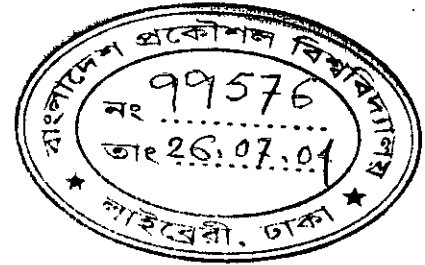


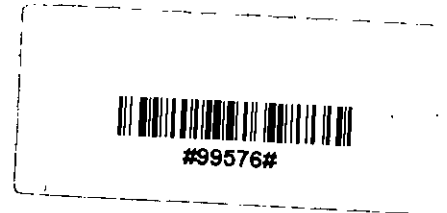
SYMBOLIC SUBSTITUTION BASED APPROACH  
FOR THE DESIGN OF CARRY-FREE BINARY  
ARITHMETIC UNIT

by

TASADDUQ IMAM



A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE AND  
ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF MASTER OF SCIENCE IN ENGINEERING  
(COMPUTER SCIENCE AND ENGINEERING)

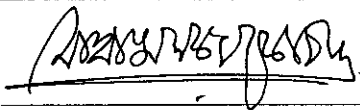
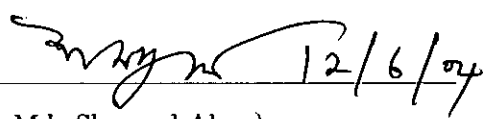
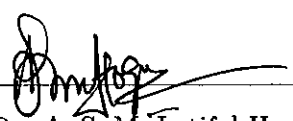



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY  
DHAKA, BANGLADESH

JUNE 2004

The thesis "SYMBOLIC SUBSTITUTION BASED APPROACH FOR THE DESIGN OF CARRY-FREE BINARY ARITHMETIC UNIT" submitted by Tasadduq Imam, Roll No. 040205033P, Registration No. 95401, Session April 2002, to the Department of Computer Science and Engineering, BUET, has been accepted as satisfactory for partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents.

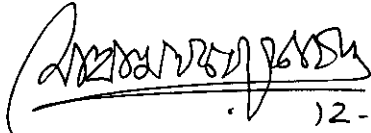
BOARD OF EXAMINERS

1.  12-06-2004  
(Dr. M. Kaykobad) Chairman  
Professor, (Supervisor)  
Department of CSE, BUET, Dhaka.
  
2.  12/6/04  
(Dr. Md. Shamsul Alam) Member  
Professor and Head, (Ex-officio)  
Department of CSE, BUET, Dhaka.
  
3.   
(Dr. A. S. M. Latiful Haque) Member  
Assistant Professor,  
Department of CSE, BUET, Dhaka.
  
4.  12.06.2004  
(Dr. Md. Mozammel Haque Azad Khan) Member  
Professor, (External)  
Department of CSE, East West University, Dhaka.

## DECLARATION

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. M. Kaykobad, Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned



12-06-2004

(Dr. M. Kaykobad)

Supervisor



12/06/04

(Tasadduq Imam)

## ACKNOWLEDGEMENT

Here I would like to take the opportunity to express my greatest gratitude to the patrons of this thesis work, without whom I could never have completed this arduous task.

First and Foremost, I express my sincere gratitude and profound respect to my supervisor **Dr. M. Kaykobad**, Professor, Dept. of Computer Science and Engineering, Bangladesh University of Engineering and Technology. It would be injustice to say only that the thesis was just completed under his kind supervision. His keen interest, deep insight and ideas and above all his time to time scholarly guidance and suggestions have made it possible to complete this thesis work.

I would also like to thank **Dr. M. A. Karim**, Professor, Dept. of Electrical and Electronic Engineering, City College of NewYork, US for guiding me to some reading materials and giving valuable suggestions.

I would also like to express my heartiest gratitude to **Dr. M. Z. Iqbal**, Professor, Dept. of Electrical and Electronic Engineering, Shahjalal University of Science and Technology, Sylhet, Bangladesh for sending me a copy of a presentation of Professor Karim, very much related to my thesis topic.

I would also like to thank the anonymous reviewers of the *International Journal of Computers and Mathematics with Applications* for their suggestions that have been helpful in providing some new insight.

I must acknowledge with due respect the constant support and patience of my parents for completing the thesis.

And I must thank the members of the board, **Prof. Dr. Md. Shamsul Alam**, **Dr. A. S. M. Latiful Haque** and **Prof. Dr. M. H. A. Khan**, for their comments and suggestions.

## ABSTRACT

Symbolic Substitution has been proposed in optical computing literature as a parallel processing technique to perform arithmetic computation. Employing a conversion table as a reference, the process iteratively substitutes a set of input patterns by pre-defined output patterns. Along with the development of this parallel processing technique, researchers have also come up with a number of non-binary representations that allow fast carry-free addition. Modified Signed Digit(MSD) and Canonical Modified Signed Digit(CMSD) number systems have been extremely popular in this regard. In contrast with the binary system, these number systems use three types of symbols: 0, 1 and -1. Redundancy due to the extra digit make these systems suitable for symbolic substitution based parallel addition process.

However, traditional computing is based on binary system and corresponding binary logic. But generation of carry in binary arithmetic is the main hindrance to employing the symbolic substitution process and making the computation fast.

The focus of this study has, therefore, been the development of a symbolic substitution based process for fast arithmetic computation of binary numbers. The idea has been the design of a fast addition unit that accepts at its interface numbers represented in binary form, then converts these numbers to some intermediate non-binary representation for processing, and then converts the result back to the binary form. The focus has been the development of symbolic substitution processes for all the intermediate phases involved like the conversion of binary to the non-binary representation, processing of the numbers and finally the conversion of the non-binary representation to binary representation. Due to its capability of unique representation of numbers and sparseness of the non-zero digits in the representation, Canonical Modified Signed Digit (CMSD) representation has been chosen as the non-intermediate representation for the arithmetic unit proposed in the study.

An important factor in the context of symbolic substitution is the required number of

substitution steps. The thesis presents a set of symbolic substitution tables and algorithms that require much less substitution steps to complete the conversion and addition operations than any other corresponding earlier schemes. Also, the addition algorithm presented in the study derives the addition result in CMSD notation and thus it could be employed to perform symbolic substitution based associative addition of a set of CMSD numbers.

An important contribution of this thesis is the development of a symbolic substitution based unit that can perform the associative addition of a set of binary numbers in computationally more efficient way than using any of the earlier proposed schemes. Such an algorithm can lead to the way of employing symbolic substitution in other arithmetic operations, like multiplication, where associative addition of a set of numbers is an integral process.

# CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT	ii
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ALGORITHMS	ix

## Chapters

<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Fast Computation and Bottlenecks of Traditional Computing System . . .	1
1.2 Approaches to Fast Computation . . . . .	2
1.3 Optical Computing and Symbolic Substitution . . . . .	2
1.4 Non-binary Number Systems . . . . .	2
1.5 Objective and Scope of the Thesis . . . . .	3
1.6 Organization of the Thesis . . . . .	4
<b>2 CARRY-FREE ARITHMETIC: A LITERATURE SURVEY</b>	<b>5</b>
2.1 Carry-free Arithmetic . . . . .	5
2.2 Carry Lookahead Adder(CLA) . . . . .	6
2.3 Introduction of Redundancy in Number Systems . . . . .	7
2.4 Different Number Systems . . . . .	7
2.4.1 Redundant Number System . . . . .	8
2.4.2 Modified Signed Digit Number System . . . . .	9
2.4.3 Canonical Modified Signed Digit Number System . . . . .	9
2.4.4 Residue Number System . . . . .	10
2.4.5 Other Variations of Redundant Number System . . . . .	11
2.4.6 Double Base Number System . . . . .	12
2.5 Carry-free Addition Schemes . . . . .	13

2.6	Choice of Number System in Carry-free Addition Scheme . . . . .	13
<b>3</b>	<b>SYMBOLIC SUBSTITUTION AND ARITHMETIC COMPUTATION</b>	<b>15</b>
3.1	Optics as a Backbone of Fast Computation . . . . .	15
3.2	Symbolic Substitution . . . . .	16
3.3	Different Symbolic Substitution Methods . . . . .	16
3.4	Intensity Coded Symbolic Substitution . . . . .	17
3.4.1	Dual-rail Encoding . . . . .	17
3.4.2	Symbolic Substitution Process . . . . .	18
3.4.3	Multiple-rules Symbolic Substitution Process . . . . .	21
3.5	Polarization Coded Symbolic Substitution . . . . .	21
3.6	Basic Symbolic Substitution Unit . . . . .	22
3.7	Symbolic Substitution and Arithmetic Computation . . . . .	23
3.8	Symbolic Substitution Based Computation Algorithms . . . . .	23
3.8.1	Three-step MSD Addition . . . . .	24
3.8.2	Two-step MSD Addition . . . . .	26
3.8.3	One-step CMSD Addition . . . . .	27
3.8.4	Conversion to CMSD . . . . .	28
3.9	Concluding Remarks . . . . .	28
<b>4</b>	<b>DESIGN OF A SYMBOLIC SUBSTITUTION BASED BINARY ARITHMETIC UNIT</b>	<b>30</b>
4.1	Underlying Motivation . . . . .	30
4.2	Proposed Arithmetic Unit . . . . .	30
4.3	Different Phases of the Proposed Arithmetic Unit . . . . .	31
4.3.1	Conversion of MSD and Binary to CMSD . . . . .	32
4.3.2	Conversion of Binary to CMSD: One-step Algorithm . . . . .	34
4.3.3	Addition of CMSD numbers . . . . .	38
4.3.4	Addition of CMSD numbers in 1 step . . . . .	40
4.3.5	Conversion of MSD to Binary . . . . .	43
4.3.6	Conversion of CMSD to Binary . . . . .	44
4.4	Comparison of the Proposed Schemes with Earlier Ones . . . . .	45
<b>5</b>	<b>STEP REDUCTION AND EXPERIMENTAL ANALYSIS</b>	<b>47</b>
5.1	Reduction of Symbolic Substitution Steps . . . . .	47
5.2	Conversion of MSD and Binary to CMSD . . . . .	49
5.3	Addition of CMSD numbers . . . . .	50
5.4	Conversion of CMSD to Binary . . . . .	53
5.5	Concluding Remarks . . . . .	54



---

<b>6</b>	<b>SYMBOLIC SUBSTITUTION BASED ASSOCIATIVE ADDITION</b>	<b>55</b>
6.1	Introduction to the Topic . . . . .	55
6.2	Associative Addition of Binary Numbers . . . . .	55
6.3	Block Level Architecture of the Unit . . . . .	57
6.4	Required Steps to Complete Addition . . . . .	57
6.5	Concluding Remarks . . . . .	59
<b>7</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	<b>60</b>
7.1	Concluding Words . . . . .	60
7.2	Recommendations for Future Work . . . . .	62
	<b>BIBLIOGRAPHY</b>	<b>64</b>
	<b>PUBLICATIONS OF THE CANDIDATE RELATED TO THE THESIS</b>	<b>68</b>
	<b>INDEX</b>	<b>69</b>

## LIST OF FIGURES

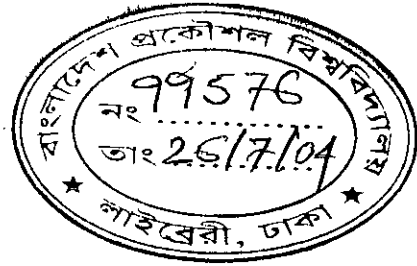
2.1	Carry-free addition . . . . .	6
3.1	Dual-rail encoding . . . . .	17
3.2	Substitution rule . . . . .	18
3.3	Symbolic substitution: Recognition phase (step 1) . . . . .	19
3.4	Symbolic substitution: Recognition phase (step 2) . . . . .	19
3.5	Symbolic substitution: Recognition phase (step 3) . . . . .	20
3.6	Symbolic substitution: Substitution phase . . . . .	20
3.7	Multiple substitution rules . . . . .	21
3.8	Block architecture of a symbolic substitution unit . . . . .	22
3.9	Symbolic substitution based arithmetic computation . . . . .	23
4.1	Logical diagram of the proposed fast addition unit . . . . .	31
5.1	Graph of total of the percentage converted vs. required no. of steps for MSD to CMSD conversion . . . . .	51
5.2	Graph of total of the percentage added vs. required no. of steps for CMSD addition . . . . .	52
5.3	Graph of total of the percentage converted vs. required no. of steps for CMSD to binary conversion . . . . .	54
6.1	Symbolic substitution architecture for associative addition . . . . .	58

## LIST OF TABLES

3.1. Symbolic substitution tables for MSD addition: Three-steps . . . . .	24
3.3 Symbolic substitution tables for MSD addition: Two-steps . . . . .	26
3.4 Symbolic substitution tables for CMSD addition: One-step . . . . .	28
3.5 Reitwiesner's conversion table and corresponding complementary conversion table . . . . .	28
4.1 Symbolic substitution table to convert MSD and binary number to CMSD in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps . . . . .	33
4.2 CheckCond.1 for Table 4.1 . . . . .	33
4.3 CheckCond.2 for Table 4.1 . . . . .	33
4.4 Symbolic substitution table for converting 4-bit binary number to CMSD representation in 1 step . . . . .	37
4.5 Symbolic substitution table for performing addition of CMSD numbers in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps and preserving CMSD property . . . . .	39
4.6 CheckCond.1 for Table 4.5 . . . . .	40
4.7 CheckCond.2 for Table 4.5 . . . . .	40
4.8 Symbolic substitution table for performing addition of CMSD numbers in one step . . . . .	41
4.9 Symbolic substitution table for converting MSD number to 2's complement binary number in $n$ steps . . . . .	43
4.10 Symbolic substitution table to convert CMSD to binary in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps . . . . .	45
5.1 Result of analysis for MSD to CMSD conversion . . . . .	49
5.2 Result of analysis for CMSD addition . . . . .	51
5.3 Result of analysis for CMSD to binary conversion . . . . .	53
6.1 Demonstration of associative addition . . . . .	56
6.1 (Contd.) . . . . .	57
6.2 Values of $m$ and $F(m)$ . . . . .	58

## LIST OF ALGORITHMS

3.1 Conversion to CMSD (Reitwiener's algorithm) . . . . .	28
4.1 Algorithm to convert MSD and binary number to CMSD in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps	32
4.2 Search for a pattern like $(10)^*11$ . used in Algorithm 4.3 . . . . .	34
4.3 Binary to CMSD Conversion Algorithm: 1 Step . . . . .	35
4.4 Algorithm for performing addition of CMSD numbers in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps . .	38
4.5 CMSD addition in 1 step . . . . .	41
4.6 Algorithm for converting MSD number to 2's complement binary number in $n$ steps . . . . .	43
4.7 Algorithm to convert CMSD to binary in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps . . . . .	44
6.1 Associative addition algorithm . . . . .	56



## Chapter 1

# INTRODUCTION

### 1.1 Fast Computation and Bottlenecks of Traditional Computing System

With the advance of science and technology, the need for processing of large amount of data at high-speed has gradually increased. Data-intensive applications like signal and image processing, weather forecasting and modelling, remote sensing require computational rates equivalent to trillions of operations per second. This high speed-up can only be achieved through parallel processing of information.

Traditional computing system is based on electronic devices and binary number system. One of the main obstacles to the increase of speed in the traditional binary computing system is the generation of carry during arithmetic operation. Any generated carry during binary arithmetic ripples through all the cascaded stages and affects the overall computation. Thus binary computing system implies sequential processing of data and results in a bottleneck in the context of fast computation. Also, high speed computing in the traditional electronic computing system is constrained by the interconnection bottleneck that results in sequential processing of digital information.

There have been efforts to improve the performance of a conventional computer system by reducing the basic cycle time through employing different fast circuits and packaging technique. However, to reach the rate of one trillion operations per second, the system requires a cycle time of less than a picosecond. This cannot be expected from the advances in electronic technology alone, because of the inherent physical limitations. Further speed-up, therefore, will have to come through parallelism.

## 1.2 Approaches to Fast Computation

Researchers have made numerous efforts to develop computing systems that can perform fast parallel arithmetic computation. These studies have been carried out mainly in two perspectives. One is the development of computing systems that can process information in parallel [1-7]. The other is the design of number systems that allow limited carry-propagation or completely carry-free arithmetic [2, 7-13].

## 1.3 Optical Computing and Symbolic Substitution

Optics has many unique features that can be exploited for high speed computation. Optical signals propagate in parallel, cross each other without interference, have lowest propagation delay of all signals, and can provide a million channel free-space interconnections with primitive lenses. Whereas communication bottleneck forces electronic computers to update the computation state space sequentially, optics allows the whole state to be changed in parallel [14]. This inherent parallelism has led to the development of parallel arithmetic processing techniques in the field of optical computing, that allow higher throughput and faster processing rate than conventional computing system.

Symbolic substitution [2, 15, 16] is an optical processing technique that has been widely proposed for applications like image processing and arithmetic computation. Basically it is a parallel pattern replacement process through which a set of given input patterns are substituted by a set of output patterns. A symbolic substitution table is used as reference during the substitution process. Considering the representation of operands as input patterns, researchers have proposed the symbolic substitution process for arithmetic processing of numbers [14]. The main advantage of the process is the absence of carry. As a result all output bits are derived in parallel, resulting in the speed up of the computation.

## 1.4 Non-binary Number Systems

Along with the development of parallel processing techniques, researchers have also come up with different non-binary representation of numbers that exploit the parallelism through limited carry-propagation or carry-free arithmetic [8, 9, 12, 14, 17, 18]. Among these number systems, the Modified Signed Digit(MSD) number system and its variation, the Canonical

Modified Signed Digit (CMSD) number system, have found its wide application in optical computing in the context of symbolic substitution process [2, 14]. Both of these systems are weighted radix 2-based number systems, comprising three types of symbols: 0, 1 & -1. The redundancy in the systems allows carry-free operation and therefore the systems have been most suitable for application in symbolic substitution process.

## 1.5 Objective and Scope of the Thesis

The earlier proposed symbolic substitution processes have concentrated on employing non-binary systems for arithmetic computing. This study, however, concentrates on the design of an arithmetic unit that combines the parallelism of the symbolic substitution process with the flexibility of binary system for data-intensive computing. Addition is the basic operation involved in any arithmetic computation and speeding up addition will result in speeding up the whole arithmetic process. So the focus has been mainly on the development of an adder that operates with binary numbers at its interface; however, employs the symbolic substitution technique for its processing. The idea was to convert the given binary operands to a non-binary representation, apply symbolic substitution technique for their processing, and finally convert the result to binary notation. For the study, the CMSD system, due to its capability of unique representation of numbers and carry-free parallel arithmetic, has been chosen as the intermediate non-binary representation. The focus has been the development of symbolic substitution tables and algorithms for all the phases involved like: conversion of binary to the non-binary representation, processing of the numbers and finally conversion of the non-binary representation to binary representation.

In any symbolic substitution process, there are two basic operations involved: the recognition of input patterns and the substitution of these patterns. A number of optical hardware systems for these operations have already been proposed [3, 6, 7, 19]. We have used in this study a simplified version of such a system as presented by Louri [6, 19]. This study, however, mainly concentrates on developing the symbolic substitution tables and algorithms involved in the proposed addition unit, rather than on designing the corresponding hardware.

An important factor in the context of symbolic substitution is the required number of substitution steps. The thesis presents a set of symbolic substitution tables and algorithms that require much less substitution steps than any earlier corresponding schemes. Another issue the study concentrates on is the development of a symbolic substitution based

approach for associative addition of binary numbers. Such an algorithm can be employed in arithmetic computation, like multiplication, where associative addition of a set of numbers is an integral process.

## 1.6 Organization of the Thesis

The thesis has been organized in different chapters, with each chapter discussing different aspects of the study. The areas covered by different chapters are briefly as follows:

- Chapter 2** Provides a literature review of the different methods and number systems that have been proposed in the context of carry-free addition
- Chapter 3** The concept of optical computing, symbolic substitution and related algorithms for arithmetic computation
- Chapter 4** A detailed discussion of the outcome of the study; introducing the different symbolic substitution based algorithms and tables
- Chapter 5** A computer analysis of reduction of steps introduced in the algorithms presented in chapter 4
- Chapter 6** A novel approach for symbolic substitution based associative addition
- Chapter 7** The concluding chapter giving some recommendations



## Chapter 2

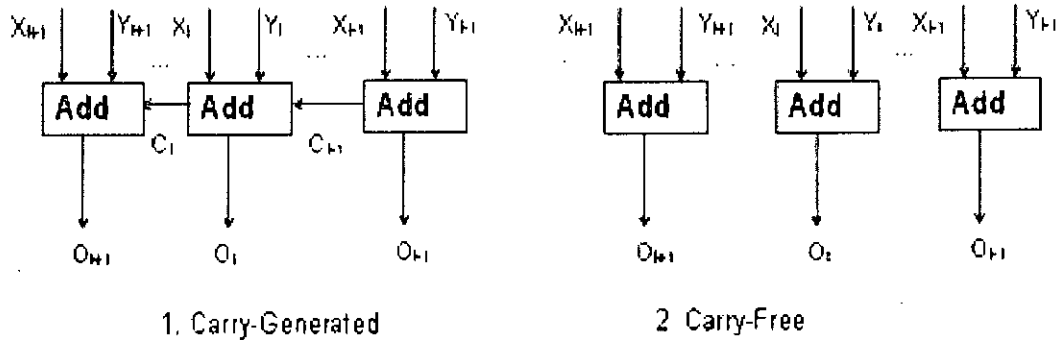
# CARRY-FREE ARITHMETIC: A LITERATURE SURVEY

### 2.1 Carry-free Arithmetic

Different researchers have developed a number of methodologies and number systems for the design of parallel carry-free arithmetic unit. Figure 2.1 illustrates what is meant by the term 'carry-free computation'. The left side of the figure depicts an arithmetic unit in which each output bit depends not only on the input bits, but also on the previous stage carries. The right side of the figure illustrates another arithmetic unit where each output bit depends on input bits only, but not on any processing result of previous stages. Thus while the left sided unit implies sequential processing, the right sided unit can derive all the resultant bits in parallel and independent of each other. The development goal of a carry-free addition unit is, hence, the design of a system, similar to the right sided unit of Figure 2.1, that can derive its operational result based on its operands only and not on any previous stage computations.

The efforts made in the context of the implementation of carry-free computation have been mainly in two perspectives. One is the design of computational system that process the digits in the computer representation of the operands in parallel. The other has been the design of number representation that allows the parallel arithmetic computing technique to be applied for carry-free computation. This chapter takes a look at these different techniques and number representation systems that have been proposed in computing literature.

Figure 2.1 Carry-free addition



## 2.2 Carry Lookahead Adder(CLA)

Carry Lookahead Adder(CLA) is an implementation proposed [20] to speed up the addition of two binary numbers. Let  $A_i$  &  $B_i$  be the addend and augend respectively in the  $i$ th bit position of a binary full adder. Also let two functions : the carry propagate,  $P_i$  and the carry generate,  $G_i$  be defined as follows:

$$P_i = A_i \oplus B_i$$

$$G_i = A_i \cdot B_i$$

Then the full adder equations can be written as follows:

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i \cdot C_i$$

Here the '+' symbol denotes logical 'or'.

Using this scheme, the addition of two binary numbers can be performed in parallel. However, the implementation is expensive, since it requires a lot of internal circuitry. For example, for a four-bit adder, the carries for various bit positions are calculated as follows:

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

while the sum bits can be calculated as follows:

$$S_1 = P_1 \oplus C_0$$

$$S_2 = P_2 \oplus C_1$$

$$S_3 = P_3 \oplus C_2$$

$$S_4 = P_4 \oplus C_3$$

As can be seen, with the increasing no. of bits, the circuit complexity and the required no. of gates increases. For large no. of input bits, this implementation becomes too expensive. Also, the inherent delay in the internal circuitry affects the computation speed and the circuit virtually becomes a sequential carry-affected adder.

## 2.3 Introduction of Redundancy in Number Systems

The alternative schemes to CLA, that have been proposed, involve the use of number systems with some kind of redundancy in the representation [8,9,11,21]. The redundancy allows multiple encoding for the same number and thus can be utilized for the development of carry-free or limited carry-propagation addition scheme. The next section focuses on some of these number systems, while the subsequent section discusses about the use of these number systems in the context of parallel carry-free addition.

## 2.4 Different Number Systems

Different number systems have been proposed for the computation of arithmetic operations by computers. These number systems may be categorized into two classes:

1. Weighted, where every integer  $X$  can be represented as in equation 2.1. Here  $r$  is the corresponding base and  $x_i \in \{0, 1, \dots, r-1\}$

$$X = \sum_{i=0}^n x_i r^i \quad (2.1)$$

2. Non-weighted, where digits are not assigned any specific weight and the ordering of the digits does not affect the representation.

The most common number systems like binary number system, hexadecimal number system are examples of the weighted number system. The Residue Number system (RNS) is an example of non-weighted number system.

In this section, we focus on the different number systems that have been proposed in computing literature in the context of carry-free parallel arithmetic. An outline of their relative advantages and disadvantages in this respect is also provided.

### 2.4.1 Redundant Number System

Avizienis [8] introduced the Redundant Number system. Formally, Redundant Number System is a redundant representation of a number that operates as follows [8, 22]:

For any radix,  $r \geq 2$ , a signed digit integer number,  $X = (x_{n-1}, \dots, x_1, x_0)$ , represented with  $n$  digits, has the algebraic value,  $X = \sum_{i=0}^{n-1} x_i r^i$ .

Here each digit  $x_i$  assumes its value in the digit set,

$$S = \{-\alpha, -\alpha + 1, \dots, -1, 0, 1, \dots, \alpha - 1, \alpha\}$$

The cardinality of the set  $S$  is  $2\alpha + 1$  and maximum digit magnitude,  $\alpha$  must satisfy,

$$\text{ceil}\left(\frac{r-1}{2}\right) \leq \alpha \leq r-1$$

here  $\text{ceil}(x)$  denotes  $\lceil x \rceil$

The redundant number system is also known as signed-digit number system in computing literature [8, 23].

A significant property of the redundant representation is that it allows limited carry-propagation or carry-free addition. This capability arises due to the redundancy of extra digits and the capability of representing number in more than one way.

Different variations of the redundant number system have been proposed in literature [2, 9, 10, 12, 18, 21]. However, the Modified Signed Digit and Canonical Modified Signed Digit number systems have found the most popularity among the researchers in the context of carry-free addition [1, 2, 14, 24–26].

### 2.4.2 Modified Signed Digit Number System

The Modified Signed Digit (MSD) number system is a redundant number system with  $\alpha = 1$ . An  $n$ -digit Modified Signed Digit (MSD) number can be represented as:

$$X = \sum_{i=0}^{n-1} x_i 2^i \quad (2.2)$$

where each digit  $x_i \in \{0, 1, \underline{1}\}$ . Here  $\underline{1}$  denotes  $-1$ .

MSD number system is thus a redundant binary number system employing the symbols '0', '1' and ' $\underline{1}$ '. The MSD number system allows carry-free and borrow-free arithmetic operations. It is essentially a superset of the binary number system. The inclusion of the extra symbol  $\underline{1}$  [denoting ' $\underline{1}$ '] makes the provision for parallel arithmetic operations.

A number in MSD number system is represented as a string of symbols, members of which are from the set  $\{0, 1, \underline{1}\}$ . For example,

$$\begin{aligned} 5 &= 1 \ 0 \ 1 = 4 + 0 + 1 \\ -5 &= \underline{1} \ 0 \ \underline{1} = -4 + 0 + -1 \end{aligned}$$

However, representation of a number in MSD system is not unique. For example the number 11 [using 6 bits] can be represented as any of the following ways:

$$\begin{aligned} 11 &= 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\ 11 &= 0 \ 1 \ 0 \ \underline{1} \ 0 \ \underline{1} \\ 11 &= 1 \ \underline{1} \ \underline{1} \ 0 \ 1 \ 1 \end{aligned}$$

This non-uniqueness of MSD number system is one of its disadvantages. However, a specialization of this number system, known as Canonical Modified Signed Digit (CMSD) number system, has been proposed [2, 26] that gives unique representation for each number.

### 2.4.3 Canonical Modified Signed Digit Number System

Canonical Modified Signed Digit (CMSD) number system is essentially a variant i.e. a recoding of the general Signed Digit number system. The property that is maintained in the representation is that no two consecutive digits are non-zero.

Formally, an  $n$ -digit Canonical Modified Signed Digit number can be represented as :

$$X = \sum_{i=0}^{n-1} x_i 2^i \quad (2.3)$$

where each digit  $x_i \in \{0, 1, \underline{1}\}$  and  $x_i x_{i-1} = 0; \forall i = 1 \dots n - 1$

The representation of number 11 [using 6 bits], for example, in CMSD is as follows:

$$11 = 0 \ 1 \ 0 \ \underline{1} \ 0 \ \underline{1}$$

The advantage of this number system is that representation is unique. Also, the sparseness of non-zero digits in the representation minimizes the number of addition and subtraction operations required in arithmetic computation. Canonic Modified Signed Digit (CMSD) is thus a good candidate for appliance in parallel computing architecture and fast processing.

#### 2.4.4 Residue Number System

Residue Number System(RNS) [27,28] is a non-weighted number system that allows carry-free addition and borrow-free subtraction. It uses positional bases that are relatively prime to each other. For example, using 2, 3 and 7 as the prime modula, we can represent the following numbers as shown:

Number	Bases: 2, 3, 7
0	0, 0, 0
7	1, 1, 0
11	1, 2, 4

Here the digit for each position is obtained by the modulus of the number to be represented divided by the prime factor for that position. The representation of '11' is thus obtained as:

$$11 = 1 (11 \bmod 2), \ 2 (11 \bmod 3), \ 4 (11 \bmod 7)$$

The dynamic range,  $DR$ , of the number system is the product of the prime moduli. The dynamic range for the example is thus 0 to 41. Negative numbers can also be accommodated by designating  $DR/2$  numbers as positive and the rest as negative. For example, using 2,3,7 as the prime moduli and employing Signed RNS, the following numbers may be represented as follows:

$$\begin{aligned} 11 &= 1, 2, 4 \\ -11 &= 1, 1, 3 \end{aligned}$$

The addition and subtraction are performed in digit by digit basis. The result of operation in each position is then represented as the modulus of the resultant divided by the prime factor at that position. The following examples (using 3,5,7 as the prime moduli) illustrate the operation:

$$\begin{array}{r} 21 + 13 \\ 0, 1, 0 \\ + \\ 1, 3, 6 \\ \hline 1, 4, 6 \\ =34 \end{array} \qquad \begin{array}{r} 17 - 38 \\ 2, 2, 3 \\ + \\ 1, 2, 4 \\ \hline 0, 4, 0 \\ =-21 \end{array}$$

Residue Number system thus allows carry-free arithmetic. However, it is more difficult to implement due to the fact that the system computation elements require a different set of prime-moduli-based logic elements for each arithmetic operation. It is thus a rather complex system.

#### 2.4.5 Other Variations of Redundant Number System

The redundant signed-digit number representations is of symmetrical type in the sense that the allowed digit range is from  $-\alpha$  to  $\alpha$ , for some value of  $\alpha$  satisfying equation 2.4.1. It is now referred to as the Ordinary Signed Digit (OSD) number system in contemporary literature [12].

An extension of this system was later proposed and is known as the Generalized Signed Digit (GSD) number system [9, 21]. The GSD number system for radix  $r > 1$  has the digit set  $-\alpha, \dots, -1, 0, 1, \dots, \beta$ , where  $\alpha \geq 0$  and  $\beta \geq 0$ . The system has been proposed for use in schemes like carry-save adders and carry-skip adders [17].

Another variation of signed digit number system, that has been recently proposed, is the Asymmetric High-radix Signed Digit (AHSD) number system [12]. The radix- $r$  asymmetric high-radix signed-digit (AHSD) number system, denoted AHSD( $r$ ), is a positional weighted number system with the digit set  $S = -1, 0, \dots, r-1$ , where  $r > 1$ . The AHSD number

system is a minimally redundant system with only one redundant digit in the digit set. An  $n$ -digit number  $X$  in AHSD( $r$ ) is represented as

$$X = (x_{n-1}, x_{n-2}, \dots, x_0) \tag{2.4}$$

where  $x_i \in S$  for  $i = 0, 1, \dots, n - 1$ , and  $S = -1, 0, 1, \dots, r - 1$  is the digit set of AHSD( $r$ ). The value of  $X$  can be represented as,  $X = \sum_{i=0}^{n-1} x_i r^i$ .

### 2.4.6 Double Base Number System

The Double Base Number System (DBNS) [11] is a 2-D representation of numbers with 2 and 3 used as bases and using 0 and 1 as the only allowable digits. The representation of a given number  $x$  is of the form:

$$X = \sum_{i,j} x_{i,j} 2^i 3^j; x_{i,j} \in \{0, 1\} \tag{2.5}$$

A simple geometric interpretation of the system may be denoted as in the following table:

	$2^0$	$2^1$	$2^2$	$2^3$
$3^0$	$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$
$3^1$	$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$
$3^2$	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$
$3^3$	$x_{3,0}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$

For example, one representation of the number '11' is as follows:

	$2^0$	$2^1$	$2^2$	$2^3$
$3^0$	0	0	0	1
$3^1$	1	0	0	0
$3^2$	0	0	0	0
$3^3$	0	0	0	0

The DBNS system have a sparseness in its representation and is thus a suitable number system for application in parallel computing architecture. However, we require much



more storage for the representation of a DBNS number than its binary notation. Also, the representation of a number is not unique.

## 2.5 Carry-free Addition Schemes

A number of schemes, based on the aforementioned number systems, have been designed. These schemes, described under different titles in computing literatures like carry-skip (single- or multilevel) adder, carry-select adder, carry-save adder and hybrid adder [5, 9, 17, 18, 29], deal with the generation of carry by limiting carry propagation to within a small number of bits or detecting the end of propagation; or speeding up propagation via some lookahead scheme or eliminating carry propagation altogether. Several digital and VLSI systems have been proposed on the basis of these schemes [9, 17, 29, 30].

Another arena, where these non-binary number systems have found their application in the context of carry-free arithmetic, is the symbolic substitution process in optical computing. Detailed description of the process has been presented in the next chapter.

## 2.6 Choice of Number System in Carry-free Addition Scheme

Each number system has its advantages and disadvantages in the context of a carry-free arithmetic scheme. Usually the questions that are asked while selecting a number system are as follows:

1. Does the number system represent every number? (completeness)
2. Are the representation of the numbers unique for every number?  
(non-ambiguous)
3. Does the number system give any advantage in employing arithmetic operations and other operations like sign detection and comparison?

Not all advantages can be gained from employing a single number system. Binary system, for example, is the simplest to implement. But it does not allow carry free operation. Redundant number system, on the other hand allows carry free arithmetic. But it is harder to implement and the representation is not unique. The same problem holds for other variants of redundant number system, residue number system and double base

number system as well. Hence no single number system is the best choice and the choice solely depends on the application and algorithm to be implemented.

The MSD number system (which is a subset of the redundant number system) and its variant the CMSD number system, though having an extra digit than binary, allow carry-free arithmetic and thus are suitable for parallel computing architecture. It is used specially in optical computing where polarization and other properties of light may be utilized to denote the three different states as required in the representation [14]. Thus the symbolic substitution based architectures have always concentrated on using these trinary systems [2, 16, 19, 31] and in this thesis also, focus will be made on the utilization of these trinary number systems.

## Chapter **3**

# **SYMBOLIC SUBSTITUTION AND ARITHMETIC COMPUTATION**

### **3.1 Optics as a Backbone of Fast Computation**

The speed of conventional computers is mainly constrained by the interconnection bottleneck and inherent delay within the electronic devices. So efforts have been made to achieve speedup by miniaturizing electronic components to a very small micron-size scale so that those electrons need to travel only very short distances within a very short time. This goal of improving computer speed has thus resulted in the development of the Very Large Scale Integration (VLSI) technology with smaller device dimensions and greater complexity. However the VLSI technology is approaching its fundamental limits in the sub-micron miniaturization process. To meet the demand of high speed processing, we therefore need to find alternative fast computing system.

Optical interconnections and optical integrated circuits may provide a way out of the limitations to computational speed and complexity inherent in conventional electronics. They are immune to electromagnetic interference and free from electrical short circuits. They have low-loss transmission and provide large bandwidth; i.e. multiplexing capability, capable of communicating several channels in parallel without interference. They are capable of propagating signals within the same or adjacent fibers with essentially no interference or cross-talk.

Another advantage of optical methods over electronic ones for computing is that optical data processing can be done much easier in parallel than can be done in electronics. Parallelism is the capability of the system to execute more than one operation

simultaneously. Electronic computer architecture is, in general, sequential, where the instructions are implemented in sequence. This implies that parallelism with electronics is difficult to construct. Parallelism in conventional electronic computing system has focussed mainly on the use of multiple processors in conjunction with the computer memory to enhance the speed. However, the effect of using a large number of processors are not necessarily an increase of computational speed, but could be in fact detrimental. This is because as more processors are used, there is more time lost in communication. On the other hand, using a simple optical design, an array of pixels can be transferred simultaneously in parallel from one point to another. Thus optical system that works simultaneously with an array of numbers, represented using some kind of spatial arrangement of pixels, and derive the arithmetic computation in parallel has been developed based on this inherent parallelism of optics [14, 19].

## 3.2 Symbolic Substitution

Symbolic substitution [14, 19, 31–35] technique was developed as an optical computing method to take advantage of the optical parallelism for 2-D image processing and arithmetic computation. At the heart of the process is a pattern replacement operation, defined by a symbolic substitution rule, that converts some given patterns to some desired patterns. The substitution process, therefore, comprises two basic phases. In the first phase, *the recognition phase*, the inputs are scanned for the location of desired input patterns. In the second phase, *the substitution phase*, the desired output pattern is written into all the locations determined by the previous phase.

## 3.3 Different Symbolic Substitution Methods

In order to process an information by a device, the data has to be encoded with a physical quantity that the device can process. While in electronic computers information is encoded with an electrical quantity (electrical voltage or current), in optics the information is encoded with an optical quantity (optical intensity, polarization, amplitude or phase). Optical intensity and the optical polarization are the two optical quantities that have been used extensively in optical computing in the context of symbolic substitution process. And according to the type of encoding, the symbolic substitution methods have been classified into two categories: intensity coded symbolic substitution (ICSS) [14, 33, 34] and polarization coded symbolic substitution (PCSS) [36]. Another method, symbolic

substitution using shadow-casting, employs both ICSS and PCSS [1, 35].

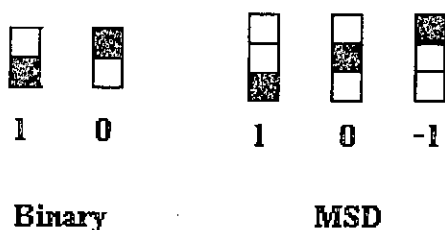
### 3.4 Intensity Coded Symbolic Substitution

Intensity coded symbolic substitution (ICSS) uses the information coded with the optical intensity. The basic units in this coding are pixels which can be either transparent or opaque. A pixel or certain spatial combination of pixels represents a certain value of the encoded information.

#### 3.4.1 Dual-rail Encoding

The simplest way of encoding an information in binary form is to attach the logical one to the transparent pixel and the logical zero to the opaque pixel. However, optical computing uses a scheme termed as *Dual-rail Encoding* [2, 14] that uses more than one pixel to represent information. In dual-rail encoding a spatial arrangement of the transparent and opaque pixels is treated as a logical one and another arrangement is treated as logical zero as shown in figure 3.1. The dark and white pixels, in this figure, indicate the opaque and the transparent pixels respectively. As has been illustrated, in dual-rail encoding, pixels are arranged into a two-pixel vertical bar, where the logical one is represented by the transparent pixel in the upper position and by the opaque pixel in the lower position and logical zero is represented vice versa. Similar spatial arrangement of the transparent and opaque pixels are also used to represent 1, 0 and -1 in the MSD representation.

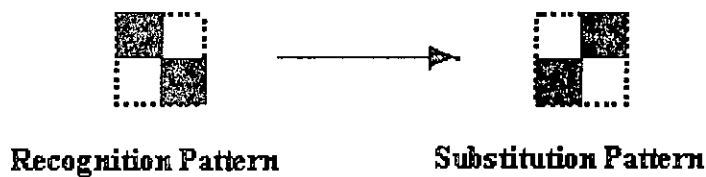
Figure 3.1 Dual-rail encoding



### 3.4.2 Symbolic Substitution Process

Symbolic substitution operation is characterized by the substitution rule. Figure 3.2 shows an example substitution rule that will be used to explain the method. The pattern at the left is the recognition pattern that has to be located in the input array. The pattern at the right is the substitution pattern, which has to replace the located pattern.

Figure 3.2 Substitution rule



#### Recognition Phase

In the recognition phase all occurrences of a recognition pattern are located. First a pixel of the recognition pattern is defined as a reference pixel. This reference pixel will determine the position of the recognition pattern in a masked array, which is a recognition phase output array. Figure 3.3 shows the recognition pattern and the reference pixel, which is the lower left pixel of the 2 X 2 square surrounding the pattern. Naturally, any other choice is also possible. The pixels in the input or output patterns are also referred to as 'pels' and we will use this term for the subsequent discussion. The first step in the recognition phase is to produce as many replicas of the input array as there are significant (opaque) pels in the left hand side (LHS) of the rule. For the demonstrated example, the number is two. One replica is associated with the upper left pixel and the other replica with the lower right pixel. The significant pels are then shifted onto the reference pixel. For the example, the replica associated with the upper left pel is shifted one pixel down and the replica associated with the lower right pel is shifted one pixel to the left. These two replicas are then superimposed to produce the superposition array.

The second step is an operation of inversion (Figure: 3.4). All dark pixels are inverted onto bright pixels and vice versa.

Figure 3.3 Symbolic substitution: Recognition phase (step 1)

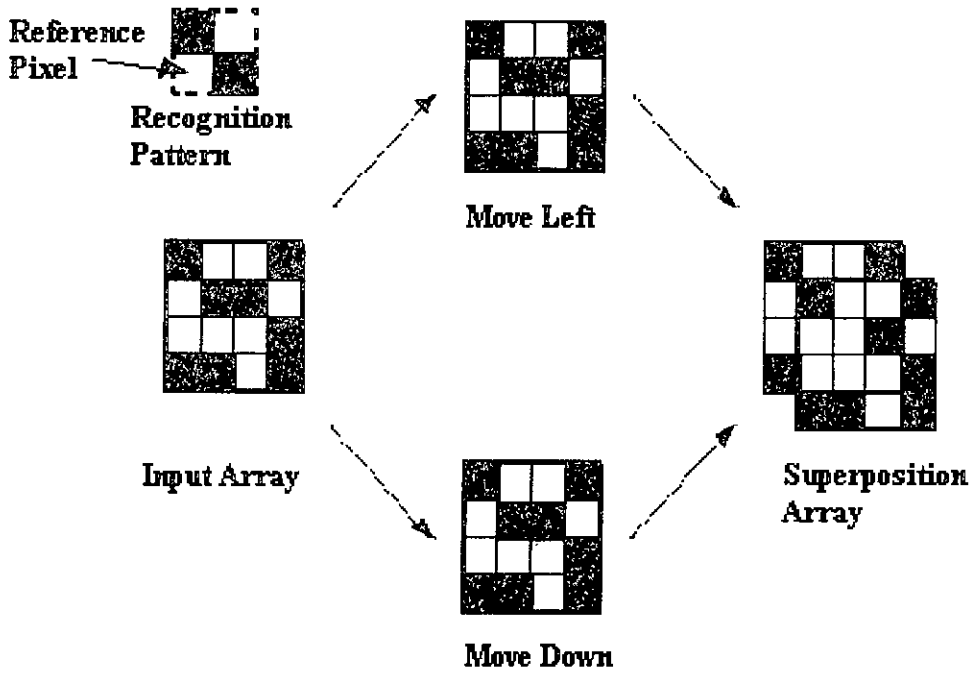
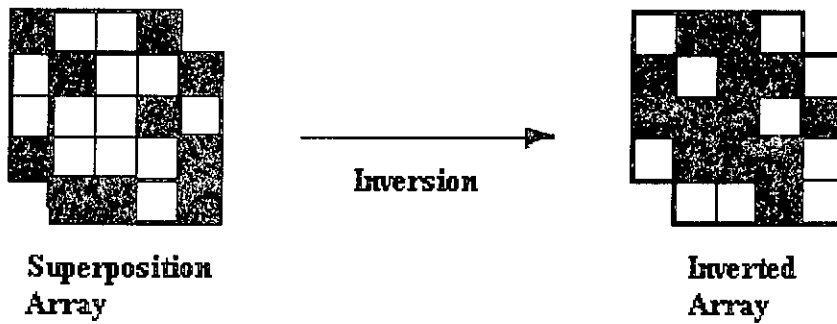
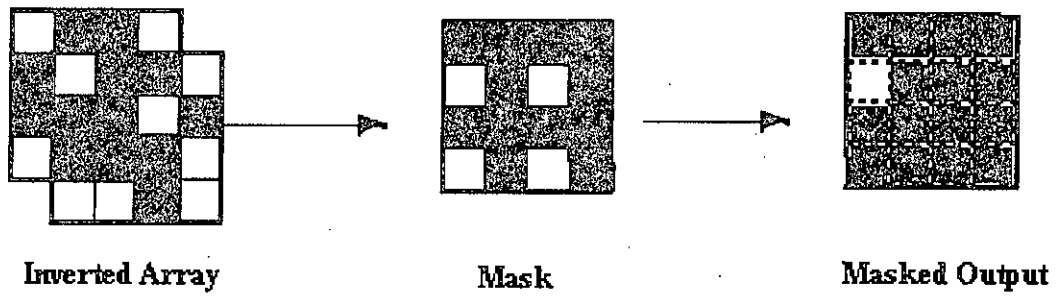


Figure 3.4 Symbolic substitution: Recognition phase (step 2)



The final step in the recognition phase (Figure: 3.5) is an operation of masking. The mask selects only the pels whose positions in the inverted array coincide with positions of holes in the mask. The output of this operation is an array with the bright pels determining the occurrences of the recognition pattern. The remaining pels are dark.

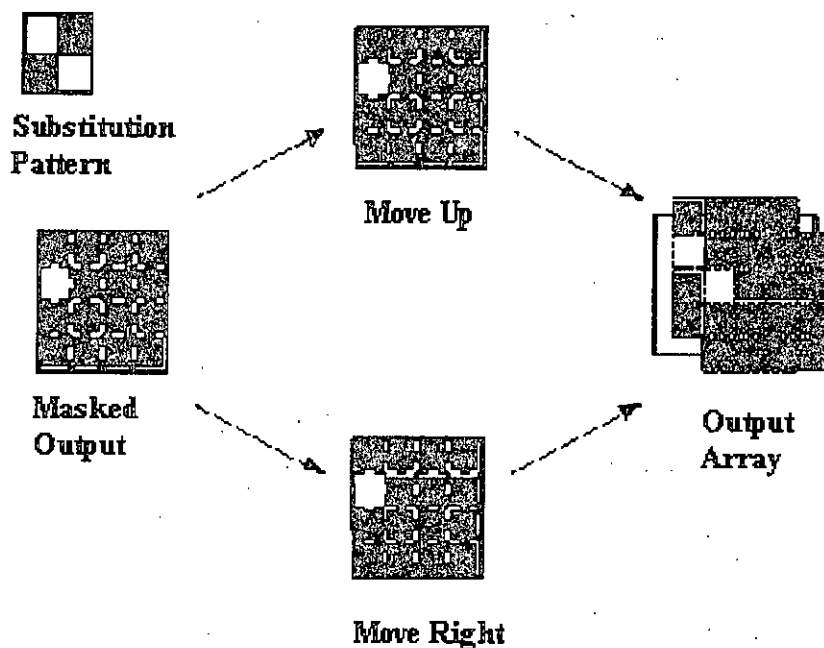
Figure 3.5 Symbolic substitution: Recognition phase (step 3)



### Substitution Phase

The first step in the substitution phase is to create as many copies of the masked output as there are bright pels in the substitution pattern. As shown in Figure 3.6, the substitution pattern, for the example, contains two bright pels and therefore two copies are made. To scribe the substitution pattern, one copy is shifted to the top and the other copy to the right. The copies are then superimposed to obtain an output array:

Figure 3.6 Symbolic substitution: Substitution phase

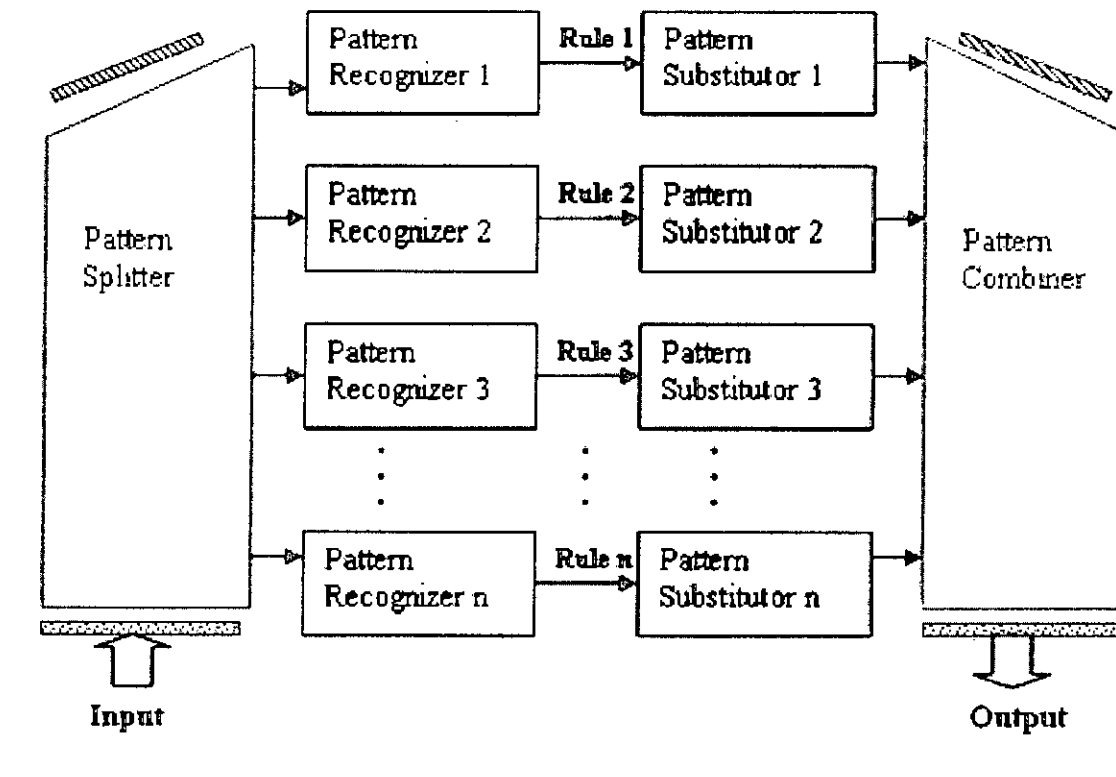




### 3.4.3 Multiple-rules Symbolic Substitution Process

In a typical symbolic substitution process, a set of symbolic substitution rules, stored in the storage element of the unit, are applied to detect a set of input patterns and scribe a set of output patterns. The unit consists of pattern recognizer and pattern substituter for each set of rules (Figure: 3.7). The pattern splitter splits the input for the recognition phase, while the pattern combiner combines the substituted patterns at the output.

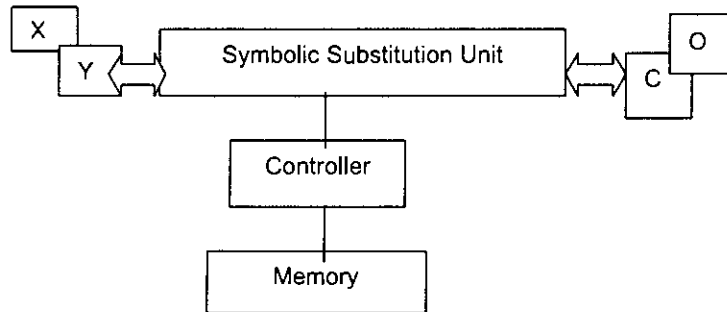
Figure 3.7 Multiple substitution rules



### 3.5 Polarization Coded Symbolic Substitution

Polarized coded symbolic substitution (PCSS) is another implementation of symbolic substitution logic. The basic feature of this method is that an information is coded in terms of polarization. The input data array in PCSS looks like in ICSS, but with one exception. Instead of bright and dark pels the input array consists of pels containing horizontally or vertically polarized light. In essence, the basic principle of PCSS is the same as in ICSS and consists of replication, shifting, superimposing, inverting and masking operations as in the ICSS. However, PCSS offers four logic values as compared to two logic

Figure 3.8 Block architecture of a symbolic substitution unit



values in ICSS. The states 'zero' and 'one' have the same meaning as in ICSS. In addition PCSS appears allows two new logic states, that after Brenner [36], are referred to as the 'never match' and 'always match' states. The state 'never match' can be used to inhibit the recognition, whereas the state 'always match' can serve as a wild card or don't care condition. Thus PCSS allows the implementation of complex symbolic substitution rules.

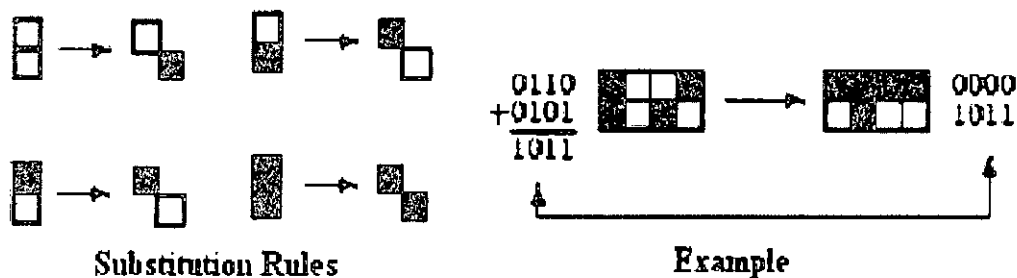
### 3.6 Basic Symbolic Substitution Unit

Figure 3.8 depicts the basic block architecture of a symbolic substitution unit. The inputs to the system are the variables  $X$  &  $Y$ .  $C$  is an auxiliary variable and  $O$  is the output variable. The symbolic substitution rules are stored in memory, while the controller controls the recognition and substitution phase. Inputs,  $X$  &  $Y$  are scanned for recognition patterns during the recognition phases. During the substitution phase, the controller substitutes these patterns with the substitution patterns, defined by the symbolic substitution rules, at the output  $O$ . Some applications require multiple substitution steps. In these cases, the auxiliary variable  $C$  is used as another output during the substitution phase. For the subsequent iteration, the values of  $O$  and  $C$  are copied back to  $X$  and  $Y$  and the symbolic substitution process is repeated with the new values of  $X$  and  $Y$ .

### 3.7 Symbolic Substitution and Arithmetic Computation

Symbolic substitution can be used for implementing arithmetic operations [2, 6, 14, 31, 33, 36]. The idea is to consider the representation of the operands as input patterns and the resultant as the output patterns. The symbolic substitution rules define the corresponding arithmetic operation. As an example, figure 3.9 illustrates an addition following the rules:  $1 + 1 = 10$ ,  $1 + 0 = 0 + 1 = 1$ ,  $0 + 0 = 0$ . ICSS process has been used for the illustration and a bright pixel, in the figure, represents the logical one, while a dark pixel represents the logical zero. The symbolic substitution rules corresponding to the addition operation have been shown at the left side of the figure. The right side shows the addition of two numbers (0110) and (0101), the addition of which equals (1011). As has been shown, the symbolic substitution process simply detects the desired input patterns in the operands and substitutes these patterns with the substitution rule-defined patterns at the output. There are four substitution rules and are applied as many times as there are bits in the numbers (in this example four times). Obviously, this addition can also be implemented with dual-rail coded or polarization coded numbers provided they use the corresponding substitution rules.

Figure 3.9 Symbolic substitution based arithmetic computation



### 3.8 Symbolic Substitution Based Computation Algorithms

This section presents a set of symbolic substitution based algorithms that have been proposed in computing literature for different arithmetic computation. These algorithms are mainly based on MSD or CMSD representation of numbers. The reason why these non-binary systems have found their popularity in the context of symbolic substitution is, as mentioned in chapter 2, the capability of these systems to perform carry-free arithmetic operations and thus the suitability of these systems to be employed in the inherently

parallel symbolic substitution process.

### 3.8.1 Three-step MSD Addition

Table 3.1 shows the rule for 3-step MSD addition.

Table 3.1: Symbolic substitution tables for MSD addition: Three-steps

$A_i B_i$	$T_{i+1} W_i$	$T_i W_i$	$\hat{T}_{i+1} \hat{W}_i$
00	00	00	00
01	<u>11</u>	01	01
<u>01</u>	<u>11</u>	<u>01</u>	<u>01</u>
10	<u>11</u>	10	<u>11</u>
11	10	11	01
<u>11</u>	00	<u>11</u>	00
<u>10</u>	<u>11</u>	<u>10</u>	<u>01</u>
<u>11</u>	00	<u>11</u>	00
<u>11</u>	<u>10</u>	<u>11</u>	<u>10</u>

The Final Sum:  $S_i = \hat{T}_i + \hat{W}_i$

The input  $A$  and  $B$  are scanned for the desired input patterns,  $(A_i B_i)$ , as depicted in the substitution table. In the first step, all these patterns are substituted by transfer and weight bits  $(T_i W_i)$  according to the defined rule. These bits are then substituted by  $\hat{T}_i \hat{W}_i$  according to the symbolic substitution table, in the second step. The last step is the direct substitution according to the addition rule.

**Example:**

$$27 + 15$$

1	1	1	0	<u>1</u>
1	0	0	0	<u>1</u>
1	1	1	0	<u>1</u>
	0	<u>1</u>	<u>1</u>	0 0

$$\begin{array}{rcccc}
 0 & 0 & 0 & 0 \\
 1 & 0 & \underline{1} & \underline{1} \\
 \hline
 1 & 1 & 0 & \underline{1} & \underline{1} & 0 \\
 \hline
 = 42
 \end{array}$$

The underlying concept of the process is that, when two single digit MSD numbers are added, a carry is generated whenever the to-be-added numbers are either 1,1 or  $\underline{1},\underline{1}$ . This carry is locally absorbed by allowing one bit propagation to the left. The generation of such carry may be avoided by mapping the two digits in question into an intermediate sum and an intermediate carry (known as weight and transfer bits) such that the  $n$ -th intermediate sum and the  $(n - 1)$ -th intermediate carry never form any of the carry-generation combinations. Thus the following steps may be used in deriving the required rules for realizing carry-free addition:

1. The set of MSD addend and augend digits are grouped into multiple classes in accordance with the value of their sums
2. For each of the above classes, all possible two-digit MSD representations of the corresponding sum are identified
3. Of the MSD representations derived in the previous step, the less significant digit is taken to represent the intermediate sum while the other digit is taken to represent the intermediate carry. The next step identifies the form of intermediate sum and carry.
4. A substitution rule for the addend and augend is selected from the different representations derived in Step 2 above. The substitution rule for a group of paired numbers that has the same sum is derived by examining the  $(n - 1)$ -th pair of digits; so that the  $n$ -th intermediate sum when added to the  $(n - 1)$ -th intermediate carry, does not generate a carry. For each substitution rule selected for a particular group of inputs, the allowable set of the  $(n - 1)$ -th pair of digits must be specified so that the intermediate sum,  $S_n$  and intermediate carry  $C_{n-1}$  generated from the group satisfy the equation:

$$|S_n + C_{n-1} < R|$$

where  $R$  is the radix of the number system under consideration; for MSD,  $R=2$ .

Table 3.3: Symbolic substitution tables for MSD addition: Two-steps

$A_i B_i$	$A_{i-1} B_{i-1}$	$T_{i+1} W_i$
00	Don't Care	00
<u>1</u> 1	Don't Care	00
<u>1</u> <u>1</u>	Don't Care	00
01	Both Positive	<u>1</u> <u>1</u>
01	Otherwise	01
10	Both Positive	<u>1</u> <u>1</u>
10	Otherwise	01
0 <u>1</u>	Both Negative	<u>1</u> <u>1</u>
0 <u>1</u>	Otherwise	0 <u>1</u>
<u>1</u> 0	Both Negative	<u>1</u> <u>1</u>
<u>1</u> 0	Otherwise	0 <u>1</u>
<u>1</u> <u>1</u>	Don't Care	<u>1</u> 0
11	Don't Care	10

### 3.8.2 Two-step MSD Addition

The Table 3.3 shows the rule for 2-step MSD addition. The symbolic substitution involved here is an example of a conditional symbolic substitution process [16]. It is so called since the substitution here depends on some input bits other than the corresponding addend and augend bits.

**Example:**

$$\begin{array}{r}
 102 + -47 \\
 \hline
 1 \ 1 \ 0 \ 1 \ 0 \ \underline{1} \ 0 \\
 \underline{1} \ 1 \ 1 \ \underline{1} \ \underline{1} \ \underline{1} \ 1 \\
 \hline
 0 \ 1 \ 1 \ 0 \ \underline{1} \ \underline{1} \ 0 \\
 0 \ 0 \ \underline{1} \ 0 \ 1 \ 0 \ 1 \\
 \hline
 0 \ 1 \ 1 \ \underline{1} \ \underline{1} \ 0 \ 0 \ 1 \\
 \hline
 = 55
 \end{array}$$

The underlying concept is similar to that of the three-step algorithm. The pair of MSD digits to be added have been classified in five groups in this case. Now, the summation of two MSD digits ranges from -2 to 2 and the sum can be represented as two-digit MSD representation as follows:

(a)  $-2 = \underline{1} 0$

(b)  $-1 = 0\underline{1}, \underline{1} 1$

(c)  $0 = 00$

(d)  $1 = 01, 1\underline{1}$

(e)  $2 = 10$

Of these options, rules for (a), (c) and (e) are fixed since there is only one way to represent the numbers in those cases. For finding substitution rules for (b) and (d), it may be noted that they have a possible intermediate sum digit of either 1 or  $\underline{1}$ . So, to determine the substitution rule, the digits at the previous positions are also considered. When the intermediate sum at the  $n$ -th position is 0, any combinations of digits may appear at the  $(n-1)$ -th position without generating a carry. On the other hand if the intermediate sum at the  $n$ -th position is 1, then a carry will be generated only if the intermediate carry from the  $(n-1)$ -th position is also 1. In other words if the digit pair at the previous position is anything but all negative (that is, anything but (1,1) or (0,1) or (1,0)) then a carry will not be generated. In this case we take the representation of intermediate carry and sum to be respectively 0 and 1. On the other hand if both the previous digits are positive, the intermediate carry and sum is taken to be 1 and  $\underline{1}$ . If the intermediate sum at the  $n$ -th position is  $\underline{1}$ , a similar requirement is taken into consideration for the design of the symbolic substitution rules.

### 3.8.3 One-step CMSD Addition

As has been illustrated, carry-free addition of two numbers represented in MSD presentation can be performed in 3 or 2 steps. However if the operands are represented in CMSD notation, then addition can be performed by a direct symbolic substitution table and in one step only. The Table 3.4 shows the one-step symbolic substitution table for CMSD addition.

The idea of the algorithm is to consider the CMSD representation of the two operands,  $A$  and  $B$  as input patterns and derive the MSD representation of their summation as output pattern and using one symbolic substitution step. For each pair of bits,  $A_i A_{i-1}, B_i B_{i-1}$ , the resultant bits,  $S_i$ , are derived by a simple substitution and in a parallel carry-free manner,  $\forall i = 0, \dots, n$  where  $n$  is the bit-size of the operands.

Table 3.4: Symbolic substitution tables for CMSD addition: One-step

$A_i B_i$	$A_{i-1} B_{i-1}$	$S_i$
<u>1</u> 0	0D	<u>1</u>
0 <u>1</u>	D0	<u>1</u>
00	<u>11</u>	<u>1</u>
01	D0	1
10	0D	1
00	11	1
00	0D	0
00	<u>11</u>	0
00	<u>11</u>	0
00	<u>11</u>	0
<u>11</u>	00	0
<u>11</u>	00	0
<u>11</u>	00	0
11	00	0

D = don't care

Although the addition can be performed in one step, the problem with this algorithms is that it does not maintain the Canonic property in result. Thus if we require to perform associative addition of a set of numbers, then the algorithm is to be applied repeatedly for each pair of numbers with the implied overhead of conversion of MSD number to CMSD for application in the subsequent associative addition.

### 3.8.4 Conversion to CMSD

Conversion from binary or MSD representation to CMSD is an important process in many computations [2, 25, 26]. Reitwiesner's algorithm [37] has been the most cited one [25, 26] in this regard. Reitwiesner's Algorithm requires that the binary expansion of a number  $x$  is padded with an initial zero. The algorithm computes the signed-digit representation  $y$  starting from the least significant digit and proceeding to the left. Initially an auxiliary binary variable  $c$  is set to 0 and subsequently the binary expansion of  $x$  is scanned. At each parallel step, the canonically recoded digit  $y_i$  and the next value of the auxiliary binary variable  $c_{i+1}$ , for  $i = 0 \dots n-1$ , are calculated using the values of  $x_i$ ,  $x_{i+1}$ , and  $c_i$  according to a conversion and complementary conversion table as shown in Table 3.5 [2, 37]. The complete CMSD representation is obtained after  $n + 1$  parallel steps. A corresponding algorithm, as presented in [2], has been reproduced in algorithm 3.1.



---

**Algorithm 3.1** Conversion to CMSD (Reitwiener's algorithm)

---

Let:  $X = x_{n-1}x_{n-2} \dots x_0$ , be an n-digit Signed-digit number

Set  $x_n = 0$  {Padding 0 at Left}

for  $i = 1$  to  $n - 1$  do

  if  $x_i \neq 0$  then

    if  $x_i > 0$  and  $x_{i+1} \geq 0$  then

      if  $x_i + x_{i+1} \geq 2$  then

$x_i = x_i - 2$

$x_{i+1} = x_{i+1} + 1$

    else

      if  $x_i < 0$  and  $x_{i+1} \leq 0$  then

        if  $x_i + x_{i+1} \leq -2$  then

$x_i = x_i + 2$

$x_{i+1} = x_{i+1} - 1$

      else

        if  $x_i > 0$  and  $x_{i+1} < 0$  then

          if  $x_i \geq x_{i+1}$  then

$x_i = x_i - 2$

$x_{i+1} = x_{i+1} + 1$

        else

          if  $x_i < 0$  and  $x_{i+1} > 0$  then

            if  $-x_i \geq x_{i+1}$  then

$x_i = x_i + 2$

$x_{i+1} = x_{i+1} - 1$

Table 3.5: Reitwiesner's conversion table and corresponding complementary conversion table

$c_i$	$x_{i+1}$	$x_i$	$c_{i+1}$	$y_i$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	<u>1</u>
1	0	0	0	1
1	0	1	1	0
1	1	0	1	<u>1</u>
1	1	1	1	0

$c_i$	$x_{i+1}$	$x_i$	$c_{i+1}$	$y_i$
0	0	0	0	0
0	0	<u>1</u>	0	<u>1</u>
0	<u>1</u>	0	0	0
0	<u>1</u>	<u>1</u>	<u>1</u>	1
<u>1</u>	0	0	0	<u>1</u>
<u>1</u>	0	<u>1</u>	<u>1</u>	0
<u>1</u>	<u>1</u>	0	<u>1</u>	1
<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	0

### 3.9 Concluding Remarks

This chapter has introduced the idea of symbolic substitution for arithmetic computation.

---

The different algorithms and tables presented here have different limitations and advantages. The next chapter depicts the main objective of the study and presents a set of improved symbolic substitution tables and algorithms.

## Chapter 4

# DESIGN OF A SYMBOLIC SUBSTITUTION BASED BINARY ARITHMETIC UNIT

### 4.1 Underlying Motivation

As has been mentioned in the previous chapter, the earlier proposed symbolic substitution based methods and algorithms have concentrated mainly on the use of non binary number systems like MSD and CMSD. Binary system, due to its incapability of processing information in parallel, has not found much application in this context.

Traditional digital computing, however, is based on binary system and corresponding binary logic. Also a binary system is simpler to implement than a corresponding non-binary based system. This study has therefore made an effort to develop system that works with binary numbers, yet use the symbolic substitution method for fast arithmetic computation. As addition is the basic operation involved in any arithmetic computation, the focus has been on the development of a symbolic substitution based fast, carry-free binary adder.

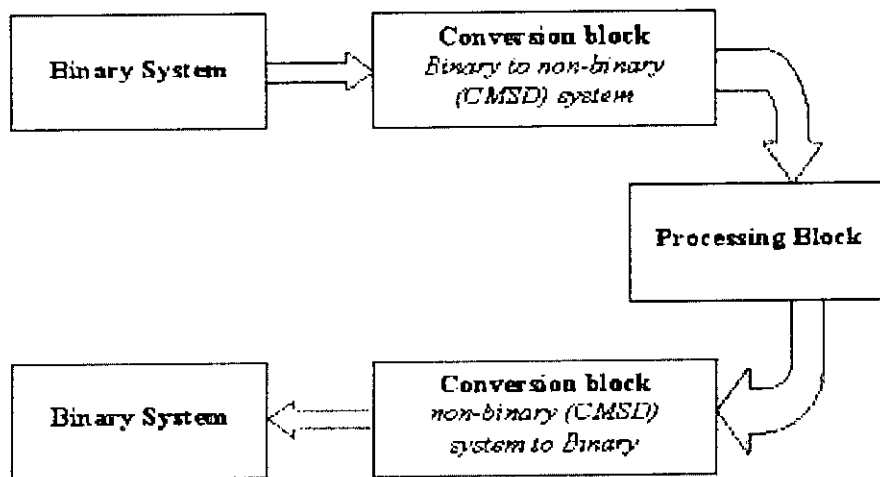
### 4.2 Proposed Arithmetic Unit

A logical diagram of the proposed arithmetic unit has been shown in Figure 4.1. The idea has been the design of a system that works with binary numbers at its interface, but uses an intermediate non-binary representation that can process data in parallel using the symbolic substitution process. The goal has been the design of symbolic substitution

logic for all the processing steps involved like the conversion of binary to the non-binary representation, processing of the non-binary numbers and finally the conversion of the non-binary representation to binary representation. The proposed unit works with binary numbers represented in both unsigned and 2's complement form. As for the intermediate non-binary representation, both MSD and CMSD are suitable. However CMSD system is preferable since it has a unique representation for every number. And also it has a sparseness among the non-zero digits in the representation and this sparseness is suitable for fast (in terms of the required number of substitution steps) symbolic substitution process.

A practical application of such unit may be an opto-electronic arithmetic unit that accepts binary numbers at its input and produces the result in binary form, but process the numbers optically using the symbolic substitution logic within the device. Such unit will result in higher computation speed than the sequential digital computing devices.

**Figure 4.1** Logical diagram of the proposed fast addition unit



### 4.3 Different Phases of the Proposed Arithmetic Unit

In this section, we take a look at the different phases involved in the proposed addition unit and develop symbolic substitution based algorithm for each of these phases.

**Algorithm 4.1** Algorithm to convert MSD and binary number to CMSD in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps

Let  $X=X_{n-1}, \dots, X_1, X_0$  be a given MSD or unsigned binary or 2's complement binary number,  $C=C_{n+1}, \dots, C_1, C_0$  be an auxiliary variable used in symbolic substitution process,  $Y=Y_{n+1}, \dots, Y_1, Y_0$  be the output CMSD number and  $CT=CT_{n+1}, \dots, CT_1, CT_0$  be the temporary auxiliary variable generated in each symbolic substitution step.

Set  $C=0$

if input is in 2's complement Binary representation then

    Pad 2 'MSB' digits at Left

else

    Pad 2 '0' digits at Left

for stepcount = 1 to  $\lfloor n/2 \rfloor + 1$  STEP 1 do

    \*(

    for  $i = 0$  to  $n+1$  STEP 2 do

        Set  $Y_{i+1}Y_i$  and  $CT_{i+3}CT_{i+2}$  based on  $X_{i+1}X_i$  and  $C_{i+1}C_i$  according to the symbolic substitution table;  $0 \leq i, (i+1), (i+2), (i+3) \leq n+1$ ; else  $Y_i = CT_i = 0$ . If there is an entry for further check(CheckCond.1 or CheckCond.2), then perform the substitution based on  $X_{i+2}$  and  $C_{i+2}$  according to the corresponding Check-condition table.

    )\*

    Set  $X = Y$ , Set  $C = CT$

The derived  $Y=Y_{n+1}, \dots, Y_1, Y_0$  is the output CMSD number

**NB:** The \*-marked code runs in parallel for all digits

### 4.3.1 Conversion of MSD and Binary to CMSD

The first phase is the conversion of binary numbers to CMSD number. As already stated in section 3.8.4, Reitwiesner's algorithm [37] is the most cited one [25, 38] in this regard. However, a symbolic substitution table based on this algorithm requires steps proportional to the number of digits in the representation of the number. If  $n$  is the bit-size of the binary or MSD number to be converted, a symbolic substitution process based on the Reitwiesner's algorithm will require  $n + 1$  substitution steps to complete the conversion.

In this section, we present a new symbolic substitution table and corresponding algorithm that requires  $\lfloor \frac{n}{2} \rfloor + 1$  substitution steps. One highlighting feature of this algorithm is that it can convert binary numbers represented in both signed 2's complement form and unsigned form to their correct representation in CMSD. Moreover, it can convert any given MSD number to CMSD representation. The algorithm and the symbolic substitution table has been shown in Algorithm 4.1 and Table 4.1. The demonstration of the algorithm has been shown in Demonstration 4.3.1, 4.3.2 and 4.3.3.

Table 4.1: Symbolic substitution table to convert MSD and binary number to CMSD in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps

Bits ( $X_{i+1}X_i$ )	Check: $C_{i+1}C_i$ Outputs( $Y_{i+1}Y_i, CT_{i+3}CT_{i+2}$ )		
	00	01	0 <u>1</u>
00	00, 00	01, 00	0 <u>1</u> , 00
01	01, 00	CheckCond.1	00, 00
0 <u>1</u>	0 <u>1</u> , 00	00, 00	CheckCond.2
10	CheckCond.1	0 <u>1</u> , 01	01, 00
<u>10</u>	CheckCond.2	0 <u>1</u> , 00	01, 0 <u>1</u>
11	0 <u>1</u> , 01	00, 01	CheckCond.1
<u>11</u>	01, 0 <u>1</u>	CheckCond.2	00, 0 <u>1</u>
1 <u>1</u>	01, 00	CheckCond.1	00, 00
<u>11</u>	0 <u>1</u> , 00	00, 00	CheckCond.2

Note: CheckCond.1 and CheckCond.2 implies further conditions as have been illustrated in Table 4.2 and Table 4.3

Table 4.2: CheckCond.1 for Table 4.1

Bits ( $X_{i+2}C_{i+2}$ )	Outputs: $Y_{i+1}Y_i, CT_{i+3}CT_{i+2}$
00 or 11 or 1 <u>1</u> or <u>11</u> or <u>11</u>	10, 00
01 or 10	<u>10</u> , 01
0 <u>1</u> or <u>10</u>	<u>10</u> , 01

Table 4.3: CheckCond.2 for Table 4.1

Bits ( $X_{i+2}C_{i+2}$ )	Outputs: $Y_{i+1}Y_i, CT_{i+3}CT_{i+2}$
00 or 11 or 1 <u>1</u> or <u>11</u> or <u>11</u>	<u>10</u> , 00
01 or 10	10, 0 <u>1</u>
0 <u>1</u> or <u>10</u>	10, 0 <u>1</u>

**Demonstration 4.3.1**

Given number: 7 (4 bit unsigned binary) = (0 1 1 1).

After padding 2 '0' bits at Left:  $X = (0 0 0 1 1 1)$ .

Initial auxiliary variable  $C = (0 0 0 0 0 0)$

STEP 1:  $Y = (0 0 0 0 1 0 \underline{1})$ ,  $CT = (0 0 0 1 0 0)$

STEP 2:  $Y = (0 0 0 1 0 0 \underline{1})$ ,  $CT = (0 0 0 0 0 0)$

STEP 3:  $Y = (0 0 0 1 0 0 \underline{1})$ ,  $CT = (0 0 0 0 0 0)$

The CMSD Output:  $(0 0 0 1 0 0 \underline{1}) = 7$ .

**Demonstration 4.3.2**

Given number: -9 (5 bit 2's complement binary) = (1 0 1 1 1).

Since Input is 2's Complement Binary, Padding 2 'MSB' bits at Left:  $X = (1 1 1 0 1 1 1)$ .

Initial auxiliary variable  $C = (0 0 0 0 0 0 0)$

STEP 1:  $Y = (0 0 \underline{1} 0 1 0 \underline{1})$ ,  $CT = (0 0 0 0 1 0 0)$

STEP 2:  $Y = (0 0 \underline{1} \underline{1} 0 0 \underline{1})$ ,  $CT = (0 0 1 0 0 0 0)$

STEP 3:  $Y = (0 0 0 \underline{1} 0 0 \underline{1})$ ,  $CT = (0 0 0 0 0 0 0)$

The CMSD Output:  $(0 0 0 \underline{1} 0 0 \underline{1}) = -9$ .

**4.3.2 Conversion of Binary to CMSD: One-step Algorithm**

In this section we present another conversion algorithm that works only for binary numbers. But the exciting feature is that it can perform the conversion of a binary number to its CMSD equivalent in just one step. Also the conversion process can begin from either side: left to right or right to left, thus implying the total parallelism. The algorithm is as shown in Algorithm 4.3.

**Algorithm 4.2** Search for a pattern like  $(10)^*11$ . used in Algorithm 4.3

*Procedure FindPattern(i)*

if there is a pattern like  $(10)^*11$  starting from bit position  $i$  then

Return True

else

Return False.

**End Procedure**

**Demonstration 4.3.3**

Given MSD number:  $-14$  (4 bit MSD) =  $(\underline{1} \ \underline{1} \ \underline{1} \ 0)$ .

After padding 2 '0' bits at Left:  $X = (0 \ 0 \ \underline{1} \ \underline{1} \ \underline{1} \ 0)$ .

Initial auxiliary variable  $C = (0 \ 0 \ 0 \ 0 \ 0 \ 0)$

STEP 1:  $Y = (0 \ 0 \ 0 \ 1 \ 1 \ 0)$ ,  $CT = (0 \ \underline{1} \ 0 \ \underline{1} \ 0 \ 0)$

STEP 2:  $Y = (0 \ \underline{1} \ 0 \ 0 \ 1 \ 0)$ ,  $CT = (0 \ 0 \ 0 \ 0 \ 0 \ 0)$

STEP 3:  $Y = (0 \ \underline{1} \ 0 \ 0 \ 1 \ 0)$ ,  $CT = (0 \ 0 \ 0 \ 0 \ 0 \ 0)$

The CMSD Output:  $(0 \ \underline{1} \ 0 \ 0 \ 1 \ 0) = -14$ .

**Algorithm 4.3 Binary to CMSD Conversion Algorithm: 1 Step****Procedure DoConversion**

Let  $X = X_{n-1}, \dots, X_1, X_0$  be a given unsigned binary or 2's complement binary number and  $n$  is even. Let  $Y = Y_{n+1}, \dots, Y_1, Y_0$  be the output CMSD number.

Pad 2 0's at right

if  $X$  is a 2's complement binary number then

Sign extend  $X$  by 1 bit by padding the MSB bit 1 times at left

else

Pad 2 0's at left

endif

for  $i = n$  downto 0

switch  $(X_i X_{i-1})$

case 00 or 11:  $Y_i = 0$

case 01:

switch  $(X_{i+1} X_{i-2})$

case 01:  $Y_i = 1$

case 11:  $Y_i = \underline{1}$

case 00:

if  $(FindPattern(i-1))$  then  $Y_i = 1$  else  $Y_i = 0$

case 10:

if  $(FindPattern(i-1))$  then  $Y_i = \underline{1}$  else  $Y_i = 0$

case 10:

switch  $(X_{i+1} X_{i-2})$

case 00:  $Y_i = 1$

case 10:  $Y_i = \underline{1}$

case 01:

if  $(FindPattern(i))$  then  $Y_i = 0$  else  $Y_i = 1$

case 11:

if  $(FindPattern(i))$  then  $Y_i = 0$  else  $Y_i = \underline{1}$

End Procedure

The complete algorithm consists of two procedures. The first procedure, given a bit position  $i$ , searches for a pattern like  $(10)^*11$ , i.e. patterns like 1011 or 101011 or 10101011 or so on, starting from the bit position  $i$ . It returns true if such pattern is found. The second procedure comprises the main conversion algorithm. Initially it pads 2 0 bits referred to as  $X_{-1}$  and  $X_{-2}$  at right of the binary expansion of the number to be converted.



Depending on the representation, it also pads the MSB bit or 2 '0' bits at left. Then for each bit position  $i$ , the algorithm checks  $X_i$  and  $X_{i-1}$  to determine the final digit  $Y_i$ , for  $i=0, \dots, n-1$ . If  $X_i X_{i-1} = 01$  or  $10$ , the algorithm checks the values of  $X_{i+1}$  and  $X_{i+1}$  also. The determination of the output digits are done in parallel for all bit-positions. Thus the full conversion is performed in just one step. The demonstration of the algorithm is as shown in Demonstration 4.3.4 and 4.3.5.

---

#### **Demonstration 4.3.4**

---

Given number: 43 (6 bit unsigned binary) = (1 0 1 0 1 1).

After padding 2 '0' bits at Right = (1 0 1 0 1 1 0 0).

Since Input is unsigned Binary pad 2 '0' bits at Left:  $X = (0 0 1 0 1 0 1 1 0 0)$

For bit position 6:  $X_6 X_5 = 01$  :  $X_7 X_4 = 00$  : FoundPattern(5) = True :  $Y_6 = 1$

For bit position 5:  $X_5 X_4 = 10$  :  $X_6 X_3 = 01$  : FoundPattern(5) = True :  $Y_5 = 0$

For bit position 4:  $X_4 X_3 = 01$  :  $X_5 X_2 = 10$  : FoundPattern(3) = True :  $Y_4 = \underline{1}$

For bit position 3:  $X_3 X_2 = 10$  :  $X_4 X_1 = 01$  : FoundPattern(3) = True :  $Y_3 = 0$

For bit position 2:  $X_2 X_1 = 01$  :  $X_3 X_0 = 11$  :  $Y_2 = \underline{1}$

For bit position 1:  $X_1 X_0 = 11$  :  $Y_1 = 0$

For bit position 0:  $X_0 X_{-1} = 10$  :  $X_1 X_{-2} = 10$  :  $Y_0 = \underline{1}$

The CMSD Output: (1 0 1 0 1 0 1) = 43.

---



---

#### **Demonstration 4.3.5**

---

Given number: -21 (6 bit 2's complement binary) = (1 0 1 0 1 1).

After padding 2 '0' bits at Right = (1 0 1 0 1 1 0 0).

Since Input is 2's complement Binary pad the MSB bit once at Left:  $X = (1 1 0 1 0 1 1 0 0)$

For bit position 5:  $X_5 X_4 = 10$  :  $X_6 X_3 = 11$  : FoundPattern(5) = True :  $Y_5 = 0$

For bit position 4:  $X_4 X_3 = 01$  :  $X_5 X_2 = 10$  : FoundPattern(3) = True :  $Y_4 = \underline{1}$

For bit position 3:  $X_3 X_2 = 10$  :  $X_4 X_1 = 01$  : FoundPattern(3) = True :  $Y_3 = 0$

For bit position 2:  $X_2 X_1 = 01$  :  $X_3 X_0 = 11$  :  $Y_2 = \underline{1}$

For bit position 1:  $X_1 X_0 = 11$  :  $Y_1 = 0$

For bit position 0:  $X_0 X_{-1} = 10$  :  $X_1 X_{-2} = 10$  :  $Y_0 = \underline{1}$

The CMSD Output: (0 1 0 1 0 1) = -21.

---

Although the algorithm can convert a binary representation to corresponding CMSD in one step, it has the disadvantage of checking for (10)\*11 pattern in the representation. This checking implies that for any bit position  $i$  of the binary number to be converted, the algorithm may require the values of bit positions  $i+1, \dots, i-n+1$  to make the conversion decision. Thus the symbolic substitution table has to contain entries for all bit positions in the check condition. In other words, if the bit-size of the binary number to be converted is  $n$ , then the symbolic substitution table has to contain column entries for bit positions  $X_{n+1}, \dots, X_0$ . However, not all the permutations of these bits are necessary for inclusion as conversion rules i.e. row entries in the symbolic substitution table. If the conditions

specified in the algorithm are co-opted in the symbolic substitution table, then for the conversion of an  $n$  bit binary number, we need a symbolic substitution table with  $(n + 1)$  column entries for check conditions, and  $4 * (n - 1)$  row entries for conversion rules. Thus for bit-size of 4, 6 and 8, we need only 12, 20 and 28 conversion rules with 5, 7 and 9 check conditions. The dependency of the symbolic substitution table on the bit size of the binary number to be converted may appear to be a disadvantage. Also the inability, to make the symbolic substitution table generalized to perform conversion of binary number of any size, may appear to be a flaw. However, at the cost of some extra entries and non-generalization, the advantage of performing the conversion in just one step may outweigh these disadvantages in many parallel processing situations.

An example of a symbolic substitution table, that can be used to convert a 4-bit binary number, is shown in table 4.4. The blank (-) entries in the table indicate don't care condition for that bit position.

Table 4.4: Symbolic substitution table for converting 4-bit binary number to CMSD representation in 1 step

$X_{i+1}$	$X_i$	$X_{i-1}$	$X_{i-2}$	$X_{i-3}$	$Y_i$
-	0	0	-	-	0
-	1	1	-	-	0
0	0	1	1	-	1
0	0	1	0	-	0
1	0	1	1	-	<u>1</u>
1	0	1	0	-	0
0	1	0	1	1	0
0	1	0	1	0	1
0	1	0	0	-	1
1	1	0	1	1	0
1	1	0	1	0	<u>1</u>
1	1	0	0	-	<u>1</u>

### 4.3.3 Addition of CMSD numbers

The next phase of the proposed arithmetic unit is the addition of two CMSD numbers. An one step substitution process has already been shown in the previous chapter. However the algorithm produces the result in MSD form. Thus to apply the resultant for further associative addition, we have to convert it again using either the Reitwiener's algorithm or the algorithm presented in section 4.3.1.

This section presents a novel symbolic substitution approach for CMSD addition that requires  $\lfloor \frac{n}{2} \rfloor + 1$  steps to complete. However this substitution process maintains the canonic property in the output. The algorithm and symbolic substitution table has been shown in Algorithm 4.4 and Table 4.5. The demonstration of the algorithm has been shown in Demonstration 4.3.6.

---

#### Algorithm 4.4 Algorithm for performing addition of CMSD numbers in $(\lfloor \frac{n}{2} \rfloor + 1)$ steps

---

```

Pad 2 '0' digits to the Left and 1 '0' digit at right of both  $n$ -digit operands ( $A$  and  $B$ )
for  $j = 1$  to  $\lfloor \frac{n}{2} \rfloor + 1$  do
  *(
  for  $i = 0$  to  $n$  STEP 2 do
    Set  $Y_{i+1}Y_i C_{i+3}C_{i+2}$  according to the symbolic substitution table on the basis of
    the values of  $A_{i+1}A_i$  and  $B_{i+1}B_i$ ;  $0 \leq i, (i+1), (i+2), (i+3) \leq n$ ; else  $Y_i = C_i = 0$ 
  )*
  Set  $A = Y$  and  $B = C$ 
 $Y = Y_n, Y_{n-1}, \dots, Y_0$  is the resultant

```

**NB:** The \*-marked code Runs in parallel

---

At each substitution step, the proposed algorithm considers 3 bits from each of the input operands,  $A_{i+1}A_iA_{i-1}, B_{i+1}B_iB_{i-1}$  as input patterns and using Table 4.5, derives as output patterns the two variables  $Y$  and  $C$  (by determining the bits,  $Y_{i+1}Y_i$  and  $C_{i+3}C_{i+2}$ ,  $\forall i = 0, \dots, n-1$ , where  $n$  is the bit-size of the operands) in parallel. In some cases, the substitution process requires to check the bits,  $A_{i+2}$  and  $B_{i+2}$  as well and Table 4.6 and Table 4.7 are used according to the corresponding entry in Table 4.5. After a step has been completed, the values of  $O$  and  $C$  are copied back as  $A$  and  $B$  for next substitution step. This substitution process is iterated  $\lfloor \frac{n}{2} \rfloor + 1$  times, at the end of which  $Y = Y_n, Y_{n-1}, \dots, Y_0$  holds the result of addition.

Table 4.5: Symbolic substitution table for performing addition of CMSD numbers in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps and preserving CMSD property

$A_{i+1}A_i$	$B_{i+1}B_i$	Check: $A_{i-1}B_{i-1}$ Outputs( $Y_{i+1}Y_i, C_{i+3}C_{i+2}$ )		
		11	<u>11</u>	otherwise
00	00	01, 00	0 <u>1</u> , 00	00, 00
	01	-	-	01, 00
	0 <u>1</u>	-	-	0 <u>1</u> , 00
	10	0 <u>1</u> , 01	01, 00	CheckCond.1
	<u>10</u>	0 <u>1</u> , 00	01, 0 <u>1</u>	CheckCond.2
01	00	-	-	01, 00
	01	-	-	CheckCond.1
	0 <u>1</u>	-	-	00, 00
	10	-	-	0 <u>1</u> , 01
	<u>10</u>	-	-	0 <u>1</u> , 00
0 <u>1</u>	00	-	-	0 <u>1</u> , 00
	01	-	-	00, 00
	0 <u>1</u>	-	-	CheckCond.2
	10	-	-	01, 00
	<u>10</u>	-	-	01, 0 <u>1</u>
10	00	0 <u>1</u> , 01	01, 00	CheckCond.1
	01	-	-	0 <u>1</u> , 01
	0 <u>1</u>	-	-	01, 00
	10	01, 00	0 <u>1</u> , 00	00, 00
	<u>10</u>	01, 00	0 <u>1</u> , 00	00, 00
<u>10</u>	00	0 <u>1</u> , 00	01, 0 <u>1</u>	CheckCond.2
	01	-	-	0 <u>1</u> , 00
	0 <u>1</u>	-	-	01, 0 <u>1</u>
	10	01, 00	0 <u>1</u> , 00	00, 00
	<u>10</u>	01, 00	0 <u>1</u> , 00	00, 00

Note: CheckCond.1 and CheckCond.2 implies further conditions as have been illustrated in Table 4.6 and Table 4.7

Table 4.6: CheckCond.1 for Table 4.5

$A_{i+2}$	$B_{i+2}$	Outputs( $Y_{i+1}Y_i, C_{i+3}C_{i+2}$ )
0	0	10, 00
1	<u>1</u>	
1	1	
<u>1</u>	1	
<u>1</u>	<u>1</u>	
otherwise	otherwise	<u>10, 01</u>

Table 4.7: CheckCond.2 for Table 4.5

$A_{i+2}$	$B_{i+2}$	Outputs( $Y_{i+1}Y_i, C_{i+3}C_{i+2}$ )
0	0	<u>10, 00</u>
1	<u>1</u>	
1	1	
<u>1</u>	1	
<u>1</u>	<u>1</u>	
otherwise	otherwise	10, <u>01</u>

#### 4.3.4 Addition of CMSD numbers in 1 step

Section 3.8.3 provides a symbolic substitution process for addition of two CMSD numbers in 1 step. In this section we provide a similar algorithm. The previous algorithm checked two input bit positions ( $A_iB_i, A_{i-1}B_{i-1}$ ) to determine one output bit( $S_i$ ). In contrast the new algorithm presented here checks three input bit positions ( $A_{i+1}B_{i+1}, A_iB_i, A_{i-1}B_{i-1}$ ) and determine two output bits( $S_{i+1}, S_i$ ). Thus, although in terms of required steps the algorithms have no difference at all (both requires 1 substitution step), the presented algorithm here, however, requires lesser number of comparison devices (pattern recognizers). While the previous algorithm requires  $n - 1$  comparisons for a  $n$ -bit number, the proposed algorithm requires  $n/2$  comparison and thus  $n/2$  pattern recognizers only. The algorithm and symbolic substitution table has been shown in Algorithm 4.5 and Table 4.8. The demonstration of the algorithm has been shown in Demonstration 4.3.9.

**Demonstration 4.3.6** Addition of CMSD numbers in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps

$A = 0 \underline{1} 0 1 = -3$

$B = 1 0 \underline{1} 0 = 6$

Padding two 0 at left and one zero at right of both A & B:

A: 0 0 0 1 0 1 0

B: 0 0 1 0 1 0 0

.....  
Y: 0 0 0 1 0 1 0 [STEP 1]

C: 0 0 0 0 0 0 0

=====  
Y: 0 0 0 1 0 1 0 [STEP 2]

C: 0 0 0 0 0 0 0

=====  
Y: 0 0 0 1 0 1 0 [STEP 3]

C: 0 0 0 0 0 0 0

=====  
0 0 1 0 1 = 3 [Final output]

**Algorithm 4.5** CMSD addition in 1 step

Pad 2 '0' digits to the Left and 1 '0' digit at right of both  $n$ -digit operands ( $A$  and  $B$ )

\*(

for  $i = 0$  to  $n$  STEP 2 do

Set  $S_{i+1}S_i$  according to the symbolic substitution table on the basis of the values of  $A_{i+1}A_i$  and  $B_{i+1}B_i$ ;

)\*

$S = S_n, S_{n-1}, \dots, S_0$  is the resultant

NB: The \*-marked code Runs in parallel

Table 4.8: Symbolic substitution table for performing addition of CMSD numbers in one step

$A_{i+1}A_i$	$B_{i+1}B_i$	Check: $A_{i-1}B_{i-1}$		
		Outputs( $S_{i+1}S_i$ )		
		<b>11</b>	<b><u>11</u></b>	otherwise
00	00	01	<u>01</u>	00
	01	-	-	01
	<u>01</u>	-	-	<u>01</u>
	10	11	<u>11</u>	10
	<u>10</u>	<u>11</u>	<u>11</u>	<u>10</u>

Contd. On next page ...

Table 4.8: Continued...

... Contd. From Previous page				
$A_{i+1}A_i$	$B_{i+1}B_i$	Check: $A_{i-1}B_{i-1}$		
		Outputs( $S_{i+1}S_i$ )		
		11	<u>11</u>	otherwise
01	00	-	-	01
	01	-	-	10
	0 <u>1</u>	-	-	00
	10	-	-	11
	<u>1</u> 0	-	-	<u>11</u>
0 <u>1</u>	00	-	-	0 <u>1</u>
	01	-	-	00
	0 <u>1</u>	-	-	<u>10</u>
	10	-	-	<u>11</u>
	<u>1</u> 0	-	-	<u>11</u>
10	00	11	<u>11</u>	10
	01	-	-	11
	0 <u>1</u>	-	-	<u>11</u>
	10	01	0 <u>1</u>	00
	<u>1</u> 0	01	0 <u>1</u>	00
<u>1</u> 0	00	<u>11</u>	<u>11</u>	<u>10</u>
	01	-	-	<u>11</u>
	0 <u>1</u>	-	-	<u>11</u>
	10	01	0 <u>1</u>	00
	<u>1</u> 0	01	0 <u>1</u>	00

---

**Demonstration 4.3.7** Addition of CMSD numbers in one step

---

$1010 = A = 10$

$1010 = B = 10$

-----  
 $0010100$  (Pad 2 0's at left and 1 '0' at right)

$0010100$

-----  
 $010100 = \text{Using table the final value} = 20$

---

### 4.3.5 Conversion of MSD to Binary

The last phase of the proposed unit is the conversion of the non-binary representation to binary. This section considers the conversion of any MSD number to its binary presentation. The next section will reflect upon the conversion of CMSD number to binary notation. The algorithm and symbolic substitution table for the MSD to binary conversion have been shown in Algorithm 4.6 and Table 4.9. The corresponding symbolic substitution process requires  $n$  substitution steps, where  $n$  is the bit-length of the number to be converted. The demonstration of the algorithm has been shown in Demonstration 4.3.8.

---

**Algorithm 4.6** Algorithm for converting MSD number to 2's complement binary number in  $n$  steps

---

Let  $X = X_{n-1}, \dots, X_1, X_0$  be the given MSD number;  $C = 0$ .

**for**  $j = 1$  to  $n$  **do**

  \*(

**for**  $i = 0$  to  $n - 1$  **do**

      Substitute  $X_i$  by  $Y_i$  according to the symbolic substitution table and also determine

$C'_{i+1}$

  )\*

  Set  $X = Y$  and  $C = C'$

---

**NB:** The \*-marked code Runs in parallel

---

Table 4.9: Symbolic substitution table for converting MSD number to 2's complement binary number in  $n$  steps

$X_i$	$C_i$	$Y_i$	$C'_{i+1}$
0	0	0	0
0	<u>1</u>	1	<u>1</u>
1	0	1	0
1	<u>1</u>	0	0
<u>1</u>	0	1	<u>1</u>
<u>1</u>	<u>1</u>	0	<u>1</u>



**Demonstration 4.3.8** Conversion of MSD number to 2's complement binary $0 \underline{1} 0 0 = -4$ , Given Input $0 \underline{1} 0 0 = X$  $0 0 0 0 = C$  $0 1 0 0$  $\underline{1} 0 0 0$  $1 1 0 0$  $0 0 0 0$  $1 1 0 0$  $0 0 0 0$  $1 1 0 0$  $0 0 0 0$  $1 1 0 0 = \text{Final Conversion}$ **4.3.6 Conversion of CMSD to Binary**

This section presents symbolic substitution table (Table 4.10) and algorithm (Algorithm 4.7) that can be used for the conversion of any CMSD number to its Binary representation (2's complement) in  $n/2+1$  steps. The demonstration of the algorithm has been shown in Demonstration 4.3.9.

**Algorithm 4.7** Algorithm to convert CMSD to binary in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps

Pad 2 '0' digits to the Left

for  $j = 1$  to  $n/2 + 1$  STEP 2 do

\*(

for  $i = 0$  to  $n$  STEP 2 doSubstitute  $X_{i+1}X_i$  by  $Y_{i+1}Y_i$  according to the symbolic substitution table and alsodetermine  $CT_{i+3}CT_{i+2}$ ;  $0 \leq i, (i+1), (i+2), (i+3) \leq n$ ; else  $Y_i = CT_i = 0$ 

)\*

Set  $X = Y$  and  $C = CT$  $Y = Y_n, \dots, Y_0$  is the resultant

Table 4.10: Symbolic substitution table to convert CMSD to binary in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps

Bits $(X_{i+1}X_i)$	Check: $C_{i+1}C_i$	
	Outputs $(Y_{i+1}Y_i, CT_{i+3}CT_{i+2})$	
	00	0 <u>1</u>
00	00, 00	11, 0 <u>1</u>
01	01, 00	00, 00
0 <u>1</u>	11, 0 <u>1</u>	Not Applicable
10	10, 00	01, 00
<u>1</u> 0	10, 0 <u>1</u>	10, 0 <u>1</u>
11	11, 00	10, 00

---

**Demonstration 4.3.9** Conversion of CMSD number to binary

---

$1\ 0\ \underline{1}\ 0 = 6$  (Given Number)

-----  
 $0\ 0\ 1\ 0\ \underline{1}\ 0 = X$  (Pad 2 0's at Left)  
 $0\ 0\ 0\ 0\ 0\ 0 = C$

-----  
 $0\ 0\ 1\ 0\ 1\ 0$  [Step 1]  
 $0\ 0\ 0\ \underline{1}\ 0\ 0$

-----  
 $0\ 0\ 0\ 1\ 1\ 0$  [Step 2]  
 $0\ 0\ 0\ 0\ 0\ 0$

-----  
 $0\ 0\ 0\ 1\ 1\ 0$  [Step 3]  
 $0\ 0\ 0\ 0\ 0\ 0$

-----  
 $0\ 0\ 0\ 1\ 1\ 0 = \text{Final Binary Representation}$

---

#### 4.4 Comparison of the Proposed Schemes with Earlier Ones

In this chapter, we have presented a set of new tables and algorithms for different arithmetic computation involved in the proposed unit. To see how the new processes lead to computational efficiency, we may first calculate the steps required using the earlier schemes as follows:

Function	Steps Required ( $n = \text{No. of bits in Input}$ )
Conversion of Binary To CMSD	$n+1$
Addition of CMSD	1
Conversion of CMSD To Binary	$n$
Total	$2n+2$

On the other hand, if the adder is implemented using the substitution tables and algorithms presented in this chapter, the steps required are as follows:

Function	Steps Required ( $n = \text{No. of bits in Input}$ )
Conversion of Binary To CMSD	1 (using Algorithm 4.3) or $\lfloor \frac{n}{2} \rfloor + 1$ (using Algorithm 4.1)
Addition of CMSD	1 (using Algorithm 4.5: resultant in MSD notation) or $\lfloor \frac{n}{2} \rfloor + 1$ (using Algorithm 4.4: resultant in CMSD notation)
Conversion of MSD/CMSD To Binary	$n$ (MSD to binary) or $\lfloor \frac{n}{2} \rfloor + 1$ (CMSD to binary)
Total	$n + 2$ or $\lfloor \frac{3n}{2} \rfloor + 2$ $n + 3$ or $\lfloor \frac{3n}{2} \rfloor + 3$

As can be seen, the required number of steps are much less than a system based on earlier schemes. The next chapter will reflect on how these new symbolic substitution processes may be improved further in terms of required substitution steps.

# Chapter 5

## STEP REDUCTION AND EXPERIMENTAL ANALYSIS

### 5.1 Reduction of Symbolic Substitution Steps

Symbolic substitution tables for different conversion and addition processes have already been discussed in the previous chapters. However, it may be noted that processing based on these earlier and new tables often perform extra calculations than necessary.

For example, let us consider a demonstration of conversion of a MSD number '1 0 1 1 0' (10) to binary equivalent using the Reitwiesner's algorithm as presented in the next page. The demonstration has shown only the positions where some kind of substitution is made. The blank spaces at the right indicate that the corresponding  $X$  bits have reached their final values since the corresponding  $C$  bits are 0. It can be seen that, after 2 steps,  $C$  becomes 0 and there is no need for any more substitution steps. But an arithmetic unit implementing the table will perform an extra step.

A similar example, using the proposed algorithm 4.1 for MSD to CMSD conversion (conversion of 20) has also been presented in the second example of the next page. In this case also, the processor performs 2 extra steps even after the value of  $C$  has become 0.

1 0 <u>1</u> <u>1</u> 0 = 10 , Given Input
1 0 <u>1</u> <u>1</u> 0 = X
0 0 0 0 0 = C
1 0 1 1 0
0 <u>1</u> <u>1</u> 0
1 1 0 1
<u>1</u> 0 0
0 1 0
0 0
0 1
0 1 0 1 0 = <i>Final Conversion</i>

20 = Given Input
1 0 <u>1</u> <u>1</u> 0 0 = 6 Bit MSD representation
0 0 1 0 <u>1</u> <u>1</u> 0 0 = Pad 2 0's at Left
0 0 0 0 0 0 0 0 = Initial C's
0 0 1 0 0 1 0 0
0 0 0 <u>1</u> 0 0
0 0 0 1 0 1
0 0 0 0
0 0 0 1
0 0
0 0
0 0 0 1 0 1 0 0 = <i>Final CMSD Representation</i>

Similar situations can be found in other processes where the substitution involves two numbers and in all these cases once the second number (C) becomes 0, no effective substitutions are made and the rest of the steps are simply unnecessary. Thus there is a scope for improvement if the substitution process regards this phenomenon and stop processing whenever the second number (C) becomes 0. For subsequent discussion, we will use the term 'auxiliary variable' to denote this second number.

As a part of this study, a set of some computer programs (provided in the source code listing), written in C++, were used to analyze the computational efficiency, in terms of the required number of steps, of units based on the proposed symbolic substitution algorithms, as presented in the previous chapter. As has already been pointed out, these algorithms perform the computation faster than any earlier scheme. However, the analysis presented in this chapter will show that these can be further optimized by stopping substitution once the auxiliary variable reaches the value of zero.

## 5.2 Conversion of MSD and Binary to CMSD

Normally a processor implementing the substitution table 4.1 will require  $\lfloor \frac{n}{2} \rfloor + 1$  steps. However if the process is stopped after the auxiliary variable, C becomes 0, then an improvement is expected. To demonstrate the optimization, a computer program to simulate the conversion process was developed. For a given bit-size ( $n$ ), the program could generate all the possible combinations of MSD numbers (in total:  $3^n$  numbers) for that particular bit-size. For each of these MSD numbers, the number of parallel steps that are necessary to convert the number to CMSD representation was recorded. From these data, the cumulative percentage of the total possible combinations that are completely converted to CMSD representation within a particular number of steps was calculated. The result of the analysis is as shown in Table 5.1 and the graph of Figure 5.1. It can be seen that for some fixed bit-size most of the MSD numbers (more than 90%) are converted to corresponding CMSD representation in steps much less than the specified maximum of  $(\lfloor \frac{n}{2} \rfloor + 1)$  parallel processing steps. As illustrated in the graph of Figure 5.1, even with a large fixed digit-size, most of the numbers (more than 97%) are converted within 5 steps. Hence for any MSD or binary number generated at random, the proposed algorithm with optimization will show superiority over other algorithms.

Table 5.1: Result of analysis for MSD to CMSD conversion

No. of Bits ( $n$ )	Total Possible MSD ( $3^n$ )	Result of Analysis		
		Steps Required for Conversion	Number of MSD requiring this Step	% of total MSD
4	81	1	41	50.62
		2	38	46.91
		3	2	2.47
6	729	1	231	31.69
		2	306	41.98
		3	182	24.97
		4	10	1.37
8	6561	1	1289	19.65
		2	2418	36.85
		3	2094	31.92

Contd. on next page ...

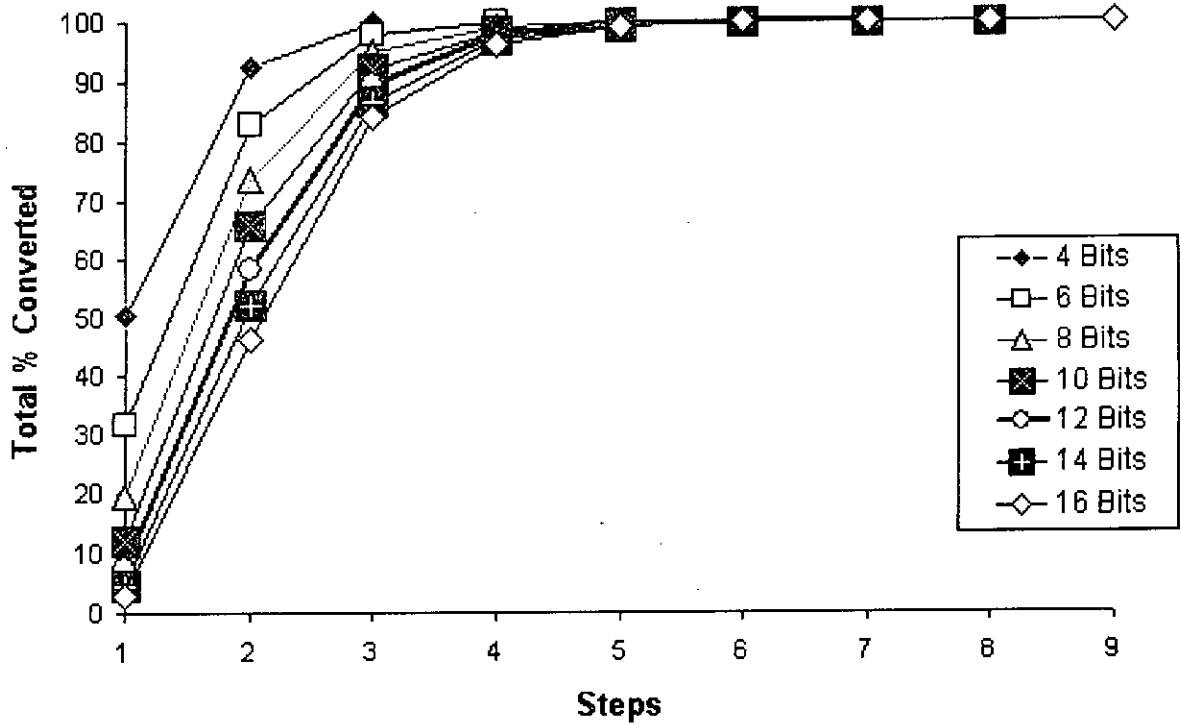
Table 5.1: Continued...

... Contd. from previous page				
No. of bits ( $n$ )	Total possible MSD ( $3^n$ )	Result of Analysis		
		Steps required for conversion	Number of MSD requiring this step	% of total MSD
		4	588	8.96
		5	172	2.62
10	59049	1	7175	12.15
		2	18338	31.06
		3	21736	36.81
		4	8426	14.27
		5	2694	4.56
		6	680	1.15
12	531441	1	39913	7.51
		2	135810	25.56
		3	207316	39.01
		4	77512	14.59
		5	34540	6.50
		6	28340	5.33
		7	8010	1.51

### 5.3 Addition of CMSD numbers

A similar experiment was performed with the CMSD addition algorithm (Algorithm 4.4) presented in the previous chapter. In this case, for a given bit-size ( $n$ ), all possible combinations of CMSD numbers (in total:  $(2^{n+2} + (-1)^{n \bmod 2+1}/3)^2$  CMSD numbers) were generated and required substitution steps to perform their addition was recorded. The result of the analysis is as shown in Table 5.2 and the graph of Figure 5.2. It can be seen that most of the additions are finished in steps much less than the specified maximum of  $(\lfloor \frac{n}{2} \rfloor + 1)$  parallel processing steps. As illustrated in the graph of Figure 5.2, even with a large fixed digit-size, most of the combinations of CMSD numbers (more than

Figure 5.1 Graph of total of the percentage converted vs. required no. of steps for MSD to CMSD conversion



97%) reaches their addition result within 4 steps. Hence for CMSD numbers generated at random, the proposed algorithm with optimization will show superiority over other corresponding algorithms.

Table 5.2: Result of analysis for CMSD addition

No. of bits	Total possible combinations	Result of Analysis		
		Steps required	Number of combinations	% of total
4	441	1	293	66.44
		2	128	29.02
		3	20	4.54
6	7225	1	3685	51.00
		2	2808	38.87
		3	648	8.97
		4	84	1.16
		1	44901	38.61
		2	53288	45.83

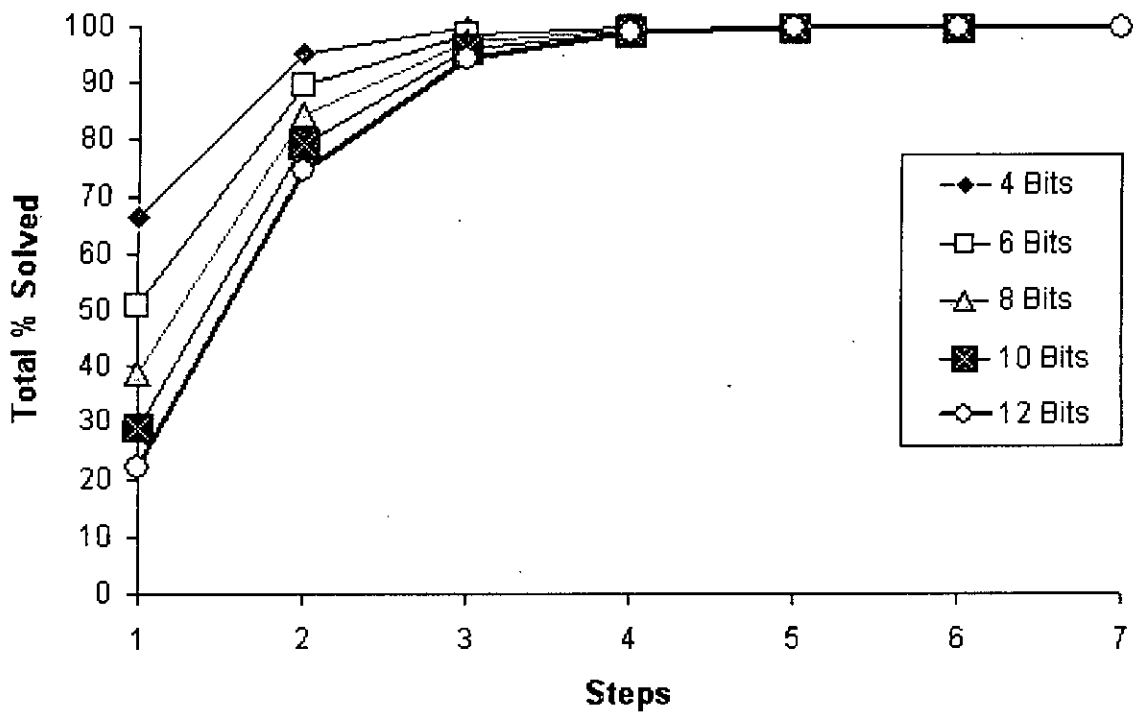
Contd. On next page ...



Table 5.2: Continued...

... Contd. From Previous page				
No. of bits	Total possible combinations	Result of Analysis		
		Steps required	Number of combinations	% of total
8	116281	3	15072	12.96
		4	2680	2.30
		5	340	0.29
10	1863225	1	541541	29.06
		2	935464	50.21
		3	311368	16.71
		4	62680	3.36
		5	10808	0.58
		6	1364	0.07

Figure 5.2 Graph of total of the percentage added vs. required no. of steps for CMSD addition



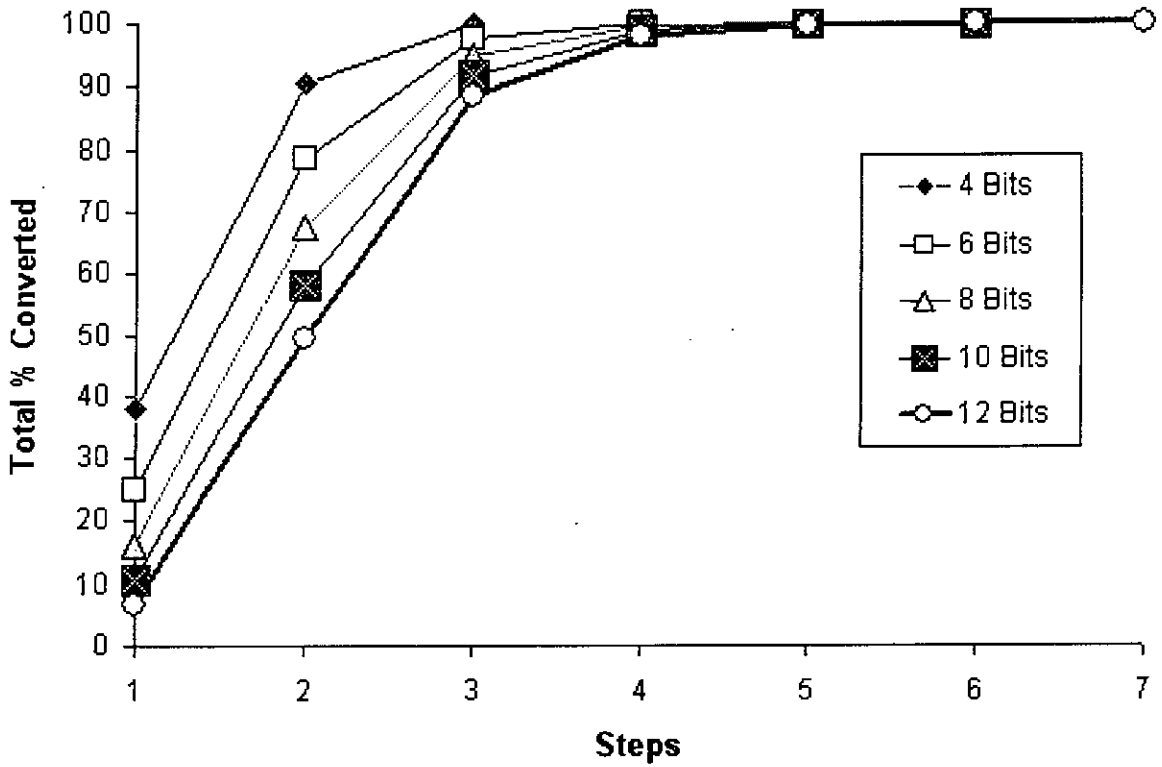
## 5.4 Conversion of CMSD to Binary

Another experiment was performed for the CMSD to binary conversion algorithm (Algorithm 4.7). In this case, all possible combinations of CMSD numbers for a fixed bit-size were generated and the required substitution steps to perform their conversion was recorded. The result of the analysis is as shown in Table 5.3 and the graph of Figure 5.3. As illustrated in the graph of Figure 5.1, even with a large fixed digit-size, most of the numbers (more than 97%) are converted within 4 steps.

Table 5.3: Result of analysis for CMSD to binary conversion

No. of bits	Total possible CMSD	Result of Analysis		
		Steps required	Number of CMSD	% of total
4	441	1	293	66.44
		2	128	29.02
		3	20	4.54
6	7225	1	3685	51.00
		2	2808	38.87
		3	648	8.97
		4	84	1.16
8	116281	1	44901	38.61
		2	53288	45.83
		3	15072	12.96
		4	2680	2.30
		5	340	0.29
10	1863225	1	541541	29.06
		2	935464	50.21
		3	311368	16.71
		4	62680	3.36
		5	10808	0.58
		6	1364	0.07

Figure 5.3 Graph of total of the percentage converted vs. required no. of steps for CMSD to binary conversion



### 5.5 Concluding Remarks

Thus we see that introducing the optimization results in faster processing for random samples. So if the proposed addition unit incorporates the optimization, the whole system will process addition in much less number of substitution steps than any other schemes.

# Chapter 6

## SYMBOLIC SUBSTITUTION BASED ASSOCIATIVE ADDITION

### 6.1 Introduction to the Topic

All the earlier proposed symbolic substitution tables and algorithms have focussed mainly on the addition of two numbers. In this chapter, however, we take a step further by incorporating a symbolic substitution based process for associative addition of numbers. The goal is to develop a symbolic substitution based method for adding a set of numbers such that the required substitution steps is much less than that required by a symbolic substitution scheme for addition of two numbers, being applied iteratively.

### 6.2 Associative Addition of Binary Numbers

The study, here, focusses on the associative addition of binary numbers only. But similar algorithm can also be developed for other number systems. The process for associative addition of binary numbers has been described in Algorithm 6.1.

The idea of the algorithm is to gradually reduce the number of operands by considering the total summation at each bit position. If  $m$   $n$ -bit numbers are added, the resultant will be of maximum  $n + \lceil \lg(m) \rceil$  bits. The target of the algorithm is to determine gradually the bit value of the total summations at each of the bit position. If there are  $m$  numbers, then the addition of all corresponding bits of  $m$  numbers at a particular position can be represented using a  $r$  bit number, where  $r = \lceil \lg(m) \rceil + 1$ .

**Algorithm 6.1** Associative addition algorithm

**Let:**  $Maxcol = n + \lceil \lg(m) \rceil$ ; the maximum number of bits possible in the total summation of the  $m$   $n$ -bit binary numbers

1. **Let:**  $X_0, X_1, \dots, X_{m-1}$ , denote  $m$   $n$ -bit binary numbers to be added, where each  $X = x_{n-1}, x_{n-2}, \dots, x_0$ ;

**Let:**  $r = \lceil \lg(m) \rceil + 1$

**Let:**  $X_{j,i}$  denote the bit at  $i$ -th position of the  $j$ -th binary number, for  $\forall j = 0, \dots, m-1$  and  $\forall i = 0, \dots, n-1$ ; Pad  $\lceil \lg(m) \rceil$  '0' at the left of all  $X_j$ .

**Let:**  $S_i = \sum_{j=0}^{m-1} X_{j,i} = \sum_{k=0}^{r-1} 2^k S_{i,k}$  where  $S_{i,k}$  is the bit at the  $k$ -th position of the binary representation of  $S_i$  with bit-length  $r$ ;

2. Deploy in the SS system  $r$  planes & Let  $P_{l,k}$  denote the  $k$ -th bit of the  $l$ -th plane, for  $l = 0, \dots, r-1$  and  $k = 0, \dots, Maxcol-1$ ; Set  $P_{l,k} = 0, \forall l, k$ .

3.

for  $count = 0$  to  $Maxcol - 1$  do

  for  $l = 0$  to  $r - 1$  do

**Set:**  $P_{l,l+count} = S_{count,l}$ ; for  $0 \leq l + count \leq Maxcol - 1$

4. **Set:**  $m = r$ ; Repeat the steps 2-4 until  $m \leq 2$

The algorithm applies this principle for all the bit positions involved and thus reduce the  $m$  numbers to  $r$  numbers at each step. The bit values of the addition result at each  $n + \lceil \lg(m) \rceil$  bit position are coordinated in such way that the total addition of the  $m$  numbers equal the total addition of  $r$  numbers. The process, in the next step, works similarly with the  $r$  operands and the reduction process continues iteratively until the number of operands is 2. Then any symbolic substitution based binary addition unit, like the proposed one in previous chapters, can be used to determine the addition.

A demonstration of the algorithm adding a set of binary numbers has been shown in Table 6.1

Table 6.1: Demonstration of associative addition

Step	Given Numbers	Comment
0	0 0 0 1 0 1 1 0 1 = 45	$n = 6$ $m = 8$ Pad $\lceil \lg(m) \rceil = 3$ '0' at left
	0 0 0 0 1 1 0 1 1 = 27	
	0 0 0 0 0 0 1 0 1 = 5	
	0 0 0 1 0 1 0 1 1 = 43	
	0 0 0 0 1 1 0 1 1 = 27	
	0 0 0 1 0 1 1 0 1 = 45	
	0 0 0 1 0 1 1 1 1 = 47	
Contd. On next page ...		

Table 6.1: (Contd.)

Step	Given Numbers	Comment
	0 0 0 1 1 1 1 1 1 = 63	
1	0 0 0 1 1 1 1 1 0 = 62 0 0 0 1 1 0 0 0 0 = 48 0 1 0 1 1 1 0 0 0 = 184 0 0 0 0 0 1 0 0 0 = 8	Reduction
2	0 1 0 1 1 1 1 1 0 = 190 0 0 1 1 1 0 0 0 0 = 112 0 0 0 0 0 0 0 0 0 = 0	Reduction
3	0 1 1 0 0 1 1 1 0 = 206 0 0 1 1 0 0 0 0 0 = 96	Reduction: last step

### 6.3 Block Level Architecture of the Unit

Figure 6.1 depicts a block architecture of the addition unit. The unit comprises a set of symbolic substitution units. Each of these units accepts  $m$  numbers at input and reduce them to  $r = \lfloor \lg(m) \rfloor + 1$  numbers at the output using the symbolic substitution logic of Algorithm 6.1. The output of a unit is passed to the subsequent unit and this continues until the last substitution unit that accepts two binary numbers as input. These two numbers are then processed using the symbolic substitution logic for binary addition.

### 6.4 Required Steps to Complete Addition

To calculate the required number of symbolic substitution steps, we define a function  $F(m)$  as follows.

Let  $2^{q-1} \leq m < 2^q$ . Then  $F(m) = 1 + F(q)$ , when  $m > 2$  and  $F(m) = 0$ , when  $m \leq 2$ . The required number of symbolic substitution step to complete associative addition of  $m$  binary numbers is then:  $S(m, n) = F(m) + G(n + \lfloor \lg(m) \rfloor)$ , where  $G(x)$  denotes the number of symbolic substitution steps required for the addition of two  $x$ -bit binary numbers. Since the reduction process results in two binary numbers of  $n + \lfloor \lg(m) \rfloor$  bits,

Figure 6.1 Symbolic substitution architecture for associative addition

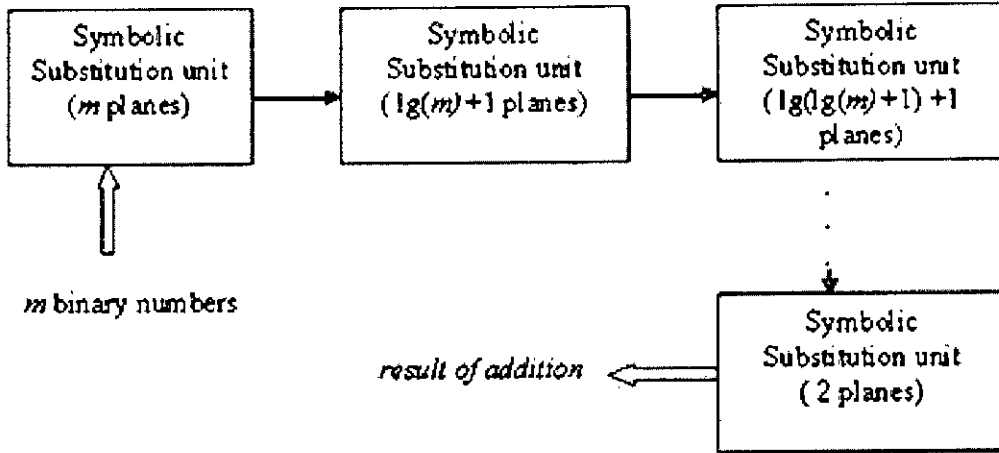


Table 6.2: Values of  $m$  and  $F(m)$

$m$	$F(m)$
2	0
3	1
4	2
7	2
8	3
16	3
127	3
128	4
$2^{127}$	5

$G(n + \lceil \lg(m) \rceil)$  denotes the steps required in the binary addition symbolic substitution unit.  $F(m)$  is in-effect a poly-logarithmic function and therefore very slow growing. Table 6.2 illustrates the values of  $F(m)$  for some value of  $m$ . As can be seen, even with a very large  $m$ ,  $F(m)$  has a very small value. In other words, only for impractically large value of  $2^{127}$ , we have  $F(m) = 5$ . Thus for practical purposes we may consider  $F(m) = 4$ . Hence, we require only a few substitution steps for the completion of associative addition.

## 6.5 Concluding Remarks

Obviously the algorithm presented here is computationally more efficient than any non-associative scheme for associative addition. A non-associative scheme would require pairwise addition for all the given set of numbers and thus will require many more steps than the presented algorithm. The algorithm is a very suitable choice for application in arithmetic computation, like multiplication, where associative addition of a set of numbers is an integral process.



# Chapter 7

## CONCLUSION AND RECOMMENDATIONS

### 7.1 Concluding Words

The thesis has performed a detailed study on the applicability of symbolic substitution technique for arithmetic computation involving binary numbers. The earlier algorithms focussed on employing non-binary systems like MSD or CMSD in symbolic substitution process. Binary system has been regarded as unsuitable for this parallel computing due to the inherent carry. This study, however, shows how a symbolic substitution based unit could be derived that works with binary numbers at its interface and use a non-binary system to perform the computation. The focus has been using the symbolic substitution process for all the intermediate phases of conversion and computation involved. The study has also sought to improve the existing algorithms and tables in the context. As has already been shown, if such an unit is designed based on existing algorithms, the unit would require  $2n + 2$  symbolic substitution steps, while the proposed unit will require (depending upon the algorithm used) only  $n + 2$  or  $\lfloor \frac{3n}{2} \rfloor + 2$  or  $n + 3$  or  $\lfloor \frac{3n}{2} \rfloor + 3$  substitution steps (as illustrated in section 4.4). The required number of steps are, therefore, much less than a system based on earlier schemes.

The thesis also makes a study on how these tables and algorithms could be further improved by introducing some efficiency in the substitution process. As has been illustrated, for the different algorithms, in chapter 5, most of the computation can be completed within much less the specified maximum steps if the substitution is stopped as soon as the second input variable involved reaches the value of 0. So a symbolic substitution unit based on this concept will give high computational efficiency, in terms of substitution steps.

A pioneer role of the thesis is the introduction of symbolic substitution based associative addition technique that may lead way to the application of this process for other arithmetic computation like multiplication. Rather than employing a non-associative addition algorithm repeatedly to perform associative addition of binary numbers, the application of the presented algorithm will require much less substitution steps.

To summarize, the main contributions presented in this research are as follows:

- A symbolic substitution table and corresponding algorithm to convert any MSD and binary number to CMSD number in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps. The earlier algorithm in this context required  $n + 1$  steps. Thus the presented algorithm has made roughly a 50% improvement in terms of substitution steps.
- A symbolic substitution table and corresponding algorithm to convert any binary number to CMSD number in one step. This algorithm can handle binary numbers in both signed 2's complement form and unsigned form. The one-step substitution implies that the algorithm reaches the fastest rate, in terms of substitution steps, as possible.
- A symbolic substitution table and corresponding algorithm to perform addition of CMSD number in one step. There was a corresponding one step algorithm for CMSD addition. However, the proposed algorithm requires less comparisons of bits and hence less number of pattern recognizers than the earlier one.
- A symbolic substitution table and corresponding algorithm to convert any MSD number to 2's complement binary number in  $n$  steps.
- A symbolic substitution table and corresponding algorithm for performing addition of CMSD number in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps but preserving CMSD property. The earlier one step algorithm, though using one step, produced the result in CMSD notation. Thus to use the algorithm for subsequent associative addition, we require the resultant to be converted to CMSD notation, using one of the previous algorithm, before applying the addition algorithm again. The presented algorithm, however, derives the result in CMSD notation and therefore requires less substitution steps than the previous algorithms applied repeatedly.
- A symbolic substitution table and corresponding algorithm to convert any CMSD number to 2's complement binary number in  $(\lfloor \frac{n}{2} \rfloor + 1)$  steps. This is a new development in the context.
- An analysis of how the substitution process can be improved by stopping the substitution process as soon as the second input variable becomes all zero.

- A symbolic substitution based approach for the associative addition of binary numbers. The approach is extremely suitable for application in the context of multiplication, where associative addition is an integral process.

## 7.2 Recommendations for Future Work

The study has concentrated on the improvement of existing symbolic substitution steps and development of symbolic substitution based approach requiring fewer substitution steps. There are, however, further scope for improvements, as listed below.

- This thesis has regarded two bits of the operands as a single unit for the symbolic substitution tables and algorithms presented. This has led to the improvement of the earlier algorithms in terms of reduction of substitution steps. Further reduction might be possible if more bits are considered as unit in the symbolic substitution process. However, considering more bits in the symbolic substitution table will increase table size and will therefore require more storage. Hence an area of future research might be, at the cost of some extra storage, use more bits in the symbolic substitution process so as to reduce the required number of steps. Efficient algorithms could be designed that will keep the table-size feasible and yet use the flexibility of using more than two bits as unit for the substitution.
- The tables and algorithms presented in the study has focussed on the use of MSD or CMSD number system as the non-intermediate number representation. There are, however, other possibilities like the signed-digit number system with higher radix or non-weighted number systems like Residue Number System, as discussed in chapter 2. Also the development of a binary like number system as a replacement of the non-binary intermediate representation, may lead the way to use this symbolic substitution process in the context of electronic computing.
- The study has concentrated on the use of symbolic substitution for addition. Research works may be undertaken to employ this process for multiplication and other arithmetic computation.
- The study has concentrated mainly on the development of symbolic substitution tables and algorithms. Existing optical hardware systems have been assumed as the backbone for the processing involved. However, a research on the design of efficient hardware for the symbolic substitution processes may lead to further computational efficiency. Also, efforts may be made on employing the concept of this optical

processing method in digital electronics domain. Research works may be undertaken to develop VLSI based systems that will allow the use of this parallel fast computation procedure in the conventional computing architecture.

- Symbolic substitution based approach has been developed for associative addition of binary numbers. Similar arrangements could be developed for associative addition of numbers represented in other number systems as well.

## BIBLIOGRAPHY

- [1] J. Ahmed and A. Awwal, "Polarization-encoded optical shadow-casting arithmetic-logic-unit design: Seperate and simultaneous output generatiou," *Applied Optics*, vol. 31, pp. 5622–5631, 1992.
- [2] M. S. Alam, A. Awwal, and A. Cherry, "Opto-electronic symbolic substitution based canonical modified signed-digit arithmetic," *Optics And Laser Technology*, vol. 29, pp. 151–157, 1997.
- [3] M. M. Eshaghian, M. E. Shaaban, and S. H. Lee, "Optical techniques for parallel image computing," *Journal Of Parallel And Distributed Computing*, vol. 23, no. 2, pp. 190–201, 1994.
- [4] A. Huang, "Architectural considerations involved in the design of an optical computer," *Proc. of the IEEE*, vol. 72, pp. 780–786, July 1984.
- [5] T. Kim, W. Jao, and S. W. K. Tjiang, "Arithmetic optimization using carry-save-adders," in *Design Automation Conference*, (California, USA), pp. 433–438, 1998.
- [6] A. Louri, "A symbolic substitution based parallel architecture and algorithms for high-speed parallel processing," in *Proc. of the 1990 ACM Annual Conference On Computer Science*, (Washington, DC, USA), pp. 173–179, February 1990.
- [7] A. Louri and K. Hwang, "A parallel architecture for optical computing," in *Proc. of the 3rd Conference On Parallel Processing For Scientific Computing*, (Philadelphia, PA, USA), pp. 414–418, Dec. 1989.
- [8] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions On Electronic Computers*, vol. 10, pp. 389–400, 1961.
- [9] B. Parhami, "Generalized signed-digit number systems: A unifying framework for redundant number representations," *IEEE Transactions On Computers*, vol. 39, no. 1, pp. 89–98, 1990.

- [10] D. S. Phatak, H. K. S. Kahle, and J. Lue, "Hybrid signed digit representations for low power arithmetic circuit," in *Proc. of the Power-Driven Micro-Architecture Workshop In Conjunction With ISCA '98*, (Barcelona, Spain), June 1998.
- [11] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "Theory and applications of the double-base number system," *IEEE Transactions On Computers*, vol. 48, pp. 1098–1106, Oct. 1999.
- [12] S. Shieh and C. Wu, "Assymmetric high-radix signed-digit number systems for carry-free addition," *Journal Of Information Science And Engineering*, vol. 19, no. 6, pp. 1015–1039, 2003.
- [13] M. G. Arnold, J. Garcia, and M. J. Schulte, "The interval logarithmic number system," in *Proc. of the 16th IEEE Symposium On Computer Arithmetic*, (Santiago de Compostela, Spain), pp. 253–261, June 2003.
- [14] A. Awwal and M. Karim, *Optical Computing: An Introduction*. John Wiley & Sons Inc., New York, 1992.
- [15] M. S. Alam, "Parallel optical computing using recoded trinary signed-digit numbers," *Applied Optics*, vol. 33, pp. 4392–4397, 1994.
- [16] Y. Li and G. Echmann, "Conditional symbolic modified signed-digit arithmetic using optical content addressable memory logic elements," *Applied Optics*, vol. 26, pp. 2328–2333, 1987.
- [17] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2001.
- [18] 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*, vol. 43, pp. 880–891, Aug. 1994.
- [19] A. Louri and K. Hwang, "A bit-plane architecture for optical computing with two-dimensional symbolic substitution," in *Proc. of the 15th Annual Symposium On Computer Architecture*, (Honolulu, Hawaii, USA), pp. 18–27, May 1988.
- [20] E. Johnson and M. Karim, *Digital Design: A Pragmatic Approach*. PWS-KENT Publishing Company, 1989.
- [21] B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *IEEE Transactions On Computers*, vol. C-37, pp. 1470–1476, Nov. 1988.

- [22] S. Cotofana and S. Vassiliadis, "Signed digit addition and related operations with threshold logic," *IEEE Transactions On Computers*, vol. 49, pp. 193–207, Mar. 2000.
- [23] P. J. Grabner, C. Heuberger, and H. Prodinger, "Carry propagation in signed digit representations," *Glasgow Mathematical Journal*, vol. 45, pp. 427–440, 2003.
- [24] J. O. Coleman and A. Yurdaku, "Fractions in the canonical-signed-digit number system," in *Proc. of Conf. on Information Sciences and Systems(CISS'2001)*, (Baltimore, USA), March 2001.
- [25] Ç. K.. Koç, "Parallel Canonical Recoding," *Electronics Letters*, vol. 32, pp. 2063–2065, Oct. 1996.
- [26] O. Eğecioğlu and Ç. K.. Koç, "Exponentiation using canonical recoding," *Theoretical Computer Science*, vol. 129, no. 2, pp. 407–417, 1994.
- [27] M. Hitz and E. Kaltofen, "Integer division in residue number system," *IEEE Transactions On Computers*, vol. 44, no. 8, pp. 983–989, 1995.
- [28] A. Lindström, M. Nordseth, L. Bengtsson, and A. Omondi, "Arithmetic circuits combining residue and signed-digit representations," in *Proc. of the Eighth Asia-Pacific Computer Systems Architecture Conference (ACSAC'2003)*, vol. 2823, (Aizu-Wakamatsu, Japan), pp. 246–257, September 2003.
- [29] W. J. Townsend, M. A. Thornton, and P. K. Lala, "On-line error detection in a carry-free adder," in *11th IEEE/ACM International Workshop On Logic & Synthesis*, (New Orleans, Louisiana), pp. 251–254, June 2002.
- [30] B. D. Andreev, E. L. Titlebaum, and E. G. Friedman, "Transformations of signed-binary number representations for efficient vlsi arithmetic," in *Proc. of the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, (Alberta, Canada), pp. 70–75, July 2003.
- [31] K. Hwang and A. Louris, "Optical arithmetic using signed-digit symbolic substitution," in *International Conference On Parallel Processing, Vol. 1 : Architecture (ICPP '88)*, (Pennsylvania, USA), pp. 55–64, Aug. 1988.
- [32] T. J. Cloonan, "Performance analyses of optical symbolic substitution," *Applied Optics*, vol. 27, pp. 1701–1707, 1988.
- [33] K. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Applied Optics*, vol. 25, pp. 3054–3060, 1986.

- [34] K. Brenner and J. N. Mait, "Optical symbolic substitution: system design using phase-only holograms," *Applied Optics*, vol. 27, pp. 1692–1700, 1988.
- [35] A. Louri, "Parallel implementation of optical symbolic substitution logic using shadow-casting and polarization," *Applied Optics*, vol. 30, pp. 540–548, 1991.
- [36] K. Brenner, "New implementation of symbolic substitution logic," *Applied Optics*, vol. 25, pp. 3061–3064, 1986.
- [37] G. Reitwiesner, "Binary arithmetic," *Advances in Computers*, vol. 1, pp. 231–308, 1960.
- [38] M. Joye and S. M. Yen, "Optimal left-to-right binary signed-digit recoding," *IEEE Transactions On Computers*, vol. 49, no. 7, pp. 740–748, 2000.
- [39] A. Saed, M. Ahmadi, and G. A. Jullien, "A number system with continuous valued digits and modulo arithmetic," *IEEE Transactions On Computers*, vol. 51, no. 11, pp. 1294–1385, 2002.
- [40] H. A. H. Fahmy and M. J. Flynn, "The case for a redundant format in floating point arithmetic," in *Proc. of the 16th IEEE Symposium On Computer Arithmetic*, (Santiago de Compostela, Spain), pp. 95–102, June 2003.
- [41] D. S. Phatak, T. Goff, and I. Koren, "Constant-time addition and simultaneous format conversion based on redundant binary representations," *IEEE Transactions On Computers*, vol. 50, no. 11, pp. 1267–1278, 2001.
- [42] S. Rajagopal and J. R. Cavallaro, "On-line arithmetic for detection in digital communication receivers," in *Proc. of the 15th IEEE Symposium On Computer Arithmetic*, (Colorado, USA), pp. 257–265, June 2001.
- [43] E. Swartzlander and A. Alexopoulos, "The sign-logarithm number system," *IEEE Transactions On Computers*, vol. C-24, pp. 1238–1242, Dec. 1975.
- [44] M. Niimura and Y. Fuwa, "Improvement of radix-2k signed-digit number for high speed circuit," *Journal Of Formalized Mathematics*, vol. 15, pp. 1–4, 2003.
- [45] Ç. K. Koç and S. Johnson, "Multiplication of signed-digit numbers," *Electronics Letters*, vol. 30, no. 11, pp. 840–841, 1994.



## PUBLICATIONS OF THE CANDIDATE RELATED TO THE THESIS

- [1] T. Imam and M. Kaykobad, "Symbolic substitution based canonical recoding algorithms," *International Journal of Computers and Mathematics with Applications*, 2004. (accepted for publication).
- [2] T. Imam and M. Kaykobad, "A new symbolic substitution based addition algorithm," *International Journal of Computers and Mathematics with Applications*, 2004. (submitted for publication).
- [3] T. Imam and M. Kaykobad, "New symbolic substitution tables and algorithms for the design of a fast addition unit," in *Proc. of the 6th International Conference on Computer & Information Technology(ICCIT)*, (Dhaka, Bangladesh), pp. 98–103, December 2003.
- [4] T. Imam and M. Kaykobad, "A new symbolic substitution based approach for the conversion of binary numbers to cmsd numbers," in *Proc. of the 6th International Conference on Computer & Information Technology(ICCIT)*, (Dhaka, Bangladesh), pp. 170–173, December 2003.

# INDEX

- AHSD, 11
- Associative addition
  - algorithm, 55
  - demonstration of algorithm, 56
  - SS unit, 57
  - steps for completion, 57
- auxiliary variable, 48
- Binary system, 1
  - bottleneck, 1
- carry free arithmetic, 5, 6
- CLA, 6
  - disadvantages, 7
- CMSD, 3, 8-10, 14, 27
  - addition, 27
  - advantage, 10
  - defined, 10
- DBNS, 12
  - advantage, 12
  - defined, 12
  - example, 12
  - interpretation, 12
  - limitations, 12
- Dual-rail encoding, 17
- electronic computing, 15
  - limitations, 15
- fast computation, 1
- fast computation approaches, 2
- GSD, 11
- ICSS, 16, 17
- MSD, 2, 8, 9, 14, 24, 26
  - addition, 24, 26
  - defined, 9
  - limitation, 9
- MSD to CMSD algorithm, 48
  - demonstration, 48
- Number System
  - Redundant, 8
- optical computing, 2, 15
  - advantages, 15
  - parallelism, 15
- parallelism, 1
- PCSS, 16, 21
- recognition phase, 18
- Redundant Number System, 8
  - advantages, 8
  - defined, 8
- Reitwiesner's algorithm, 48
  - demonstration, 48
- RNS, 10
  - addition, 11
  - defined, 10
  - disadvantage, 11
  - dynamic range, 10
  - example, 10
- substitution phase, 20
- Symbolic substitution, 16-18, 30, 32, 34, 38, 43
  - addition CMSD, 50
  - arithmetic Unit, 30

associative addition, 55  
Binary to CMSD, 32, 34  
CMSD addition, 27, 38  
definition, 2  
ICSS, 16, 17  
methods, 16  
MSD Addition, 26  
MSD addition, 24  
MSD to Binary, 43  
MSD to CMSD, 32  
multiple rules, 21  
one-step, 27  
optimization, 47  
optimized Binary to CMSD, 49  
optimized CMSD to binary, 53  
optimized MSD to CMSD, 49  
PCSS, 16, 21  
phases, 18, 20  
three-step, 24  
two-step, 26

