THE UNIVERSITY OF CALGARY

Redundant Number Multiply-Accumulate-Modularized Digital Filters

by

Vishwas M. Rao

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER

ENGINEERING

CALGARY, ALBERTA

August, 1996

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Redundant Number Multiply-Accumulate-Modularized Digital Filters" submitted by Vishwas M. Rao in partial fulfillment of the requirements for the degree of Master of Science.

B. Nowrouzian

Supervisor, Dr. B. Nowrouzian
Dept. of Electrical and Computer Engineering

L.T. Bruton

Dr. L.T. Bruton
Dept. of Electrical and Computer Engineering

R.A. Stein

Dr. R.A. Stein
Dept. of Electrical and Computer Engineering

M. Ahmadi

Dr. M. Ahmadi
Dept. of Electrical Engineering,
University of Windsor, Canada.

Date: Sept. 4, 1996

As a blazing fire burns firewood to ashes,
so does the fire of knowledge burn to ashes
all reactions to material activities ...

— *The Gītā.*

ABSTRACT

This thesis presents mathematical and graph-theoretic techniques for the design and implementation of digital filters incorporating two important practical features, namely, structural uniformity and fast processing speeds. The desired structural uniformity is achieved through multiply-accumulate (MAC) modularization of the digital filter, where MAC-modularization is defined as the process of translating the filter algorithm consisting of separate multiplication and addition operations into a corresponding algorithm consisting of MAC operations only. The desired fast processing speed, on the other hand, is achieved by using redundant number arithmetic to implement the constituent MAC operations. The use of redundant number arithmetic eliminates the carry-propagation in the corresponding arithmetic operations, leading to a processing speed which is totally independent of the signal wordlength of the digital filter. The proposed techniques are illustrated through their application to the MAC-modularization of a LDI Jaumann digital filter and its implementation using novel redundant number MAC arithmetic architectures.

ACKNOWLEDGEMENTS

The author is deeply indebted to his supervisor Dr. B. Nowrouzian who has been his friend, philosopher, and guide throughout the course of this research. This research was made possible by Dr. Nowrouzian's tremendous support, encouragement, and motivation. The author also wishes to thank him for the extremely careful reading of this thesis and for all the valuable suggestions.

The author greatly appreciates the strong encouragement provided by Dr. R. Stein throughout this program. Many thanks also to the staff within the Department of Electrical and Computer Engineering for all their help in navigating the administrative maze of being a graduate student.

The author wishes to thank his research colleagues: Thomas Borsodi, Robert Morris, Arthur Fuller, and Yvan Botteron, for the reviews during the group meetings, and for the several interesting discussions. A special note of thanks to Umaiyalan Paramananthan, Sridhar Krishnan, and several other friends for their support during demanding times.

The author gratefully acknowledges the financial support for this research, provided in part, by the Department of Electrical and Computer Engineering through the *Graduate Research Scholarship*, by the Natural Sciences and Engineering Research Council of Canada, by Micronet, by Nortel North America, and by the Faculty of Graduate Studies through the *Robert B. Paugh Memorial Scholarship in Engineering.*

The foundation for this research was established by the many researchers whose technical contributions remain a source of inspiration to the author. The author is forever indebted to them.

Finally, the author owes a special thanks to his family for being a constant source of encouragement, support, and optimism throughout.

To

my mother, my father,

and

in the memory of

my grandmother.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

(n, p)	(negative, positive)
(s, v)	(sign, value)
ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
CLA	Carry Lookahead Addition
DSP	Digital Signal Processing
DFF	D-type Flip-Flop
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
IEEE	Institute of Electrical and Electronics Engineers
LDI	Lossless Discrete Integrator
LSB	Least-Significant Bit
LSW	Least-Significant Word
MAC	Multiply-Accumulate
MHz	Megahertz
MSB	Most-Significant Bit
MSW	Most-Significant Word
MUX	Multiplexer
RNE	Rounding to the Nearest/Even
RNU	Rounding to the Nearest/Up
ROM	Read Only Memory
SB	Signed-Binary
SBNR	Signed-Binary Number Representation
SD	Signed-Digit
SDNR	Signed-Digit Number Representation

SFG	Signal Flow Graph
T1-Adder	Type-1 Adder
TC	Two's Complement
TCNR	Two's Complement Number Representation
VLSI	Very Large Scale Integrated
$\lceil . \rceil$	Ceiling function
$\lfloor . \rfloor$	Floor function
r	Radix of a given number
α	Maximum digit magnitude
σ_r	Allowable digit set for a radix-r SD number
x	Any value from the digit set $\{\bar{1}, 0, 1\}$ or the digit set $\{0, 1\}$
$+$	Addition if any operand is non-boolean, OR operation otherwise
$-$	Subtraction operation
$\bar{.}$	Negation operation
$\oplus$	Exclusive-OR operation
$Int(.)$	Integer part of the argument
$< \ldots >$	Group of values
$(.)_r$	Argument represented in radix-r
d	Don't care condition
$sign(.)$	Sign of the argument
$.^{TC}$	Given number is in TCNR
$.^{SB}$	Given number is in SBNR
M	Multiplicand wordlength
N	Multiplier wordlength
D	Digit size
$lcm(\ldots)$	Least common multiple of the arguments

$.mod.$	Modulus operation
θ	Re-pipelining interval
T	Computational time
$max(\ldots)$	Maximum of the arguments
F	Maximum number of inputs-per-gate available in the given technology
l	Lookahead factor
$log_x(.)$	Logarithm to the base x
H	Maximum throughput of a given multiplier
$G(V, E)$	Directed reduced SFG with vertices V and directed edges E
$H(V, E')$	Directed reduced spanning tree of $G(V, E)$ with $E' \subset E$
$\wp$	Root node of $G(V, E)$
n_i	Addition node ($n_i \in V$)
e_{ij}	Directed edge from node n_i to node n_j
m_{ij}	Multiplication coefficient associated with the edge e_{ij}
$d_{in}(v)$	In-degree of the vertex v
$d_{out}(v)$	Out-degree of the verted v
$\mathbf{D}$	In-degree matrix of $G(V, E)$
s	Input node of $G(V, E)$
g_i	Gain from s to n_i in $H(V, E')$
$fitness(H_k(V, E_k))$	Fitness function measuring the fitness of the k^{th} directed reduced spanning tree
$edge\_fitness(e_{ij}, H_k(V, E_k))$	Fitness of the edge e_{ij} with respect to the k^{th} directed reduced spanning tree
$O_{i_{MAX}}$	L1-Norm value of the maximum overflow associated with $H_k(V, E_k)$
$\sum R'_i$	Sum of L1-Norm contributions of cotree multipliers in $H_k(V, E_k)$
$oflow\_weight$	User defined overflow weighting in the fitness function
$roff\_weight$	User defined roundoff weighting in the fintess function

CHAPTER 1

INTRODUCTION

About three decades ago, the introduction of Fast Fourier transform by Cooley and Tukey [19] and the growing expertise in digital circuit technology led to a new era in signal processing [27]. This era was marked by the rapid growth of digital filtering [1] and digital signal processing (DSP) [5].

The field of DSP is concerned with the processing of signals represented in digital form. It is widely used in digital audio and video processing, sonar and radar processing, bio-medical signal processing, speech processing, digital communications, and a host of other applications.

Early developments in the field of DSP borrowed several ideas from analog filter theory and analog signal processing. This was natural since the fields of analog filter theory and analog signal processing were already well established. However, with the passage of time, scientists and engineers quickly realized that digital processing techniques promised several new important practical features, and, at the same time, did not suffer from some of the shortcomings (e.g. aging, drift, etc.) of the corresponding analog processing techniques. This permitted the digital implementation of new and highly sophisticated signal processing algorithms. In addition, the continuing improvements in the allied fields of digital processing technology led to greater acceptance and increased importance of DSP, culminating in an accelerated growth in its potential applications.

Along with the developments in DSP, a parallel breakthrough took place in the field of electronics. Component technology took a major leap forward as it evolved from discrete transistors to very large scale integrated (VLSI) circuits containing hundreds of thousands of transistors on a single chip. With the increased complexity

of VLSI circuits, there was a need to free the designer from many of the low-level design details. This resulted in the birth of computer-aided design (CAD) tools which allowed higher levels of abstraction in VLSI circuit design. These tools permitted substantial improvements in the productivity on the part of the system designers.

During the past decade, rapid advancements have taken place in the fields of VLSI circuits and CAD, along with novel mathematical and algorithmic advances in DSP. This has resulted in a sharp growth in the potential applications of signal processing. However, the growing complexity of modern DSP systems has placed an ever increasing demand on performance, sophistication, and real-time processing, strongly indicating the need for massive computational power.

The availability of low-cost, high-density, fast VLSI circuits makes high-speed processing of large volumes of data practical and cost-effective. In addition, the extraction of concurrency in DSP algorithms results in ultra-high throughputs and permits major technical advancements in real-time DSP applications. However, it is quite obvious that the full potential of VLSI circuits cannot be realized with the existing DSP systems and their underlying arithmetic architectures. This is because the data-dependency limitations of these systems do not permit highly concurrent operations, and the speed limitations inherent in the existing arithmetic architectures do not allow high-speed processing. Furthermore, large design and layout costs suggest the utilization of a repetitive modular structure. Such a structure is not consistent with the existing DSP-systems, mainly due to the fact that the conventional DSP algorithms do not inherently possess modularity. Therefore, modern DSP systems must be designed to possess the desirable features of concurrency, fast processing speeds, and structural modularity.

In the design of modern DSP systems, there exists a serious need for providing means of seamless progression from signal processing theory and algorithms to the corresponding VLSI processor architectures and implementations. This means that

there must be a design philosophy which can be employed for the systematic development of a high-speed modular DSP system starting from the algorithmic level. The development of such a design methodology requires a harmonious blend of ideas from the disciplines of computer engineering and computer science, signal processing, and VLSI circuit design.

This thesis presents a novel systematic design philosophy for the realization of digital filters as a class of high-speed modular DSP architectures. This design philosophy progresses from the algorithmic level to the hardware implementation level achieving two main goals as illustrated in Fig. 1.1.

1. Modularization of the digital filter algorithm.

2. Use of high-speed arithmetic architectures based on redundant number arithmetic.

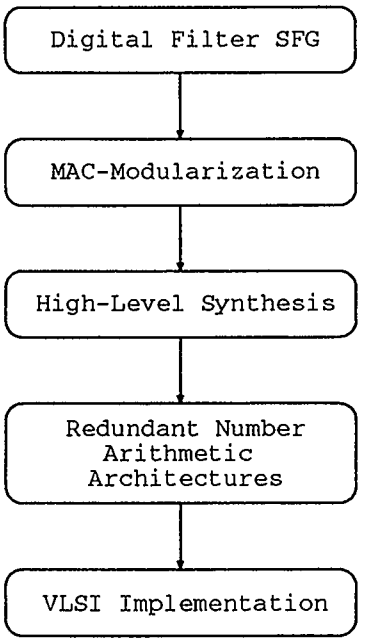

Figure 1.1. Proposed Design Philosophy for DSP Systems

Digital filter algorithms are generally represented in the form of signal-flow graphs

(SFGs) consisting of multiplication and addition operations. Unfortunately, the non-homogeneous nature of these operations does not permit a corresponding straightforward concurrent implementation. However, it is possible to translate these algorithms into suitable equivalent forms involving combined multiply-accumulate (MAC) operations (see Definition 1 below). The translation of a digital filter algorithm consisting of separate multiplication and addition operations to a corresponding algorithm consisting of MAC operations is referred to as MAC-modularization. The resulting MAC-modularized digital filter algorithm possesses regularity and modularity, thereby permitting efficient use of the available computing resources, simpler scheduling, and easier design and implementation. MAC-modularization constitutes the first step in the proposed design philosophy.

Definition 1 *The MAC operation is defined as a composite operation involving the multiplication of a multiplicand X and a multiplier Y and the addition of an addend Z in accordance with*

$$P = X.Y + Z. \tag{1.1}$$

The next step in the above design philosophy involves the high-level synthesis [10] of the MAC-modularized digital filter SFG. The goal of high-level synthesis is to produce a register-transfer level implementation of the digital filter subject to certain specified constraints. This implementation includes a data-path as well as a control-path design.

High-level synthesis includes the tasks of scheduling and allocation. The process of scheduling involves the assignments of operations to various time steps. The process of allocation, on the other hand, involves the binding of the scheduled operations to the corresponding MAC units, as well as the subsequent allocation of the auxiliary resources (registers, multiplexers etc.) to facilitate the data transfers.

The high-level synthesis information is used for the architectural design of a DSP processor for the implementation of the digital filter algorithm. The resulting processor typically consists of a control-unit and a data-path. The control-unit is designed by using the sequence of operations specified in the schedule, and the data-path is designed by using the allocation information to allocate the operations to MAC arithmetic architectures. These architectures achieve high-speed operation by employing redundant number arithmetic. Redundant numbers allow redundancy to exist in the number representation, thereby eliminating carry-propagation to result in very high-speed computation.

Finally, the processor can be implemented in a VLSI circuit by using the full-custom or semi-custom design techniques.

Chapter 2 presents a theoretical background for fixed-point DSP arithmetic by introducing various number systems and arithmetic processing methodologies. This is followed by a rigorous mathematical analysis concerning recoding, rounding, and overflow processing of redundant numbers. A novel 5-digit overlapped scanning technique is presented for modified radix-4 recoding [53] of radix-2 redundant numbers [50]. Furthermore, two techniques for product rounding in radix-2 redundant number arithmetic are developed. Finally, arithmetic overflow processing issues for radix-2 redundant numbers are discussed.

In Chapter 3, the results in Chapter 2 are exploited and applied to the design and implementation of novel high-speed VLSI arithmetic architectures for multiplication and MAC operations. This includes a novel approach for very high-speed mixed-redundant digit-serial [22] modified-Booth [36] multiplication. It is shown that the area-time efficiency and throughput of the resulting multipliers far surpass those of the existing digit-serial modified-Booth multipliers [15]. It is also shown that redundant number arithmetic provides best results for fully-parallel multiplication or MAC operations. Next, a novel architecture for high-speed mixed-redundant parallel

modified-Booth MAC arithmetic operation is presented. Finally, this architecture is extended to handle radix-2 redundant number multiplication by employing the modified radix-4 recoding technique, and is subsequently used for the design of a high-speed fully-redundant parallel MAC arithmetic architecture. These parallel MAC architectures employ new techniques such as partitioned accumulation and concurrent rounding and overflow correction. The resulting architectures are subsequently parameterized in terms of their area-time requirements for corresponding Actel 1.2μ technology implementations, and are verified by using Viewlogic simulations.

Chapter 4 is concerned with a rigorous theoretical approach to MAC-modularization of digital filter SFGs. This approach consists of graph-theoretic techniques and their subsequent translation into algorithms for MAC-modularization. Taking into account the fact that several MAC-modularized digital filter SFGs can result starting from the same initial SFG, a fitness function is developed for the selection of the optimal SFG. This fitness function is based on finite-precision arithmetic effects exhibited by the corresponding MAC-modularized digital filters. Subsequently, enumerative and heuristic approaches to MAC-modularization are developed on the basis of the proposed fitness function. Finally, these approaches are incorporated in a software package called **MAC-M** for the MAC-modularization of digital filters.

In Chapter 5, the high-speed MAC arithmetic architectures developed in Chapter 3 and the MAC-modularization technique developed in Chapter 4 are illustrated by applying them to the design and implementation of a practical lowpass LDI [28] Jaumann [6] digital filter. The optimal MAC-modularized LDI Jaumann digital filter is obtained by using **MAC-M**. The resulting Jaumann digital filter is then partitioned into three separate data-path modules by taking into account the inherent concurrency in the digital filter structure. This concurrency is then exploited to develop a schedule in terms of state equations for each data-path module in order to facilitate efficient high-speed parallel implementation of the filter. The simulation

results demonstrate the achievable operational clock speed of 50 MHz, corresponding to a maximum permissible sample rate of 8.33 MHz for an Actel 1.2μ technology implementation. This implementation is verified by using impulse response simulations. The striking feature of this implementation is that its speed of operation is completely independent of the signal wordlength within the digital filter.

Finally, Chapter 6 presents the conclusions and suggestions for future related research.

CHAPTER 2

THEORETICAL BACKGROUND FOR REDUNDANT NUMBER ARITHMETIC

2.1 Introduction

A generic DSP system consists of an arithmetic unit interposed between the input and output sub-systems, and a controller, as shown in the schematic in Fig. 2.1. The input sub-system accepts digital signals and converts them into a prescribed format required for further processing by the arithmetic unit. The arithmetic unit processes these digital signals in accordance with the operations required in the DSP algorithm. The controller governs the pattern of data flow within the DSP system. The combined operation of the input sub-system, arithmetic unit, and the controller, results in the application of the desired DSP algorithm to the internal input signal, generating the corresponding internal output signal. This output signal is then passed to the output sub-system which converts it into a format required at the output of the DSP system.

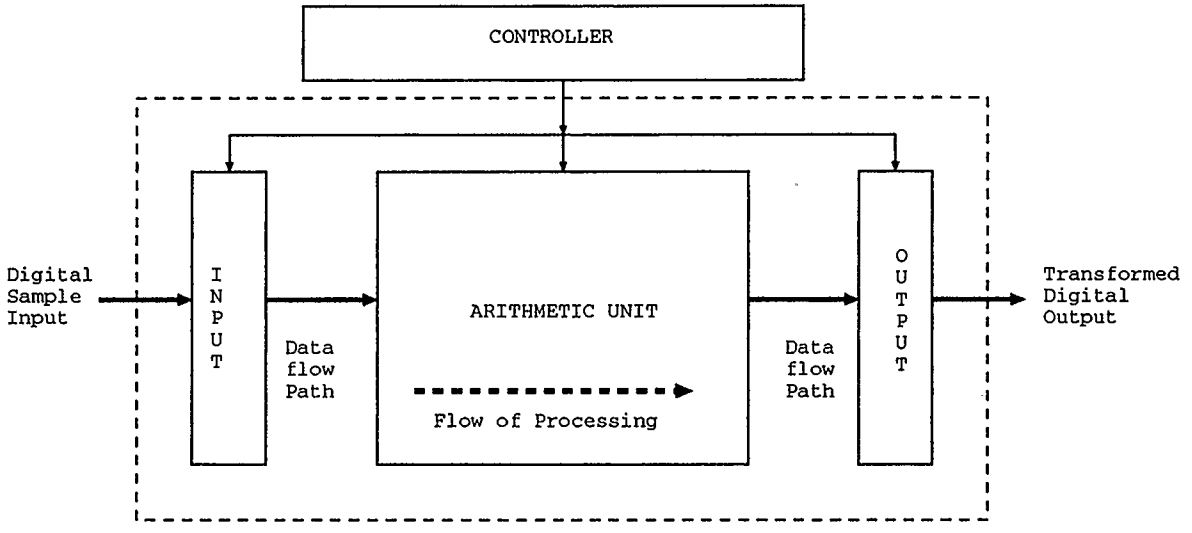

Figure 2.1. A Generic DSP System

The main computation-intensive processing in a DSP algorithm is carried out by the arithmetic unit. It is therefore the computational capability of the arithmetic unit that has a major impact on the performance of the DSP system. The capability of the arithmetic unit is affected by a large number of factors, most importantly, the number representation, the processing methodology, the architecture, and the speed and density of the available VLSI technology.

Section 2.2 presents a detailed discussion of the various number systems and processing methodologies suitable for fixed-point DSP arithmetic functional units. This discussion is followed by an overview of the concepts underlying redundant number arithmetic in Section 2.3. Section 2.4 presents a rigorous mathematical approach for high-speed multiplication and MAC arithmetic operation using redundant number representations. This approach includes a 5-digit overlapped scanning technique for the recoding of radix-2 redundant numbers. This is followed by the development of two techniques to facilitate rounding of radix-2 redundant numbers. Finally, arithmetic overflow processing issues for these numbers are discussed.

2.2 Arithmetic Schemes for Fixed-Point DSP

The computational capability of the arithmetic functional unit is affected by a number of factors, most importantly, the number representation, the processing methodology, the architecture, and the speed and density of the available VLSI technology. Of these, the number representation and the processing methodology have a direct bearing on the maximum speed achievable by the DSP system. This section is concerned with an introduction to the various arithmetic number systems, their features and properties, followed by an overview of the various available processing methodologies.

2.2.1 Number Representation

The main factors involved in the implementation of a DSP algorithm are the manner in which numerical data are stored in memory, and how they are processed by the arithmetic functional unit. The choice of an appropriate internal number representation system has a simultaneous impact on the architecture and performance of the arithmetic unit as well as on the numerical scope available on the corresponding DSP system.

Finite-precision digital arithmetic units restrict the permissible numerical representations to finite length. Therefore, a good choice for the internal number representation and the processing methodology, together with a clever architectural design, affects both the accuracy of approximated real arithmetic as well as the efficient implementation of the machine operations.

Broadly speaking, arithmetic number systems can be divided into the following three categories:

1. Traditional Number Systems

2. Non-traditional Number Systems

3. Quasi-traditional Number Systems

2.2.1.1 Traditional Number Systems

The basis for most of the existing arithmetic functional units is the traditional radix number system. In this system, a radix-r number X is represented by a digital vector of $(n+k)$-tuples as [20]

$$X = (x_{n-1}, \ldots, x_0.x_{-1}, \ldots, x_{-k})_r, \tag{2.1}$$

where the component x_i for $-k \leq i \leq n-1$ is called the i^{th} *digit* of the vector X with $x_i \in \{0, 1, \ldots, (r-1)\}$, and where r (≥ 2) is the *radix* of the number system.

Moreover, the first n digits $(x_{n-1}, \ldots, x_0)$ form the *integer* portion of the number X, and the remaining k negatively indexed digits $(x_{-1}, \ldots, x_{-k})$ form the *fractional* portion of the number X. A *radix-point* is used to divide these portions.

Traditional number systems may be further categorized as [20]:

1. Fixed-radix number system
2. Mixed-radix number system
3. Weighted-radix number system

In the *fixed-radix number system*, all digits assume the same radix value r. A fixed-radix number A having a radix r is represented as

$$A = (a_{n-1}a_{n-2}\ldots a_0)_r, \tag{2.2}$$

where a_{n-1} represents the sign digit which assumes the value $r-1$ (0) if $A < 0$ (≥ 0). For $A \geq 0$, $a_{n-1} = 0$, and the magnitude of A may be represented as

$$|A| = (m_{n-2}m_{n-1}\ldots m_1 m_0)_r, \tag{2.3}$$

where $m_i = a_i$ for all $i \in \{(n-2), \ldots, 1, 0\}$. For the corresponding negative number represented by $\bar{A}$, on the other hand, there are the following three distinct fixed-point number representations:

Sign-Magnitude Number Representation

$$\bar{A} = ((r-1)m_{n-2}\ldots m_1 m_0)_r, \tag{2.4}$$

where m_i for $i \in \{(n-2), \ldots, 1, 0\}$ represents the i-th magnitude digit, and where A and $\bar{A}$ differ only in their sign digits.

Diminished-Radix Complement Number Representation

$$\bar{A} = ((r-1)\bar{m}_{n-2}\ldots \bar{m}_1 \bar{m}_0)_r, \tag{2.5}$$

where $\bar{m}_i = (r-1) - m_i$ for $i \in \{(n-2), \ldots, 1, 0\}$, and where $\bar{A} = r^n - 1 - A$. The radix-2 version of this number representation is called the *one's complement number representation.*

Radix Complement Number Representation

$$\bar{A} = (((r-1)\bar{m}_{n-2} \ldots \bar{m}_1 \bar{m}_0) + 1)_r, \tag{2.6}$$

where $\bar{m}_i = (r-1) - m_i$ for $i \in \{(n-2), \ldots, 1, 0\}$, and where $\bar{A} = r^n - A$. The radix-2 version of this number representation is called the *two's complement number representation.*

Two's complement (TC) number representation (TCNR) is in widespread use nowadays due to its simplicity in representation, processing, and storage [17]. TCNR system forms a major topic of discussion in this thesis.

The *mixed-radix number system* assumes different radix values in different digit positions, and the *weighted-radix number system* associates a variable weighting factor to each digit position [20].

2.2.1.2 Non-Traditional Number Systems

Traditional number representations suffer from the inherent drawback of carry (borrow)-propagation in the addition (subtraction) arithmetic operation which adversely affects the maximum processing speed achievable by the corresponding arithmetic functional units. Non-traditional number systems employ unconventional means of number representation to limit/eliminate carry (borrow)-propagation. These number systems find application in time-critical DSP systems.

Non-traditional number systems can be mainly categorized as:

1. Residue Number System

2. Logarithmic Number System

3. Rational Number System

4. Redundant Number System

Residue numbers [11, 20] have no weighting factor assigned to the digit positions. The residue digits in a residue number can be processed independently, allowing totally carry-free computation.

Logarithmic numbers [12, 20] are represented by exponentials in order to speed up multiplication and division in terms of addition and subtraction operations, respectively. They also enable geometric rounding in order to enhance number accuracy. However, the addition and subtraction of logarithmic numbers is achieved by using look-up tables.

Rational numbers [20] allow the representation of numeric quantities as fractions in terms of numerator-denominator integer pairs. Any arithmetic operation on such numbers always results in rational numbers. This allows closed operations [20] without resorting to infinite precision arithmetic, leading to extremely high accuracy. The rational number system is still in the theoretic stage with regards to implementation.

Redundant numbers [2, 20, 35] allow redundancy to exist in the number representation, thereby permitting a single algebraic value to be represented in several different ways. This redundancy forms the key to the elimination of carry-propagation, allowing very high-speed computation. Redundant arithmetic forms the main component of the discussion in this thesis and is dealt with in greater detail in Section 2.3.

2.2.1.3 Quasi-Traditional Number Systems

Quasi-traditional number systems are a hybrid combination of traditional and non-traditional number systems. A typical quasi-traditional number system is the radix-2 hybrid-redundant number system proposed in [8]. In this number system, parts of the number are in radix complement representation, while the remaining parts are in

redundant number representation, allowing the carry-propagation chains to be limited only to a certain desired fraction of the complete wordlength. The quasi-traditional number systems are beyond the scope of the present thesis.

2.2.2 Processing Methodology

The processing methodology dictates the nature of the data-flow within the DSP system. Therefore, it has a direct impact on the maximum achievable speed and the hardware area requirement of the DSP system. There are four distinct processing methodologies available:

1. Bit-Serial Processing
2. Bit-Parallel Processing
3. Digit-Serial Processing
4. Serial-Parallel Processing

Bit-serial implementations process one input bit at a time and are ideal for low-speed applications [25]. Such implementations require fewer interconnections, less hardware, and less pin-out. Bit-parallel implementations, on the other hand, process all the bits of a word/sample in a single clock-cycle, and require the largest amount of area, interconnection, and pin-out [47, 13]. These implementations are ideal for applications requiring maximum speed. Digit-serial implementations attempt to strike a compromise between the bit-serial and bit-parallel implementations by processing more than one bit per clock-cycle [22]. These implementations are ideal for moderate speed applications, where the bit-serial processing is too slow, and the bit-parallel processing is more expensive in terms of area than necessary. Serial-parallel implementations also attempt to take advantage of both bit-serial and bit-parallel imple-

mentations by processing one of the operands serially and the other in parallel in a single clock-cycle [39].

2.3 Redundant Number Arithmetic

Traditional number representation systems suffer from the inherent problems of carry (borrow) propagation in the addition (subtraction) operation, limiting the performance of DSP systems. Redundant numbers, on the other hand, allow redundancy to exist in the number representation, thereby permitting a single algebraic value to be represented in several different ways. This redundancy forms the key to the elimination of carry (borrow) propagation, for very high-speed computation.

2.3.1 Signed-Digit Number Representation

Signed-Digit (SD) Number Representation (SDNR) [2, 7, 20] systems form a class of number representations which employ redundancy in representation to limit the carry (borrow) propagation to one digit position to the left (assuming right to left arithmetic operation) during addition (subtraction) operation, thereby allowing the elimination of carry (borrow) propagation chains. SDNR may be considered as an extension of the fixed-radix system in the sense that it permits positive and negative weighted digits in the allowable digit-set.

Given a radix r, each digit of a SD number can assume the following $2\alpha + 1$ values [20]

$$\sigma_r = \{\bar{\alpha}, \ldots, \bar{1}, 0, 1, \ldots, \alpha\}, \tag{2.7}$$

where $\bar{\alpha}$ represents $-\alpha$, and where the maximum digit magnitude α must be within the region

$$\lceil \frac{r-1}{2} \rceil \leq \alpha \leq r - 1. \tag{2.8}$$

In the above equation, $\lceil . \rceil$ denotes the *ceiling function.* Taking into account the fact

that $\alpha \geq 1$, Eqn. (2.8) implies that $r \geq 2$. In order to ensure *minimum redundancy* in the balanced digit set σ_r, α must be chosen equal to $\lfloor \frac{r}{2} \rfloor$, where $\lfloor . \rfloor$ denotes the *floor function.* The radix-2 SD system has the digit-set $\sigma_2 = \{\bar{1}, 0, 1\}$ and is also known as the signed-binary (SB) number representation (SBNR) [29]. Similarly, the radix-4 SD system can have the digit-set $\sigma_4^1 = \{\bar{2}, \bar{1}, 0, 1, 2\}$ which constitutes the minimum redundancy digit-set and is also known as the *modified radix-4 representation.* Another possible digit-set for $r = 4$ is $\sigma_4^2 = \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$.

2.3.1.1 Conversion between SDNR and Traditional Radix-r Number System

Let $X = (x_{n-1}, \ldots, x_1, x_0)_r$ be a traditional radix-r number, and let $Y = (y_{n-1}, \ldots, y_1, y_0)_r$ be the equivalent SD number representing the same algebraic value. Then, the conversion from X to Y is carried out by generating an *interim difference* digit d_i for every digit x_i as [20]

$$d_i = x_i - r.b_{i+1}, \tag{2.9}$$

where $b_{i+1} = 0$ (1), if $x_i < \alpha$ ($\geq \alpha$). The i-th SD y_i is then obtained as

$$y_i = d_i + b_i. \tag{2.10}$$

The conversion of the SD number Y back to the traditional form X is achieved by

$$X = |Y^+| - |Y^-|, \tag{2.11}$$

where Y^+ (Y^-) represents the positive (negative) component of Y, and where $|.|$ denotes the *magnitude function.*

2.3.1.2 SDNR Addition and Subtraction

Consider a SD number represented by $n+m+1$ digits z_i ($i = m, \ldots, 1, 0, -1, \ldots, -n$) and having the algebraic value

$$Z = \sum_{i=-n}^{m} z_i r^i, \tag{2.12}$$

where the values of r and z_i are such that the following requirements are satisfied [2].

1. r is a positive integer.
2. The algebraic value $Z = 0$ must have a unique SD representation.
3. There exist transformations between conventional sign-magnitude m-digit representation and SD m-digit representation for every algebraic value within the machine representable range.
4. Totally parallel addition and subtraction is possible for all digits in corresponding positions of the two SD operands.

The arithmetic operations of totally-parallel addition and subtraction of the two digits z_i and y_i from the corresponding i-th positions of the representation of the SD numbers Z and Y are defined as follows [2]:

Definition 2 *Addition of digits z_i and y_i are considered totally parallel if the following two conditions are satisfied:*

1. *The sum digit s_i (i-th digit of the sum $S = Z + Y$) is a function of only z_i, y_i, and the transfer digit t_i from the $(i-1)$-th position on the right: $s_i = f(z_i, y_i, t_i)$.*
2. *The transfer digit t_{i+1} at the $(i+1)$-th position on the left is a function only of the augend digit z_i and the addend digit y_i: $t_{i+1} = g(z_i, y_i)$.*

Definition 3 *Totally parallel subtraction of y_i from z_i is performed as the totally parallel addition of the additive inverse of y_i, i.e., $z_i - y_i = z_i + (\overline{y_i})$.*

The addition of the two digits z_i and y_i are therefore carried out in two successive steps. In the first step, an outgoing transfer digit t_{i+1} and an interim sum digit w_i are formed such that

$$z_i + y_i = r.t_{i+1} + w_i. \tag{2.13}$$

In the second step, the sum digit s_i is formed in accordance with

$$s_i = w_i + t_i. \tag{2.14}$$

As shown in [2], the above definitions lead to the conditions

$$|z_i| \leq r - 1, \tag{2.15}$$

$$t_i \in \{\bar{1}, 0, 1\}, \tag{2.16}$$

and

$$|w_i| \leq r - 2. \tag{2.17}$$

Finally, by using Eqns. (2.15) through (2.17), one arrives at [2]

$$r > 2, \tag{2.18}$$

for totally parallel addition using SDNR.

2.3.2 Signed-Binary Number Representation

The requirement for carry-free addition and subtraction places a restriction on the value of r (c.f. Eqn. (2.18)). However, a need to develop carry-free computation for the case of $r = 2$ (which represents SBNR) arose due to the fact that it allows simple two-level logic realization, and is extremely suitable for VLSI implementation. It has been demonstrated in [2] that totally carry-free addition is possible for a *modified* SD representation if the addition rules in Eqns. (2.13) and (2.14) are modified to allow for the propagation of the transfer digit over two digit positions to the left. The resulting addition is executed in three successive steps in accordance with

$$z_i + y_i = r.t'_{i+1} + w'_i, \tag{2.19}$$

$$w'_i + t'_i = r.t''_{i+1} + w''_i, \tag{2.20}$$

and

$$s_i = w_i'' + t_i'', \tag{2.21}$$

where z_i, y_i, and s_i represent the i-th modified SD positions.

The relationships in Eqns. (2.19) through (2.21) can be translated for a two-step carry-propagation free addition in SBNR as follows [34]. In the first step, the intermediate carry $c_i \in \{\bar{1}, 0, 1\}$ and the intermediate sum digit $s_i' \in \{\bar{1}, 0, 1\}$ are determined at each digit position so that they satisfy the relationship $z_i + y_i = 2c_{i+1} + s_i'$, as shown in Table 2.1 [34], where X can assume any value from the digit set $\{\bar{1}, 0, 1\}$. In the second step, s_i' and c_i are added to form s_i, noting that this addition does not generate any carry. Fig. 2.2 demonstrates an example for such a carry-free addition in SBNR.

Table 2.1. Computation Rule for First Step in Carry-Propagation Free Addition

z_i	y_i	z_{i-1}, y_{i-1}	c_{i+1}	s_i'
1	1	X	1	0
1/0	0/1	*both* ≥ 0	1	$\bar{1}$
1/0	0/1	*either or both* < 0	0	1
0	0	X	0	0
1/$\bar{1}$	$\bar{1}$/1	X	0	0
$\bar{1}$/0	0/$\bar{1}$	*both* ≥ 0	0	$\bar{1}$
$\bar{1}$/0	0/$\bar{1}$	*either or both* < 0	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	X	$\bar{1}$	0

2.3.2.1 SBNR Two-Level Encoding

There are two main methods for the two-level logic encoding of SB numbers for digital implementation, each of which can be *minimally-redundant* or *maximally-*

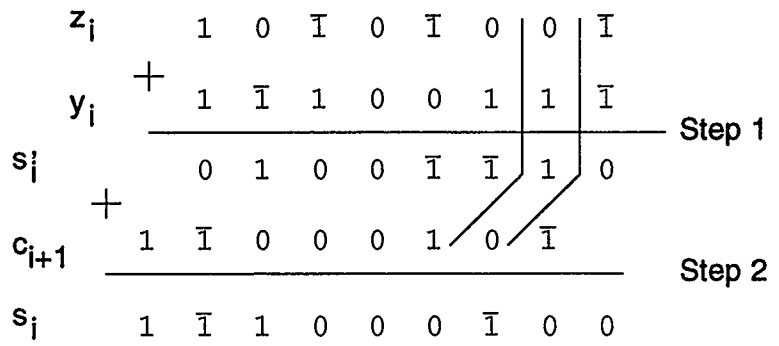

Figure 2.2. Example of Carry-Free Addition using SBNR

redundant. Minimally-redundant encodings employ a restricted set of two-level logic values to represent the SB-digits, leading to a unique two-level logic representation for each SB-digit. Maximally-redundant encodings, on the other hand, exploit the maximum redundancy available in two-level logic, leading to a non-unique two-level logic representation for certain SB-digits.

The above mentioned methods for two-level SB number encoding are:

1. *Negative-Positive* (n,p) *Encoding:* In the (n,p)-encoding, the digits y_i of a SB number Y are represented in their $(negative, positive)$ form given by (y_i^-, y_i^+), where y_i^- represents the negative part and y_i^+ represents the positive part of the digit y_i.

2. *Sign-Value* (s,v) *Encoding:* In the (s,v)-encoding, the digits y_i of a SB number Y, are represented in their $(sign, value)$ form given by (y_i^s, y_i^v), where y_i^s represents the sign and y_i^v represents the value (magnitude) of the digit y_i.

These encodings are given in their minimally and maximally redundant forms as shown in Table 2.2.

2.3.2.2 SBNR Addition and Subtraction Cells

SBNR addition and subtraction cells for minimal two-level encoding have been reported in [34] and [42].

Table 2.2. SBNR Two-Level Encodings

y_i	*minimal* : (y_i^-, y_i^+)	*maximal* : (y_i^-, y_i^+)	*minimal* : (y_i^s, y_i^v)	*maximal* : (y_i^s, y_i^v)
0	$(0,0)$	$(0,0),\ (1,1)$	$(0,0)$	$(0,0),\ (1,0)$
1	$(0,1)$	$(0,1)$	$(0,1)$	$(0,1)$
$\bar{1}$	$(1,0)$	$(1,0)$	$(1,1)$	$(1,1)$

Adder Cell for Minimal (n,p)-encoding

[34] formalized the results in Table 2.1 for the addition of Z and Y using minimal (n,p)-encoding in accordance with

$$z_{id} = z_i^- + z_i^+ \tag{2.22}$$

$$y_{id} = y_i^- + y_i^+ \tag{2.23}$$

$$\rho_i = \overline{z_i^-}.\overline{y_i^-} \tag{2.24}$$

$$u_i = \overline{z_{id}.\overline{y_{id}}.\rho_{i-1} + \overline{z_{id}}.y_{id}.\rho_{i-1} + z_i^+.y_{id} + z_{id}.y_i^+} \tag{2.25}$$

$$t_i = z_{id}.\overline{y_{id}}.\rho_{i-1} + \overline{z_{id}}.y_{id}.\rho_{i-1} + z_{id}.y_{id}.\overline{\rho_{i-1}} + \overline{z_{id}}.\overline{y_{id}}.\overline{\rho_{i-1}} \tag{2.26}$$

$$s_i^+ = \overline{t_i}.\overline{u_{i-1}} \tag{2.27}$$

$$s_i^- = t_i.u_{i-1} \tag{2.28}$$

Adder Cell for Minimal (s,v)-encoding

Similarly, [42] formalized the results in Table 2.1 for the addition of Z and Y using minimal (s,v)-encoding in accordance with

$$\rho_i = z_i^s + y_i^s \tag{2.29}$$

$$t_i^v = z_i^v \bigoplus y_i^v \tag{2.30}$$

$$u_i = t_i^v \bigoplus \rho_{i-1} \tag{2.31}$$

$$v_i = \overline{(t_i^v.\rho_{i-1})}.\overline{(z_i^s.y_i^s)}.(z_i^v + y_i^v) \tag{2.32}$$

$$s_i^s = u_i.\overline{v_{i-1}} \tag{2.33}$$

$$s_i^v = u_i \bigoplus v_{i-1} \tag{2.34}$$

2.3.3 Mixed SB/TC Number Addition and Subtraction

Redundant number arithmetic has found widespread application in TC multiplication due to its carry-free addition property. Several bit-parallel [47] multipliers use intermediate redundant number representation to achieve high-speed TC multiplication. These approaches carry out the intermediate partial product summations in purely redundant number arithmetic. However, such approaches require fully redundant adders which are two to three times more VLSI area expensive compared to conventional full-adders.

Several methods have been investigated to reduce the area costs of TC multipliers employing redundant number arithmetic. One such method is to use mixed SB/TC number addition and subtraction [4, 42].

2.3.3.1 Theoretical Background for Mixed SB/TC Number Addition and Subtraction

Consider a N-digit SB number Y given by

$$Y = \sum_{i=0}^{N-1} y_i 2^i \quad y_i \in \{\bar{1}, 0, 1\}, \tag{2.35}$$

and a N-bit TC number X given by

$$X = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \quad x_i \in \{0, 1\}. \tag{2.36}$$

Let the digits y_i in Eqn. (2.35) be represented in their maximal (n, p)-encoding.

The sum S of Y and X is expressed as

$$S = Y + X. \tag{2.37}$$

Substituting Eqns. (2.35) and (2.36) in Eqn. (2.37), one obtains

$$S = (-x_{N-1} + y^+_{N-1} - y^-_{N-1})2^{N-1} + \sum_{i=0}^{N-2}(x_i + y^+_i - y^-_i)2^i. \tag{2.38}$$

In order to achieve uniform addition, Y (X) is extended one position to the left by padding with a zero (sign extension), to yield

$$S = -x_N 2^N + \sum_{i=0}^{N-1}(x_i + y^+_i - y^-_i)2^i. \tag{2.39}$$

Table 2.3. Generation of s^+_{i+1} and s^-_i

y_i	x_i	s^+_{i+1}	s^-_i
0	0	0	0
1	0	1	1
$\bar{1}$	0	0	1
0	1	1	1
1	1	1	0
$\bar{1}$	1	0	0

The SB sum terms in their maximal (n, p)-encoded form are generated as shown in Table 2.3. By using the results in Table 2.3, Eqn. (2.39) yields

$$S = -x_N 2^N + \sum_{i=0}^{N-1}(2s^+_{i+1} - s^-_i)2^i. \tag{2.40}$$

By using the fact that the 2^N-th term in Eqn. (2.40) corresponds to a negative power, one obtains

$$S = s^-_N 2^N + \sum_{i=0}^{N-1}(2s^+_{i+1} - s^-_i)2^i + s^+_0, \tag{2.41}$$

where $s^+_0 = 0$. Eqn. (2.41) can now be written as

$$S = \sum_{i=0}^{N}(s^+_i - s^-_i)2^i = \sum_{i=0}^{N} s_i 2^i. \tag{2.42}$$

It can be observed that Eqn. (2.42) represents S in its SB format. Moreover, it can be observed from Eqns. (2.39), (2.40), and Table 2.3 that the process of mixed SB/TC number addition is totally carry-free. Mixed SB/TC number subtraction $Y - \dot{X}$ can be achieved in a similar manner, the only difference being that all the bits of X are now complemented and $s_0^+ = 1$.

The above addition and subtraction operations can be implemented by using Type-1 Full-Adders [4, 20]. Fig. 2.3 and Fig. 2.4 demonstrate mixed SB/TC number addition and subtraction, respectively.

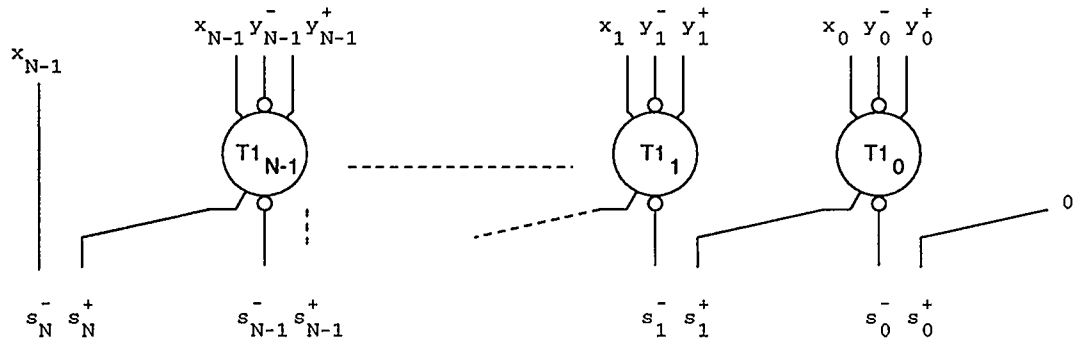

Figure 2.3. Mixed SB/TC Number Addition

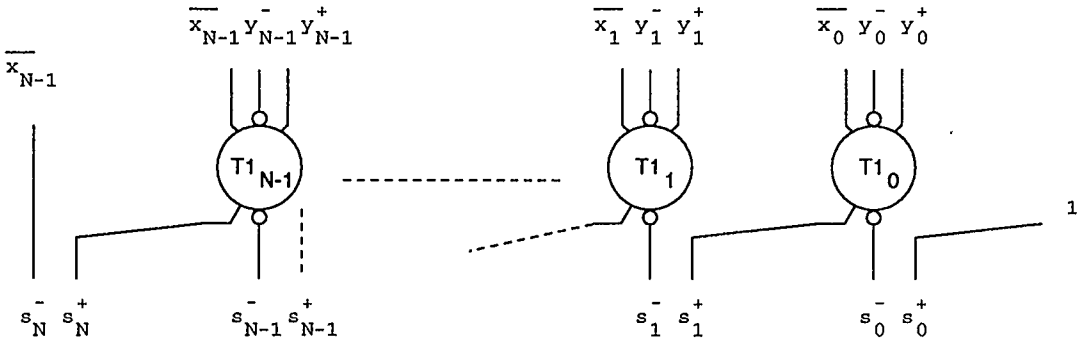

Figure 2.4. Mixed SB/TC Number Subtraction

A similar theoretical approach can be developed for the (s, v)-encoded representa-

tion of Y and S. The addition and subtraction circuits for the minimal (s, v)-encoding based mixed SB/TC number arithmetic are shown in [42].

2.4 Theoretical Background for High-Speed Multiplication and MAC Operation using SB Arithmetic

This section outlines a theoretical background for high-speed multiplication and MAC operation using SB arithmetic. A novel overlapped scanning technique for the modified radix-4 recoding of SB numbers is presented together with its mathematical proof, algorithm, and implementation. This is followed by a mathematical development of rounding techniques for SB arithmetic, resulting in two algorithms for high-speed SB rounding. Finally, the overflow processing aspects of SB arithmetic are discussed.

2.4.1 Modified Radix-4 Recoding of SB Numbers

The key point in high-speed multiplication is the reduction of the number of intermediate partial-product components, and the carry-free computation of the corresponding partial-product sums. While the former involves number transformation, the latter is achieved by redundant number addition/subtraction.

Number transformations require the recoding of a given N-digit number to a $\lceil \frac{N}{k} \rceil$-digit redundant number, where $k \geq 1$. Such transformations are called multibit-recoding and are well known in the case of TC numbers. They permit the recoding of TC numbers to minimally-redundant radix-r SD numbers (with $r \geq 2$). Of these, the most popular recodings involve the cases of $r = 2$ (conventional-Booth recoding [3]) and $r = 4$ (modified-Booth recoding [36]). The modified-Booth recoding is used in almost all the recoded-TC multiplication units, and conducts a 3-bit overlapped scanning on a given TC multiplier in order to convert the multiplier into its corresponding modified radix-4 representation having the digit-set $\{\bar{2}, \bar{1}, 0, 1, 2\}$. This allows the generation of the intermediate partial-product components by uniform shifting, zeroing,

and/or negation of the multiplicand, thereby permitting high-speed and efficient VLSI implementation. However, there is need for such a recoding scheme for SB numbers in order to facilitate high-speed fully-SB multiplication.

In the following, a novel 5-digit overlapped scanning technique is developed for recoding SB numbers to their corresponding modified radix-4 format [49] for high-speed fully-SB multiplication.

2.4.1.1 Theoretical Background for Modified Radix-4 Recoding of SB-Numbers

Let Y represent an N-digit SB-number in accordance with

$$Y = \sum_{i=0}^{N-1} y_i 2^i, \quad y_i \in \{\bar{1}, 0, 1\}. \tag{2.43}$$

Moreover, let a 5-digit overlapped scanning of the digit-sets

$$< \quad y_{2n+3} \quad y_{2n+2} \quad y_{2n+1} \quad y_{2n} \quad y_{2n-1} \quad >$$

be carried out successively for $n = -1, 0, \ldots, (\lceil \frac{N}{2} \rceil - 1)$ (taking $y_i = 0$ for $i < 0$ or $i > (N-1)$) to form the digits

$$z_{n+1} = w_{n+1} + t_{n+1}, \tag{2.44}$$

where w_{n+1} represents the weight digit in accordance with

$$w_{n+1} = \begin{cases} -y_{2n+3} & \text{if } y_{2n+3} = y_{2n+2} \\ -2y_{2n+3} & \text{if } y_{2n+2} = 0 \text{ and } y_{2n+3} = y_{2n+1} \\ 2y_{2n+3} + y_{2n+2} & \text{otherwise} \end{cases} \tag{2.45}$$

and t_{n+1} represents the transfer digit in accordance with

$$t_{n+1} = \begin{cases} Int((y_{2n+1} + y_{2n})/2) & \text{if } y_{2n} \neq 0 \\ Int((y_{2n+1} + y_{2n-1})/2) & \text{if } y_{2n} = 0, \end{cases} \tag{2.46}$$

and where $Int(\cdot)$ represents the integer part of its argument. Finally, let the $(\lceil \frac{N}{2} \rceil + 1)$-digit radix-4 number Z be formed in accordance with

$$Z = \sum_{n=-1}^{\lceil \frac{N}{2} \rceil - 1} z_{n+1} 4^{n+1}. \tag{2.47}$$

Theorem 2.1 *The number Z in Eqn. (2.47) constitutes the modified radix-4 representation of the SB-number Y in Eqn. (2.43).*

Proof Consider the case when $|2y_{2n+1} + y_{2n}| \geq 2$ (including $|2y_{2n+3} + y_{2n+2}| \neq 2$ and $|2y_{2n+3} + y_{2n+2}| = 2$), and the case when $|2y_{2n+1} + y_{2n}| < 2$. By applying Eqns. (2.44), (2.45) and (2.46), and by carrying out an exhaustive enumeration, one obtains

$$|z_{n+1}| \leq 2. \tag{2.48}$$

Furthermore, by considering the cases, $|2y_{2n+1} + y_{2n}| < 2$, $|2y_{2n+1} + y_{2n}| = 2$, and $|2y_{2n+1} + y_{2n}| > 2$, and by applying Eqns. (2.45) and (2.46) exhaustively for every possible instance of the transfer and weight digit, one obtains

$$4t_{n+1} + w_n = 2y_{2n+1} + y_{2n}. \tag{2.49}$$

Eqn. (2.43) can now be expressed as

$$Y = \sum_{i=0}^{(\lceil \frac{N}{2} \rceil - 1)} (2y_{2i+1} + y_{2i})4^i. \tag{2.50}$$

By using Eqn. (2.49) and by partitioning, one obtains

$$Y = \sum_{i=0}^{(\lceil \frac{N}{2} \rceil - 1)} t_{i+1}4^{i+1} + \sum_{i=0}^{(\lceil \frac{N}{2} \rceil - 1)} w_i 4^i. \tag{2.51}$$

By making use of the fact that $t_0 = 0$, and the fact that $w_{\lceil \frac{N}{2} \rceil} = 0$, and by applying Eqn. (2.44) to Eqn. (2.51), one obtains

$$Y = \sum_{i=-1}^{(\lceil \frac{N}{2} \rceil - 1)} z_{i+1}4^{i+1}, \tag{2.52}$$

where the right-hand side corresponds to Eqn. (2.47), leading to

$$Z = Y. \tag{2.53}$$

The proof is established by the fact that Eqn. (2.48) restricts the magnitude of each digit to less than 3, and by the fact that Eqn. (2.53) indicates that the algebraic value of the original number is preserved. ■

2.4.1.2 Algorithm for Modified Radix-4 Recoding of SB-Numbers

The SB-number Y can now be recoded into its corresponding modified radix-4 form Z by using Theorem 2.1. This can be achieved by using the pseudo-code in Algorithm 1 below.

Algorithm 1

```
input:  Y in SBNR;
output:  Z in modified radix-4 SDNR;
begin
   read Y;
   initialize Z = 0;
   set y_i = 0 for i < 0 or i > (N − 1);
   for n = −1 to (⌈N/2⌉ − 1)
   begin
      select the digits y_{2n+3}, y_{2n+2}, y_{2n+1}, y_{2n}, y_{2n−1};
      compute w_{n+1} from the selected digits using Eqn. (2.45);
      compute t_{n+1} from the selected digits using Eqn. (2.46);
      z_{n+1} = w_{n+1} + t_{n+1};
      Z = Z + z_{n+1} 4^{n+1};
   end
   write Z;
end.
```

Fig. 2.5 demonstrates the application of Algorithm 1 (Page 28) to a 16-digit SB-number $Y = (1110100\bar{1}\bar{1}\bar{1}\bar{1}0001\bar{1})_2$ $(= (58913)_{10})$. It is clearly observed that the final output $Z = (10\bar{2}2\bar{2}1\bar{2}01)_4$ is the modified radix-4 representation of Y, and that it preserves the algebraic value $(58913)_{10}$.

2.4.1.3 Implementation of the Modified Radix-4 Recoding of SB-Numbers

This section is concerned with the implementation of the above 5-digit overlapped scanning technique. In what follows, t_{n+1} and w_{n+1} are derived in terms of their

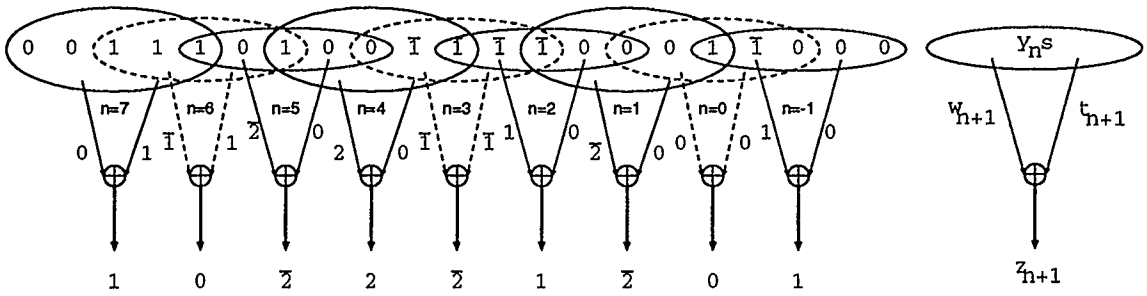

Figure 2.5. Illustration of the Application of the Recoding in Algorithm 1 (Page 28)

(s, v)-encoded forms given by

$$w_{n+1} \equiv < s_{w_{n+1}}, b_{w_{n+1}}, a_{w_{n+1}} > \tag{2.54}$$

$$t_{n+1} \equiv < s_{t_{n+1}}, a_{t_{n+1}} > \tag{2.55}$$

In these equations, $s_{w_{n+1}}$ and $s_{t_{n+1}}$ represent the signs of w_{n+1} and t_{n+1}, respectively. $a_{w_{n+1}}$ and $a_{t_{n+1}}$ represent the 2^0-component of w_{n+1} and t_{n+1}, respectively. $b_{w_{n+1}}$ represents the 2^1-component of w_{n+1}.

The equations for the (s, v) representations of w_{n+1} and t_{n+1} are developed in terms of y_i^0, y_i^1, and $y_i^{\bar{1}}$. Here y_i^0, y_i^1, and $y_i^{\bar{1}}$ are used to indicate that the y_i-th digit of the input word Y is 0, 1, and $\bar{1}$ respectively.

$$s_{w_{n+1}} = y_{2n+3}^0 \cdot y_{2n+2}^{\bar{1}} + \overline{y_{2n+3}^0} \cdot y_{2n+2}^1 + y_{2n+3}^1 \cdot y_{2n+2}^0 \cdot y_{2n+1}^1 + y_{2n+3}^{\bar{1}} \cdot y_{2n+2}^0 \cdot \overline{y_{2n+1}^{\bar{1}}} \tag{2.56}$$

$$b_{w_{n+1}} = \overline{y_{2n+3}^0} \cdot y_{2n+2}^0 \tag{2.57}$$

$$a_{w_{n+1}} = \overline{y_{2n+2}^0} \tag{2.58}$$

$$s_{t_{n+1}} = y_{2n+1}^{\bar{1}} \tag{2.59}$$

$$a_{t_{n+1}} = y_{2n+1}^1 \cdot y_{2n}^0 \cdot y_{2n-1}^1 + y_{2n+1}^{\bar{1}} \cdot y_{2n}^0 \cdot y_{2n-1}^{\bar{1}} + y_{2n+1}^1 \cdot y_{2n}^1 + y_{2n+1}^{\bar{1}} \cdot y_{2n}^{\bar{1}} \tag{2.60}$$

Eqns. (2.56) to (2.60) are now used to derive the final output in terms of z_{n+1}. Again, z_{n+1} is represented in its (s, v)-encoded form as shown below

$$z_{n+1} \equiv < s_{z_{n+1}}, b_{z_{n+1}}, a_{z_{n+1}} > . \tag{2.61}$$

In Eqn. (2.61), $s_{z_{n+1}}$, $b_{z_{n+1}}$, and $a_{z_{n+1}}$ represent the sign, 2^1-component, and the 2^0-component of z_{n+1} respectively, and are given by

$$s_{z_{n+1}} = s_{w_{n+1}} + s_{t_{n+1}}.\overline{a_{w_{n+1}}}.\overline{b_{w_{n+1}}}.a_{t_{n+1}} \tag{2.62}$$

$$b_{z_{n+1}} = \overline{s_{w_{n+1}}}.a_{w_{n+1}}.\overline{s_{t_{n+1}}}.a_{t_{n+1}} + s_{w_{n+1}}.a_{w_{n+1}}.s_{t_{n+1}}.a_{t_{n+1}} + b_{w_{n+1}}.\overline{a_{t_{n+1}}} \tag{2.63}$$

$$a_{z_{n+1}} = \overline{a_{w_{n+1}}}.\overline{b_{w_{n+1}}}.a_{t_{n+1}} + a_{w_{n+1}}.\overline{a_{t_{n+1}}} + b_{w_{n+1}}.a_{t_{n+1}} \tag{2.64}$$

2.4.2 Rounding Techniques for SB-Arithmetic

The multiplication [47] of two single-precision numbers results in a double-precision product. For further storage and processing, this double-precision product must be reduced to a single-precision result while ensuring minimum deviation from the double-precision product. This reduction is achieved by *rounding* [32]. Rounding forms a critical operation in arithmetic functional units. In order to ensure widespread compatibility, current and future arithmetic functional units must adhere to well defined and efficient rounding schemes. Several such rounding schemes already exist, most notably the IEEE standard 754 default rounding to the nearest/even (RNE) [32], and the rounding to nearest/up (RNU) schemes being in widespread use [18].

The following discussion deals with the development of rounding techniques for SB-arithmetic. A relationship that exists between the number truncation in TC-arithmetic and the corresponding truncation in SB-arithmetic is derived. This relationship is subsequently exploited and applied to the development of a pair of novel techniques for SB rounding. These techniques are then translated into algorithms suitable for two-level logic implementation.

2.4.2.1 SB RNU and RNE Techniques

The relationship between the number truncation using TC arithmetic and the corresponding operation using SB arithmetic is derived. This relationship is subse-

quently exploited together with the existing TC RNU and RNE techniques [32] to arrive at the corresponding techniques for RNU and RNE of SB numbers [53].

Truncation of TC numbers and its equivalent in SBNR

The relationship between TC number truncation and the corresponding operation for SB numbers is presented.

Let U represent a N-bit TC number in accordance with

$$U = -u_{N-1}2^{N-1} + \sum_{i=0}^{N-2} u_i 2^i,\ u_i \in \{0,1\}. \tag{2.65}$$

Similarly, let V represent a N-digit SB-number

$$V = \sum_{i=0}^{N-1} v_i 2^i,\ v_i \in \{\bar{1},0,1\}. \tag{2.66}$$

Moreover, let U (V) be partitioned into its most significant word U_{MSW} (V_{MSW}) and least significant word U_{LSW} (V_{LSW}) in accordance with

$$U_{MSW} = -u_{N-1}2^{N-1} + \sum_{i=K}^{N-2} u_i 2^i \quad (V_{MSW} = \sum_{i=K}^{N-1} v_i 2^i), \tag{2.67}$$

$$U_{LSW} = \sum_{i=0}^{K-1} u_i 2^i \quad (V_{LSW} = \sum_{i=0}^{K-1} v_i 2^i), \tag{2.68}$$

where the term 2^K corresponds to the least significant bit (digit) of U_{MSW} (V_{MSW}), so that

$$U = U_{MSW} + U_{LSW}, \tag{2.69}$$

and

$$V = V_{MSW} + V_{LSW}. \tag{2.70}$$

Finally, let

$$U = V. \tag{2.71}$$

The following lemma establishes the relationship between U_{LSW} and V_{LSW}.

Lemma 1 *U_{LSW} can be expressed in terms of V_{LSW} as*

$$U_{LSW} = \begin{cases} V_{LSW} & \text{if } V_{LSW} \geq 0 \\ V_{LSW} + 2^K & \text{otherwise.} \end{cases} \tag{2.72}$$

Proof Let V_{LSW} be expressed as

$$V_{LSW} = \overline{v}_K 2^K + \sum_{i=0}^{K-1} \widehat{v}_i 2^i, \tag{2.73}$$

where $\overline{v}_K \in \{0, 1, \bar{1}\}$ represents the sign digit, and $\widehat{v}_i \in \{0, 1\}$, for V_{LSW}. Then,

$$0 \leq \sum_{i=0}^{K-1} \widehat{v}_i 2^i \leq (2^K - 1). \tag{2.74}$$

Based on Eqns. (2.69) through (2.71), and Eqn. (2.73), a relationship between U_{LSW} and V_{LSW} can be determined based on the following two cases.

Case (a): $V_{LSW} \geq 0$. In this case, Eqns. (2.73) and (2.74) lead to $\overline{v}_K = 0$. Therefore, by using $0 \leq U_{LSW} \leq (2^K - 1)$, together with Eqn. (2.71) and the fact that values in this range are not expressible in powers-of-2 higher than $K - 1$, it follows that

$$U_{LSW} = V_{LSW}. \tag{2.75}$$

Case (b): $V_{LSW} < 0$. In this case, Eqns. (2.73) and (2.74) lead to $\overline{v}_K = \bar{1}$. Therefore, by using Eqn. (2.73), one obtains

$$V_{LSW} = -2^K + \sum_{i=0}^{K} \widehat{v}_i 2^i. \tag{2.76}$$

From Eqn. (2.74) and case *(a)*, it follows that

$$\sum_{i=0}^{K} \widehat{v}_i 2^i = U_{LSW}. \tag{2.77}$$

Substituting Eqn. (2.77) into Eqn. (2.76), one obtains

$$U_{LSW} = V_{LSW} + 2^K. \tag{2.78}$$

Together, cases *(a)* and *(b)* complete the proof. ■

Let U_{TRUN} be the result of truncating U by dropping U_{LSW}. Mathematically, this can be represented as

$$U_{TRUN} = U - U_{LSW}. \tag{2.79}$$

Then, U_{TRUN} can be obtained from V in accordance with the following lemma.

Lemma 2 *U_{TRUN} can be obtained from V as*

$$U_{TRUN} = \begin{cases} V_{MSW} & \text{if } V_{LSW} \geq 0 \\ V_{MSW} - 2^K & \text{otherwise.} \end{cases} \tag{2.80}$$

Proof By using Eqn. (2.79), Eqn. (2.71), and by partitioning V on the basis of Eqns. (2.69) and (2.70), one obtains

$$U_{TRUN} = V_{MSW} + (V_{LSW} - U_{LSW}). \tag{2.81}$$

The proof is completed at once through the application of Lemma 1 to Eqn. (2.81). ■

Rounding of TC numbers and its equivalent in SBNR

The existing TC RNU and RNE techniques are exploited together with Lemma 2 to derive the corresponding operation for SB numbers.

Let U_{LSW} (V_{LSW}) be represented as

$$U_{LSW} = u_{K-1}2^{K-1} + \hat{U}_{LSW}, \tag{2.82}$$

$$V_{LSW} = v_{K-1}2^{K-1} + \hat{V}_{LSW}, \tag{2.83}$$

where

$$\hat{U}_{LSW} = \sum_{i=0}^{K-2} u_i 2^i, \tag{2.84}$$

$$\hat{V}_{LSW} = \sum_{i=0}^{K-2} v_i 2^i, \tag{2.85}$$

and where u_{K-1} (v_{K-1}) represents the *round* [32] bit (digit).

Table 2.4. RNU for SB-numbers

Rounded Value	$\widehat{V}_{LSW} \geq 0$		$\widehat{V}_{LSW} < 0$	
	$v_{K-1} = 0, \bar{1}$	$v_{K-1} = 1$	$v_{K-1} = 0, 1$	$v_{K-1} = \bar{1}$
V_{RNU}	V_{MSW}	$V_{MSW} + 2^K$	V_{MSW}	$V_{MSW} - 2^K$

Theorem 2.2 *The RNU value of V as represented by V_{RNU} can be derived from Eqn. (2.83) and Eqn. (2.70) as given in Table 2.4.*

Proof The proof proceeds by applying TC RNU to U, and by deriving its corresponding equivalent operation as applied to V through the help of Lemma 2.

It is known that TC RNU of U is carried out by the addition of a term 2^{K-1} to U, followed by the truncation of the resulting number up to and including the term 2^{K-1} [32].

The addition of a 2^{K-1} term to U can be mathematically represented by using Eqns. (2.69) and (2.82) as

$$(U + 2^{K-1}) = U_{MSW} + (u_{K-1} + 1)2^{K-1} + \widehat{U}_{LSW}. \tag{2.86}$$

By invoking Eqns. (2.71), (2.70), and (2.83) in Eqn. (2.86) one obtains

$$(U + 2^{K-1}) = V_{MSW} + (v_{K-1} + 1)2^{K-1} + \widehat{V}_{LSW}. \tag{2.87}$$

Next, let $(U + 2^{K-1})_{TRUN}$ be the result of subtracting the algebraic value of $\widehat{U}_{LSW}$ from $(U + 2^{K-1})$ given by

$$(U + 2^{K-1})_{TRUN} = (U + 2^{K-1}) - \widehat{U}_{LSW}. \tag{2.88}$$

Then, based on Eqns. (2.87) and (2.88), one can distinguish two cases:

Case (a): $\widehat{V}_{LSW} \geq 0$. In this case, Lemma 2 yields

$$(U + 2^{K-1})_{TRUN} = V_{MSW} + (v_{K-1} + 1)2^{K-1}. \tag{2.89}$$

To obtain V_{RNU} from Eqn. (2.89), the effect of $(v_{K-1}+1)2^{K-1}$ must be taken into account first, followed by the truncation of the resulting 2^{K-1}-th term.

By substituting the possible values of v_{K-1} in Eqn. (2.89), one obtains

$$(U+2^{K-1})_{TRUN} = \begin{cases} V_{MSW}+2^{K-1} & \text{if } v_{K-1}=0 \\ V_{MSW}+2^{K} & \text{if } v_{K-1}=1 \\ V_{MSW} & \text{if } v_{K-1}=\bar{1} \end{cases} \tag{2.90}$$

By retaining the terms $2^K,\ldots,2^{N-1}$ in the above equation, and through a second application of Lemma 2, one obtains

$$V_{RNU} = \begin{cases} V_{MSW}+2^{K} & \text{if } v_{K-1}>0 \\ V_{MSW} & \text{otherwise.} \end{cases} \tag{2.91}$$

Case (b): $\widehat{V}_{LSW} < 0$. In this case, Lemma 2 yields

$$(U+2^{K-1})_{TRUN} = V_{MSW} + v_{K-1}2^{K-1}. \tag{2.92}$$

By substituting the possible values of v_{K-1} in Eqn. (2.92), one obtains

$$(U+2^{K-1})_{TRUN} = \begin{cases} V_{MSW} & \text{if } v_{K-1}=0 \\ V_{MSW}+2^{K-1} & \text{if } v_{K-1}=1 \\ V_{MSW}-2^{K-1} & \text{if } v_{K-1}=\bar{1} \end{cases} \tag{2.93}$$

By retaining the terms $2^K,\ldots,2^{N-1}$ in the above equation, and through a second application of Lemma 2, one obtains

$$V_{RNU} = \begin{cases} V_{MSW}-2^{K} & \text{if } v_{K-1}<0 \\ V_{MSW} & \text{otherwise.} \end{cases} \tag{2.94}$$

Together, cases *(a)* and *(b)* complete the proof. ■

Theorem 2.2 forms the basis for the derivation of the RNE value of V.

Theorem 2.3 *The RNE value of V as represented by V_{RNE} can be derived from Eqn. (2.83) as given in Tables 2.5 and 2.6.*

Proof The proof is established by demonstrating the validity of the results in Table 2.5, followed by establishing the validity of the results in Table 2.6.

Table 2.5. SBNR RNE for $\widehat{V}_{LSW} \neq 0$

Rounded Value	$\widehat{V}_{LSW} > 0$		$\widehat{V}_{LSW} < 0$	
	$v_{K-1} = 0, \bar{1}$	$v_{K-1} = 1$	$v_{K-1} = 0, 1$	$v_{K-1} = \bar{1}$
V_{RNE}	V_{MSW}	$V_{MSW} + 2^K$	V_{MSW}	$V_{MSW} - 2^K$

Table 2.6. SBNR RNE for $\widehat{V}_{LSW} = 0$

Rounded Value	$\widehat{V}_{LSW} = 0$				
	$v_{K-1} = 0$	$v_{K-1} = 1$		$v_{K-1} = \bar{1}$	
		$v_K = 0$	$v_K \neq 0$	$v_K = 0$	$v_K \neq 0$
V_{RNE}	V_{MSW}	V_{MSW}	$V_{MSW} + 2^K$	V_{MSW}	$V_{MSW} - 2^K$

The relationship between RNU and RNE in TC number representation is as given in Table 2.7 [32], where X can take on any value from the digit set $\{0, 1\}$, and where d represents a don't care condition. Clearly, the results produced by RNU and RNE are identical when $\widehat{U}_{LSW} \neq 0$. This implies that $\widehat{V}_{LSW} \neq 0$, and consequently the results in Table 2.5 hold from Theorem 2.2.

From Table 2.7, it can be seen that RNE differs from RNU only when $u_K = 0$, $u_{K-1} = 1$, and $\widehat{U}_{LSW} = 0$ (the latter implying that $\widehat{V}_{LSW} = 0$). This leads to the following three cases regarding V, depending on the possible values for v_{K-1}.

Case (a): $v_{K-1} = 0$. This case implies that $u_{K-1} = 0$. The use of Theorem 2.2 and Table 2.7 yields

$$V_{RNE} = V_{MSW}. \tag{2.95}$$

Table 2.7. Relationship between TC RNE and RNU

Before Rounding			Add to u_{K-1}		u_K After Rounding	
u_K	u_{K-1}	$\hat{U}_{LSW}$	RNE	RNU	u_K^{RNE}	u_K^{RNU}
X	0	$=0$	d	1	X	X
X	0	$\neq 0$	d	1	X	X
0	1	$=0$	0	1	0	1
1	1	$=0$	1	1	0	0
X	1	$\neq 0$	1	1	$\overline{\text{X}}$	$\overline{\text{X}}$

Case (b): $v_{K-1} = 1$. In this case, the use of Theorem 2.2 yields

$$V_{RNU} = V_{MSW} + 2^K. \tag{2.96}$$

By substituting various possible values for v_K, and by using the result in Eqn. (2.96), one obtains the following.

1. $v_K = 0$: This implies that $u_K = 0$ and $u_{K-1} = 1$. By using Table 2.7 and Eqn. (2.96),

$$V_{RNE} = V_{RNU} - 2^K = V_{MSW}. \tag{2.97}$$

2. $v_K \neq 0$: This implies that $u_K = 1$ and $u_{K-1} = 1$. By using Table 2.7 and Eqn. (2.96),

$$V_{RNE} = V_{RNU} = V_{MSW} + 2^K. \tag{2.98}$$

Case (c): $v_{K-1} = \bar{1}$. In this case the application of Theorem 2.2 yields

$$V_{RNU} = V_{MSW}. \tag{2.99}$$

By substituting various possible values for v_K, and by using the result in Eqn. (2.99), one obtains the following.

1. $v_K = 0$: This implies that $u_K = 1$ and $u_{K-1} = 1$. By using Table 2.7 and Eqn. (2.99),

$$V_{RNE} = V_{RNU} = V_{MSW}. \tag{2.100}$$

2. $v_K \neq 0$: This implies that $u_K = 0$ and $u_{K-1} = 1$. By using Table 2.7 and Eqn. (2.99),

$$V_{RNE} = V_{RNU} - 2^K = V_{MSW} - 2^K. \tag{2.101}$$

The proof for the results in Table 2.6 is completed by cases (a), (b), and (c). This completes the proof of Theorem 2.3. ■

2.4.2.2 Algorithms for SB RNU and RNE

Theorems 2.2 and 2.3 are recast into algorithms suitable for RNU and RNE of SB numbers. These algorithms employ a pair of two-level logic based $SIGN$ and $STICKY$ indicators for the state of $\widehat{V}_{LSW}$, where $SIGN$ is 0 (1) if $\widehat{V}_{LSW} \geq 0$ (< 0), and $STICKY$ is 0 (1) if $\widehat{V}_{LSW} = 0$ $(\neq 0)$. The pseudo-code for the algorithms is as follows.

Algorithm 2 *RNU of SB Numbers.*

```
input:  V;
output:  V_RNU;
begin
   read V;
   decompose V into V_MSW, v_{K-1}, and V̂_LSW;
   compute SIGN from V̂_LSW;
   V_MSW >> K;    /* Shift K digit positions right. */
   if (SIGN == 0)
      if (v_{K-1} == 1)
         V_RNU = V_MSW + 1;
      else
```

```
        V_RNU = V_MSW;
      endif
   else
      if (v_{K-1} == 1̄)
        V_RNU = V_MSW − 1;
      else
        V_RNU = V_MSW;
      endif
   endif
   write V_RNU;
endif.
```

Algorithm 3 *RNE of SB Numbers.*

```
input:  V;
output:  V_RNE;
begin
   read V;
   decompose V into V_MSW, v_{K-1}, and V̂_LSW;
   compute STICKY from V̂_LSW;
   if (STICKY == 0)
      V_MSW >> K;    /* Shift K digit positions right. */
      if (v_{K-1} == 0)
        V_RNE = V_MSW;
      else if (v_{K-1} == 1)
        if (v_K == 0)
          V_RNE = V_MSW;
        else
          V_RNE = V_MSW + 1;
        endif
      else
        if (v_K == 0)
          V_RNE = V_MSW;
        else
          V_RNE = V_MSW − 1;
        endif
```

```
      endif
   else
      compute V_RNU from V using Algorithm 2 (Page 38);
      V_RNE = V_RNU;
   endif
   write V_RNE;
end.
```

2.4.2.3 Implementation of the SB RNE Algorithm

Generalized equations for the implementation of the SB RNE algorithm (c.f. Algorithm 3 (Page 39)) are developed. In order to obtain V_{RNE}, a correction (c_R) is applied to the least significant digit of V_{MSW}. In order to simplify the implementation, c_R is represented in terms of its minimal (n,p)-encoded SB format as (c_R^-, c_R^+). By using Algorithm 3 (Page 39), one can generate c_R as shown in Table 2.8, where X can take on any value from the digit set $\{\bar{1}, 0, 1\}$ for v_K, and any value from the digit set $\{0, 1\}$ for $SIGN$ and $STICKY$.

Table 2.8. SBNR Correction Generation for RNE

v_{K-1}	v_K	$STICKY$	$SIGN$	c_R^+	c_R^-	$Add\ to\ V_{MSW}$
0	X	X	X	0	0	0
1	0	0	X	0	0	0
1	$\neq 0$	0	X	1	0	1
1	X	1	1	0	0	0
1	X	1	0	1	0	1
$\bar{1}$	0	0	X	0	0	0
$\bar{1}$	X	1	1	0	1	$\bar{1}$
$\bar{1}$	$\neq 0$	0	X	0	1	$\bar{1}$
$\bar{1}$	X	1	0	0	0	0

From Table 2.8, the generalized equations for generating c_R are given by

$$c_R^+ = (v_{K-1} = 1).(v_K \neq 0).\overline{STICKY} \; + \; (v_{K-1} = 1).STICKY.\overline{SIGN} \tag{2.102}$$

$$c_R^- = (v_{K-1} = \bar{1}).(v_K \neq 0).\overline{STICKY} \; + \; (v_{K-1} = \bar{1}).STICKY.SIGN \tag{2.103}$$

2.4.3 Overflow Processing for SB Arithmetic

Consider the addition of two SB numbers X and Y to generate the final SB represented sum S. In traditional arithmetic, an overflow occurs if the algebraic value of $X + Y$ is not representable in the available wordlength of S. However, in the case of redundant number arithmetic, an overflow can occur even when the algebraic value of $X + Y$ is within the machine representable length of S. In this way, overflow in SBNR addition can be categorized as either *correctable* or *non-correctable.*

Let the N-digit SB numbers X and Y be represented as

$$X = \sum_{i=0}^{N-1} x_i 2^i \quad x_i \in \{\bar{1}, 0, 1\} \tag{2.104}$$

and

$$Y = \sum_{i=0}^{N-1} y_i 2^i \quad y_i \in \{\bar{1}, 0, 1\}, \tag{2.105}$$

respectively. Furthermore, let the SB sum $S \; (= X + Y)$ be represented as

$$S = s_N 2^N + \sum_{i=0}^{N-1} s_i 2^i \quad s_i \in \{\bar{1}, 0, 1\}, \tag{2.106}$$

where s_N represents the overflow detection-and-correction digit.

Lemma 3 *The overflow indicated by $s_N \neq 0$ is correctable if the first non-zero digit to the right, given by s_k, satisfies the condition*

$$sign(s_N) \neq sign(s_k), \tag{2.107}$$

where sign() denotes the sign of its argument.

Proof Given that an overflow exists, Eqn. (2.107) indicates that the corrected digits s'_k through s'_{N-1} assume the algebraic value given by $s_N 2^N - s_k 2^k$. Since this algebraic value is representable in the digit range $\{k, \ldots, (N-1)\}$, the overall algebraic value of the sum S is representable in the digit range $\{0, \ldots, (N-1)\}$. Therefore, the overflow is correctable. ■

Alternative proofs for correctable overflow can be found in [2] and [35].

Correctable overflows can be corrected by additional processing involving extra computation cycles or extra hardware resources.

An overflow is left uncorrected, if after the attempted correction, the corrected digit $s'_N \neq 0$. Provided that the system is appropriately designed, all the overflows occurring within the system must be correctable. DSP-systems employing fixed-point fractional arithmetic are designed to prevent overflows. In such systems, only *directly correctable overflows* can occur.

Definition 4 *A directly correctable overflow occurs if*

$$(s_N \neq s_{N-1}) \neq 0. \tag{2.108}$$

In this case the corrected digit values s'_N and s'_{N-1} can be determined as

$$s'_N = 0 \tag{2.109}$$

$$s'_{N-1} = 2s_N - s_{N-1}. \tag{2.110}$$

Lemma 4 *In fixed-point DSP systems based on fractional arithmetic, only directly correctable overflows can occur.*

Proof In fixed-point DSP systems based on fractional arithmetic, the absolute maximum algebraic value of S (given by $|S_{MAX}|$) is restricted as

$$|S_{MAX}| \leq 1. \tag{2.111}$$

Based on Eqn. (2.111) and the fact that s_{N-1} corresponds to the 2^0-term in fractional arithmetic, any resulting overflow must obey Eqn. (2.108). Therefore, the overflow is directly correctable (c.f. Definition 4). ■

A different treatment of overflow effects in SBNR can be found in [9].

2.5 Chapter Summary

This chapter has presented a rigorous theoretical background for redundant number arithmetic for applications in DSP systems.

Section 2.2 has dealt with the arithmetic schemes available for fixed-point DSP. This included a discussion of the various possible arithmetic number systems, namely, the traditional, non-traditional, and quasi-traditional number systems. In addition, a discussion was included regarding the processing methodology for DSP systems.

In Section 2.3, the theoretical background has been presented for redundant number arithmetic. The properties of SDNR and the concept underlying carry-free addition and subtraction have been discussed, leading to the extension of the results to SB and mixed SB/TC number arithmetic.

In Section 2.4, a rigorous mathematical approach has been presented for high-speed multiplication and MAC arithmetic operation using SBNR. This approach included a novel 5-digit overlapped scanning technique for the modified radix-4 recoding of SB numbers. This has been followed by the development of two techniques facilitating RNU and RNE of SB numbers. These techniques have been established by developing a relationship between number truncation in SB and TC arithmetic, and by exploiting this relationship together with the available TC RNU and RNE techniques. Finally, arithmetic overflow processing issues for SB numbers have been discussed together with its ramifications regarding *directly correctable overflows* in fixed-point DSP systems.

CHAPTER 3

HIGH-SPEED REDUNDANT NUMBER ARITHMETIC ARCHITECTURES

3.1 Introduction

This chapter presents the exploitation of the theoretical results in Chapter 2 for the development of novel design and implementation techniques for high-speed VLSI multiplication and multiply-accumulate (MAC) arithmetic operations.

The discussions begin with the development of a technique for very high-speed mixed SB/TC digit-serial [22] modified-Booth [36] multiplication, where the high-speed property is realized by eliminating the carry propagation in partial product sum computation. The resulting multipliers incorporate IEEE Standard 754 default rounding presented in Algorithm 3 (Page 39). It is shown that the area-time efficiency and throughput of these multipliers far surpass those of the existing digit-serial modified-Booth multipliers [15]. It is also shown that the use of redundant number arithmetic is most attractive for fully parallel multiplication and MAC operations. The discussions proceed by using mixed SB/TC number arithmetic together with SB RNE technique (c.f. Algorithm 3 (Page 39)) for the development of a high-speed mixed SB/TC MAC arithmetic architecture. The resulting architecture employs the new techniques of partitioned accumulation and concurrent rounding and overflow correction. Subsequently, the modified radix-4 recoding technique (c.f. Algorithm 1 (Page 28)) is used to extend this architecture to handle fully-SB parallel MAC arithmetic operation.

The above developments also include the parameterization of each of the proposed architectures in terms of their area-time requirements for the corresponding Actel 1.2μ technology implementations. The resulting implementations are subsequently verified by using Viewlogic simulations.

The proposed high-speed mixed SB/TC digit-serial modified-Booth multipliers are developed in Section 3.2. Section 3.3 presents the architecture for high-speed mixed SB/TC parallel modified-Booth MAC arithmetic operation. Finally, Section 3.4 presents the high-speed fully-SB parallel MAC arithmetic architecture.

3.2 High-Speed Mixed SB/TC Digit-Serial Modified-Booth Multiplication

In practical DSP applications, it may be desirable to combine the area-efficiency of a bit-serial architecture with the time-efficiency of a corresponding bit-parallel architecture into a single area-efficient and time-efficient digit-serial architecture [16]. Digit-serial architectures [22, 38] process multiple bits of the input data word per clock cycle, where the number of bits processed in each clock cycle is referred to as the digit size. A systematic unfolding technique was presented in [22] for the design of digit-serial architectures. In [16], this approach was exploited and applied to the design and implementation of TC digit-serial modified-Booth [36, 3] multipliers. Unfortunately, TC number arithmetic suffers from the inherent carry/borrow propagation problems. For higher values of the digit-size, the carry/borrow chain increases in length, resulting in an increased critical path length and reduced achievable operational speed. In addition, the underlying digit-serial unfolding technique increases the bit-level pipelining distance by the digit-size, forming another propagation path which is orthogonal to the original carry/borrow propagation path.

The creation of the above propagation paths sets an upper limit on the achievable operational speed in the existing digit-serial modified-Booth multipliers, rendering these multipliers area-efficient and time-efficient only for digit-sizes close to 4. In addition, these modified-Booth multipliers require sign extension of the intermediate partial product sum components for correct TC multiplication which leads to lengthy and non-uniform interconnects in the corresponding hardware implementation. Such interconnects grow with increasing the digit size.

This section deals with a novel approach to high-speed digit-serial modified-Booth multiplication based on mixed SB/TC number arithmetic [48]. The resulting high-speed property is realized by eliminating the above carry/borrow propagation in the constituent partial product sum computations. This is achieved by computing the partial product sum components in SB format, while maintaining the corresponding intermediate partial product components in TC format. IEEE Standard 754 rounding of the resulting full-precision SB product is achieved by using Algorithm 3 (Page 39). The rounded product is subsequently converted into its TC format by employing a simple high-speed look-ahead conversion.

The salient feature of the resulting digit-serial modified-Booth multipliers is that they permit very high throughputs for arbitrary values of the digit size. Moreover, they lead to combined area-efficient and time-efficient implementations even for the values of the digit size exceeding 4 where the conventional modified-Booth multipliers begin to become inefficient. In addition, they do not involve any sign extension, permitting uniform implementations with highly localized interconnections (suitable for practical implementation in VLSI).

3.2.1 Theoretical Background for High-Speed Mixed SB/TC Digit-Serial Modified-Booth Multiplication

In the proposed mixed SB/TC digit-serial modified-Booth multiplication approach, the M-bit TC multiplicand X^{TC} and the N-bit TC multiplier Y^{TC} are first decomposed into a set of D radix-2^D components X_i^{TC} and Y_i^{TC}, respectively, where $i \in \{0, 1, \ldots, D-1\}$, and where D is the digit size (the mathematical details underlying this decomposition are presented in [15]). The conventional modified-Booth recoding technique is then applied to the decomposed components Y_i^{TC} by an overlapped scan on triplets of the multiplier bits y_{2n+1}, y_{2n}, and y_{2n-1} (for $n \in \{0, 1, \ldots, \lceil \frac{N}{2} \rceil - 1\}$,

with $y_{-1} = 0$) to obtain the full-precision TC product as

$$P^{TC} = \sum_{n=0}^{\lceil \frac{N}{2} \rceil - 1} z_n X^{TC} 4^n, \tag{3.1}$$

where

$$z_n = q_n(2b_n + a_n), \tag{3.2}$$

where a_n, b_n, and the sign bit s_n are obtained from the multiplier bits as

$$a_n = y_{2n-1} \bigoplus y_{2n}, \tag{3.3}$$

$$b_n = \overline{(y_{2n-1} \bigoplus y_{2n})}.(y_{2n} \bigoplus y_{2n+1}), \tag{3.4}$$

$$s_n = y_{2n+1}, \tag{3.5}$$

and where q_n is defined in terms of the sign bit s_n as

$$q_n = \begin{cases} 1 & \text{if } s_n = 0 \\ \bar{1} & \text{if } s_n = 1. \end{cases} \tag{3.6}$$

By representing the full-precision TC product in Eqn. (3.1) in its SB format as

$$P^{SB} = \sum_{i=0}^{D-1} P_i^{SB} = \sum_{n=0}^{M+N-1} p_n 2^n, \; p_n \in \{\bar{1}, 0, 1\}, \tag{3.7}$$

one can form the corresponding digit-serial full-precision product components P_i^{SB} in accordance with

$$P_i^{SB} = \sum_{n=0}^{\lceil \frac{N}{2} \rceil - 1} pp_i^{TC}, \tag{3.8}$$

where the constituent intermediate partial product components pp_i^{TC} are given by [15]

$$pp_i^{TC} = p_n[a_n X^{TC}_{(i-2n-1) mod D} 2^{\lceil (2n+1-i)/D \rceil D} + b_n X^{TC}_{(i-2n) mod D} 2^{\lceil (2n-i)/D \rceil D}], \tag{3.9}$$

for $i \in \{0, 1, \ldots, D-1\}$.

The full-precision product components P_i^{SB} for each value of i can be determined as

$$P_i^{SB} = PPS^{SB}_{i,N/2} \tag{3.10}$$

through successive applications of the recursive relationship

$$PPS^{SB}_{i,n+1} = PPS^{SB}_{i,n} + pp^{TC}_{i} \tag{3.11}$$

in n.

The salient feature of Eqns. (3.11) is that it permits the computation of the terms $PPS^{SB}_{i,n+1}$ independently of each other for the various values of i, which is made possible due to the absence of carry/borrow propagation between the terms $PPS^{SB}_{i_1,n+1}$ and $PPS^{SB}_{i_2,n+1}$ for $i_1, i_2 \in \{0, 1, \ldots, D-1\}$. Then, the corresponding TC product is obtained as

$$P^{TC}_{i} = |P^{SB+}_{i}| - |P^{SB-}_{i}|. \tag{3.12}$$

The proposed digit-serial modified-Booth multipliers exploit the above parallelism in terms of i (arising from mixed SB and TC computation) together with that in terms of n (arising from the nature of digit-serial computation) to secure a very high-speed multiplication.

3.2.1.1 Algorithm for Mixed SB/TC Digit-Serial Modified-Booth Multiplication

The pseudo-code for the mixed SB/TC digit-serial modified-Booth multiplication is as follows.

Algorithm 4

```
input:  X^TC, Y^TC, M, N, D;
   /* M and N are multiplicand and multiplier wordlengths, respectively. */
   /* D is the digit size. */
output: P^TC_RNE;
begin
   read X^TC, Y^TC, M, N, D;
   for i = 0 to D-1 do
   begin
```

$PPS_{i,0}^{SB} = 0$

decompose X^{TC} and Y^{TC} into radix-2^D components X_i^{TC} and Y_i^{TC}

end

for n = 0 to $\lceil \frac{N}{2} \rceil - 1$ do

begin

$z_n = (-2y_{2n+1} + y_{2n} + y_{2n-1})$;

evaluate p_n, a_n and b_n from z_n;

for i = 0 to D-1 do

begin

$pp_{i,n}^{TC} = p_n(b_n X_{(i-2n-1) mod D} 2^{\lceil (2n+1-i)/D \rceil D} + a_n X_{(i-2n) mod D} 2^{\lceil (2n-i)/D \rceil D})$;

$PPS_{i,n+1}^{SB} = PPS_{i,n}^{SB} + pp_{i,n}^{TC}$;

end

end

initialize $P^{SB} = 0$;

for i = 0 to D-1 do

begin

$P_i^{SB} = PPS_{i,N/2}^{SB}$;

$P^{SB} = P^{SB} + P_i^{SB}.2^i$;

end

decompose P^{SB} into its MSW and LSW;

Apply Algorithm 3 (Page 39) to P^{SB} and obtain P_{RNE}^{SB};

$P_{RNE}^{TC} = |P_{RNE}^{SB+}| - |P_{RNE}^{SB-}|$;

write P_{RNE}^{TC};

end.

3.2.2 Mixed SB/TC Digit-Serial Addition and Subtraction

The architecture of a mixed SB/TC digit-serial adder for a digit size of D is as shown in Fig. 3.1, where Δ denotes the bit-delay operator. This architecture shows the addition of a TC number X and a SB number Y. There are $lcm(N, D)/N$ two-input multiplexors associated with this adder, where $lcm()$ denotes the *lowest common multiple* of its arguments. The multiplexors are shown in dotted lines since they exist

only when the condition $i = NmodD, 2NmodD, ..., [(lcm(N,D)/N - 1)N]modD$ is satisfied. If the condition is not satisfied, the associated multiplexor does not exist and the signal c_i^+ is directly connected to the output s_i^+. It can be observed that the absence of any carry-propagation path from one T1-adder to another ensures constant addition time (equal to the delay through a T1-adder), regardless of the digit-serial unfolding factor D.

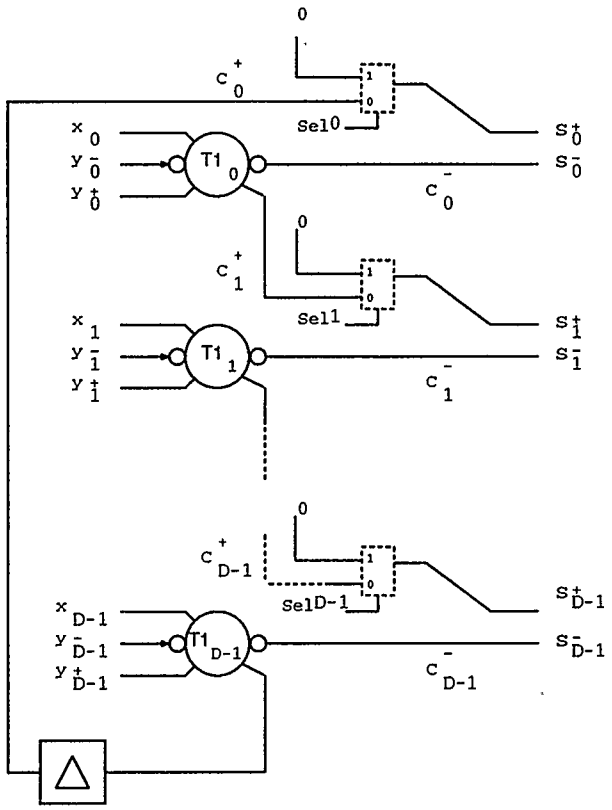

Figure 3.1. Mixed SB/TC Digit-Serial Addition

The architecture of a mixed SB/TC digit-serial subtractor is as shown in Fig. 3.2. The subtraction is achieved by complementing all the bits of the TC number X to be subtracted from the SB-number Y. When the LSB signal is indicated by the SEL_i signal, a 1 is fed to the s_i^+ output. Again, it can be seen that the subtraction operation is totally borrow-free.

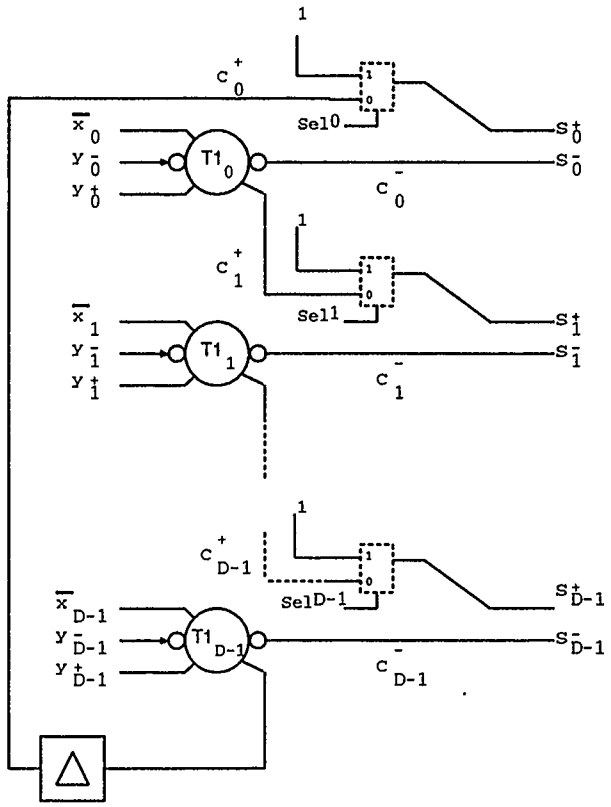

Figure 3.2. Mixed SB/TC Digit-Serial Subtraction

3.2.3 Architecture for High-Speed Mixed SB/TC Digit-Serial Modified-Booth Multipliers

This subsection is concerned with the translation of Algorithm 4 (Page 48) into an architecture for the design and implementation of very high-speed mixed SB/TC digit-serial modified-Booth multipliers. The proposed implementation consists of a tandem connection of $(\lceil N/2 \rceil + 1)$ modules as shown in the schematic diagram in Fig. 3.3. The first $(\lceil N/2 \rceil - 1)$ modules in this diagram are structurally identical. Furthermore, the $\lceil N/2 \rceil$-th module is the same as its predecessor modules save for the fact that it contains an additional pre-rounding circuit. Finally, the $(\lceil N/2 \rceil + 1)$-th module contains the circuit for rounding and digit-serial SB to TC conversion.

The communication between the $(\lceil N/2 \rceil + 1)$ modules in the proposed digit-serial

multiplier occurs as follows. The first $\lceil N/2 \rceil$ modules have XI_i, YI_i, $MSWI_i$, $LSWI_i$, CI, and CLI as their input signals, while having XO_i, YO_i, $MSWO_i$, $LSWO_i$, CO, and CLO as their corresponding output signals (except for the output signals of the $\lceil N/2 \rceil$-th module which will be discussed later). Here, XI_i (XO_i) and YI_i (YO_i) represent the decomposed TC multiplier input (output) and TC multiplicand input (output) components, respectively. Moreover, $MSWI_i$ ($MSWO_i$) and $LSWI_i$ ($LSWO_i$) represent the SB most-significant digit input (output) and SB least-significant digit input (output) components, respectively. Finally, CI (CO) represents the multiplication control signal input (output), and CLI (CLO) represents the rounding control signal input (output).

The $\lceil N/2 \rceil$-th module generates the output signals MO_i, *Sticky*, *Sign*, and CO, which are then consumed as input signals by the $(\lceil N/2 \rceil + 1)$-th module. Here, MO_i represents the truncated product, and *Sticky* and *Sign* represent the rounding correction to be applied to MO_i to result in the final rounded SB product. The $(\lceil N/2 \rceil + 1)$-th module generates the IEEE Standard 754 rounded TC product P_i with its least significant bit synchronized to the rising edge of the signal CO.

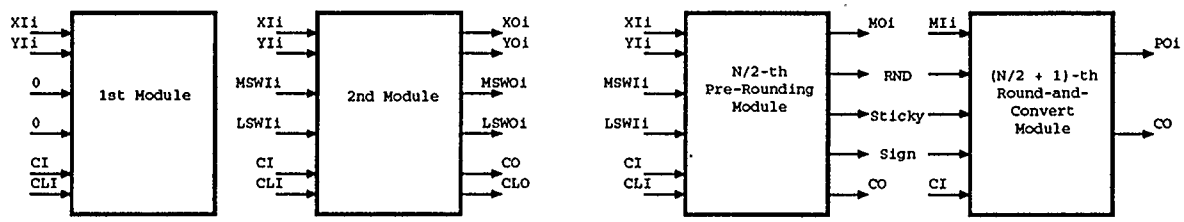

Figure 3.3. Level-1 Architecture of Mixed SB/TC Digit-Serial Multiplication Unit

Each of the modules are described in the following, for generalized values of D, n and i.

3.2.3.1 Generic Hardware Module for Mixed SB/TC Digit-Serial Modified-Booth Unit for General Values of D, n, and i

The generic hardware module constitutes the modules 1 through $(\lceil N/2 \rceil - 1)$, as shown in Fig. 3.4 for general values of D, n, and i. This module has D vertically stacked sub-cells for computation of the partial product sum components in terms of $LSWO_{(i+1)modD}$ and $MSWO_{(i+1)modD}$. The dotted elements are present only if the condition given is satisfied. If the condition fails, the output of the preceding element is directly connected to the input of the current element. The heavy lines represent the pair of signals required to carry a single SB-digit.

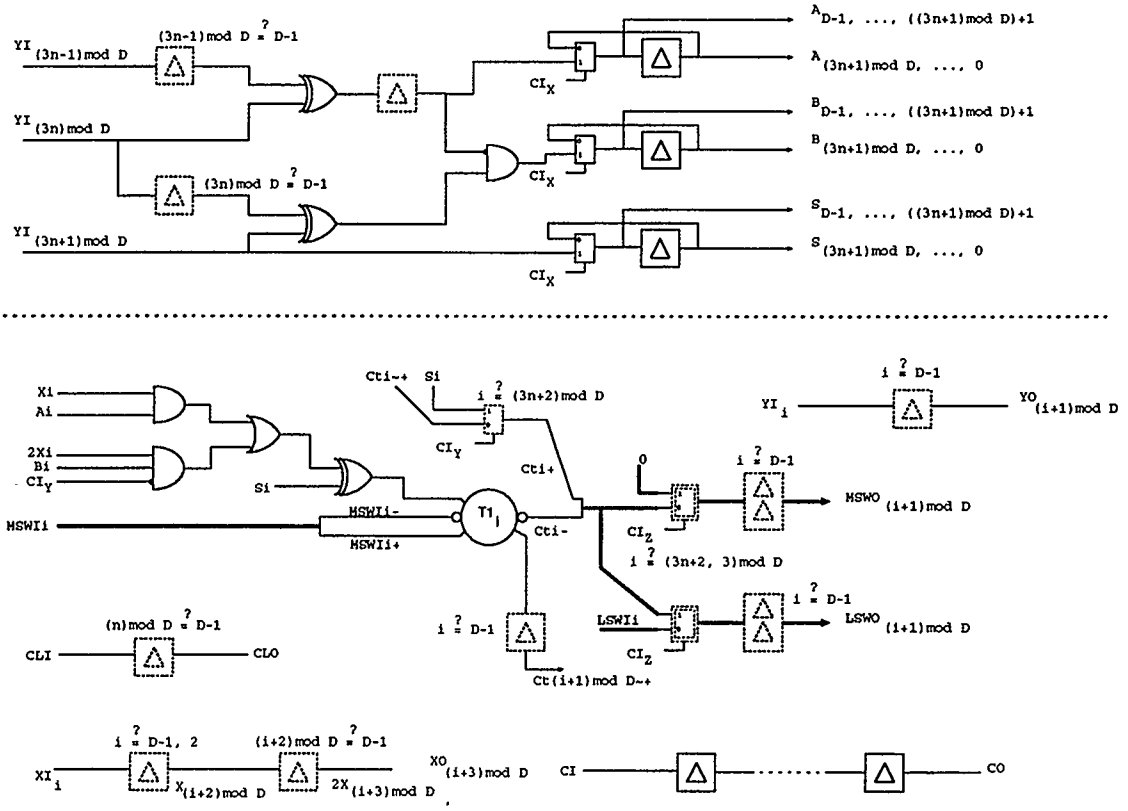

Figure 3.4. Generic Hardware Mixed SB/TC Digit-Serial Multiplication Module for Generalized Values of D, n and i.

The decomposed multiplier components Y_i shown in the module in Fig. 3.4, are

used to form the terms A_i, B_i and S_i, which are then used to form the intermediate TC partial-product components pp_i. The i-th T1-adder sums the pp_i-th component with the $MSWI_i$-th component. This addition generates the components Ct_i^- and Ct_{i+1}^+. The component Ct_i^- is subsequently combined with Ct_i^+ to generate the $(i+1)modD$-th SB-digit in the form of $PPSO_{(i+1)modD}$. The assertion of the signal CI_z forces the first digit of the corresponding $PPSO_i$ to be sent out as the $LSWO_i$. The component Ct_{i+1}^+ however, constitutes the positive part of the next higher power-of-two digit, and is generated in the same or the next clock cycle depending on the value of i (c.f. Fig. 3.4).

The control signals associated with the modules are given by [15]

$$CI_x = CI[\lfloor(3n+1)/D\rfloor - \lfloor(3-D)n/D\rfloor - n] \tag{3.13}$$

$$CI_y = \begin{cases} CI[\lfloor(3n+2)/D\rfloor - \lfloor(3-D)n/D\rfloor - n] & \text{for } i = (3n+2)modD \\ GND & \text{otherwise} \end{cases} \tag{3.14}$$

$$CI_z = \begin{cases} CI[\lfloor(3n+2)/D\rfloor - \lfloor(3-D)n/D\rfloor - n] & \text{for } i = (3n+2)modD \\ CI[\lfloor(3n+3)/D\rfloor - \lfloor(3-D)n/D\rfloor - n] & \text{for } i = (3n+3)modD \end{cases} \tag{3.15}$$

In addition, the control-signal CLI is passed through these modules with the delay existing only if the specified condition is satisfied. This signal indicates the computation of the least-significant-digit of the product.

It can be observed that the partial product sum is in SBNR. This means that there is no sign-extension required for the $MSWO_i$ components. Therefore, it is sufficient to attach zeros to the most-significant part of $MSWO_i$ during the assertion of CI_z. This is unlike traditional TC multiplication, where sign-extension is required. The absence of sign-extension in the proposed multipliers results in local interconnections of short length within the modules. This is a definite advantage in comparison to the fully TC digit-serial modified-Booth multipliers, where sign-extension introduces global connections of large lengths within the corresponding modules.

From Fig. 3.4, it can be observed that the $PPSO_i$ components are computed in parallel, with the computation time being independent of the digit-size (due to the absence of carry-propagation).

3.2.3.2 Pre-Rounding Module for the Mixed SB/TC Digit-Serial Modified-Booth Unit for General Values of D, n, and i

The architecture of the Pre-rounding module (which corresponds to the $\lceil \frac{N}{2} \rceil$-th module) is shown in Fig. 3.5. This module consists of three sections, namely, the Encoder, the Decoder and the Pre-Round section. The Encoder and Decoder sections operate in the manner described in the previous section except for the fact that $PPSO_i$ redirection towards $LSWO_i$ does not occur in the Decoder. Furthermore, the MSW of the product is directed towards MO_i once CI_y is asserted in the Decoder. The round-digit corresponding to p^{SB}_{N-2} is extracted and passed separately to the next module in the form of RND. This digit is formed at the $(3n+2) mod D$-th cell in the Decoder on the assertion of CI_y. Simultaneously, the Pre-Round section computes the $SIGN$ and $STICKY$ bits. $SIGN$ is generated by first computing a generalized bit-value SC_i for the i-th $LSWI$ by taking into consideration the sign of the its immediate least-significant word. Table 3.1 depicts the encoding scheme for the generalized SC_i, where X can take on any value from the digit set $\{0, 1\}$.

Table 3.1. Computation of the Generalized SC_i

$LSWI_i$	*Intermediate* $\overline{SIGN}$ *upto* $LSWI_i$	SC_i
0	0	0
0	1	1
1	X	1
$\bar{1}$	X	0

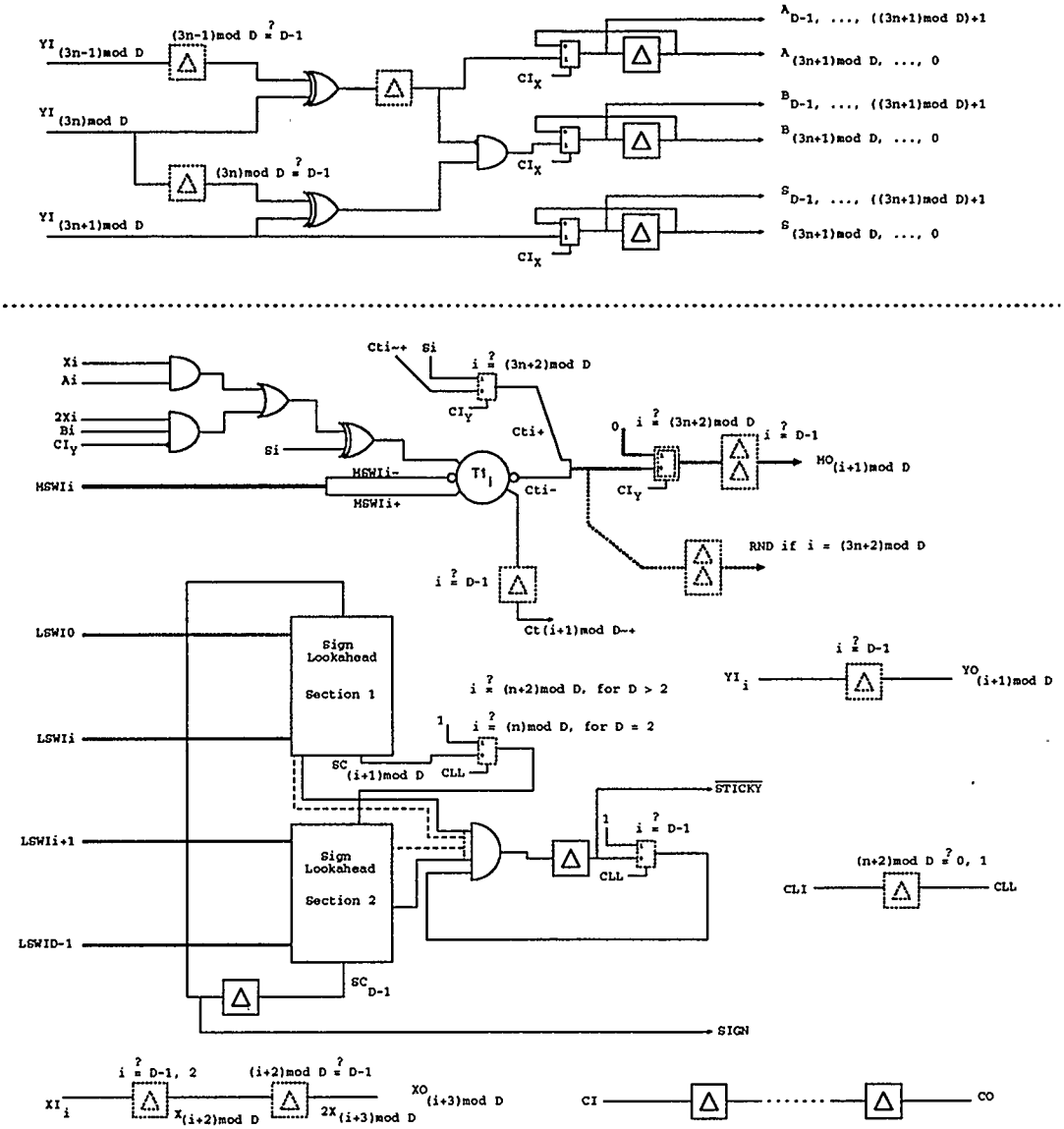

Figure 3.5. Pre-Rounding Module for Generalized Values of D, n, and i.

The components SC_i are all generated in parallel using a lookahead section [20]. This is achieved by examining the positive and negative parts of each $LSWI_i$ as shown in Fig. 3.5. The computation of the least-significant SB-component (p_0^{SB}) of the product is indicated by the assertion of the signal CLL. This initializes the SIGN and STICKY-bit generation logic. $\overline{SIGN}$ corresponds to the last SC_i generated on the assertion of CI_y. $STICKY$ is also generated from the lookahead section. It can

be observed that, computing the $STICKY$-bit is very simple since all the $LSWI_i$s are in SB-format. The assertion of CI_y causes the $SIGN$ and $STICKY$-bits to be latched for further processing by the $(\lceil N/2 \rceil + 1)$-th module.

3.2.3.3 Round and Convert Module for the Mixed SB/TC Digit-Serial Modified-Booth Unit for General Values of D, n, and i

The outputs of the Pre-round module are fed to the Round and Convert module. This module carries out the following functions:

- Rounding (IEEE Standard 754 RNE) of the SB-MSW.
- Conversion of the final SB-RNE result into its corresponding TC-form.

The IEEE Standard 754 RNE is carried out by using the $(N-1)$-th digit (i.e. LS-digit of the SB-MSW) in conjunction with the $SIGN$, $STICKY$ and RND information generated by the Pre-Round module. The correction c_R (c.f. Eqns. (2.102) and (2.103)) is generated as shown in Fig. 3.6, and is subsequently applied to the MSW of the product in two steps. The principle underlying this two-step correction is explained with an example in Fig. 3.7, which depicts the application of a correction of $+1$ $((c_R^-, c_R^+) = (0,1))$ to the SB-number $\bar{1}101$.

The incoming SB-MSW is first recoded in *Step-1* as shown in Table 3.2. This allows the negative component of the correction (c_R^-) to be applied without generating a carry/borrow. The resulting SB-number after the negative correction is re-recoded as shown in Table 3.3, to allow the positive component of the correction (c_R^+) to be consumed without generating a carry/borrow. Fig. 3.7 depicts the process.

The rounding is triggered on the assertion of the signal CI_x, by the application of the two-step correction to MI_i. RO_i are generated in parallel, and are subsequently consumed by the lookahead converter [20] in order to form the TC-result P_i^{TC}. The degree of lookahead possible for a given digit-size depends upon the number of inputs

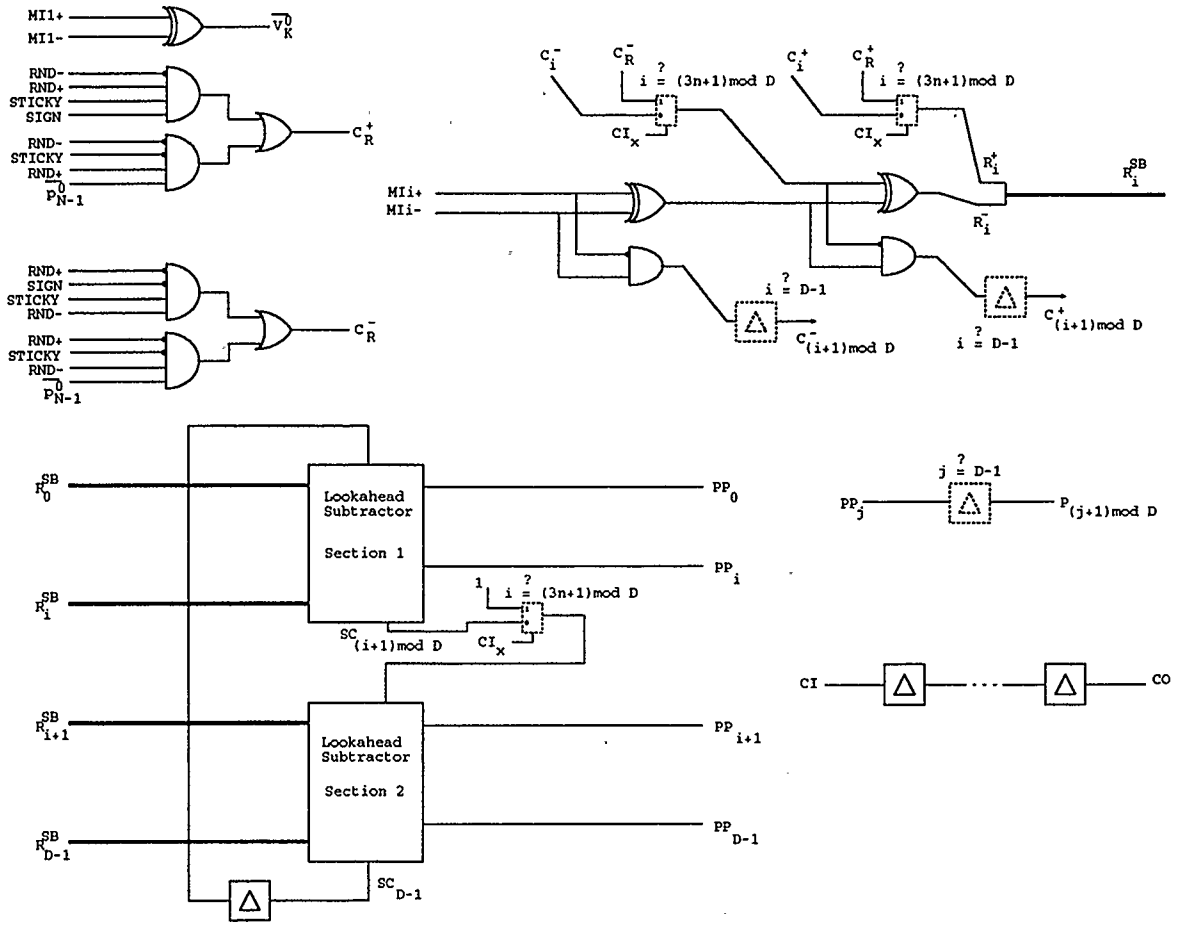

Figure 3.6. Round and Convert Module for Generalized Values of D, n, and i.

per gate available in the given technology, and the position of the multiplexer as shown in the lookahead section of Fig. 3.6. The least significant bit of P_i^{TC} is synchronized with the CO signal.

3.2.4 Re-pipelining

Digit-serial structures are generated by systematic unfolding [22] of the corresponding bit-serial structures. However, the unfolding process also increases the critical path lengths by spreading them over the digit-size.

The unfolding of conventional fully TC bit-serial multipliers to result in the corresponding TC digit-serial multipliers gives rise to two critical paths as shown in the

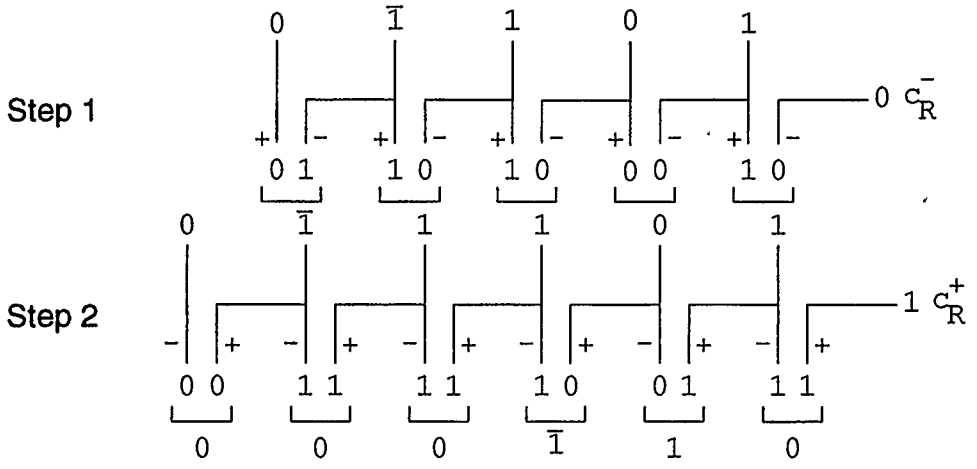

Figure 3.7. Principle Underlying the SB Rounding (An Example).

Table 3.2. Step 1 of SBNR Correction

Incoming SB − digit	*Recoded set*
0	0, 0
1	0, 1
$\bar{1}$	$\bar{1}$, 1

schematic in Fig. 3.8. The first critical path is the carry-propagation path which is shown by vertical dotted lines. This is eliminated in the proposed mixed SB/TC digit-serial modified-Booth multipliers by utilizing the carry-free addition/subtraction property of mixed SB/TC number arithmetic. The second critical path is the partial-product sum computation path which is shown by horizontal heavy lines. This path arises due to the spreading of the bit-level pipeline over the digit-size after unfolding. Re-pipelining is employed to break these horizontal propagation paths (achieved by inserting latches judiciously at certain intervals between modules). The required re-pipelining interval is determined in terms of the desired speed, look-ahead circuit speed, chip area, and permissible latency.

Table 3.3. Step 2 of SBNR Correction

Incoming SB − digit	*Recoded set*
0	0, 0
$\bar{1}$	0, $\bar{1}$
1	1, $\bar{1}$

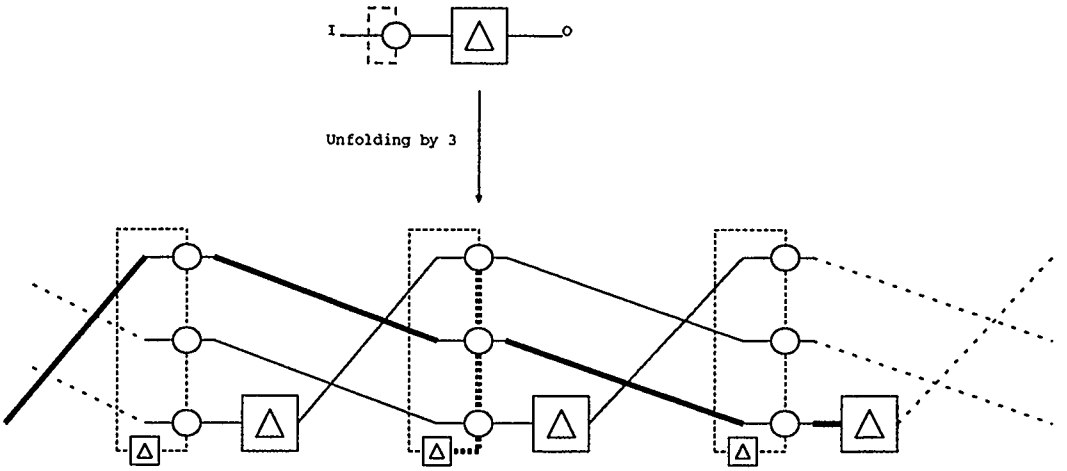

Figure 3.8. Critical Paths arising from Bit-Serial to Digit-Serial Unfolding.

The optimal re-pipelining interval θ required to maximize the efficiency of the multipliers can be computed as

$$\theta = max[\lfloor (T - t_0)/t_{T1} \rfloor, \theta_{MIN}], \tag{3.16}$$

where

$$t_0 = 3t_G + t_{DFF} + 2t_{MUX}, \tag{3.17}$$

and where t_{T1}, t_G, t_{DFF}, and t_{MUX} represent the delays through a T1-adder, gate, D-flip-flop (DFF), and multiplexer (MUX), respectively. Moreover, θ_{MIN} represents the minimum re-pipelining interval (which is equal to 1 in the present discussion), and T represents the minimum allowable clock period (c.f. Eqn. (3.19)).

3.2.5 Performance Analysis

In this section, the proposed digit-serial multipliers are first parameterized in terms of their hardware and time requirements. Subsequently, their throughputs and efficiencies are compared with those of the existing digit-serial modified-Booth multipliers both for various wordlengths and for various digit sizes.

3.2.5.1 Parameterization

Hardware Requirements

The hardware requirements for the digit-serial multipliers are calculated in terms of the number of DFFs, T1-adders, two-input MUXs, and auxiliary gates. The results are as shown in Table 3.4, where

$$\beta = \begin{cases} 1 & \text{if } (\lceil N/2 \rceil + 1) mod D = 0, 1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.18}$$

where F represents the maximum number of inputs-per-gate available in the target technology, and where l represents the look-ahead factor such that $0 < l < 1$.

Table 3.4. Area Requirements

Type of Hardware Cell	*Requirement*
T1-Adders	$D(N/2+1)$
2-input Multiplexors	$\lceil (12-4\lfloor 1/D \rfloor)(N/2-1) \rceil + 10$
D-Flip-Flops	$14(N/2)+12+\lfloor (3-D)N/(2D) \rfloor + (6D+2)(\lceil (N/(2\times\theta)) \rceil - 1) + \beta - \sum_{n=0}^{N/2-1} \lceil (3n+1) mod D/D \rceil$
AND, OR and XOR gates	$(4D + 3 - \lfloor 1/D \rfloor)N/2 + 10 + 5D + \lceil D/(F-1) \rceil + 2\lceil Dl/(F-1) \rceil + 2\sum_{i=0}^{(Dl-1)/2} \lceil (i+1)/(F-1) \rceil + 2\sum_{i=0}^{(D-Dl-1)/2} \lceil (i+1)/(F-1) \rceil + \sum_{i=0}^{\lceil (Dl-1)/2 \rceil} \sum_{k=0}^{i} \lceil (k+1)/(F-1) \rceil + \sum_{i=0}^{\lceil (D-Dl-1)/2 \rceil} \sum_{k=0}^{i} \lceil (k+1)/(F-1) \rceil$

Minimum Allowed Clock Period

The minimum allowed clock period T can be calculated through a worst-case analysis

(involving the remaining critical path) in accordance with

$$T = (\lceil log_F(Dl+1) \rceil + \lceil log_F(D-Dl+1) \rceil) t_G + 4t_G + t_{MUX} + t_{DFF}, \tag{3.19}$$

if θ is to be chosen as given in Eqn. (3.16). However, if θ is chosen otherwise, the minimum allowed clock period is given by T' in accordance with

$$T' = max[\, T,\ (t_0 + \lceil \frac{D}{\theta} \rceil t_{T1}) \,]. \tag{3.20}$$

Computational Delay

The computational delay of the proposed multipliers is given by $\lfloor (3N)/(2D) \rfloor + \lceil \frac{N}{2\theta} \rceil + 1$ bit-clock periods. Here, the computational delay refers to the time interval between the arrival of the least significant bit of the multiplicand (starting with CI to the first module being asserted) and the departure of the most-significant bit of the LSWO (i.e the least-significant bit of the rounded and converted product), starting with CO having been asserted.

3.2.5.2 Comparison with Existing Digit-Serial Multipliers

Fig. 3.9 shows the maximum possible throughput

$$H = D/(MT) \tag{3.21}$$

for various values of the multiplicand wordlength M and the digit-sizes D with $F = 4$, $l = 0.5$, and $M = N$. Here, $F = 4$ represents the maximum available inputs-per-gate for the Actel 1.2μ technology, and $l = 0.5$ represents the average degree of lookahead possible for the various values of D and n.

From Fig. 3.9, it can be seen that the throughputs of the proposed modified-Booth multipliers increase linearly with increasing the digit size. This is due to the totally carry/borrow-free nature of the partial product sum computation, the look-ahead nature of the final SB to TC conversion, and the re-pipelining employed in

the partial product sum formation. It can also be observed that the throughputs of the proposed multipliers (solid curves) are more than 2 to 3 times higher than those of the modified-Booth multipliers in [15] (dashed curves). In this way, the former multipliers are suitable for DSP applications requiring very high processing speeds.

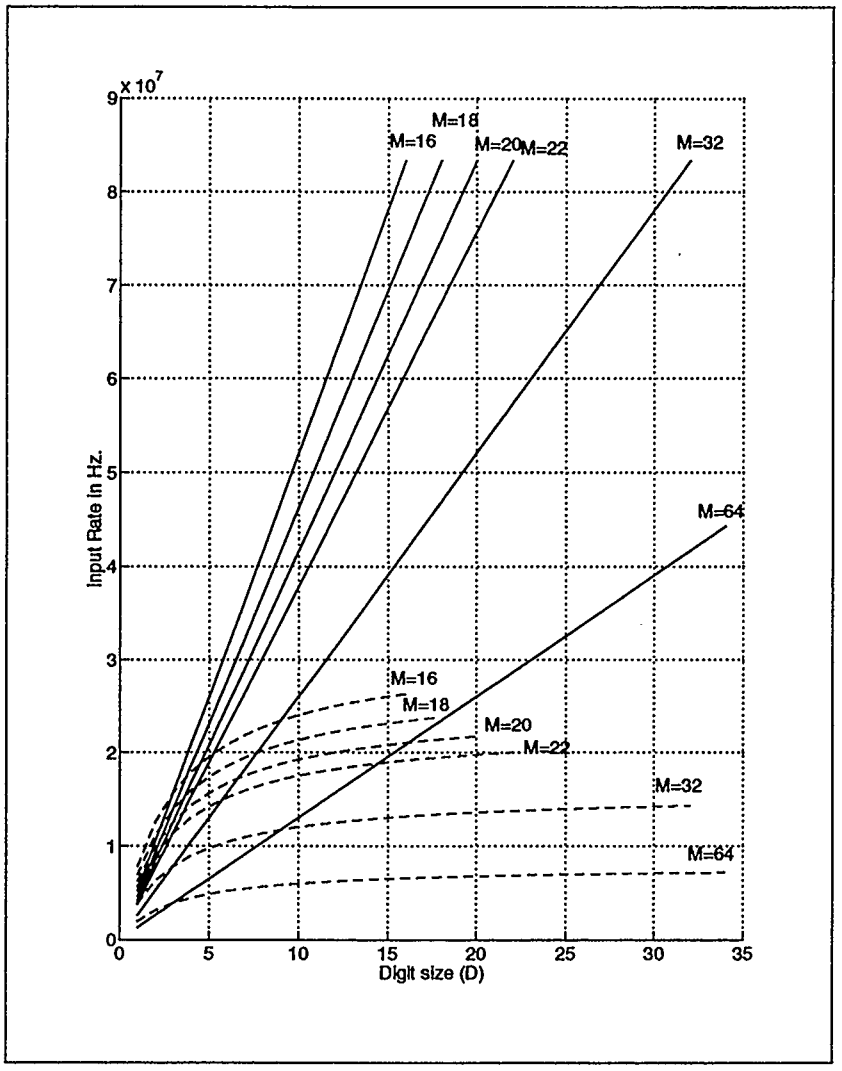

Figure 3.9. Throughput for $F = 4$, and $l = 0.5$, for the proposed multipliers (solid lines) and the multipliers in [15] (dashed lines)

Fig. 3.10 shows the hardware area requirement of the proposed multipliers (solid

lines) for various values of the multiplicand wordlength M and the digit-sizes D with $F = 4$, $l = 0.5$, and $M = N$. A comparison with the corresponding hardware area requirements of the modified-Booth digit-serial multipliers in [15] (dashed lines) shows that the proposed multipliers require substantially higher hardware area. The hardware area requirement for the proposed multipliers shows sharper increase with increasing digit-sizes compared to the existing multipliers due to the increased need of repipelining latches.

Fig. 3.11 shows the efficiency of the proposed multipliers for various values of the multiplicand wordlength M and the digit-sizes D with $F = 4$, $l = 0.5$, and $M = N$, where the efficiency is defined as throughput per unit area. By comparing the efficiencies for the novel modified-Booth multipliers (solid lines) with the corresponding efficiencies for the modified-Booth digit-serial multipliers in [15] (dashed lines), it can be observed that the efficiencies of the former multipliers increase monotonically with increasing the digit-size, whereas those of the latter decrease after a digit-size of 4. This indicates that the novel multipliers are most suitable for digit sizes greater than 4. Moreover, the efficiencies of the novel multipliers reach their maxima at $D = M$, indicating that *the most area-time efficient radix-2 mixed SB/TC multiplier is the fully parallel multiplier.*

In summary, once the digit-size D and the multiplicand and multiplier wordlengths M and N are selected, one can use the design parameters F and θ to arrive at the desired throughput and efficiency. This is in contrast to the existing digit-serial modified-Booth multipliers where the throughput and efficiency are fixed given D, M, and N. Moreover, most DSP algorithms require $N < M$, in which case there is marked reduction in the number of re-pipelining latches required in the proposed multipliers. This is predicted to result in a substantial reduction in the required area for a fixed achievable throughput, leading to a corresponding increase in the efficiency. The above predicted reduction is due to the strong dependence of the area

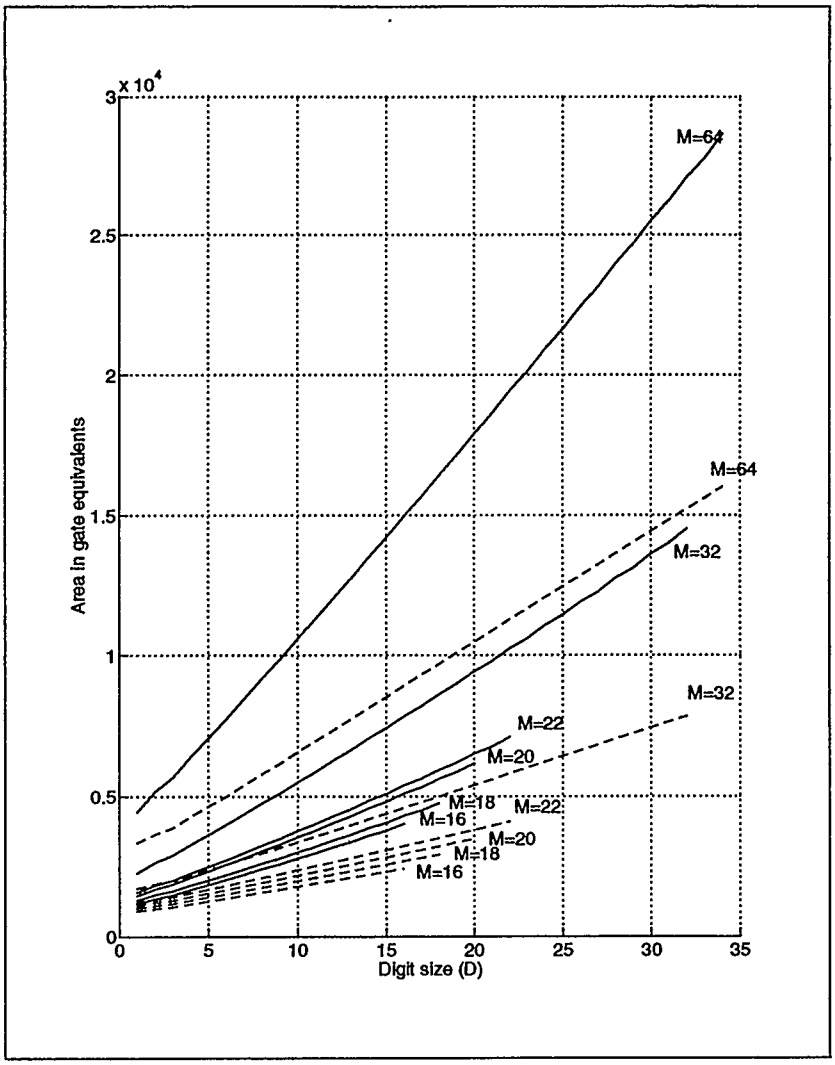

Figure 3.10. Hardware Area Requirements for $F = 4$, and $l = 0.5$, for the proposed multipliers (solid lines) and the multipliers in [15] (dashed lines)

requirements of the proposed multipliers on the value of N.

3.2.6 Verification

The proposed digit-serial modified-Booth multipliers are verified through Viewlogic simulations using the Actel 1.2μ technology. The simulation results for a digit-size of 2 (3) are as shown in Fig. 3.12 (3.13), with the corresponding test vectors being as

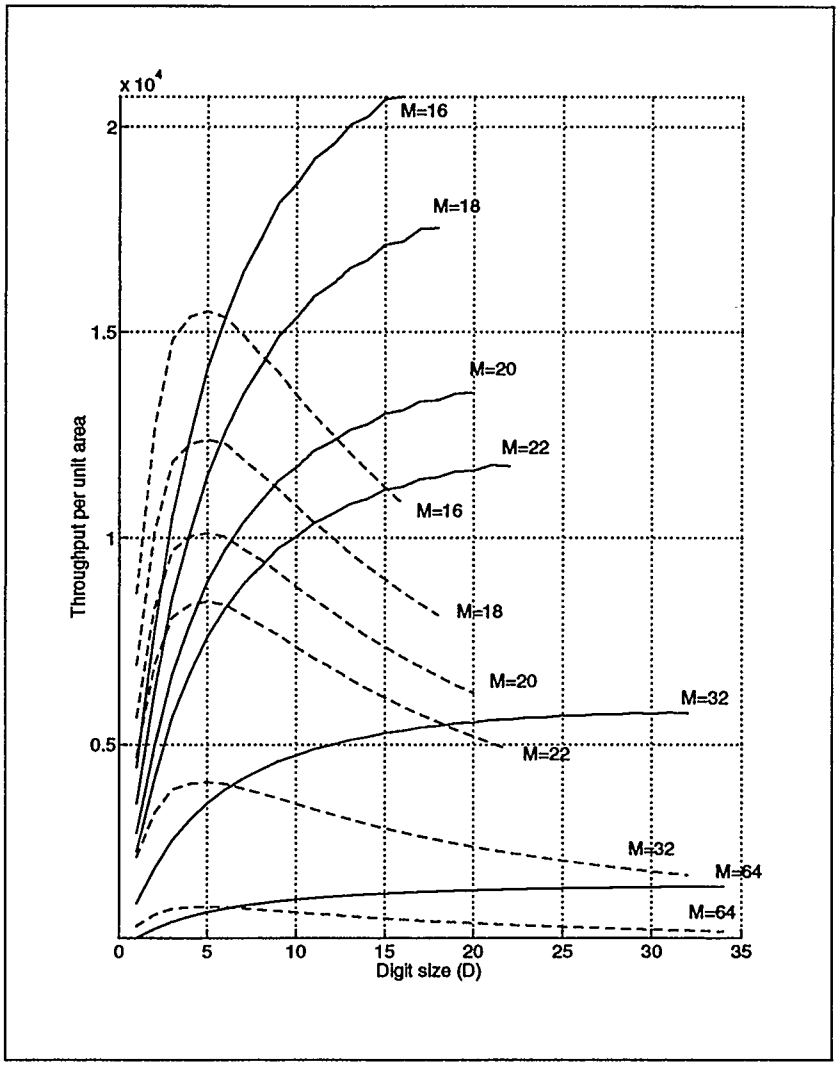

Figure 3.11. Efficiency for $F = 4$ and $l = 0.5$, for the proposed multipliers (solid lines) and the multipliers in [15] (dashed lines)

given in Table 3.5.

In Figs. 3.12 and 3.13, $XI[. : 0]$ and $YI[. : 0]$ represent the decomposed components of the multiplicand and multiplier, respectively, and $PO[. : 0]$ represents the decomposed components of the final TC product. Moreover, CI represents the input control signal with which the LSBs of the multiplicand and multiplier components are aligned. In addition, SIGN and STICKY bits are indicated by $SIGN$ and

Table 3.5. Verification Test Vectors

Test Vector	X^{TC}	Y^{TC}	*IEEE Rounded Product (TC)*
1	0.0011010	0.1010110	0.0010001
2	1.1101001	0.1010110	1.1110001
3	0.0011010	1.0110111	1.1110001
4	1.1101001	1.0110111	0.0001101
5	0.0011000	0.0011000	0.0000100

SKY, respectively. Finally, $MO[. : 0]$ indicate the SB-MSW components in their non-rounded form and RND indicates the SB round-digit. $RO[. : 0]$ indicate the rounded SB-components. The LSB of $PO[. : 0]$ is synchronized with the rising edge of the CO-signal. All the SB-signals are indicated in their two-bit bus format using the maximal (n, p)-encoding. Therefore, the bus values of 3, 2, 1, and 0 indicate the SB-digit values of 0, $\bar{1}$, 1 and 0, respectively. Note that the above multipliers have been built with $\theta = 3$.

The functionality of the above multipliers is verified by comparing the simulation results in Fig. 3.12 and Fig. 3.13 with the expected results in Table 3.5. Further, the maximum operational speed and the computational delays of these multipliers have also been verified to be in agreement with the theoretically calculated values. Details regarding the architecture and operation of the above multipliers for $D = 2$ and $D = 3$ are available in [46].

3.3 High-Speed Mixed SB/TC Parallel Modified-Booth MAC Arithmetic Architecture

The operation of many DSP algorithms is based on repetitive accumulation of independently formed multiplication products. In such algorithms, the multiplication and accumulation operations can be combined naturally into a single indivisible

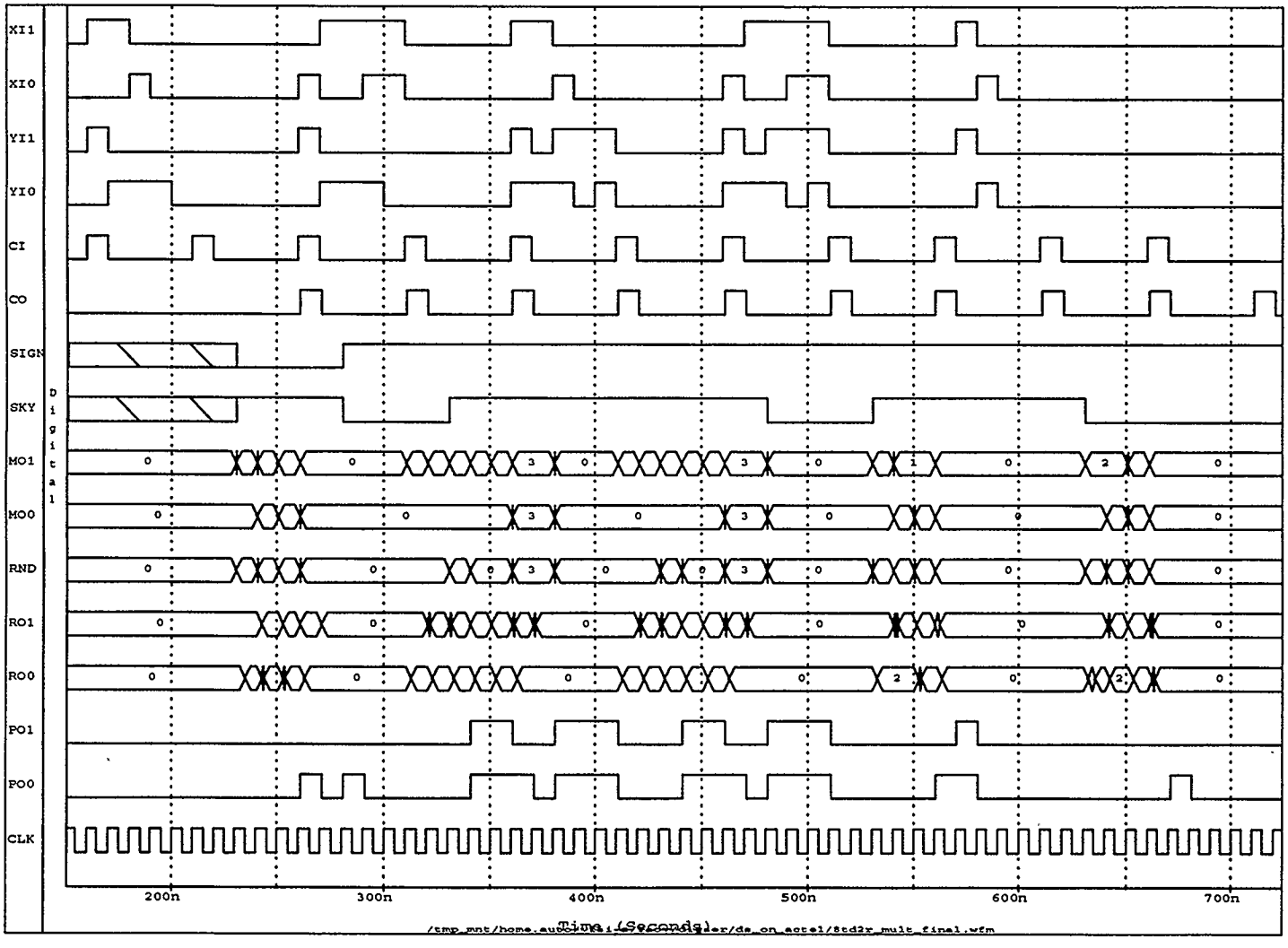

Figure 3.12. Viewlogic Simulation Results for $D = 2$

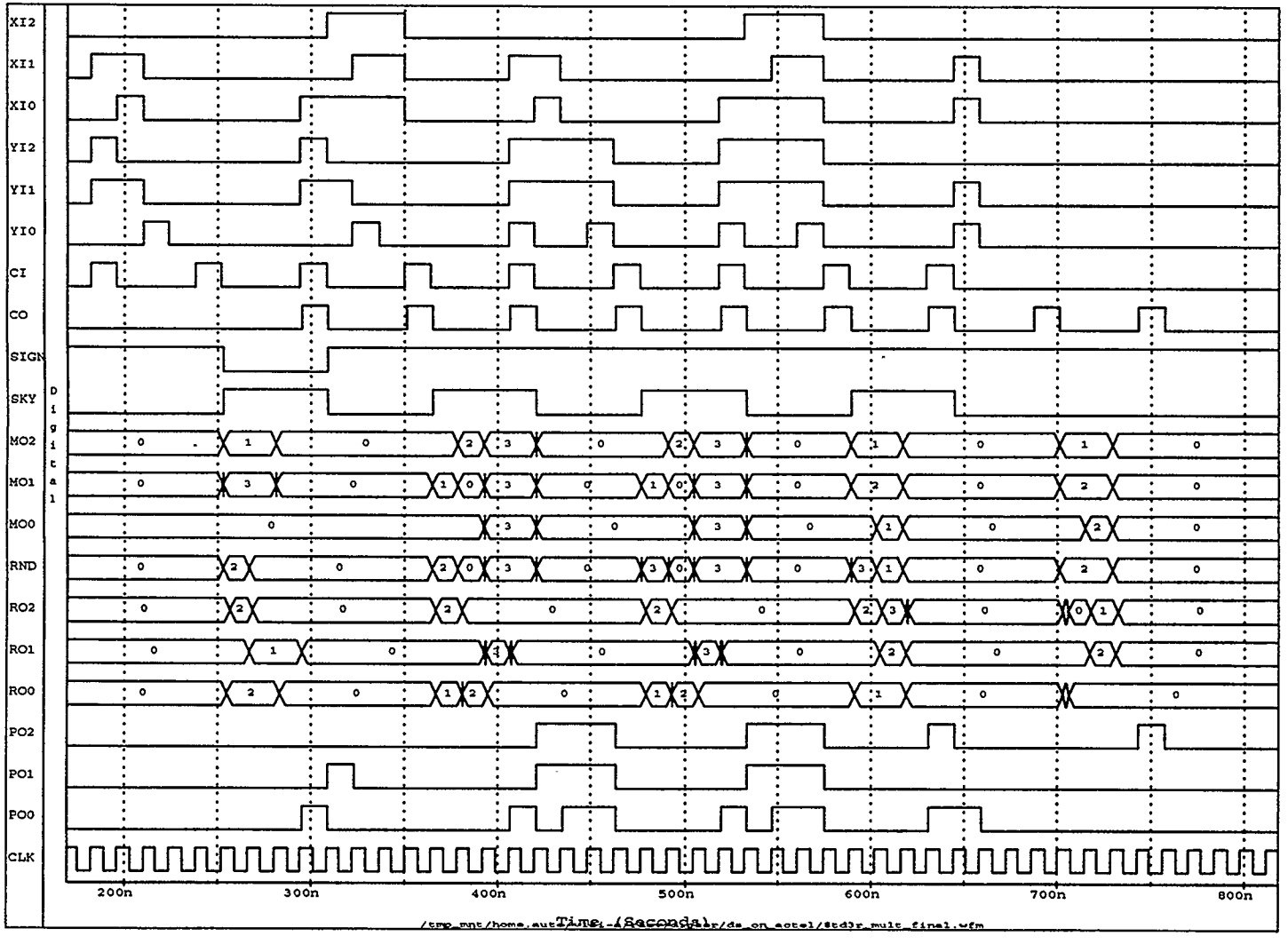

Figure 3.13. Viewlogic Simulation Results for $D = 3$

MAC [47] operation. This operation permits finite-precision arithmetic architectures which are less susceptible to the harmful effects of roundoff noise. The resulting architectures lead to reduced chip bussing [40] and increased speed of computation in the corresponding VLSI implementations. Therefore, designing fast MAC arithmetic architectures is of key theoretical and practical importance on the part of computer scientists and engineers.

The existing DSP systems usually employ TC number arithmetic [20]. The problems associated with the use of TC number arithmetic have already been discussed. These problems result in considerable reduction in the computational speed of DSP systems having large wordlengths. Redundant number arithmetic, on the other hand, features totally carry-free addition and subtraction, permitting the corresponding architectures to be clocked at very high speeds. However, a design based on fully redundant number arithmetic is expensive in terms of VLSI chip area. This is the reason why redundant number arithmetic is used only in the intermediate processing stages in TC arithmetic functional units [4].

In [34], redundant number arithmetic was incorporated in the modified-Booth algorithm [36] for the computation of the intermediate partial product sums, leading to the design of a high-speed bit-parallel TC digital multiplier. However, this technique requires the use of fully redundant number digital adders. Such adders are more expensive in terms of chip area than the conventional full-adders.

Mixed SB/TC number arithmetic has shown a lot of promise for high-speed multiplication using minimum VLSI chip area. In the previous section, it was shown that the most area-time efficient mixed SB/TC modified-Booth multipliers are the parallel multipliers. In this section, mixed SB/TC arithmetic is exploited together with the SB RNE technique (c.f. Algorithm 3 (Page 39)) for the development of a high-speed mixed SB/TC MAC arithmetic architecture [50].

3.3.1 High-Speed Mixed SB/TC Parallel Modified-Booth MAC Operation

Consider a MAC operation involving a M-bit TC multiplicand X^{TC}, a N-bit TC multiplier Y^{TC}, and a $M+N-1$-digit SB addend A^{SB}, in accordance with

$$P^{SB} = X^{TC}.Y^{TC} + A^{SB}, \tag{3.22}$$

where X^{TC}, Y^{TC}, and A^{SB} are given by

$$\begin{aligned} X^{TC} &= -x_{M-1}2^{M-1} + \sum_{i=0}^{M-2} x_i 2^i \quad x_i \in \{0,1\}, \\ Y^{TC} &= -y_{M-1}2^{N-1} + \sum_{i=0}^{N-2} y_i 2^i \quad y_i \in \{0,1\}, \end{aligned} \tag{3.23}$$

and

$$A^{SB} = \sum_{i=0}^{M+N-2} a_i 2^i \quad a_i \in \{\bar{1},0,1\}. \tag{3.24}$$

Furthermore, let A^{SB} be partitioned into its most-significant word (MSW) A^{SB}_{MSW} and least-significant word (LSW) A^{SB}_{LSW} in accordance with

$$A^{SB}_{MSW} = \sum_{i=N-1}^{M+N-2} a_i 2^i \tag{3.25}$$

and

$$A^{SB}_{LSW} = \sum_{i=0}^{N-2} a_i 2^i. \tag{3.26}$$

In the MAC arithmetic operation, the modified-Booth recoding is applied to the multiplier Y^{TC} by an overlapped scan on triplets of bits y_{2n+1}, y_{2n}, and y_{2n-1} (for $n \in \{0,1,\ldots,\lceil \frac{N}{2} \rceil - 1\}$, with $y_{-1} = 0$), to obtain the full-precision MAC result P^{SB} as

$$P^{SB} = \sum_{n=0}^{\lceil \frac{N}{2} \rceil - 1} z_n X^{TC} 4^n + A^{SB} = \sum_{i=0}^{M+N-2} p_i 2^i \quad p_i \in \{\bar{1},0,1\}, \tag{3.27}$$

where z_n is computed in accordance with Eqn. (3.1).

The mixed SB/TC parallel MAC arithmetic operation is initiated by applying the modified-Booth recoding technique to the multiplier Y^{TC} to generate the recoded digits z_n. These digits are used to compute the intermediate partial product components $z_n X^{TC} 4^n$ in TC format, while computing the intermediate partial product sums P_n^{SB} in their SB formats by using the recursive relationship

$$P_n^{SB} = z_n X^{TC} 4^n + P_{n-1}^{SB}, \tag{3.28}$$

where $P_{-1}^{SB} = A_{LSW}^{SB}$, $P_{n<-1}^{SB} = 0$, and where $P_{n\geq 0}^{SB}$ is given by

$$P_{n\geq 0}^{SB} = \sum_{j=0}^{n} z_j X^{TC} 4^j + A_{LSW}^{SB}. \tag{3.29}$$

It is interesting to note that the generation of the intermediate partial product components $z_n X^{TC} 4^n$ need not entail any loss of computational time as these components can be obtained as hard-wired shifted versions of the multiplicand X^{TC}. It is also interesting to note that the the intermediate partial product sum components P_{n-1}^{SB} do not require any sign-extension as they are already in their SB formats. Therefore, it suffices to use, instead, zero insertion at the most significant part of P_{n-1}^{SB}, simplifying the corresponding implementation.

The recursion in Eqn. (3.28) proceeds for successive values of $n = \{0, 1, \ldots, \lceil \frac{N}{2} \rceil - 1\}$, to generate the resulting SB-product $P_{\lceil \frac{N}{2} \rceil - 1}^{SB} = P^{SB} - A_{MSW}^{SB}$. This SB-product is then decomposed into its constituent MSW $P_{MSW}^{SB'}$ and LSW P_{LSW}^{SB}. Subsequently, $P_{MSW}^{SB'}$, A_{MSW}^{SB}, and c_R are added concurrently using a radix-2 fully-redundant addition in accordance with

$$P_{RNE}^{SB} = P_{MSW}^{SB'} + A_{MSW}^{SB} + c_R, \tag{3.30}$$

where c_R represents an IEEE Standard 754 rounding correction. The rounded result P_{RNE}^{SB} is converted to its TC format by using

$$P_{RNE}^{TC} = |P_{RNE}^{SB+}| - |P_{RNE}^{SB-}|, \tag{3.31}$$

where P_{RNE}^{TC} constitutes the final MAC result.

In the case of DSP algorithms involving repetitive MAC arithmetic operations, the full-precision SB results obtained from the intermediate (i.e. all but the last) MAC operation(s) serve as the full-precision addends for the subsequent operations. Therefore, $c_R = 0$ for the intermediate MAC operations, while $c_R \in \{0, 1, \bar{1}\}$ for the last operation. The last operation also involves the conversion of the overall SB result to its TC format in accordance with Eqn. (3.31).

3.3.2 Mixed SB/TC Parallel MAC Arithmetic Architecture

In this section, the theoretical results in Section 3.3.1 are exploited for the development of an architecture for high-speed MAC arithmetic operation. The resulting MAC architecture is as shown in the schematic diagram in Fig. 3.14.

In the above MAC arithmetic architecture, the TC multiplier Y is fed to a bank of *Modified-Booth Recoder* modules. These recoder modules convert the multiplier Y into its corresponding modified radix-4 redundant number representation [49]. The TC multiplicand X is fed to a bank of *Modified-Booth Decoder* modules within the MAC arithmetic functional unit kernel. The modified-Booth decoded outputs are fed in their TC formats to the *Mixed SB/TC Adder* rows. The first row of mixed SB/TC adders has the LSW of the result accumulated in the previous clock cycle (A_{LSW}^{SB}) as its input. The full-precision result P^{SB} is generated from the least significant digit p_0, at the top-right corner, to the most significant digit p_{M+N-2}, at the bottom-left corner of the MAC kernel. Each mixed SB/TC adder row generates the intermediate partial product sums P_n^{SB} which contain the two digits p_{2n+1} and p_{2n} belonging to P_{LSW}^{SB}. These digits are consumed by the *SIGN and STICKY Generation* module to generate the indicators $SIGN$ and $STICKY$. These indicators are fed to the *Correction Logic* module in order to generate the rounding correction c_R. The $(\lceil \frac{N}{2} \rceil - 1)$-th mixed SB/TC adder row generates $P_{MSW}^{SB'}$ which is fed to the *Rounding, MSW Addition,*

and Overflow Correction module, to generate the IEEE Standard 754 rounded result P_{RNE}^{SB}. The rounding operation for P_{RNE}^{SB} is controlled by the *RND* signal. The result P_{RNE}^{SB} is subsequently accumulated by the *MSW Accumulator* while the LSW P_{LSW}^{SB} is accumulated by the *LSW Accumulator*. The content of the MSW Accumulator is subsequently converted to its TC format P_{RNE}^{TC} by the *CLA Conversion* module.

In the proposed architecture, the overflow correction, rounding, and MSW-addition all occur concurrently. This is because redundant number arithmetic permits the addition operations at the most-significant part of the MSW-adder to be decoupled from those at the least-significant part of the MSW-adder. The overflow correction is applied at the most-significant part, while the rounding correction is applied at the least-significant part of MSW, to facilitate concurrency in computation. Additional information regarding the various modules in the architecture in Fig. 3.14 is given in the following subsections.

3.3.2.1 Modified-Booth Recoder Modules

The modified-Booth recoders carry out the 3-bit overlapped scan of the TC multiplier Y to generate z_n in Eqn. (3.27) in terms of a_i, b_i, and s_i in accordance with Eqns. (3.3), (3.4), and (3.5).

3.3.2.2 Modified-Booth Decoder Modules

The modified-Booth decoders generate 0, ± 1 and ± 2 times the multiplicand X depending on the output of the modified-Booth recoders at the corresponding row position. The modified-Booth decoders in the n-th row generate the partial product components

$$z_n X 4^n = (o_N 2^N + \sum_{i=0}^{N-1} o_i 2^i) 4^n, \tag{3.32}$$

where the decoded components o_i are determined in accordance with

$$o_i = s_i \bigoplus (a_i x_{i-1} + b_i x_i), \tag{3.33}$$

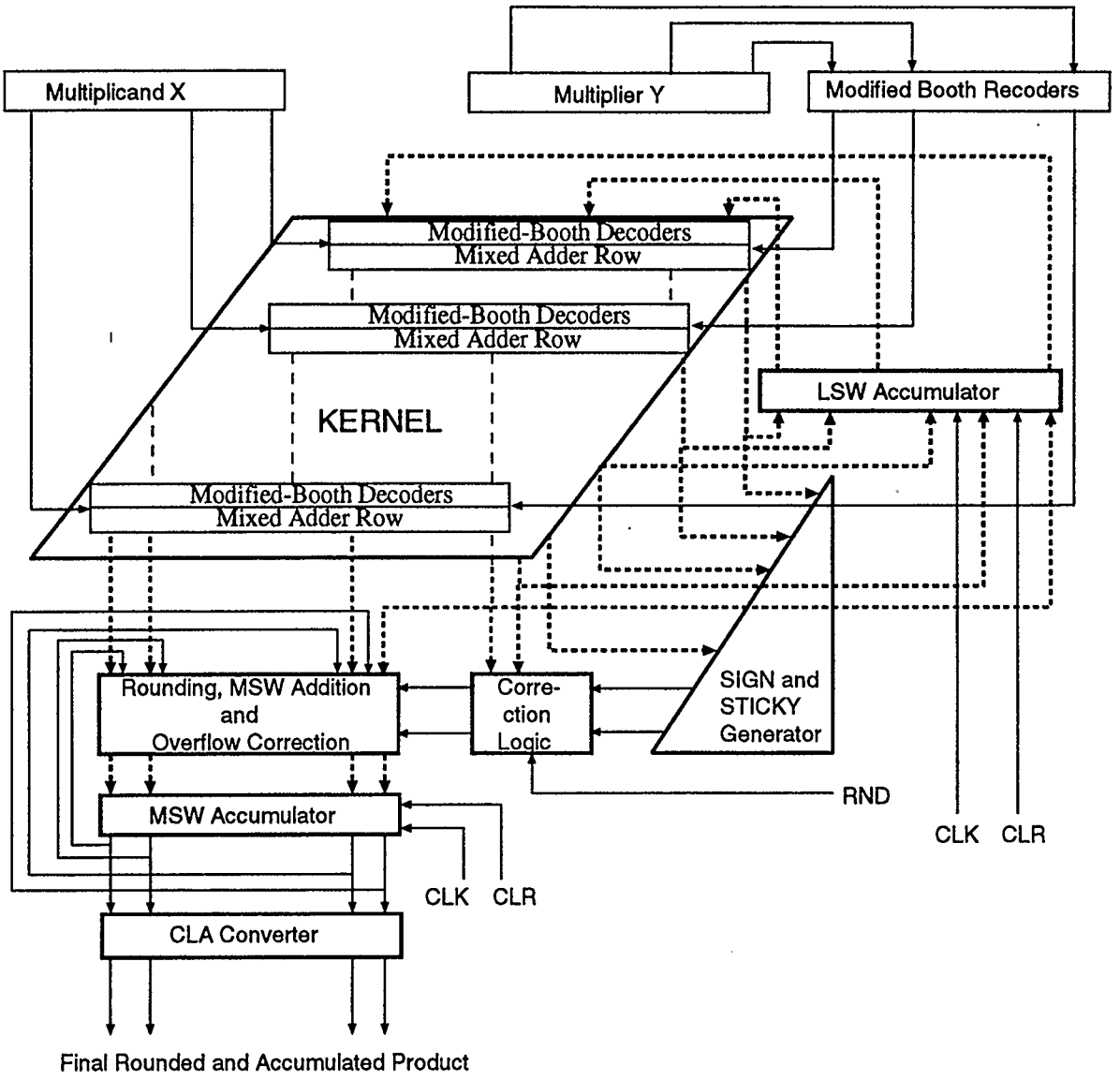

Figure 3.14. The Mixed SB/TC Parallel MAC Arithmetic Architecture

where $o_N = x_{N-1}$, and where $x_{-1} = 0$. Moreover, s_i is fed as a carry to the corresponding n-th mixed SB/TC adder row.

3.3.2.3 Mixed SB/TC adder row

The terms P_n^{SB} are generated by the mixed SB/TC adder for the n-th row.

3.3.2.4 *SIGN* and *STICKY* Generation Module

As mentioned above, the MAC kernel generates the least significant digits two at a time. In this way, the *SIGN and STICKY Generator* computes the $SIGN$ and $STICKY$ indicators by considering two least significant digits at a time. Each two-digit computation cell consumes $sticky_{in}$, $sign_{in}$, the i^{th} least significant digit p_i, and the $i-1^{th}$ least significant digit p_{i-1} as its inputs, and generates $sticky_{out}$ and $sign_{out}$ in accordance with

$$sticky_{out} = p_{i-1}^{v} + p_i^{v}.sticky_{in}, \tag{3.34}$$

$$sign_{out} = p_i^{s} + p_{i-1}^{s}.\overline{p_i^{v}} + sign_{in}.\overline{p_{i-1}^{v}}.\overline{p_i^{v}}. \tag{3.35}$$

In this way, the output of the last (leftmost) computation cell corresponds to $STICKY$ and $SIGN$.

It can be observed from Eqns. (3.34) and (3.35) that the time required to compute $sticky_{out}$ and $sign_{out}$ for each pair of least significant digits is less than the time required for the mixed SB/TC addition. This means that for the (s, v) representation, $sticky_{out}$ and $sign_{out}$ can be computed faster than the least significant digit generation process, and consequently this operation can be carried out concurrently with the partial product addition operation. In contrast, the (n, p) representation requires longer time and a greater number of gates for such a computation. Therefore, the (s, v) representation is particularly attractive on the part of the *SIGN and STICKY Generator*. Moreover, it can be observed that the generation of the $STICKY$ indicator is much faster and easier in this case as compared to the cases of the traditional multipliers.

3.3.2.5 Correction Logic Module

The correction logic examines the *round* digit p_{N-2}, the least significant digit of the MSW p_{N-1}, $SIGN$, and $STICKY$ and generates the correction information

required for proper rounding of the SB MSW. The RND signal enables rounding when high (logic 1), and disables rounding when low (logic 0). During intermediate repetitive MAC operations in a DSP algorithm, RND is held low to permit full-precision accumulation. During the last MAC operation in the algorithm, RND is pulled high in order to round the SB result. Rounding is achieved by generating c_R through the use of p_{N-1} and p_N in accordance with

$$c_R^+ = \overline{p_{N-2}^s}.p_{N-2}^v.p_{N-1}^v.\overline{STICKY} + \overline{p_{N-2}^s}.p_{N-2}^v.STICKY.\overline{SIGN} \tag{3.36}$$

and

$$c_R^- = p_{N-2}^s.p_{N-2}^v.p_{N-1}^v.\overline{STICKY} + p_{N-2}^s.p_{N-2}^v.STICKY.SIGN \tag{3.37}$$

(See also Eqns. (2.102) and (2.103)).

3.3.2.6 Rounding, MSW-addition, and Overflow Correction Module

This module carries out the rounding and overflow correction of the result of the addition of P_{MSW}^{SB} to A_{MSW}^{SB}. The central part of this module is a fully redundant MSW adder [42] which adds $P_{MSW}^{SB'}$ to A_{MSW}^{SB}. However, SBNR addition in fixed-point fractional arithmetic DSP-systems can result in a *directly correctable overflow* [49], requiring overflow correction immediately after the addition. An overflow occurs if the digit p_{N+M-1} generated by the MSW adder is non-zero. The overflow is always *directly correctable* if the absolute value of the final result is less than one. This occurs if $p_{N+M-1}^s \neq p_{N+M-2}^s$ and $|p_{N+M-1}| = |p_{N+M-2}| \neq 0$. The sign and value components of the most-significant digit of the final result after overflow correction are given by

$$msd_{corrected}^s = p_{N+M-1}^s + \overline{p_{N+M-1}^v}.p_{N+M-2}^s, \tag{3.38}$$

$$msd_{corrected}^v = p_{N+M-1}^v + \overline{p_{N+M-1}^v}.p_{N+M-2}^v. \tag{3.39}$$

The computation of $msd_{corrected}^s$ and $msd_{corrected}^v$ occurs concurrently with the application of the rounding correction to the MSW, permitting very high-speed operation

in the resulting MAC arithmetic implementations.

3.3.2.7 Accumulator Module

This module accumulates the $N + M - 1$-digit result. It is divided into two parts, namely the *MSW Accumulator* and the *LSW Accumulator*. In this module, the p_N-th digit is required for the MSW Accumulator because it forms a part of P_{SB}^{MSW} and is also required for the LSW Accumulator because it forms a part of P_{SB}^{LSW} (the latter is in turn used to compute the rounding correction c_R).

The MSW Accumulator and LSW Accumulator are both clocked by using the CLK signal and can be cleared (initialized) by using the CLR signal.

3.3.2.8 CLA Conversion Module

The *CLA* module is a high-speed carry-lookahead converter [20], used for converting the final rounded SB result into its TC form. This module is separated from the MAC-kernel by the *MSW Accumulator*, thereby creating a pipeline facilitating high-speed operation.

3.3.3 Performance Characteristics

In this section, the performance characteristics of the MAC arithmetic architecture in Fig. 3.14 are discussed in terms of the hardware area and computational time requirements. These requirements are parameterized at the gate-level for corresponding ASIC implementations.

3.3.3.1 Hardware Area Requirement

The hardware area requirement of the above arithmetic architecture in terms of gate-equivalents is as shown in Table 3.6. Here, n_j refers to the number of digits input to the j^{th} CLA block, nlb is the number of lookahead blocks in the converter, F

Table 3.6. Hardware Area Requirements

Type of Hardware Cell	*Requirement*	*Gate Equivalents per Cell*
Modified-Booth Recoders	$\lceil \frac{N}{2} \rceil$	6
Modified-Booth Decoders	$M\lceil \frac{N}{2} \rceil$	4
Mixed Adders	$(M+1)\lceil \frac{N}{2} \rceil$	5
SIGN and *STICKY* Generators	$\lceil \frac{N}{2} \rceil - 1$	5
Correction Generator	1	8
Overflow Corrector	1	4
Redundant Adders	$(M+1)$	10
CLA Converter	1	$\sum_{j=1}^{nlb}(5n_j +$ $\sum_{i=0}^{n_j-1}\lceil (i+1)/(F-1) \rceil +$ $\sum_{i=0}^{n_j-1}\sum_{k=0}^{i-1}\lceil (k+1)/(F-1) \rceil)$
DFFs with Reset	$2(M+N-1)$	6
Auxiliary Gates	1	$2\lceil \frac{N}{2} \rceil$

is the maximum number of inputs per gate permitted by the underlying technology, and M and N are the multiplicand and multiplier wordlengths, respectively.

3.3.3.2 Computational Time Requirement

The computational time requirement of the MAC arithmetic architecture is given in terms of the minimum achievable clock period T. By considering the worst case critical path in the architecture, T is obtained as

$$T = max[\ \{t_{DFF} + t_{RECODE} + t_{DECODE} + t_{CR} + \lceil \frac{N}{2} \rceil t_{MADD} + t_{RADD}\},$$
$$\{t_{DFF} + t_G(4 + \sum_{j=1}^{nlb}(\lceil log_F(n_j+1) \rceil + 1)\}]. \qquad (3.40)$$

In Eqn. (3.40), t_{RECODE} (t_{DECODE}) represents the time required for modified-Booth recoding (decoding), t_{CR} represents the time required for generating the rounding correction c_R, t_{MADD} (t_{RADD}) represents the time required by the mixed SB/TC (radix-2 fully-redundant) addition, t_{DFF} represents the delay through a D-Flipflop,

and t_G represents the delay through a logic gate.

3.3.4 Verification

An $8 \times 8 + 15$ parallel MAC arithmetic functional unit was designed for implementation using the Actel 1.2μ technology parameters. This arithmetic functional unit was simulated using the maximum achievable clock rate of 40 MHz for a corresponding implementation having a typical logic gate delay of 1 nanosecond. The Viewlogic simulation results shown in Fig. 3.15 depict the intermediate signals generated in the course of the MAC operations together with the last operation which also includes rounding. In Fig. 3.15, buses Y (X) represent the 8-bit TC multiplier (multiplicand). RND (CLR) represents the rounding (clear) signal, and CLK represents the system clock. Furthermore, $(MSWS, MSWV)$ represent the MSW $P_{MSW}^{SB'}$ in its minimal-(s, v) form. Finally, $RNDS$ and $RNDV$ represent the p_{N-1}^{th} and p_{N-2}^{th} digits, and $(LSWS, LSWV)$ represent the LSW P_{LSW}^{SB} of the result. The overall TC result after CLA conversion is given by ANS (which corresponds to P_{RNE}^{TC}).

The test vectors associated with the simulation results in Fig. 3.15 are given in Table 3.7 together with the expected full-precision MAC results, accordingly. The simulation begins with the MSW and LSW accumulators being cleared by setting $CLR = 1$. Subsequently CLR is set to zero and four successive multiplications are carried out with accumulation in full-precision. The RND signal is pulled up with the rising edge of the 4^{th} clock cycle to enable the IEEE standard 754 rounding of the multiplied and accumulated result collected during the first 4 clock cycles. This result is converted to its TC form by using the CLA converter and is available on the ANS bus during the 5^{th} clock cycle (made possible by the inherent pipelining in the architecture).

The simulation results in Fig. 3.15 are in complete agreement with the corresponding expected results in Table 3.7.

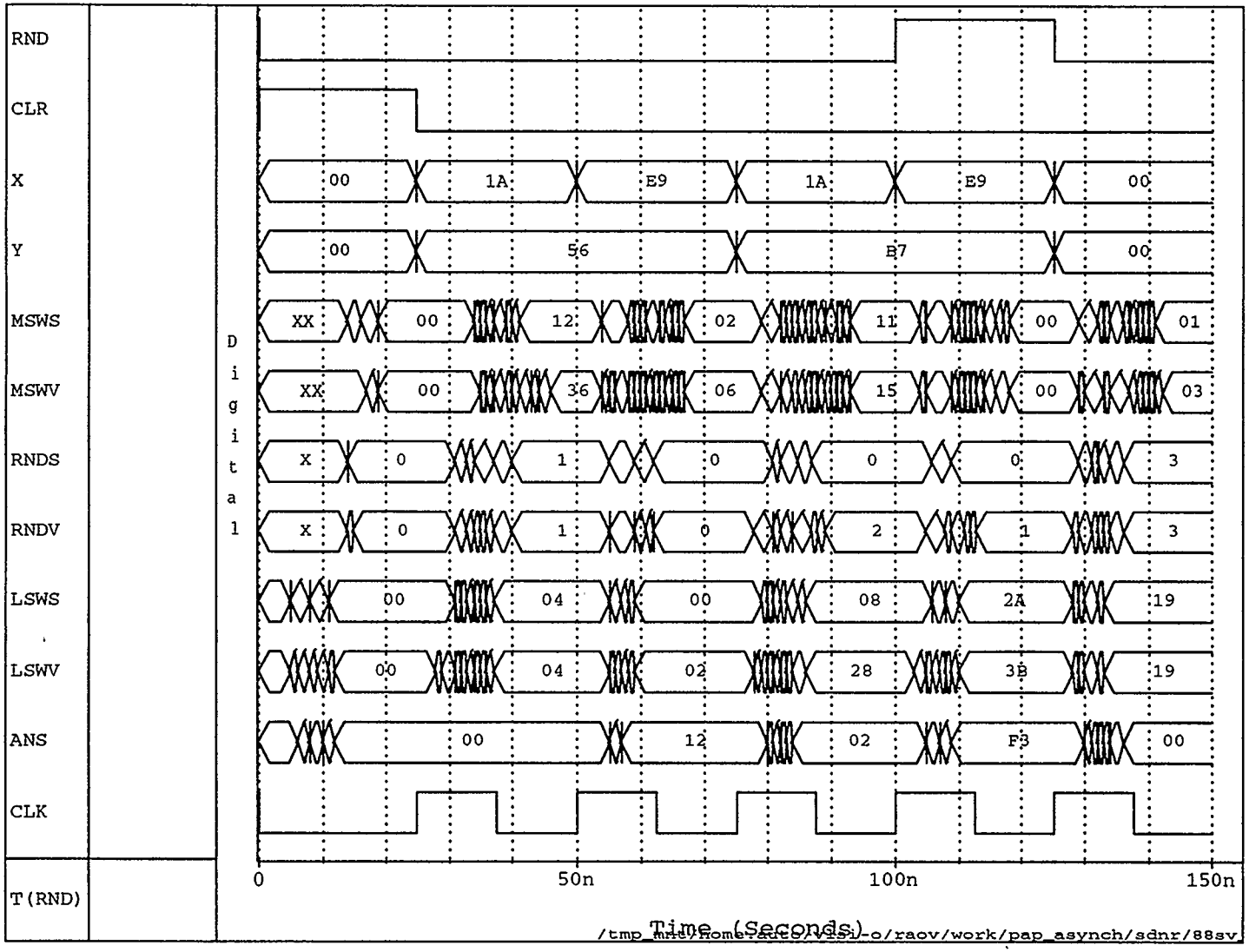

Figure 3.15. Viewlogic Simulation Results for the $8 \times 8 + 15$ Mixed SB/TC Parallel MAC Arithmetic Unit

Table 3.7. Series of MAC Arithmetic Operations for Verification

$CLKCycle$	Y^{TC}	X^{TC}	$MAC\ Result$
0	0.0000000	0.0000000	0.00000000000000
1	0.1010110	0.0011010	0.00100010111100
2	0.1010110	1.1101001	0.00000100000010
3	1.0110111	0.0011010	1.11100110011000
4	1.0110111	1.1101001	0.00000000100111

3.4 High-Speed Fully-SB Parallel MAC Arithmetic Architecture

The previous section dealt with the application of redundant number arithmetic to TC MAC architectures. However, the resulting architectures prove to be slow for certain time-critical DSP applications. This is due to the overhead in conversion of the final SB result to its corresponding TC form, becoming a major bottleneck for applications requiring large signal wordlengths.

This section presents a technique for high-speed parallel MAC operation based on fully-SB arithmetic [49]. In this technique, SBNR is employed throughout to represent the multiplier and the multiplicand, the intermediate partial products, and the final multiply-accumulated result.

The above technique combines the power of the 5-digit overlapped scanning technique (c.f. Section 2.4.1) and the carry-free addition property of SB numbers to achieve high multiplication speed at reduced hardware requirement. This is coupled with partitioned accumulation and concurrent high-performance rounding and overflow correction to result in an overall fast multiplication and accumulation. This resulting MAC architecture finds use in critical high-speed DSP applications.

The proposed MAC technique consists of three distinct functions. The first func-

tion involves a 5-digit overlapped scanning technique for the parallel recoding of the multiplier into a corresponding modified radix-4 redundant number representation (c.f. Algorithm 1 (Page 28)). The recoded multiplier is subsequently used to generate the intermediate partial products by uniform shifting, negation, or zeroing of the SB-multiplicand. The advantage of this recoding scheme is that it leads to a reduction in the number of intermediate partial products by a factor of two. This facilitates fast multiplication, and, at the same time, leads to a reduced hardware requirement.

The second function is the addition of the intermediate partial products for the generation of the full-precision multiplication result prior to rounding. This addition is achieved in an entirely carry-free manner, facilitating very high multiplication speed independently of the multiplicand wordlength.

Finally, the third function is accumulation, rounding, and overflow processing of the resulting full-precision multiplication product to generate the IEEE standard 754 RNE result (c.f. Algorithm 3 (Page 39)).

3.4.1 High-Speed Fully-SB Parallel MAC Operation

Consider a MAC operation involving a M-digit SB multiplicand X^{SB}, a N-digit SB multiplier Y^{SB}, and a $M+N-1$-digit SB addend A^{SB}, in accordance with

$$P^{SB} = X^{SB}.Y^{SB} + A^{SB}, \tag{3.41}$$

where X^{SB}, Y^{SB}, and A^{SB} are given by

$$\begin{aligned} X^{SB} &= \sum_{i=0}^{M-1} x_i 2^i \quad x_i \in \{\bar{1}, 0, 1\}, \\ Y^{SB} &= \sum_{i=0}^{N-1} y_i 2^i \quad y_i \in \{\bar{1}, 0, 1\}, \end{aligned} \tag{3.42}$$

and

$$A^{SB} = \sum_{i=0}^{M+N-2} a_i 2^i \quad a_i \in \{\bar{1}, 0, 1\}. \tag{3.43}$$

Furthermore, let A^{SB} be partitioned into its MSW A^{SB}_{MSW} and LSW A^{SB}_{LSW} in accordance with Eqn. (3.25) and (3.26), respectively.

In the MAC arithmetic operation, the 5-digit overlapped scanning technique (c.f. Algorithm 1 (Page 28)) is applied to Y^{SB} to obtain the full-precision MAC result P^{SB} as

$$P^{SB} = \sum_{n=-1}^{\lceil \frac{N}{2} \rceil - 2} z_{n+1} X^{SB} 4^{n+1} + A^{SB} = \sum_{i=0}^{M+N-2} p_i 2^i \quad p_i \in \{\bar{1}, 0, 1\}, \tag{3.44}$$

where the digits z_{n+1} are computed in accordance with Eqn. (2.44).

The fully-SB parallel MAC arithmetic operation is initiated by applying the modified radix-4 recoding technique to the multiplier Y^{SB} to generate the recoded digits z_{n+1}. These digits are used to compute the intermediate partial product components $z_{n+1} X^{TC} 4^{n+1}$ in TC format, while computing the intermediate partial product sums P^{SB}_{n+1} in their SB formats by using the recursive relationship

$$P^{SB}_{n+1} = z_{n+1} X^{SB} 4^{n+1} + P^{SB}_{n}, \tag{3.45}$$

where $P^{SB}_{-1} = A^{SB}_{LSW}$, $P^{SB}_{n<-1} = 0$, and where $P^{SB}_{n+1 \geq 0}$ is given by

$$P^{SB}_{n+1 \geq 0} = \sum_{j=-1}^{n} z_{j+1} X^{SB} 4^{j+1} + A^{SB}_{LSW}. \tag{3.46}$$

It is interesting to note that the generation of the intermediate partial product components $z_{n+1} X^{SB} 4^{n+1}$ need not entail any loss of computational time as these components can be obtained as hard-wired shifted versions of the multiplicand X^{SB}. It is also interesting to note that the intermediate partial product sum components P^{SB}_{n} do not require any sign-extension as they are already in their SB formats. Therefore, it suffices to use, instead, zero insertion at the most significant part of P^{SB}_{n}, simplifying the corresponding implementation.

The recursion in Eqn. (3.45) proceeds for successive values of $n = \{-1, 0, 1, \ldots, \lceil \frac{N}{2} \rceil - 2\}$, to generate the resulting SB-product $P^{SB}_{\lceil \frac{N}{2} \rceil - 1} = P^{SB} - A^{SB}_{MSW}$. This SB-product

is then decomposed into its constituent MSW $P^{SB'}_{MSW}$ and LSW P^{SB}_{LSW}. Subsequently, $P^{SB'}_{MSW}$, A^{SB}_{MSW}, and c_R are added concurrently using a radix-2 fully-redundant addition in accordance with

$$P^{SB}_{RNE} = P^{SB'}_{MSW} + A^{SB}_{MSW} + c_R, \tag{3.47}$$

where c_R represents an IEEE Standard 754 rounding correction.

3.4.2 Fully-SB Parallel MAC Arithmetic Architecture

In this subsection, the theoretical results in Section 3.4.1 are exploited for the development of an architecture for high-speed MAC arithmetic operation. The resulting MAC architecture is as shown in the schematic diagram in Fig. 3.16.

In the above MAC arithmetic architecture, the SB multiplier Y is fed to a bank of *Modified Radix-4 Recoder* modules. These recoder modules convert the multiplier Y into its corresponding modified radix-4 redundant number representation [49]. The SB multiplicand X is fed to a bank of *Modified Radix-4 Decoder* modules within the MAC arithmetic functional unit kernel. The modified radix-4 decoded outputs are fed in their SB formats to the *SB Adder* rows. The first row of SB adders has the LSW of the result accumulated in the previous clock cycle (A^{SB}_{LSW}) as its input. The full-precision result P^{SB} is generated from the least significant digit p_0, at the top-right corner, to the most significant digit p_{M+N-2}, at the bottom-left corner of the MAC kernel. Each SB adder row generates the intermediate partial product sums P^{SB}_{n+1} which contain the two digits p_{2n+1} and p_{2n} belonging to P^{SB}_{LSW}. These digits are consumed by the *SIGN and STICKY Generation* module to generate the indicators $SIGN$ and $STICKY$. These indicators are fed to the *Correction Logic* module in order to generate the rounding correction c_R. The $(\lceil \frac{N}{2} \rceil - 1)$-th SB adder row generates $P^{SB'}_{MSW}$ which is fed to the *Rounding, MSW Addition, and Overflow Correction* module, to generate the IEEE Standard 754 rounded result P^{SB}_{RNE}. The rounding operation for P^{SB}_{RNE} is

controlled by the *RND* signal. The result P_{RNE}^{SB} is subsequently accumulated by the *MSW Accumulator* while the LSW P_{LSW}^{SB} is accumulated by the *LSW Accumulator*.

Note that in the proposed architecture, the overflow correction, rounding, and MSW-addition all occur concurrently.

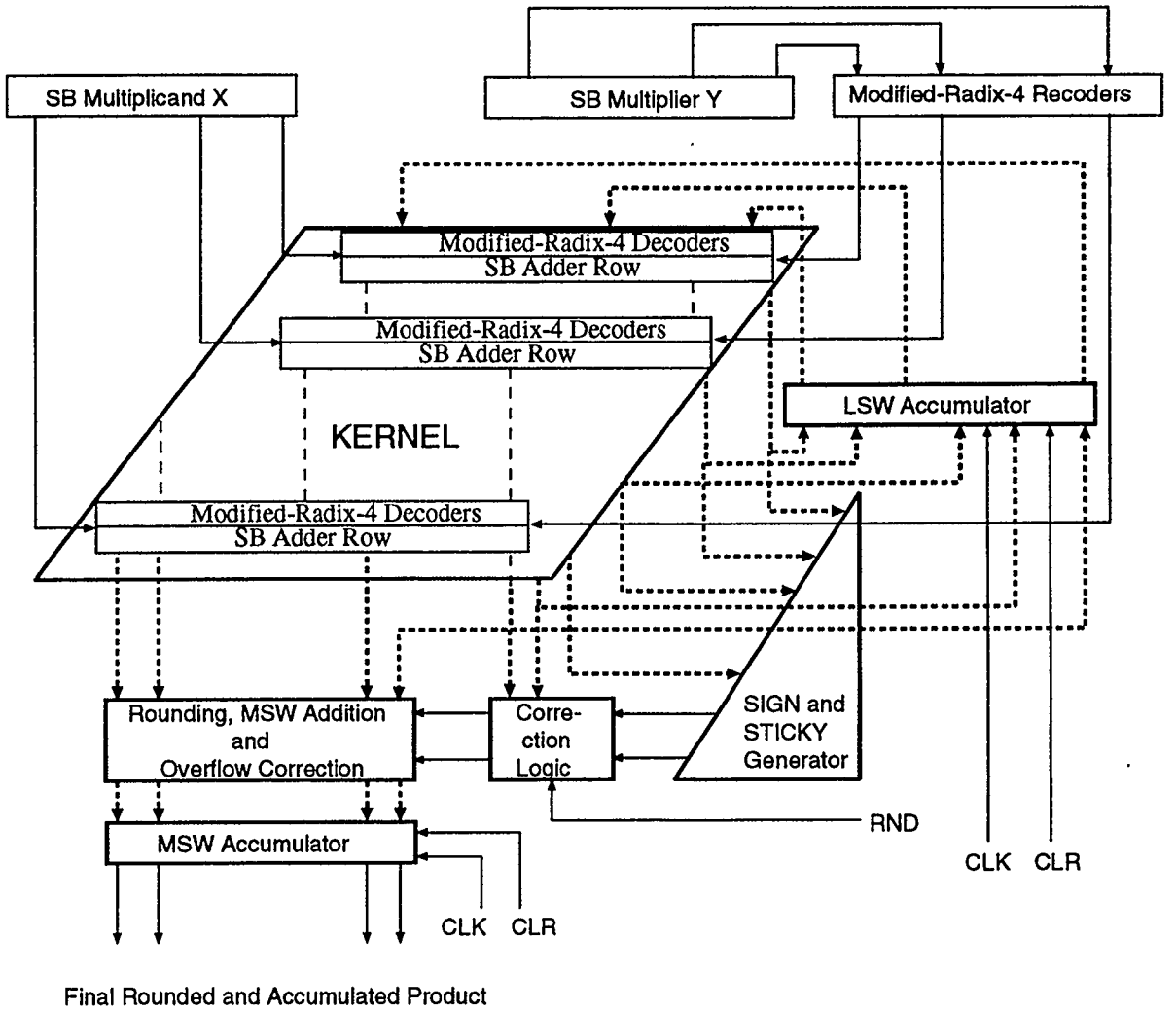

Figure 3.16. The Fully-SB Parallel MAC Arithmetic Architecture

The details of each of the modules in Fig. 3.16 are as given below.

3.4.2.1 Modified Radix-4 Recoder Module

The modified radix-4 recoders recode the SB-multiplier Y into its corresponding modified radix-4 number Z by using Algorithm 1 (Page 28). The components z_{n+1} of Z are encoded in accordance with Eqn. (2.61) and are fed to separate rows of decoders to generate the intermediate partial products.

3.4.2.2 Modified Radix-4 Decoder Module

The modified radix-4 decoders have z_{n+1} and the SB-multiplicand X as their inputs, and generate the corresponding intermediate SB partial product as their output. The decoders generates their output by appropriate shifting, negation or zeroing of the SB-multiplicand based on the value of z_{n+1}.

3.4.2.3 SB Adder Row

The terms P_{n+1}^{SB} are generated by the SB adders (c.f. Section 2.3.2.2) for the n-th row.

3.4.2.4 *SIGN* and *STICKY* Generation Module

The operation of this module is as described in Section 3.3.2.4.

3.4.2.5 Correction Logic Module

The operation of this module is as described in Section 3.3.2.5

3.4.2.6 Rounding, MSW-Addition, and Overflow Correction Module

The operation of this module is as described in Section 3.3.2.6.

3.4.2.7 Accumulator Module

The operation of this module is as described in Section 3.3.2.7

3.4.3 Performance Characteristics

In this subsection, the performance characteristics of the MAC arithmetic architecture in Fig. 3.16 are discussed in terms of the hardware area and computational time requirements. These requirements are parameterized at the gate-level for corresponding ASIC implementations.

3.4.3.1 Hardware area requirement

The hardware area requirement of the above arithmetic architecture in terms of gate-equivalents is as shown in Table 3.8.

Table 3.8. Hardware Area Requirements

Type of Hardware Cell	*Requirement*	*Gate Equivalents per Cell*
Modified-Radix-4 Recoders	$\lceil \frac{N}{2} \rceil$	25
Modified-Radix-4 Decoders	$M\lceil \frac{N}{2} \rceil$	12
Fully-Redundant Adders	$(M+1)\lceil (\frac{N}{2} \rceil + 1)$	10
$SIGN$ and $STICKY$ Generators	$\lceil \frac{N}{2} \rceil - 1$	5
Correction Generator	1	8
Overflow Corrector	1	4
DFFs with Reset	$2(M+N-1)$	6

3.4.3.2 Computational Time Requirement

The computational time requirement of the MAC arithmetic architecture is given in terms of the minimum achievable clock period T. By considering the worst case critical path in the architecture, T is obtained as

$$T = (t_{DFF} + t_{RECODE} + t_{DECODE} + t_{CR} + (\left\lceil \frac{N}{2} \right\rceil + 1)t_{SBADD}) \tag{3.48}$$

In Eqn. (3.48), t_{RECODE} (t_{DECODE}) represents the time required for the modified-radix-4 recoding (decoding), t_{CR} represents the time for generating the rounding correction c_R, t_{SBADD} is the time required for SB addition, and t_{DFF} represents the delay through a D-Flip-flop.

3.4.4 Verification

An $8 \times 8 + 15$ parallel fully-SB MAC arithmetic unit employing minimal-(s, v)-encoding was designed for implementation using the Actel 1.2μ technology. The Viewlogic simulation results shown in Fig. 3.17 depict the intermediate signals generated in the course of the MAC operations. In Fig. 3.17, the buses $Y0$ ($X0$) through $Y7$ ($X7$) represent the SB digit-components of the 8-digit multiplier Y (multiplicand X). The accumulate input has been shown using the buses $R0$ through $R14$. The IEEE standard 754 rounded and accumulated result is given as P using the buses $P0$ through $P7$, with $P7$ representing the most significant digit after overflow detection and correction.

The test vectors associated with the simulation results in Fig. 3.17 are given in Table 3.9 together with the full-precision and the expected RNE SB MAC results, accordingly. The computational time required by the above MAC unit was also verified by comparing it with the theoretically expected result.

Table 3.9. Verification Test Vectors

Test Vector	$(Y)_{SB}$	$(X)_{SB}$	$(Accumulate\ Input)_{SB}$	*RNE SB Result*
1	$0.11\bar{1}10\bar{1}0$	$0.1\bar{1}0\bar{1}010$	$\bar{1}.11101001001001$	$0.00010\bar{1}0$
2	$0.10110\bar{1}0$	$\bar{1}.111\bar{1}1\bar{1}\bar{1}$	0.00010100101000	$0.000\bar{1}10\bar{1}$
3	$\bar{1}.011100\bar{1}$	$0.010\bar{1}010$	$0.11010011\bar{0}1\bar{0}\bar{1}0$	$1.\bar{1}10\bar{1}1\bar{1}0$
4	$0.\bar{1}\bar{1}10111$	$\bar{1}.1101\bar{1}\bar{1}\bar{1}$	$0.\bar{1}\bar{1}0100010\bar{1}0001$	$\bar{1}.1\bar{1}110\bar{1}0$

3.5 Chapter Summary

In this chapter, the theoretical results in Chapter 2 have been exploited for the development of novel design and implementation techniques for high-speed VLSI arithmetic multiplication and MAC arithmetic operations.

In Section 3.2, a technique has been developed for very high-speed mixed SB/TC digit-serial modified-Booth multiplication. The salient feature of the resulting multipliers is that they permit very high throughputs for arbitrary values of digit size. Moreover, they do not involve any sign extension, permitting uniform implementations with highly localized interconnections. It has been shown that the area/time efficiency and throughput of the resulting multipliers far surpass those of the existing digit-serial modified-Booth multipliers. It has also been shown that mixed SB/TC number arithmetic is most attractive for fully parallel multiplication and MAC arithmetic operations.

In Section 3.3, an architecture has been presented for high-speed mixed SB/TC parallel modified-Booth MAC operation. This architecture features new techniques such as partitioned accumulation and concurrent rounding and overflow correction. It is most suitable for DSP algorithms that employ repetitive accumulation of independently formed multiplication products.

In Section 3.4, the modified radix-4 recoding technique has been used to extend the architecture in Section 3.3 to handle fully-SB parallel MAC arithmetic operation.

The above multiplication and MAC arithmetic architectures have been parameterized in terms of their area-time requirements for the corresponding Actel 1.2μ technology implementations. The resulting implementations have been verified by using Viewlogic simulations.

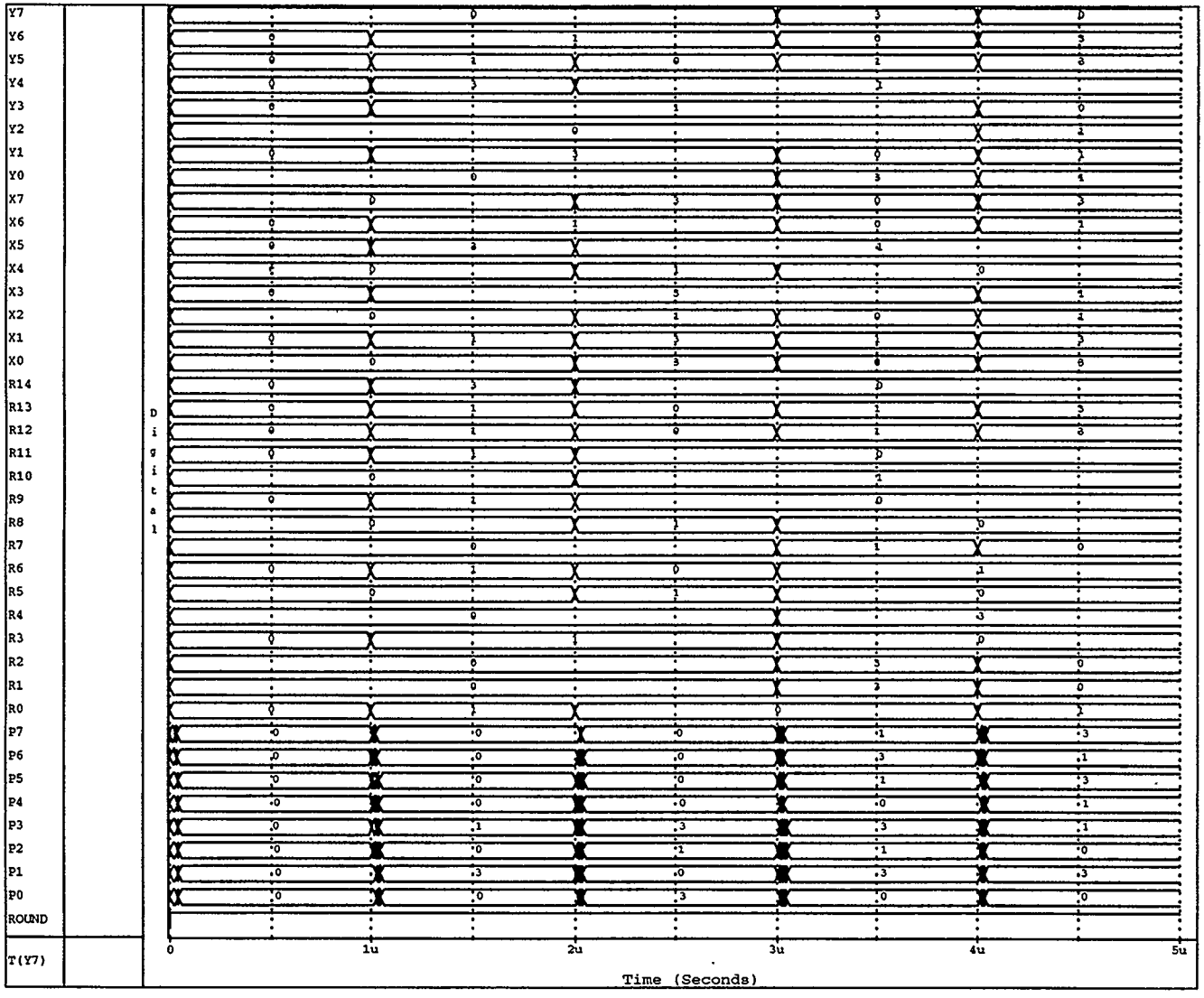

Figure 3.17. Simulation Result for the $8 \times 8 + 15$ Fully-SB Parallel MAC Arithmetic Unit

CHAPTER 4

MAC-MODULARIZATION OF DIGITAL FILTERS

4.1 Introduction

In the VLSI implementation of digital filter algorithms, it is often desirable to employ uniform building blocks in conjunction with regular and modular architectures [44, 21]. Such architectures have several advantages in terms of localized interconnections, pipelinability, ease in scheduling, and simplification of design and implementation effort.

Recent advances in VLSI technology have led to the availability of highly parallel processing elements (e.g. systolic arrays) for demanding modern digital filter applications [14]. The resurgence of non-traditional arithmetic has led to an increase in the computing power available by such parallel processing elements. The exploitation of the available computing power for efficient high-speed processing depends on the regular distribution of the data dependencies, and the regularity of basic operations in the digital filter algorithm.

Digital filter algorithms are generally represented in the form of signal-flow graphs (SFGs) [1], consisting of multiplication and addition operations. Unfortunately, the non-homogeneous nature of these operations does not permit straightforward parallel and systolic realization. However, it is possible to translate these algorithms into suitable equivalent forms involving combined multiply-accumulate (MAC) arithmetic operations [52]. The translation of a digital filter algorithm consisting of separate multiplication and addition operations to a corresponding algorithm consisting of MAC operations only is referred to as MAC-modularization. The resulting MAC-modularized digital filter algorithm possesses regularity and modularity, thereby permitting efficient use of the available computing elements, simpler scheduling, and easier design and implementation.

There are however certain limitations of digital filter SFGs which prevent the application of MAC-modularization techniques to them. Consequently, digital filter SFGs need to be converted to a more suitable representation called the directed reduced SFG.

MAC-modularization is based on the application of graph-theoretic [33, 24, 41] techniques to the directed reduced SFG. These techniques achieve MAC-modularization by exploiting certain properties of the directed reduced spanning tree of the directed reduced SFG. Multiplication operations are moved in a systematic manner along the directed reduced spanning tree in order to result in the corresponding MAC-modularized directed reduced SFG. Finally, the resulting SFG is converted back to its corresponding MAC-modularized digital filter SFG. In all, the MAC-modularization procedure can be summarized as follows.

1. Generate a directed reduced SFG for the given digital filter SFG,

2. Generate a directed reduced spanning tree for the directed reduced SFG,

3. Apply MAC-modularization techniques to the directed reduced SFG using the directed reduced spanning tree, and

4. Generate a digital filter SFG consisting of MAC operations using the resulting MAC-modularized directed reduced SFG.

It should be pointed out that there exists a unique MAC-modularized digital filter SFG corresponding to a given directed reduced spanning tree. However, a directed reduced SFG can have several directed reduced spanning trees. This results in several MAC-modularized solutions for a given digital filter SFG. Although all these solutions are equivalent in terms of infinite-precision arithmetic realization, they exhibit remarkably different properties for the corresponding finite-precision arithmetic realizations.

In this thesis, the finite-precision arithmetic effects of overflow and roundoff noise are taken into account to develop a fitness function in order to facilitate the selection of an optimal MAC-modularized solution. This fitness function is then used to develop two different techniques for MAC-modularization, one based on exhaustive enumeration of all the possible solutions, and the other based on a heuristic approach. The enumerative approach generates all the possible solutions and then employs the fitness function to select the optimal solution. This approach is suitable for small and medium size problems. However, large problems are computationally intractable because of the rapid enlargement of the solution space. Therefore, a heuristic approach is also developed. This heuristic approach operates by the local application of the fitness function in order to generate a near optimal solution at the expense of minimal computational effort. It is shown that this heuristic approach generates a solution which is indeed close to the optimal solution.

Finally, the above techniques are implemented in the form of a powerful object-oriented software package for MAC-modularization. This software has weights associated with the finite-precision effects of overflow and roundoff noise, thereby permitting a certain degree of user control on the fitness function. This software package implements both, the enumerative, and the heuristic approaches. It has been tested with a variety of dense and sparse multiplication problems, and the results are presented in this chapter.

Section 4.2 deals with the theoretical basis and techniques for MAC-modularization of digital filter SFGs. Graph-theoretic techniques are developed using the directed reduced SFG and the directed reduced spanning trees, and are supported with the help of mathematical theorems and proofs. These techniques are subsequently translated into algorithms for MAC-modularization. However, these techniques result in several solutions for a given digital filter SFG, necessitating the development of a fitness function for the selection of the optimal solution. Section 4.3 is concerned

with the development of such a fitness function based on finite-precision arithmetic effects. Sections 4.4 and 4.5 present enumerative and heuristic approaches to MAC-modularization. Finally, these techniques are used for the development of a powerful object-oriented MAC-modularization software package in Section 4.6.

4.2 Principle underlying MAC-Modularization

The first step in the process of MAC-modularization involves the conversion of a given digital filter SFG into its corresponding directed reduced SFG as defined in the following.

Definition 5 *A directed reduced SFG is a class of single-input single-output connected SFGs consisting of directed edges and vertices (nodes), having*

- *one input node with no incoming edges,*
- *one output node with no outgoing edges,*
- *addition nodes with two incoming edges,*
- *one multiplication operation per edge, and*
- *no self loops.*

Note that by definition the directed reduced SFG is devoid of any delays.

Let $G(V, E)$ represent a directed reduced SFG, where V is a set of vertices and E is a set of directed edges. Every vertex v ($\in V$) represents an addition node. Every directed edge e_{ij} ($\in E$) is associated with a multiplication coefficient m_{ij}, and a signal-flow and precedence relationship from the node n_i to the node n_j. Let $d_{in}(v)$ represent the in-degree of the vertex v, and be defined as the number of directed edges which have v as their terminating vertex. Similarly, let $d_{out}(v)$ represent the out-degree of the vertex v, and be defined as the number of directed edges which have v as their originating vertex. Further, let the *direct-addition input* of a node $n_j \in V$ be defined as the directed edge e_{ij} with a multiplication coefficient of unity. Lastly,

let the *multiplication input* of the node n_j be defined as the directed edge e_{hj} with a multiplication coefficient other than unity. Then, one can define a root node for $G(V, E)$ as follows.

Definition 6 *The root node $\wp$ is defined as a unique node associated with $G(V, E)$ such that*

1. $\wp \in V$,
2. $d_{in}(\wp) = 0$, *and*
3. *every other node in the set V is reachable from $\wp$ along a directed path consisting of the edges contained in the set E.*

In the MAC-modularization procedure, a reduced SFG consisting of the operations of multiplication and addition is converted to a corresponding SFG consisting of MAC operations (c.f. Definition 1) as follows.

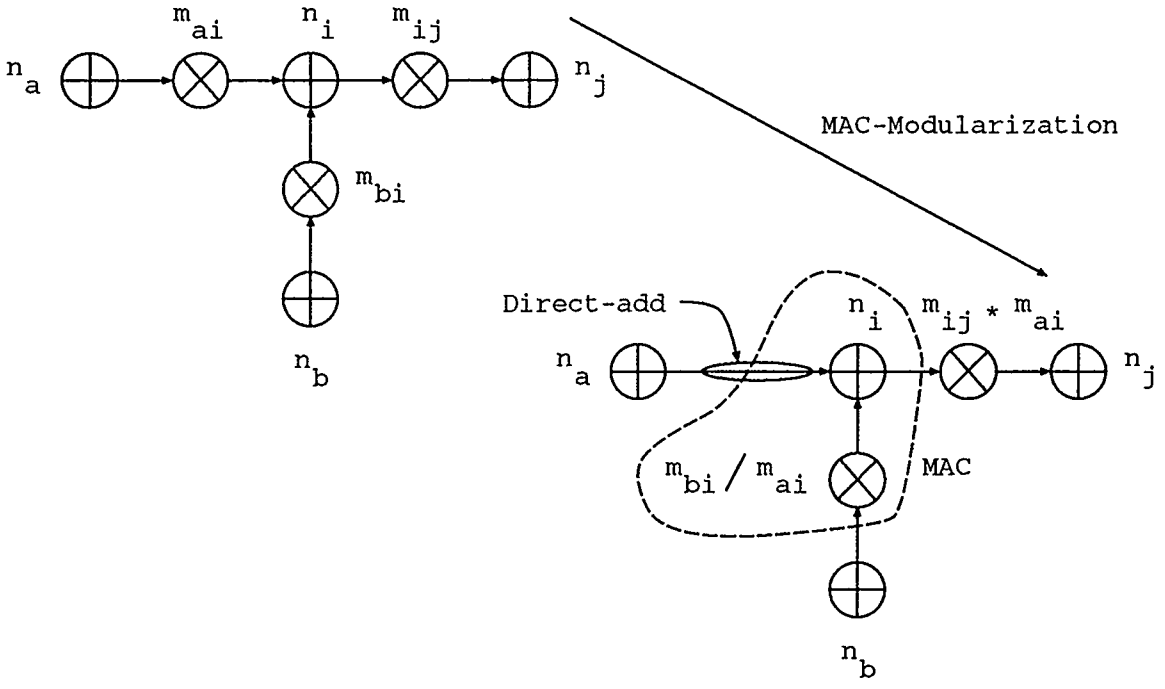

Figure 4.1. Principle Underlying MAC-Modularization

Let the schematic in Fig. 4.1 represent a section of $G(V, E)$, consisting of the nodes $\{n_a, n_b, n_i, n_j\} \in V$, and the directed edges $\{e_{ai}, e_{bi}, e_{ij}\} \in E$. Fig. 4.1 shows the conversion of the edge e_{ai} into a direct-addition input to the node n_i in order to achieve the MAC-modularization of the node n_i. By taking into account the fact that the addition node n_i has two incoming edges e_{ai} and e_{bi}, the process of MAC-modularization can be seen as the procedure of moving the multiplication coefficient on one of the incoming edges to translate that edge into a direct-addition input, while suitably adjusting the multiplication coefficients on the other incoming edge and the outgoing edges of the node n_i. From a graph-theoretic viewpoint, if e_{ai} represents a *tree edge*, then e_{bi} would represent the *co-tree edge*. At the same time, every other edge e_{ix} would represent an outgoing edge from n_i to an arbitrary node $n_x \in V$. In summary, the MAC-modularization can be described as the conversion of e_{ai} into a direct-addition edge, the scaling of the multiplication coefficient on e_{bi} by $\frac{1}{m_{ai}}$, and the scaling of the multiplication coefficient on e_{ix} by m_{ai}. The multiplication operation on the co-tree edge e_{bi} can now be combined with the addition operation on the node n_i itself, to result in one composite MAC operation.

The MAC-modularization procedure is developed in terms of a corresponding directed reduced spanning tree associated with the directed reduced SFG. The directed reduced spanning tree is defined as follows.

Definition 7 *The directed reduced spanning tree is a directed tree of a directed reduced SFG in which every node except the input node has an in-degree of unity.*

A directed graph is a *directed tree* if it has a root node and its underlying undirected graph is a tree, i.e., it is connected and circuit free. A subgraph H of $G(V, E)$ is called a *directed reduced spanning tree* of G, if H is a directed tree which includes all the vertices of G. Therefore, H can be designated as $H(V, E')$, where $E' \subset E$. Furthermore, every vertex $v \in V$ in $H(V, E')$ has a unique directed path from the

root $\wp$ to itself, consisting of the edges in the set E'.

Theorem 4.1 *The root node of a directed reduced SFG is the input node.*

Proof Let the input node of the directed reduced SFG be denoted by s. Since the directed reduced SFG is defined to be connected (c.f. Definition 5), there exists at least one directed path from the node s to every other node in the directed reduced SFG. However, the node s has no incoming edge on account of the fact that $d_{in}(s) = 0$. Therefore, it can be seen that no other node n_j qualifies to be the root because there exists no directed path from any node n_j to the node s. Hence, the node s qualifies as the root node. ■

MAC-modularization of a directed reduced SFG is carried out by translating the tree arcs to direct-addition arcs, requiring the existence of at least one directed reduced spanning tree.

Theorem 4.2 *There exists at least one directed reduced spanning tree in a given directed reduced SFG.*

Proof By contradiction.

Let U denote the set of un-visited nodes and S denote the set of visited nodes. Starting with the input node s, with $U = V - \{s\}$ and $S = \{s\}$, add nodes to S in accordance with the following procedure:

while $(U \neq \emptyset)$ do
begin
  choose a node $n \in S$;
  choose $u \in U$ such that n and u are connected by an arc e_{nu};
  $S = S + \{u\}$;
  $U = U - \{u\}$;
end

In the above procedure, each node is visited only once. This results in a directed reduced spanning tree of G which is constructed systematically by adding the the arcs e_{nu} in each iteration. Therefore, each node except the node s has an in-degree of unity.

This procedure fails to terminate if there exists at least one node in U which is not reachable from any node in S. This means that such a node is not connected to any other node in G. Therefore, G is not a directed reduced SFG (c.f. Definition 5), leading to a contradiction. ■

It can be noted that the above procedure can be used exhaustively to generate all the possible directed reduced spanning trees of G rooted in the node s.

By using Theorems 4.1 and 4.2, it can be shown that MAC-modularization is possible if and only if the spanning tree of G is a directed reduced spanning tree rooted in the node s. In order to demonstrate this, let the in-degree matrix $\mathbf{D}$ of G be defined as

$$\mathbf{D}(n_i, n_j) = \begin{cases} d_{in}(n_i) & \text{if } n_i = n_j \\ -k & \text{if } n_i \neq n_j \end{cases} \tag{4.1}$$

where k is the number of edges in G from n_i to n_j. Then,

Lemma 5 *A finite directed graph with no self loops is a directed tree with root $\wp$ if and only if its in-degree matrix $\mathbf{D}$ satisfies the following properties:*

1.

$$\mathbf{D}(n_i, n_i) = \begin{cases} 0 & \text{if } n_i = \wp \\ 1 & \text{if } n_i \neq \wp. \end{cases} \tag{4.2}$$

2. The co-factor resulting from the erasure the $\wp$-th row and column of $\mathbf{D}$ yields 1.

Proof See [41]. ■

It can be seen easily that the directed reduced spanning tree satisfies Lemma 5.

Theorem 4.3 *The given directed reduced SFG G can be MAC-modularized if and only if its underlying spanning tree is a directed reduced spanning tree rooted in the node s.*

Proof

a) If part: From first principles. If the underlying spanning tree is a corresponding directed reduced spanning tree rooted in the node s, the multipliers from the tree edges can be moved to the corresponding co-tree edges, and the resulting multiplication operations on the co-tree edges can be combined with the corresponding additions at the nodes to form MAC operations (c.f. Fig. 4.1). This begins at the input node s and progresses along the given directed reduced spanning tree towards the leaf nodes, resulting in a MAC-modularized reduced SFG. This establishes the *if* part of the theorem.

b) Only if part: By contradiction. The proof is developed along the lines of the proof of Lemma 5.

Define a graph $H'(V, E')$ which spans all the nodes of the G starting at the input node s, with the arcs consisting of the direct-addition arcs corresponding to a MAC-modularized reduced SFG of G. Further, assume that $H'(V, E')$ is not a directed reduced spanning tree of G.

Let $\mathbf{D}$ be the in-degree matrix of $H'(V, E')$. Then $\mathbf{D}(s, s) = 0$, since the input node has no predecessors. Furthermore, $\mathbf{D}(n_i, n_i) = 1$ if $n_i \neq s$, since each node in a MAC-modularized directed reduced SFG has only one direct-addition input by the definition of $H'(V, E')$. Thus, property (1) of Lemma 5 holds.

Now, let the vertices of $H'(V, E')$ be remembered in such a manner that the root node s is numbered 1, and if $n_i \longrightarrow n_j$, then $i < j$. This is achieved by numbering the vertices which are successors of 1 as 2, 3, ... and sequentially numbering vertices which are the successors of 2, 3, etc. till all the vertices have been numbered. The new

in-degree matrix $\mathbf{D}'$ is derivable from the original in-degree matrix $\mathbf{D}$ by performing some permutations on the rows and some permutations on the columns. Since such permutations do not change the determinant, the new in-degree matrix will satisfy the following properties.

$$\mathbf{D}'(i,j) = \begin{cases} 0 & \text{if } i = j = 1 \text{ or } i > j \\ 1 & \text{if } i = j \text{ and } i = 2, 3, \ldots, n \end{cases} \tag{4.3}$$

Therefore, the co-factor resulting from the erasure of the first row and column of $\mathbf{D}'$ yields 1. Thus, property (2) of Lemma 5 also holds. This implies that $H'(V, E')$ is indeed a directed reduced spanning tree of G, contradicting the assumption. ■

As demonstrated earlier (see proof of Theorem 4.2), a directed reduced SFG can have several directed reduced spanning trees rooted in s. Theorems 4.1 and 4.3 can easily be used to show that any of these directed reduced spanning trees can be used to MAC-modularize the directed reduced SFG.

Theorem 4.4 *A given directed reduced SFG can be modularized into an equivalent SFG consisting of MAC operations by any one of the directed reduced spanning trees rooted in the node s.*

Proof MAC operations are three input operations (c.f. Definition 1), where Z corresponds to the direct-addition input (tree arc), X corresponds to the multiplied input (co-tree arc), and Y corresponds to the multiplication coefficient on the co-tree arc. These operations are combined together to represent a single node in a given MAC-modularized directed reduced SFG. Each such node is reachable from the input node s, but the node s in not reachable from any other node. Therefore, the directed reduced spanning trees must be rooted in the node s. This, in conjunction with Theorems 4.1 and 4.3, completes the proof. ■

The number of MAC operations resulting from the MAC-modularization of a directed reduced SFG is given in accordance with the following theorem.

Theorem 4.5 *A directed reduced SFG with n addition operations can be modularized into an equivalent SFG consisting of* $(n+1)$ *MAC operations using any of the directed reduced spanning trees, except in the case when the directed path from the root node to the output node in the corresponding directed reduced spanning tree has unity gain. In such a case only n MAC operations are required.*

Proof For MAC-modularization, all incoming arcs to addition nodes in a given directed reduced spanning tree must be made direct-addition arcs. The resulting incoming arc to the output node reflects the total gain from the root node to the output node along that directed reduced spanning tree. This output edge may have a non-unity gain since the output node has an out-degree of 0. Therefore, a redundant direct-addition edge with an addend of zero must be augmented to the output node, and the output node, output edge, and the redundant direct-addition edge must be combined into a MAC operation, resulting in $(n + 1)$ MAC operations.

In the case that the directed path from the root node to the output node has a gain of unity, no such redundant direct-addition is required, resulting in n MAC operations. ■

4.2.1 Co-Tree Multiplier Value Computation

One can exploit the directed reduced spanning tree in conjunction with the directed reduced SFG in order to determine the co-tree multiplier values in the corresponding MAC-modularized directed reduced SFG.

Lemma 6 *Any co-tree multiplier* m_{ij} *between the nodes* n_i *and* n_j *corresponding to the arc* e_{ij} *in the directed reduced SFG assumes a new value* m'_{ij}*, given by*

$$m'_{ij} = m_{ij}.(g_i/g_j), \tag{4.4}$$

where g_k *represents the gain from the input node to the node* n_k *along the corresponding directed reduced spanning tree.*

Proof Consider an arbitrary directed reduced spanning tree of $G(V, E)$, with a pair of nodes n_i and n_j shown explicitly, in Fig. 4.2. Let the co-tree edge e_{ij} have an initial gain of m_{ij}. Let the unique paths from the node s to the node n_i and from the node s to the node n_j have the gains g_i and g_j, respectively.

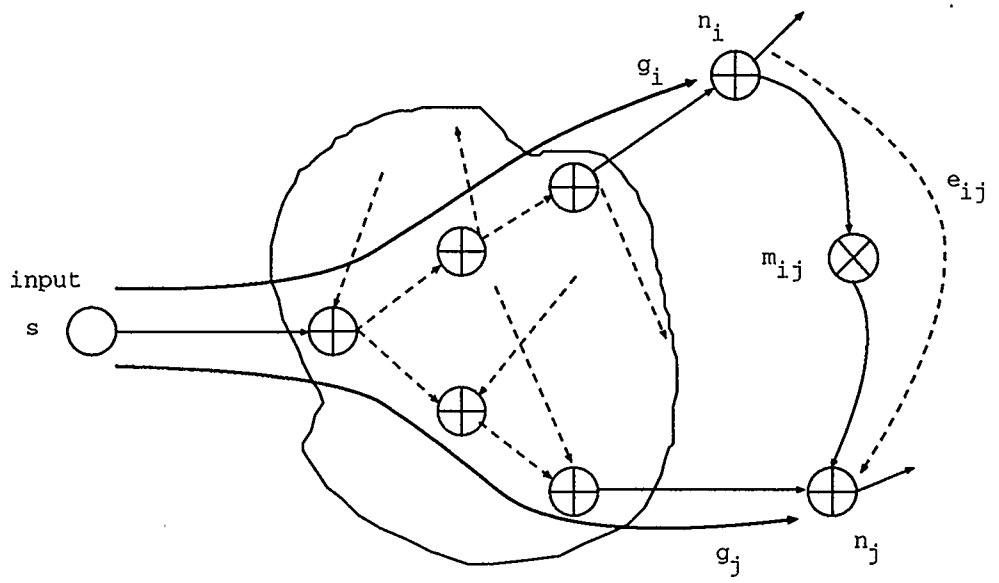

Figure 4.2. Co-tree Multiplier Value Computation

MAC-modularization involves the systematic movement of the multiplications along these paths in order to make these path gains unity. Therefore, the effective multiplication coefficient movement across n_i (n_j) is g_i (g_j). Since m_{ij} corresponds to the output (input) multiplication coefficient of n_i (n_j), $m'_{ij} = m_{ij}.(g_i/g_j)$. This completes the proof. ■

4.2.2 Output Multiplier Value Computation

The value of the output multiplier is equal to the gain from the root (input) node to the output node along the directed reduced spanning tree. If this gain is not unity, an additional MAC operation is required at the output node (c.f. Theorem 4.5) with

its addend input of zero.

4.2.3 Node-numbering in the directed reduced SFG

Let N and U represent a set of numbered and un-numbered nodes. Let $G(V, E)$ be the directed reduced SFG whose nodes are to be numbered. Let s be the input node of G.

Algorithm 5

```
begin
   node_num = 0;
   assign the number node_num to the node s;
   N = {s}; U = V − {s};
   while (U ≠ ∅) do
   begin
      B = {n_i, ..., n_k} such that
         a) n_i, ..., n_k ∈ U
         b) n_i, ..., n_k are connected as terminating nodes by unit edges
            with N;
      U = U − B;
      while (B ≠ ∅) do
      begin
         node_num = node_num + 1;
         assign the number node_num to n_j, where n_j ∈ B;
         N = N + {n_j};
         B = B − {n_j};
      end
   end
end.
```

4.2.4 Self-loop Elimination

The conversion of a digital filter SFG to a corresponding directed reduced SFG requires delay removal. However, delay removal can result in the creation of self-loops at certain nodes. Such self-loops prevent the successful application of the graph-theoretic techniques for MAC-modularization.

The above problem is best resolved by eliminating these self-loops. This is achieved by disconnecting the outgoing part of the self-loop edge from the originating node, and connecting it to the input node. Fig. 4.3 demonstrates the elimination of the self-loop at the node n_i. The self-loop is eliminated by breaking the self-loop and connecting the outgoing part of the self-loop edge to the input node with the multiplier m_{si} given by

$$m_{si} = \begin{cases} 1 & \text{if } m_{ii} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

This technique of self-loop elimination permits the generation of all the directed reduced spanning trees with e_{ai} as the co-tree arc when $m_{ii} = 1$ (allowing the self-loop to be a direct-addition input to n_i). Note that the original self-loop multiplication coefficients must be remembered as they need to be restored after generating the required directed reduced spanning trees.

4.2.5 Algorithm for converting a digital filter SFG to its corresponding directed reduced SFG

Let V denote the set of all the nodes in the given digital filter SFG. The algorithm for converting a digital filter SFG to its corresponding directed reduced SFG is as follows.

Algorithm 6

`begin`

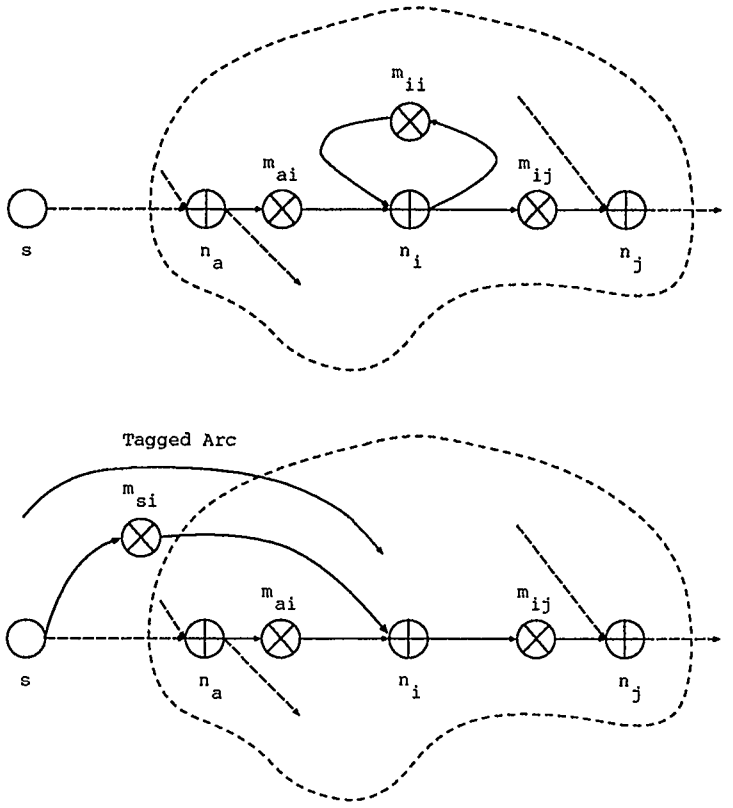

Figure 4.3. Self-Loop Elimination

```
   Eliminate all the delays and short their corresponding connection
      points;
   Eliminate all the self-loops using the technique in Sec. 4.2.4;
   Assign a tag to each eliminated self-loop arc;
   Remember the original multiplication coefficient values of each of
      the self-loop arcs;
   Number each node n_i ∈ V using Algorithm 5 (Page 104);
end.
```

4.2.6 Algorithm for MAC-modularizing the directed reduced SFG

The following algorithm converts a given directed reduced SFG (generated using Algorithm 6 (Page 105)), into a corresponding MAC-modularized directed reduced

SFG.

Algorithm 7

```
begin
   Construct a directed reduced spanning tree for the
      directed reduced SFG (see the following);
   Compute the new co-tree multiplier values by using the constructed
      tree and Lemma 6;
   Insert the output MAC operation using the technique in Sec. 4.2.2;
   Convert all the tree arcs (except the output arc) to
      direct-addition arcs;
   Combine the resulting tree and co-tree;
   Combine the node additions and their corresponding co-tree
      multiplications into composite MAC operations;
end.
```

4.2.7 Algorithm for converting a MAC-modularized directed reduced SFG to the corresponding digital filter SFG consisting of MAC operations

The following algorithm converts the given MAC-modularized directed reduced SFG (generated using Algorithm 7 (Page 107)) into its corresponding digital filter SFG consisting of MAC operations.

Algorithm 8

```
begin
   Reconstruct the self-loops using the tagged arcs;
   Restore self-loop multipliers for tagged arcs with m_si = 0;
   Insert the delays on the appropriate arcs;
end.
```

4.3 Optimal MAC-Modularized digital filter SFG Selection

In the previous section, it was shown that one can use any of the directed reduced spanning trees of a given directed reduced SFG in order to carry out the MAC-modularization of the corresponding digital filter SFG. Since each of these spanning trees leads to a unique MAC-modularized solution, one can obtain a family of solutions for a given digital filter SFG. Although the resulting solutions are all equivalent in terms of infinite-precision arithmetic, they exhibit remarkably different properties for the corresponding finite-precision arithmetic implementations. In this thesis, the finite-precision behaviour of the solutions is used as a criterion to arrive at the optimal MAC-modularized digital filter SFG.

There are primarily three finite-precision arithmetic effects in fixed-point digital filter implementations. These are [37]:

1. Transfer function deviation due to coefficient quantization.

2. Overflow oscillations (limit cycles) due to signal growth within the system.

3. Roundoff noise caused by multiplication product rounding.

The first of the above three effects is linear in nature and simply leads to errors in the time-domain and frequency-domain response characteristics of the implementation. The second and third effects, on the other hand, are non-linear in nature and influence the stability of the digital filter. Furthermore, they determine the dynamic range and the signal-to-noise ratio of the system. The finite-precision arithmetic effects associated with the overflow oscillations and roundoff noise serve as a valuable criteria for the selecting the optimal MAC-modularized digital filter SFG.

It is important to note that, since the linear effects (measured using sensitivity characteristics) are strongly influenced by the topology, and since the MAC modularization preserves the general topology of the SFG, the proposed MAC modularization does not degrade the sensitivity features of the original digital filter.

In the following, the L1-norm calculations [25] for the possible degree of overflow and roundoff noise are used to select the optimal MAC-modularized digital filter SFG.

The possible degree of overflow is determined by the maximum gain from the input node to the addition nodes within the digital filter. The roundoff noise is determined by summing the gains from the multiplier outputs to the output node of the digital filter. The possible degree of overflow affects the number of bits/digits to be allocated to the most-significant part of the signal word, whereas the roundoff noise affects the number of bits/digits to be allocated to the least-significant part of the signal word.

This section is concerned with the development of a fitness function for finding the optimal MAC-modularized solution for a given digital filter SFG. This fitness function is developed in terms of the possible degree of overflow and roundoff noise in the digital filter. It is also shown that the overflow and roundoff noise characteristics of the MAC-modularized solution can be determined by using the corresponding directed reduced spanning tree in conjunction with the overflow and roundoff noise characteristics of the original digital filter SFG.

4.3.1 Fitness Function for MAC-Modularized Reduced-SFGs

The fitness function for any given k^{th} MAC-modularized digital filter SFG can be determined by

$$fitness(H_k(V, E_k)) = \frac{1}{(oflow\_weight \times O'_{i_{MAX}}) + (roff\_weight \times \sum R'_i)} \quad (4.6)$$

In this equation, $H_k(V, E_k)$ represents the k^{th} directed reduced spanning tree associated with the directed reduced SFG $G(V, E)$ of the original digital filter SFG, with $E_k \subset E$. Moreover, $O'_{i_{MAX}}$ represents the L1-norm value of the maximum overflow at the addition nodes in the MAC-modularized digital filter SFG. In addition, $\sum R'_i$ represents the sum of the L1-norm values of the roundoff noise contributions of each of the co-tree arcs of $H_k(V, E_k)$ that have non-unity absolute gains. Finally,

oflow_weight and *roff_weight* are the user specified weights associated with the overflow and roundoff noise contributions, respectively.

The fitness function in Eqn. (4.6) is used for the selection of a MAC-modularized solution that inherently minimizes the harmful finite-precision effects of overflow oscillations and roundoff noise. Such a solution is optimal in terms of the user defined weights for overflow and roundoff noise.

4.3.2 Overflow and Roundoff Noise Calculations

In this subsection, a technique is developed to determine the L1-norm values of the possible degree of overflow and roundoff noise in any MAC-modularized solution. This technique is based on the corresponding directed reduced spanning tree and the L1-norm values of the possible degree of overflow and roundoff noise in the original digital filter SFG. The resulting technique permits the computation of the fitness function in a very simple and straightforward manner.

The possible degree of overflow and the roundoff noise in a given digital filter SFG are computed in terms of the maximum gains from the input to any addition node and the sum of the gains from the multiplier outputs to the output node, respectively. To facilitate the discussion, it is shown that the MAC-modularization of a node affects the gain at that node only, leaving the gains at all the other nodes unchanged. This property is subsequently used to develop the technique for computing the possible degree of overflow and roundoff noise in the MAC-modularized digital filter SFG.

Theorem 4.6 *MAC-modularization of an arbitrary node n_i in given digital filter SFG results in a change in the gain at that node only, leaving the gains at all other nodes unchanged.*

Proof Let the schematic in Fig. 4.4 represent a section of a digital filter SFG with a node n_i in its pre-modularized state. Let g_p represent the gain from the

input node s to the node n_p. Moreover, let the digital filter under consideration be infinite-precision linear.

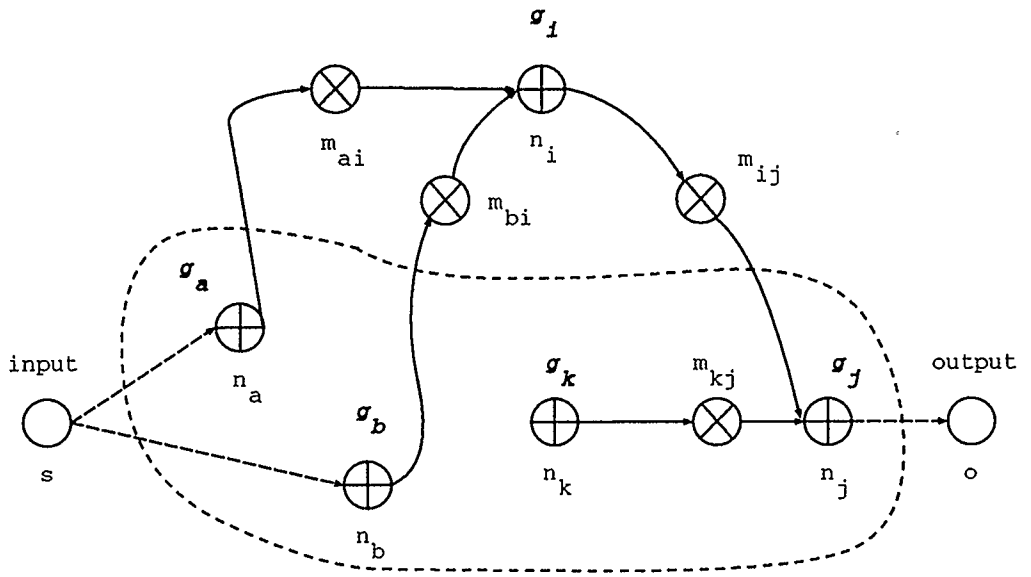

Figure 4.4. Section of Reduced-SFG: Pre-Modularized State

Further, let the application of MAC-modularization to the node n_i by the movement of the multiplier m_{ai} result in the configuration as shown in Fig. 4.5. Let g'_p now represent the corresponding new gain at the node n_p.

Due to the fact that the system is linear, the movement of the multiplier m_{ai} (c.f. Fig. 4.5) does not disturb any of the loop gains in the system. Moreover, since the nodes n_a and n_b can be reached from n_i through feedback paths only, the gains at nodes n_a and n_b remain unchanged in accordance with

$$g'_a = g_a \tag{4.7}$$

and

$$g'_b = g_b, \tag{4.8}$$

respectively.

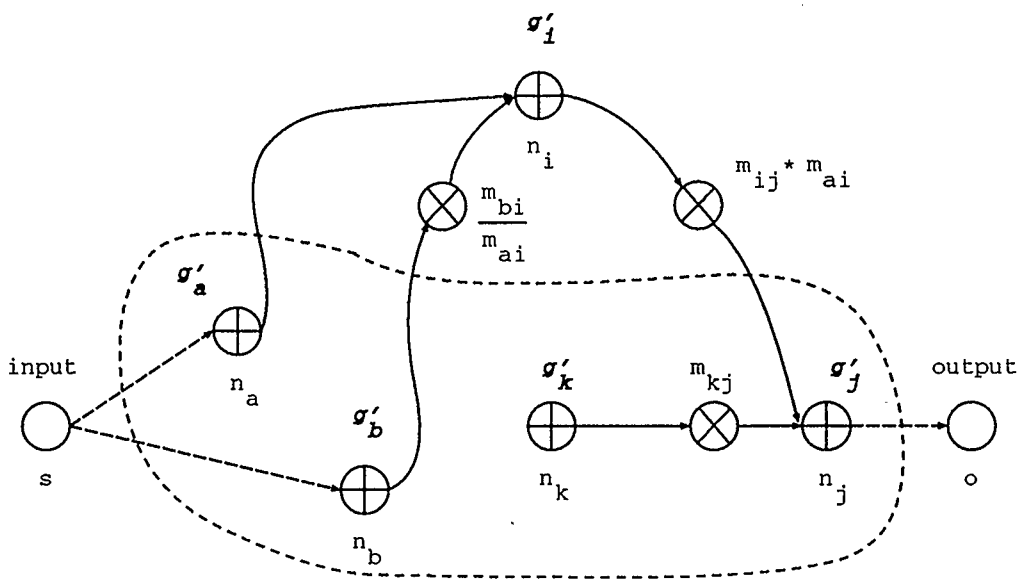

Figure 4.5. Movement of m_{ai} across n_i: Post-Modularized State

The gain from the input node s to the node n_i before MAC-modularization can be expressed as

$$g_i = m_{ai}g_a + m_{bi}g_b. \tag{4.9}$$

The corresponding gain after modularization becomes

$$g_i' = g_a + \frac{m_{bi}}{m_{ai}}g_b \tag{4.10}$$

By using Eqns. (4.9) and (4.10), one obtains

$$g_i' = \frac{1}{m_{ai}}g_i. \tag{4.11}$$

Similarly, the gain from the input node s to the node n_j before MAC-modularization can be expressed as

$$g_j = m_{ij}g_i + m_{kj}g_k. \tag{4.12}$$

The corresponding gain after modularization becomes

$$g_j' = m_{ij}m_{ai}g_i' + m_{kj}g_k. \tag{4.13}$$

By using Eqn. (4.11) in Eqn. (4.13), one obtains

$$g'_j = g_j. \tag{4.14}$$

Therefore, from Eqns. (4.7), (4.8), (4.11), and (4.14), it can be seen that MAC-modularization of the node n_i changes the gain at that node only, leaving the gains at all other nodes unchanged. This completes the proof. ■

Let n_i represent an arbitrary node in a given directed reduced SFG. Moreover, let O_i represent the L1-norm value of the possible degree of overflow at node n_i. Further, let R_{1i} and R_{2i} represent the L1-norm values of the roundoff noise caused by each of the two incoming edges to n_i. Then, the node n_i is characterized in terms of its overflow and roundoff noise contributions with the 3-tuple

$$< O_i, R_{1i}, R_{2i} > . \tag{4.15}$$

The same node n_i, after MAC-modularization is characterized by a corresponding 3-tuple

$$< O'_i, R'_{1i}, R'_{2i} > . \tag{4.16}$$

Note that R'_{1i} or R'_{2i} is zero if the corresponding multiplier in the MAC-modularized directed reduced SFG is 1 or -1. The 3-tuple in Eqn. (4.16) can be derived from the corresponding 3-tuple in Eqn. (4.15) in accordance with the following theorem.

Theorem 4.7 *Given a directed reduced SFG with the L1-norm values of the possible degree of overflow and roundoff noise for each node n_i represented using the 3-tuple in Eqn. (4.15), the corresponding L1-norm values after MAC-modularization, as given in Eqn. (4.16), can be derived as*

$$O'_i = (\frac{1}{g_i})O_i \tag{4.17}$$

and

$$R'_{xi} = \begin{cases} g_i R_{xi} & \text{if } |m_{xi}| \neq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{4.18}$$

where g_i is the gain from the root (input) node to node n_i along the corresponding directed reduced spanning tree, and m_{xi} is the multiplier coefficient along the directed edge e_{xi}.

Proof MAC-modularization of a given directed reduced SFG is carried out by moving the multipliers along the directed reduced spanning tree starting at the root node and moving towards the leaf nodes until all the tree edges (except for the output edge) become direct-addition edges. This procedure results in a multiplier movement of g_i across node n_i. Since the overflow is computed as a function of the maximum gain from the input to that node n_i, the multiplier movement of g_i across n_i results in $O'_i = (\frac{1}{g_i})O_i$ in accordance with Eqn. (4.11). On the other hand, the roundoff error is computed using the gain from the multiplier output incoming to the node n_i to the output node. A multiplier movement of g_i across the node n_i increases this gain by g_i, thereby resulting in $R'_{xi} = g_i R_{xi}$. However if $|m_{xi}| = 1$, then the roundoff error contribution becomes 0 since multiplication by 1 or -1 does not induce any roundoff error. This, in conjunction with Theorem 4.6 completes the proof. ■

In summary, the directed reduced spanning tree can be used to compute the new overflow and roundoff error values for the corresponding MAC-modularized digital filter SFG.

4.4 An Enumerative Approach to MAC-Modularization

For digital filter SFGs having small to medium sizes, it is possible to perform an exhaustive enumeration of MAC-modularized solutions for finding the optimal solution. This exhaustive enumeration involves, the generation of all the corresponding directed reduced spanning trees of the given directed reduced SFG, and the selection of the optimal solution based on the fitness function in Eqn. (4.6).

4.4.1 Algorithm for Directed Reduced Spanning Tree Enumeration

Let S represent the set of all the nodes in a given directed reduced SFG. At each iteration of the following algorithm, let the set S be partitioned into two sets, namely V signifying the set of visited nodes, and U signifying the set of unvisited nodes. Moreover, let T represent a set of partial directed reduced spanning trees at each iteration of the algorithm. Further, let $max\_node\_num(P)$ return the maximum number associated with a node in the given set of nodes P, and let $num(n_i)$ return the node number of the node n_i. Also, let $adj(e_{ji})$ return the adjacent incoming edge to the edge e_{ji} at node n_i. Finally, let $mult(e_{ji})$ return the multiplier coefficient m_{ji} along the edge e_{ji}. Then, the algorithm for directed reduced spanning tree enumeration is as follows:

Algorithm 9

read the directed reduced SFG with nodes S;
read the root node r of the directed reduced SFG;
begin
Step 1:
$T = \emptyset$; $V = \{r\}$; $U = S - V$;
Step 2:
Find node $n_i \in U$ such that
$num(n_i) == max\_node\_num(V) + 1$;
Step 3:
Add n_i to every $t_i \in T$ separately for each edge e_{ji} (with $mult(e_{ji}) \neq 0$)
from V to n_i;
/* If T had k trees, it now has k or $2k$ trees (forward direction). */
Step 4:
/* Now for the trees generated in the backward direction. */
for (every edge e_{ip} (with $mult(e_{ip}) \neq 0$) from n_i to V) do
begin
$T_{temp} = \emptyset$
for (every tree $t_j \in T$) do

```
        begin
            t_temp = t_j;
            if ((t_temp has no edge e_ip) && (adj(e_ip) does not arise from n_i))
                t_temp = t_temp + e_ip − adj(e_ip);
            if (t_temp is a valid directed reduced spanning tree)
                T_temp = T_temp + {t_temp};
        end
        T = T + T_temp;
    end
Step 5:
    V = V + {n_i}; U = U − {n_i};
Step 6:
    if (U ≠ ∅)
        repeat steps 2 through 6;
end.
```

4.4.2 Proof of Operation

The proof of working of Algorithm 9 (Page 115) is established through the following theorem.

Theorem 4.8 *The application of Algorithm 9 (Page 115) to a directed reduced SFG results in a set T of all the possible directed reduced spanning trees of that directed reduced SFG.*

Proof The proof is deferred until Lemmas 7, 8, and 9 are established. ■

The following lemma deals with the directed reduced spanning trees of the set $V + \{n_i\}$ with the newly added node n_i as the leaf node.

Lemma 7 *The addition of the node n_i in Step 3 of Algorithm 9 (Page 115) results in all the possible partial directed reduced spanning trees associated with the set $V + \{n_i\}$ having n_i as one of the leaf nodes.*

Proof By contradiction. Let $t_i \in T$ for $i = 1, 2, \ldots, p$ represent all the possible, say p, partial directed reduced spanning trees associated with the visited set V. Then, after the application of Step 3 of Algorithm 9 (Page 115), one obtains the new set T of either p new partial directed reduced spanning trees if only one directed edge connects V to n_i, or $2p$ partial directed reduced spanning trees if two directed edges connect V to n_i. If the new set T is not the exhaustive set of partial directed reduced spanning trees associated with $V + \{n_i\}$ having n_i as one of the leaf nodes, then three cases can arise. Either n_i is not connected directly to V, or T is not the exhaustive set of partial directed reduced spanning trees associated with the set V, or both. The former case is not possible since n_i is numbered one higher than the $max\_node\_num(V)$, meaning that n_i is directly connected to V. The latter case leads to the violation of the assumption that T is the exhaustive set of partial directed reduced spanning trees of the set V before the application of Step 3. All these lead to a contradiction, completing the proof. ■

Lemma 8 *The addition of an outgoing edge e_{i1} from the node n_i to V in Step 4 of Algorithm 9 (Page 115) results in all the possible partial directed reduced spanning trees associated with the set $V + \{n_i\}$, having n_i as one of the non-terminating nodes, and having a single outgoing edge e_{i1} from n_i to V.*

Proof By contradiction. At the beginning of Step 4 (i.e. after the completion of Step 3), the set T contains p (or $2p$) partial directed reduced spanning trees associated with the set $V + \{n_i\}$. By applying the inner *for* loop of Step 4 for all $t_j \in T$, one obtains the set T_{temp} of partial directed reduced spanning trees, where each $t_l \in T_{temp}$ contains n_i as the non-leaf node having e_{i1} as the corresponding outgoing edge. Assume that there exists a tree t_l such that

1. t_l is a partial directed reduced spanning tree associated with the set $V + \{n_i\}$,

2. t_l contains n_i as a non-leaf node having e_{i1} as the corresponding outgoing edge towards the set V, and

3. $t_l \notin T_{temp}$

Then, Step 4 of Algorithm 9 (Page 115) leads to any, some, or all of the following conditions. *a)* e_{i1} does not connect n_i to V, *b)* e_{i1} has no adjacent incoming edge in the set $V + \{n_i\}$, or *c)* T is not an exhaustive set of partial directed reduced spanning trees of V. Condition a violates our assumption regarding the tree t_l. Condition b means that the set V has an unconnected node in it, leading to a contradiction. Condition c leads to a direct contradiction by using Lemma 7. This completes the proof. ■

The above lemmas can be used to prove that the addition of a new node to an already existing set partial directed reduced spanning trees of nodes V in accordance with Algorithm 9 (Page 115) results in all the possible directed reduced spanning trees of $V + \{n_i\}$.

Lemma 9 *The addition of arbitrary edges $e_{i1}, e_{i2}, \ldots, e_{ik}$ from node n_i to the set V in Step 4 of Algorithm 9 (Page 115), results in all the possible partial directed reduced spanning trees associated with the set $V + \{n_i\}$.*

Proof From Lemma 7 and 8, one can obtain an exhaustive set of two types of partial directed reduced spanning trees of the set $V + \{n_i\}$. The first type has n_i as the leaf node, and the second type has n_i as the non-leaf node and also has e_{i1} as its corresponding outgoing edge.

Let e_{i2} be added to connect n_i to V. Then by applying Step 4 of Algorithm 9 (Page 115) for e_{i2}, and reasoning along the lines as for Lemma 8, one obtains the set T_{temp} of two types of partial directed reduced spanning trees. The first type has e_{i2} as the only outgoing edge from the node n_i to the set V, and the second type has both

e_{i1} and e_{i2} as the outgoing edges from the node n_i to the set V. Therefore, the set $T + T_{temp}$ is an exhaustive set of partial directed reduced spanning trees associated with the set $V + \{n_i\}$ having the edges e_{i1} and e_{i2}.

In this way, one can show that the addition of the edges $e_{i3}, \ldots, e_{ik}$ leads to all the possible partial directed reduced spanning trees associated with the set $V + \{n_i\}$ having $e_{i1}, e_{i2}, \ldots, e_{ik}$ as their corresponding outgoing edges. This completes the proof. ■

Finally, one can use the above lemmas to establish the proof of Theorem 4.8.

Proof of Theorem 4.8: By induction. Consider a section of a directed reduced SFG with nodes r, n_1, and n_2, numbered as 0, 1, and 2, respectively, as shown in the partial SFGs in Fig. 4.6. These SFGs show all the possible ways of connecting the nodes r, n_1, and n_2.

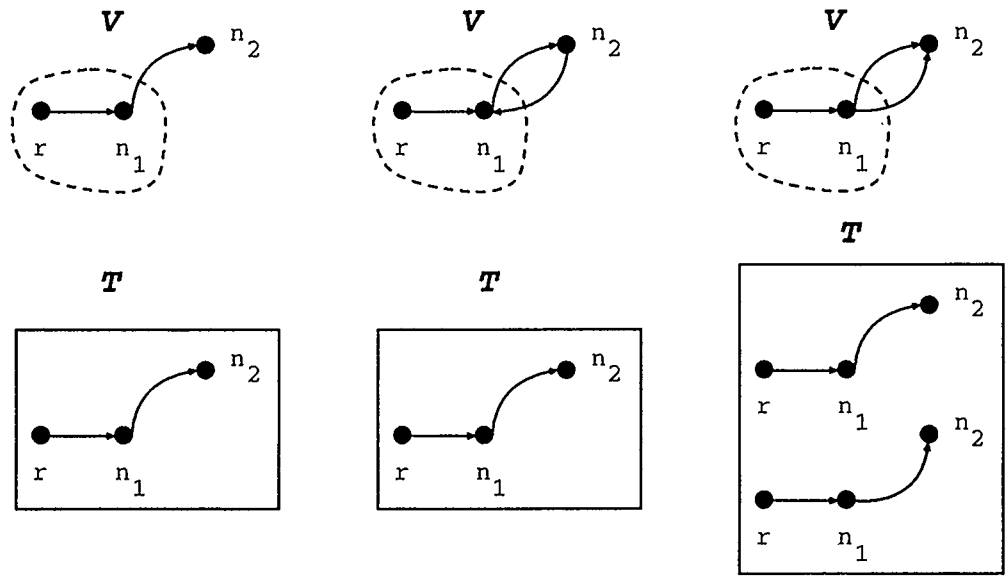

Figure 4.6. Schematic for Proof of Theorem 4.8

Since r connects only to n_1, after the first iteration of Algorithm 9 (Page 115) one obtains, $T = \{t_1\}$, $V = \{r, n_1\}$, and $U = S - V$. Here, t_1 is the only possible partial

directed reduced spanning tree associated with the visited set V. The addition of n_2 to the set V in the second iteration of Algorithm 9 (Page 115) results in the updated set T of one or two trees as shown in Fig. 4.6. It is clear that the set T still remains an exhaustive set of partial directed reduced spanning trees associated with the set V through the first and second iterations.

Assuming that T is the exhaustive set of directed reduced spanning trees for $V = \{r, n_1, n_2, \ldots, n_{k-1}\}$, the application of Algorithm 9 (Page 115) to n_k in conjunction with Lemma 9 results in the updated set of trees T which contains all possible partial directed reduced spanning trees of the set $V = \{r, n_1, n_2, \ldots, n_{k-1}, n_k\}$. This completes the proof. ■

4.4.3 Algorithm for Enumerative MAC-Modularization

Algorithm 9 (Page 115) is used for the design of an algorithm for enumerative MAC-modularization as follows.

Algorithm 10

```
begin
   input: The digital filter SFG;
   convert the digital filter SFG to G(V,E);
   /* G(V,E) is the directed reduced SFG */
   Determine all n directed reduced spanning trees of G(V,E) given by
      H_k(V,E_k), k = 1,2,...,n in accordance with Algorithm 9 (Page 115);
   ∀ H_k(V,E_k) do
      select the best H_p(V,E_p) based on Eqn. (4.6);
   MAC-modularize the digital filter SFG using H_p(V,E_p), Algorithm 7
      (Page 107), and Algorithm 8 (Page 107);
end.
```

4.5 A Heuristic Approach to MAC-Modularization

This section presents an heuristic approach to MAC-modularization for problems having medium to large sizes. This heuristic operates on the basis of a locally optimal edge selection based on the fitness function. This heuristic is shown to generate near-optimal solutions very rapidly.

4.5.1 Algorithm for Heuristic Spanning Tree Generation

Let S represent the set of all nodes in a given directed reduced SFG. At each iteration of the following algorithm, let the set S be partitioned into two sets, namely V signifying the set of visited nodes, and U signifying the set of unvisited nodes. Moreover, let $H_{heur}(V, E_{heur})$ represent the heuristically obtained partial directed reduced spanning tree at each iteration of the algorithm. Further, let $edge\_fitness(e_{ij}, H_{heur})$ represent the fitness index of the edge e_{ij} due to the movement of the multiplier m_{ij} from the node $n_i \in V$ to the node $n_j \in U$, by using the partial directed reduced spanning tree $H_{heur}(V, E_{heur})$, given by

$$edge\_fitness(e_{ij}, H_{heur}(V, E_{heur})) = \frac{1}{(oflow\_weight \times O'_j) + (roff\_weight \times R'_{start\_node(adj(e_{ij})),j})} \tag{4.19}$$

Then, the algorithm for heuristically obtaining a near-optimal directed reduced spanning tree is given by the following pseudo-code.

Algorithm 11

```
read the directed reduced SFG with nodes S;
read the root node r of the directed reduced SFG;
begin
   U = S − {r}; V = {r}; D = E = {};
   construct H_heur(V, E);
   while (U ≠ ∅) do
   begin
```

```
        ∀ (n_i ∈ U) do
        if ((∃ n_x ∈ V) && (n_x == predecessor(n_i)))
            D = D + {(n_x, n_i)};
        select an arbitrary edge e_αβ ∈ D;
        D = D − {e_αβ};
        while (D ≠ ∅) do
        begin
            select an arbitrary edge e_pq ∈ D;
            if (edge_fitness(e_pq, H_heur(V, E)) > edge_fitness(e_αβ, H_heur(V, E)))
                e_αβ = e_pq;
            D = D − {e_pq};
        end
        /* n_β is the terminating node of the resulting e_αβ */
        U = U − {n_β};
        V = V + {n_β};
        E = E + {e_αβ};
        construct H_heur(V, E);
    end
    output H_heur(V, E);
end.
```

The above algorithm progresses successively from the root (input) node onwards, and at each iteration, updates the set of visited nodes V by adding the node $n_\beta \in U$ which is connected from V to U with the edge having the best edge fitness in accordance with Eqn. (4.19). Therefore, at each iteration of the algorithm, a new locally optimal edge (and its corresponding node) is added to the already existing partial directed reduced spanning tree $H_{heur}(V, E)$. The final resulting heuristically obtained directed reduced spanning tree $H_{heur}(V, E_{heur})$ is the locally optimal solution, and can be employed for MAC-modularization of the directed reduced SFG as shown in the following algorithm.

4.5.2 Algorithm for Heuristic MAC-Modularization

Algorithm 11 (Page 121) is used for the design of an algorithm for heuristic MAC-modularization as follows.

Algorithm 12

```
begin
   input: The digital filter SFG;
   convert the digital filter SFG to G(V,E);
   /* G(V,E) is the directed reduced SFG */
   Use Algorithm 11 (Page 121) to determine H_heur(V,E);
   MAC-modularize the digital filter SFG using H_heur(V,E), Algorithm 7
      (Page 107), and Algorithm 8 (Page 107);
end.
```

4.6 Implementation of MAC-Modularization Algorithms

Algorithms 10 and 12 (Pages 120 and 123) have been applied successfully to the design and development of a software package called **MAC-Modularizer (MAC-M)** for automated MAC-modularization of digital filter SFGs. This package has been developed by employing an object-oriented design strategy and has been implemented using the C++ programming language. **MAC-M** has been applied to a variety of digital filters having SFGs with dense or sparse multiplication operations. **MAC-M** consists of two modules:

1. *Enum module:* In this module, Algorithm 10 (Page 120) is used to exhaustively generate all the possible solutions and then select the optimal solution. This module provides the user with a list of all the possible solutions together with the L1-norm values for the possible degree of overflow and roundoff noise. Further, it can identify the optimal and the worst solutions.

2. *Heur module:* In this module, Algorithm 12 (Page 123) is used generate the heuristic solution rapidly. This module also provides the L1-norm values for the possible degree of overflow and roundoff noise in the heuristic solution.

4.6.1 Test Cases

In the following, the usefulness of **MAC-M** is demonstrated through its application to the MAC-modularization of a pair of digital filters, one having dense, and another sparse multiplication operations.

1. *Case 1:* This case constitutes a digital low-pass filter in [21] having a SFG with dense multiplication operations. The low-pass filter schematic is shown in Fig. 4.7. The corresponding optimal MAC-modularized realization is shown in the schematic in Fig. 4.8. The *Enum* module generated 25600 solutions for this case, taking a CPU time of approximately 14 minutes on a Sun SPARC-10. The *Heur* module generated the heuristic solution within approximately 5 seconds. In this case, the fitness functions (c.f. Eqns. (4.6) and ((4.19)) were evaluated by using $oflow\_weight = roff\_weight = 1.0$.

2. *Case 2:* This case constitutes a digital LDI-Jaumann low-pass filter in [6] having a SFG with sparse multiplication operations. The LDI-Jaumann low-pass filter schematic is shown in Fig. 4.9. The corresponding optimal MAC-modularized realization shown in the schematic in Fig. 4.10. The *Enum* module generated 4612 solutions for this case, taking a CPU time of approximately 5 minutes on a Sun SPARC-10. The *Heur* module generated the heuristic solution within approximately 3 seconds. In this case, the fitness functions were again evaluated by using $oflow\_weight = roff\_weight = 1.0$.

A comparison of the solutions obtained by using **MAC-M** for Cases 1 and 2 in terms of the L1-Norm measures of their overflow and roundoff noise values is as shown

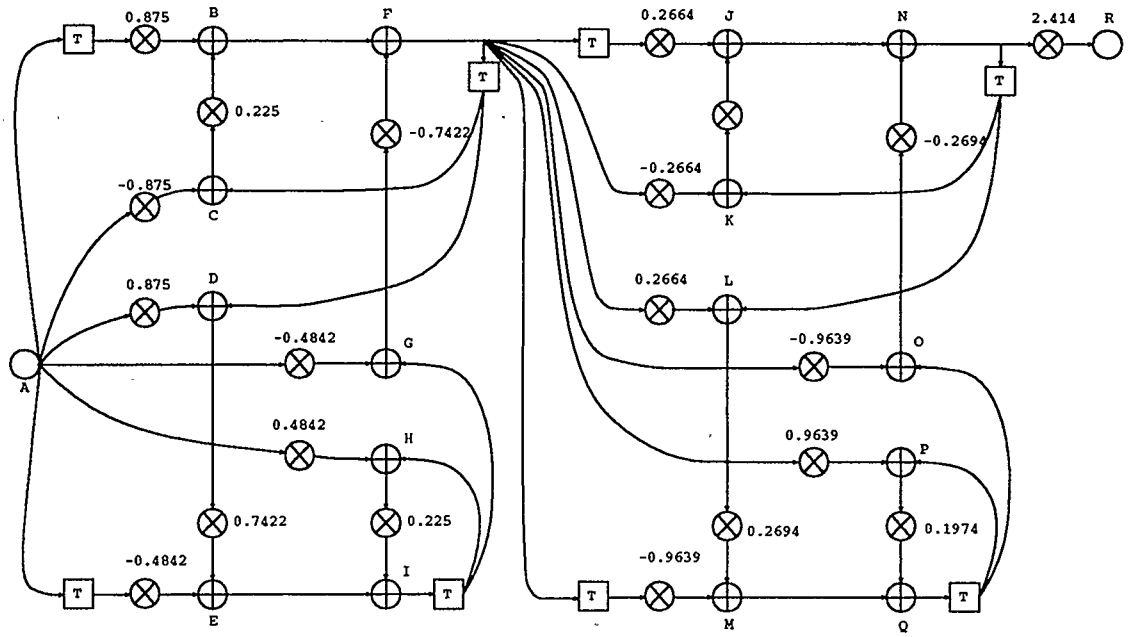

Figure 4.7. Case 1: Unmodularized Original digital filter SFG

in Tables 4.1 and 4.2, respectively. It is observed from the results that the optimal and worst solutions differ greatly in their fitness values, and that the heuristic algorithm produces a near-optimal result.

Table 4.1. Comparison of solutions for Case 1

L1-Norm Values	*Optimal*	*Worst*	*Heuristic*
Overflow	8.79888	5586.37	19.9865
Roundoff	10.8912	7.67213	10.7343
Total	19.69008	5594.04213	30.7208

4.7 Chapter Summary

This chapter has presented a rigorous theoretical approach to the MAC-modularization of digital filter SFGs.

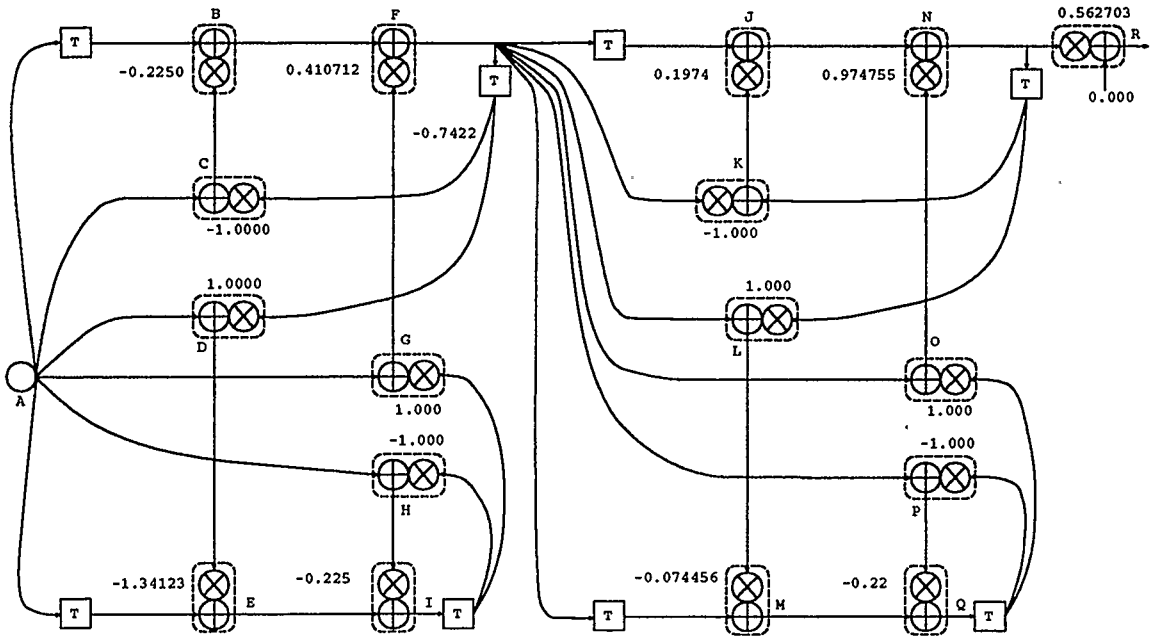

Figure 4.8. Case 1: Optimal MAC-Modularized digital filter SFG

Table 4.2. Comparison of solutions for Case 2

L1-Norm Values	*Optimal*	*Worst*	*Heuristic*
Overflow	5.53031	183.289	5.53031
Roundoff	13.4639	6.7878	14.8879
Total	18.99421	190.0768	20.41821

Section 4.2 has dealt with the theoretical basis and techniques for MAC-modularization of digital filter SFGs. This includes graph-theoretic techniques for MAC-modularization using the directed reduced SFG and the directed reduced spanning trees. These techniques have subsequently been translated into algorithms for MAC-modularization.

In Section 4.3, it has been recognized that the above techniques result in several solutions for a given digital filter SFG. This has necessitated the development of a fitness function for the selection of the optimal solution. The fitness function has been

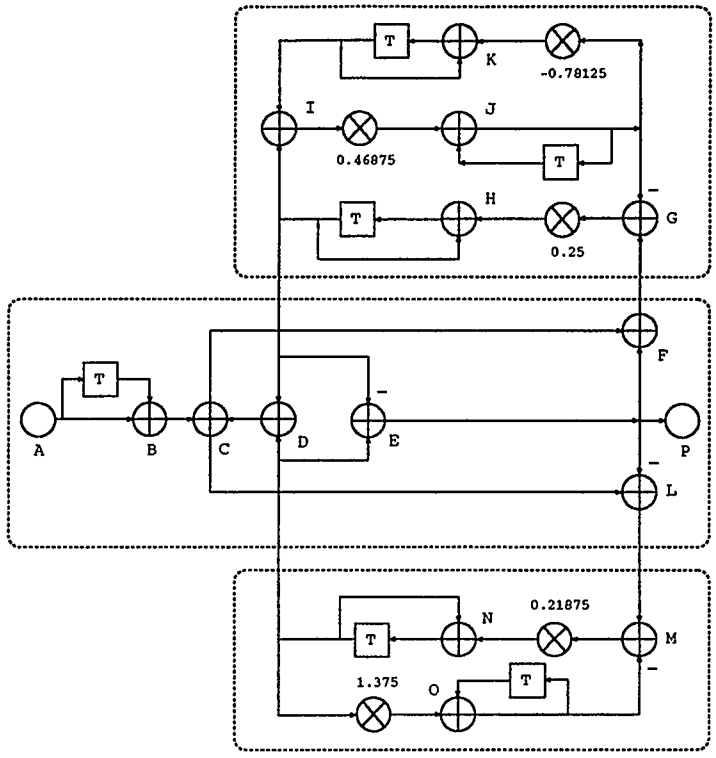

Figure 4.9. Case 2: Unmodularized Original digital filter SFG

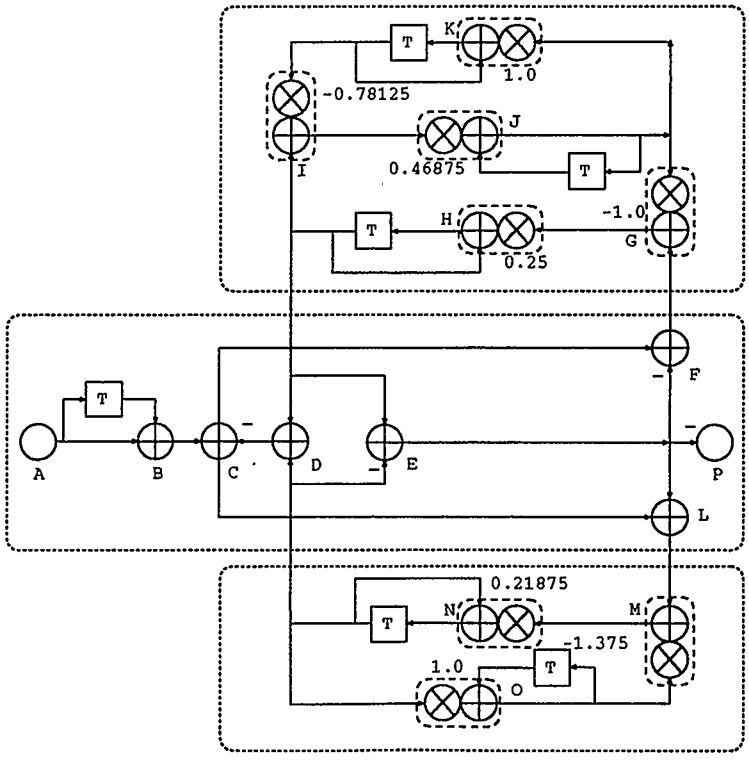

Figure 4.10. Case 2: Optimal MAC-Modularized digital filter SFG

developed based on the finite-precision arithmetic effects of overflow and roundoff noise.

Section 4.4 has presented an enumerative technique for MAC-modularization. This technique permits the generation of all the directed reduced spanning trees of a given directed reduced SFG, facilitating an exhaustive consideration of all the possible MAC-modularized digital filter SFGs.

The above enumerative technique is suitable for small and medium size problems. However, large problems are computationally intractable because of the rapid enlargement of the solution space. This has led to the development of a corresponding heuristic technique in Section 4.5 for MAC-modularization. This technique has been based on local application of the fitness function in order to generate a near optimal solution, reducing the required computational effort substantially.

Finally, the enumerative and heuristic techniques have been used for the development of a powerful object-oriented MAC-modularization software package called **MAC-M** in Section 4.6. The usefulness of **MAC-M** has been demonstrated through its application to a pair of digital filters, one having dense, and the other sparse multiplication operations. It has also been demonstrated that the heuristic technique produces a near-optimal result.

CHAPTER 5

DESIGN AND IMPLEMENTATION OF A REDUNDANT NUMBER MAC-MODULARIZED LDI JAUMANN DIGITAL FILTER

5.1 Introduction

This chapter is concerned with an illustration of the results obtained in Chapters 3 and 4, and their application to the design and gate level implementation of a practical LDI [28] Jaumann [6] digital filter.

The initial step in the above design involves MAC-modularization of the digital filter which yields a uniform and modular structure, lending itself to efficient VLSI implementation. The next step involves the exploitation of the inherent parallelism in the Jaumann digital filter structure by identifying subsystems which can operate concurrently. This is followed by the development of the state update equations for characterizing the operation of the overall filter in terms of the identified subsystems.

The digital filter obtained after MAC-modularization may suffer from the harmful effects of finite-precision arithmetic, namely, the transfer function errors (linear phenomenon), and the overflow saturation and roundoff noise (non-linear phenomena). Transfer function errors are brought within acceptable limits by selecting the appropriate MAC coefficient wordlengths. Overflow saturation is counteracted by adding upper guard bits/digits to the signal word while roundoff noise is counteracted by adding lower guard bits/digits. This is followed by the selection of the proper number representations for the signal and MAC coefficient words in order to maximize the operational speed of the digital filter.

The state update equations involving finite-precision arithmetic are subsequently translated into a gate-level implementation of the digital filter. This consists of separate data-paths for the implementation of each of the subsystems together with a

global control unit. The implementation is simplified due to the modularity in the filter structure resulting from MAC-modularization. The constituent MAC architectures are implemented using redundant number arithmetic in order to exploit the maximum speed available by the given implementation technology.

Section 5.2 deals with the MAC-modularization of a fifth-order lowpass LDI Jaumann digital filter [25], together with the development of the corresponding state update equations for the constituent subsystems. The signal and MAC coefficient wordlengths required for the finite-precision arithmetic implementation of the MAC-modularized LDI Jaumann digital filter are calculated in Section 5.3. Section 5.4 presents the number representations adopted for the signal and MAC coefficient words. The data-path and control unit implementations for the Jaumann digital filter are presented in Section 5.5. In Section 5.6, the resulting Actel 1.2μ technology implementation is verified by using its impulse response Viewlogic simulations. The simulation results demonstrate an achievable clock rate of 50 MHz yielding a sample rate of 8.33 MHz. The striking feature of this implementation is that its speed of operation is completely independent of the signal wordlength within the digital filter.

5.2 MAC-Modularization of the LDI Jaumann Digital Filter

The schematic in Fig. 5.1 shows the SFG of a fifth-order lowpass LDI Jaumann digital filter [6]. This Jaumann digital filter can be visualized as consisting of three different subsystems, namely, the top and bottom subsystems $\mathcal{N}_1$ and $\mathcal{N}_2$ incorporating forward and backward Euler digital integrators, and the central interconnection subsystem $\mathcal{N}_0$. The top subsystem $\mathcal{N}_1$ contains 3 digital integrators, and the bottom subsystem $\mathcal{N}_2$ contains 2 digital integrators.

The digital filter in Fig. 5.1 has an important property of offering a high degree of parallelism in the constituent arithmetic operations. There are several ways to exploit this parallelism for achieving a corresponding high-speed implementation. For exam-

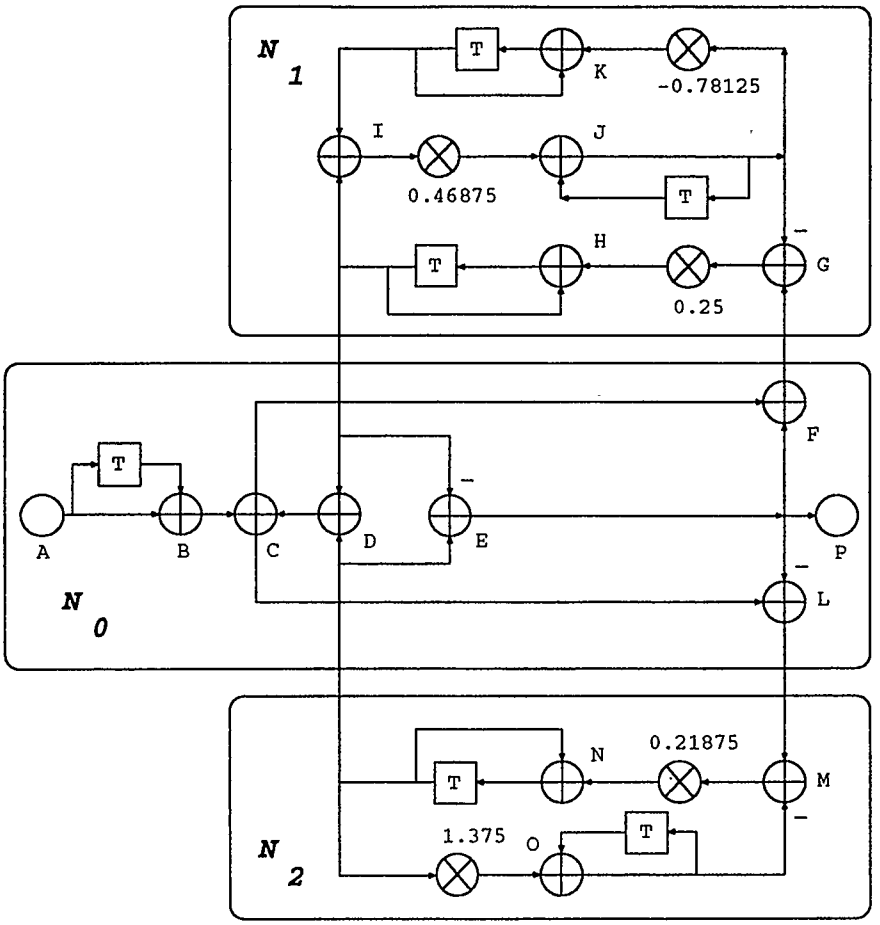

Figure 5.1. Lowpass LDI Jaumann Digital Filter before MAC-Modularization

ple, the top and bottom subsystems $\mathcal{N}_1$ and $\mathcal{N}_2$ can operate concurrently, permitting separate data-path implementations.

The MAC-modularization of the digital filter in Fig. 5.1 is carried out by using the software package MAC-M introduced in Chapter 4. The resulting optimal MAC-modularized SFG is shown in the schematic diagram in Fig. 5.2. In this way, the subsystems $\mathcal{N}_0$, $\mathcal{N}_1$, and $\mathcal{N}_2$ are translated to the new subsystems $\widehat{\mathcal{N}_0}$, $\widehat{\mathcal{N}_1}$, and $\widehat{\mathcal{N}_2}$, respectively. The subsystems $\widehat{\mathcal{N}_1}$ and $\widehat{\mathcal{N}_2}$ consist of 5 and 3 MAC operations, respectively. In order to achieve the maximum possible operational speed, the filter is implementing by using two pieces of hardware redundant number arithmetic MAC units. One MAC unit is dedicated to the subsystem $\widehat{\mathcal{N}_1}$, and the other to the

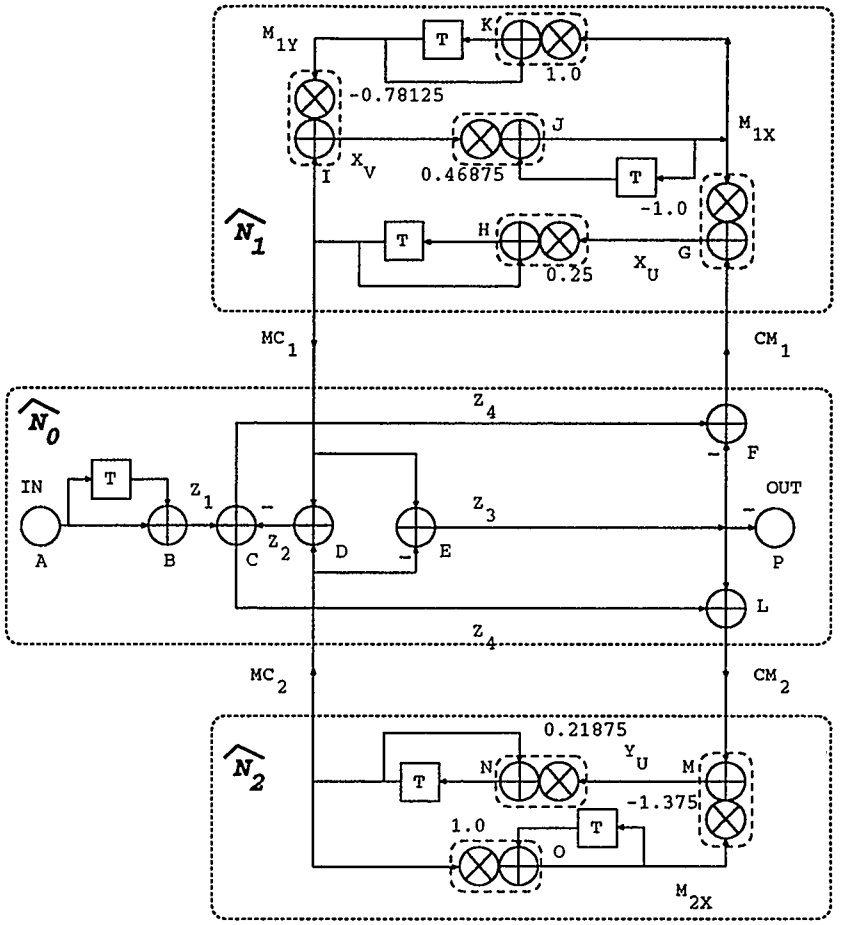

Figure 5.2. Optimal MAC-Modularized Lowpass LDI Jaumann Digital Filter

subsystem $\widehat{\mathcal{N}_2}$.

It is observed that the process of MAC-modularization maintains all the MAC coefficients in the subsystem $\widehat{\mathcal{N}_0}$ to either $+1$ or -1. Therefore, the resulting subsystem is implemented by using a single redundant arithmetic combinational unit.

The state variables associated with the MAC-modularized digital filter are shown in Fig. 5.2. The state variables CM_1 and CM_2 are used for communicating from $\widehat{\mathcal{N}_0}$ to $\widehat{\mathcal{N}_1}$ and $\widehat{\mathcal{N}_0}$ to $\widehat{\mathcal{N}_2}$, respectively. Similarly, the state variables MC_1 and MC_2 are used for communicating from $\widehat{\mathcal{N}_1}$ to $\widehat{\mathcal{N}_0}$ and $\widehat{\mathcal{N}_2}$ to $\widehat{\mathcal{N}_0}$, respectively. The input to the filter is represented by the variable IN, and its output is represented by the variable OUT. In addition, each subsystem is also associated with a corresponding

set of internal variables. The subsystem $\widehat{\mathcal{N}_0}$ has Z_1, Z_2, Z_3, and Z_4 as its internal set of variables. The subsystem $\widehat{\mathcal{N}_1}$ has M_{1X}, M_{1Y}, X_U, and X_V as its internal set of variables. Similarly, the subsystem $\widehat{\mathcal{N}_2}$ has M_{2X} and Y_U as its internal set of variables.

The relationships between the above variables are expressed in terms of state update equations which describe the operation of the overall MAC-modularized digital filter. The state update equations for this digital filter in terms of those of $\widehat{\mathcal{N}_0}$, $\widehat{\mathcal{N}_1}$, and $\widehat{\mathcal{N}_2}$ are combined and presented in the following algorithm.

Algorithm 13

input: IN;

output: OUT;

begin

/* The inputs to the subsystems */

input MC_1^n, MC_2^n, IN to $\widehat{\mathcal{N}_0}$;

input CM_1^n, MC_1^n, M_{1X}^n, M_{1Y}^n to $\widehat{\mathcal{N}_1}$;

input CM_2^n, MC_2^n, M_{2X}^n to $\widehat{\mathcal{N}_2}$;

/* Equations characterizing the operation of subsystem $\widehat{\mathcal{N}_0}$ */

compute Z_1 using $Z_1 = IN^n + IN^{n-1}$;

compute Z_2 using $Z_2 = MC_1^n + MC_2^n$;

compute Z_3 using $Z_3 = MC_1^n - MC_2^n$;

compute Z_4 using $Z_4 = Z_1 - Z_2$;

compute CM_1^{n+1} using $CM_1^{n+1} = Z_4 + Z_3$

compute CM_2^{n+1} using $CM_2^{n+1} = Z_4 - Z_3$

compute OUT using $OUT = \overline{Z_3}$;

/* State update equations characterizing the operation of subsystem $\widehat{\mathcal{N}_1}$ */

compute X_V using $X_V = MC_1^n + (-0.78125)M_{1Y}^n$;

compute M_{1X}^{n+1} using $M_{1X}^{n+1} = M_{1X}^{n} + (0.46875)X_V$;

compute X_U using $X_U = CM_1^n + (-1.0)M_{1X}^n$;

compute M_{1Y}^{n+1} using $M_{1Y}^{n+1} = M_{1Y}^{n} + (1.0)M_{1X}^n$;

compute MC_1^{n+1} using $MC_1^{n+1} = MC_1^n + (0.25)X_U$;

/* State update equations characterizing the operation of subsystem $\widehat{\mathcal{N}_2}$ */

compute M_{2X}^{n+1} using $M_{2X}^{n+1} = M_{2X}^{n} + (1.0)MC_2^n$;

compute Y_U using $Y_U = CM_2^n + (-1.375)M_{2X}^n$;

compute MC_2^{n+1} using $MC_2^{n+1} = MC_2^n + (0.21875)Y_U$;

/* The outputs of the subsystems */

output CM_1^{n+1}, CM_2^{n+1}, OUT from $\widehat{\mathcal{N}_0}$;

output MC_1^{n+1} from $\widehat{\mathcal{N}_1}$;

output MC_2^{n+1} from $\widehat{\mathcal{N}_2}$;

write OUT;

end.

In the above algorithm, the superscript $n-1$, n, or $n+1$ associated with any variable indicates the value of that variable in the $n-1^{th}$, n^{th}, or $n+1^{th}$ operation cycle.

5.3 Calculation of the Required MAC Coefficient and Signal Wordlengths

In this section, the harmful effects of finite-precision arithmetic are investigated for the calculation of the required wordlengths for the MAC coefficients and the internal signals.

5.3.1 Calculation of the Required MAC Coefficient Wordlength

In a fixed-point digital filter implementation, the values of the constituent MAC coefficients must be quantized to a finite wordlength. The quantized MAC coefficients give rise to errors in the time-domain and the frequency-domain response of the digital filter. Therefore, the MAC coefficient wordlengths must be determined in such a manner that the above errors are confined to acceptable limits.

The lowpass LDI Jaumann digital filter exhibits exceptionally low passband sensitivity [25] with respect to coefficient quantization errors. Therefore, it is possible to quantize the MAC coefficient values to only 6 bits/digits as demonstrated in [25] and [15].

5.3.2 Calculation of the Required Signal Wordlength

Let $O'_{i_{MAX}}$ represent the L1-norm value of the maximum signal gain from the digital filter input to any node within the filter and let $\sum R'_i$ represent the sum of the L1-norm values of the signal gains from each MAC node (having a coefficient other than $+1$ or -1) to the output node of the digital filter. Furthermore, let l_{Signal} represent the input signal wordlength, and let l_{Upper} and l_{Lower} represent the upper and lower guard bits/digits required to counteract the effect of overflow and roundoff noise, respectively. Then, l_{Upper} can be determined by using $O'_{i_{MAX}}$ in accordance with

$$l_{Upper} = \lceil log_2(O'_{i_{MAX}}) \rceil. \tag{5.1}$$

Similarly, l_{Lower} can be determined by using $\sum R'_i$ in accordance with

$$l_{Lower} = \begin{cases} \lceil log_2(\sum R'_i + g') \rceil & \text{for truncation} \\ \lceil log_2(\frac{1}{2}\sum R'_i + g') \rceil & \text{for rounding,} \end{cases} \tag{5.2}$$

where g' is determined in terms of the input to output gain g_{io} as

$$g' = \begin{cases} g_{io} & \text{if the filter output is obtained by truncation} \\ \frac{1}{2}g_{io} & \text{if the filter output is obtained by rounding.} \end{cases} \tag{5.3}$$

The minimum required internal wordlength of the filter $l_{Internal}$ is determined by using Eqns. (5.1) and (5.2) as

$$l_{Internal} = l_{Upper} + l_{Signal} + l_{Lower}, \tag{5.4}$$

where l_{Upper} and l_{Lower} add to the most-significant and least-significant part of the signal word, respectively.

For the optimal MAC-modularized lowpass LDI Jaumann digital filter shown in Fig. 5.2, $O'_{i_{MAX}} = 5.53031$, $\sum R'_i = 13.4639$, and $g_{io} = 2.02602$. By using Eqns. (5.1) and (5.2) one obtains $l_{Upper} = 3$, and $l_{Lower} = 4$. Since the signal wordlength is given to be $l_{Signal} = 12$ bits/digits, Eqn. (5.4) yields $l_{Internal} = 19$ bits/digits.

5.4 Number Representation of the Signal and MAC Coefficient

This section presents the number representation adopted for the signal and coefficient word.

5.4.1 Representation of the Signal Word

In this implementation, the signal is represented by using SB number representation (c.f. Section 2.3). The SB signal is represented in two-level logic using the minimal-(n, p) encoding (c.f. Table 2.2). This has two advantages.

1. It permits simple two-level logic realization, and

2. It permits simple and straightforward negation of the signal word.

The latter can be achieved by simply swapping the n and p bits at each digit position without incurring any hardware cost. SBNR is employed because it eliminates carry/borrow propagation in addition/subtraction, thereby permitting the digital filter to operate at a very high speed independently of the signal wordlength.

5.4.2 Representation of the MAC Coefficient

Algorithm 1 (Page 28) is employed to convert the given MAC coefficients into their corresponding modified radix-4 representation (c.f. Section 2.4.1). The result is encoded in two-level logic by using Eqn. (2.61). Such a representation of the MAC coefficients eliminates the need for the modified radix-4 recoders in the corresponding MAC architectures leading to an increase in the maximum achievable speed in the resulting implementation. Moreover, this representation also permits multiplication by unity, which is otherwise not possible using TCNR.

5.5 Gate-Level Implementation of the MAC-Modularized LDI Jaumann Digital Filter

The optimal MAC-modularized digital filter in Fig. 5.2 is used together with Algorithm 13 (Page 133) to arrive at gate-level implementation of the digital filter. The resulting implementation can be visualized as consisting of three separate data-paths implementing the subsystems $\widehat{\mathcal{N}_0}$, $\widehat{\mathcal{N}_1}$, and $\widehat{\mathcal{N}_2}$, and a global control unit as shown in the schematic in Fig. 5.3. The data-path subsystems communicate by using the data buses MC_1, MC_2, CM_1, and CM_2 which consist of the 19×2-bit wide signals associated with the corresponding state variables in accordance with Algorithm 13 (Page 133).

The global control unit controls the operation of the digital filter by communicating with each of the data-path subsystems through the *Control Bus*, *COEFF1 Bus*, and *COEFF2 Bus*. The *Control Bus* transmits the digital filter control word from the control unit to the data-path subsystems. This control word carries the information regarding the state updates required in each clock cycle of the digital filter operation. The details regarding the control word are discussed in subsection 5.5.2. The *COEFF1 Bus* and *COEFF2 Bus* carry the 3×3-bit wide signals associated with the MAC coefficients for the subsystems $\widehat{\mathcal{N}_1}$ and $\widehat{\mathcal{N}_2}$, respectively.

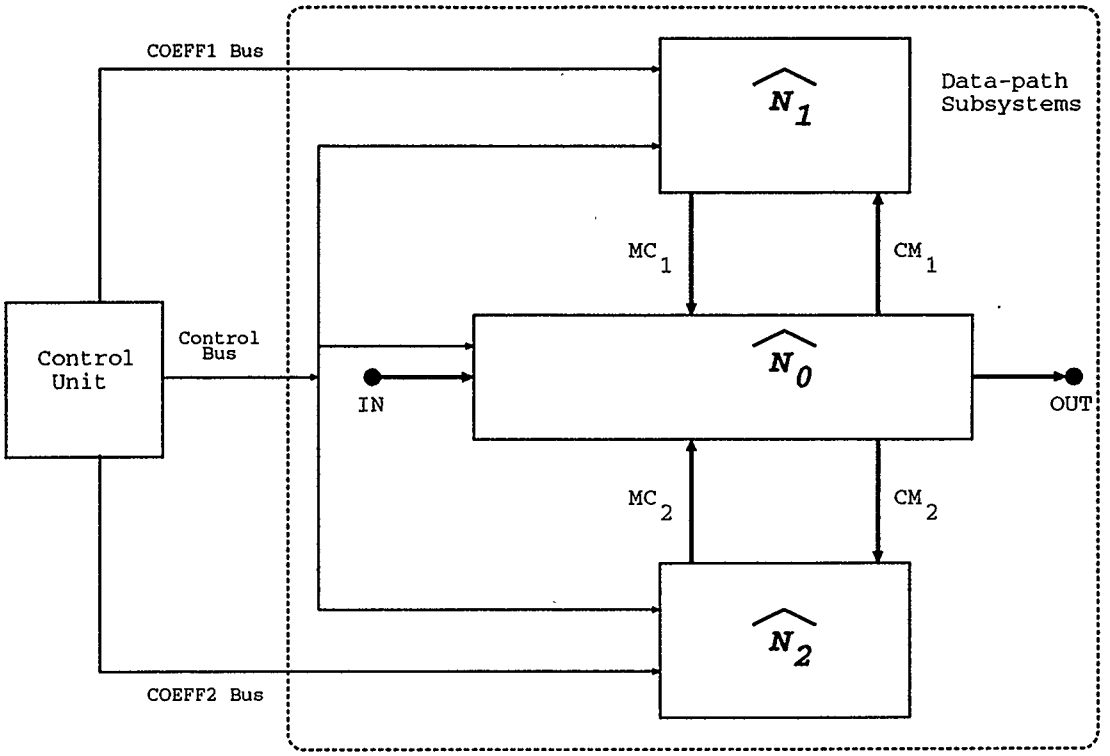

Figure 5.3. Architecture of the MAC-Modularized Lowpass LDI Jaumann Digital Filter

The implementation of the data-path subsystems and control unit by using the basic hardware cells given in [45] is discussed in the following.

5.5.1 Gate-Level Implementation of the Data-Path Subsystems

The implementation of the data-path subsystems $\widehat{\mathcal{N}_0}$, $\widehat{\mathcal{N}_1}$, and $\widehat{\mathcal{N}_2}$, is shown in the schematic in Fig. 5.4. This implementation is developed by exploiting the corresponding set of state update equations given in Algorithm 13 (Page 133). The heavy lines in the schematic diagram in Fig. 5.4 represent the data buses associated with the state variables, while the other lines represent the control signals. The labels CLK and CLR represent the clock and reset signals to the data-path subsystems, respectively.

The implementation of the $\widehat{\mathcal{N}_0}$ data-path subsystem is shown in the central portion

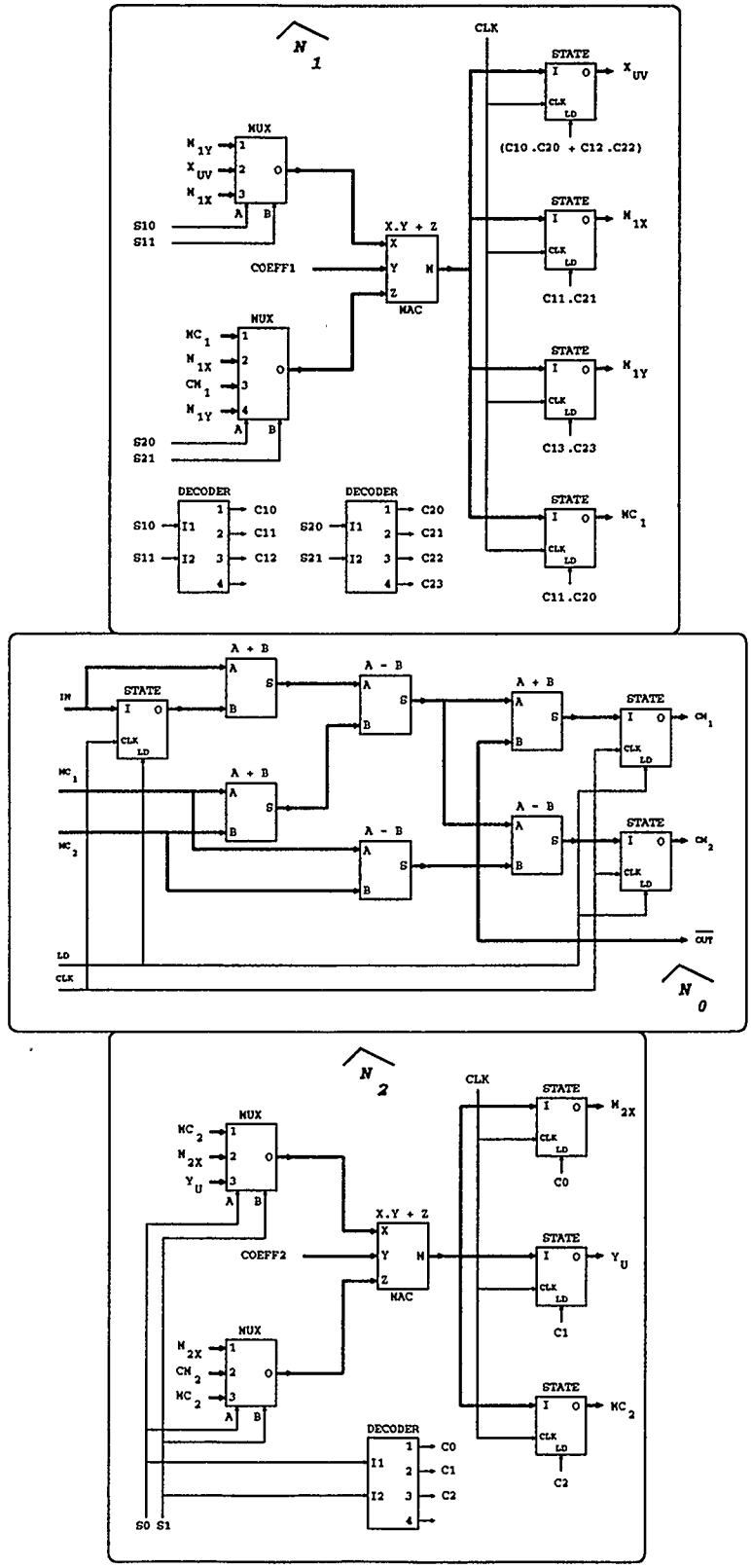

Figure 5.4. Implementation of the Data-Path Subsystems

of the schematic in Fig. 5.4. The input signals to this subsystem are IN, MC_1, and MC_2. The output signals generated by this subsystem are $\overline{OUT}$, CM_1, and CM_2. The control signal associated with this subsystem is LD.

The implementation of the $\widehat{N_1}$ data-path subsystem is shown in the top portion of the schematic in Fig. 5.4. It consists of 1 MUX-3, 1 MUX-4, 1 MAC unit, 2 DECODERs, and 4 STATE registers. Observe that the variables X_U and X_V time-share the same state register representing the new state variable X_{UV}. The control signals associated with this subsystem are $S11$, $S10$, $S21$, and $S20$.

The implementation of the $\widehat{N_2}$ data-path subsystem is shown in the bottom portion of the schematic in Fig. 5.4. It consists of 2 MUX-3s, 1 MAC unit, 1 DECODER, and 3 STATE registers. The control signals associated with this subsystem are $S1$ and $S0$.

5.5.2 Gate-Level Implementation of the Control Unit

The control unit is implemented after developing the control word required for the MAC-modularized digital filter.

5.5.2.1 Development of the Control Word

The control word carries the information regarding the state updates required in each clock cycle of the digital filter operation. It is derived by using the state update equations in Algorithm 13 (Page 133).

LD	S1	S0	S11	S10	S21	S20

Figure 5.5. Control Word for the MAC-Modularized Lowpass LDI Jaumann Digital Filter

The control word is as shown in Fig. 5.5 and is a 7-bit value formed by using

the control signals LD, $S11$, $S10$, $S21$, $S20$, $S1$, and $S0$. These control signals are required for the proper operation of the data-path subsystems and are as shown in Fig. 5.4. The sequence of control word values required for processing one sample of the input signal is derived as follows. In the first clock cycle, the input sample is read, and the data-path subsystem $\widehat{N_0}$ computes the state variables $\overline{OUT^n}$, CM_1^n, and CM_2^n. In the subsequent five clock cycles, the data-path subsystems $\widehat{N_1}$ and $\widehat{N_2}$ operate concurrently. The state variable MC_1^{n+1} is computed by the subsystem $\widehat{N_1}$ by using five of these clock cycles, while the state variable MC_2^{n+1} is computed by the subsystem $\widehat{N_2}$ by using three of these clock cycles. Therefore, six clock cycles are required for processing one single sample of the input signal.

The value of the control word changes periodically with a period of 6 clock cycles. The sequence of the control word values for one period of operation is as shown in Table 5.1. Therefore, it is very simple to build the control FSM for this filter as there is only a repetitive pattern of six 7-bit signals to be generated.

Table 5.1. Control Word Values for Processing One Sample

Clock Cycle	*Control Word Value*
0	1000000
1	0000000
2	0010101
3	0101010
4	0111011
5	0110100

5.5.2.2 Implementation of the Control Unit

The control unit is implemented as shown in the schematic in Fig. 5.6. The COUNTER-05 cell counts from 0 to 5 repetitively, thereby enabling the execution of the 6 control instructions in Table 5.1.

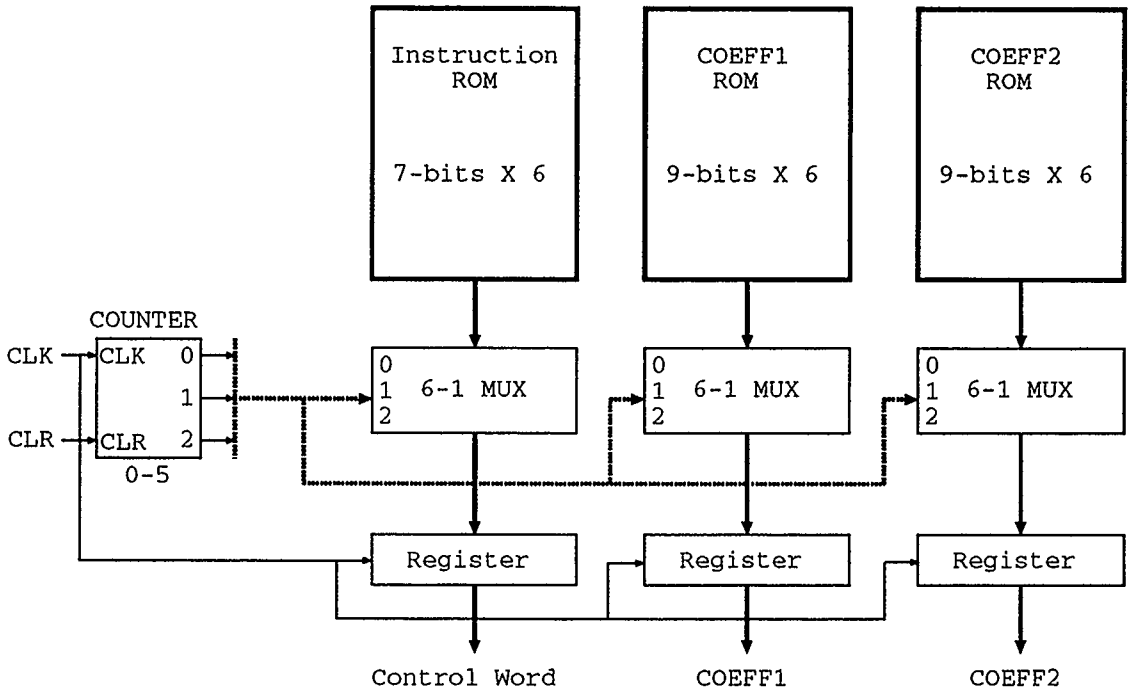

Figure 5.6. Implementation of the Control Unit

The schematic in Fig. 5.6 also contains three ROM banks, namely the *Instruction ROM*, *COEFF-1 ROM*, and the *COEFF-2 ROM*. The *Instruction ROM* stores the control instructions shown in Table 5.1. The *COEFF-1 ROM* and *COEFF-2 ROM* store the MAC coefficients required for the operation of $\widehat{N_1}$ and $\widehat{N_2}$, respectively. These ROMs are connected to $6 - to - 1$ multiplexers to allow the correct data to be fed to the corresponding control and coefficient buses. These multiplexers are controlled by the six-state counter COUNTER-05. Finally, the control word bits and the MAC coefficient bits for each cycle are latched before being fed to the data-path

subsystems.

5.6 Viewlogic Verification

The above lowpass LDI Jaumann digital filter was successfully verified using the Viewlogic simulations of the corresponding impulse response for an Actel 1.2μ technology implementation.

The simulation results are shown in Fig. 5.7, where LD, $S1[1:0]$, $S1\_10[1:0]$, and $S2\_10[1:0]$ represent the control word components LD, $(S1, S0)$, $(S11, S10)$, and $(S21, S20)$, respectively. $Y1[8:0]$ and $Y0[8:0]$ represent the 9-bit (representing 3 digits) MAC coefficients fed to the subsystems $\widehat{\mathcal{N}_1}$ and $\widehat{\mathcal{N}_2}$, respectively. The input signal IN is represented in its minimal-(n,p) format using the bus $INPP[18:0]$ for the p-part of the digits, and the bus $INPN[18:0]$ for the n-part of the digits. Similarly, the output signal OUT is represented in its minimal-(n,p) format using the bus $OUTP[18:0]$ for the p-part of the digits, and the bus $OUTN[18:0]$ for the n-part of the digits. The final output is derived by removing the overflow and roundoff digits from OUT. The signals CLR and CLK are used to reset and clock the registers within the digital filter, respectively.

The simulation results presented in Fig. 5.7 have been carried out at the maximum clock rate of 50 MHz. Since each sample requires 6 clock cycles to be processed, the maximum sample rate achievable by this implementation is 8.33 MHz. These simulations employed a typical delay of 1 nanosecond per gate. Therefore, it is seen that the choice of redundant arithmetic permits the exploitation of the maximum speed available by the given technology. It is again worth mentioning that this sample rate is totally independent of the signal wordlength of the digital filter.

5.7 Chapter Summary

In this chapter, the high-speed MAC arithmetic architectures developed in Chapter 3 and the MAC-modularization technique developed in Chapter 4 have been il-

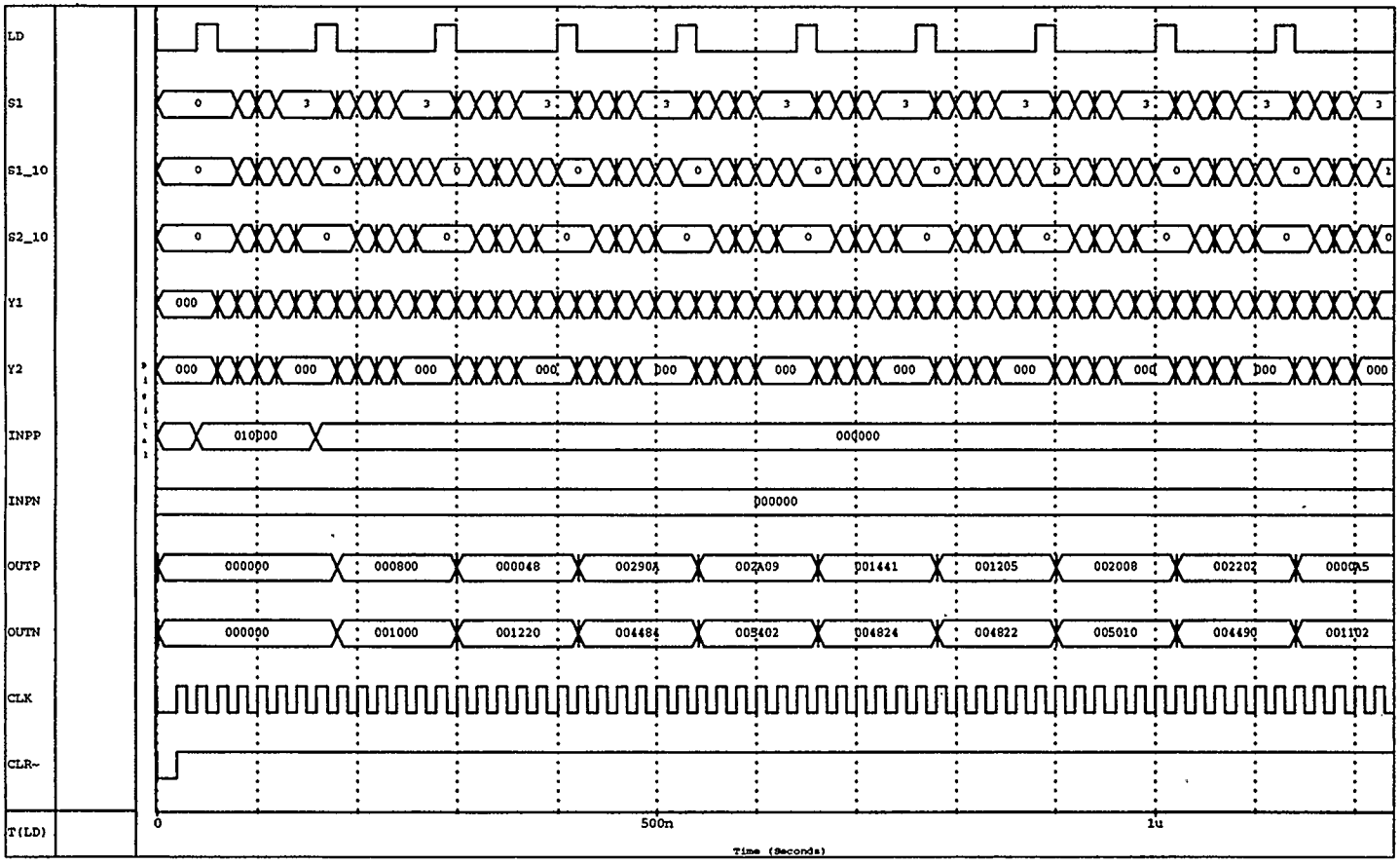

Figure 5.7. Impulse Response Simulation Results for the Lowpass LDI Jaumann Digital Filter

lustrated by applying them to the design and implementation of a practical lowpass LDI Jaumann digital filter. Section 5.2 has dealt with the MAC-modularization of the LDI Jaumann digital filter, together with the development of the corresponding state update equations for the constituent subsystems. The signal and MAC coefficient wordlengths required for the finite-precision arithmetic implementation of the MAC-modularized Jaumann digital filter have been calculated in Section 5.3. In Section 5.4, the number representations adopted for the signal and MAC coefficient words have been defined. The data-path and control unit implementations for the digital filter have been presented in Section 5.5. In Section 5.6, the resulting Actel 1.2μ technology implementation has been verified by using its impulse response Viewlogic simulations. The simulation results demonstrate an achievable clock rate of 50 MHz yielding a sample rate of 8.33 MHz. The striking feature of this implementation is that its speed of operation is completely independent of the signal wordlength within the digital filter.

CHAPTER 6

CONCLUSIONS

6.1 Summary of the Thesis

This thesis has presented the theoretical foundation underlying a novel systematic design philosophy for the realization of digital filters as a class of high-speed modular DSP architectures. Mathematical and graph-theoretic techniques have been presented for incorporating two important practical features in the resulting digital filters, namely, structural uniformity and fast processing speeds. The desired structural uniformity has been achieved through multiply-accumulate (MAC) modularization of the digital filter. The desired fast processing speed, on the other hand, has been achieved by using redundant number arithmetic to implement the constituent MAC operations.

In Chapter 2, a theoretical background was presented for fixed-point DSP arithmetic by introducing various number systems and arithmetic processing methodologies. This was followed by a rigorous mathematical analysis concerning recoding, rounding, and overflow processing of redundant numbers. A novel 5-digit overlapped scanning technique was presented for modified radix-4 recoding of SB numbers. Furthermore, two techniques for product rounding in SB number arithmetic were developed, namely, the RNU and RNE techniques. Finally, arithmetic overflow processing issues for SB numbers were discussed together with the concept of *directly correctable overflow* in fixed-point DSP systems.

In Chapter 3, the results in Chapter 2 were exploited and applied to the design and implementation of novel high-speed VLSI arithmetic architectures for multiplication and MAC operations. This included a novel approach for very high-speed mixed SB/TC digit-serial modified-Booth multiplication. It was shown that the area-time

efficiency and throughput of the resulting multipliers far surpass those of the existing digit-serial modified-Booth multipliers. It was also shown that redundant number arithmetic provides best results for fully parallel multiplication or MAC operations. Next, a novel architecture for high-speed mixed SB/TC parallel modified-Booth MAC arithmetic operation was presented. Finally, this architecture was extended to handle SB number multiplication by employing the modified radix-4 recoding technique, and was subsequently used for the design of a high-speed fully-SB parallel MAC arithmetic architecture. These parallel MAC architectures employ new techniques such as partitioned accumulation and concurrent rounding and overflow correction. The resulting architectures were subsequently parameterized in terms of their area-time requirements for corresponding Actel 1.2μ technology implementations, and were verified by using Viewlogic simulations.

Chapter 4 was concerned with a rigorous theoretical approach to MAC-modularization of digital filter SFGs. This approach consists of graph-theoretic techniques and their subsequent translation into algorithms for MAC-modularization. Taking into account the fact that several MAC-modularized digital filter SFGs can result starting from the same initial SFG, a fitness function was developed for the selection of the optimal SFG. This fitness function is based on finite-precision arithmetic effects exhibited by the corresponding MAC-modularized digital filters. Subsequently, enumerative and heuristic techniques for MAC-modularization were developed on the basis of the proposed fitness function. These techniques have been incorporated in a software package called **MAC-M** for the MAC-modularization of digital filters. Finally, the usefulness of **MAC-M** was demonstrated through its application to a pair of digital filters, one having dense, and another sparse multiplication operations.

In Chapter 5, the high-speed MAC arithmetic architectures developed in Chapter 3 and the MAC-modularization technique developed in Chapter 4 were illustrated by applying them to the design and implementation of a practical lowpass LDI [28]

Jaumann [6] digital filter. The optimal MAC-modularized LDI Jaumann digital filter was obtained by using **MAC-M**. The resulting Jaumann digital filter was then partitioned into three separate data-path modules by taking into account the inherent concurrency in the digital filter structure. This concurrency was then exploited to develop a schedule in terms of state equations for each data-path module in order to facilitate efficient high-speed parallel implementation of the filter. The simulation results demonstrated the achievable operational clock speed of 50 MHz, corresponding to a maximum permissible sample rate of 8.33 MHz for an Actel 1.2μ technology implementation. This implementation was verified by using impulse response simulations. The striking feature of this implementation is that its speed of operation is completely independent of the signal wordlength within the digital filter.

6.2 Contribution of the Thesis

To the best of the author's knowledge, the following contributions of the present thesis are original.

6.2.1 Chapter 1

- The systematic design philosophy for the realization of digital filters as a class of high-speed redundant number arithmetic modular DSP architectures.

6.2.2 Chapter 2

- The 5-digit overlapped scanning technique for modified radix-4 recoding of SB numbers (Section 2.4.1).
- The techniques for RNU and RNE of SB numbers to facilitate high-speed rounding (Section 2.4.2).
- The concept of SB *directly correctable overflow* for fixed-point DSP systems (Section 2.4.3).

6.2.3 Chapter 3

- The high-speed mixed SB/TC digit-serial modified-Booth multipliers featuring very high throughputs and efficiencies (Section 3.2).
- The architecture for high-speed mixed SB/TC parallel modified-Booth MAC arithmetic operation (Section 3.3).
- The architecture for high-speed fully-SB parallel MAC arithmetic operation based on the modified radix-4 recoding technique (Section 3.4).
- The concepts of partitioned accumulation and concurrent rounding and overflow processing (Sections 3.3 and 3.4).

6.2.4 Chapter 4

- The graph-theoretic approach for MAC-modularization of digital filters (Section 4.2).
- The enumerative technique for MAC-modularization (Section 4.4).
- The heuristic technique for MAC-modularization (Section 4.5).
- The software package **MAC-M** for automated MAC-modularization (Section 4.6).

6.2.5 Appendix A

- The proof for the modified-Booth recoding technique based on non-redundant radix-4 and modified radix-4 representation of TC numbers (Section A.2).

6.3 Suggestions for Future Related Research

This work has led to the opening of several windows of opportunity for future related research in redundant number arithmetic, MAC-modularization, and the allied fields of their potentialities.

There is a need for the development of mathematical techniques in the areas of recoding, rounding, and overflow processing of higher-radix redundant numbers. These techniques can be exploited in conjunction with most-significant-digit-first [30] digit-serial processing [31] for low latency ultra-fast DSP applications. The area/time tradeoffs in such systems must be carefully weighed against that of least-significant-digit-first systems. Moreover, area/time benefits can potentially be enhanced through the introduction of limited redundancy in the number representation. This can be achieved by employing quasi-redundant number representation [8] which is still an unexplored area.

From the implementation point of view, advances in ternary logic [43] can lead to a breakthrough in the applications of SBNR. This is because ternary logic forms a natural implementation platform for SBNR.

The use of genetic algorithms for MAC-modularization of digital filters is a logical extension of the research presented in this thesis. Automated high-level synthesis of MAC-modularized digital filters is also an interesting topic for further research. This is because, unlike the conventional synthesis approaches which use two-input one-output addition and multiplication operators, this approach will involve three-input one-output MAC operators. Moreover, the identification of chained accumulations in MAC data-flow graphs and their exploitation in building the schedules can result in potential benefits in terms of increased speed of operation and lower roundoff errors. In addition, there is scope in the area of scheduling MAC DSP algorithms for mapping to systolic array MAC arithmetic processors.

Asynchronous architectures have gained renewed popularity due to their low power consumption property. The design and high-level synthesis of asynchronous MAC-modularized digital filters is yet to be explored and is a candidate for future research. The initial work in this area has been carried out in [47], where several novel asynchronous parallel MAC arithmetic architectures have been developed and compared.

APPENDIX A

ALTERNATIVE PROOF OF MODIFIED-BOOTH RECODING BASED ON NON-REDUNDANT RADIX-4 NUMBER ARITHMETIC

A.1 Introduction

The modified-Booth recoding algorithm was developed by Macsorley [36] more than three decades ago. This recoding algorithm finds important practical applications in two's complement (TC) multiplication, particularly due to the fact that it reduces the number of intermediate partial product components generated during the course of multiplication by a factor of two. This leads not only to a substantial reduction in the multiplication time, but also to a marked economy in terms of the real-estate area in a corresponding VLSI hardware implementation.

Consider a N-bit number Y represented in TC format in accordance with

$$Y = -y_{N-1}2^{N-1} + \sum_{n=0}^{N-2} y_n 2^n, \tag{A.1}$$

where the bits $y_n \in \{0, 1\}$, and where N is an even integer. Then, through the application of the above recoding algorithm, the number Y is replaced by a corresponding modified radix-4 signed-digit (SD) representation [49] given by [36]

$$Z = \sum_{n=0}^{\frac{N-2}{2}} z_n 4^n, \tag{A.2}$$

where the digits $z_n \in \{0, \pm 1, \pm 2\}$. These digits are determined by using

$$z_n = -2y_{2n+1} + y_{2n} + y_{2n-1}, \tag{A.3}$$

with $y_{-1} = 0$.

The salient feature of the number representation in Eqn. (A.2) is that *all* of the digits z_n take on values from the same *balanced* digit-set $\{0, \pm 1, \pm 2\}$, making the corresponding summation uniform.

In 1975, Rubinfield [26] presented a proof of the modified-Booth recoding algorithm by establishing the algebraic equivalence of the number Y in Eqn. (A.1) and the number Z in Eqn. (A.2). This equivalence was established by splitting Eqn. (A.1) as

$$Y = -y_{N-1}2^{N-1} + \sum_{n=0(even)}^{N-2} y_n 2^n + \sum_{n=1(odd)}^{N-3} y_n 2^n, \tag{A.4}$$

followed by the addition and subtraction of the summation $\sum_{n=1(odd)}^{N-3} y_n 2^n$ to yield

$$Y = -\sum_{n=1(odd)}^{N-1} y_n 2^n + \sum_{n=0(even)}^{N-2} y_n 2^n + 2\sum_{n=1(odd)}^{N-3} y_n 2^n. \tag{A.5}$$

Subsequently, Eqn. (A.5) can be simplified to

$$Y = \sum_{n=0(even)}^{N-2} (-2y_{n+1} + y_n + y_{n-1})2^n, \tag{A.6}$$

by combining the constituent summations. Finally, Eqn. (A.2) was established by changing the summation index from n to $2n$ in Eqn. (A.6), and by invoking Eqn. (A.3) in the result.

In this thesis, an alternative proof for the modified-Booth recoding algorithm is established [51] by successively transforming the number Y from its TC to its non-redundant radix-4 [23], and from its non-redundant radix-4 to its modified radix-4 SD representation.

A.2 Proof of the Modified-Booth Recoding Algorithm

The TC number Y in Eqn. (A.1) can be expressed in its non-redundant radix-4 representation in accordance with [23]

$$Y = (-2y_{N-1} + y_{N-2})4^{\frac{N-2}{2}} + \sum_{n=0}^{\frac{N-4}{2}} (2y_{2n+1} + y_{2n})4^n, \tag{A.7}$$

where the term $(-2y_{N-1} + y_{N-2})$ represents the sign, and the terms $(2y_{2n+1} + y_{2n})$ represent the magnitude of the number Y. Unfortunately, the representation in Eqn. (A.7) suffers from two different (but interrelated) problems:

1. The sign term $(-2y_{N-1} + y_{N-2})$ takes on values from the digit-set $\{0, \pm 1, -2\}$, whilst the magnitude terms $(2y_{2n+1} + y_{2n})$ take on values from the digit-set $\{0, 1, 2, 3\}$, making the number representation non-uniform.

2. The permitted value of 3 for the magnitude terms $(2y_{2n+1} + y_{2n})$ can make a corresponding hardware implementation potentially impracticable.

In this contribution, it is proposed to recast Eqn. (A.7) into an equivalent representation in such a manner that the resulting sign term and magnitude terms all take on values from the same balanced digit-set. This is achieved through the introduction of the transformation

$$4t_{n+1} + w_n = 2y_{2n+1} + y_{2n} \tag{A.8}$$

in Eqn. (A.7), where the *transfer digits* t_{n+1} and the *weight digits* w_n can be determined *uniquely* from Eqn. (A.8). In particular, by recalling that $y_n \in \{0, 1\}$, Eqn. (A.8) can be solved for t_{n+1} and w_n to yield

$$t_{n+1} = y_{2n+1} \tag{A.9}$$

and

$$w_n = -2y_{2n+1} + y_{2n}, \tag{A.10}$$

implying that $t_{n+1} \in \{0, 1\}$ and $w_n \in \{0, \pm 1, -2\}$.

By invoking Eqns. (A.9) and (A.10), Eqn. (A.7) can be transformed to

$$Y = \sum_{n=0}^{\frac{N-2}{2}} (w_n + t_n) 4^n. \tag{A.11}$$

By taking into account the possible values for the transfer and weight digits t_n and w_n, it can be shown that $(w_n + t_n) \in \{0, \pm 1, \pm 2\}$, making Eqn. (A.11) a uniform

representation of the number Y. In order to complete the proof, it remains to be shown that

$$w_n + t_n = z_n. \tag{A.12}$$

By invoking Eqn. (A.10) for the sign term and Eqn. (A.8) for the magnitude summation terms in Eqn. (A.7), one obtains

$$Y = w_{\frac{N-2}{2}} 4^{\frac{N-2}{2}} + \sum_{n=0}^{\frac{N-4}{2}} (4t_{n+1} + w_n)4^n, \tag{A.13}$$

where the term $w_{\frac{N-2}{2}}$ represents the transformed sign term. By splitting the summation in Eqn. (A.13), one can write

$$Y = w_{\frac{N-2}{2}} 4^{\frac{N-2}{2}} + \sum_{n=0}^{\frac{N-4}{2}} t_{n+1}4^{n+1} + \sum_{n=0}^{\frac{N-4}{2}} w_n 4^n. \tag{A.14}$$

In order to proceed further, Eqn. (A.14) is simplified by making use of the fact

$$w_{\frac{N-2}{2}} 4^{\frac{N-2}{2}} + \sum_{n=0}^{\frac{N-4}{2}} w_n 4^n = \sum_{n=0}^{\frac{N-2}{2}} w_n 4^n. \tag{A.15}$$

Moreover, by substituting $y_{-1} = 0$ in Eqn. (A.9), one obtains $t_0 = 0$, implying

$$\sum_{n=0}^{\frac{N-4}{2}} t_{n+1}4^{n+1} = \sum_{n=0}^{\frac{N-2}{2}} t_n 4^n. \tag{A.16}$$

Then, by substituting Eqns. (A.15) and (A.16) in Eqn. (A.14), and by combining the resulting summations, one immediately arrives at Eqn. (A.11). But, by using Eqns. (A.9) and (A.10), the terms $w_n + t_n$ appearing under the summation in Eqn. (A.11) are given by

$$w_n + t_n = -2y_{2n+1} + y_{2n} + y_{2n-1}. \tag{A.17}$$

Finally, by comparing Eqns. (A.17) and (A.3) one arrives at Eqn. (A.12), establishing the proof. ■

REFERENCES

[1] A. ANTONIOU, *Digital Filters - Analysis, Design, and Applications*, McGraw-Hill, Inc., (1993).

[2] A. AVIZIENIS, *Signed Digit Numbe(r) Representation for Fast Parallel Arithmetic*, IRE Transactions on Electronic Computers, (September 1961), pp. 389–400.

[3] A. D. BOOTH, *A Signed Binary Multiplication Technique*, Quart. J. Mech. Appl. Math., 4 (1951).

[4] A. VANDEMEULEBROECKE, E. VANZIELEGHEM AND P. JESPERS, *A New Carry-Free Division Algorithm and its Application to a Single-Chip 1024-bit RSA Processor*, IEEE Journal on Solid-State Circuits, 25 (June 1990), pp. 748–755.

[5] A. W. OPPENHEIM AND R. SCHAFER, *Digital Signal Processing*, Prentice-Hall Ltd., (1994).

[6] B. NOWROUZIAN, N.R. BARTLEY AND L.T. BRUTON, *Design and DSP-Chip Implementation of a Novel Bilinear-LDI Digital Jaumann Filter*, IEEE Transactions on Circuits and Systems, CAS-37 (June 1990), pp. 695–706.

[7] B. PARHAMI, *Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations*, IEEE Transactions on Computers, 39 (January 1990), pp. 89–98.

[8] D. S. PHATAK AND I. KOREN, *Hybrid Signed-Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains*, IEEE Transactions on Computers, 43 (August 1994), pp. 880–890.

[9] D. TIMMERMANN AND B.J. HOSTICKA, *Overflow Effects in Redundant Binary Number Systems*, Electronics Letters, 29 (March 1993), pp. 440–441.

[10] D.E. THOMAS, E.D. LANGNESE, R.A. WALKER, J.A. NESTOR, J.V. RAJAN, AND R.L. BLACKBURN, *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*, Kulwer Academic Publishers, (1990).

[11] F. J. TAYLOR, *A VLSI Residue Arithmetic Multiplier*, IEEE Transactions on Computers, C-31 (June 1982).

[12] G.-K. MA AND F. J. TAYLOR, *Multiplier Policies for Digital Signal Processing*, IEEE ASSP Magazine, (January 1990), pp. 6–19.

[13] G. PANEERSELVAM AND B. NOWROUZIAN, *Multiply-Add Fused RISC Architecture for DSP Applications*, Proceedings of IEEE Pacific RIM on Communications, Computers and Signal Processing, Victoria, B.C. Canada, (May 1993), pp. 108–111.

[14] H.T. KUNG, *Why Systolic Architectures*, IEEE Computer, (January 1982), pp. 37–42.

[15] J. H. SATYANARAYANA, *A Powerful Genetic Algorithm for the High Level Synthesis of Globally Optimal Digit-Serial Digital Filters*, M.Sc. Thesis, Dept. of Electical and Computer Engineering, The University of Calgary, Canada, (June 1994).

[16] J. H. SATYANARAYANA AND B. NOWROUZIAN, *Design and FPGA implementation of Digit-Serial Modified Booth Multipliers*, Journal of Circuits, Systems and Computers, (In Press).

[17] ——, *A Comprehensive Approach to the Design of Digit-Serial Modified Booth Multipliers*, Proc. 26th IEEE South-eastern Symp. on System Theory, Athens, OH, (March 1994), pp. 229–233.

[18] J. M. YOHE, *Roundings in Floating Point Arithmetic*, IEEE Transactions on Computers, C-22 (June 1973), pp. 577–586.

[19] J.W. COOLEY AND J.W. TUKEY, *An Algorithm for Machine Computation of the Complex Fourier Series*, Math. Computation, 19 (1965), pp. 297–301.

[20] K. HWANG, *Computer Arithmetic - Principles, Architecture and Design*, John Wiley & Sons, 1979.

[21] K. ITO AND H. KUNIEDA, *VLSI System Compiler for Digital Signal Processing: Modularization and Synchronization*, IEEE Transactions on Circuits and Systems, CAS-38 (April 1991), pp. 423–433.

[22] K.K. PARHI, *A Systematic Approach for the Design of Digit-Serial Signal Processing Architectures*, IEEE Transactions on Circuits and Systems, CAS-38 (June 1991), pp. 358–375.

[23] K.K. PRIMLANI AND J. L. MEADOR, *A Nonredundant-Radix-4 Serial Multiplier*, IEEE Journal of Solid-State Circuits, 24 (December 1989), pp. 1729–1736.

[24] L. M. MAXWELL AND M. B. REED, *The Theory of Graphs — A Basis for Network Theory*, Pergamon Press, (1971).

[25] L. M. SMITH, *Design and Bit-Serial Implementation of LDI-Jaumann Digital Filters*, M.Sc. Thesis, Dept. of Electical and Computer Engineering, The University of Calgary, Canada, (September 1993).

[26] L. P. RUBINFIELD, *A Proof of the Modified Booth's Algorithm for Multiplication*, IEEE Transactions on Computers, (October 1975), pp. 1014–1015.

[27] L. R. RABINER AND B. GOLD, *Theory and Application of Digital Signal Processing*, Prentice-Hall Ltd., (1993).

[28] L.T. BRUTON, *Low Sensitivity Digital Ladder Filters*, IEEE Transactions on Circuits and Systems, CAS-22 (March 1975), pp. 168–176.

[29] M. ANDREWS, *A Systolic SBNR Adaptive Signal Processor*, IEEE Transactions on Circuits and Systems, CAS-33 (February 1986), pp. 230–238.

[30] M. J. IRWIN AND R. M. OWENS, *Design Issues in Digit Serial Processors*, International Symp. on Circuits and Systems, (1989), pp. 441–444.

[31] ——, *Fully Digit On-Line Networks*, IEEE Transactions on Computers, C-32 (April 1983), pp. 402–406.

[32] M.R. SANTORO, G. BEWICK AND M. A. HOROWITZ, *Rounding Algorithms for IEEE Multipliers*, Proceedings of the 9-th Symposium on Computer Arithmetic, Santa Monica, CA, USA, (September 1989), pp. 176–183.

[33] N. CHRISTOFIDES, *Graph Theory — An Algorithmic Approach*, Academic Press, (1975).

[34] N. TAKAGI, H. YASUURA AND S. YAJIMA, *High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree*, IEEE Transactions on Computers, C-34 (September 1985), pp. 789–796.

[35] O. SPANIOL, *Computer Arithmetic*, John Wiley & Sons, (1981).

[36] O.L. MACSORLEY, *High Speed Arithmetic in Binary Computers*, Computer Arithmetic, Dowden Hutchinson and Ross Inc., 21 (1980), pp. 100–104.

[37] R. A. ROBERTS AND C. T. MULLIS, *Digital Signal Processing*, Addison-wesley Publishing Company, (1987).

[38] R. HARTLEY AND P. CORBETT, *Digit-Serial Processing Techniques*, IEEE Transactions on Circuits and Systems, CAS-37 (June 1990), pp. 707–719.

[39] R.F. LYON, *Two's Complement Pipeline Multipliers*, IEEE Transactions on Communications, (April 1976), pp. 418–425.

[40] R.K. MONTOYE, E. HOKENEK AND S.L. RUNYON, *Design of the IBM RISC System/6000 Floating-Point Execution Unit*, IBM Journal Res. Develop., (January 1990), pp. 59–70.

[41] S. EVEN, *Graph Algorithms*, Computer Science Press, (1979).

[42] S. KUNINOBU, H. EDAMATSU, T. TANIGUCHI AND N. TAKAGI, *Design of High Speed MOS Multiplier and Divider using Redundant Binary Representation*, Proc. of the 8-th Symp. on Computer Arithmetic, (May 1987), pp. 80–86.

[43] S. L. HURST, *Multiple-Valued Logic — Its Status and Its Future*, IEEE Transactions on Computers, C-33 (December 1984), pp. 1160–1179.

[44] S.-Y. KUNG, H.J. WHITEHOUSE AND T. KAILATH - EDITORS, *VLSI and Modern Signal Processing*, Prentice-Hall Information and System Sciences Series, (1985).

[45] V. M. RAO, *Design and Implementation of a High-Speed Redundant Number Multiply-Accumulate-Modularized LDI-Jaumann Digital Filter*, Internal Technical Report, Dept. of Electrical and Computer Engineering, The University of Calgary, Canada, (July 1996).

[46] ——, *Design and Implementation of Novel High-Speed Digit-Serial Modified-Booth Multipliers*, Internal Technical Report, Dept. of Electrical and Computer Engineering, The University of Calgary, Canada, (July 1996).

[47] V. M. RAO AND B. NOWROUZIAN, *Design and Implementation of Asynchronous Parallel Multiply-Accumulate Arithmetic Architectures*, Proceedings of

the 38th Midwest Symposium on Circuits and Systems, Rio de Janeiro, Brazil, (August 1995), pp. 761–764.

[48] ——, *A Novel Approach to the Design and Implementation of Very High-Speed Digit-Serial Modified-Booth Multipliers*, Proceedings of the 39th Midwest Symposium on Circuits and Systems, Ames, Iowa, U.S.A., (August 1996), p. in press.

[49] ——, *A Novel High-Speed Parallel Multiply-Accumulate Arithmetic Architecture Employing Modified Radix-4 Signed-Binary Recoding*, Proceedings of the 39th Midwest Symposium on Circuits and Systems, Ames, Iowa, U.S.A., (August 1996), p. in press.

[50] ——, *Novel High-Speed Bit-Parallel Multiply-Accumulate Arithmetic Architecture.*, Proceedings of the Advanced Signal Processing Algorithms, Architectures and Implementations Conference of the SPIE VI, Denver, CO, U.S.A., (August 1996), p. in press.

[51] ——, *Alternative Proof of Modified-Booth Recoding*, submitted to the Electronics Letters, (July 1996).

[52] ——, *A Novel Modularization Approach to the Design and Implementation of High-Speed Redundant Arithmetic DSP Architectures*, Proceedings of the Micronet Annual Workshop, Ottawa, Canada, (March 1996), pp. 45–46.

[53] ——, *Rounding Techniques for Signed Binary Arithmetic*, Proceedings of the 1996 Canadian Conference on Electrical and Computer Engineering, Calgary, Canada, (May 1996), pp. 294–297.