs to the set $\{1, 2, \ldots, W_a\}$, and where n represents the time instant. At time instant n, the multiplicand word is given by

$$A_w = \sum_{i=1}^{W_a} a_{i,n} 2^{-i}; \quad a_{i,n} \in \{-1, 0, 1\}, \tag{4.2}$$

where w represents the operation number and is given by

$$w = \left\lfloor \frac{n}{W_{b,eff}} \right\rfloor . \tag{4.3}$$

[1]The efficiency is calculated as the quotient of the throughput and the area of a given architecture.

Given that $W_{b,\mathit{eff}}$ consecutive values of n yield the same operation number w, the multiplicand digits of the w-th operation must be made available $W_{b,\mathit{eff}}$ time instants in a row. Therefore, for any $w \in \mathbb{N}$, for any $i \in \{1, \ldots, W_a\}$, and for any $\rho \in \{2, \ldots, W_{b,\mathit{eff}}\}$,

$$a_{i,wW_{b,\mathit{eff}}+\rho-1} = a_{i,wW_{b,\mathit{eff}}}, \tag{4.4}$$

must hold, where ρ represents the iteration number of the w-th operation in accordance with

$$\rho = 1 + (n \bmod W_{b,\mathit{eff}}) \, . \tag{4.5}$$

One can observe that w and $\rho - 1$ represent the quotient and remainder of the division of n by $W_{b,\mathit{eff}}$, respectively, and thus satisfy

$$n = wW_{b,\mathit{eff}} + \rho - 1. \tag{4.6}$$

The bit-serial stream of SB multiplier digits $\mathcal{B}$ is represented by a sequence $(b_n)_{n\geq 0}$, where b_n belongs to the SB digit set $\{-1, 0, 1\}$. The w-th SB multiplier word B_w can be defined in accordance with

$$B_w = \sum_{\rho=1}^{W_{b,\mathit{eff}}} b_{wW_{b,\mathit{eff}}+\rho-1} 2^{-\rho}, \tag{4.7}$$

where w belongs to $\mathbb{N}$, and where ρ not only represents the iteration number (as before), but also the index of the corresponding multiplier digit in the w-th multiplier word.

The non-rounded SB online multiplication result stream $\widetilde{\mathcal{R}}_{mul}$ of effective wordlength $W_{b,\mathit{eff}}$ is represented by the SB sequence $(\tilde{r}_{mul,n})_{n\geq 0}$, where $\tilde{r}_{mul,n} \in \{-1, 0, 1\}$. By using Eqns. 4.3 and 4.5, the w-th SB non-rounded online multiplication result word $\widetilde{\mathcal{R}}_{mul,w}$ can be defined in accordance with

$$\widetilde{\mathcal{R}}_{mul,w} = \sum_{\rho=1}^{W_{b,\mathit{eff}}} \tilde{r}_{mul,wW_{b,\mathit{eff}}+\rho-1} 2^{\delta_{mul}-\rho}. \tag{4.8}$$

It is important to observe that at iteration ρ, the online multiplication result digit of weight $2^{\delta_{mul}-\rho}$ is generated.

Finally, the IEEE 754 SB RNE-rounded online multiplication result stream $\widetilde{\mathcal{R}}_{rnd}$ of effective wordlength $W_{b,\mathit{eff}}$ is represented by the SB sequence $(\tilde{r}_{rnd,n})_{n\geq 0}$, where $\tilde{r}_{rnd,n} \in \{-1, 0, 1\}$.

Let us recall that the w'-th online rounding operation happens between iterations 2 and $W_{b,eff}$ of the w'-th online multiplication operation, and at iteration 1 of the $w'+1$-th online multiplication operation (c.f. Subsection 3.4.1). Therefore, the w'-th SB IEEE 754 SB RNE-rounded online multiplication result word $\widetilde{\mathcal{R}}_{rnd,w'}$ is defined in accordance with

$$\widetilde{\mathcal{R}}_{rnd,w'} = \sum_{\rho'=1}^{W_{b,eff}} \widetilde{r}_{rnd,w'W_{b,eff}+\rho'-1} 2^{\delta_{mul}+\delta_{rnd}-\rho'}, \tag{4.9}$$

where the word number w' is given by

$$w' = \left\lfloor \frac{n-1}{W_{b,eff}} \right\rfloor, \tag{4.10}$$

and where the index ρ' is given by

$$\rho' = 2 + (n-1) \bmod W_{b,eff}. \tag{4.11}$$

Together, w' and $\rho' - 2$ represent the quotient and remainder of the division of $n-1$ by $W_{b,eff}$, and thus satisfy

$$n = w'W_{b,eff} + \rho' - 1. \tag{4.12}$$

Again, it is important to note that at iteration ρ', the online rounded multiplication result digit has a weight $2^{\delta_{mul}+\delta_{rnd}-\rho'}$.

Given the repetitive nature of the definition of the bit-serial streams, the operations on the input data streams have to be repeated every $W_{b,eff}$ time instants. Therefore, it is sufficient to consider the multiplication of a single multiplicand A by a single multiplier B to yield a single online multiplication result R, corresponding to a MAC operation where $C = 0$, but the notion of time must be taken into account to schedule initializations as necessary. The algorithm developed in Subsection 3.2.2 can thus be used as a bit-serial online multiplication algorithm, but must be modified so as to be compatible with the algorithm described in Subsection 3.4.1 for online IEEE 754 SB RNE rounding. Subsequently, the resulting online multiplication and rounding algorithms can be applied successively to perform the desired online multiply-and-round algorithm.

Let us perform the required modifications to the bit-serial online multiplication algorithm. The result digit of weight 2^{-W_b} is generated at iteration $\rho_0 = W_b + \delta_{mul}$. Therefore, the scaled online error must be cleared at iteration ρ_0+1. Moreover, the effective wordlength of B is determined by taking into account the latencies δ_{mul} of the multiplication operation and δ_{rnd} of the rounding operation. In this way, the multiplier digits having indices $wW_{b,eff}+\rho_0+1$ through $wW_{b,eff}+W_{b,eff}-1$ must be set to the algebraic value 0, where $w \in \mathbb{N}$, and where the effective wordlength $W_{b,eff}$ is given by

$$W_{b,eff} = W_b + \delta_{mul} + \delta_{rnd}. \tag{4.13}$$

In Subsection 3.4.1, the rounding operation latency δ_{rnd} was calculated to be equal to 2. The latency δ_{mul} of the online multiplication operation will be determined shortly.

Algorithm 27 *Bit-Serial Signed-Binary Online Multiplication With Online Error Clearing*

```
Step 1  Set $\rho \leftarrow 1$,
Step 2  Set $\epsilon_{mul,0} \leftarrow 0$,
Step 3  Read $b_\rho$,
Step 4  Calculate $P_{mul,\rho} \leftarrow A_w b_\rho 2^{-\delta_{mul}}$,
Step 5  If $\rho = \rho_0 + 1$ then set $\epsilon^{scaled}_{mul,\rho-1} \leftarrow 0$ else set $\epsilon^{scaled}_{mul,\rho-1} \leftarrow 2\epsilon_{mul,\rho-1}$
Step 6  Calculate $\widetilde{P}_{mul,\rho} \leftarrow \epsilon^{scaled}_{mul,\rho-1} + P_\rho$,
Step 7  Round $\widetilde{P}_{mul,\rho}$ to $\widetilde{r}_{mul,\rho-\delta_{mul}}$,
Step 8  Calculate $\epsilon_{mul,\rho} \leftarrow \widetilde{P}_{mul,\rho} - \widetilde{r}_{mul,\rho-\delta_{mul}}$,
Step 9  Write $\widetilde{r}_{mul,\rho-\delta_{mul}}$,
Step 10 Calculate $\rho \leftarrow \rho + 1$,
Step 11 Store $\epsilon_{mul,\rho}$,
Step 12 If $\rho > W_{b,eff}$, then go to Step 3, else done.
```

In the above algorithm, the iteration number ρ is initialized to 1, and the online multiplication error at the iteration previous to 1 (i.e. 0), $\epsilon_{mul,0}$, is initialized to 0. Then, an off-line partial multiplication update $P_{mul,\rho}$ is calculated, and the scaled online multiplication error at the previous iteration, $\epsilon^{scaled}_{mul,\rho-1}$, is either set to the value of the online multiplication error at the previous iteration $\epsilon_{mul,\rho-1}$ multiplied by 2 (i.e. scaled up by one digit position), or cleared if the iteration number is equal to ρ_0+1. Next, $P_{mul,\rho}$ and $\epsilon^{scaled}_{mul,\rho-1}$ are combined to form an online partial multiplication result, $\widetilde{P}_{mul,\rho}$, which is rounded to a single-digit integer $\widetilde{r}_{mul,\rho-\delta_{mul}}$ referred to as the online multiplication result digit. This low-precision rounding (c.f. Subsection 3.4.1) yields a new online multiplication error $\epsilon_{mul,\rho}$. Finally, the online multiplication error $\epsilon_{mul,\rho}$ is stored for use in the next iteration, the iteration number is updated, and the process is repeated. The multiplication operation is complete when the iteration number is strictly larger than the effective wordlength $W_{b,eff}$.

Let us introduce the notion of time (as opposed to the notion of iteration) back into the proposed algorithm. At a time instant n corresponding to an iteration 1 of the proposed algorithm, the online error should be cleared. However, since $\epsilon^{scaled}_{mul,\rho_0}$ is set to zero in the previous multiplication operation, and since the digits of B at all the following iterations (ρ_0+1, through $W_{b,eff}-1$) are equal to zero, the online multiplication error remains equal to zero during these iterations. Therefore, the new multiplication operation does not require a clearing of the online error, except at the first time instant.

The internal wordlength γ can be determined leading to the latency δ_{mul} of the algorithm, and yielding the latency of the multiply-and-round operation. For SB numbers, the radix β is equal to 2, implying $\eta_r = \eta_{\bar{p}} = \eta_b = \eta_a = 1$, where $\eta_c = 0$ for a multiplication operation. The bound l_A on the absolute value of A_w can be calculated to be equal to $1-2^{-W_a}$. As a result, the bound on the off-line partial multiplication update l_P is given by

$$l_P = 2^{-\delta_{mul}} - 2^{-W_a-\delta_{mul}}, \tag{4.14}$$

which leads to a lower bound on the latency $\delta_{mul,\min}$ given by

$$\delta_{mul,\min} = \left\lceil \log_2\left(\frac{1-2^{-W_a}}{\frac{1}{2}-2^{1-\gamma}}\right)\right\rceil. \tag{4.15}$$

It can be seen that the minimum value that can be chosen for γ is 3. If $\gamma = 3$, then $\delta_{mul} = 2$. As the latency of the online rounding operation δ_{rnd} is equal to 2, the effective wordlength of the multiplier becomes

$$W_{b,eff} = W_b + 4. \tag{4.16}$$

4.3 Extension to a Digit-Serial Signed-Binary Online Multiply-and-Round Algorithm

Bit-serial (BS) operations process one digit of a data stream per BS time instant $n \in \mathbb{N}$, whereas digit-serial (DS) operations process D consecutive digits of the same data stream per DS time instant $n' \in \mathbb{N}$. Since D consecutive digits of a given data stream correspond to D consecutive BS time instants, a DS algorithm consists of D parallel sub-algorithms (numbered $d \in \{0, 1, \ldots, D-1\}$) invoking one iteration of the BS algorithm. Of course, all the D parallel sub-algorithms interact with each other to yield the desired result. Consequently, DS time instant/sub-algorithm number pair (n', d) corresponds exactly to the BS time instant n as given by

$$n = Dn' + d. \tag{4.17}$$

Moreover, a word number/iteration number pair (w, ρ) or (w', ρ') also corresponds exactly to one particular BS time instant (c.f. Eqns. 4.6 and 4.12). These unique decompositions of the BS time instant are at the root of the (sometimes complicated) equations underlying all DS sub-algorithms.

The resulting algorithm for DS SB online multiplication and rounding is given below.

Algorithm 28 *Digit-Serial Signed-Binary Online Multiplication and Rounding*

```
Step 1 Set $n' \leftarrow 0$,
Step 2 Read $b_{Dn'}, b_{Dn'+1}, \ldots$, and $b_{Dn'+D-1}$,
Step 3 For $d$ in $\{0, 1, \ldots, D-1\}$ do,
    Step 3.a Calculate $w \leftarrow \left\lfloor \frac{Dn'+d}{W_{b,eff}} \right\rfloor$ and $\rho \leftarrow 1 + (Dn'+d) \bmod W_{b,eff}$,
    Step 3.b Perform steps 3 through 9 for iteration $\rho$ of the bit-serial online mul-
             tiplication algorithm, making use of the multiplicand word $A_w$ and the
             multiplier digit $b_{Dn'+d}$.
    Step 3.c Calculate $w' \leftarrow \left\lfloor \frac{Dn'+d-1}{W_{b,eff}} \right\rfloor$ and $\rho' \leftarrow 2 + (Dn'+d-1) \bmod W_{b,eff}$,
    Step 3.d Perform steps 2 through 7 for iteration $\rho'$ of the bit-serial online round-
             ing algorithm on the $w'$-th online multiplication result,
    End
Step 4 Store $\epsilon_{mul,Dn'+D-1}$, $\tilde{r}_{mul,Dn'+D-1}$, and $\epsilon_{rnd,Dn'+D-1}$,
Step 5 Calculate $n' \leftarrow n' + 1$, and go to Step 2
```

At a given DS time instant n', D consecutive digits of the BS multiplier digit sequence are read, separated, and consumed by the D BS algorithm invocations (or sub-algorithms). The word numbers w and w' and iteration numbers ρ and ρ' are calculated as given by

Eqns. 4.3, 4.5, 4.10, and 4.11, by calculating the BS time instant n from the DS time instant n' and sub-algorithm number d in accordance with Eqn. 4.17. These numbers are subsequently used to determine the set of elementary operations that must be performed at the given DS time instant to perform the invocation of the desired iteration of the BS algorithm. It can be observed that the word number w may be different for two invocations of the BS algorithm at a given DS time instant n', i.e. two or more operations may be computed in the DS algorithm at a given time instant. The number of operations being computed in the DS algorithm is smaller than two if one chooses $D \leq W_{b,eff}$.

In the BS algorithm, the online error generated at the BS time instant n is consumed as the online error at the previous iteration at the BS time instant $n+1$. From the uniqueness of the decomposition of the BS time instant n into a DS time instant n' and a sub-algorithm number d, it follows that an online error generated by the d-th BS algorithm invocation at the n'-th DS time instant is used as an input by the $(d+1) \bmod D$-th BS algorithm invocation at the n'-th DS time instant if d belongs to the set $\{0, 1, \ldots, D-2\}$. However, if $d = D-1$, then $(Dn'+d)+1 = Dn'+D$ which can be re-written as $Dn'+D = D(n'+1)+0$. Therefore, the online error generated by the $D-1$-th BS algorithm invocation at the n'-th DS time instant must be stored for use by the 0-th BS algorithm invocation at the $n'+1$-th DS time instant. Similarly, the online result digit and online multiplication error generated by the $D-1$-th BS multiplication algorithm invocation must be delayed by one DS time instant, and be fed to the 0-th invocation of the BS rounding algorithm.

Let us denote by L the least common multiple of D and $W_{b,eff}$. Then, consider the BS time instants 0 to $L-1$, and calculate the DS time instants and sub-algorithm numbers at which a given iteration ρ^* occurs. If one assumes $W_{b,eff} = 3$, $D = 2$ —yielding $L = 6$—, and $\rho^* = 1$, then iteration ρ^* occurs at BS time instants 0 and 3. The corresponding DS time instant and sub-algorithm number pairs are (0,0) and (1,1). Moreover, at DS time instant 2, iteration ρ^* does not occur. Next, let us repeat this process for BS time instants L to $2L-1$. Iteration ρ^* occurs at BS time instants 6 and 9. The corresponding DS time instants and sub-algorithm numbers are (3,0) and (4,1). Moreover, at DS time instant 5, iteration ρ^* does not occur. One can observe a cyclic pattern, which emerges from three different decompositions of the BS time instant into a quotient and a remainder. The first two decompositions use the effective wordlength $W_{b,\mathit{eff}}$ as the divisor to obtain the word numbers (w and w'). The third decomposition uses the digit-size D as the divisor to obtain the DS time instants (n').

One can make use of the above cyclic pattern by observing that some invocations do not require performing a particular iteration of the BS algorithm. In the above example, if $\rho^* = \rho_0 + 1$ and if $\rho^* = 1$, then iteration $\rho_0 + 1$ is never performed by the sub-algorithm number 2. As a result, the conditional clearing operation need not be performed for the sub-algorithm number 2. More generally, if a conditional statement is true only for a given iteration, and if that iteration is not performed under a given sub-algorithm number, then the translation of that sub-algorithm to a corresponding architecture does not require translating the conditional statement. Step 3 of the simplified DS unfolding algorithm (c.f. Subsection 2.3.1) calculates the DS time instants and sub-algorithm numbers at which conditional statements must be performed, and uses the results to determine which conditional statements need to be implemented. This technique has been used in many occasions to reduce the hardware requirements (Parhi, 1991; Satyanarayana and Nowrouzian, 1996; Rao, 1996).

One can calculate the DS time instances and sub-algorithm numbers for iteration 1 in accordance with

$$n' = \left\lfloor \frac{1 + iW_{b,\mathit{eff}}}{D} \right\rfloor, \quad \text{and} \quad d = (1 + iW_{b,\mathit{eff}}) \bmod D, \tag{4.18}$$

where i successively takes on all the values from the set $\{0, 1, \ldots, \frac{\operatorname{lcm}(D, W_{b,\mathit{eff}})}{W_{b,\mathit{eff}}} - 1\}$. Similarly, the DS time instants n' and sub-algorithm numbers d corresponding to iteration $\rho_0 + 1$ can be calculated in accordance with

$$n' = \left\lfloor \frac{\rho_0 + 1 + iW_{b,\mathit{eff}}}{D} \right\rfloor, \quad \text{and} \quad d = (\rho_0 + 1 + iW_{b,\mathit{eff}}) \bmod D, \tag{4.19}$$

where i successively takes on all the values from the set $\{0, 1, \ldots, \frac{\operatorname{lcm}(D, W_{b,\mathit{eff}})}{W_{b,\mathit{eff}}} - 1\}$. If $W_{b,\mathit{eff}}$ is not a constant, i.e. if it changes dynamically, then Algorithm 13 (page 29) can be used for the on-the-fly determination of the control bits indicating the desired iterations. In that situation, one must have $D < W_{b,\mathit{eff}}$.

4.4 Architecture for Digit-Serial Signed-Binary Online Multiplication and Rounding

In this section, the proposed digit-serial algorithm is translated to a corresponding architecture for a subsequent FPGA or ASIC hardware implementation.

The resulting architecture consists of three main types of units, namely, multiplicand selection units, online multiplication units, and online rounding units as shown in Figure 4.1 for general values of the digit-size and wordlengths of the multiplicand and multiplier.

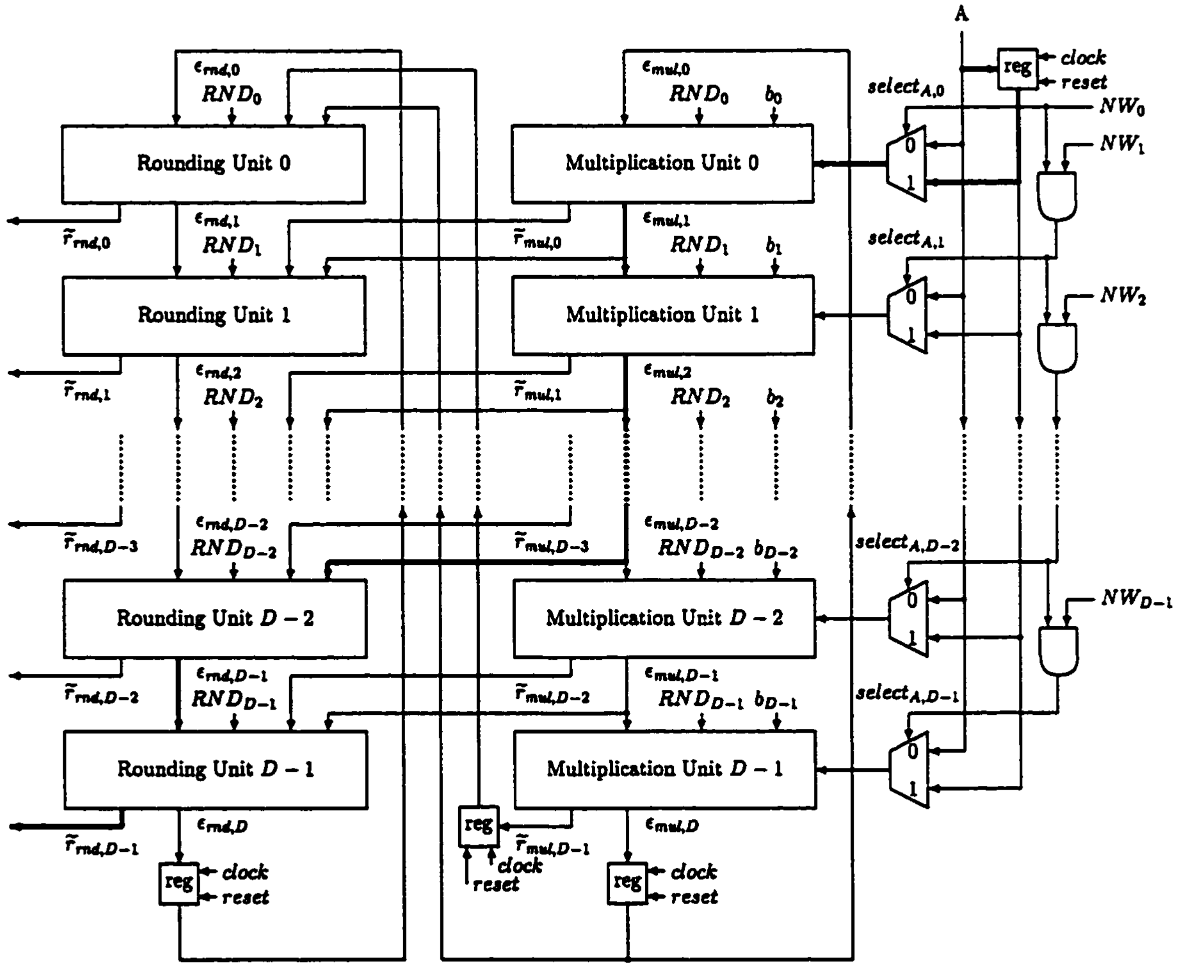


Figure 4.1: Architecture for Digit-Serial Online Multiply-and-Round Operation

The architecture contains two types of control signals, namely RND_d and NW_d. First, the "new word" signal NW_d indicates iteration 1 (i.e. a new operation) in the d-th online multiplication invocation when it is set to the logic value 0. The DS time instants and sub-algorithm numbers corresponding to these iterations are given by Eqn. 4.18. Second, the "round" signal RND_d indicates iteration $\rho_0 + 1$ (i.e. the iteration for rounding) in the d-th online multiplication invocation when it is set to the logic value 0. The DS time instants and sub-algorithm numbers corresponding to these iterations are given by Eqn. 4.19.

The d-th online multiplication unit and the d-th multiplicand selection unit implement Steps 3 through 9 of the BS algorithm for online multiplication (c.f. Algorithm 27, page 94). Similarly, the d-th online rounding unit implements Steps 2 through 7 of the BS algorithm for online IEEE 754 SB RNE rounding (c.f. Algorithm 21, page 74). The d-th multiplication selection unit consists of a multiplexer controlled by a $select_{A,d}$ signal, and a logic gate performing an AND operation between $select_{A,d-1}$ and a signal NW_d if $d > 0$. The units will be described individually in the following subsections. The critical path[2] of the proposed architecture is shown in bold lines. One can observe that a signal x_d carries the variable value $x_{Dn'+d}$ at the DS time instant n'.

Now, let us describe how signals flow in the above architecture. As a new multiplicand A is made available, the d-th selection unit ($0 \leq d \leq D-1$) selects the appropriate multiplicand to be fed to the d-th multiplication unit. Then, the multiplicand A, the multiplier digit b_d, and the online error $\epsilon_{mul,d}$ —whose value is controlled by RND_d— are combined to yield a result digit $\tilde{r}_d$ and a new online error $\epsilon_{mul,d+1}$. Of course, $\epsilon_{mul,d+1}$ is fed to the online multiplication unit $(d+1) \bmod D$, and if $d = D-1$, then the online error $\epsilon_{mul,D}$ —generated by the multiplication unit $D-1$— is delayed by one DS time instant. A delay by one DS time instant is obtained by feeding a signed-binary signal to a register "reg," where each digit of the online error is assigned a pair of D-type flip-flops (DFFs). The input signals are latched on the positive edge of a *clock* signal, and the output signals of every pair of DFFs in the register are initialized to 01 (code for algebraic 0) by a general active-low *reset* signal.

At time instant n', the online multiplication result digit $\tilde{r}_{mul,d}$ and the online multiplication error $\epsilon_{mul,d+1}$ are fed to the online rounding unit $(d+1) \bmod D$. Again, if $d = D-1$, then the results are delayed by one DS time instant. Afterwards, the d-th online rounding unit combines these values with that of the input online rounding error, and generates an

[2]The critical path of an architecture corresponds to the path of longest delay between an input and an output of the architecture

online rounded result digit $\tilde{r}_{rnd,d}$ and an online rounding error $\epsilon_{rnd,d+1}$. This online error is fed to the online rounding unit $(d+1) \bmod D$. As before, the online error of the rounding unit number $D-1$ must be delayed by one DS time instant so as to be fed to the rounding unit 0 appropriately. The signal $\tilde{r}_{rnd,d}$ at the DS time instant n' carries the variable value $\tilde{r}_{Dn'+d}$.

4.4.1 Multiplicand Selection Unit

If one assumes $D < W_{b,eff}$, the iterations performed by the D online multiplication units belong to two separate multiplication operations at most. As a result, it is sufficient for one to make available in parallel the new multiplicand value A, and to delay that value for use in the next DS time instant. Consequently, one reduces the multiplicand wire requirements from $D \times W_a$ to W_a (by a factor D). The following gives a description of the corresponding architecture.

Let us assume that at a given DS time instant n', the control signal NW_{d_0} $(0 \leq d_0 \leq D-1)$ is equal to logic 0, indicating that the d_0-th online multiplication unit performs the first iteration of an operation. Then, the online multiplication units 1 through $d_0 - 1$ must use the stored value of the multiplicand, while the online multiplication units d_0 through $D-1$ must use the new multiplicand A. This selection is obtained by setting selection control bits $select_{A,0}$ through $select_{A,d_0-1}$ to logic 1, and $select_{A,0}$ through $select_{A,d_0-1}$ to logic 0. This is obtained by setting

$$select_{A,0} \leftarrow NW_0, \tag{4.20}$$

and by setting

$$select_{A,d} \leftarrow NW_d \hat{\ } select_{A,d-1} \tag{4.21}$$

for all d in $\{1, 2, \ldots, D-1\}$, where $\hat{\ }$ represents the logic AND operation. These operations are explicitly shown in Figure 4.1. Finally, the routing of the appropriate multiplicand value to the corresponding online multiplication unit is obtained by multiplexers.

4.4.2 Online Multiplication Units

An online multiplication unit implements all the elementary operations required at any iteration of the proposed algorithm for bit-serial online multiplication. The corresponding architecture is shown in Figure 4.2. This unit features an addition/low-precision rounding

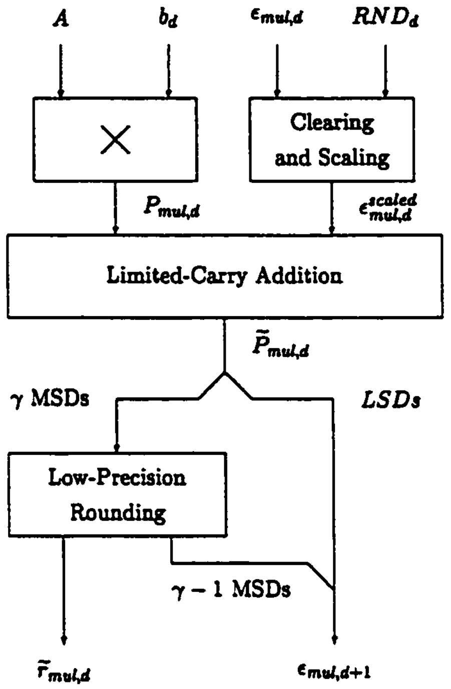


Figure 4.2: Architecture of an Online Multiplication Unit

core unit that can also be utilized for operations different from multiplication. The resulting added practical advantage originates from the design of the online algorithms in terms of operation-independent variables.

The multiplication of the multiplicand word A by the multiplier digit b_d is performed by an array of single-digit multipliers (c.f. Subsection 3.3.1). Then, the result is scaled down by δ_{mul} digit positions to yield the off-line partial multiplication update $P_{mul,d}$. Next, the online error $\epsilon_{mul,d}$ is cleared if RND_d is equal to logic 0 (c.f. Subsection 3.3.1), scaled up, and then a zero digit is padded as its LSD, resulting in $\epsilon^{scaled}_{mul,d}$. Zero digits are padded as MSDs of $P_{mul,d}$ to conform to the format of $\epsilon^{scaled}_{mul,d}$. Subsequently, the $\epsilon^{scaled}_{mul,d}$ is added by cascaded limited-carry adders to $P_{mul,d}$ to generate an online partial multiplication result $\widetilde{P}_{mul,d}$. It is important to remember that the addition of the last two MSDs is obtained by carry-propagate adders (c.f. Subsection 3.4.1), while the other digits are combined by using limited-carry propagation adders (c.f. Subsection 2.4.2). Further, the γ MSDs of $\widetilde{P}_{mul,d}$ are taken for low-precision rounding to the online multiplication result digit $\widetilde{r}_{mul,d}$ (c.f. Subsection 3.4.1), generating the MSDs of the online multiplication error $\epsilon_{mul,d+1}$. The

LSDs of the online partial multiplication result form the LSDs of the online multiplication error.

The addition and low-precision rounding units can be can be merged into a single core unit for the implementation of other recursive SB online operations as, for example, online rounding units (Natter and Nowrouzian, 2000b) (c.f. Subsection 3.2.2). Of course, the wordlength of the online partial update may differ from operation to operation, subsequently necessitating the adaptation of the wordlengths of the online error and the online partial result.

4.4.3 Online Rounding Units

An online rounding unit implements all the elementary operations required at any iteration of the algorithm for bit-serial online rounding. The proposed architecture is shown in Figure 4.3.

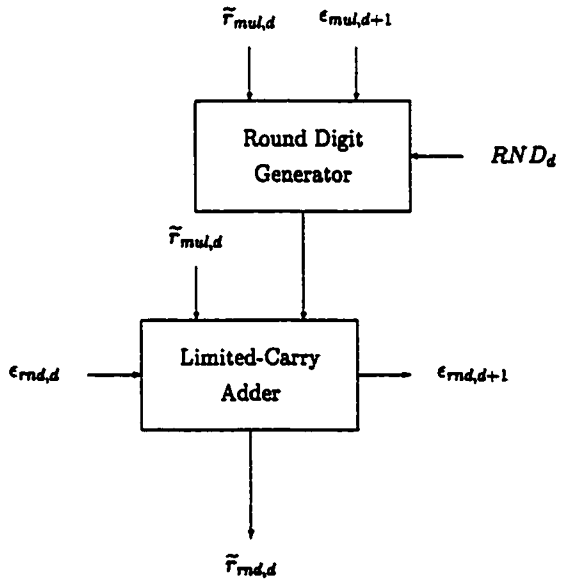


Figure 4.3: Architecture of an Online Rounding Unit

The online multiplication error and result digit are used to generate an IEEE 754 SB RNE round digit, as described in Subsection 3.4.1. The round digit is cleared (set to the algebraic value 0) if RND_d is equal to logic 1, as opposed to 0 for the online multiplication error in the online multiplication unit. Therefore, it is always cleared, except at iteration $\rho_0 + 1$, i.e. when the online multiplication error represents the LSDs of the final online multiplication result. Then, the round digit and the online multiplication result digit are added to the input carries (representing the online rounding error) by a limited-carry adder, generating an online rounded digit and new carries. The carries of the $D-1$-th adder are stored for use in the next DS time instant by the 0-th adder.

4.5 Computer Simulation Results and Performance Comparison

The correct functionality of the proposed architecture is confirmed by computer simulations. Then, its speed and area performances will be measured, and subsequently compared to those of existing architectures.

4.5.1 Simulation Results

In the following, simulation results are given to confirm the correct functionality of the proposed architecture. The wordlengths of the multiplicand and multiplier have both been chosen to be equal to 8. The latencies of the online multiplication and rounding algorithms are both equal to 2, as determined previously. Two values of the digit-size, 2 and 3, have been chosen for the demonstration of the correct functionality under varying digit-size values. In this way, the same test vectors will be applied to two different implementations to verify their correct functionality by comparing their outputs against the expected results. A VHDL code implementing the proposed architecture was developed by using Max+Plus II, compiled by using the underlying Synopsys FPGA Design Compiler into a target FPGA referenced as EPF 10K20 RC240-4, and simulated by using Max+Plus II.

The multiplicand, multiplier, and result values as expressed in decimal are listed in Table 4.1. Both the full-precision ($A \times B$) and the expected IEEE 754 RNE-rounded ($\tilde{R}_{rnd}$) results are given. The corresponding signed-binary values of the multiplicand, multiplier, and expected rounded result are listed in Table 4.2. The simulation results, as expressed in hexadecimal values, are shown in Figure 4.4, and correspond to a digit-size of 2. The correspondence between hexadecimal and signed-binary values is listed in Table 4.3.

In order to verify the results, it is important to remember that the signed-binary input and output representations flow in an online fashion, i.e. the MSD first. Moreover, the MSD of the result of the first operation appears at the BS time instant 1 (the first BS time

Table 4.1: Test Vectors for Digit-Serial Multiply-Round Operation in Decimal

A	B	$A \times B$	$\widetilde{R}_{rnd}$
-0.19921875	-0.86328125	0.1719818115234375	0.17187500
0.16796875	0.28515625	0.0478973388671875	0.04687500
-0.21875000	-0.51562500	0.1127929687500000	0.11328125
-0.57031250	-0.95312500	0.5435791015625000	0.54296875
0.96484375	-0.83203125	-0.8027801513671875	-0.80468750

Table 4.2: Test Vectors for Digit-Serial Multiply-Round Operation in Signed-Binary

A	B	$\widetilde{R}_{rnd}$
$0.\,0\,0\,\bar{1}\,\bar{1}\,0\,0\,\bar{1}\,\bar{1}$	$0.\,\bar{1}\,\bar{1}\,0\,\bar{1}\,\bar{1}\,\bar{1}\,0\,\bar{1}$	$0\,0\,0\,0.\,0\,1\,\bar{1}\,1\,\bar{1}\,1\,0\,0$
$0.\,0\,0\,1\,0\,1\,0\,1\,1$	$0.\,0\,1\,0\,0\,1\,0\,0\,1$	$0\,0\,0\,0.\,0\,0\,0\,1\,\bar{1}\,1\,0\,0$
$0.\,0\,0\,\bar{1}\,\bar{1}\,\bar{1}\,0\,0\,0$	$0.\,\bar{1}\,0\,0\,0\,0\,\bar{1}\,0\,0$	$0\,0\,0\,0.\,0\,0\,1\,0\,\bar{1}\,1\,0\,1$
$0.\,\bar{1}\,0\,0\,\bar{1}\,0\,0\,\bar{1}\,0$	$0.\,\bar{1}\,\bar{1}\,\bar{1}\,\bar{1}\,0\,\bar{1}\,0\,0$	$0\,0\,0\,0.\,1\,0\,0\,1\,\bar{1}\,1\,\bar{1}\,1$
$0.\,1\,1\,1\,1\,0\,1\,1\,1$	$0.\,\bar{1}\,\bar{1}\,0\,\bar{1}\,0\,\bar{1}\,0\,\bar{1}$	$0\,0\,0\,\bar{1}.\,0\,1\,0\,\bar{1}\,0\,0\,1\,0$

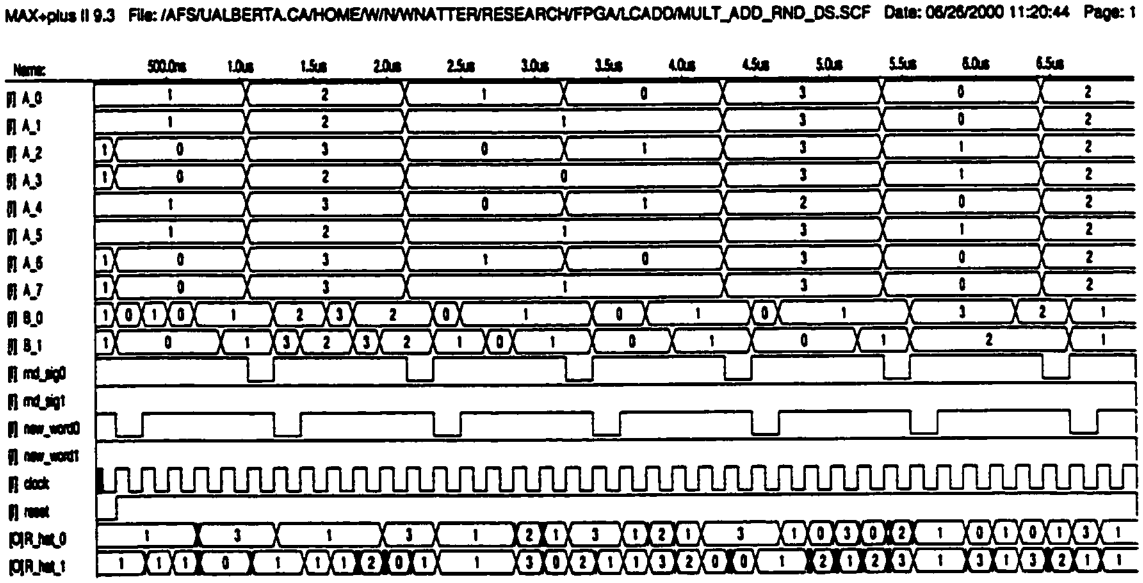


Figure 4.4: Simulation Results for $D = 2$ and $W_a = W_b = 8$

instant is 0), corresponding to the DS time instant 0 and the sub-algorithm number 1 in the above simulation results.

The same vectors have been applied to a proposed architecture with digit-size 3, as shown in Figure 4.5. From both simulations, it can be concluded that the results are as

Table 4.3: Hexadecimal to Signed-Binary Conversion Table

Hex.	SB
0	-1
1	0
2	0
3	1

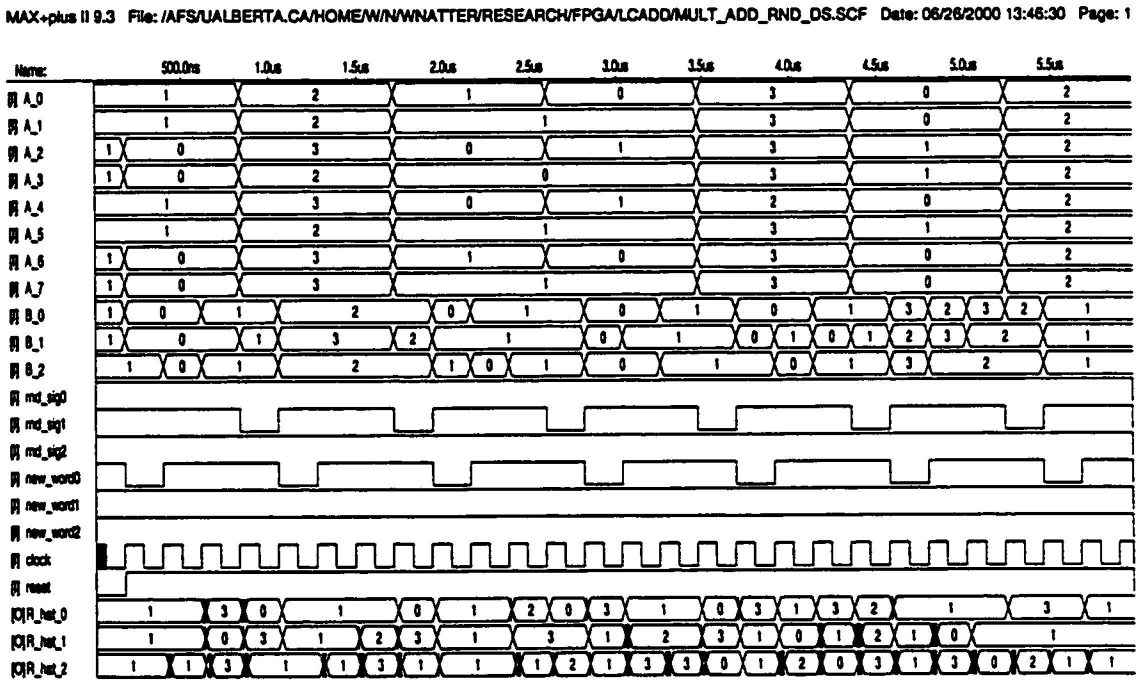

MAX+plus II 9.3 File: /AFS/UALBERTA.CA/HOME/W/N/WNATTER/RESEARCH/FPGA/LCADD/MULT_ADD_RND_DS.SCF Date: 06/26/2000 13:46:30 Page: 1

Figure 4.5: Simulation Results for $D = 3$ and $W_a = W_b = 8$

expected, and that the functionality of the proposed architecture is thus correct.

4.5.2 Throughput Parameterization and Improvement Via Re-Pipelining

The delay along the critical path, shown in bold lines in Figure 4.1, will be calculated in terms of the multiplicand wordlength and the digit-size. The critical path originates from the LSD of the newly latched multiplicand. Then, a signal following the critical path goes through a selection unit and a single-digit signed-binary multiplier. Next, the signal propagates diagonally (because of the scaling of the online error) through $D-2$ limited-carry adders and $D-3$ online error clearing units (or 0 if $D < 3$). Further, the signal goes through an IEEE 754 SB RNE round digit generator, and one limited-carry adder in the $D-2$-th online rounding unit. Finally, the signal propagates through one limited-carry adder before reaching the online rounded result digit $\tilde{r}_{rnd,D-1}$ in the $D-1$-th online rounding unit.

Two components are dominant in the above critical path, namely, the limited-carry adders in the multiplication units, and the IEEE 754 SB RNE round digit generator. The first component is due to the unfolding of the bit-serial architecture. In fact, consider the digit-serial unfolding of a mere bit-serial full-adder with a digit-size equal to the wordlength. The result is a bit-parallel carry-propagate adder, whose critical path is stretched across the digit-size (Rao, 1996). However, in the proposed architecture, the required IEEE 754 rounding of the result stretches the critical path across the horizontal direction for computation of the sign and sticky bits.

Let us find ways to improve the throughput of the proposed architecture. One may reduce the delay through the critical path by employing a look-ahead technique for the computation of the sign and sticky bits. Another solution, proposed in (Rao, 1996), is to re-pipeline the proposed architecture. Re-pipelining consists of cutting the critical path in a given architecture into sub-paths of approximately equal length and introducing latches in the cut locations. The signals are thus transmitted from one set of latches to the next, the delay on the critical path now corresponding to a fraction of the original critical path. As a result, the throughput of the architecture is increased considerably. However, one needs to wait for as many clock cycles as there are pipelining stages to obtain the output, and the area of a corresponding ASIC hardware implementation is considerably increased.

Because of the non-homogeneity of the proposed architecture in the MSD part of the multiplication unit, it is quite complicated to offer a parameterized solution to its re-pipelining. Therefore, only the pipeline cut delivering the minimum delay is described. The proposed architecture is shown in Figure 4.6, where the re-pipelining cuts are shown in vertical dashed lines, where $W_a = 5$, and where $D = 4$. It is important to point out that the selection units have not been represented to simplify the figure. The selection units are located just before the single-digit multipliers, between two pipe cuts. In this way, only A, its delayed version, and the $select_{A,d}$ control bits have to be pipelined, instead of A_0 through A_{D-1}.

Figure 4.6 is arranged as Figure 4.1 in order to identify each component with its corresponding unit. A square box with a "+" inside represents a limited-carry adder. An

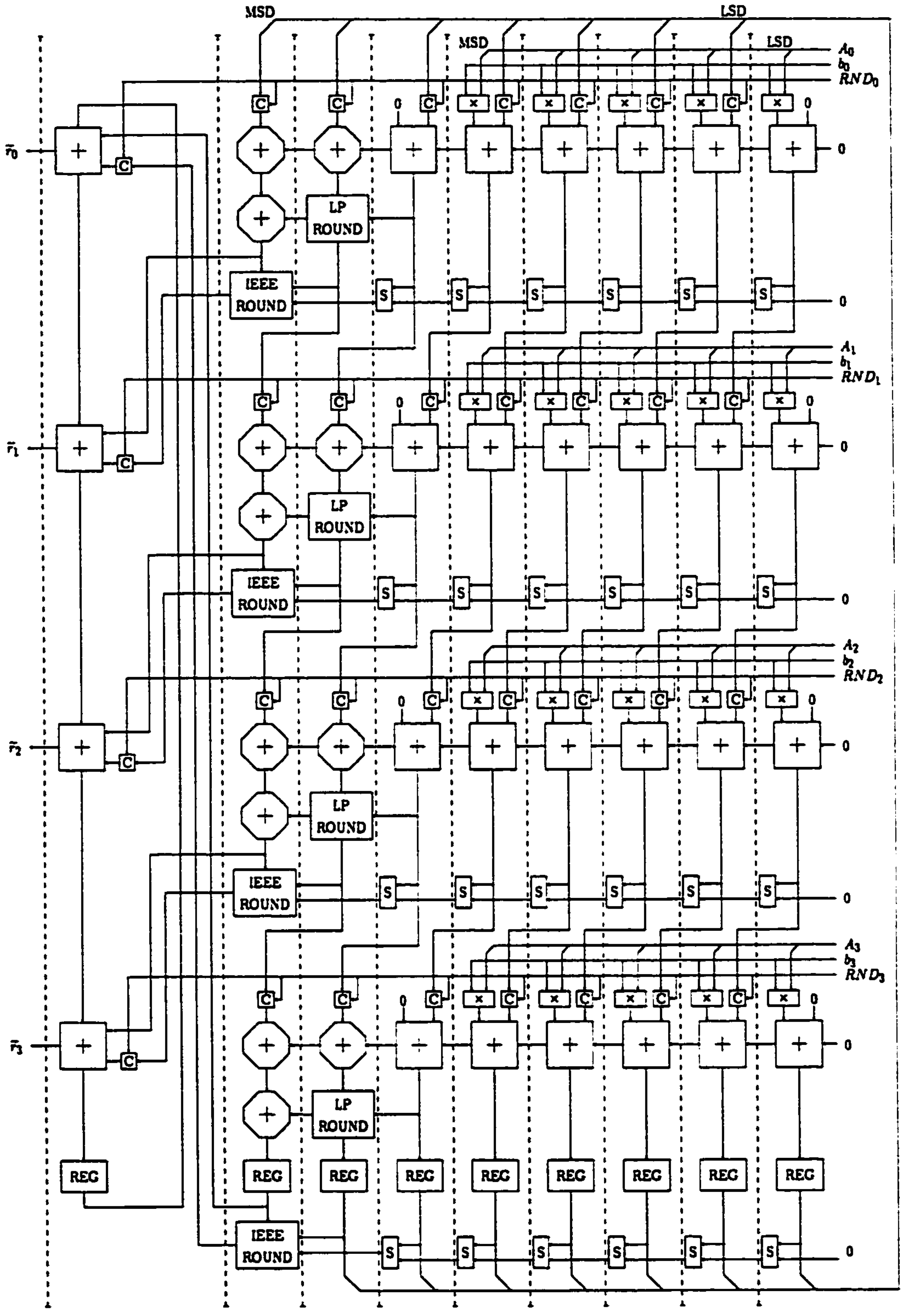


Figure 4.6: Re-Pipelined Architecture for MAC Operation and IEEE 754 RNE Rounding, with $W_a = 5$ and $D = 4$

octagonal box with a "+" inside represents a carry-propagate adder. A square box with a "C" inside represents a single-digit clearing unit. A square box with a "×" inside represents a single-digit multiplier. A rectangular box with a "S" inside represents a sign and sticky computation unit. Finally, a rectangular box with "LP ROUND" or "IEEE ROUND" represents a unit generating a low-precision or IEEE 754 SB RNE round digit, respectively. It can be observed that the LSD to MSD direction of limited-carry addition, implemented using 4:2 compressors (Kanie et al., 1994), has been used for the accumulation of partial multiplication updates, because re-pipelining can be applied intuitively. However, the MSD to LSD direction of limited-carry addition has been used for the addition of the round digits with the result digits to suit the description of an algorithm for digit-serial online addition (Natter and Nowrouzian, 1999).

A digit of weight 2^{-i} of the online multiplication error $\epsilon_{mul,D}$ is fed as a digit of weight 2^{-i+1} of the online multiplication error $\epsilon_{mul,0}$ two DS time instants after being generated.

The longest delay T between two re-pipelining cuts is obtained in accordance with

$$T = 10\tau_g + \tau_{mux}. \tag{4.22}$$

This longest delay results from the combination of one clearing unit, two carry-propagate adders, and an IEEE 754 SB RNE round digit generation unit. There are $W_a + \delta_{mul} + 2$ pipelining stages. A result of effective wordlength $W_{b,eff}$ is thus obtained in $W_a + \delta_{mul} + \left\lceil \frac{W_{b,eff}}{D} \right\rceil + 1$ DS time instants.

4.5.3 Area Parameterization

The proposed architecture makes use of a number of elements described in the previous chapters. The area requirement for gate-level implementation is given in Table 4.4.

Some reductions in the hardware requirements are possible. First, one need not calculate the carry-out digit of the carry-propagate adder generating the MSD of $\widetilde{P}_{mul,d}$. Second, one can use the sign and sticky bits generated for IEEE 754 rounding as sign and sticky bits for low-precision rounding, thereby also allowing a reduction in the latency of online multiplication. As a result of these reductions, the total area of a corresponding ASIC hardware implementation of the proposed re-pipelined architecture can be calculated in

Table 4.4: Area Requirement for the Multiplication Architecture

Unit	Area		Number of Units
-	Gates	Multiplexers	-
Limited-Carry Adder	4	2	$(W_a + \delta_{mul})D$
Carry-Propagate Adder	13	2	$3D$
Low-Precision Rounding	$6\gamma - 7$	-	D
Multiplier	-	2	DW_a
Clearing	2	-	$(W_a + \delta_{mul} + 1)D$
Selection	$D - 1$	$2W_aD$	1
D-flip-flop	6	-	$2W_a^2 + 10W_aD + 22D + 2\delta_{mul} + 5$
RNE Rounding	$6W_a + 6\delta_{mul} - 5$	2	D

accordance with

$$A = (12W_a^2 + 72DW_a + 12D\delta_{mul} + 168D + 12\delta_{mul} + 29)a_g + (6DW_a + 2D\delta_{mul} + 8D)a_{mux}. \tag{4.23}$$

where a_g and a_{mux} represent the area of a gate and a multiplexer, respectively.

4.5.4 Performance Comparisons

Let us assume $\tau_g = 1\text{ns}$, $\tau_{mux} = 1.5\text{ns}$, $a_g = 25\mu\text{m}^2$, and $a_{mux} = 37.5\mu\text{m}^2$. Moreover, let us assume that W_a and W_b have the same value denoted by W (to simplify the comparisons). Then, the throughput H of the proposed re-pipelined architecture is obtained by using

$$H = \frac{D}{WT}. \tag{4.24}$$

Next, the efficiency of an architecture is calculated as the quotient of its throughput and its total area. It is extremely important to note that neither the latency of the online operation nor an estimate of the power consumption of a corresponding hardware implementation is taken into account in the above efficiency calculation.

The proposed re-pipelined architecture is compared to the re-pipelined architecture for LSD-first digit-serial multiplication and IEEE 754 rounding of two's complement numbers proposed in (Rao, 1996). The main differences between these architectures are, (a) the number representation of the inputs and output, (b) the mode of the arrival of the multiplicand (bit-parallel or digit-serial), and (c) the direction (LSD-first versus MSD-first) of computation of the result.

A plot of the throughput versus the digit-size is shown in Figure 4.7 for various values of W. A plot of the logarithm of the efficiency versus the digit-size is shown in Figure 4.8

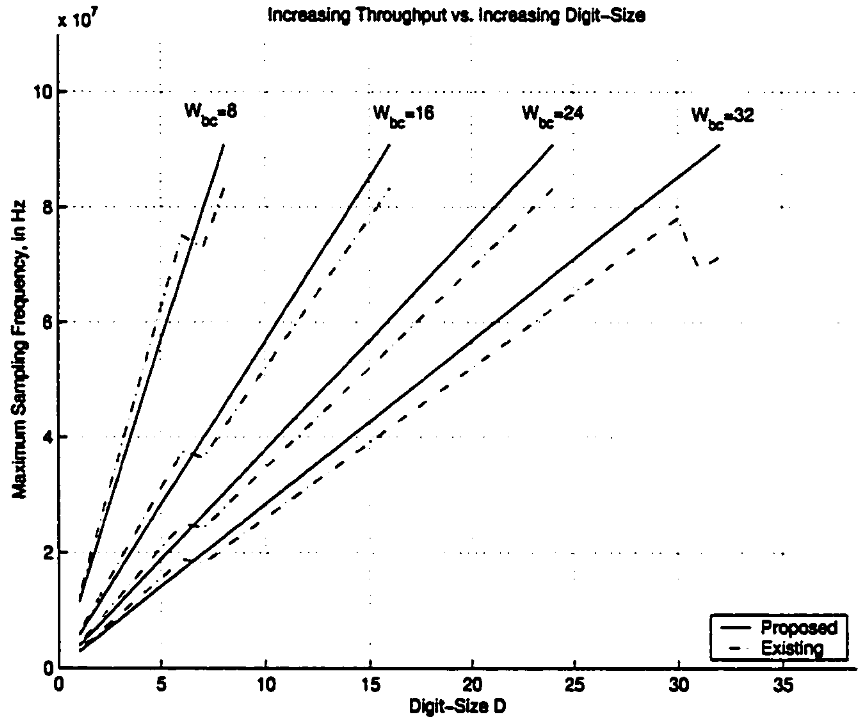


Figure 4.7: Throughput Versus Digit-Size of Proposed and Existing Digit-Serial Multiplication Architectures

for the same values of W. From these plots, it can be concluded that the throughput of the proposed architecture is approximately equal to that of the existing architecture. The non-linearities of the curves are due to modulo arithmetic effects. As in (Rao, 1996), the most efficient architecture is the bit-parallel one. However, the relative efficiency is much lower than the existing architecture, given that the vertical scale is logarithmic. This dramatic difference in terms of area performance arises from two fundamental differences between the architectures, namely, (a) the use of a redundant number representation (requiring large units for computing the result), and (b) the use of selectors to choose the appropriate multiplicand, also increasing largely the area of a corresponding ASIC hardware implementation. It can also be observed that the computation of the sign bit was not obtained by look-

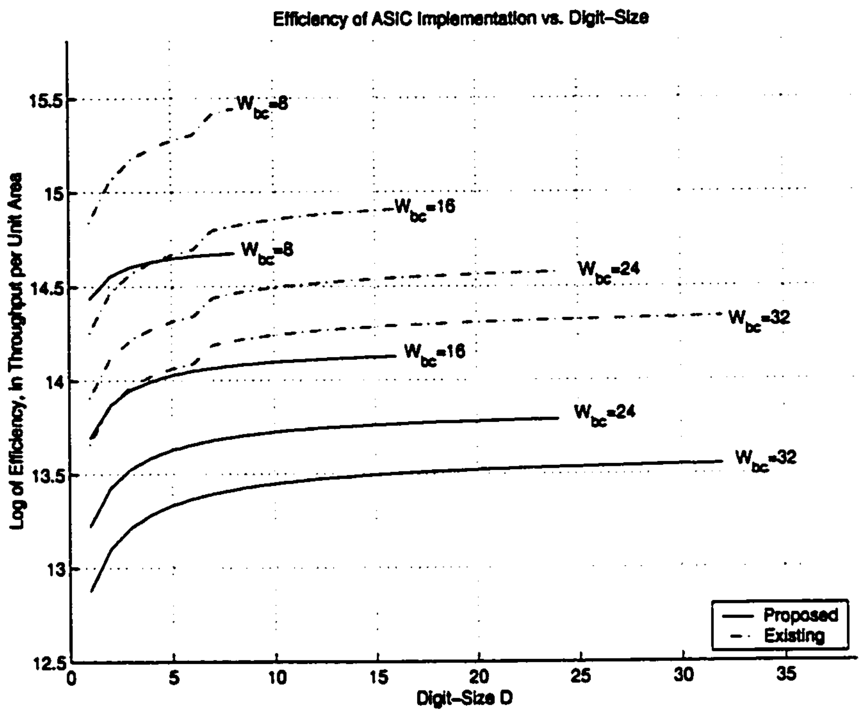


Figure 4.8: Log of Efficiency Versus Digit-Size of Proposed and Existing Digit-Serial Multiplication Architectures

ahead techniques, making the critical path longer. However, this point does not have a large impact on the results for the wordlengths in Figs. 4.7 and 4.8.

It is possible to make the multiplicand available in a bit-serial fashion in the bit-serial online algorithm for multiplication, as described in (Irwin, 1977) for signed-magnitude multiplicand and multiplier inputs. As a result, it is possible to make the multiplicand available in a digit-serial fashion in the digit-serial algorithm. However, in the resulting algorithm, the wordlength of the multiplier is constrained to be equal to that of the multiplicand. This becomes a drawback if the multiplicand is read from a memory, because the computation always requires W_a iterations. By contrast, the proposed algorithm can be developed for any multiplier wordlength, which becomes a definite asset in a dynamically changing multiplier wordlength case.

4.6 Conclusion

In this chapter, an algorithm has been developed for digit-serial online purely signed-binary multiplication employing IEEE 754 SB RNE rounding, illustrating the combination of the digit-serial and the online arithmetic techniques. A corresponding architecture for subsequent FPGA hardware implementation has been proposed for general values of the digit-size and multiplicand and multiplier wordlengths, and confirmed functionally correct through numerous simulations. The resulting architecture has been re-pipelined for throughput maximization. Gate-level speed and area estimates have been calculated. The efficiencies (ratio of throughput by area) of the proposed architecture were compared unfavorably to those of existing architectures, mainly because of additional low-precision rounding units required for MSD-first operation, and discrepancies between the operation themselves. Also, neither the power consumption nor the latency was taken into account in the efficiency comparisons. Therefore, additional work is required to determine suitability of digit-serial online operations for high-speed applications.

Chapter 5

Architecture for Online Signed-Binary Bit-Parallel Multiply-Accumulate Operation

5.1 Introduction

This chapter is concerned with the design and subsequent FPGA hardware implementation issues of a purely signed-binary bit-parallel online multiply-accumulate operation employing the SB to MRB4 multiplier recoding, IEEE 754 RNE rounding, and overflow correction techniques. This includes, (a) the presentation of an algorithm for bit-parallel online signed-binary MAC operation employing signed-binary to minimally redundant base-4 recoding and IEEE 754 result rounding to nearest/even, (b) the translation of the algorithm to a corresponding architecture for subsequent field-programmable gate array or application-specific integrated circuit implementation, and (c) the comparison of the resulting architecture to existing similar architectures.

In Section 5.2 an algorithm is developed for purely signed-binary bit-parallel online multiply-accumulate operation employing the SB to MRB4 recoding technique. Moreover, the format of the result of the operation conforms to that of the multiplicand and multiplier by the IEEE 754 SB RNE rounding technique and the overflow correction technique discussed in Chapter 3, allowing one to maintain a standard wordlength across a digital processor employing the proposed algorithm. Then, Section 5.3 describes a corresponding architecture for subsequent FPGA hardware implementation. Finally, Section 5.4 presents the simulation results demonstrating that the resulting architecture is functionally correct, and compares the throughputs and efficiencies of the architecture and existing similar architectures (via speed and area gate-level estimates).

Simulation results will be presented for verification of correct functionality. In order to improve the throughput and allow a comparison of the proposed architecture to existing similar architectures, Next, the proposed architecture is re-pipelined in order to both improve its throughput, and allow the comparison to existing similar architectures (Rao, 1996) by measuring their respective throughputs and efficiencies[1].

In Section 5.2, an algorithm is described for purely signed-binary bit-parallel online multiply-accumulation operation employing the SB to MRB4 recoding technique. Moreover, the result of the operation will be expressed in the same format as the input multiplier and multiplicand by applying the IEEE 754 SB RNE rounding technique and the overflow correction technique discussed in Chapter 3, thus allowing one to maintain a standard wordlength across a digital processor employing the proposed algorithm. Then, a corresponding architecture for subsequent FPGA hardware implementation will be presented. Speed and area gate-level estimates will allow a comparison with existing similar architectures.

5.2 Bit-Parallel Signed-Binary MAC Algorithm Employing SB to MRB4 Recoding

The signed-binary (SB) to minimally redundant base-4 (MRB4) recoding technique presented in Subsection 3.3.2 permits the reduction of the number of partial MAC updates by a factor of 2, increasing the speed of execution of a corresponding architecture by a factor of 2. In the following, an algorithm for bit-parallel online SB MAC operation employing SB to MRB4 multiplier recoding is developed.

Consider the SB result R^{SB} of the MAC arithmetic operation given by

$$R^{SB} = A^{SB}B^{SB} + C^{SB}, \tag{5.1}$$

where A^{SB}, B^{SB}, and C^{SB} are referred to as the SB MAC operation operands, and represent the SB multiplicand, the SB multiplier, and the SB addend, respectively.

[1]The efficiency will be measured as the ratio of the throughput by the area of a given architecture.

The SB multiplicand and multiplier can be defined in accordance with

$$A^{SB} = \sum_{i=1}^{W_a} a_i^{SB} 2^{-i}; \quad a_i^{SB}\{-1, 0, 1\}, \tag{5.2}$$

and

$$B^{SB} = \sum_{i=1}^{W_b} b_i^{SB} 2^{-i}; \quad b_i^{SB}\{-1, 0, 1\}, \tag{5.3}$$

where W_a and W_b represent the wordlengths of A and B, respectively. By padding A and B with an appropriate number of zeros as LSDs, the values of W_a and W_b can be assumed to be even in accordance with

$$W_a = 2W_a' \quad \text{and} \quad W_b = 2W_b', \tag{5.4}$$

where W_a' and W_b' are integers. As a result, one can use the SB to MRB4 recoding technique presented in Subsection 3.3.2 to obtain a corresponding MRB4 representation B^{MRB4} of B^{SB} given by

$$B^{MRB4} = \sum_{j=0}^{W_b'} b_j^{MRB4} 4^{-j}; \quad b_j^{MRB4} \in \{-2, -1, 0, 1, 2\}. \tag{5.5}$$

In this way, B^{MRB4} satisfies

$$B^{MRB4} = B^{SB}. \tag{5.6}$$

Since B^{MRB4} is expressed in a base-4 representation, the algorithm for online MAC operation must process base-4 numbers. The latency of the base-4 online MAC operation is denoted by δ_{mac}. Let us define the SB representation of the addend in accordance with

$$C^{SB} = \sum_{i=-1-2\delta_{mac}}^{W_a+W_b} c_i^{SB} 2^{-i}; \quad c_i^{SB}\{-1, 0, 1\}. \tag{5.7}$$

Then, one has to express A^{SB} and C^{SB} in equivalent base-4 representations A^{B4} and C^{B4}, respectively. One can obtain these representations by regrouping the odd and even digits, as seen in the first step of the SB to MRB4 recoding technique in Subsection 3.3.2, resulting in

$$A^{B4} = \sum_{i=1}^{W_a'} a_i^{B4} 4^{-i}; \quad a_i^{B4} \in \{-3, -2, -1, 0, 1, 2, 3\}, \tag{5.8}$$

and

$$C^{B4} = \sum_{i=\delta_{mac}}^{W'_a} a_i^{B4} 4^{-i}; \quad a_i^{B4} \in \{-3,-2,-1,0,1,2,3\}, \tag{5.9}$$

where the values are preserved in accordance with

$$A^{B4} = A^{SB} \quad \text{and} \quad C^{B4} = C^{SB}. \tag{5.10}$$

Of course, this recoding is totally parallel. Finally, the result of the MAC arithmetic operation on A^{B4}, B^{MRB4}, and C^{B4} is defined as

$$R^{B4} = A^{B4} B^{MRB4} + C^{B4}. \tag{5.11}$$

By substituting the base-4 numbers by their SB equivalents in Eqns. 5.6 and 5.10, and by comparing the result to Eqn. 5.1, the base-4 result is proven to satisfy

$$R^{B4} = R^{SB}. \tag{5.12}$$

The proposed algorithm for bit-parallel online SB MAC operation will be based on the recursive algorithm developed in Subsection 3.2.2. All the iterations of the proposed algorithm will be performed at the same time instant.

Let us decompose C^{B4} into two components in accordance with

$$C^{B4} = C^{B4}_{msw} + C^{B4}_{lsw}, \tag{5.13}$$

where

$$C^{B4}_{msw} = \sum_{j=-\delta_{mac}}^{W'_a-1} c_j^{B4} 4^{-j} \tag{5.14}$$

represents the most significant word of C^{B4}, and where

$$C^{B4}_{lsw} = \sum_{j=W'_a}^{W'_a+W'_b} c_j^{B4} 4^{-j} \tag{5.15}$$

represents the least significant word of C^{B4}. The first component, C^{B4}_{msw}, will be made available at the outset of the proposed algorithm, while the second component, C^{B4}_{lsw}, will be made available one digit per iteration.

Since the operation is performed in a bit-parallel fashion, A^{SB}, B^{SB}, and C^{SB} are available at the outset, i.e. at the first iteration. The digits of B^{MRB4} and C^{B4}_{lsw} are made available one per iteration, the MSD first. In this way, the words

$$B^{MRB4}_{\rho} = \sum_{i=0}^{\rho} b_i^{MRB4} 4^{-i} \tag{5.16}$$

and

$$C^{B4}_{lsw,\rho} = \sum_{i=W'_a}^{W'_a+\rho} c_i^{B4} 4^{-i} \tag{5.17}$$

are referred to as the partially known multiplier and LSW addend at iteration ρ, respectively. Next, the off-line iterative MAC result R^{B4}_{ρ} is given by

$$R^{B4}_{\rho} = A^{B4} B^{MRB4}_{\rho} + C^{B4}_{msw} + C^{B4}_{lsw,\rho}, \tag{5.18}$$

thus yielding the off-line partial MAC update

$$P^{B4}_{\rho} = \left(A^{B4} b^{MRB4}_{\rho} + c^{B4}_{W'_a+\rho} 4^{-W'_a} \right) 4^{-\delta_{mac}}, \tag{5.19}$$

where $i_C = W'_a$ (c.f. Subsection3.2.1). The resulting MAC operation can be performed by using Algorithm 29.

Algorithm 29 *Bit-Parallel Online Signed-Binary MAC Operation*

```
Step 1 Set $\rho \leftarrow 0$ and $\epsilon^{B4}_{-1} \leftarrow C^{B4}_{msw} 4^{-\delta_{mac}}$,
Step 2 Read $b^{MRB4}_{\rho}$ and $c^{B4}_{W'_a+\rho}$,
Step 3 Calculate $P^{B4}_{\rho} \leftarrow \left( A^{B4} b^{MRB4}_{\rho} + c^{B4}_{W'_a+\rho} 4^{-W'_a} \right) 4^{-\delta_{mac}}$,
Step 4 Calculate $\widetilde{P}^{B4}_{\rho} \leftarrow 4\epsilon^{B4}_{\rho-1} + P^{B4}_{\rho}$,
Step 5 Round $\widetilde{P}^{B4}_{\rho}$ to $\widetilde{r}^{B4}_{\rho-\delta_{mac}}$,
Step 6 Calculate $\epsilon^{B4}_{\rho} \leftarrow \widetilde{P}^{B4}_{\rho} - \widetilde{r}^{B4}_{\rho-\delta_{mac}}$,
Step 7 Write $\widetilde{r}^{B4}_{\rho-\delta_{mac}}$,
Step 8 Calculate $\rho \leftarrow \rho + 1$,
Step 9 If $\rho < W'_b + 1$, then go to Step 2, else done end.
```

In the above algorithm, the iteration ρ has to be initialized to 0 instead of 1 because the MSD of B^{MRB4} has a weight 4^0. Moreover, C^{B4}_{msw} is scaled down in the same manner as $A^{B4}b^{MRB4}_{\rho} + c^{B4}_{W'_a+\rho}4^{-W'_a}$ and fed as the initialization value of the scaled online error ϵ_{-1}. In this way, it is imperative for $C^{B4}_{msw}4^{-\delta_{mac}}$ to satisfy the same constraint as ϵ_{-1}, i.e.

$$\left|C^{B4}_{msw}4^{-\delta_{mac}}\right| \leq l_{\epsilon} \tag{5.20}$$

must hold true. Other than these two points, the above algorithm corresponds to a bit-serial online algorithm for MAC operation as described in Subsection 3.2.2. At the last iteration, the online error $\epsilon^{B4}_{W'_b}$ is padded to the online result word $\widetilde{R}^{B4}_{W'_b}$ to form the final result R^{B4}.

Let us calculate an upper bound on the addend. By applying Eqn. 5.20, one can obtain

$$\left|C^{B4}\right| \leq l_{\epsilon}4^{\delta_{mac}} + 4^{-W'_a} - 4^{-W'_a-W'_b}. \tag{5.21}$$

By observing that $l_{\epsilon} = \frac{1}{2} + (4^{1-\gamma} - 4^{-W'_a-\delta_{mac}})$, Eqn. 5.21 becomes

$$\left|C^{B4}\right| \leq \left(\frac{1}{2} + 4^{1-\gamma}\right)4^{\delta_{mac}} - 4^{-W'_a-W'_b}. \tag{5.22}$$

If Eqn. 5.22 is satisfied, then it is guaranteed that the above algorithm generates the MAC result as expected. Moreover, by applying Eqn. 3.12 (page 51) at iteration W'_b of the above algorithm, one obtains the base-4 MAC operation result in accordance with

$$R^{B4}_{W'_b} = \widetilde{R}^{B4}_{W'_b} + \epsilon^{B4}_{W'_b}4^{\delta_{mac}-W'_b}. \tag{5.23}$$

Then, Eqn. 5.18 implies that

$$R^{B4}_{W'_b} = A^{B4}B^{MRB4}_{W'_b} + C^{B4}_{msw} + C^{B4}_{lsw,W'_b} \tag{5.24}$$

holds. Next, by observing that $B^{MRB4}_{W'_b} = B^{MRB4}$ and that $C^{B4}_{lsw,W'_b} = C^{B4}_{lsw}$, Eqn. 5.11 yields

$$R^{B4}_{W'_b} = R^{B4}, \tag{5.25}$$

as desired.

It is very important to observe that Eqn. 5.23 implies that R^{B4} is represented in accordance with

$$R^{SB} = \sum_{i=-1-2\delta_{mac}}^{W_a+W_b} r^{SB}_i 2^{-i}; \quad \in \{-1,0,1\}, \tag{5.26}$$

which conforms to the representation of the addend C^{B4}. Therefore, R^{B4} can be fed back as the addend, permitting an accumulation of the MAC results, but requiring the addend, and the successive multiplicand and multiplier values to be constrained so as to yield a result R^{B4} satisfying Eqn. 5.22.

The MAC arithmetic operation online result that will be output must be expressed in the same format as the multiplier, so that one can use it as a multiplier in another online MAC arithmetic operation. The re-formatting of the result requires two operations, (a) rounding in accordance with the IEEE 754 RNE standard, and (b) overflow correction.

The MAC arithmetic operation result R^{SB} can be rounded in accordance with the IEEE 754 RNE standard as explained in Subsection 3.4.1. Therefore, R^{SB} is decomposed in accordance with

$$R^{SB} = R^{SB}_{msw} + R^{SB}_{lsw}, \tag{5.27}$$

where

$$R^{SB}_{msw} = \sum_{i=-1-2\delta_{mac}}^{W_b} r_i^{SB} 2^{-i} \tag{5.28}$$

represents its most significant word, and where

$$R^{SB}_{lsw} = \sum_{i=W_b+1}^{W_a+W_b} r_i^{SB} 2^{-i} \tag{5.29}$$

represents its least significant word, and a signed-binary rounded result R^{SB}_{rnd} is calculated, and is of the form

$$R^{SB}_{rnd} = \sum_{i=-2-2\delta_{mac}}^{W_b} r^{SB}_{rnd,i} 2^{-i}. \tag{5.30}$$

The extra digit of weight $2^{2+2\delta_{mac}}$ has been generated by the addition making part of the rounding operation.

Then, the rounded result R^{SB}_{rnd} is fed to the overflow detection and correction algorithm given in Subsection 3.4.2. Therefore, R^{SB}_{rnd} is decomposed in accordance with

$$R^{SB}_{rnd} = R^{SB}_{rnd,msw} + R^{SB}_{rnd,lsw}, \tag{5.31}$$

where

$$R^{SB}_{rnd,msw} = \sum_{i=-2-2\delta_{mac}}^{0} r^{SB}_{rnd,i} 2^{-i} \tag{5.32}$$

represents its most significant word, and where

$$R^{SB}_{rnd,lsw} = \sum_{i=1}^{W_b} r^{SB}_{rnd,i} 2^{-i} \tag{5.33}$$

represents its least significant word, and a final output result R is generated. The final result R may be saturated (to the maximum or minimum representable value), but the corresponding internal result may not be saturated. In this way, the extra MSDs of the R^{SB} required by the non-zero latency δ_{mac} act as guard digits.

It is important to observe that the online partial MAC result $\tilde{P}^{B4}_{\rho}$ is obtained as the addition of P^{B4}_{ρ} and $4\epsilon^{B4}_{\rho-1}$, but can also be viewed as the addition of $A^{B4} b^{MRB4}_{\rho} 4^{-\delta_{mac}}$ with $4\epsilon^{B4}_{\rho-1} + c^{B4}_{W_a+\rho} 4^{-\delta_{mac}-W'_a}$. This observation implies that, instead of padding ϵ^{B4}_{ρ} with a zero as its LSD in a corresponding architecture to obtain a multiplication by 4, one can pad $c^{B4}_{W_a+\rho}$ as the LSD, to calculate $4\epsilon^{B4}_{\rho-1} + c^{B4}_{W_a+\rho} 4^{-\delta_{mac}-W'_a}$. This is due to the fact that $i_c = W_a$ (c.f. Subsection 3.2.2).

Along with the development of the algorithm, one must determine the values of the internal wordlength γ and the latency δ_{mac} of the base-4 online MAC arithmetic operation. The digits of $\tilde{P}^{B4}_{\rho}$ are assumed to belong to the set $\{-3, -2, \ldots, 3\}$ such that the resulting addition architecture can be derived from limited-carry addition architectures (c.f. Subsection 2.4.1). In order to yield a small latency value, the digits $\tilde{r}^{B4}_{\rho-\delta_{mac}}$ are assumed to belong to $\{-3, \ldots, 3\}$. Consequently, the lower bound on the latency (c.f. Eqn. 3.32 page 57) is given by

$$\delta_{mac,\min} = \left\lceil \log_4 \left(\frac{1 - 4^{-W'_a}}{\frac{1}{2} - 4^{1-\gamma}} \right) \right\rceil . \tag{5.34}$$

Of course, the choice $\gamma = 1$ is not valid ($\frac{1}{2} - 1$ is negative). One can observe that if γ could be equal to 1, then no rounding of $\tilde{P}_{\rho}$ would be necessary. Then, the minimum valid γ value is 2, and happens to yield the smallest latency value $\delta_{mac} = 1$. As a result, one chooses $\gamma = 2$ and $\delta_{mac} = 1$. It is interesting to note that the latency of the resulting signed-binary operation is twice that of the base-4 operation.

5.3 Architecture for MAC Operation Employing Signed-Binary Multiplier Recoding

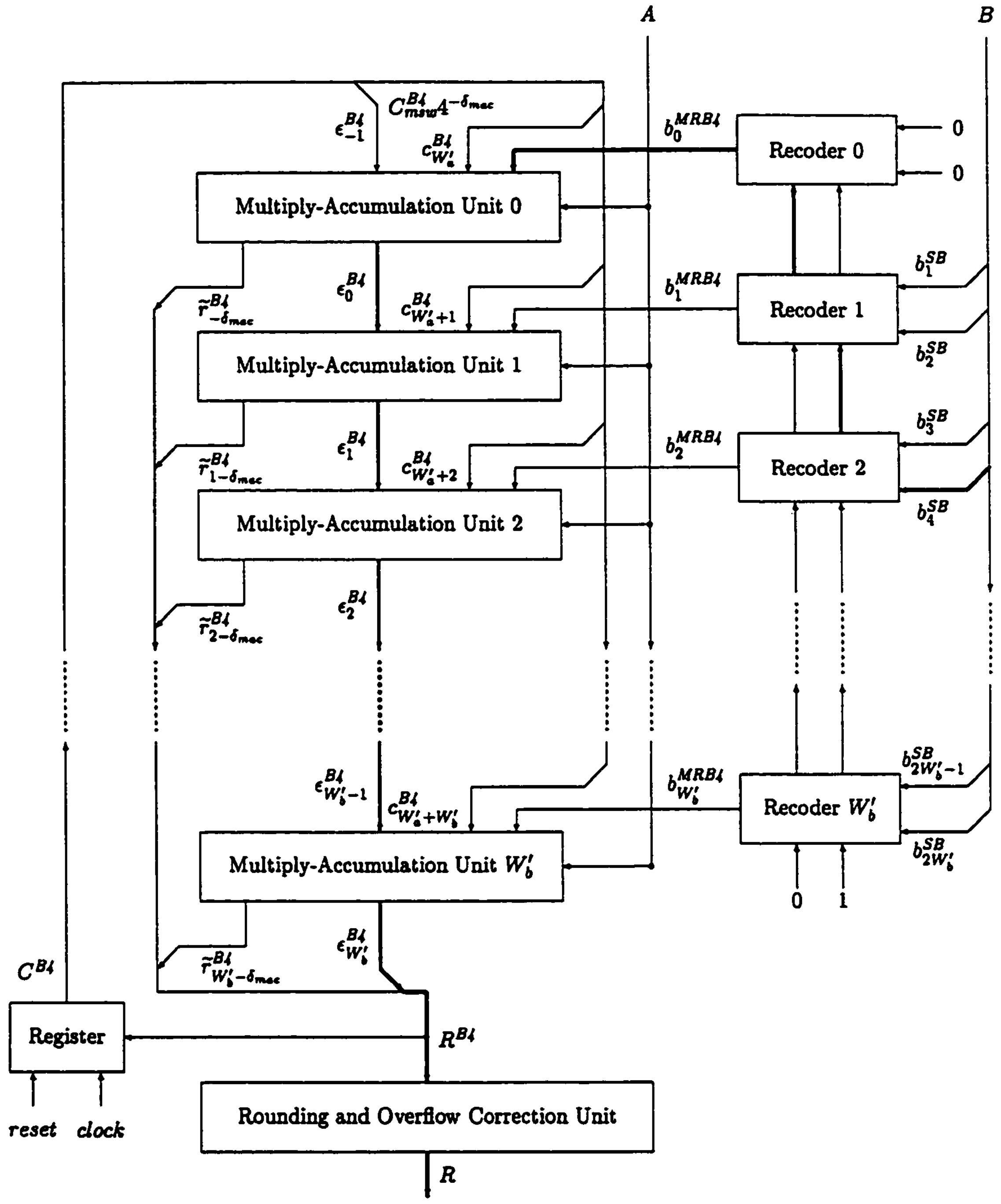
In this section, the proposed algorithm for bit-parallel online signed-binary multiply-accumulate operation will be translated to a corresponding architecture for a subsequent FPGA or ASIC hardware implementation. The resulting architecture is shown in Figure 5.1 for general values of the wordlengths of the input multiplicand and multiplier. This architecture comprises three main parts, namely, a bit-parallel multiplier recoding unit, serial online multiplication units, and a bit-parallel rounding and overflow unit. Each of these units will be described individually in the following subsections. The critical path of the proposed architecture is shown in bold lines.

At a given time instant, the SB multiplier B^{SB} is fed to the bit-parallel recoder unit described in Subsection 3.3.2. The resulting digits of the MRB4 representation B^{MRB4} of the multiplier are fed individually to each of the $W_b' + 1$ online multiplication units. The ρ-th multiplication unit takes as other inputs the current SB multiplicand value A^{SB} and the online error $\epsilon_{\rho-1}^{B4}$ generated in the $(\rho-1)$-th multiplication unit. One can note that both the online multiplication unit number and the index of the corresponding MRB4 multiplier digit correspond to the iteration number in the algorithm. Then, the result digits coming out of the online multiplication units are gathered to form $\tilde{R}_{W_b'}^{B4}$, and are combined to the digits of the online error $\epsilon_{W_b'}^{B4}$ of the last online multiplication unit, yielding the full-precision result R^{B4}, or, equivalently, R^{SB}.

Next, R^{B4} branches off into two different paths. In the first path, R^{SB} is delayed by one sample time by registers, as described in Section 4.3, and the delayed signal is fed back as the accumulation value C^{B4}. In the second path, R^{B4} is rounded in accordance with the IEEE 754 RNE standard, as described in Subsection 3.4.1, and fed to an overflow correction unit, as described in Subsection 3.4.2. The last two operations are required to restrict the format of the output result to match that of the multiplier and multiplicand.

5.3.1 Signed-Binary to Minimally Redundant Base-4 Recoders

The recoder is implemented as a cascade of signed-binary to minimally redundant base-4 recoders. These recoders are as described in Subsection 3.3.2.



A
B
Multiply-Accumulation Unit 0
Multiply-Accumulation Unit 1
Multiply-Accumulation Unit 2
Multiply-Accumulation Unit W_b'
Recoder 0
Recoder 1
Recoder 2
Recoder W_b'
Register
Rounding and Overflow Correction Unit
reset
clock
C^{B4}
R^{B4}
R
0
1

Figure 5.1: Architecture for Bit-Parallel Online Multiply-Accumulate Operation Employing SB to MRB4 Recoding

5.3.2 Online Multiplication Units

The online multiplication units employed here are slightly different from those described in Section 4.3 (page 95). Since the algorithm implemented by this architecture is based on radix-4 representations, the online error is scaled by 4 as opposed to 2. The first difference is that the online error is shifted up by two signed-binary digit positions instead of one. Then, the multiplier digits can take on values from the set $\{-2,-1,0,1,2\}$. Therefore, the multiplicand word by multiplier digit multiplication is performed by using dedicated multipliers, which have been described in Subsection 3.3.2.

The second difference comes from the observation that no clearing of the online error is required for any of the intermediate online errors, because there is at most one operation happening at a given time instant within the architecture. Therefore, no online clearing unit is needed, and only a clearing of the accumulated result (C^{B4}) is necessary (achieved by setting the corresponding register to the appropriate value).

Let us now consider the low-precision rounding unit. The internal wordlength γ is equal to 2 for the base-4 representations. Therefore, the truncated online partial MAC result comprises two base-4 digits. The base-4 MSD corresponds to the digit of weight 1, and the base-4 LSD corresponds to the base-4 MSD of the fraction. These two maximally redundant base-4 digits are represented by four signed-binary digits, two in the integer part, two in the fraction part. Consequently, the circuit described in Subsection 3.4.1 can be used for the low-precision rounding of that word.

5.3.3 Rounding and Overflow Correction Unit

In this unit, the result $\widetilde{R}^{SB}$ of the MAC operation is first rounded by employing the IEEE 754 RNE rounding technique to eliminate the extra LSDs as described in Subsection 3.4.1 (page 70). The addition of the IEEE 754 RNE round digit is obtained by an array of limited-carry adders. The overflow correction unit has been described in Subsection 3.4.2 (page 79).

5.4 Computer Simulation Results and Performance Comparison

The correct functionality of the proposed architecture will be confirmed by computer simulations. Then, its speed and area performances will be measured, and subsequently compared to those of existing architectures.

5.4.1 Simulation Results

The wordlengths of both the multiplicand and the multiplier have been set equal to 8 for a subsequent simulation of FPGA hardware implementation by using Max+Plus II and the underlying Synopsys compiler. The reference of the target FPGA is EPF 10K20 RC240-4. The latency of the proposed algorithm is as chosen in the algorithm development (c.f. Section 4.3, page 95).

The test vectors in decimal notation are listed in Table 5.1. It is important to observe

Table 5.1: Test Vectors for Parallel MAC Operation in Decimal

A	B	R_{acc}	$R_{acc,rnd}$
-0.33203125	0.46875000	-0.1556396484375000	-0.15625000
-0.81640625	-0.42578125	0.1919708251953125	0.19140625
0.35156250	0.55468750	0.3869781494140625	0.38671875
-0.91796875	-0.92578125	1.2368164062500000	0.99609375
0.11718750	-0.12109375	1.2226257324218750	0.99609375
-0.15625000	-0.17968750	1.2507019042968750	0.99609375
-0.53906250	0.57031250	0.9432678222656250	0.94140625
-0.55468750	0.19921875	0.8327636718750000	0.83203125
-0.67187500	0.48046875	0.5099487304687500	0.51171875
0.89843750	0.18359375	0.6748962402343750	0.67578125

how some of the full-precision results exceed the absolute value $1-2^{-8}$, which is the limit of the range of representable values in the chosen format. As a consequence, the corresponding expected result values are saturated to the maximum representable value, but the internal full-precision accumulated result can still hold these values without any true internal overflow. The same vectors have been translated by a Matlab program into their equivalent signed-binary representations in Table 5.2 for use in a Matlab emulation of the proposed architecture.

The simulation results are shown in hexadecimal values in Figure 5.2. The correspondence between hexadecimal and signed-binary values is listed in Table 4.3 (page 106). In Figure 5.2, the signed-binary result values correspond exactly to the expected results, thus proving the correct functionality of the proposed architecture. Numerous other simulations

Table 5.2: Test Vectors for Parallel MAC Operation in Signed-Binary

A	B	$R_{acc,rnd}$
$0.0\bar{1}0\bar{1}0\bar{1}0\bar{1}$	0.01111000	$0.0\bar{1}10\bar{1}000$
$0.\bar{1}\bar{1}0\bar{1}000\bar{1}$	$0.0\bar{1}\bar{1}0\bar{1}\bar{1}0\bar{1}$	$0.010\bar{1}0001$
0.01011010	0.10001110	$0.1\bar{1}1001\bar{1}1$
$0.\bar{1}\bar{1}\bar{1}0\bar{1}0\bar{1}\bar{1}$	$0.\bar{1}\bar{1}\bar{1}0\bar{1}\bar{1}0\bar{1}$	0.11111111
0.00011110	$0.000\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$	0.11111111
$0.00\bar{1}0\bar{1}000$	$0.00\bar{1}0\bar{1}\bar{1}\bar{1}0$	0.11111111
$0.\bar{1}000\bar{1}0\bar{1}0$	0.10010010	$0.1111001\bar{1}$
$0.\bar{1}000\bar{1}\bar{1}\bar{1}0$	0.00110011	$0.111\bar{1}1\bar{1}01$
$0.\bar{1}0\bar{1}0\bar{1}\bar{1}00$	0.01111011	0.10000011
0.11100110	0.00101111	$0.110\bar{1}0\bar{1}01$

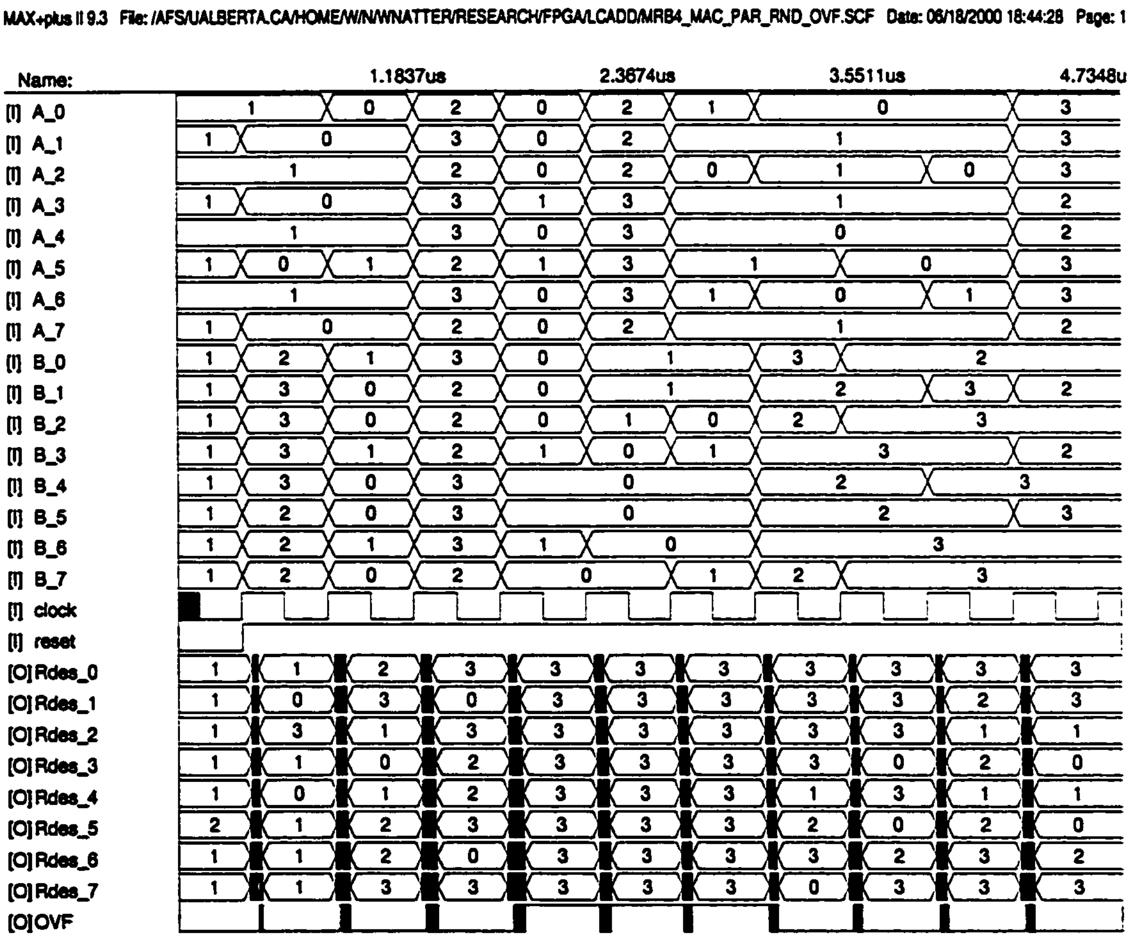


Figure 5.2: Simulation Results for $W_a = W_b = 8$

have been carried out to confirm the correct functionality.

5.4.2 Throughput Parameterization

The various delays corresponding to the units are listed in Table 5.3. The delay of the

Table 5.3: List of Delays of Architectural Units

Unit	Delay		Number of Units
-	Gates	Multiplexers	-
SB to MRB4 Recoder	6	1	1
Limited-Carry Adder	3	-	$W_b' + 2$
SB by MRB4 Digit Multiplier	2	1	1
RNE Rounding	$2W_a - 2W_b' - 2$	1	1
Overflow Correction	-	1	1
D-flip-flop	3	-	1

IEEE 754 RNE round digit generation unit has been restricted to the delay concerning the critical path. Let us denote by τ_g and τ_{mux} the delay through a gate and a multiplexer, respectively. As a result, the delay T of the proposed architecture can be calculated in accordance with

$$T = (2W_a + W_b' + 18)\tau_g + 4\tau_{mux}. \tag{5.35}$$

5.4.3 Area Parameterization

The proposed architecture makes use of a number of elements described in the previous chapters. A summary of the needs is listed in Table 5.4.

Some reductions of the hardware requirements are possible. For example, the addition of the round digit to obtain the IEEE 754 SB RNE-rounded result does not require limited-carry adders to their full extent. As a result of these reductions, the total area A of a corresponding ASIC hardware implementation can be calculated in accordance with

$$\begin{aligned} A =&(8W_aW_b' + 155W_b' + 26W_a - 8W_b + 214)a_g + \\ &(4W_b'W_a + 17W_b' + 4W_a + 2W_b + 22)a_{mux}. \end{aligned} \tag{5.36}$$

Table 5.4: List of Areas of Units Constituent in the MAC Architecture

Unit	Area		Number of Units
-	Gates	Multiplexers	-
SB to MRB4 Recoders	15	1	W_b'
Limited-Carry Adder	4	2	$(W_a+1)W_b'+1$
Carry-Propagate Adder	13	2	$5W_b'$
Low-Precision Rounding	11	-	W_b'
Multiplier	4	2	$(W_a+1)W_b'$
D-flip-flop	6	-	$2W_a+2W_b'+4$
RNE Digit Calculation	$6W_a+6W_b'-5$	2	1
Overflow Correction	$36W_b'-8W_b+36$	$2W_b'+2W_b+1$	1

5.4.4 Comparison

Let us assume $\tau_g = 1\text{ns}$, $\tau_{mux} = 1.5\text{ns}$, $a_g = 25\mu\text{m}^2$, and $a_{mux} = 37.5\mu\text{m}^2$. Moreover, one will assume that W_a and W_b have the same value denoted by W, thereby simplifying the comparison. Then, the throughputs H of the proposed architecture is obtained as

$$H = \frac{W}{T}. \tag{5.37}$$

Next, the efficiency of an architecture is calculated as the ratio of its throughput by its total area. It is extremely important to note that neither the latency of the online operation nor an estimate of the power consumption of a corresponding hardware implementation is taken into account in the above efficiency calculation.

The proposed architecture is compared to the architecture for LSD-first bit-parallel MAC operation and IEEE 754 rounding of signed-binary numbers proposed in (Rao, 1996). The main differences between these architectures are the direction (LSD-first versus MSD-first) of computation of the result, and the full-wordlength overflow detection and correction of the result. A comparative plot of the throughput versus the wordlength is shown in Figure 5.3 for some values of W. A comparative plot of the efficiency versus the wordlength is shown in Figure 5.4 for the same values of W. The curves are not smooth due to slight differences between architectures when the wordlength is odd or even. The speed and efficiency are similar, although the proposed architecture performs worse than the existing LSD-first

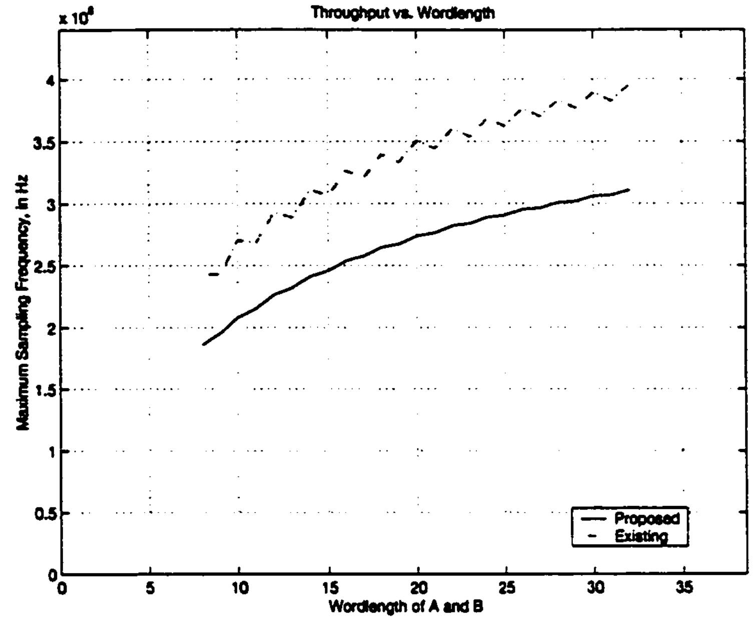


Figure 5.3: Throughput Versus Wordlength of Proposed and Existing Bit-Parallel MAC Architectures

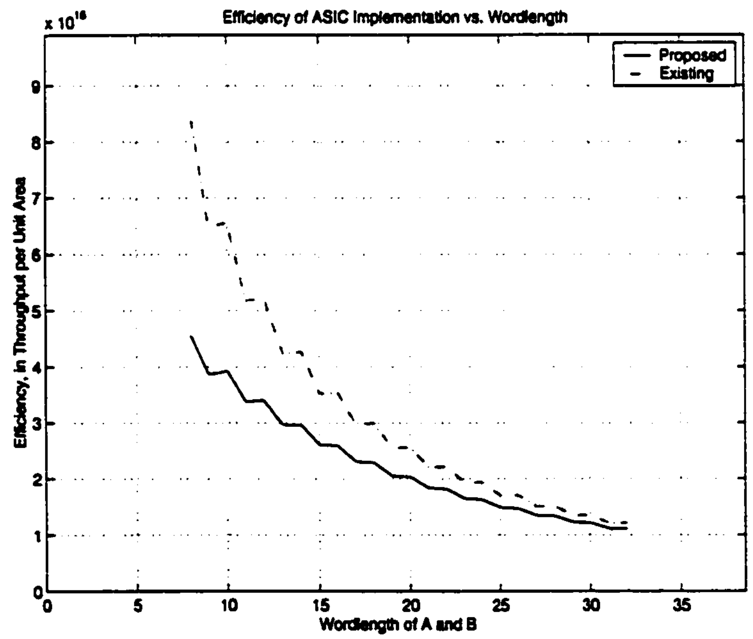


Figure 5.4: Efficiency Versus Wordlength of Proposed and Existing Bit-Parallel MAC Architectures

architecture, despite the numerous efforts to increase the speed of computation and reduce the area of a resulting ASIC hardware implementation. The worse delay calculation is related to the fact that the signed-binary digit encoding does not embed the sign of the digit directly. Therefore, additional calculations are required to determine the sign of the least significant word to be rounded using the IEEE 754 SB RNE rounding technique. Moreover, the area of the proposed architecture is larger because of two factors. The first factor is the low-precision rounding and carry-propagate addition units required by the online processing of data. The second factor is the large overflow correction unit performing an overflow detection and correction over the entire result word, instead of only on the MSD. It must be observed here that it has not been assumed that $|R|$ is smaller than $1 - 2^{-W_b}$. It must also be observed that the computation of the sign bit has not been obtained by look-ahead techniques, thereby making the critical path longer. However, this point does not have a large impact on the results for the wordlengths under consideration.

As a conclusion, the proposed architecture may yield a slightly slower ASIC hardware implementation than existing LSD-first architectures. However, since current practical analog-to-digital and digital-to-analog converters generate and consume digits the MSD first, employing the proposed architecture might prove more attractive and efficient.

5.5 Conclusion

In this chapter, an algorithm has been developed for bit-parallel online purely signed-binary multiply-accumulate operation employing the proposed SB to MRB4 recoding, IEEE 754 SB RNE result rounding, and overflow detection and correction techniques. Corresponding architectures for the above algorithms have been proposed, and the subsequent FPGA hardware implementations have been confirmed functionally correct. The efficiencies (ratio of throughput by area) of the proposed architectures were compared partially to those of existing architectures. The differences observed were attributed to additional low-precision rounding units required for MSD-first operation, and discrepancies between the operations themselves. However, neither the latency of the online operation nor an estimate of the power consumption of a corresponding hardware implementation was taken into account in the efficiency calculations. Therefore, additional work is required to reach a conclusion on the suitability of online operations for high-speed applications.

Chapter 6

Conclusion

6.1 Review of Material Presented

This thesis has been concerned with the design, development, and implementation of digit-serial online signed-digit arithmetic operations for applications in digital signal processing.

Chapter 1 has introduced arithmetic operations in the digital signal processing context, has briefly described the online and the digit-serial arithmetic techniques, and has described the existing advances and their respective extents.

Then, in Chapter 2, fixed-point arithmetic in generalized signed-digit number systems has been described. The digit-serial and online arithmetic techniques have been discussed, the existing digit-serial unfolding algorithm has been simplified, and the unfolding of bit-serial operations into operations featuring a dynamically changing wordlength have been introduced. Finally, redundant binary addition schemes have been characterized by a new offset parameter, and subsequently separated into two equivalence classes. An addition scheme belonging to a given class of equivalence possesses the same addition mechanisms as the other addition schemes of that class, thus allowing the determination of the smallest and fastest addition scheme under varying digit encoding. The 4:2 compressor has been proven to be not only a redundant binary adder, but also one of the most efficient.

Next, Chapter 3 provided the foundation for online generalized signed-digit multiply-accumulate (MAC) operations by developing a recursion-based algorithm featuring operation-independent variables. A different operation can be performed by solely calculating a partial operation update differently. In this way, a generalization of bounds on these variables has been introduced, leading to a procedure for the determination of the various parameters of the MAC operation. These parameters include the number system radix, the redundancy indices, the internal wordlength, and the latency. Also, multiplication and clearing units have been described, and techniques for IEEE 754 SB RNE rounding, low-precision rounding, and overflow handling have been presented along with corresponding architectures.

Chapter 4 has applied the proposed algorithm to the development of an algorithm for digit-serial online purely signed-binary multiplication employing IEEE 754 SB RNE rounding, illustrating the combination of the digit-serial and online arithmetic techniques. A corresponding architecture for subsequent FPGA hardware implementation has been proposed and verified functionally correct. The resulting architecture has been re-pipelined for throughput maximization. The efficiencies (ratio of throughput by area) of the proposed architecture were compared to those of existing architectures.

Finally, in Chapter 5, an algorithm has been developed for bit-parallel online purely signed-binary multiply-accumulate operation employing the proposed SB to MRB4 recoding, IEEE 754 SB RNE result rounding and overflow detection and correction techniques. A corresponding architecture for subsequent FPGA hardware implementation has been proposed and verified functionally correct. The efficiency of the proposed architecture has been compared to those of existing architectures.

In Chapters 4 and 5, the efficiency comparisons have been unfavorable to online operations, mainly because of additional low-precision rounding units required for most-significant-digit-first operation, but also because of functional discrepancies. However, latency and power consumption have not been used as parameters in the efficiency calculations. Therefore, more work will be needed to reach a conclusion regarding the suitability of bit-serial, bit-parallel, and digit-serial online operations for high-speed applications.

6.2 Proposed Areas of Future Work

Addition and multiplication properties of ordinary signed-digit number systems have been presented and extended, but a determination of the speed and area of corresponding efficient hardware implementations has not been carried out. Such a determination would allow one to find which ordinary signed-digit number system is most efficient in terms of area and speed. In particular, it is believed that OSD and USD number systems have another advantage in multiplication over other GSD number systems. Considering an odd (even) GSD number system of digit set D, the cardinality of the set whose elements result from the multiplication of two digits belonging to D should be smallest when the number system is OSD (USD). As a result, the smallest multiplication units would be obtained for OSD and USD number systems.

The dynamically changing wordlength technique has to be defined more rigorously, in order to determine its drawbacks and advantages exactly. In particular, the case of a dynamically changing wordlength taking on values that are all multiples of a small wordlength value may lead to an algorithm similar to that for digit-serial unfolding of bit-serial architectures. It is possible to define such an unfolding algorithm at an algorithmic level instead of an architectural level, which would allow software engineers to benefit from this technique.

In the context of digit-serial online operations, gate-level power estimates for corresponding hardware implementations and latency values should be taken into account in the calculation of the efficiency of these operations. The use of redundant number systems, which are known for their property of stopping the carry propagation, is expected to reduce the switching activity in a hardware implementation, and, consequently, its power consumption. This work would allow one to reach a definitive conclusion regarding the suitability of digit-serial online operations for high-speed low-power applications.

The proposed online algorithm can be used for inner product operation. The author developed an example in Matlab code, but did not disclose it in this thesis. A direct application is the design of finite impulse response filters, since one can see the operation of such a filter to be the inner product of a vector of coefficients with a vector of delayed input sample values. The small latency of such operations seems to be also suitable for their use in infinite impulse response filters. Similarities between the inner product coefficients may be exploited for low-area low-power design, especially if the coefficients are hard-wired. In the latter situation, multiplier-less architectures may be developed.

Similarly, it is believed that the proposed online algorithm can be used for incremental multiplication, whereby both the multiplicand and the multiplier are made available in an online fashion (Irwin, 1977). One should also be able to extend the resulting algorithm to a MAC operation. Such an operation could be used for modulation.

A significant reduction in the delay required for rounding the least significant word (LSW) of a multiplication result can be obtained by first truncating the LSW to only a few of its most significant digits. Then, rounding errors would occur only if the truncated LSW is equal to 0.

6.3 Concluding Remarks

It is most important to remark that the benefits of the online arithmetic technique are obtained when using a bit-serial transmission of the data. If the transmission is digit-serial of bit-parallel, then a hardware implementation of the corresponding LSD-first arithmetic operation can achieve higher sampling frequencies and much lower areas, since no intermediate estimation of the MSD is required. Moreover, since the result is obtained the LSD first, rounding techniques can be applied concurrently to the computation of the result of the arithmetic operation, thus avoiding additional delays.

Bibliography

Avizienis, A. (1961). Signed-digit numbe[r] representations for fast parallel arithmetic, *IRE Transactions on Electronic Computers* EC-**10**(3): 389–400.

Balsara, P., Owens, R. and Irwin, M. (1991). Digit serial multipliers, *Journal of Parallel and Distributed Computing* **11**: 156–162.

Brackert, Jr, R., Ercegovac, M. and Wilson, Jr, A. (1989). Design of an on-line multiply-add module for recursive digital filters, *Proceedings of the 9th Symposium on Computer Arithmetic*, IEEE, Santa Monica, CA, pp. 34–41.

Carter, T. and Robertson, J. (1990). The set theory of arithmetic decomposition, *IEEE Transactions on Computers* **39**(8): 993–1005.

Chow, C. (1980). *A variable precision processor module*, Phd thesis, Departement of Computer Science, University of Illinois, Champaign-Urbana, IL 61801. technical report.

Chow, C. and Robertson, J. (1978). Logical design of a redundant binary adder, *Proceedings of the 4th Symposium on Computer Arithmetic*, IEEE, Santa Monica, CA, pp. 109–115.

Dadda, L. (1976). On parallel digital multipliers, *Alta Frequenza* **45**: 574–580.

Duprat, J., Herreros, Y. and Muller, J. (1989). Some results about on-line computation of functions, *Proceedings of the 9th Symposium on Computer Arithmetic*, IEEE, Santa Monica, CA, pp. 112–118.

Ercegovac, M. (1984). On-line arithmetic: an overview, *SPIE, Real-Time Signal Processing VII*, Vol. 495, IEEE, San Diego, pp. 86–93.

Ercegovac, M. and Lang, T. (1989). On-line arithmetic for DSP applications, *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, Urbana-Champaign, IL, USA, pp. 365–368.

Ercegovac, M. and Lang, T. (1990). Fast multiplication without carry-propagate addition, *IEEE Transactions on Computers* **39**(11): 1385–1390.

Fernando, J. and Ercegovac, M. (1992). On-line arithmetic modules for recursive digital filters, *Record of the 26th Asilomar Conference on Signals, Systems, and Computers*, Vol. 2, pp. 681–685.

Gorji-Sinaki, A. and Ercegovac, M. (1981). Design of a digit-slice on-line arithmetic unit, *Proceedings of the 5th Symposium on Computer Arithmetic*, IEEE, University of Michigan, Ann Arbor, MI, pp. 72–80.

Goto, G., Inoue, A., Ohe, R., Kashiwakura, S., Mitarai, S., Tsuru, T. and Izawa, T. (1997). 4.1-ns compact 54×54-b multiplier utilizing sign-select booth encoders, *IEEE Journal of Solid-State Circuits* **32**(11): 1676–1681.

Guyot, A., Herreros, Y. and Muller, J. (1989). JANUS, an on-line multiplier/divider for manipulating large numbers, *Proceedings of the 9th Symposium on Computer Arithmetic*, IEEE, Santa Monica, CA, pp. 106–111.

Guyot, A. and Kusumaputri, Y. (1991). OCAPI: a prototype for high precision arithmetic, *in* A. Halaas and P. B. D. (Eds) (eds), *Proceedings of the International Conference on Very Large Scale Integration (IFIP) 1991*, TC 10/WG 10.5, Elsevier Science Publishers B. V. (North-Holland), pp. 11–18.

Hagihara, Y., Inui, S., Yoshikawa, A., Nakazato, S., Iriki, S., Ikeda, R., Shibue, Y., Inaba, T., Kagamihara, M. and Yamashina, M. (1998). 2.7ns 0.25μm CMOS 54 $\times$ 54b multiplier, *Proceedings of the 1998 IEEE 45th International Solid-State Circuits Conference*, pp. 296–297.

Hwang, K. (1979). *Computer arithmetic - principles, architecture and design*, John Wiley & Sons.

Irwin, M. (1977). *An arithmetic unit for on-line computation*, Ph.D. thesis, Department of Computer Science, University of Illinois, Champaign-Urbana, IL 61801. Technical Report UIUCDCS-R-77-873.

Irwin, M. and Owens, R. (1987). Digit-pipelined arithmetic as illustrated by the paste-up system: a tutorial, *Computer* **20**(4): 61–73.

Irwin, M. and Owens, R. (1988). A comparison of two digit serial VLSI adders, *Proceedings of the IEEE Conference on Computer Design 1988*, pp. 227–229.

Kanie, Y., Kubota, Y., Toyoyama, S., Iwase, Y. and Tsuchimoto, S. (1994). 4-2 compressor with complementary pass-transistor logic, *IEICE Transactions on Electronics* **E77-C**(4): 647–649.

Kornerup, P. (1994). Digit-set conversions: generalizations and applications, *IEEE Transactions on Computers* **43**(5): 622–629.

Lapointe, M., Huynh, H. and Fortier, P. (1993). Systematic design of pipelined recursive filters, *IEEE Transactions on Computers* **42**(4): 413–426.

Larsson, P. and Nicol-Chris, J. (1996). Transition reduction in carry-save adder trees, *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, pp. 85–88.

Law, C., Rofail, S. and Yeo, K. (1999). Low-power circuit implementation for partial-product addition using pass-transistor logic, *IEE Proc.-Circuits Devices Syst.*, Vol. 146, IEE, pp. 124–129. No. 3.

Lin, H. and Sips, H. (1990). On-line CORDIC algorithms, *IEEE Transactions on Computers* **39**(8): 1038–1052.

McNally, O., McCanny, J. and Woods, R. (1990). Optimised bit-level architectures for IIR filtering, *Proceedings of the IEEE Conference on Computer Design 1990*, pp. 302–306.

McQuillan, S. and McCanny, J. (1995). A systematic methodology for the design of high performance recursive digital filters, *IEEE Transactions on Computers* **44**(8): 971–982.

Muller, J. (1994). Some characteristics of functions computable in on-line arithmetic, *IEEE Transactions on Computers* **43**(6): 752–755.

Natter, W. and Nowrouzian, B. (1999). Digit-serial digit-online addition, *Proceedings of the Canadian Conference on Electrical and Computer Engineering 1999 (CCECE'99)*, pp. 583–588.

Natter, W. and Nowrouzian, B. (2000a). A novel algorithm for signed-digit online multiply-accumulate operation and its purely signed-digit hardware implementation, *Proceedings of the International Symposium on Circuits and Systems (ISCAS) 2000.*

Natter, W. and Nowrouzian, B. (2000b). Signed-Digit Online MAC Operation and its FPGA Hardware Implementation, *ISIAC 2000.*

Owens, R. (1980). *Digit-online algorithms for pipeline architectures*, Phd thesis, Department of Computer Science, Pennsylvania State University, University Park, PA 16802. Technical Report CS-80-21.

Owens, R. (1981). Compound algorithms for digit online arithmetic, *Proceedings of the* 5th *Symposium on Computer Arithmetic*, IEEE, University of Michigan, Ann Arbor, MI, pp. 64–71.

Owens, R. (1983). Techniques to reduce the inherent limitations of fully digit online arithmetic, *IEEE Transactions on Computers* **C-32**(4): 406–411.

Parhami, B. (1988). Carry-free addition of recoded binary signed-digit numbers, *IEEE Transactions on Computers* **37**(11): 1470–1476.

Parhami, B. (1990). Generalized signed-digit number systems: a unifying framework for redundant number representations, *IEEE Transactions on Computers* **39**(1): 89–98.

Parhi, K. (1991). A systematic approach for design of digit-serial signal processing architectures, *IEEE Transactions on Circuits and Systems* **38**(4): 358–375.

Perlee, C. and Casasent, D. (1989). Optical systems for digit-serial computation, *Applied Optics* **28**(3): 611–626.

Phatak, D. and Koren, I. (1994). Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains, *IEEE Transactions on Computers* **43**(8): 880–891.

Pillai, R., Al-Khalili, D. and Al-Khalili, A. (1996). Energy delay analysis of partial product reduction methods for parallel multiplier implementation, *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, pp. 201–204.

Privat, G. (1990). A novel class of serial-parallel redundant signed-digit multipliers, *1990 IEEE International Symposium on Circuits and Systems*, Vol. 3, IEEE, New Orleans, LA, pp. 2116–2119.

Rao, V. (1996). *Redundant number multiply-accumulate-modularized digital filters*, M.sc. thesis, The University of Calgary.

Rao, V. and Nowrouzian, B. (1997). A novel high-speed bit-parallel multiply-accumulate arithmetic architecture employing mixed SB/TC number arithmetic, *Canadian Journal of Electrical and Computer Engineering* **22**(4): 169–175.

Rao, V. and Nowrouzian, B. (1999). 5-digit overlapped-scanning technique for the modified radix-4 recoding of signed-binary numbers, *IEE Proc.-Circuits Devices Syst.*, Vol. 146, IEE, pp. 1–4. No. 6.

Santoro, M., Bewick, G. and Horowitz, M. (1989). Rounding algorithms for IEEE multipliers, *Proceedings of the* 9th *Symposium on Computer Arithmetic*, IEEE, Santa Monica, CA, pp. 176–183.

Satyanarayana, J. and Nowrouzian, B. (1996). Design and FPGA implementation of digit-serial modified booth multipliers, *Journal of Circuits, Systems and Computers* **6**(5): 485–501.

Shim, D. and Kim, W. (1997). Design of 16 × 16 wave pipelined multiplier using fan-in equalization technique, *Proceedings of the 1997 40th Midwest Symposium on Circuits and Systems*, Vol. 1, Sacramento, CA, USA, pp. 336–339.

Sips, H. and Lin, H. (1990). A new model for on-line arithmetic with an application to the reciprocal calculation, *Journal of Parallel and Distributed Computing* **8**: 218–230.

Srinivas, H. and Parhi, K. (1983). Computer arithmetic architectures with redundant number systems, *IEEE Transactions on Computers* **C-32**(4): 406–411.

Thornton, M. (1997). Signed binary addition circuitry with inherent even parity outputs, *IEEE Transactions on Computers* **46**(7): 811-816.

Timmermann, D. and Hosticka, B. (1993). Overflow effects in redundant binary number systems, *Electronics Letters* **29**(5): 440–441.

Trivedi, K. and Ercegovac, M. (1977). On-line algorithms for division and multiplication, *IEEE Transactions on Computers* **C-26**(7): 681-687.

Wallace, C. (1964). A suggestion for a fast multiplier, *IEEE Transactions on Computers* **EC-14**: 14–17.

Woods, R., McNally, O. and McQuillan, S. (1993). Saturation circuitry for redundant number based IIR filters, *Electronics Letters* **29**(5): 440–441.

Appendix A

Systematic Enumeration of Redundant Binary Addition Schemes

This appendix provides the proofs of the various theorems relating to redundant binary addition schemes stated in this thesis. The goal of these theorems is to introduce a framework to study similar addition schemes which can be mapped to the same hardware implementations. Section A.1 determines the cardinalities of the sets of the variables involved in a redundant binary addition scheme. Then, Section A.2 determines the values of the elements of these sets in a systematic manner. Next, Section A.3 provides the proofs of the theorem characterizing the proposed addition schemes. Finally, Section A.4 provides the proofs of the theorem characterizing the proposed addition schemes.

A.1 Determination of Set Cardinalities

This section is concerned with a determination of all the possibilities for the cardinalities of the digit sets S_{c_1}, S_α, S_{c_2}, and S_β. The cardinalities can be determined by using the following lemma.

Lemma 30 *Let $S_a = \{a_1, \ldots, a_m\}$ and $S_b = \{b_1, \ldots, b_n\}$ represents sets such that*

$$\forall i \in [1, m-1],\ a_i < a_{i+1};\ \text{and}\ \forall i \in [1, n-1],\ b_i < b_{i+1}. \tag{A.1}$$

Then,

$$m + n - 1 \leqslant \left| S_a \dot{+} S_b \right| \leqslant m \times n. \tag{A.2}$$

Proof. The proof consists of two parts.

- Using the relationships between a_i's and b_j's, one obtains

$$a_1 + b_1 < \ldots < a_1 + b_n < a_2 + b_n < \ldots < a_m + b_n. \tag{A.3}$$

  The number of different combinations is $m + n - 1$, therefore $S_a \dot{+} S_b$ contains at least $m + n - 1$ elements.

- Each of the m elements of S_a can be added to at most n elements of S_b. Therefore, the maximum number of elements in $S_a \dot{+} S_b$ is $m \times n$.

These two points establish the proof. ■

In order for Eqn. 2.63 to be satisfied, one must impose

$$2 \leqslant |S_\alpha|,\ 2 \leqslant |S_{c_1}|,\ \text{and}\ (|S_\alpha|, |S_{c_1}|) \neq (2,2). \tag{A.4}$$

Otherwise, by using Lemma 30, one can show that

$$\left|S_{c_1} \dot{+} S_\alpha\right| \leqslant 4 < \left|S_a \dot{+} S_b\right|, \tag{A.5}$$

contradicting Eqn. 2.63. Similarly, Eqn. 2.65 imposes that

$$|S_\beta| \leqslant 2,\ \text{and}\ |S_{c_2}| \leqslant 2. \tag{A.6}$$

Otherwise, Lemma 30 imposes the constraint

$$3 < \left|S_\beta \dot{+} 2S_{c_2}\right|, \tag{A.7}$$

contradicting of Eqn. 2.65, because $|S_s| = 3$.

Together, Eqn. A.4 and Lemma 30 impose the constraint

$$3 \leq \left|S_\alpha \dot{+} 2S_{c_1}\right|. \tag{A.8}$$

If $|S_\beta| = 1$, then

$$\left|S_\beta \dot{+} S_{c_2}\right| = |S_{c_2}|\ \text{and}\ |S_{c_2}| \leq 2, \tag{A.9}$$

contradicting Eqn. 2.64. Similarly, if $|S_{c_2}| = 1$, then

$$\left|S_\beta \dot{+} S_{c_2}\right| = |S_\beta|\ \text{and}\ |S_\beta| \leq 2, \tag{A.10}$$

also contradicting Eqn. 2.64. Therefore, one can conclude that

$$|S_\beta| = 2\ \text{and}\ |S_{c_2}| = 2. \tag{A.11}$$

Finally, let us prove that

$$(|S_{c_1}|, |S_\alpha|) = (2,3)\ \text{or}\ (3,2). \tag{A.12}$$

If Eqn. A.12 is not satisfied, then Eqn. A.4 and Lemma 30 impose the constraint

$$4 < \left|S_\alpha \dot{+} 2S_{c_1}\right|, \tag{A.13}$$

whereas Eqn. A.11 fixes

$$\left|S_{c_2} \dot{+} S_\beta\right| \leq 4, \tag{A.14}$$

thus contradicting Eqn. 2.64. If Eqn. A.12 is satisfied, then Eqns. 2.63–2.65 are satisfied.

A summary of the possible cases of set cardinalities is given in Table 2.1, Chapter 2.

A.2 Systematic Determination of the Sets S_{c_1}, S_α, S_{c_2}, and S_β

The values of the elements of S_β and S_{c_2} are determined first, followed by the determination of the values of the elements of S_α and S_{c_1}.

In accordance with Table 2.1, let us denote

$$S_{c_2} = \{c_{21}, c_{22}\} \text{ and } S_\beta = \{\beta_1, \beta_2\}, \tag{A.15}$$

where one assumes $c_{21} < c_{22}$, and where $\beta_1 < \beta_2$, which can be obtained by swapping the elements. Let us further denote $\Delta c_2 = c_{22} - c_{21} > 0$ and $\Delta\beta = \beta_2 - \beta_1 > 0$.

The following must hold for the migration from the right-hand side of Eqn. 2.61 to its left-hand side to be possible:

$$S_\beta \dot{+} 2S_{c_2} \subset 4S, \tag{A.16}$$

implying

$$\{\beta_1 + 2c_{21}, \beta_1 + 2c_{22}, \beta_2 + 2c_{21}, \beta_2 + 2c_{22}\} \subset \{4\gamma - 4, 4\gamma, 4\gamma + 4\}. \tag{A.17}$$

Since

$$\beta_1 + 2c_{21} < \beta_1 + 2c_{22} < \beta_2 + 2c_{22}, \tag{A.18}$$

and since

$$\beta_1 + 2c_{21} < \beta_2 + 2c_{21} < \beta_2 + 2c_{22}, \tag{A.19}$$

one can identify $\beta_1 + 2c_{21}$ and $\beta_2 + 2c_{22}$ as the minimum and maximum element values in $S_\beta \dot{+} 2S_{c_2}$. The value $\beta_1 + 2c_{22}$ must be equal to $\beta_2 + 2c_{21}$, otherwise $S_\beta \dot{+} 2S_{c_2} \not\subset 4S_s$. Therefore,

$$\beta_1 + 2c_{21} = 4\gamma - 4 \tag{A.20}$$
$$\beta_1 + 2c_{22} = 4\gamma \tag{A.21}$$
$$\beta_2 + 2c_{21} = 4\gamma \tag{A.22}$$
$$\beta_2 + 2c_{22} = 4\gamma + 4. \tag{A.23}$$

By simple manipulations, one can find

$$\Delta\beta = 4 \text{ and } \Delta c_2 = 2, \tag{A.24}$$

and

$$\beta_1 = 4\gamma - 2c_{21} - 4 \tag{A.25}$$
$$\beta_2 = 4\gamma - 2c_{21}. \tag{A.26}$$

Therefore, one concludes that

$$S_{c_2} = \{c_{21}, c_{21} + 2\} \tag{A.27}$$

and

$$S_\beta = \{4\gamma - 4 - 2c_{21}, 4\gamma - 2c_{21}\}. \tag{A.28}$$

The digit sets S_{c_2} and S_β are determined as functions of the parameter c_{21}. In the following, the cardinalities of the sets S_α and S_{c_1} are taken into account. Two situations can occur, namely $|S_{c_1}| = 3$ and $|S_\alpha| = 2$, or $|S_{c_1}| = 2$ and $|S_\alpha| = 3$.

A.2.1 Case 1: $|S_{c_1}| = 3$ and $|S_\alpha| = 2$

Similar to the determination of S_{c_2} and S_β, let us denote the elements of S_{c_1} and S_α by

$$S_{c_1} = \{c_{11}, c_{12}, c_{13}\} \text{ and } S_\alpha = \{\alpha_1, \alpha_2\}, \tag{A.29}$$

where $c_{11} < c_{12} < c_{13}$ and $\alpha_1 < \alpha_2$. Also, let us define

$$\begin{aligned} \Delta\alpha &= \alpha_2 - \alpha_1 > 0 \\ \Delta c_{12} &= c_{12} - c_{11} > 0 \\ \Delta c_{13} &= c_{13} - c_{11} > 0. \end{aligned} \tag{A.30}$$

By combining these notations successively with Eqns. 2.64 and 2.63, one can exhaustively determine the values of the elements of S_{c_1} and S_α.

Eqn. 2.64 is satisfied when

$$\{\alpha_1 + 2c_{11}, \dots, \alpha_2 + 2c_{13}\} \subset S_{c_2} \dot{+} S_\beta, \tag{A.31}$$

where

$$S_{c_2} \dot{+} S_\beta = \{4\gamma - 4 - c_{21}, 4\gamma - 2 - c_{21}, 4\gamma - c_{21}, 4\gamma + 2 - c_{21}\}. \tag{A.32}$$

By identification,

$$\alpha_1 + 2c_{11} = 4\gamma - c_{21} - 4 \tag{A.33}$$

$$\alpha_2 + 2c_{13} = 4\gamma - c_{21} + 2. \tag{A.34}$$

Moreover, one can observe that

$$\{\alpha_1 + 2c_{12}, \alpha_1 + 2c_{13}, \alpha_2 + 2c_{11}, \alpha_2 + 2c_{12}\} \subset \{4\gamma - 2 - c_{21}, 4\gamma - c_{21}\}. \tag{A.35}$$

Since $\alpha_1 + 2c_{12} < \alpha_1 + 2c_{13}$ and $\alpha_2 + 2c_{11} < \alpha_2 + 2c_{12}$,

$$\alpha_1 + 2c_{12} = 4\gamma - c_{21} - 2 \tag{A.36}$$

$$\alpha_1 + 2c_{13} = 4\gamma - c_{21} \tag{A.37}$$

$$\alpha_2 + 2c_{11} = 4\gamma - c_{21} - 2 \tag{A.38}$$

$$\alpha_2 + 2c_{12} = 4\gamma - c_{21} \tag{A.39}$$

is the only set of equations allowing Eqn. A.35 to be satisfied. Therefore, one can calculate

$$\Delta\alpha = 2,\ \Delta c_{12} = 1, \text{ and } \Delta c_{13} = 2 \tag{A.40}$$

The sets are now given by

$$S_{c_1} = \{c_{11}, c_{11} + 1, c_{11} + 2\} \tag{A.41}$$

$$S_\alpha = \{4\gamma - c_{21} - 2c_{11} - 4, 4\gamma - c_{21} - 2c_{11} - 2\}. \tag{A.42}$$

Now, one can apply the constraint given by Eqn. 2.63 to obtain

$$\left\{\begin{array}{l} 2\gamma - 2, \\ 2\gamma - 1, \\ 2\gamma, \\ 2\gamma + 1, \\ 2\gamma + 2 \end{array}\right\} \subset \left\{\begin{array}{l} 4\gamma - c_{21} - c_{11} - 4, \\ 4\gamma - c_{21} - c_{11} - 3, \\ 4\gamma - c_{21} - c_{11} - 2, \\ 4\gamma - c_{21} - c_{11} - 1, \\ 4\gamma - c_{21} - c_{11} \end{array}\right\}. \tag{A.43}$$

Therefore,

$$c_{21} + c_{11} = 2\gamma - 2. \tag{A.44}$$

If c_{11} and c_{21} are considered as two dimensions of a three-dimensional space, and if γ is considered as a parameter, then this is the equation of a plane, and one can find a real Δ such that

$$\begin{cases} c_{11} = \gamma + \Delta - 1 \\ c_{21} = \gamma - \Delta - 1 \end{cases} \tag{A.45}$$

As a consequence, one can write

$$S_\alpha = \{\gamma - \Delta - 1, \gamma - \Delta + 1\} \text{ and } S_{c_1} = \{\gamma + \Delta - 1, \gamma + \Delta, \gamma + \Delta + 1\}, \tag{A.46}$$

as shown in Case 1 of Table 2.2 (page 40).

A.2.2 Case 2: $|S_{c_1}| = 2$ and $|S_\alpha| = 3$

Once again, let us define

$$S_{c_1} = \{c_{11}, c_{12}\} \tag{A.47}$$

$$S_\alpha = \{\alpha_1, \alpha_2, \alpha_3\}, \tag{A.48}$$

where $c_{11} < c_{12}$ and $\alpha_1 < \alpha_2 < \alpha_3$. Moreover, let us define

$$\Delta\alpha_2 = \alpha_2 - \alpha_1 > 0 \tag{A.49}$$

$$\Delta\alpha_3 = \alpha_3 - \alpha_1 > 0 \tag{A.50}$$

$$\Delta c_1 = c_{12} - c_{11} > 0. \tag{A.51}$$

By exchanging the roles of α and c_1 in the above proof, one can obtain

$$\alpha_1 + 2c_{11} = 4\gamma - c_{21} - 4 \tag{A.52}$$

$$\alpha_1 + 2c_{12} = 4\gamma - c_{21} - 2 \tag{A.53}$$

$$\alpha_2 + 2c_{11} = 4\gamma - c_{21} - 2 \tag{A.54}$$

$$\alpha_2 + 2c_{12} = 4\gamma - c_{21} \tag{A.55}$$

$$\alpha_3 + 2c_{11} = 4\gamma - c_{21} \tag{A.56}$$

$$\alpha_3 + 2c_{12} = 4\gamma - c_{21} + 2, \tag{A.57}$$

which allows Eqn. 2.64 to be satisfied. By some manipulations, one obtains

$$\Delta\alpha_2 = 2,\ \Delta\alpha_3 = 4, \text{ and } \Delta c_1 = 1. \tag{A.58}$$

The sets can now be written as

$$S_{c_1} = \{c_{11}, c_{11} + 1\} \text{ and } S_\alpha = \{4\gamma - c_{21} - 2c_{11} - 4, 4\gamma - c_{21} - 2c_{11} - 2\}. \tag{A.59}$$

Similar to the previous demonstration, one can apply the constraint given by Eqn. 2.63 to obtain

$$S_a \dot{+} S_b \subset S_\alpha \dot{+} S_{c_1}. \tag{A.60}$$

Therefore,

$$\left\{\begin{array}{l} 2\gamma-2, \\ 2\gamma-1, \\ 2\gamma, \\ 2\gamma+1, \\ 2\gamma+2 \end{array}\right\} \subset \left\{\begin{array}{l} 4\gamma-c_{21}-c_{11}-4, \\ 4\gamma-c_{21}-c_{11}-3, \\ 4\gamma-c_{21}-c_{11}-2, \\ 4\gamma-c_{21}-c_{11}-1, \\ 4\gamma-c_{21}-c_{11}, \\ 4\gamma-c_{21}-c_{11}+1 \end{array}\right\}. \tag{A.61}$$

This problem has two solutions. Either

$$c_{21}+c_{11}=2\gamma-2, \tag{A.62}$$

or

$$c_{21}+c_{11}=2\gamma-1. \tag{A.63}$$

Eqn. A.62 is equivalent to choosing $\Delta \in \mathcal{R}$ such that

$$\left\{\begin{array}{ll} c_{11} & =\gamma+\Delta-1 \\ c_{21} & =\gamma-\Delta-1 \end{array}\right., \tag{A.64}$$

and Eqn. A.63 is equivalent to choosing $\Delta \in \mathcal{R}$ such that

$$\left\{\begin{array}{ll} c_{11} & =\gamma+\Delta \\ c_{21} & =\gamma-\Delta-1 \end{array}\right.. \tag{A.65}$$

In the case of Eqn. A.62, one can write

$$S_\alpha=\{\gamma-\Delta-1,\gamma-\Delta+1,\gamma-\Delta+3\} \text{ and } S_{c_1}=\{\gamma+\Delta-1,\gamma+\Delta\}, \tag{A.66}$$

as shown in Case 2 of Table 2.2 (page 40). In the case of Eqn. A.62, one can write

$$S_\alpha=\{\gamma-\Delta-3,\gamma-\Delta-1,\gamma-\Delta+1\} \text{ and } S_{c_1}=\{\gamma+\Delta,\gamma+\Delta+1\}, \tag{A.67}$$

as shown in Case 3 of Table 2.2 (page 40). This completes the enumeration of redundant binary addition schemes, whose parameters are $\gamma \in \{-1,0,1\}$ and $\Delta \in \mathbb{R}$.

A.3 Characterization of Redundant Binary Number Addition Schemes

This section is concerned with a proof of Theorem 15, characterizing redundant binary addition schemes, and with the proof of for the equivalence of these schemes.

Let us recall that Theorem 15 states that all redundant binary addition schemes are characterized by γ, which determines the digit-set of the representation, and by Δ, which determines the sets of the weight and transfer digits.

Proof. Most of the proof has been given previously. The only sets S_β and S_{c_2} that can be defined for a given Δ are

$$S_{c_2}=\{c_{21},c_{21}+2\} \tag{A.68}$$

$$S_\beta=\{4\gamma-4-2c_{21},4\gamma-2c_{21}\} \tag{A.69}$$

where $c_{21} = \gamma - \Delta - 1$, i.e.

$$S_{c_2} = \{\gamma - \Delta - 1, \gamma - \Delta + 1\} \tag{A.70}$$

$$S_\beta = \{2\gamma + 2\Delta - 2, 2\gamma + 2\Delta + 2\} \tag{A.71}$$

because the number c_{21} was always given the same value in the previous demonstrations.

- If $|S_\alpha| = 2$ and $|S_{c_1}| = 3$, then one can only find

$$S_{c_1} = \{c_{11}, c_{11} + 1, c_{11} + 2\} \tag{A.72}$$

$$S_\alpha = \{4\gamma - c_{21} - 2c_{11} - 4, 4\gamma - c_{21} - 2c_{11} - 2\} \tag{A.73}$$

  where $c_{21} = \gamma - \Delta - 1$, and where $c_{11} = \gamma + \Delta - 1$. Hence,

$$S_{c_1} = S \dot{+} \{\Delta\} \tag{A.74}$$

$$S_\alpha = \{\gamma - 1, \gamma + 1\} \dot{+} \{-\Delta\} \tag{A.75}$$

- If $|S_\alpha| = 3$ and $|S_{c_1}| = 2$, then one can only find

$$S_{c_1} = \{c_{11}, c_{11} + 1\} \tag{A.76}$$

$$S_\alpha = \{4\gamma - c_{21} - 2c_{11} - 4, 4\gamma - c_{21} - 2c_{11} - 2\} \tag{A.77}$$

  where $c_{21} = \gamma - \Delta - 1$, and where either $c_{11} = \gamma + \Delta - 1$ or $c_{11} = \gamma + \Delta$. Hence, one either has

$$S_{c_1} = \{\gamma - 1, \gamma\} \dot{+} \{\Delta\} \tag{A.78}$$

$$S_\alpha = \{\gamma - 1, \gamma + 1, \gamma + 3\} \dot{+} \{-\Delta\} \tag{A.79}$$

  or has

$$S_{c_1} = \{\gamma, \gamma + 1\} \dot{+} \{\Delta\} \tag{A.80}$$

$$S_\alpha = \{\gamma - 3, \gamma - 1, \gamma + 1\} \dot{+} \{-\Delta\} \tag{A.81}$$

This completes the proof. ∎

A.4 Equivalence of Redundant Binary Number Addition Schemes

This section is concerned with a proof of Theorem 18, stating that certain addition schemes are equivalent, i.e. two addition schemes may result in the same hardware implementation.

First, one introduces functions that transform an addition scheme into another. Second, an equivalence relationship for addition schemes is defined. Finally, Theorem 18 is proven by demonstrating that there exist only two classes of equivalence for the above relationship.

Let us introduce addition scheme transformations.

Lemma 31 *Consider the redundant binary number addition schemes A with parameters (γ, Δ) and A' with parameters (γ', Δ'). If these addition schemes satisfy*

$$\forall x \in \{c_1, \alpha\} \, |S_x| = |S_x| \,, \tag{A.82}$$

then there exist an ϵ belonging to $\{\bar{1},1\}$ and transformations

$$\begin{aligned}
T_{c_1} &: x \in S_{c_1} \mapsto (\epsilon(x-(\gamma+\Delta)) + \gamma' + \Delta') \in S'_{c_1}\\
T_{c_2} &: x \in S_{c_2} \mapsto (\epsilon(x-(\gamma-\Delta)) + \gamma' - \Delta') \in S'_{c_2}\\
T_{\alpha} &: x \in S_{\alpha} \mapsto (\epsilon(x-(\gamma-\Delta)) + \gamma' - \Delta') \in S'_{\alpha}\\
T_{\beta} &: x \in S_{\beta} \mapsto (\epsilon(x-2(\gamma+\Delta)) + 2(\gamma' + \Delta')) \in S'_{\beta}
\end{aligned}$$

and

$$\forall d \in \{a,b,s\},\ T_d : x \in S_d \mapsto (\epsilon(x-\gamma)+\gamma') \in S'_d$$

such that $T_{c_1}(S_{c_1}) = S'_{c_1}$, $T_{c_2}(S_{c_2}) = S'_{c_2}$, $T_\alpha(S_\alpha) = S'_\alpha$, $T_\beta(S_\beta) = S'_\beta$, $T_a(S_a) = S'_a$, $T_b(S_b) = S'_b$, and $T_s(S_s) = S'_s$. Moreover, one can define the inverse transformations

$$\begin{aligned}
T_{c_1}^{-1} &: x \in S_{c_1} \mapsto (\epsilon(x-(\gamma'+\Delta')) + \gamma + \Delta) \in S'_{c_1}\\
T_{c_2}^{-1} &: x \in S_{c_2} \mapsto (\epsilon(x-(\gamma'-\Delta')) + \gamma - \Delta) \in S'_{c_2}\\
T_{\alpha}^{-1} &: x \in S_{\alpha} \mapsto (\epsilon(x-(\gamma'-\Delta')) + \gamma - \Delta) \in S'_{\alpha}\\
T_{\beta}^{-1} &: x \in S_{\beta} \mapsto (\epsilon(x-2(\gamma'+\Delta')) + 2(\gamma + \Delta)) \in S'_{\beta}
\end{aligned}$$

and

$$\forall d \in \{a,b,s\},\ T_d^{-1} : x \in S_d \mapsto (\epsilon(x-\gamma')+\gamma) \in S'_d$$

such that $T_{c_1}^{-1}(S'_{c_1}) = S_{c_1}$, $T_{c_2}^{-1}(S'_{c_2}) = S_{c_2}$, $T_\alpha^{-1}(S'_\alpha) = S_\alpha$, $T_\beta^{-1}(S'_\beta) = S_\beta$, $T_a^{-1}(S'_a) = S_a$, $T_b^{-1}(S'_b) = S_b$, and $T_s^{-1}(S'_s) = S_s$.

The proof of this lemma relies on the following two axioms.

Axiom 32 *If one defines $S' = S \dot{+} \{n\}$, then $S = S' \dot{+} \{n'\}$, where $n' = -n$.*

Axiom 33 *If one defines $S' = nS$, then $S = n'S'$, where $n' = \frac{1}{n}$.*

Proof. The existence of the transformation from the digit set $S = \{\gamma-1, \gamma, \gamma+1\}$ to the digit set $S' = \{\gamma'-1, \gamma', \gamma'+1\}$ is obtained immediately. In fact, if one subtracts γ from S, then one obtains $S_0 = \{\bar{1},0,1\}$ (Axiom 32). Then, from Axiom 33,

$$\forall \epsilon \in \{\bar{1},1\},\ \epsilon S_0 = S_0 \tag{A.83}$$

Hence, $\epsilon(S \dot{+} \{-\gamma\}) \dot{+} \{\gamma'\} = S'$ (Axiom 32). By inverting the roles of S and S', one obtains the inverse transformation.

Let us now consider each digit-set cardinality case separately to prove the existence of the transformations for c_1, α, c_2, and β and of their respective inverses. Axioms 32 and 33 will be used throughout this demonstration, but will not be cited.

- Situation $|S_{c_1}| = 3$, $|S_\alpha| = 2$. Since there is only one addition scheme with such parameters, one can write

$$\begin{aligned}
S_{c_1} &= \{\bar{1},0,1\} \dot{+} \{\gamma+\Delta\}\\
S_{\alpha} &= \{\bar{1},1\} \dot{+} \{\gamma-\Delta\}\\
S_{c_2} &= \{\bar{1},1\} \dot{+} \{\gamma-\Delta\}\\
S_{\beta} &= 2\{\bar{1},1\} \dot{+} 2\{\gamma+\Delta\}
\end{aligned} \tag{A.84}$$

Hence, for all ϵ in $\{\bar{1}, 1\}$,

$$\begin{aligned} \epsilon(S_{c_1} \dot{+} \{-\gamma - \Delta\}) &= \{\bar{1}, 0, 1\} \\ \epsilon(S_{\alpha} \dot{+} \{-\gamma + \Delta\}) &= \{\bar{1}, 1\} \\ \epsilon(S_{c_2} \dot{+} \{-\gamma + \Delta\}) &= \{\bar{1}, 1\} \\ \epsilon(S_{\beta} \dot{+} 2\{-\gamma - \Delta\}) &= 2\{\bar{1}, 1\} \end{aligned} \tag{A.85}$$

By adding $\gamma' + \Delta'$ or $\gamma' - \Delta'$ appropriately, one can identify the results

$$\begin{aligned} (\gamma' + \Delta') \dot{+} \epsilon(S_{c_1} \dot{+} \{-\gamma - \Delta\}) &= S'_{c_1} \\ (\gamma' - \Delta') \dot{+} \epsilon(S_{\alpha} \dot{+} \{-\gamma + \Delta\}) &= S'_{\alpha} \\ (\gamma' - \Delta') \dot{+} \epsilon(S_{c_2} \dot{+} \{-\gamma + \Delta\}) &= S'_{c_2} \\ (\gamma' + \Delta') \dot{+} \epsilon(S_{\beta} \dot{+} 2\{-\gamma - \Delta\}) &= S'_{\beta} \end{aligned} \tag{A.86}$$

The last equality was obtained by identification of the results. Thus, one found a transformation from A to A', and the reverse transformation exists and can be found by exchanging the roles of A and A'.

- Situation $|S_{c_1}| = 2$, $|S_{\alpha}| = 3$. There are two possible addition schemes A (and two A', respectively), namely,

$$\begin{aligned} S_{c_1} &= \{\bar{1}, 0\} \dot{+} \{\gamma + \Delta\} \\ S_{\alpha} &= \{\bar{1}, 1, 3\} \dot{+} \{\gamma - \Delta\} \\ S_{c_2} &= \{\bar{1}, 1\} \dot{+} \{\gamma - \Delta\} \\ S_{\beta} &= 2\{\bar{1}, 1\} \dot{+} 2\{\gamma + \Delta\}, \end{aligned} \tag{A.87}$$

or

$$\begin{aligned} S_{c_1} &= \{0, 1\} \dot{+} \{\gamma + \Delta\} \\ S_{\alpha} &= \{\bar{3}, \bar{1}, 1\} \dot{+} \{\gamma - \Delta\} \\ S_{c_2} &= \{\bar{1}, 1\} \dot{+} \{\gamma - \Delta\} \\ S_{\beta} &= 2\{\bar{1}, 1\} \dot{+} 2\{\gamma + \Delta\}. \end{aligned} \tag{A.88}$$

When A and A' are both in case 2 of Table 2.2, or both in case 3, one must choose $\epsilon = 1$. When A and A' are in different cases (e.g. A is in case 2 and A' is in case 3), one must choose $\epsilon = \bar{1}$. The transformations are obtained in the same way as in the previous situation, and the existence of the reverse transformation is obtained by exchanging the roles of A and A'. The existence of the transformations and their inverses is thus proven.

This completes the proof. ∎

Let us now introduce an equivalence relationship for addition schemes. Consider the redundant binary numbers addition schemes A and A', having parameters (γ, Δ), and (γ', Δ'), respectively. If these addition schemes satisfy Eqn. A.82, then A and A' are said to be related through $\mathcal{R}$. In this way, Lemma 31 assures the existence of functions assuring that the sets of A can be transformed into the sets of A', and those of A' into those of A.

In order to class the space of addition schemes into two groups, one needs to prove that $\mathcal{R}$ is an equivalence relationship.

Lemma 34 *$\mathcal{R}$ is an equivalence relationship.*

Proof. A relationship is an equivalence relationship when it is reflexive, symmetric, and transitive.

- The relationship $\mathcal{R}$ is reflexive. Given $\gamma' = \gamma$, Theorem 15 proves that there exist $\epsilon = 1$ and $\Delta' = \Delta$, such that $A \mathcal{R} A$.
- The relationship $\mathcal{R}$ is symmetric, because the inverse transformations exist.
- The relationship $\mathcal{R}$ is transitive: if $A \mathcal{R} A'$, and if $A' \mathcal{R} A''$, then there exists (ϵ, ϵ') in $\{\bar{1}, 1\}^2$, and there exists $(\Delta, \Delta', \Delta'')$ in $\mathcal{Z}^3$ such that

$$\begin{aligned} T_{c_1}(x) &= \epsilon(x-(\gamma+\Delta))+\gamma'+\Delta' \\ T_{\alpha}(x) &= \epsilon(x-(\gamma-\Delta))+\gamma'-\Delta' \\ T_{c_2}(x) &= \epsilon(x-(\gamma-\Delta))+\gamma'-\Delta' \\ T_{\beta}(x) &= \epsilon(x-2(\gamma+\Delta))+2(\gamma'+\Delta') \end{aligned} \tag{A.89}$$

and such that

$$\begin{aligned} T'_{c_1}(x) &= \epsilon'(x-(\gamma'+\Delta'))+\gamma''+\Delta'' \\ T'_{\alpha}(x) &= \epsilon'(x-(\gamma'-\Delta'))+\gamma''-\Delta'' \\ T'_{c_2}(x) &= \epsilon'(x-(\gamma'-\Delta'))+\gamma''-\Delta'' \\ T'_{\beta}(x) &= \epsilon'(x-2(\gamma'+\Delta'))+2(\gamma''+\Delta'') \end{aligned} \tag{A.90}$$

One can show that

$$\forall v \in \{c_1, \alpha, c_2, \beta\},\ \exists T''_v = T'_v \circ T_v \tag{A.91}$$

Indeed, there exists $\epsilon'' = \epsilon' \times \epsilon$ in $\{\bar{1}, 1\}^2$ and Δ'' in $\mathcal{Z}$ such that

$$\begin{aligned} T''_{c_1}(x) &= \epsilon'\epsilon(x-(\gamma+\Delta)) + \epsilon'(\gamma'+\Delta'-(\gamma'+\Delta'))+\gamma''+\Delta'' \\ T''_{\alpha}(x) &= \epsilon'\epsilon(x-(\gamma-\Delta)) + \epsilon'(\gamma'-\Delta'-(\gamma'-\Delta'))+\gamma''-\Delta'' \\ T''_{c_2}(x) &= \epsilon'\epsilon(x-(\gamma-\Delta)) + \epsilon'(\gamma'-\Delta'-(\gamma'-\Delta'))+\gamma''-\Delta'' \\ T''_{\beta}(x) &= \epsilon'\epsilon(x-2(\gamma+\Delta))+2\epsilon'((\gamma'+\Delta')-2(\gamma'+\Delta'))+2(\gamma''+\Delta'') \end{aligned} \tag{A.92}$$

i.e.

$$\begin{aligned} T''_{c_1}(x) &= \epsilon'\epsilon(x-(\gamma+\Delta)) + (\gamma''+\Delta'') \\ T''_{\alpha}(x) &= \epsilon'\epsilon(x-(\gamma-\Delta)) + (\gamma''-\Delta'') \\ T''_{c_2}(x) &= \epsilon'\epsilon(x-(\gamma-\Delta)) + (\gamma''-\Delta'') \\ T''_{\beta}(x) &= \epsilon'\epsilon(x-2(\gamma+\Delta))+2(\gamma''+\Delta'') \end{aligned} \tag{A.93}$$

i.e. $A \mathcal{R} A''$.

This completes the proof. ∎

Now that one has defined an equivalence relationship for addition schemes, one can find all the corresponding equivalence classes. Theorem 15 provided the background to prove Lemma 31. The lemma established a relationship between A and A' as soon as they were in the same situation listed in Table 2.1. Therefore, $(|S_{c_1}|, |S_{\alpha}|) = (3, 2)$ implies $A\mathcal{R}A'$, and $(|S_{c_1}|, |S_{\alpha}|) = (2, 3)$ also implies $A\mathcal{R}A'$. Moreover, as $\mathcal{R}$ is an equivalence relationship on the set of addition schemes. Since every addition scheme in that set must verify $(|S_{c_1}|, |S_{\alpha}|) = (3, 2)$ or $(|S_{c_1}|, |S_{\alpha}|) = (2, 3)$, then one can separate the set of addition schemes into two disjoint subsets C_{32} and C_{23}, respectively, and $C_{32} \cup C_{23}$ is equal to the whole set of addition schemes itself, thus establishing that there are only two equivalence classes for $\mathcal{R}$.

In the following, a proof is given for Theorem 18.

Proof. Assuming a hardware implementation of A's architecture were found, then three sub-circuits are defined corresponding to the mappings S_1, S_2, and S_3. Using the transformations between A and A' defined earlier, one can obtain a circuit for the cells S_1', S_2', and S_3', i.e. derive a circuit for A'. Define the mapping S_1' as

$$S_1' : (a_{A'}, b_{A'}) \in S_a \times S_b \mapsto (c'_{1,A'}, \alpha_{A'}) \in S'_{c_1} \times S'_\alpha$$

such that

$$\begin{aligned} S_1'(a_{A'}, b_{A'}) &= (c'_{1,A'}, \alpha_{A'}) \\ &= (T_{c_1}(c'_{1,A}), T_\alpha(\alpha_A)) \end{aligned}$$

where $(c'_{1,A}, \alpha_A) = S_1(T_a^{-1}(a_{A'}), T_b^{-1}(b_{A'}))$. Then, define the mapping S_2' as

$$S_2' : (c_{1,A'}, \alpha_{A'}) \in S'_{c_1} \times S'_\alpha \mapsto (c'_{2,A'}, \beta_{A'}) \in S'_{c_2} \times S'_\beta$$

such that

$$\begin{aligned} S_2'(c_{1,A'}, \alpha_{A'}) &= (c'_{2,A'}, \beta_{A'}) \\ &= (T_{c_2}(c'_{2,A}), T_\beta(\beta_A)) \end{aligned}$$

where $(c'_{2,A}, \beta_A) = S_2(T_{c_1}^{-1}(c_{1,A'}), T_\alpha^{-1}(\alpha_{A'}))$. Next, define the mapping S_3' as

$$S_3' : (c_{2,A'}, \beta_{A'}) \in S'_{c_2} \times S'_\beta \mapsto (4s_{A'}) \in S'_s$$

such that

$$\begin{aligned} S_3'(c_{2,A'}, \beta_{A'}) &= 4s_{A'} \\ &= 4T_s(s_A) \end{aligned}$$

where $4s_A = S_3(T_{c_2}^{-1}(c_{2,A'}), T_\beta^{-1}(\beta_{A'}))$. Given the above definitions, the mappings S_1', S_2', and S_3' are A''s mappings, as proven below.

- The inputs and outputs of the defined architecture belong to the expected sets of A' by construction and by definition of the relationship $\mathcal{R}$. Moreover, the intermediate sets are of the expected size.
- One has now to prove that each mapping performs the expected function:

  S_1'. One can write

$$\begin{aligned} c'_{1,A'} + \alpha_{A'} &= T_{c_1}(c'_{1,A}) + T_\alpha(\alpha_A) \\ &= \epsilon(c'_{1,A} - (\gamma + \Delta)) + \gamma' + \Delta' + \epsilon(\alpha_A - (\gamma - \Delta)) + \gamma' - \Delta' \\ &= \epsilon(c'_{1,A} + \alpha_A - 2\gamma) + 2\gamma' \\ &= \epsilon\left(T_a^{-1}(a_{A'}) + T_b^{-1}(b_{A'}) - 2\gamma\right) + 2\gamma' \\ &= \epsilon\left(\epsilon(a_{A'} - \gamma') + \gamma + \epsilon(b_{A'} - \gamma') + \gamma - 2\gamma\right) + 2\gamma' \\ &= a_{A'} + b_{A'} \end{aligned}$$

  because $\epsilon \in \{\bar{1}, 1\}$ (hence $\epsilon^2 = 1$), proving that S_1' performs the expected operation.

S_2'. One can write

$$\begin{aligned}
c_{2,A'}' + \beta_{A'} &= T_{c_2}(c_{2,A}') + T_\beta(\beta_A) \\
&= \epsilon(c_{2,A}' - (\gamma - \Delta)) + \gamma' - \Delta' + \epsilon(\beta_A - 2(\gamma + \Delta)) + 2(\gamma' + \Delta') \\
&= \epsilon(c_{2,A}' + \beta_A - 3\gamma - \Delta) + 3\gamma' + \Delta' \\
&= \epsilon\left(2T_{c_1}^{-1}(c_{1,A'}) + T_\alpha^{-1}(\alpha_{A'}) - 3\gamma - \Delta\right) + 3\gamma' + \Delta' \\
&= 2\epsilon\left(\epsilon(c_{1,A'} - (\gamma' + \Delta')) + (\gamma + \Delta)\right) \\
&\quad + \epsilon(\epsilon(\alpha_{A'} - (\gamma' - \Delta')) + \gamma - \Delta - 3\gamma - \Delta) + 3\gamma' + \Delta' \\
&= 2c_{1,A'} + \alpha_{A'}
\end{aligned}$$

because $\epsilon \in \{\bar{1}, 1\}$, proving that S_2' performs the expected operation.

S_3'. One can write

$$\begin{aligned}
4s_{A'} &= 4T_s(s_A) \\
&= 4\left(\epsilon\left(\frac{1}{4}[2T_{c_2}^{-1}(c_{2,A'}) + T_\beta^{-1}(\beta_{A'})] - \gamma\right) + \gamma'\right) \\
&= \epsilon[2\epsilon(c_{2,A'} - [\gamma' - \Delta']) + 2(\gamma - \Delta) + \epsilon(\beta_{A'} - 2[\gamma' + \Delta']) + 2(\gamma + \Delta)] - 4\epsilon\gamma + 4\gamma' \\
&= 2\epsilon^2 c_{2,A'} + 2\epsilon(\gamma - \Delta) + \epsilon^2\beta_{A'} + 2\epsilon(\gamma + \Delta) - 4\epsilon\gamma \\
&= 2c_{2,A'} + \beta_{A'}
\end{aligned}$$

because $\epsilon \in \{\bar{1}, 1\}$, proving that S_3' performs the expected operation.

Therefore, simple transformations can be used to transform an architecture for A into an architecture for A'. If one chooses the same code for a_A and $T_a(a_A)$, and repeat the same procedure for all the variables and their corresponding transformations, then the circuit that performs S_1' is exactly the same as the one found for S_1, because the same input codes must lead to the same output codes. This remark can also be applied to S_2' and S_3'. Therefore, the same circuit performs addition scheme A and addition scheme A', establishing the proof. ∎

A.5 Conclusion

The developments of the previous sections have established several theorems. The resulting mathematical addition scheme framework allows one to further explore addition scheme similarities.