# ALTERNATIVE ARITHMETIC STRUCTURES USING REDUNDANT NUMBERS AND MULTI-VALUED CIRCUIT TECHNIQUES

by

Uğur Çini

B.S., Electronics and Communications Engineering, Yıldız Technical University, 1999

M.S., Electrical and Electronics Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Electrical and Electronics Engineering

Boğaziçi University

2010

# ACKNOWLEDGEMENTS

I am very thankful to Professor Avni Morgül, my thesis supervisor, who encouraged me to study all the time to complete this thesis. I hope I will be able to go on working together throughout my entire academic career. I am especially thankful to Professor Günhan Dündar and Professor Ali Toker for their constructive comments on the improvement of this thesis through the years of this study. I should also mention Professor Mitchell Thornton for teaching us advanced digital design and logic synthesis courses in my year of study at Southern Methodist University, TX, USA.

I am grateful to Professor Ece Olcay Güneş and Professor Şenol Mutlu for accepting to be in my thesis jury and for spending their valuable time for reading the manuscript.

Many thanks specially go to Mustafa Aktan for his great friendship and technical support in my thesis work. I should also acknowledge Emre Arslan, Cem Çakır, Yasin Çitkaya, Umut Yazkurt, Engin Deniz and BETA members for their friendship throughout my study.

I am very thankful to my mother, brother and especially my wife Büşra for their support and love. I would also thank all of my friends for their friendship and support throughout the years of this study.

I also thank Boğaziçi University Research Fund for supporting our work under the Project No: 05HA201.

# ABSTRACT


## ALTERNATIVE ARITHMETIC STRUCTURES USING REDUNDANT NUMBERS AND MULTI-VALUED CIRCUIT TECHNIQUES

Arithmetic circuits play a crucial role in VLSI technology. Arithmetic blocks are usually the most power consuming parts in a system since the switching activity is quite high. Alternative arithmetic implementations can be a solution to reduce power consumption and to increase the performance of the whole system.

Static CMOS digital design has robust working performance, where logic levels are kept at the two extremes, either the ground voltage or supply voltage. However, the voltage excursion between the supply voltage and ground at all nodes causes excessive power dissipation. This condition also generates noise over the whole circuitry, which is not desirable especially in mixed signal designs. Current-mode digital design techniques can be a solution for this issue especially whenever the switching activity is high. In the first part of the thesis, alternative current-mode arithmetic structures are built focusing on multi-valued circuits. Together with multi-valued logic implementations, signed-digit numbers and redundant number systems are also analyzed. The design issues of multi-valued circuits are discussed and novel building blocks for multi-operand addition are developed.

In the second part of the thesis, redundant arithmetic schemes for new generation reconfigurable systems are also analyzed. These techniques proposed here can be implemented efficiently by using recently introduced 6-input look-up table based field programmable gate array (FPGA) systems. A redundant double carry-save mode addition technique is proposed for the new generation FPGA devices. Using the proposed technique, efficient multiply-accumulate operations and finite impulse response filter structures for reconfigurable systems are developed.

# ÖZET


## YEDEKLİ SAYILAR VE ÇOK DEĞERLİ DEVRE TEKNİKLERİ İLE GELİŞTİRİLEN ALTERNATİF ARİTMETİK YAPILAR

Aritmetik devreler tümleşik devre tasarımında önemli bir yere sahiptir. Genellikle aritmetik bloklar bir sistemde en yüksek güç harcayan birimlerdir. Bunun nedeni, bu devrelerin anahtarlama aktivitelerinin oldukça yüksek olmasıdır. Alternatif aritmetik uygulamalarla tüm sistemin güç gereksinimi azaltılabilir ve sistem performansı arttırılabilir.

Statik CMOS sayısal tasarım dayanıklı çalışma performansına sahiptir. Bunun nedeni lojik seviyelerin iki uç değer olan besleme gerilimi ve toprak arasında korunmasıdır. Buna karşılık tüm düğümlerin besleme voltajı değeri miktarı salınması yüksek güç tüketimine ve devrelerde gürültüye neden olmaktadır. Bu durum özellikle karma sinyal devrelerinde istemeyen bir durumdur. Özellikle anahtarlama yoğunluğunun fazla olduğu durumlarda akım modlu sayısal tasarım bu duruma bir çözüm olabilir. Tezin ilk kısmında alternatif akım modlu aritmetik yapılar çok değerli mantık devreleri kullanılarak geliştirilmiştir. Çok değerli mantık devreleri ile beraber işaretli sayılar ve yedekli sayı sistemleri de ele alınmıştır. Çok değerli mantık devrelerinin tasarım gereksinimleri ele alınmış ve özgün çok terimli toplama devreleri önerilmiştir.

Tezin ikinci kısmında yeniden yapılandırılabilir sistemler için yedekli sayı sistemlerinin kullanımı incelenmiştir. Burada önerilen teknikler yeni nesil 6-girişli hafızalı sahada programlanabilen kapı dizisi (FPGA) sistemleri üzerinde etkin olarak gerçeklenebilmektedir. FPGA sistemleri için yedekli çift saklamalı toplama tekniği önerilmiştir. Önerilen teknik kullanılarak yeniden yapılandırılabilir sistemler üzerinde etkili çarp-topla işlemleri ve sonlu dürtülü filtre yapıları gerçeklenebilmektedir.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS/ABBREVIATIONS

| | |
|---|---|
| BDD | Binary Decision Diagram |
| BSD | Binary Signed-Digit |
| BSC | Binary Stored-Carry |
| BSCB | Binary Stored Carry-or-Borrow |
| CCWC | Counter Clockwise Cyclic |
| CML | Current Mode Logic |
| CMOS | Complementary Metal Oxide Semiconductor |
| CWC | Clockwise Cyclic |
| DCVSL | Differential Cascode Voltage Switch Logic |
| DSL | Differential Split-level Logic |
| FA | Full Adder |
| FIR | Finite Impulse Response |
| FPGA | Field Programmable Gate Array |
| LUT | Look-up Table |
| MAC | Multiply-Accumulate |
| MCML | MOS Current Mode Logic |
| MOS | Metal Oxide Semiconductor |
| MV | Multi-Valued |
| MVL | Multi-Valued Logic |
| NMOS | N-type Metal Oxide Semiconductor |
| OSD | Ordinary Signed Digit |
| PDN | Pull Down Network |
| PMOS | P-type Metal Oxide Semiconductor |
| SB | Radix-r Stored-Borrow |
| SC | Radix-r Stored-Carry |
| SCB | Radix-r Stored Carry-or-Borrow |
| SCL | Source-Coupled Logic |
| SD | Signed-Digit |
| VLSI | Very Large Scale Integration |

# 1. INTRODUCTION

Arithmetic design blocks play a crucial role in the digital and mixed-signal systems. The bottleneck of digital signal processing and many other digital systems is arithmetic blocks where addition and multiplication operations are the core units. As the VLSI technology reduces to nanometer size devices, power dissipation reduces and device speeds increase every year. However, silicon technology is reaching its physical limits, where the design sizes already have reached the atomic levels. In this thesis, alternative circuit design techniques are proposed for arithmetic systems that can help designing analog friendly circuits; or increase performance with lower power consumption.

Addition of numbers is the most basic operation of all arithmetic systems, such as subtraction, multiplication and division. In the basic addition structure, the addition time is proportional to the bit size of the numbers to be added since the next output digit depends on previous carry-out of each digit. This causes an unacceptable delay for many systems especially when the word-length of the input operands is high. Therefore, carry signal propagation must be eliminated in the arithmetic circuits. There have been various techniques developed for breaking the long carry chains of the arithmetic circuits [1-3]. Basically, these techniques are based on redundant number representations.

Redundant representation means that a number is represented in more than one way in the number system. As an example, if the number base is radix-4 (each weight of the digit is a multiple of $4^n$ where $n$ represents the $n$th digit) and digit set is selected to be {−3, −2, −1, 0, 1, 2, 3}, the numbers $0013$, $002\bar{1}$, and $01\bar{2}\bar{1}$ all have arithmetic value 7. In this representation, $\bar{x}$ represents $-x$ for each digit. The redundancy in the number system can provide the carry chains in the addition algorithms to be broken. This makes the arithmetic operations independent on the bit size of the operands [1]. The details of redundant systems and carry-free addition circuits are explained in details in the next chapter.

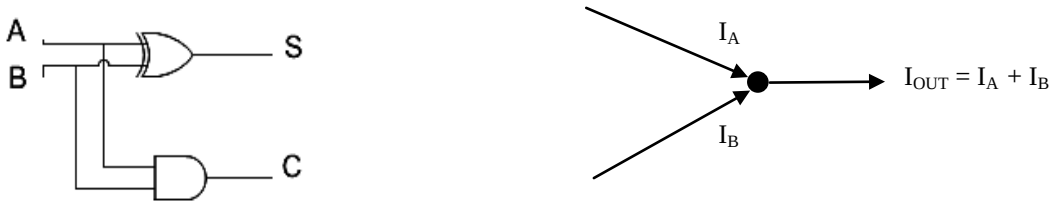
Redundant systems are not necessarily binary, but the numbers computed can be encoded as binary. As a result, more than one digit is required for the representation of

redundant systems when the system is implemented in binary logic. Hence, the redundancy in numbers means that the number structure is non-binary. Multi-valued circuit techniques are suitable to implement the arithmetic operations with such redundant numbers.

Considering redundant number systems in the number theoretic approach, carry propagation free algorithm development is achieved by using signed digit numbers or using carry save arithmetic. A signed digit system can be represented as $A - B$, where $A$ represents positive digits and $B$ represent negative digits. On the other hand, carry save format numbers can be represented as the addition of two numbers; $Z = \{S, C\}$, i. e., sum digits ($S$) and carry digits ($C$), respectively [4]. Shortly, carry-save mode addition techniques may also be considered as redundant number addition techniques where negative numbers do not exist in the digit set. The details of redundant systems are analyzed in Chapter 2. Since the redundant systems are not binary in principle, they can be implemented as multi-valued circuits. By multi-valued circuit techniques, carry-save mode redundant architectures or signed digit systems can be implemented.

Multi-valued logic (MVL) has always been an alternative logic design style, however, the circuit implementation drawbacks have been the obstacles over the years. In general, current-mode and voltage-mode implementation of MVL is possible. Voltage mode implementations of MVL are difficult in today's technology, since the low voltage design requirements do not leave enough voltage room for each logic level. Current-mode implementations of multi-valued circuits have the advantage of dynamic range, which gives flexibility in circuit design to have multiple logic levels on the same node. Another point is that, addition of two or more input operands is very easy in current mode MVL [5, 9]. The addition is implemented by simply interconnecting two current carrying wires (Figure 1.1), according to the Kirchoff's Current Law. Since addition of two branches needs no logic circuit and hence happens without delay in theory, it is clearly a great advantage against voltage mode circuits.

In this work, the focus is mostly on the implementation of arithmetic circuits by using MVL circuits. They have certain advantages over standard binary CMOS circuits. Standard CMOS digital design has robust working performance, where each node is kept in two extremes, either ground or supply voltage. However, the voltage excursion between




(a) (b)

Figure 1.1. (a) Voltage mode addition; (b) current mode addition

the supply voltage and ground at all nodes causes excessive power dissipation. Moreover, it generates noise over the whole circuitry. This is not desirable especially in mixed-system designs. Current-mode design techniques can be a solution for this issue especially when the switching activity is high. In static CMOS design, the power consumption is negligible when the circuit is not switching. Nevertheless, in current-mode designs the supply current is not zero and power is consumed all the time. As the supply voltages scale down and device geometry reduces to nanometer sizes, leakage current in voltage mode circuits becomes significant, giving a chance to the current-mode design techniques [10,11]. The significance of leakage currents for static CMOS appears at low voltage designs especially for the geometries lower than 0.18 μm.

Another property of the current-mode design is that the switching noise of the digital circuitry can be eliminated by using proper circuit techniques. This property is especially important for mixed-signal systems, where digital circuit noise is an important artifact. The current-mode circuits are driven by approximately constant current. This reduces d*i*/d*t* noise, which is generated by the inductive effect on the bonding wires of the chip.

The aim of this thesis work is twofold. The first is to implement alternative arithmetic blocks in multi-valued style with a simple and efficient design methodology. The second part of the research work is to implement efficient arithmetic algorithms using redundant number techniques with alternative number representations and implementations on reconfigurable systems for digital signal processing applications.

For the multi-valued circuit design, the efficiency metric is minimizing the area and interconnect and obtaining lower power at high speed. The research work here on the multi-valued circuit design is concentrated in enhancing the current comparators and the current switching circuits. Current comparison and current switching is the main design problem of all systems. Glitch free and fast current switching makes the system an alternative to standard CMOS logic. In an alternative multi-valued circuit design technique, it is also important to select a good arithmetic algorithm suitable for the hardware. Here, the aim is to find systems where multi-valued circuits have advantages over standard binary logic circuits. Therefore, designing with redundant numbers or signed digit multiple valued arithmetic is focused. Multi-valued to binary conversion may be required at the end, depending on the application of the system.

The starting point of this work has been designing high performance arithmetic circuits using alternative multi-valued and current-mode circuit design techniques. However, very high performance circuits can be built by binary logic by properly arranging the device sizes or by using dynamic circuit techniques like domino logic [12]. Gigahertz range addition circuits are realizable in binary circuits and these performance benchmarks are difficult to be beaten by current-mode multi-valued circuits. The reason for the limitations of current-mode logic circuits is that, the performance is limited by the current source and increasing the current to very high levels causes excessive current dissipation. On the other hand, current mode techniques allow controlling power consumption. Therefore, the power may be adjusted efficiently according to system requirements.

In the first part of this thesis work, novel current-mode multi-valued circuit topologies are developed that are especially efficient for multi-operand addition. The circuits developed here are original and unique to the best of our knowledge.

The secondary research effort in this work is to implement efficient arithmetic systems, especially multi-operand adders and redundant number systems on reconfigurable architectures, i.e., Field Programmable Gate Array (FPGA) devices. FPGAs, especially new generation high performance 6-input look-up table based structures can be used as multiple valued input  binary output devices, where efficient arithmetic structures can be

built using redundant multi-operand addition techniques. Finite Impulse Response (FIR) filter structures are implemented on FPGAs as signal processing applications to show the advantages of multiple valued addition schemes to achieve high operation speeds.

To summarize, Chapter 2 gives some background in the basic theory of redundant numbers and gives a short introduction to multi-valued logic. Chapter 3 provides basic current-mode circuits both in binary and multi-valued sense. Chapter 4 discusses the proposed multi-valued circuits and explores the design both in circuit techniques sense and redundant number system approach. Chapter 5 contains FPGA implementations, which are realized by using redundant numbers. Chapter 6 is the conclusion, which summarizes the outcome of the work done.

# 2. MATHEMATICAL BACKGROUND OF REDUNDANT NUMBER SYSTEMS AND MULTI-VALUED LOGIC

In this chapter, the mathematical basis of redundant numbers is given. A short and general introduction of multi-valued logic is also given for the analysis of various multi-valued logic implementations in literature.

## 2.1. Redundant Number Systems

In redundant number systems, a number can be represented in alternative ways. The redundancy gives extra information for representing a number. Moreover, redundancy is required for breaking the carry propagation chains of the addition process and is very useful when building arithmetic circuits, such as adders, adder trees, multipliers, etc.

Signed-digit numbers and carry-save arithmetic are well-known examples of the redundant number systems. Signed-digit arithmetic was first introduced by Avizienis [13]. Signed-digit systems were conceived with the purpose of implementing totally parallel addition [13]. Avizienis proposed an arithmetic system where carry propagation is eliminated. Carry propagation is eliminated by making each digit of the resulting sum a function of only two input digits. This is made possible by introducing redundancy in the number representation. A proper intermediate representation of the operand digit summation, $x_i + y_i$ is selected so that the final addition result can be generated using without requiring or generating carry signals [14].

In conventional number systems, numbers can be represented as $x_i \in \{0, 1, \ldots r-1\}$, where $r$ is the radix, $x_i$ is the $i$th digit of the number. In a signed digit number system, $x_i$ can take over the values such that $x_i \in \{ \overline{(r-1)}, \overline{(r-2)}, \ldots \bar{1}, 0, 1, \ldots (r-1)\}$. In this scheme, $\overline{(r-1)}$ equals to $-(r-1)$, $\bar{1}$ equals to $-1$, etc. Each digit can have positive and negative values. Therefore the system is named signed-digit (SD) representation [2].

Signed-digit representation gives redundancy in signal representation. As an example, –1 can be represented as $0\bar{1}$ or $\bar{2}1$. In radix 10 ($r = 10$), a *two-digit* number $X$ can have values in the range of $\overline{99} \leq X \leq 99$. Since $X$ is a two-digit number, each digit can have 19 (–9 to 9) different representation, where $X$ can have $19^2 = 361$ representations. The number $X$ can include 199 values (–99 to +99). Therefore, in this number system $X$ has 361 – 199 = 162 redundant numbers, which means that $X$ has 81 per cent (162/199) redundancy.

The algebraic value of a signed-digit number is given by

$$Z = \sum_{i=0}^{m} z_i r^i \ .$$

In this representation, $r$ is a positive integer called the radix, and each digit is denoted as $z_i$. In a redundant representation with radix $r$, each digit can assume more than $r$ values. In conventional number representations digits can assume exactly $r$ values, i.e. $z_i \in \{0, 1, \ldots (r-1)\}$. The values of the radix and the number digits, $z_i$, should satisfy the condition of a unique representation for the algebraic value $Z = 0$. It is then easy to prove that the algebraic value $Z$ is zero if, and only if, all digits of its signed-digit representation have the value $z_i = 0$. It is also evident that the sign of the algebraic value $Z$ is determined by the sign of the most significant non-zero digit. Similarly, the signed-digit representation of $-Z$, the additive inverse of $Z$, is obtained by changing the sign of every non-zero $z_i$ digit of $Z$.

Figure 2.1 depicts the totally parallel addition approach in a signed-digit arithmetic system. In the figure, $u_i$ is named as interim sum. The addition of two digits $x_i$ and $y_i$ is totally parallel if two conditions are satisfied. First, the interim sum digit $u_i$ is only the function of the operand digits, $x_i$ and $y_i$ [14]. Second, the carry digit to the next position $c_i$ is function only of the operand digits, $x_i$ and $y_i$. Totally parallel subtraction $x_i - y_i$ is realized as the totally parallel addition of $x_i$ and the additive inverse of $y_i$, that is, $x_i - y_i = x_i + (-y_i)$. The details of the operations are given in the following sections.

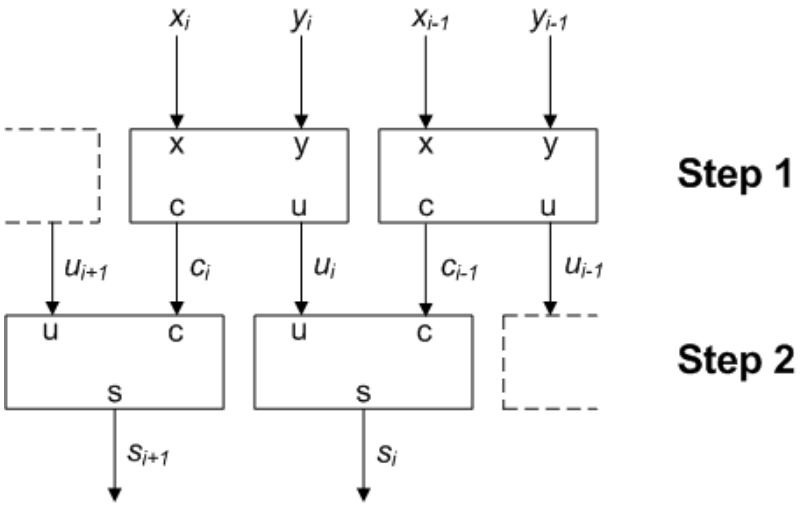



Figure 2.1. Parallel addition of two signed-digit numbers [13]

### 2.1.1. Generalized Redundant Number Systems

Generalized signed-digit number is a positional system with the digit set $\{-\alpha, -\alpha + 1, \ldots, 0, \ldots, \beta - 1, \beta\}$ where, $\alpha \geq 0, \beta \geq 0$, and for redundant number systems [1]:

$$\alpha + \beta + 1 > r \tag{2.1}$$

Here, $r$ is the number representation radix. For the case where $\alpha + \beta + 1 = r$ results in non-redundant number representation systems. The conventional radix-$r$ system with $\alpha = 0$ *and* $\beta = r - 1$ is the case for the generic $r$'s complement numbers and the 2's complement system is the most popular one. Redundancy index of a generalized signed-digit number system is defined as:

$$\rho = \alpha + \beta + 1 - r \tag{2.2}$$

A number can be represented as

$$Z = \sum_{i=0}^{m} z_i r^i \tag{2.3}$$

In this representation, $r$ is the radix of the number system and $z_i$ is inside the digit set $z_i \in \{-\alpha , -\alpha + 1, \ ..., 0, \ldots, \beta - 1, \beta\}$ where, $\alpha \geq 0, \beta \geq 0$. Parhami classifies some special cases inside the generalized signed-digit system [1]:

i. Binary stored-carry (BSC): $r = 2, \alpha = 0, \beta = 2$.
ii. Radix-r stored carry (SC): $\alpha = 0, \beta = r$.
iii. Binary stored-borrow or binary signed digit (BSB or BSD): $r = 2, \alpha = \beta = 1$.
iv. Radix-r stored-borrow (SB): $\alpha = 1, \beta = r - 1$.
v. Binary stored carry-or-borrow (BSCB): $r=2, \alpha = 1, \beta = 2$.
vi. Radix-r stored-carry-or-borrow (SCB): $\alpha = 1, \beta = r$.
vii. Minimally redundant symmetric signed-digit: $r \geq 4, 2\alpha = 2\beta = r$.
viii. Ordinary signed-digit (OSD): $r \geq 3, 1/2r < \alpha = \beta < r$ :
    a. Minimally redundant: $\alpha = \beta = \left\lfloor \frac{1}{2} r \right\rfloor + 1$.
    b. Maximally redundant: $\alpha = \beta = r - 1$.

Signed-digit systems for $r > 2$ are were first proposed by Avizienis [13]. Moreover, Parhami [1] proposed generalized signed-digit number representation that explains redundant number systems including binary case and other alternatives. Avizienis's signed digit system is named as ordinary signed-digit number system (OSD) in Parhami's work. Carry-save arithmetic is also one of the redundant number systems and has similar properties with signed-digit systems and it is defined in the following sections.

### 2.1.2. Ordinary Signed-Digit Number Representation

Adding some redundancy to a number system can be beneficial in some of the arithmetic operations; on the other hand, it is costly when it is used more than needed. Amount of redundancy in a signed digit representation can be restricted as follows:

$$z_i \in \{ \bar{a}, \overline{a-1}, \ldots \bar{1}, 0, 1, \ldots a\text{-}1, a\} \text{ with } \left\lceil \frac{r-1}{2} \right\rceil \leq a \leq r-1 \qquad (2.4)$$

where $z_i$ is the $i$th digit of a number. A number can be represented as in (2.3) using the digit set of (2.4). According to the definition of the generalized redundant number systems in Section 2.1.1, $\alpha = \beta = a$ in the ordinary signed digit representation. At least $r$ different digits in the digit set are necessary to represent a number in a radix $r$ system. In addition, each digit $z_i$ should satisfy $\bar{a} \leq z_i \leq a$ for each digit in this representation. There are $2a + 1$ alternatives for each digit in this representation. Therefore, the inequality $2a + 1 \geq r$ must be satisfied and the lower bound in the inequality is for that issue [13, 14]. As an example, in radix-8 representation, the value of '$a$' is between $4 \leq a \leq 7$ according to (2.4).

As stated above, the main advantage of the signed-digit (SD) representation is to eliminate the carry propagation chains in addition and subtraction. This is done by breaking the addition into two steps [13].

Step 1: Compute an interim sum $u_i$ and carry digit $c_i$:

$$u_i = x_i + y_i - rc_i \tag{2.5}$$

where

$$c_i = \begin{cases} 1 & \text{if } (x_i + y_i) \geq a \\ \bar{1} & \text{if } (x_i + y_i) \leq \bar{a} \\ 0 & \text{if } |x_i + y_i| < a \end{cases} \tag{2.6}$$

Step 2: Calculate the final sum:

$$s_i = u_i + c_{i-1} \tag{2.7}$$

Consider the example where $r = 10$ and $a = 6$. Step 1 will result in

$$u_i = x_i + y_i - 10c_i$$

where $c_i$ is calculated from:

$$c_i = \begin{cases} 1 \text{ if } (x_i + y_i) \geq 6 \\ \bar{1} \text{ if } (x_i + y_i) \leq \bar{6} \\ 0 \text{ otherwise} \end{cases}$$

If two numbers, 4536 and 1466 are added in a classical way,

$$\begin{array}{r} 4\ 5\ 3\ 6 \\ +\ 1\ 4\ 6\ 6 \\ \hline 6\ 0\ 0\ 2 \end{array}$$

Carry propagates from the first digit through the last one. If they are added in SD representation:

$$\begin{array}{rcl} 4\ 5\ 3\ 6 & & \\ +\ 1\ 4\ 6\ 6 & & \\ \hline 0\ 1\ 1\ 1 & & c_i \\ +\ 5\ \bar{1}\ \bar{1}\ 2 & & u_i \\ \hline 6\ 0\ 0\ 2 & & s_i \end{array}$$

As can be seen from this example no carry propagation appeared in the SD addition scheme.

To find an SD representation of a number, the aforementioned algorithm can be used. For example, to find the SD representation of 27956, each digit is represented as $x_i + y_i$. In other words, here, each digit can be viewed as an interim sum of a previous addition:

$$\begin{array}{lccccc} x_i + y_i & 2 & 7 & 9 & 5 & 6 \\ c_i \quad 0 & 1 & 1 & 0 & 1 & \\ u_i & 2 & \bar{3} & \bar{1} & 5 & \bar{4} \\ \hline s_i & 3 & \bar{2} & \bar{1} & 6 & \bar{4} \end{array}$$

When implementing SD addition scheme, to guarantee that no new carry is generated, the range of $a$ must be selected properly. In the case where $x_i + y_i = a$, which is the smallest value for which $c_i$ is still 1, $u_i = a - r < 0$. Substituting $|u_i| = r - a$ into $|u_i| \le (a-1)$ yields the inequality $2a \ge r+1$. Hence, the selected digit must satisfy:

$$\left\lceil \frac{r+1}{2} \right\rceil \le a \le r-1.$$

### 2.1.3. Binary Signed-Digit Architecture

In binary signed-digit (SD) addition schemes, it is not guaranteed that no new carry will be generated when computing $s_i$. Interim sum $u_i$ and carry generations should be revised, according to the Table 2.1 to assure that no new carry will occur in the generation of $s_i$ [15]. As can be seen in Table 2.1, interim sum and carry generations are made according to the $i$th digit $x_iy_i$ and the previous digit $x_{i-1}y_{i-1}$.

Table 2.1. Modified rules for adding binary SD numbers

| $x_iy_i$ | 00 | 01 | 01 | $0\bar{1}$ | $0\bar{1}$ | 11 | $\bar{1}\bar{1}$ |
|---|---|---|---|---|---|---|---|
| $x_{i-1}y_{i-1}$ | Not used | Neither is $\bar{1}$ | At least one is $\bar{1}$ | Neither is $\bar{1}$ | At least one is $\bar{1}$ | Not used | Not used |
| $c_i$ | 0 | 1 | 0 | 0 | $\bar{1}$ | 1 | $\bar{1}$ |
| $u_i$ | 0 | $\bar{1}$ | 1 | $\bar{1}$ | 1 | 0 | 1 |

For radix-2, the arithmetic here is named as binary signed-digit (BSD) or binary stored-borrow and the rules diverges from the ordinary SD systems. The modified rules for adding binary SD numbers as shown in Table 2.1 is not unique. Various binary signed-digit addition schemes can be generated [16, 17]. SD binary arithmetic implementations can be seen in [3, 18-23].

The binary signed digit addition scheme can be analyzed using the following example:

$$
\begin{array}{rl}
 & 1\ 0\ \bar{1}\ \bar{1}\ 1\ 0\ 0 \\
+ & 0\ 1\ \bar{1}\ 0\ 1\ 0\ \bar{1} \\
\hline
 & 1\ 1\ 0\ \bar{1}\ 0\ 0\ \bar{1} \qquad \text{interim sum } (u_i) \\
 & 0\ \bar{1}\ \bar{1}\ 0\ 1\ 0\ 0\ 0 \qquad \text{interim carry } (c_i) \\
\hline
 & 0\ 0\ 0\ 0\ 0\ 0\ 0\ \bar{1} \qquad \text{final sum } (s_i)
\end{array}
$$

where two SD numbers are added up based on the binary SD addition principles. The interim sum and carry digits are generated based on Table 2.1. The final sum is calculated by adding up the interim results. Note that, no new carry occurs in the final sum generation.

Another issue related to redundant systems is the format conversion. For SD systems, redundant binary to normal binary conversion is trivial. Since each number is represented as positive digits and negative digits, subtraction of negative digits from positive digits results the conversion to normal binary. There are other algorithmic approaches for these implementations for speed improvement of format conversion. In order to improve the speed of the conversion operation, other algorithmic approaches have been proposed [21, 22]. Conversion from unsigned normal binary to SD binary is also trivial, such that, all non-zero bits of the normal binary number are equal to the positive digits of the signed digit numbers, and all the negative digits are equal to zero. For 2's complement binary, sign bit is equal to the corresponding negative digit of the signed-digit number and all other positive digits are equal to the non-zero bits of the normal binary number, same as the normal binary case.

### 2.1.4. Carry-Save Arithmetic

Carry-save adder concept has been used for more than four decades. Wallace named the classical full-adder architecture for the usage of carry-save arithmetic as pseudo adder, since it reduces three input operands to two [24]. Wallace also designed parallel reduction tree for the reduction of the partial products of the multiplier. Carry-save reduction has been used in countless arithmetic applications where it is mostly used for partial product

reduction of multipliers. Carry-save arithmetic can also be generalized as a redundant binary system [1, 4].

Before defining the carry-save arithmetic, the most basic addition scheme, i.e. full adder, should be mentioned here. A full adder has three input operands $x$, $y$, $z$ and two outputs $s$ and $c$. The operation is described as

$$x + y + c_{in} = 2c_{out} + s$$

where, '+' operation is arithmetic addition here and each $x$, $y$, $z$, $c$ and $s \in \{0, 1\}$ here. The single bit cell of a full adder and a multi-bit ripple carry adder are depicted in Figure 2.2 (a) and 2.2 (b), respectively.

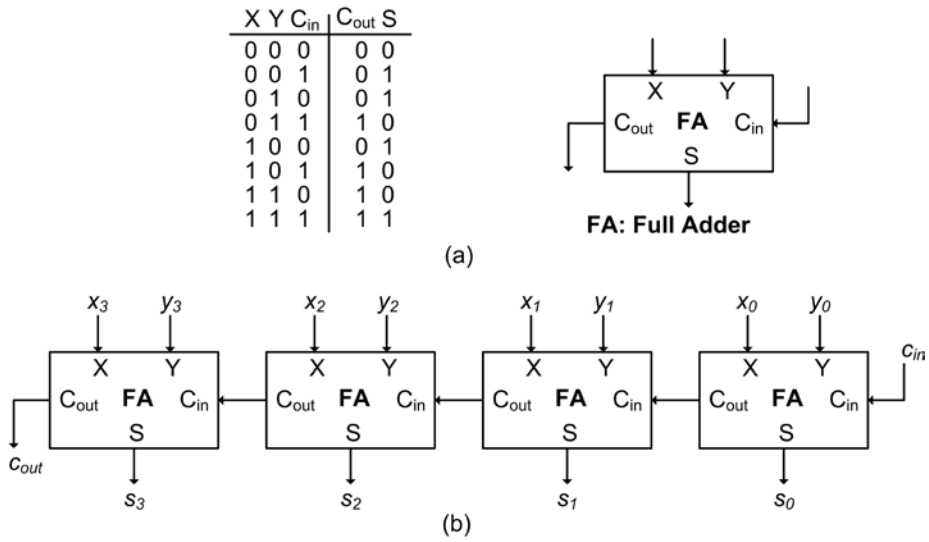



Figure 2.2. (a) Full adder representation; (b) ripple carry addition using full adders

Counter circuits are used for multi-operand addition of binary numbers. The simplest counter circuit is the (3, 2) counter which is simply a full adder. The only difference is the application, such that, all $x$, $y$, and $c_{in}$ inputs have same functionality, and, there is no carry ripple action contrary to the situation shown in Figure 2.2 (b). A (3, 2) counter counts the number of non-zero inputs and encodes the result into a two digit binary number. The maximum number of non-zero input digits is 3, hence we need at least 2 output digits to represent the result as shown in Figure 2.3 (a). The delay of the circuit is the single cell (3, 2) counter delay (single FA delay) as total, independent of the bit width of the operands.

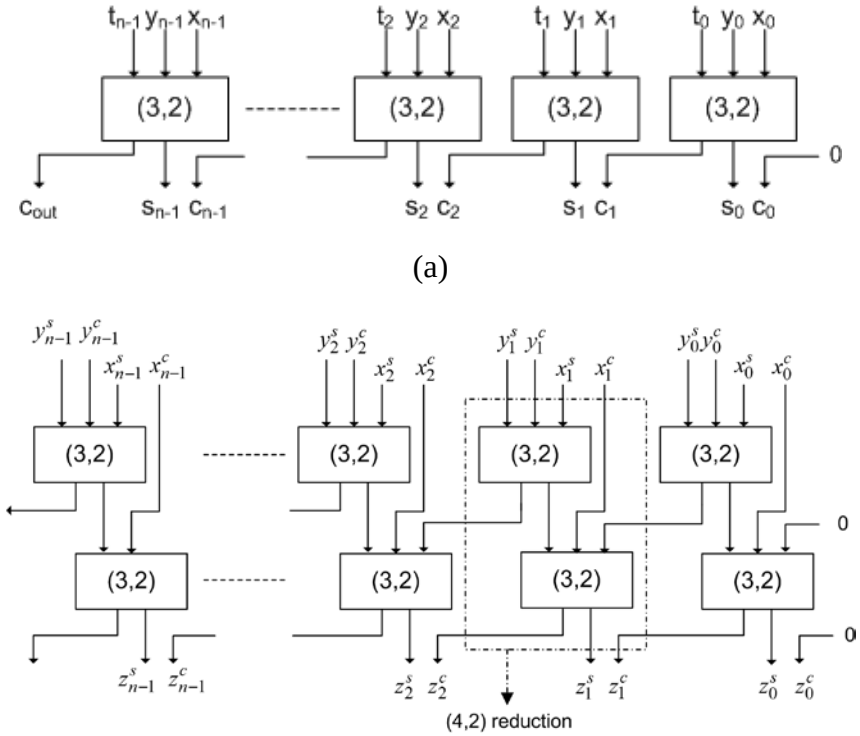



Figure 2.3. Carry-save adder structure: (a) normal binary inputs, (b) redundant inputs

In a carry-save system, each digit is represented as summation of two numbers, *sum* and *carry*. However, the final binary result, i.e. addition of this sum and carry, is not computed unless it is required. The output of the (3, 2) counter, composed of two numbers *S* and *C*, is an example to a carry-save number. In carry-save representation, each digit is represented as:

$$z_i = \{s_i, c_i\} \tag{2.8}$$

where the summation Z is represented as $Z = \{S, C\}$.

Carry-save arithmetic is a redundant system where the digit set is described as $z_i \in \{0, 1, 2\}$ and each digit of Z is equivalent to $z_i = s_i + c_i$. Carry-save arithmetic also provides carry propagation free addition similar to SD systems. Two arbitrary numbers in carry-save format can be added up using a (4, 2) reduction scheme, seen in Figure 2.3 (b). The (4, 2) reduction is realized by a two stage (3, 2) counter reduction scheme. Moreover, more compact (4, 2) reduction schemes for VLSI design exist for faster performance [25].

Carry-save arithmetic is an advantageous alternative to SD systems in many of the carry-free arithmetic applications. It should be noted here that, it is also possible to implement higher radix carry-save arithmetic providing less redundancy in the system [3].

Normal binary to carry-save conversion and the reverse should be noted here as well:

i. A normal binary number $X$ can be represented as in (2.8) in a way that $S = X$ and $C = 0$.
ii. Addition of two normal binary numbers $X$ and $Y$ can be represented as a single carry-save number such that: $S = X$ and $C = Y$ without applying any addition hardware.
iii. Three normal binary numbers can be converted to carry-save format (addition of three normal binary numbers with the output in carry save format) as shown in Figure 2.3 (a).
iv. And finally, a carry-save format number can be converted to normal binary by using a two operand adder, such as a ripple carry adder, where $s_i$ and $c_i$ components are added up resulting a normal binary number such that $Result = S + C$.

## 2.2. MVL Function Representation and Logical Operators

The proposed circuits in this thesis are not based on the formal definitions of MVL algebra. However, there are various works referenced in the thesis using MVL logical operators and formal definitions. This section is given as an introduction to general MVL function definitions.

Multi-Valued Logic (MVL) is a generalization of binary logic. Binary logic is a subset of MVL [26]. Multi-valued logic is a logic system which has more values than 0 and 1 contrary to binary logic.

Boolean logic can be binary Boolean logic or multi-valued Boolean logic where in multi-valued Boolean algebra the variable size must be a power of 2. There are other multi-valued logic systems rather than Boolean logic. Especially chain based Post algebra is an important research area of multi-valued logic which has functional completeness. Furthermore, contrary to multi-valued Boolean logic, in chain based Post algebra the

variable space need not to be a power of 2. Post algebra was first introduced by Emile Post in 1921 [27] and has been very famous among the multi-valued logic community. A particular set of Post algebra is the chain based Post Algebra which is functionally complete and attained much attention and welcomed much among the multi-valued logic research community. For the chain based Post algebra, when the variable space is restricted to binary, it is equivalent to binary Boolean Logic. The chain based Post algebra or chained Post algebra is defined as $\langle M,+,\cdot,L,\mathbf{0},\mathbf{1}\rangle$ [26-29], where $M$ is a totally ordered finite set containing $m$ elements {0, 1, …, $m$ – 1}, '+' is the "max" operator, '.' is the "min" operator and $L$ is the literal. Min, max and literal operators are defined in next section. In chain based Post algebra, logical zero **0** = 0 and logical one **1** = $m$ – 1 for an $m$-valued logic system [26-28].

For the Boolean case, the logic definition is only functionally complete in the set $m = 2^n$, $\forall n \in \{1,2,3...\}$. For chain based Post algebra, there is no restriction for functional completeness, and, $m$ may have any arbitrary positive integer value, i.e. it is functionally complete where $m$ = {1, 2, 3, …}. This means that chain based Post algebra gives more general view than Boolean logic. As stated above, whenever $m$ = 2 (binary case), Boolean algebra and chain based Post algebra are equivalent.

Multi-valued logic is studied for various intentions. For some researchers, multi-valued logic is a mathematical concept and can be viewed as a pure mathematical research area, which has a broad view in logic. Secondly, multi-valued logic can be used for logic synthesis of conventional binary logic circuits. Another research area is multi-valued circuit generation which also deals with electronics and circuit theory.

### 2.2.1. Basic Definitions of Multi-Valued Logic in Chain Based Post Algebra

Considering a multiple valued combinational function *f(x) in chain based Post algebra*, which is $m$ valued, where $X = \{x_1, x_2, ... , x_n\}$, each $x_i$ can take $m$ values from the set $M = \{0, 1, ... m\text{-}1\}$. Here, the mapping of the function is defined as f: $M^n \rightarrow M$. There are $m^{m^n}$ different functions that can be implemented. Table 2.2 shows the number of

functions that can be implemented, depending on the radix and variable count. Basic operations in MVL and their similar counterparts in binary logic are shown in Table 2.3.

Table 2.2. Number of combinational logic functions by MVL

| *n* variable / radix (*r*) | 2 | 3 | 4 |
|---|---|---|---|
| 2 | 16 | 256 | 65536 |
| 3 | 19683 | $7.63 \times 10^{12}$ | $4.43 \times 10^{38}$ |
| 4 | $4.29 \times 10^{9}$ | $3.4 \times 10^{38}$ | $1.53 \times 10^{154}$ |

Definition: A min operator can be defined as:

$$\min(a_1, a_2, \ldots a_n) = a_1 \cap a_2 \cap a_3 \ldots \cap a_n \tag{2.9}$$

where $a_1, a_2, \ldots a_n \in R = \{0, 1, \ldots r\text{-}1\}$. For $a_1 = 5$, $a_2 = 2$, $a_3 = 3$, min(5,2,3) = 2. The operator '$\cap$' will be replaced by '.' in the following representations for simplicity [30-32].

Definition: A max operator is defined as:

$$\max(a_1, a_2, \ldots a_n) = a_1 \cup a_2 \cup \ldots \cup a_n \tag{2.10}$$

where $a_1, a_2, \ldots a_n \in R = \{0, 1, \ldots r\text{-}1\}$. For $a_1 = 5$, $a_2 = 2$, $a_3 = 3$, max(5,2,3) = 5. The operator '$\cup$' will be replaced by '+' in the following representations for simplicity.

Table 2.3. Multiple-valued counterparts of basic MVL operations

| **Binary** | **MVL** |
|---|---|
| NOT | Complement, Cycle |
| AND | MIN |
| OR | MAX |
| XOR | SUM (mod *r*) |

Definition: Complement of $x$, is defined as:

$$\bar{x} = (r-1) - x \tag{2.11}$$

For 4 valued-logic, if $x = 2$, $\bar{x} = (4-1) - 2 = 1$.

Definition: Cycle operator can has two definitions, *y*-clockwise cyclic operator (CWC) and y-counter clockwise cyclic operator (CCWC):

$$\begin{aligned} y\text{-CWC: } & x^{k\rightarrow}: (x + y) \bmod r \\ y\text{-CCWC: } & x^{k\leftarrow}: (x - y) \bmod r \end{aligned} \tag{2.12}$$

Clockwise cyclic operator is equivalent to modular sum of the two variables. Table 2.4 shows the truth table of min, max, and modular sum operators.

Definition: The literal is defined as:

$$x^S = \begin{cases} r-1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

where $S \subseteq M$. If $m = 2$, i.e. binary case, then we get $x^{\{1\}} = x$ and $x^{\{0\}} = x'$. If $S$ is a single element set the brackets can be omitted and $x^{\{1\}}$ can be written as $x^1$.

Definition: The window literal or, literal is defined as:

$$x^{a,b} = \begin{cases} r-1 & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

Definition: The complementary window literal is defined as:

$$\overline{x^{a,b}} = \begin{cases} r-1 & \text{if } x < a \text{ or } x > b \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

MVL unary operations can be seen in Table 2.5.

Definition: A product term, having the value of $k$ ($k \leq r-1$) is given by

$$P_k : \bigcap_{i\in\{0,1,2...n\}} g(x_i) = k \tag{2.16}$$

where, $g(x_i)$ refers to a unary operator such as literal, cyclic operator or any other, and, '$\cap$' refers to min operator over unary operators.

Definition: A multiple-valued combinational function can be expressed in terms of products as:

$$f(x_0, x_1 ... x_n) = \bigcup P_j \tag{2.17}$$

where '$\cup$' implies the max operator and $P_j$ represents all the products for the implementation of the function.

Any multi-valued function has a canonical expression in chain-based Post algebra in terms of max, min and literals, which are defined above. As an example, a two-variable, 3-valued function shown in Figure 2.4 has the following canonical expression:

$$f(x_1, x_2) = 1 \cdot x_1^1 \cdot x_2^0 + 1 \cdot x_1^1 \cdot x_2^2 + 2 \cdot x_1^0 \cdot x_2^1 + 2 \cdot x_1^1 \cdot x_2^1 + 2 \cdot x_1^2 \cdot x_2^1$$

| $x_2$ \ $x_1$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 2 | 2 | 2 |
| 2 | 0 | 1 | 0 |

Figure 2.4. An example multi-valued function

Table 2.4. Truth table of Min, Max and Modular Sum (Msum) operators

| Min | 0 | 1 | 2 | ... | *r*-1 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 0 |
| 1 | 0 | 1 | 1 | | 1 |
| 2 | 0 | 1 | 2 | | 2 |
| ... | | | | | |
| *r*-1 | 0 | 1 | 2 | | *r*-1 |

| Max | 0 | 1 | 2 | ... | *r*-1 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | | *r*-1 |
| 1 | 1 | 1 | 2 | | *r*-1 |
| 2 | 2 | 2 | 2 | | *r*-1 |
| ... | | | | | |
| *r*-1 | *r*-1 | *r*-1 | *r*-1 | | *r*-1 |

| Msum | 0 | 1 | 2 | ... | *r*-1 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | | *r*-1 |
| 1 | 1 | 2 | 3 | | 0 |
| 2 | 2 | 3 | 4 | | 1 |
| ... | | | | | |
| *r*-1 | *r*-1 | 0 | 1 | | 2x(r-1) mod *r* |

Table 2.5. MVL unary operations

| **Name** | **Notation** | **Definition** |
|---|---|---|
| Cycle | $x^{k\rightarrow}$ | *(x + k)* mod r |
| Cycle – Counter clockwise | $x^{k\leftarrow}$ | *(x - k)* mod r |
| Successor | *x+* | *(x + 1)* mod r |
| Predecessor | *x-* | *(x – 1)* mod r |
| Complement (negation) | *x'* | *(r – 1) – x* |
| Selection literal (literal) | $x^{S}$ | *r – 1* if *x* in *S*; 0 otherwise |
| Window literal | $x^{a,b}$ | *r – 1* if *a<=x<=b*; 0 otherwise |
| Complementary window literal | $\overline{x^{a,b}}$ | *r – 1* if *x<a* or *x>b*; 0 otherwise |

**Property 2.1**: The following properties of literals hold:

$x^i \cdot x^j = \mathbf{0}$, for any $i, j \in M, i \neq j$.

$$\sum_{i=0}^{m-1} x^i = \mathbf{1}.$$

For example, the function in Figure 2.4 can be represented as

$$f(x_1, x_2) = 1 \cdot x_1^1 \cdot x_2^0 + 1 \cdot x_1^1 \cdot x_2^2 + 2 \cdot x_2^1$$

where '+' is max and '.' is min operators. Here, max(*a*, *b*) = *a* for any $a \geq b, a, b \in M$ . The following rule can also be used for simplification [28].

**Property 2.2**: Let *a* and *b* be constants in *M* such that $a \leq b$ . Then

$$a \cdot x_1^i + b \cdot x_2^j = a \cdot (x_1^i + x_2^j) + b \cdot x_2^j .$$

Using the above property, the function in Figure 2.4 can be reduced to:

$$f(x_1, x_2) = 1 \cdot x_1^1 + 2 \cdot x_2^1 .$$

Furthermore, the constant 2 can be omitted from the product terms since 2 is the unit function (*m*-1 = 2) by definition. As a result the function reduces to:

$$f(x_1, x_2) = 1 \cdot x_1^1 + x_2^1$$

There are various references about multi-valued logic minimization. Some of which appear in [33-37].

### 2.2.2. Functional Completeness of Multi-Valued Logic of Post Algebra

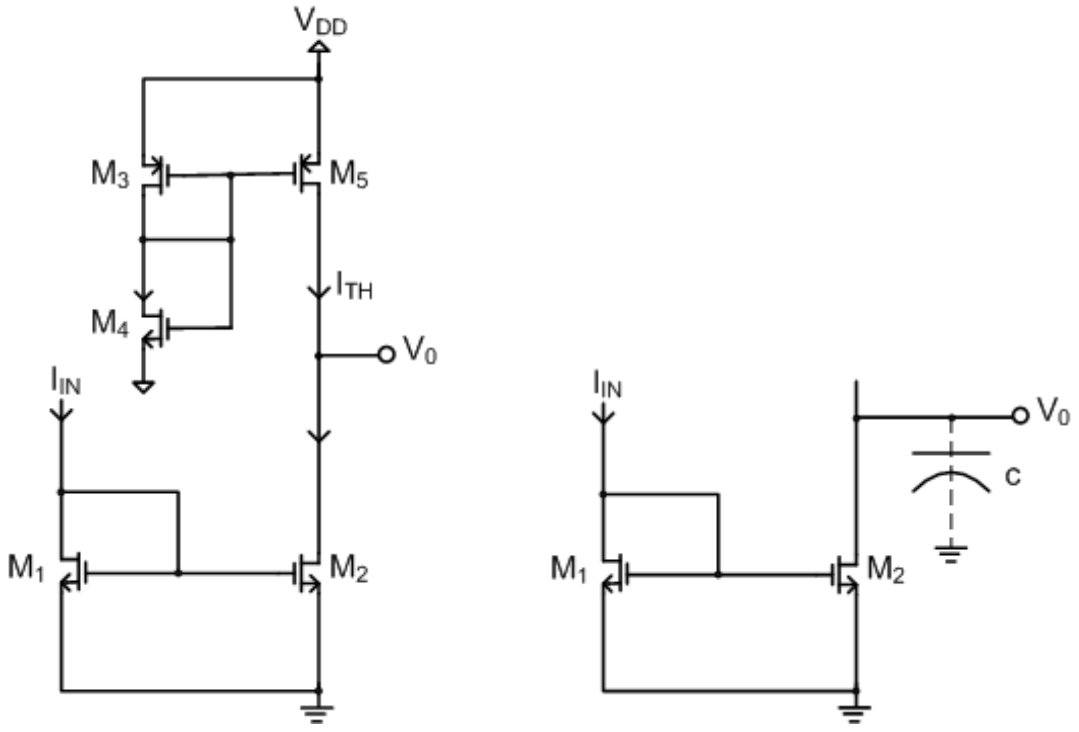
Functional completeness is a critical issue for logic functions where, if it is provided, any kind of logic circuit can be realized. Once a complete set of functions is defined, any logic circuit can be constructed from the gates implementing the primitive functions from this set. Basic definitions of functional completeness of chain based Post algebra is defined in [28, 29].

# 3. BINARY AND MULTI-VALUED CURRENT-MODE LOGIC DESIGN

In this chapter, an introduction to current comparators and current-mode binary logic (source-coupled logic) is given. Furthermore, some of the current mode multi-valued circuits appearing in the literature are explored. The aim is to give a basis and comparison for the contributions in this work.

## 3.1. Current Comparator Circuits

Current comparator circuits are the most common building blocks of current mode logic circuits. One of the simplest and most popular current comparator circuits has been proposed in [38]. The circuit is shown in Figure 3.1.




(a) (b)

Figure 3.1. (a) Current comparator circuit with voltage mode output; (b) representation of equivalent output capacitance

Here, the input current $I_{IN}$ is compared to the threshold current $I_{TH}$. In the circuit, the current $I_{IN}$ is copied to $M_2$. When the input current is less than the threshold current, the output voltage will become high.

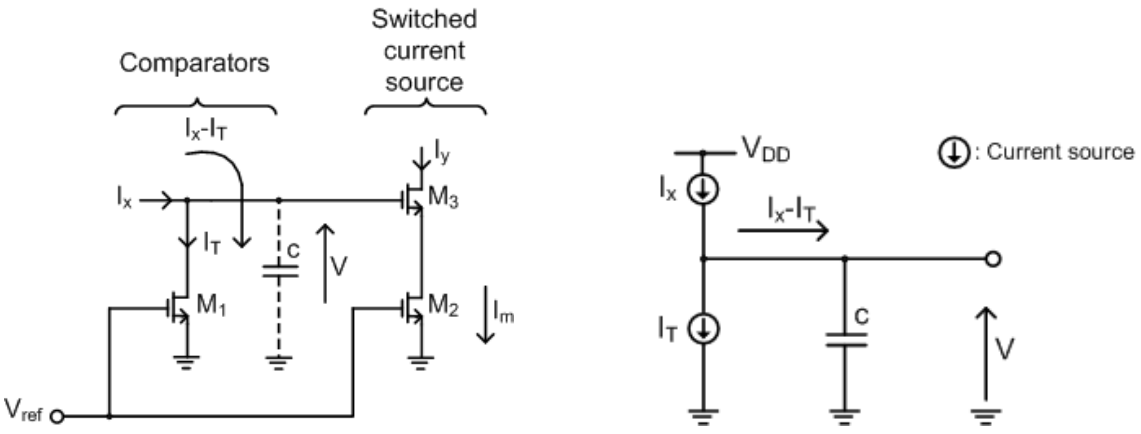
$$V_0 = \begin{cases} \text{High} & \text{if } I_{IN} < I_{TH} \\ \text{Low} & \text{if } I_{IN} \geq I_{TH} \end{cases} \tag{3.1}$$

The transistor $M_2$ behaves as an active loaded inverter. A large gain is desired to provide a sharp comparator transition and greater noise margin. Assuming that the equivalent capacitance at the output is assumed as $C$, the delay of the circuit is estimated by:

$$t_{DELAY} \ \alpha \ \ C\frac{V_{DD}}{|I_x - I_T|} \tag{3.2}$$

This delay is controlled by the input current range ($|I_x - I_T|$), supply voltage ($V_{DD}$) and equivalent input capacitance $C$.

Another current comparator circuit is proposed in [7]. The circuit and its equivalent is shown in Figure 3.2.




(a) (b)

Figure 3.2. (a) Threshold comparator [7]; (b) equivalent circuit

The threshold function is defined as:

$$I_y = \begin{cases} 0 & \text{if } I_x < I_T \\ I_m & \text{if } I_x \geq I_T \end{cases} \quad (3.3)$$

Here, the transistor $M_1$ is assumed to operate in the saturation region. For multi-valued logic design, each logic level is defined as multiples of $I_0$ unit current. The comparator values are set between in between each logic level such as $I_T = 0.5\ I_0$ to detect logic level 1 (i.e. $I_0$), $I_T = 1.5\ I_0$ to detect logic level 2 etc. In the worst case, the difference between the comparison current and variable current is $0.5\ I_0$. Using (3.2) the worst case delay of a comparator is estimated as:

$$t_{DELAY}\ \alpha\ \ C\frac{V_{DD}}{0.5I_0} \quad (3.4)$$

It is possible to reduce the delay by decreasing the supply voltage $V_{DD}$, or alternatively increasing the unit current $I_0$. However, increasing the logic level current increases power dissipation. In the following sections, current mode binary logic and a similar scheme for multiple-valued implementation will be explained.

### 3.2. Binary Current-Mode Logic (Source-Coupled Logic) Circuits

Since current-mode multi-valued circuits will be explored, binary current mode logic (source-coupled logic) circuits are analyzed here in order to compare the functionality of the multi-valued circuits. In addition, it is possible to use current mode binary circuits in some building blocks of MVL circuits when needed.

A digital circuit style that seems to be promising in both reducing power consumption and providing an analog friendly environment is MOS Current Mode Logic (MCML). The Source-Coupled Logic (SCL) is synonymous with MOS Current-Mode Logic (MCML), where SCL expression is more popular in the literature. In the past, high-speed digital ICs were often realized in Si bipolar and III/V compound technologies. However, as the feature sizes scales down, nowadays it is possible to realize high speed

ICs at Gigabit/s speeds with CMOS circuits [39]. Moreover, if current mode logic (or source couple logic equivalently) circuits are used, even higher speeds can be achieved because of its shorter logic level voltage swing [40, 41]. While bipolar CML, a derivative of emitter coupled logic (ECL), has been used for years in high performance applications, it has become less desirable over time due to its high static power consumption and reliance on bipolar processing [42].

The ideal gate implemented in source-coupled logic manner is shown in Figure 3.3. It consists of three main parts: the pull up resistors, the pull down network switch, and the current source. The inputs to the pull down network (PDN) are fully differential. In other words, the true and complement of all logical inputs must be present. The PDN can implement any logic function but must have a definite value for all possible input combinations. In general, the design of the SCL pull down network is similar to other differential logic styles such as differential cascode voltage switch logic (DCVSL) or differential split-level logic (DSL).

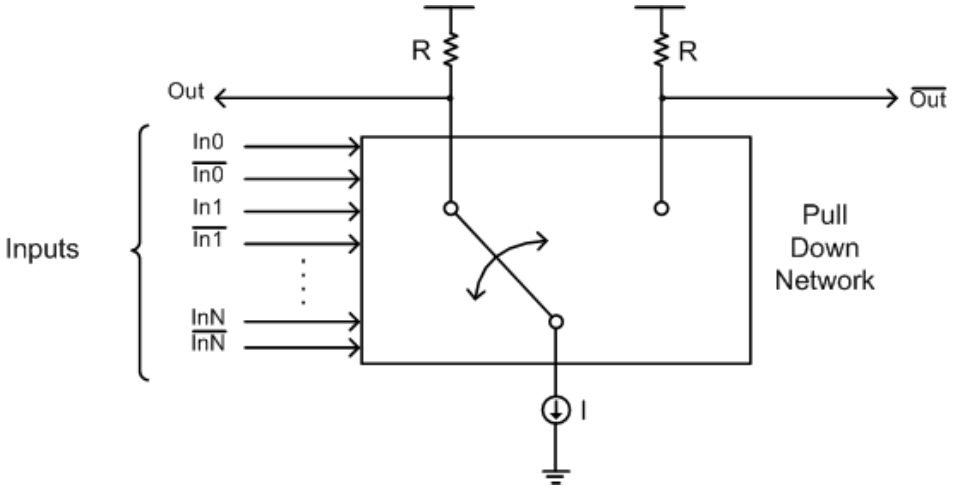



Figure 3.3. Binary Current Mode Logic circuit

Unlike DCVSL or DSL, the pull down network in MCML circuits is regulated by a constant current source. The pull down network steers the current $I$ to one of the pull up resistors based upon the logic function being implemented. The resistor connected to the current source through the PDN will have current $I$ and a voltage drop equal to $\Delta V = I \times R$. The other resistor will not have any current flowing through it and its output node will be pulled up to $V_{DD}$ in the DC state. If we look at the differential output voltage, the total

voltage swing is set exclusively by the amount of current (*I*) and the value of the pull up resistance (*R*). This voltage swing is generally much smaller than $V_{DD}$, in the order of a few hundred millivolts.With this simple model in mind, some basic transient properties of a circuit composed of SCL can be derived. For simplicity, assuming that the circuit is a linear chain of *N* identical gates, all with identical load capacitance *C* on each output node. The total propagation delay (*D*), power (*P*), and power delay (*PD*) product is approximated by [42]:

$$D_{MCML} = NRC = \frac{NxCx\Delta V}{I} \tag{3.5}$$

$$P_{MCML} = NxIxV_{DD} \tag{3.6}$$

$$PD_{MCML} = N^2 xCx\Delta VxV_{DD} \tag{3.7}$$

For comparison, the delay, power and power-delay for static CMOS logic are given by:

$$D_{CMOS} = \frac{N \times C \times V_{DD}}{\frac{k}{2} \times (V_{DD} - V_T)^{\alpha}} \tag{3.8}$$

$$P_{CMOS} = N \times C \times V_{DD}{}^2 \times \frac{1}{D_{CMOS}} \tag{3.9}$$

$$PD_{CMOS} = N \times C \times V_{DD}{}^2 \tag{3.10}$$

Figure 3.4 shows an inverter/buffer designed in SCL style. All SCL gates have one current source device and two load devices. Different logic functions are implemented with different pull down networks. The implementation of a logic function using SCL gates can easily be determined by using a Binary Decision Diagram (BDD) [43]. One important property to note is the relative homogeneity of gate topologies in SCL circuits. If the leftmost gate in Figure 3.5 is explored, it can be seen that the AND, OR, NAND, and NOR functions all have exact the same topology and hence the same sizing, delay, power, etc. The only difference in implementing these functions is the ordering of the inputs and outputs. This uniformity leads to more predictability in the timing and area of cells and reduces the need for Boolean manipulation in order to transform into inverting logic [41, 42].

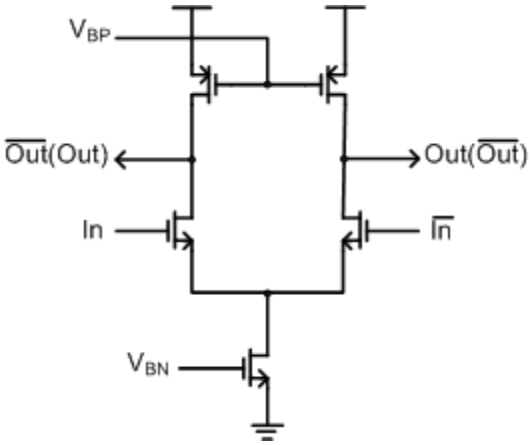



Figure 3.4. MCML inverter/buffer

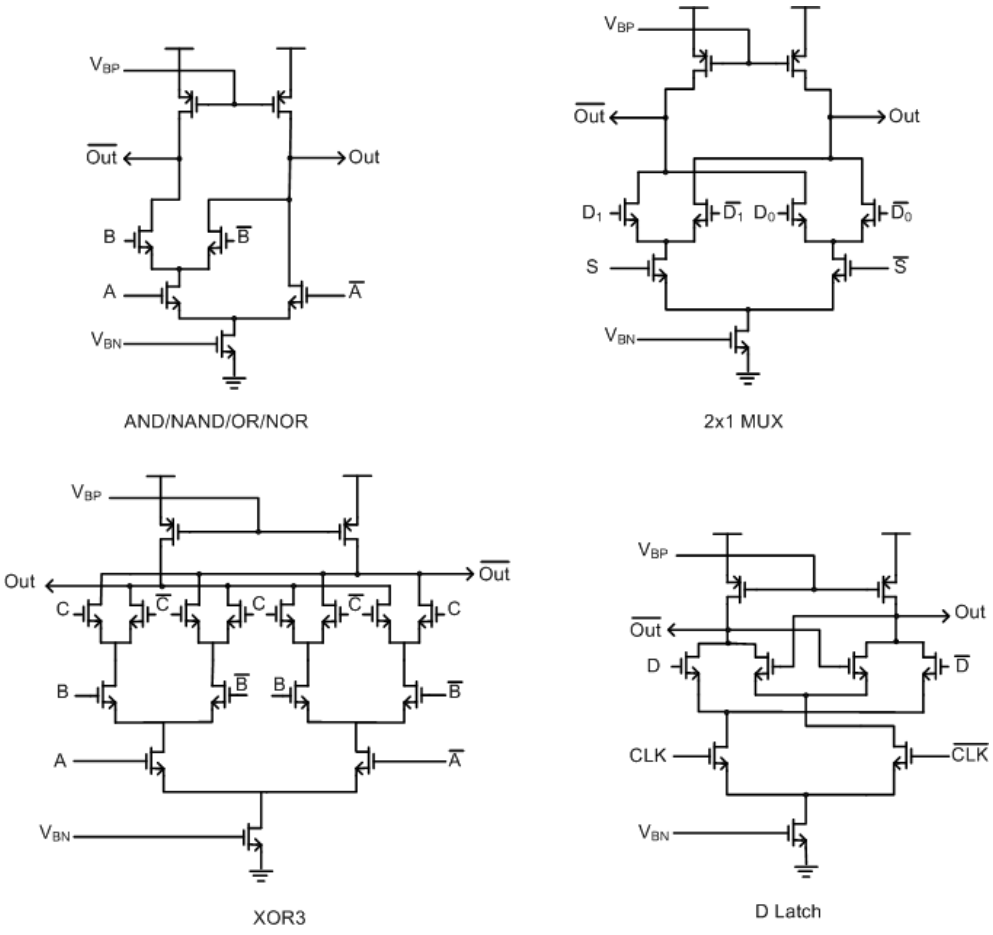



Figure 3.5. Realization of Boolean functions with MOS Source-Coupled Logic (SCL)

### 3.3. Arithmetic Building Blocks of Multi-valued Logic Circuits

Current-mode circuits provide the advantage that interconnecting two of the outputs current sources results in the addition of the currents at the interconnected nodes. This provides great saving in the addition process and is beneficial for the generation of multi-valued arithmetic circuit implementations.

Some of the first implementations of the current-mode multi-valued arithmetic circuits appear in [44] and [45]. One of the first redundant binary addition applications is presented in [46]. The circuits designed in [46] contained bi-directional current sources. The bi-directional implementation of addition and subtraction by currents results in slower operation, since the addition and subtraction operations for the currents are implemented at the same node. The reason for the slow operation is that, the bi-directional connections overload the interconnection nodes with high amount of capacitances of the active devices by double operations of addition and subtraction at the same node. Bi-directional current source implementations also appear in [30, 31] together with arithmetic and logic multi-valued logic implementations. There are similar redundant binary applications in [5, 47]. An improved work for the redundant arithmetic appears in [7] where differential multi-valued design style is employed. This application provides faster response with better noise immunity; on the other hand, the circuit size is doubled.

An improved work of signed-digit redundant addition approach appears in [8]. Here, redundant binary arithmetic applications are handled using differential circuit design, similar to binary source coupled logic systems.

Besides these, Etiemble [48] presents a simple addition circuit which is applicable to multi-operand addition as well; the circuit presented is simple and useful for the arithmetic implementations. Another approach for multi-valued logic for arithmetic is presented by Temel [9, 32]. Temel implements high radix addition algorithms that contain novel circuit implementations. In the following sections of this chapter, some of the selected works of multi-valued arithmetic circuit implementations are explained in more detail to give better insight in current-mode multi-valued arithmetic applications. In the next section, the methodology of this thesis work is presented for various arithmetic applications. This work

focuses on multi-operand addition which makes it different from the aforementioned approaches.

### 3.3.1. Basic Radix-4 Current Mode Adder Architecture

K. W. Current presents radix-4 current mode adder architecture in [49]. In that work, radix-4 inputs of *A* and *B*, and binary input $C_i$ are added as currents. $I_{in}$ *(*$I_{in} = A + B + C_i$*)* current is the input of the adder scheme. Since the inputs *A* and *B* are radix-4 (values between 0 - 3) and $C_i$ is binary, 3 + 3 + 1 = 7 different current levels exist. The circuit of radix-4 full adder is shown in Figure 3.6. Here, *A, B, C* and *D* are the outputs of the current comparators. *D* output is compared whether the current value is greater than *4*I (I is the unit current corresponding to a logic value of 1). If the value is equal or greater than 4, then *D* output is activated and carry output of the circuit will be active. Here, when the output *D* is active, a current 4I is fed-back and subtracted from the result, so as to reduce the comparison value. Therefore, current comparators compare the values of 0 to 4 instead of 0 to 7, by the help of the feedback circuit. Transmission gates provide the current switching in the circuitry. The summation output is regenerated using unit currents at the output.

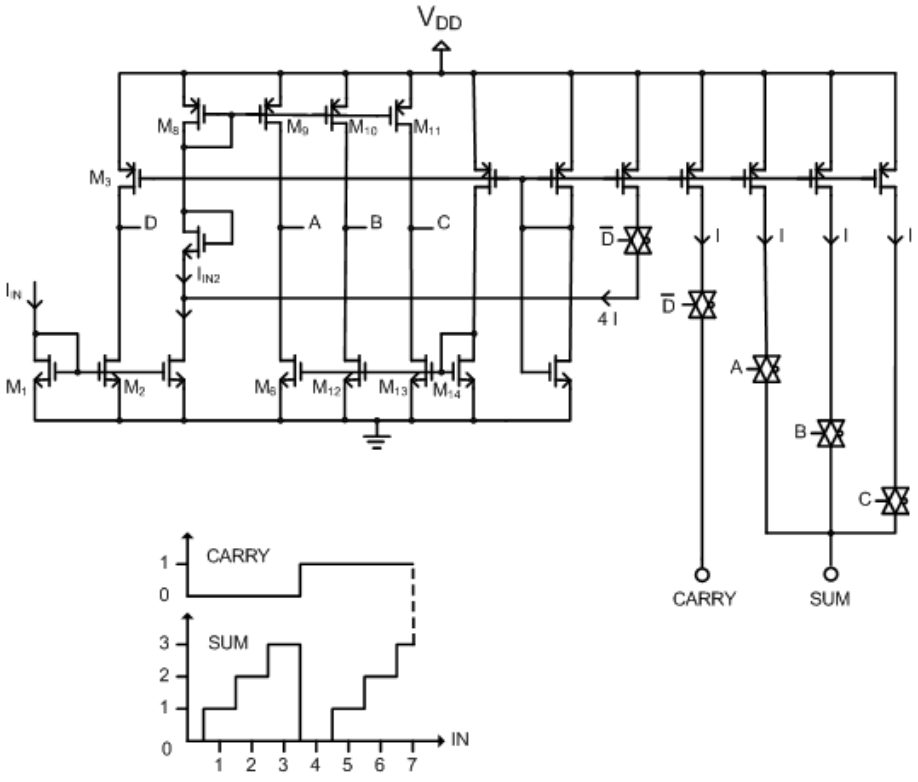



Figure 3.6. Current mode radix-4 adder circuit and its logical diagram [49]

### 3.3.2. Implementing MVL Functions Using Current-Mode Logic

Current-Mode Logic (Source-Coupled Logic) can be used in multiple valued functions as well. T. Ike et. al. proposed a new style of source coupled circuits to be used with multiple valued logic style [8]. The common part of this new style is the multiple valued source coupled comparator which is shown in Figure 3.7.

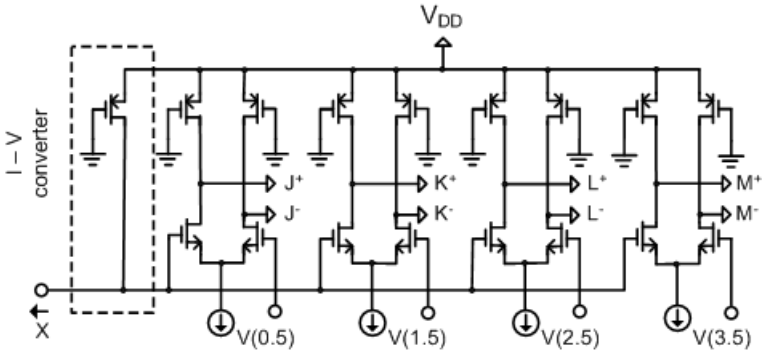
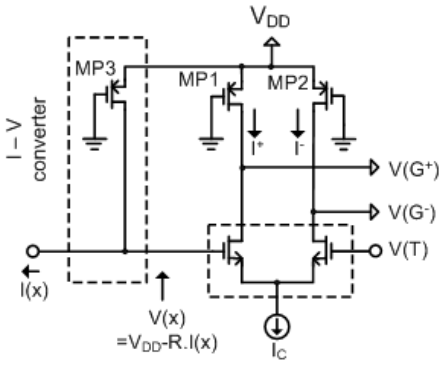



Figure 3.7. Multiple-valued Source Coupled comparator

Here, the current value *X* is converted to a voltage value and compared with four source-coupled pair circuits to detect five different levels. Detailed analysis of this source coupled comparator is shown in Figure 3.8.




| X | V(x) | I+ | I- | V(G+) | V(G-) | G+,G- |
|---|---|---|---|---|---|---|
| >T | <V(T) | 0 | $I_C$ | $V_{DD}$ | $V_{DD}$-$R_C$.$I_C$ | (1,0) |
| <T | >V(T) | $I_C$ | 0 | $V_{DD}$-$R_C$.$I_C$ | $V_{DD}$ | (0,1) |

Figure 3.8. Analysis of Source Coupled Comparator circuit

In this circuit, Current *I(X)* is converted to *V(X)*. The variable voltage *V(X)* is compared with a constant voltage level *V(T)*. *V(X)* value is the drain voltage of the PMOS transistor MP3. Since, the gate of the transistor is at ground voltage, the transistor is always

in linear region. The voltage *V(X)* is derived from the basic MOS transistor linear region equations:

$$V(X) = k - \sqrt{k^2 - \frac{2}{\beta} \cdot I(X)} \qquad (3.11)$$
$$k = V_{DD} - V_T$$

where $V_T$, $\beta$, and $V_{DD}$ are the threshold voltage of MP3,the gain constant of MP3 and the supply voltage of the comparator, respectively.

T. Ike et. al. has proposed a full adder architecture and built a 54x54 signed-digit multiplier based on the source coupled multiple valued logic. The designed radix-2 signed-digit full adder architecture is shown in Figure 3.9.

| Logic value | -2 | -1 | 0 | 1 | 2 |
|---|---|---|---|---|---|
| X(i),Y(i),W(i),C(i),S(i) | - | 0 | $I_0$ | $2I_0$ | - |
| Z(i) | 0 | $I_0$ | $2I_0$ | $3I_0$ | $4I_0$ |

($I_0$:Unit current)

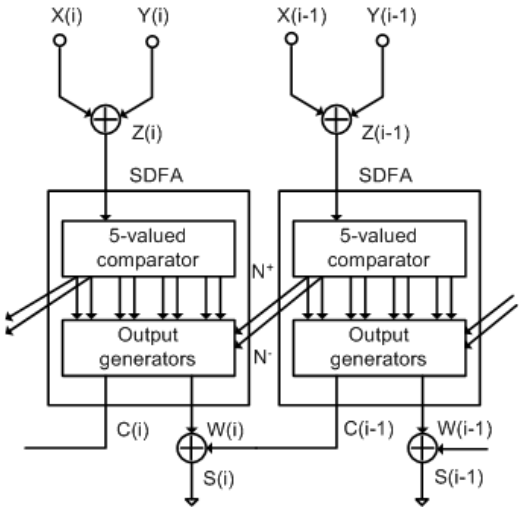



Figure 3.9. Signed digit full adder architecture

In this signed digit full adder architecture, *X(i)* and *Y(i)* are input currents which are added to produce *Z(i)*. *C(i)* and *W(i)* correspond to the carry and interim sum, respectively. Final sum value is *S(i)*. The signed digit addition implemented here can be formulized below:

$$
\begin{aligned}
&Z(i) = X(i) + Y(i) \\
&W(i) = Z(i) - 2C(i) \text{ or: } 2C(i)+W(i) = Z(i) \\
&S(i) = W(i) + C(i-1)
\end{aligned}
\tag{3.12}
$$

To retain the final sum $S(i)$ within the set {-1, 0, 1}, $C(i)$ and $W(i)$ are determined by $Z(i-1)$ together with $Z(i)$ as follows [8]:

$$
\begin{aligned}
&C(i) = 1,\ W(i) = 0 \quad \text{if } Z(i) = 2 \\
&C(i) = 1,\ W(i) = -1 \quad \text{if } Z(i) = 1 \text{ and } Z(i-1) \geq 1 \\
&C(i) = 0,\ W(i) = 1 \quad \text{if } Z(i) = 1 \text{ and } Z(i-1) < 1 \\
&C(i) = 0,\ W(i) = 0 \quad \text{if } Z(i) = 0 \\
&C(i) = 0,\ W(i) = -1 \quad \text{if } Z(i) = -1 \text{ and } Z(i-1) \geq 1 \\
&C(i) = -1,\ W(i) = 1 \quad \text{if } Z(i) = -1 \text{ and } Z(i-1) < 1 \\
&C(i) = -1,\ W(i) = 0 \quad \text{if } Z(i) = -2.
\end{aligned}
\tag{3.13}
$$

The equations above can be verified using Table 2.1.

Intermediate sum and carry functions can be implemented in source coupled multi-valued logic as shown in Figure 3.10. In this scheme, the variable pair (N+ and N-) represents the condition that $Z(i-1) \geq 1$ and $Z(i-1) < 1$.

### 3.3.3. High Radix Current Mode Full Adder Architecture

The addition operation is implemented by truncated difference block [32]. Truncated difference operation is defined as [50]:

$$
x \Xi ky = \begin{cases} x - ky & \text{iff } x \geq ky \\ 0 & \text{otherwise} \end{cases}
\tag{3.14}
$$

The multi-valued min operator can be defined using truncated difference operation:

$$
\min(x,y) = x \Xi (x \Xi y) = y \Xi (x \Xi y)
\tag{3.15}
$$

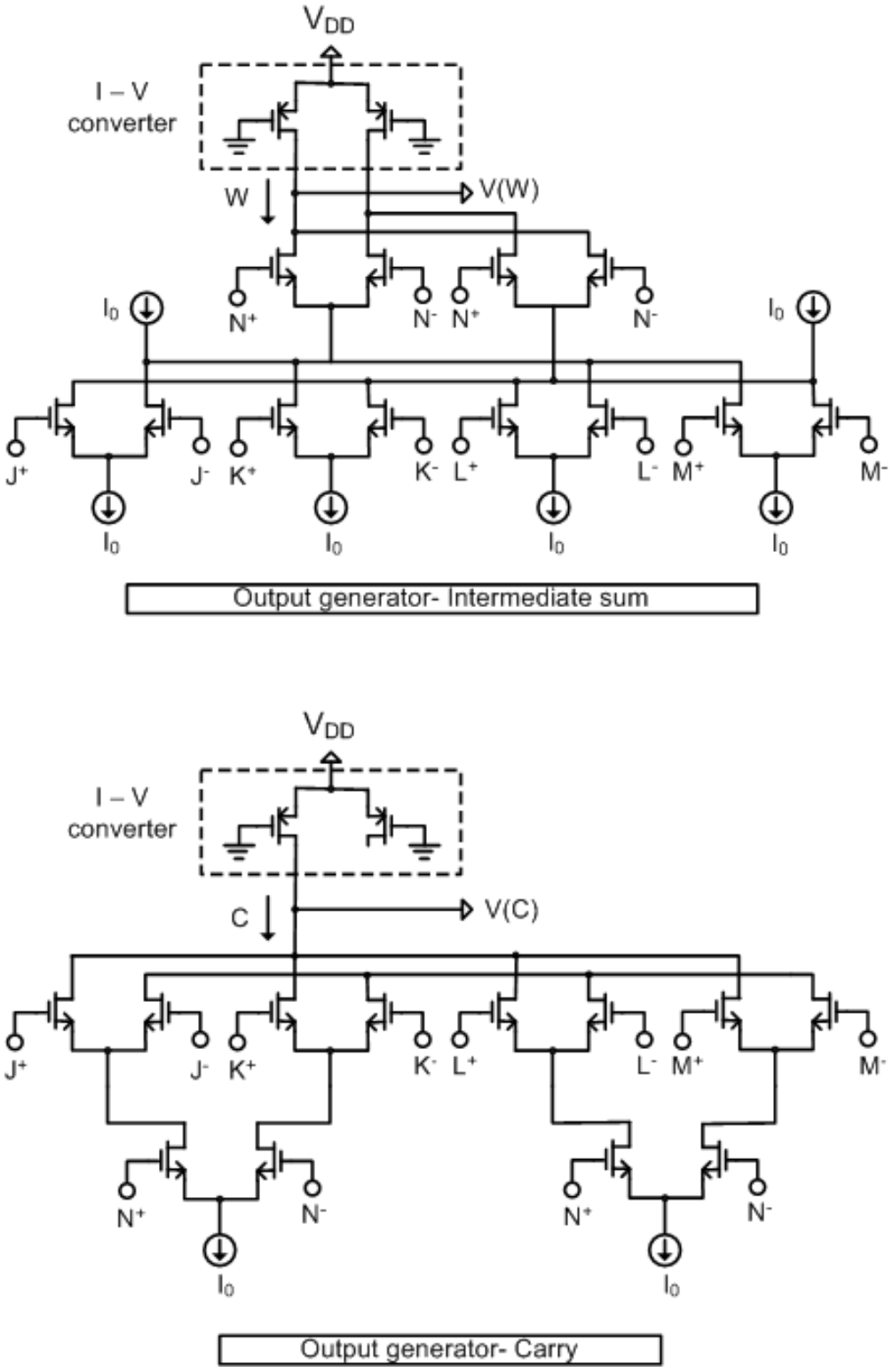



Figure 3.10. Carry and sum generation for the circuit proposed in [8]

The truncated difference operation can be realized using the circuit shown in Figure 3.11. By using the truncated difference operator, current-mode threshold operations can be realized.

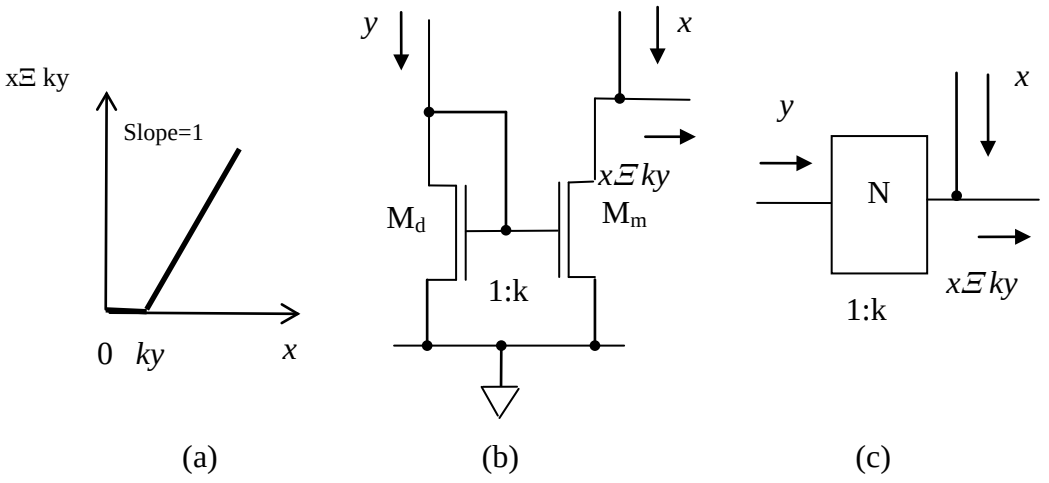
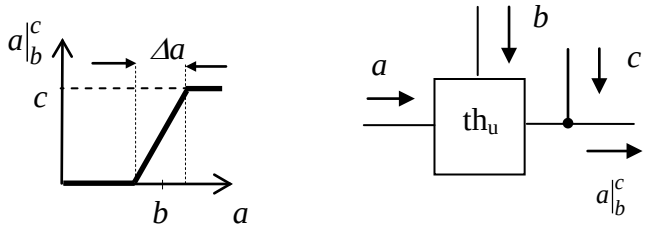



Figure 3.11. Truncated difference: DC characteristics, circuit and symbol [48]

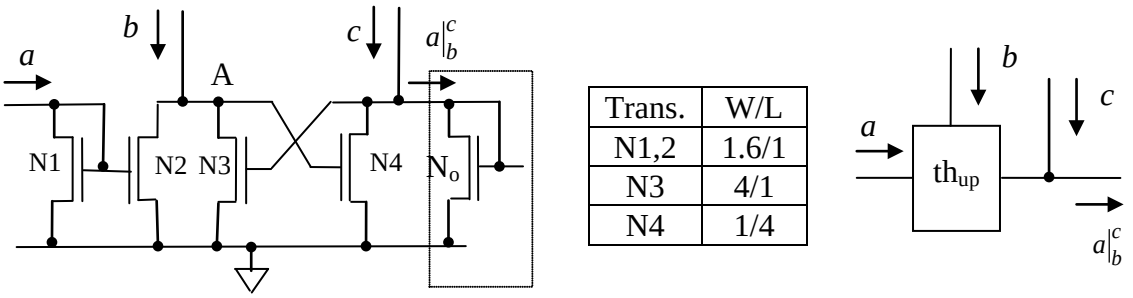
The upper and lower threshold operations for $a, b, c, d \in \mathrm{R}$ are defined as:

$$\text{Upper-threshold, } th_u: \quad a\Big|_b^c = \begin{cases} c & \text{if } a \geq b \\ 0 & \text{otherwise} \end{cases}$$
$$\text{Lower-threshold, } th_l: \quad {}_b^c\Big|a = \begin{cases} c & \text{if } a \leq b \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

A robust upper threshold design based on positive feedback is shown in Figure 3.12 [32, 51]. This type of upper threshold circuit will be denoted by $th_{up}$ (where 'p' denoting the positive feedback). The quantity Δa is the total transition width and it is adjusted by the aspect ratios of cross-connected transistors N3 and N4. The given aspect ratios of the transistors ensure Δa = 2μA from 0 through 75μA. Figure 3.12 (b) shows the complete circuit diagram. The inputs *a* and *b* are compared by the truncated difference circuit that consists of N1 and N2.




(a)




| Trans. | W/L |
|---|---|
| N1,2 | 1.6/1 |
| N3 | 4/1 |
| N4 | 1/4 |

(b)

Figure 3.12 (a) Upper-threshold DC characteristics and two-output block diagram; (b) circuit diagram, minimum feature size transistor aspect ratios

N2 is "on" if $a>b$ forcing the gate of N4 to a low voltage and turn it "off". So the output is equal to $c$. Otherwise the node "A" is high which drives transistor N4 to turn "on", hence setting the output to "0". The positive feedback between transistors N3 and N4 ensures the bi-stable operation (i.e. one of the transistor is "on" while the other is "off"). It should also be noted that the topology can be configured as multi-output through the output transistor, $N_o$. Lower-threshold operation is achieved by interchanging inputs $a$ and $b$ in upper threshold configuration.

Cyclic gates can be implemented by using truncated difference and upper threshold circuits. They have equivalent functionality to the modular summing circuit. The cyclic operation is defined as:

$$x^{k\rightarrow} = (x + y) \bmod r : (x + y)\,\Xi(x + y)\,|_{r'}^{r} \tag{3.17}$$

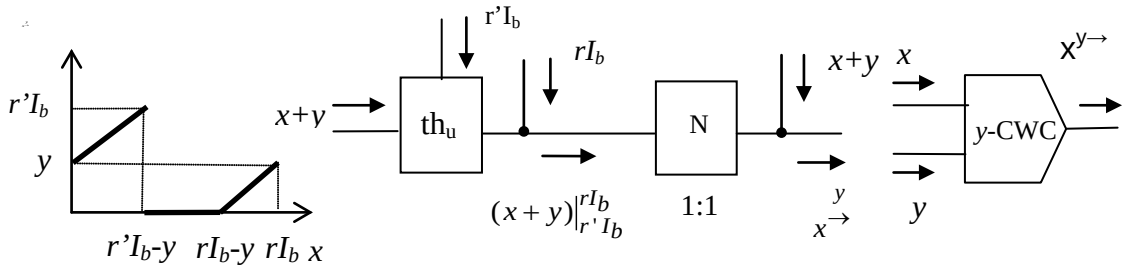
By using the equation above, the circuit can be realized as seen in Fig 3.13. Here, is $I_b$ is equivalent to the lowest logic value, '1', $r$' is equal to ($r$ – 0.5). The value of 0.5 represents

here the noise margin in the circuit. With this operation, the high radix full adder circuit can be implemented. The equation of the full adder circuit is given below:

$$S = x^{y+C_{in}\rightarrow} = \begin{cases} x + y + C_{in} & \text{if } x + y + C_{in} < r' \\ x + y + C_{in} - r & \text{otherwise} \end{cases} \tag{3.18}$$

$$C = \begin{cases} 0 & \text{iff } x + y + C_{in} < r' \\ 1 & \text{otherwise} \end{cases} \tag{3.19}$$

The summing circuit is shown in Figure 3.14.




(a) (b) (c)

Figure 3.13. (a) y-CWC operation DC characteristics, (b) block diagram, (c) circuit diagram

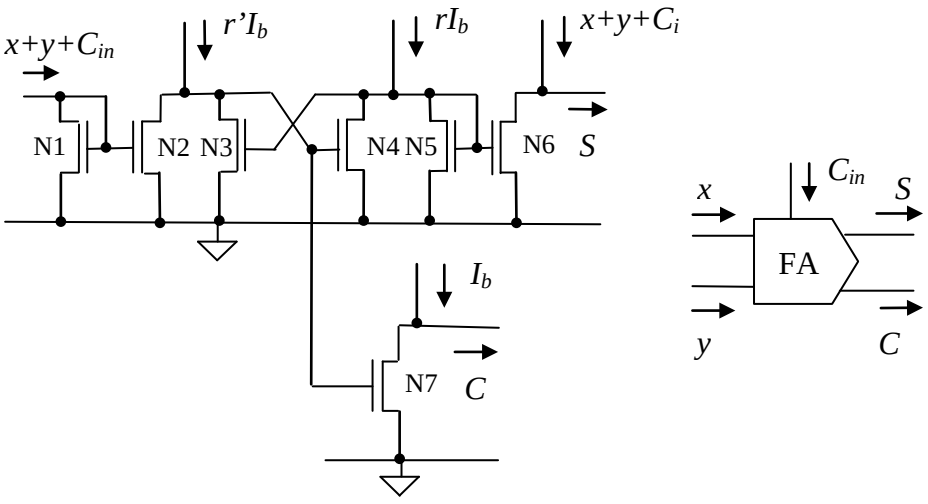



Figure 3.14. A high radix current mode full adder circuit [32]

### 3.3.4. Comments on the Multi-valued Arithmetic Architectures

Throughout the sub-sections of this chapter some of the interesting multi-valued applications are analyzed. The first multi-valued arithmetic circuit [49] explained in Section 3.3.1 is has a very basic style of current switching. The circuit suffers from slow switching characteristics based on the current comparator scheme implemented where the performance metric of a current comparator has been given in Equations (3.1) through (3.4).

The next multi-valued application [8] explained in Section 3.3.2 has novel structure that current comparison has been made through source-coupled pairs through current-voltage converters. The circuit has an improved performance as well as robust switching characteristics based on differential design style employed. However, binary signed-digit implementation is complex and many binary source-coupled logic blocks are used for resolving the output signals.

The last design [32] explained in Section 3.3.3 has advantageous features and has been defined using formal multi-valued logic definitions. The advantage of the system is that the circuit is designed to work for high radix systems, such as radix-8 as given in the example. However, high radix also means lower noise margins and the circuit application areas may be restricted.

An important work of A. K. Jain, R. J. Bolton, M. H. Abd-El-Barr [30, 31], should be mentioned here for a thorough analysis of MVL and arithmetic circuits, as well as, for better insight in MVL design principles.

It is not possible to make an in-depth analysis for all multi-valued applications. Therefore, only some of the pioneering works for the arithmetic applications are summarized here. The general problems in current-mode multi-valued logic are constant current consumption of the circuits, complicated nature of the design style, slow switching of the current comparators, lower noise margins and robustness issues of the circuits. These are the difficulties for researchers who are interested in designing multi-valued circuits.

In the next chapter, various novel multi-valued arithmetic circuits are proposed, which is the main contribution of the first part of the thesis. Multi-operand addition schemes where advantages of multi-valued circuits can be explored are focused in the proposed systems.

# 4. MULTI-VALUED SIGNED DIGIT CIRCUITS AND COUNTER CIRCUITS

There are numerous examples of multi-valued arithmetic implementations in the literature [5, 7-9, 49]. Most of these blocks are based on two-operand addition. Besides, there are very successful two operand addition schemes in binary logic based on either redundant arithmetic techniques [15, 21] or parallel prefix addition based binary structures [52-55]. Many of these high performance binary arithmetic blocks are analyzed together in [56] and [2]. However, there is almost no multi-valued and multi-operand addition scheme existing in the literature. Multi-valued and multi-operand circuits can be used to reduce partial products in multiplier circuits as well as other multi-operand addition applications such as fixed coefficient multiplication or similar applications. They provide small-size arithmetic blocks for multiplication and similar operations.

When exploring the multi-valued arithmetic circuits in the literature, the addition algorithms in [32, 48, 49] utilize single transistor switching schemes which results in glitches in the current comparison operations. In [8], differential design technique is used, which provides better performance with less glitch and increased noise immunity. The design style proposed here is based on differential design techniques. Throughout this chapter, multi-valued multi-operand addition schemes are presented. First, a multi-operand addition scheme is presented employing a signed digit scheme for the addition of normal binary operands, resulting with signed digit outputs. The design is suitable for adding multiple operands in average sized multiplier circuits.

After the signed-digit implementation proposed in Section 4.1, multi-operand addition circuits are proposed in Section 4.2, similar to the binary circuit (7, 3) counter. Two multi-operand addition circuits are proposed in Section 4.2.1 and 4.2.2. The proposed circuit in 4.2.2 has an advantage that the output is self restored to binary. It can be used as an equivalent to the binary (7, 3) circuit. The proposed unique design style for multi-operand addition here provides analog-friendly arithmetic design alternatives in circuit design.

## 4.1. Signed-Digit Multi-Valued Implementation

Signed-digit (SD) number systems provide alternative redundant representations of numbers that can be exploited for carry-propagation free addition algorithms [13]. In general, SD systems are defined for two operand additions, and many of the multi-valued logic implementations are realized for two operand additions. However, multi-operand addition is important especially for the multipliers, where multiple input operands need to be added up fast for efficient arithmetic operation. Here, a novel and efficient multi-operand multi-valued addition scheme is proposed. The proposed system has six normal binary inputs which are added up to form a signed digit output. The six inputs are labeled as $X_0$ ... $X_5$. Each input $X_j$ is composed of $n$ digits in the range of $x_{i,j} \in \{0, 1\}$. The multi-valued sum output is organized to have the value digit set of {−1, 0, 1, 2, 3} at the output. This is the digit set of radix-4 for positive values. Each of the six input operands $X_0$ ... $X_5$ are defined as:

$$X_j = 2^{n-1} x_{n-1,j} + 2^{n-2} x_{n-2,j} + \ldots + 2^1 x_{1,j} + 2^0 x_{0,j}$$
$$X_j = \sum_{i=0}^{n-1} 2^i x_{i,j} \tag{4.1}$$

To retain the value of the sum in the digit set of {−1, 0, 1, 2, 3}, carry-out digit is transferred to two next cell as an input to carry-in. For the parallel addition of each digit of $X_j$, each digit is added up together and the output is processed in the following equation:

$$u_i = \sum_{j=0}^{5} x_{i,j} - r \cdot 2c_{i+1} \tag{4.2}$$

Here, $u_i$ is called as the interim sum. Since each digit is in radix-2, $r$ is 2. The carry-out is transferred to the two next digit, in the application. Therefore, a constant of ‘2’ in the equation is to make radix-4 positive digit-set be compatible to radix-2 system radix. The sum $s_i$ and carry out $c_{i+1}$ ($c_{out}$) can be calculated as:

$$c_{i+1} = \begin{cases} 1 \text{ if } & \sum_{j=0}^{5} x_{i,j} \geq 3 \\ 0 \text{ if } & \sum_{j=0}^{5} x_{i,j} < 3 \end{cases} \tag{4.3}$$

$$s_i = u_i + c_{i-1} \tag{4.4}$$

The output digit set is planned to stay in the digit set {–1, 0, 1, 2, 3}. It is a mixed radix representation where the value of each digit is defined in radix-4 and each weight of the digits are in radix-2. As mentioned above, in this format, $s_i \in \{-1, 0, 1, 2, 3\}$. $S$ is defined as:

$$S = \sum_{i=0}^{n+1} 2^i s_i \tag{4.5}$$

The arithmetic operation of the signed digit multi-operand adder is illustrated in Figure 4.1. As the decimal lines added up together, the result is:

$$31 + 30 + 30 + 26 + 24 + 24 = 165.$$

The multiple valued addition result is:

$$1x2^0 + 0x2^1 + (-1)x2^2 + 3x2^3 + 3x2^4 + 1x2^5 + 1x2^6$$

$$= 1 – 4 + 24 + 48 + 32 + 64 = 165.$$

The operation of the multi-operand adder is verified.

This new multi-operand adder circuit is designed to have an SD arithmetical structure that allows for carry propagation free arithmetic. The proposed multi-operand adder is based on multi-valued logic design techniques and current-mode addition principles are applied for the addition of the currents at each step.

| | | | | Binary inputs | | | | | Decimal Equivalent |
|---|---|---|---|---|---|---|---|---|---|
| $X_0$ | | | | 1 | 1 | 1 | 1 | 1 | 31 |
| $X_1$ | | | | 1 | 1 | 1 | 1 | 0 | 30 |
| $X_2$ | | | | 1 | 1 | 1 | 1 | 0 | 30 |
| $X_3$ | | | | 1 | 1 | 0 | 1 | 0 | 26 |
| $X_4$ | | | | 1 | 1 | 0 | 0 | 0 | 24 |
| $X_5$ | | | + | 1 | 1 | 0 | 0 | 0 | 24 |
| Input Block | | | | 6 | 6 | 3 | 4 | 1 | |
| Interim Sum | | | | 2 | 2 | -1 | 0 | 1 | |
| Carry | + | 1 | 1 | 1 | 1 | | | | |
| Result: | | 1 | 1 | 3 | 3 | -1 | 0 | 1 | |

Figure 4.1. Signed digit multi-operand adder example

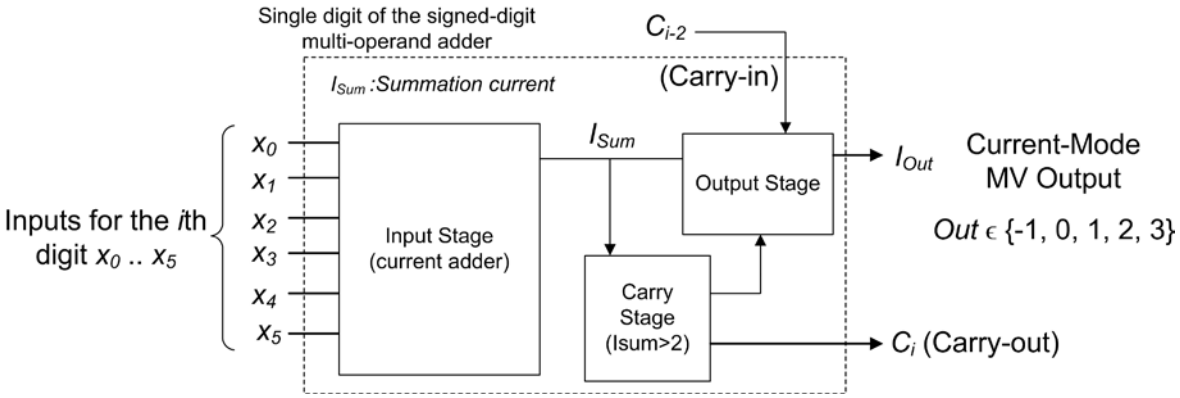



Figure 4.2. Block diagram of the signed-digit multi-operand adder

Next to the arithmetic definitions, multi-valued circuit implementation should be described. Figure 4.2 shows the block diagram of a single digit of the system. The each digit of the circuit is composed as an input stage, carry generation stage and output stage. Figure 4.3 shows the input block of the adder structure. Here, $x_0 \ldots x_5$ are a bit slice of six numbers of the binary inputs of the circuit. It should be noted that, here, we are considering a single digit addition of $X_0 \ldots X_5$ for the input block of $X_j$, so that $x_{i,j}$ is named as $x_j$ (a vertical bit slice) for simplicity in the circuit diagram. The vertical bit-slice can be seen in Fig 4.1 as each of the six input operands is circled. The voltage mode binary inputs are converted to current values with current sources and the summation of the input currents are accumulated over diode connected transistors. The diode connected transistors $M_{in1}$ and $M_{in2}$ conduct $I_{sum}$ and $I_{sum}$' currents that are complements of each other. Here, the circuit operates in fully differential mode. The unit current $I_0$ is selected to be 5 µA. A bias current

of 5 µA is also supplied to the input circuit for better operation over a large dynamic range. The corresponding current values for the summation are listed in Table 4.1. The logical values of ‘4’, ‘5’ and ‘6’ are not defined in radix-4 system, nevertheless the summation values appear in the $I_{sum}$ node which will be subtracted in the next level. As can be seen from Table 4.1, $I_{sum}$ values increase by 5 µA steps whereas $I_{sum}$’ decrease by 5 µA steps when the logic level increases.

Since the number set being used for each digit is in the range of {−1, 0, 1, 2, 3} and six binary inputs are added up to have an initial sum of 6, a carry-out must be generated to make the number system operational when the input value reaches ‘3’. Since SD representation is used for the system, the output should be defined in the set of {−1, 0, 1, 2} so that the carry-in signal from a previous stage should not generate an extra carry-out [2]. It should be noted here that, the carry-in is always positive in this addition algorithm.

Table 4.1. Logic levels of $I_{sum}$ and $I_{sum}$’ at the input stage

| **Current Levels (µA)** | **Logic Levels** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| $I_{sum}$ | 10 | 15 | 20 | 25 | 30 | 35 | 40 |
| $I_{sum}$’ | 35 | 30 | 25 | 20 | 15 | 10 | 5 |

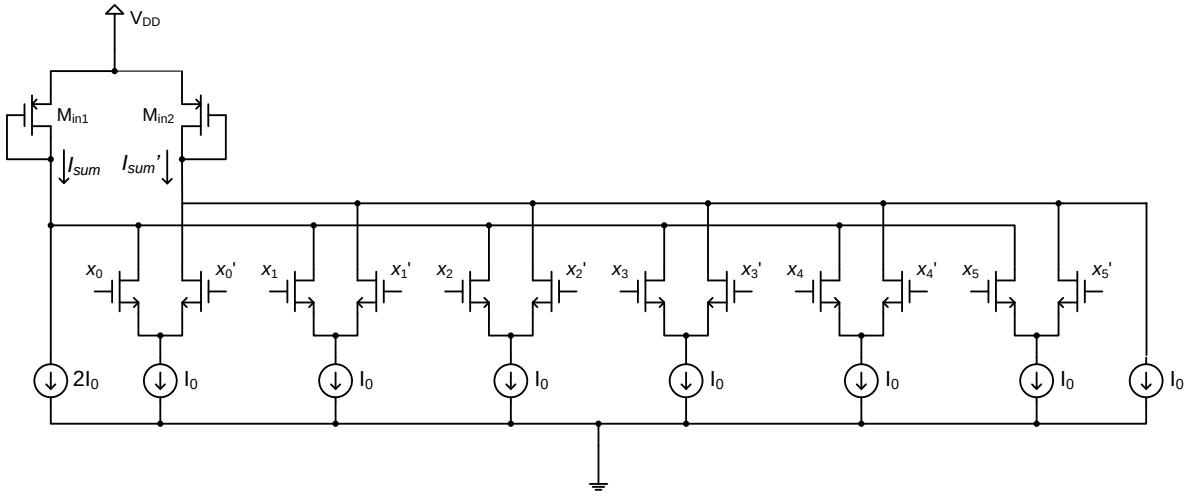



Figure 4.3. Input block of the signed digit multi-operand adder

Whenever the $I_{sum}$ current is equal or greater than logical '3', then the carry-out is generated in the system. The comparison is made by complementary pairs that is in the decision boundary where $I_{sum}$ increases from 20 µA to 25 µA and at the same time $I_{sum}$' decreases from 25 µA to 20 µA. The cross-over of the current values can be sensed by a current comparator to sense this transition. Since the input transistors $M_{in1}$ and $M_{in2}$ in Figure 4.3 are diode connected, their input resistances are equivalent to [57]:

$$r_i = \frac{1}{g_m + g_{ds}} \cong \frac{1}{g_m} = \frac{1}{\sqrt{2\mu_n C_{ox}\left(\frac{W}{L}\right)_{Min1,2} I_D}} \tag{4.6}$$

Here, $g_m$ is the gate transconductance, $g_{ds}$ is the drain transconductance, which is smaller compared to $g_m$. $\mu_n$ is the mobility of n-type transistor. $C_{ox}$ is the gate oxide capacitance. $W$ and $L$ are the width and length of the corresponding transistor. The voltages generated over the source to drain of the input transistors are:

$$V_{SD(M_{in1})} = r_i I_{sum}$$

$$= \frac{I_{sum}}{\sqrt{2\mu_n C_{ox}\left(\frac{W}{L}\right)_{Min1} I_{sum}}} = \sqrt{\frac{I_{sum}}{2\mu_n C_{ox}\left(\frac{W}{L}\right)_{Min1}}} \tag{4.7}$$

Similarly,

$$V_{SD(M_{in2})} = \sqrt{\frac{I_{sum}'}{2\mu_n C_{ox}\left(\frac{W}{L}\right)_{Min2}}}$$

Shortly, the voltage drop over the $M_{in1}$ and $M_{in2}$ can be detected and compared to the voltage difference where the cross-over occurs whenever the interim addition result exceeds logical value of '3'. These voltage levels can also be observed by HSPICE

simulations shown in Figure 4.4. For each current step, each voltage step for the source to drain voltages is approximately 50mV. Both of the diode connected transistors accumulate input currents and copy them to other stages through current mirrors. At the same time, the voltage generated over the drain nodes of these transistors are fed into the differential amplifier for sensing the carry-out condition. When this transition is fed into the comparator, the currents are converted to voltages through diode connected transistors, i.e. $M_{in1}$ and $M_{in2}$. At the transition point, comparator senses the voltage difference and generates differential carry-out signal.

Figure 4.4 shows the variation of currents and the voltage generated over sources nodes of the input transistors $M_{in1}$ and $M_{in2}$. In the figure, logic inputs are swept from '0' to '5' (i.e. $x_0$ to $x_5$ are activated one after another) and corresponding changes of $I_{sum}$, and $I_{sum}$' as 10 µA to 40 µA and 35 µA to 5 µA respectively are simulated. The differential currents are useful for the protection of the logic levels since each logic level is represented by two differential currents.

Figure 4.5 shows the carry-out circuit which is actually a differential comparator. $M_{CL1}$ and $M_{CL2}$ transistors are used to clamp the output voltages of the carry-out and carry-out' in certain levels. The cross-connected load pair in the differential stage boosts the gain for carry generation [58]. The cross-connected load transistors provide positive feed-back in the structure and provides an adjustable gain. The load transistors have symmetrical structure as shown in Figure 4.5. The gain of the differential pair is:

$$A_d = \sqrt{\frac{\mu_n \left(\frac{W}{L}\right)_{MD1}}{\mu_p \left(\frac{W}{L}\right)_{ML1}}} \frac{1}{1-\alpha} \tag{4.8}$$

$$\alpha = \frac{\left(\frac{W}{L}\right)_{ML2}}{\left(\frac{W}{L}\right)_{ML1}} \tag{4.9}$$

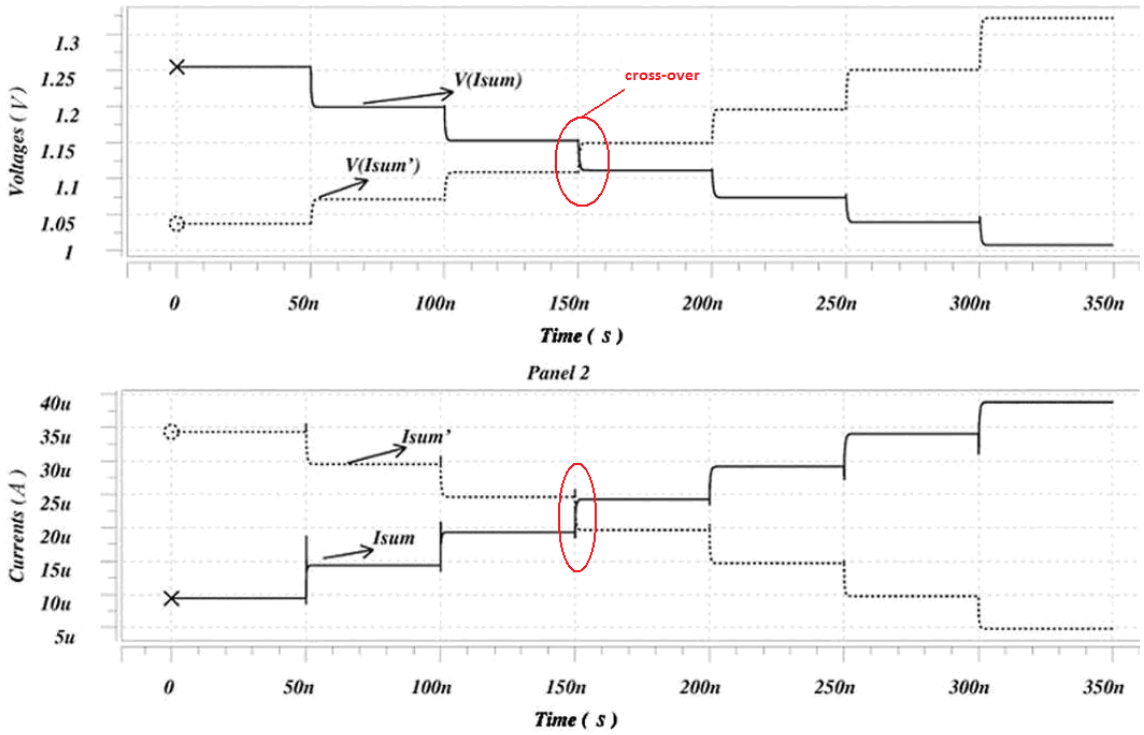



Figure 4.4. The input currents $I_{sum}$ and $I_{sum}$' and generated voltages over the source to drain of the input transistors $M_{in1}$ and $M_{in2}$

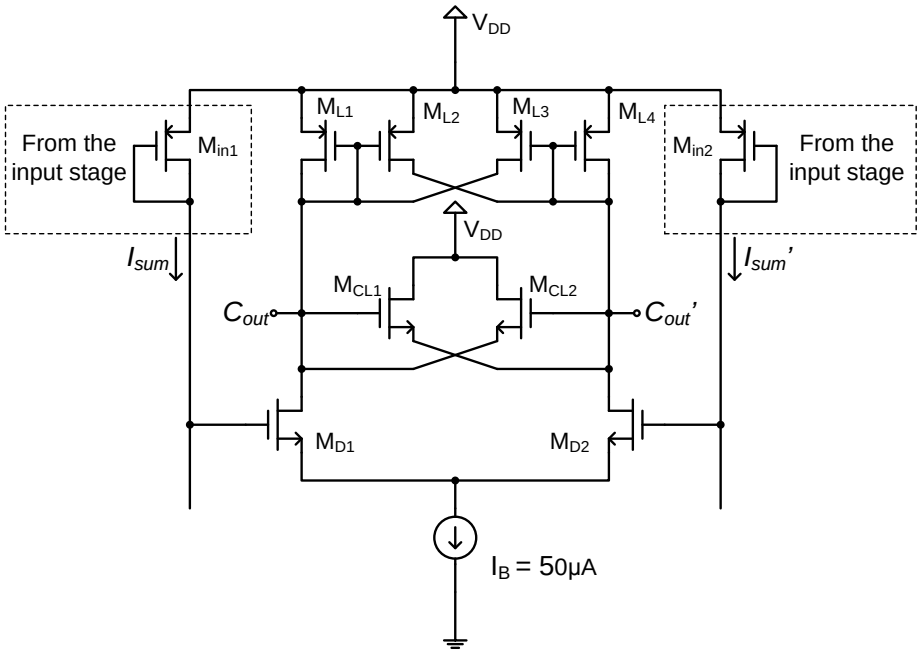



Figure 4.5. Voltage-mode carry generation circuit

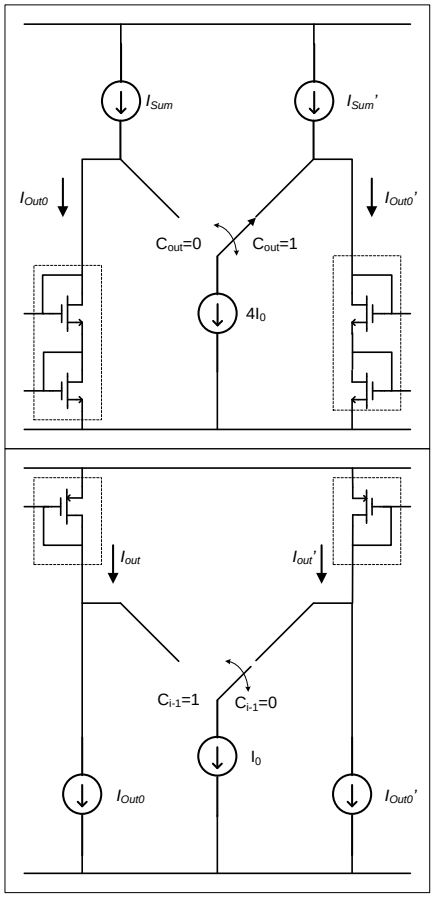



Figure 4.6. Generation of output currents

Since the comparator is used as a switch for subtraction, the gain should be high in order to sense small voltage difference and guarantee proper operation. $\alpha$ should be set to a value below unity such as 0.8 which boosts gain by a factor of 5. The total differential gain should be large, such as 20, to guarantee the generation of a sufficiently large voltage for carry-out. When the carry is generated, logic '4' is subtracted from the $I_{sum}$ value and transferred to output as $I_{out0}$. In Figure 4.6, $I_{sum}$ and $I_{sum}$' values are copied from the input block. Then, regarding to (4.2) i.e. $2r = 4$, $4I_0$ is subtracted from $I_{sum}$ depending on the carry-out condition. The $4I_0$ value, which is 4x5 = 20 μA here is subtracted from $I_{sum}$ or $I_{sum}$' regarding to carry-out signal for complementary operation. There is still a free slot for carry-in input. The carry-in is fed into the circuit from two previous stages, and added up to $I_{out0}$. The end result appears as $I_{out}$ and $I_{out}$' that are again complementary outputs. In Figure 4.6, the load transistors are a combination of two diode connected transistors in series. It provides cascode current mirror with precise current copying property that

maintains more precise current values. For exact subtraction of $4I_0$ value from the input stage, the gain of the comparator circuit must be high so that the $4I_0$ current flows only from one side of the differential circuit. The current subtraction is also done by a differential pair where $I_{out0}$ and $I_{out0}$' is generated. The circuit implementation of the output stage is shown in Figure 4.7 (a). As it can be seen in the figure, whenever $4I_0$ is subtracted from $I_{out0}$ at the same time $4I_0$ is added to $I_{out0}$' for complementary operation. In the same circuit, there also exists a carry-in signal which is coming from two previous stages. The current values corresponding to logical values for $I_{out0}$ and $I_{out0}$' are given in Table 4.2. A current mode carry-out signal can easily be generated from the $C_{out}$ signal of Figure 4.5 by using the simple circuit shown in Figure 4.7 (b). The simulation results for $I(C_{out})$ and $I_{out0}$ are shown in Figure 4.8. The complementary output, $I_{out0}$' is not shown in the figure for clarity. The carry-in signal in Figure 4.8 is added up to $I_{out0}$ via a current mode addition circuit and the final result is $I_{out1}$. It is important to point out that, new binary input operands can be added up to the transistor $M_{in3}$ and $M_{in4}$ seen in Figure 4.7. *Three more binary input operands can be added up to the nodes where transistors $M_{in3}$ and $M_{in4}$ are connected*. Then, a second stage carry generation logic must be added to the circuitry. In this scheme, six normal binary adders can be added up together, or alternatively, three more binary inputs can be added up with multi-valued output node.

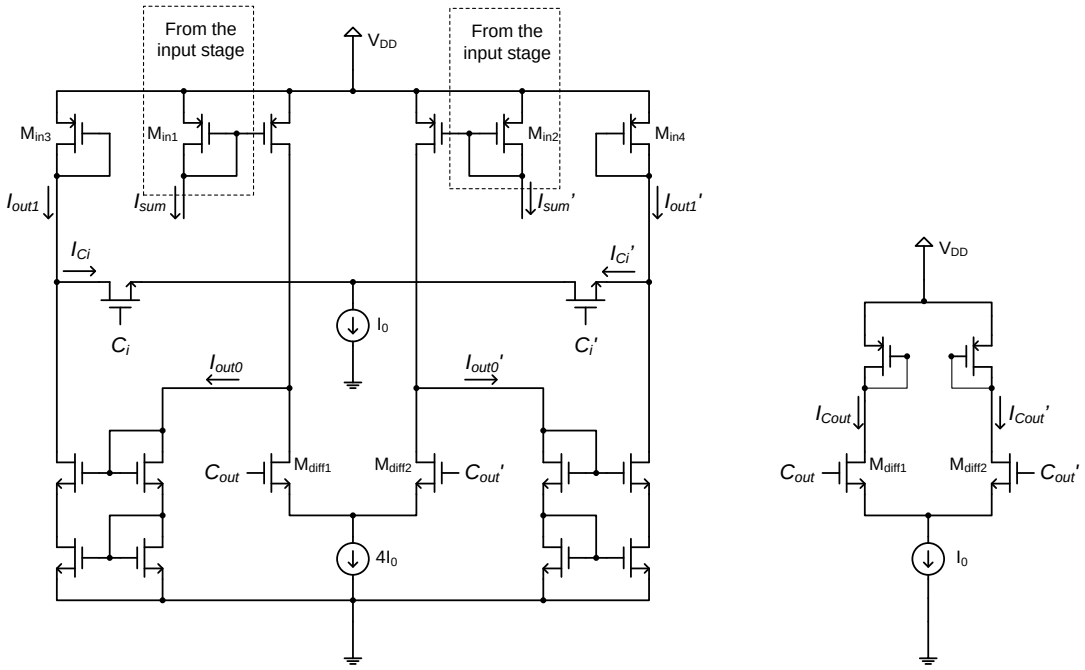



Figure 4.7. (a) Circuit for output current generation; (b) current-mode carry-out generation circuit

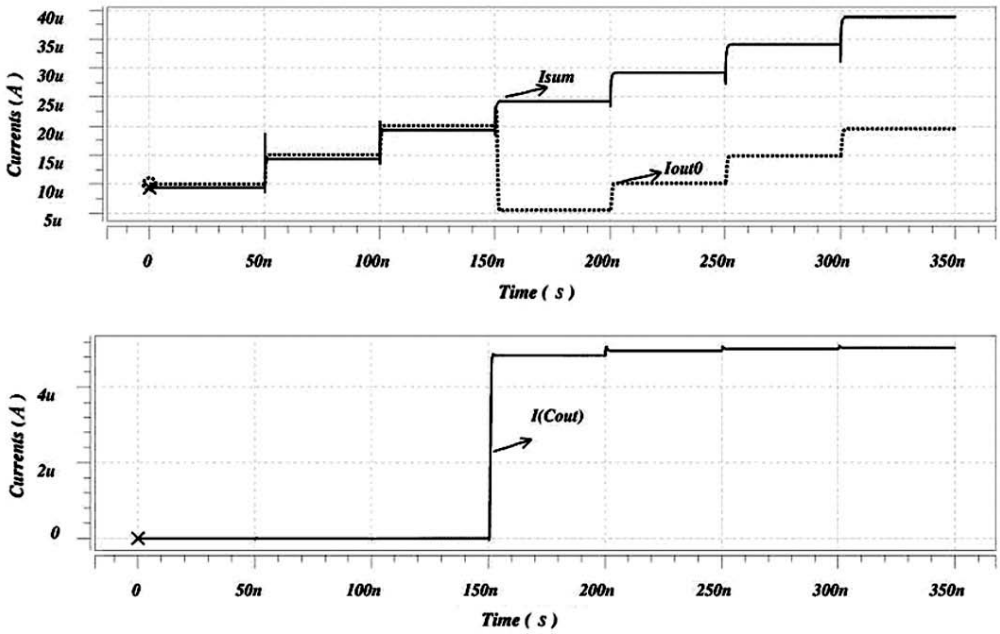



Figure 4.8. Output current and carry-out

Table 4.2. Logic levels of $I_{out1}$ and $I_{out1}$' at the output stage

| **Current Levels (µA)** | **Logic Levels** | | | | |
|---|---|---|---|---|---|
| | **-1** | **0** | **1** | **2** | **3** |
| $I_{out1}$ | 5 | 10 | 15 | 20 | 25 |
| $I_{out1}$' | 25 | 20 | 15 | 10 | 5 |

After the addition process is completed, the result can be converted to binary again in the last stage. It was stated that the input of the system is binary. On the other hand, the output digit set is {−1, 0, 1, 2, 3} for each digit. As a result, each digit has a weight of radix-2 and each output is in the range of $s_i \in \{-1, 0, 1, 2, 3\}$ which causes a mismatch in the arithmetic. This output can be realigned to binary by converting all digits again to binary and adding up even and odd digits by right shifting all the odd digits. In (4.5), the output is defined and each digit can be expressed as a composition of two binary numbers. $s_i$ is denoted as $s_i = s_i^H s_i^L$ where $s_i^H$ and $s_i^L$ are binary digits since a radix-4 number can be represented as two digit radix-2 number. Then S can be rewritten as:

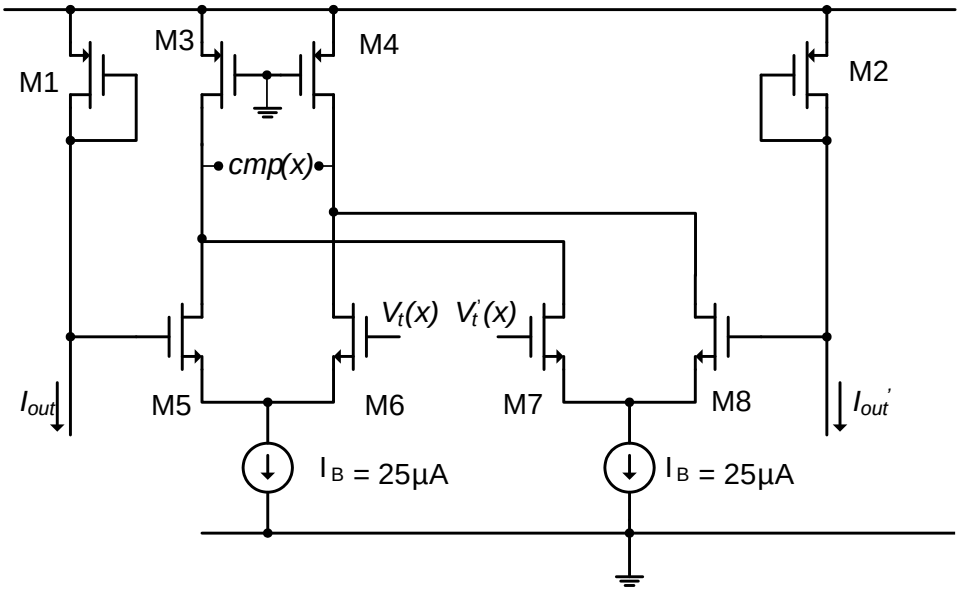
$$S = \sum_{i=0}^{n+1} 2^i s_i = \sum_{i=0}^{n+1} 2^i s_i^L + 2 \cdot \sum_{i=0}^{n+1} 2^i s_i^H \tag{4.10}$$

In this representation, $s_i^L \in \{-1, 0, 1\}$ and $s_i^H \in \{0, 1\}$. It should be noted here that $s_i^L$ can be represented as $s_i^L = s_i^{L+} - s_i^{L-}$, i.e, values of $s_i^L = \{-1, 0, 1\}$ can be denoted as tuples $(s_i^{L+}\ s_i^{L-}) = \{(01), (00), (10)\}$ respectively. The corresponding values of $s_i^H$ and $s_i^L$ for MV data are listed in Table 4.3. The logic for generating the $s_i$ values are:

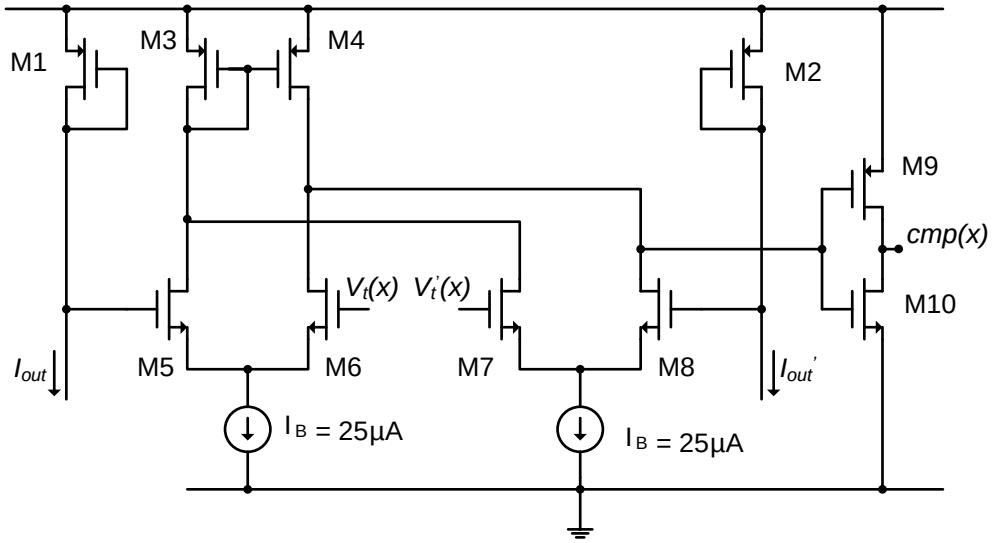
$$\begin{aligned} s_i &= s_i^H s_i^L \\ s_i^{L-} &= \overline{cmp(0)} \\ s_i^{L+} &= cmp(1) \cdot (cmp(3) + \overline{cmp(2)}) \\ s_i^H &= cmp(2) \end{aligned} \tag{4.11}$$

Here, $s_i^{L+}$ can be generated using a single and-or-invert [59] gate by using De-Morgan's law. The corresponding values of $s_i^H$ and $s_i^L$ for MV data comparator outputs can be seen in Table 4.3. The values of $s_i$ are generated from comparator circuits. Four comparator circuits convert MV data to binary comparing the variable digit with logic 0, 1, 2 and 3. The comparator circuits for MV to binary conversion are shown in Figure 4.9. The circuits in Figure 4.9 compare the differential data with differential reference voltages *Vt(x)* and *Vt'(x).*

If the outputs are required in differential source-coupled logic mode, the circuit in Figure 4.9 (a) is suitable. On the other case, if standard CMOS output logic is needed, this time Figure 4.9 (b) of the comparator output is useful. These *Vt(x)* and *Vt'(x)* reference voltages are generated for logic levels of $x$ = 0, 1, 2 and 3. The voltage drop over M1 and M2 transistors that contain the differential summation currents are compared to *Vt(x)* and *Vt'(x)* voltage levels. Four of these comparator blocks are used to resolve the logic value of each summation current. The M1 and M2 transistors can also be biased in triode region by connecting the gate inputs to proper bias voltages. The HSPICE simulation results for the output of the comparator circuits can be depicted in Figure 4.10 which is designed for single ended output configuration.




(a)




(b)

Figure 4.9. MVL to voltage mode binary comparator: (a) Differential ended; (b) Single ended

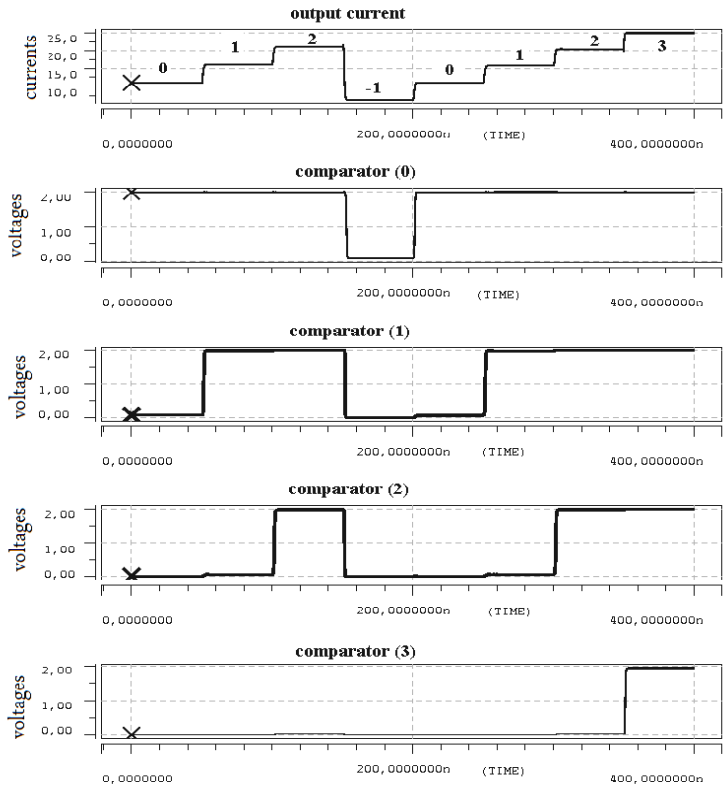



Figure 4.10. Comparator outputs

The addition process defined in Equation 4.11 above can be implemented by a single full adder block without any carry propagation delay. Redundant addition of two numbers, where one is regular, the other is signed digit can be accomplished by a single full adder without any carry propagation [3]. Figure 4.11 shows the addition process to have a signed digit binary output. In the end, the signed digit output can be represented as

$$S = \sum_{i=0}^{n+1} 2^i s_i = \sum_{i=0}^{n+1} 2^i (p_i^+ - p_i^-)$$

$$S = \sum_{i=0}^{n+1} 2^i p_i, \qquad (4.12)$$

$$p_i = (p_i^+ - p_i^-)$$

where each digit of output $p_i \in \{-1, 0, 1\}$.

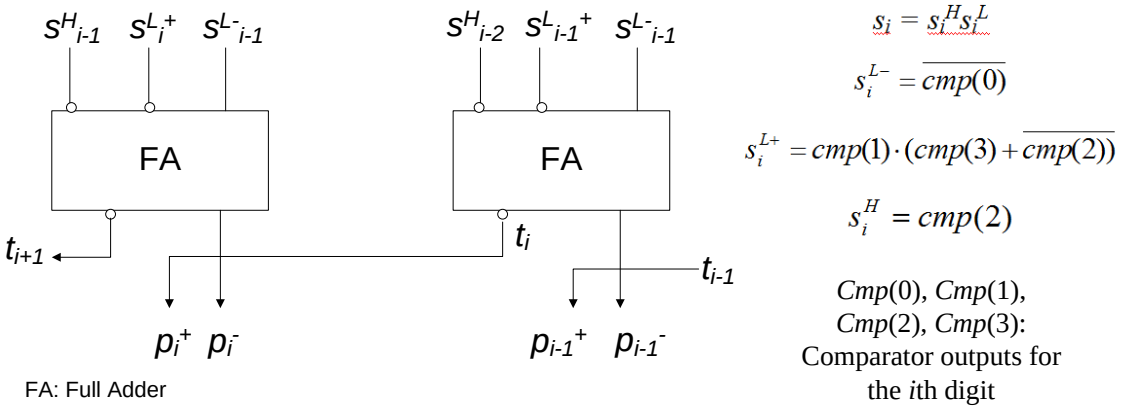



Figure 4.11. Signed digit binary outputs

Table 4.3. Comparator output values and $s_i$

| $s_i$ | *cmp(0)* | *cmp(1)* | *cmp(2)* | *cmp(3)* | $s_i^{L-}$ | $s_i^{L+}$ | $s_i^H$ |
|---|---|---|---|---|---|---|---|
| –1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

The multi-valued circuit proposed here is compared with a conventional binary multi-operand carry propagate adder that is tiled for multi-operand addition with same bit lengths and having six operands. The addition delays of six binary numbers with different bit lengths are simulated and compared with the proposed circuit. The worst-case delay of the proposed multi-operand adder is 3.3 ns acquired from HSPICE simulation results with 2V supply voltage. According to the simulation results, the delays of six-operand adders of various input lengths can be seen in Table 4.4. The input to carry-out delay of a single full adder in conventional CMOS design is 0.28 ns using a 2V supply voltage. The delay increases proportionally to the word-length of the operands.

Single bit slice of the proposed multi-valued multi-operand adder block contains 136 transistors, including the output stage. Six-operand carry-propagate adder has 140 transistors in one bit-slice. Almost same amount of circuit elements are required for the proposed carry-propagation free structure compared to a binary multi-operand adder. The

binary six-operand adder average current consumption is 1.72 mA for 8-bit binary input operands. The result is acquired by applying random inputs to the binary circuit. A snapshot of the current consumption of the binary adder is shown Figure 4.12. For a single bit slice of the circuit, the current consumption is approximately 215µA at the speed of 200 MHz. The current consumption of the proposed multi-valued circuit for one bit slice is approximately 400 µA and mostly constant. The current consumption can be adjusted as desired by changing the current levels. In this configuration, the circuit can work up to 300 MHz. The proposed multi-operand adder circuit is built in fully differential mode, which increases noise immunity of the system. Moreover, the core circuit consumes only static power and switching power is minimal which provides an analog friendly design.

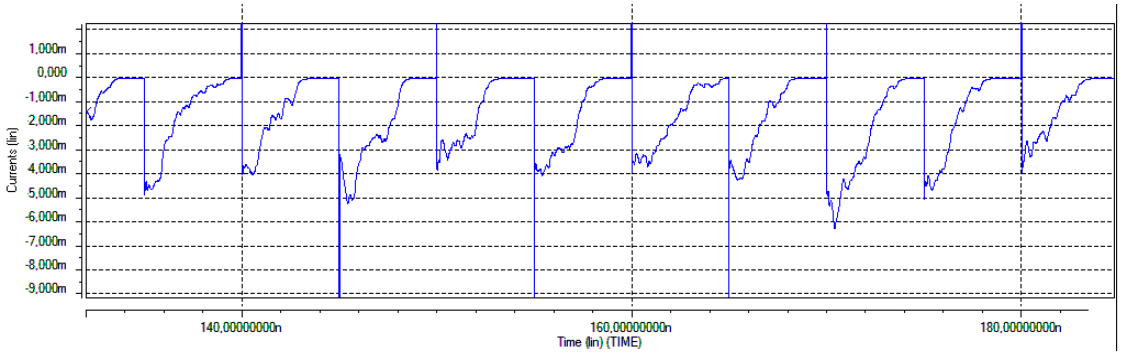



Figure 4.12. Six operand ripple carry adder architecture current consumption

The circuit is especially advantageous for design of constant coefficient FIR filters and similar applications which require multi-operand addition circuits extensively. Figure 4.13 shows the layout for one-bit slice of the adder having area of 30µm × 60µm.

Table 4.4. Delay of the adders for various bit lengths

| Adder type | Delay (ns) | | | |
|---|---|---|---|---|
| | 8-bit | 16-bit | 32-bit | 64-bit |
| Proposed multi-operand redundant adder | 3.3 | 3.3 | 3.3 | 3.3 |
| Conventional multi-operand adder | 3.92 | 6.16 | 10.64 | 19.6 |

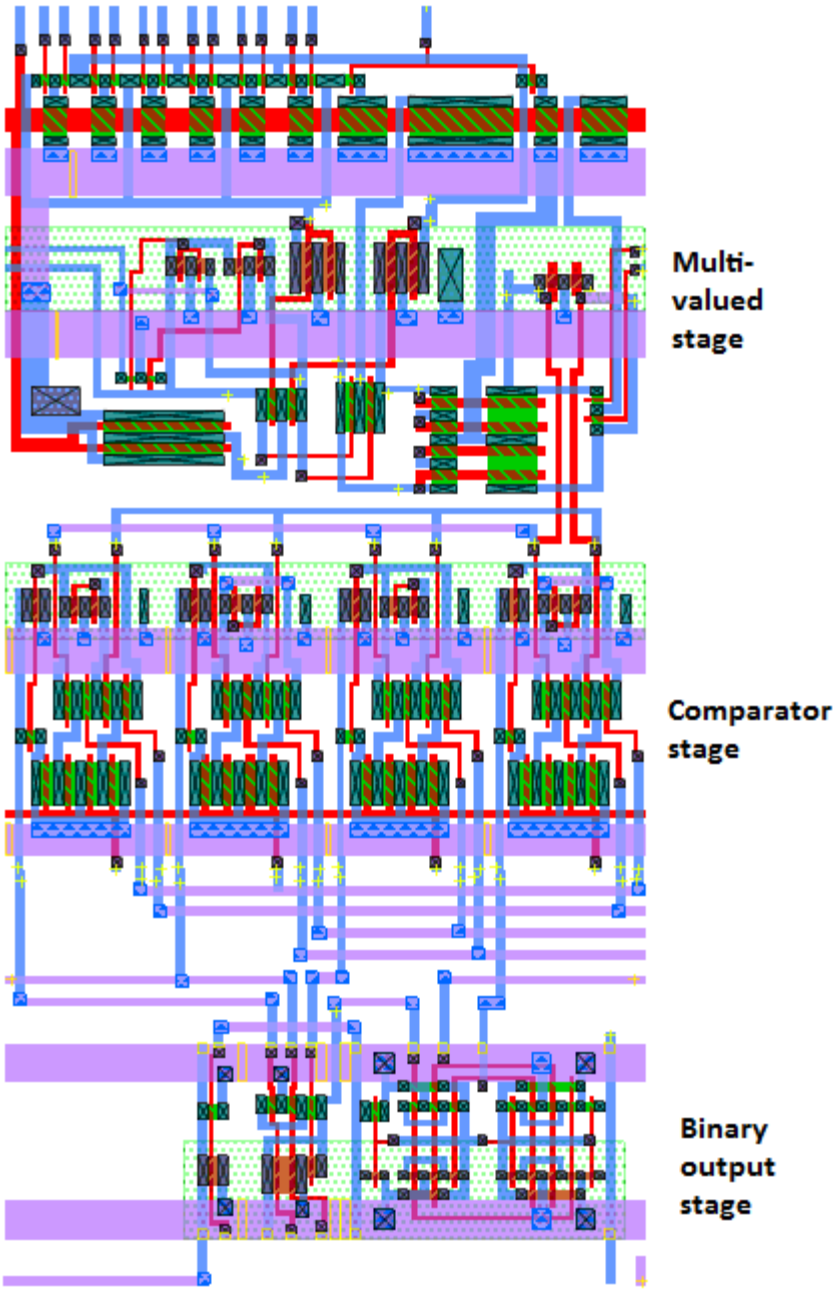



Figure 4.13. Single bit slice of the multi-operand adder layout

## 4.2. Multi-operand Addition and Counter Circuits

The counter circuits are used in arithmetic circuits for multi-operand addition [2, 25]. The simplest counter circuit is a (3, 2) counter, which is equivalent to a full adder. It reduces three input operands to two. The operation of a (3, 2) counter where the input operands are *x, y, z* and outputs are *s* (sum) and *c* (carry) is described as:

$$x + y + z = 2c + s \tag{4.13}$$

Here, '+' is the arithmetic addition operand and each variable is single-bit operands. The (3, 2) counter has an equal functionality with a full adder, where counter circuits are generally used for partial product reduction.

As an example, a three operand four bit redundant adder, namely, a carry-save adder (CSA); is built using four full adders, i.e. one full adder is required for each bit. Figure 4.14 (a) shows the redundant addition scheme where four bit *X*, *Y* and *Z* inputs are added up with the result *S* + *C*. In the figure, the rectangular blocks represent full adders (FA). The redundant structures require a completion adder for getting the final result.

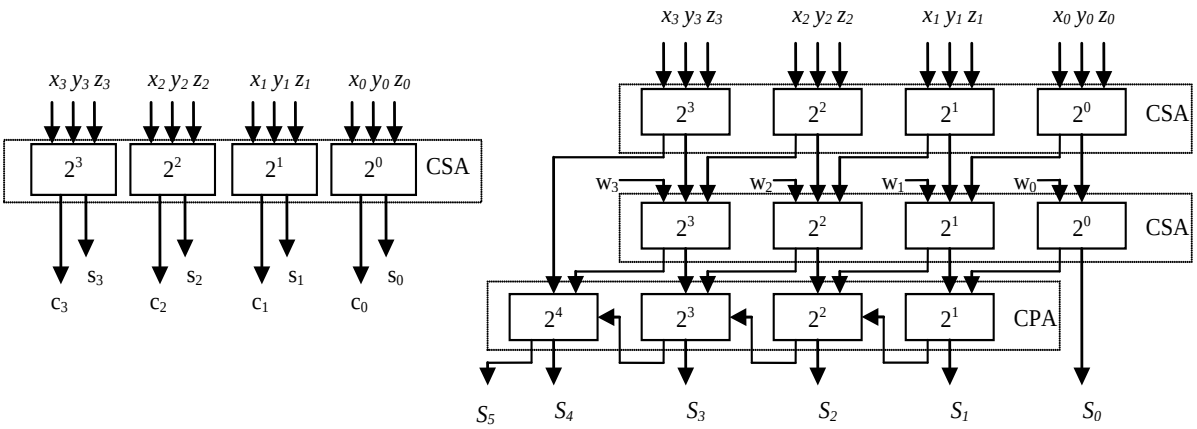



Figure 4.14**.** (a) Three operand carry-save adder; (b) four operand carry-save adder with completion adder

Figure 4.14 (b) shows a four-operand adder with two stage redundant adder (two-stage CSA adder) scheme with a carry propagate adder (CPA) at the third stage [2]. Here, four bit numbers, *X*, *Y*, *Z* and *W* are added up, with a six-bit *S* output. It is possible to build counters for more than three input operands. The mostly used counter circuits are (3, 2), (7, 3) and (15, 4) counters, where first operand implies the number of inputs and the second one is the number of the outputs. Larger counters can be built by combining smaller ones [2, 3]. The (*m*, *k*) counter is defined as [25]:

$$\sum_{j=0}^{k-1} s_j 2^j = \sum_{i=0}^{m-1} x_i \tag{4.14}$$

A sample multiplier configuration is given in Fig. 4.15. The multiplier scheme given here is a sample multiplier configuration for the implementation of counter circuits. The empty dots are filled with '0' as the inputs in the right side of the diagram. Here, the partial products are reduced using (3, 2) counters until the operands to be added are reduced to two. The partial products are grouped in three bits and reduced by (3, 2) counters in Figure 4.15. After that, the operands are added using carry-propagate adders. Here, (3, 2) counters and full adders have equivalent logic function, however, (3, 2) counters are named for partial product reduction, and, full adders are named for carry-propagate addition. The multiplier scheme is only given for the partial product reduction methodology using counters, here optimization of the circuit is not considered. Similar schemes can be generated using various counter circuits, such as (7, 3) counters, (15, 4) counters etc. Further information can be achieved from [2, 3, 25].

In a (7, 3) counter, from (4.14), $k = 3$, $m = 7$; the inputs are $x_0 \ldots x_6$ and the outputs are ($s_2 s_1 s_0$). Figure 4.16 represents construction of a (7, 3) counter using (3, 2) counters. FA notation in the figure represents Full Adder, or (3, 2) counter, which can be used interchangeably. When the linear structure is used, the critical path is four adder delays and whenever the tree structure is used, the critical path delay reduces to three adder delays. They both require four full adders. Both of the structures are functionally equivalent, however the tree structure has better performance in speed. Further discussion is available in [25].

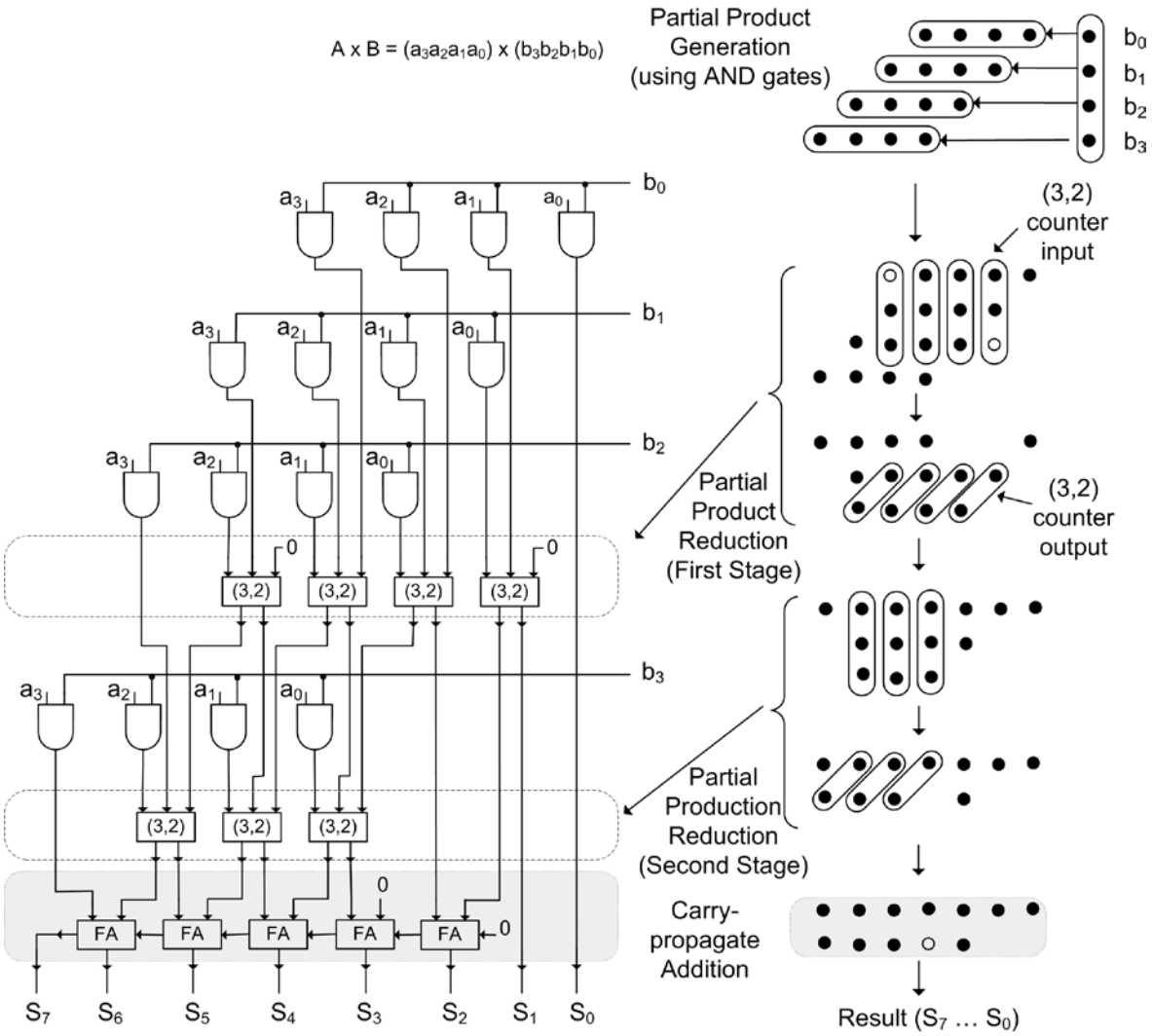



Figure 4.15. Multiplier using (3, 2) counters

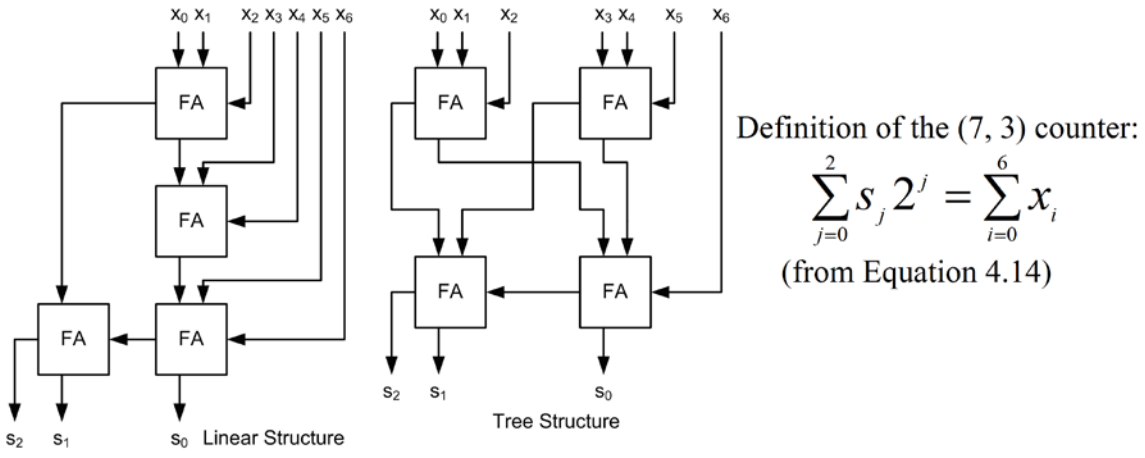



Figure 4.16. Building a (7, 3) counter using full adders

In Figure 4.17, an example of addition of seven operands using a series of (7, 3) counters is shown. Here, in the first stage, each digit is fed into a (7, 3) counter, each digit produces three digit output. The three digit outputs are tiled according to their bit weights. This time they are fed into a (3, 2) counter for reducing the number of outputs to two. If the regular representation of the added number is required, the output in carry save format is fed into a ripple carry adder like in Figure 4.14 (b). In the end, conventional non-redundant addition result is generated. In general, for building carry-save based algorithms, counter and compressor circuits are used. Compressor circuits are a special case of counter circuits, which are built by specially tiling the carry-save operators [25].

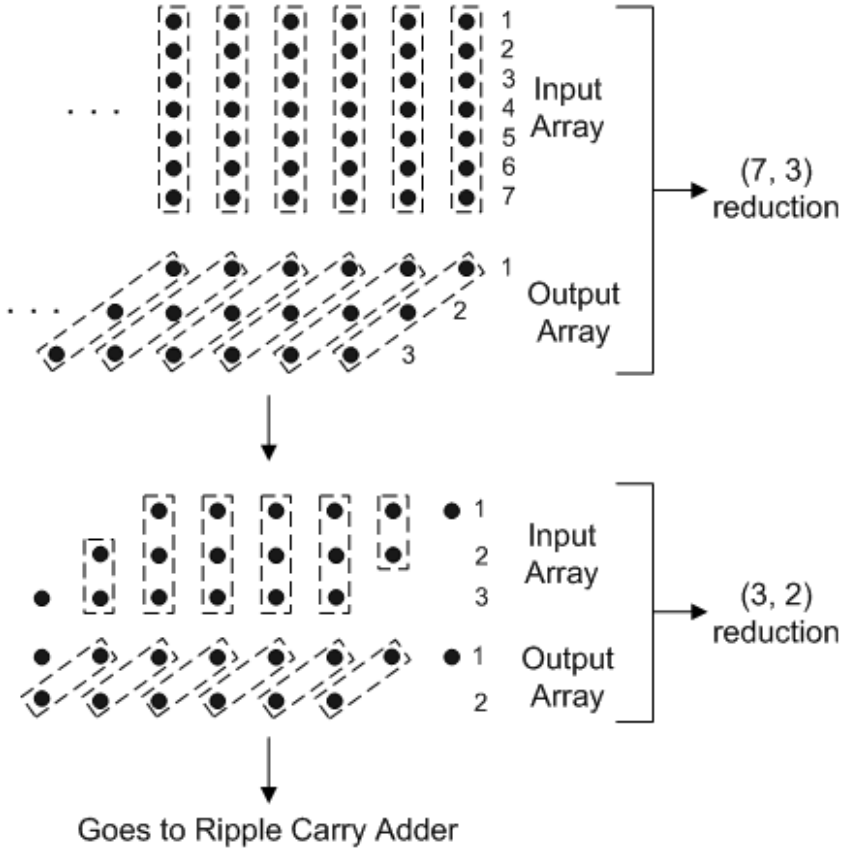



Figure 4.17. Partial product reduction using (7,3) and (3,2) counters

In the next two sub-sections, i.e. Section 4.2.1 and Section 4.2.2, two different implementations of counter circuits are proposed for the multi-operand addition:

- Counter circuit with multi-valued output
- Binary restored multi-valued (7, 3) counter

### 4.2.1. Counter Circuit with Multi-valued Output

In this section, a multi-valued 7-operand addition circuit is constructed. The circuit has binary inputs and multi-valued outputs. The general scheme for the proposed circuit can be depicted in Figure 4.18. The circuit is similar to the binary (7, 3) counter with seven inputs. In a regular binary (7, 3) counter, the output is composed of three bit tuple, i. e., ($s_2s_1s_0$). In the proposed scheme, the *Carry-out* signal corresponds to the third bit ($s_2$) of the regular (7, 3) counter, the first two bits ($s_1s_0$) are encoded as multi-valued signals as shown as *Out* in Figure 4.18. It should be noted that, every signal has its complement in the proposed circuit.

The input block of the circuit is shown in Figure 4.19. HSPICE transient response simulation result of the input block is shown in Figure 4.20. The circuit is implemented using TSMC 0.25 µm technology with a 2V supply voltage. It should be noted that, the performance of the circuit is mainly affected by bias currents rather than supply voltages as long as the load transistors are in saturation region. Table 4.5 shows the addition and corresponding currents passing through the load transistors $M_{in1}$ and $M_{in2}$ of Figure 4.19.

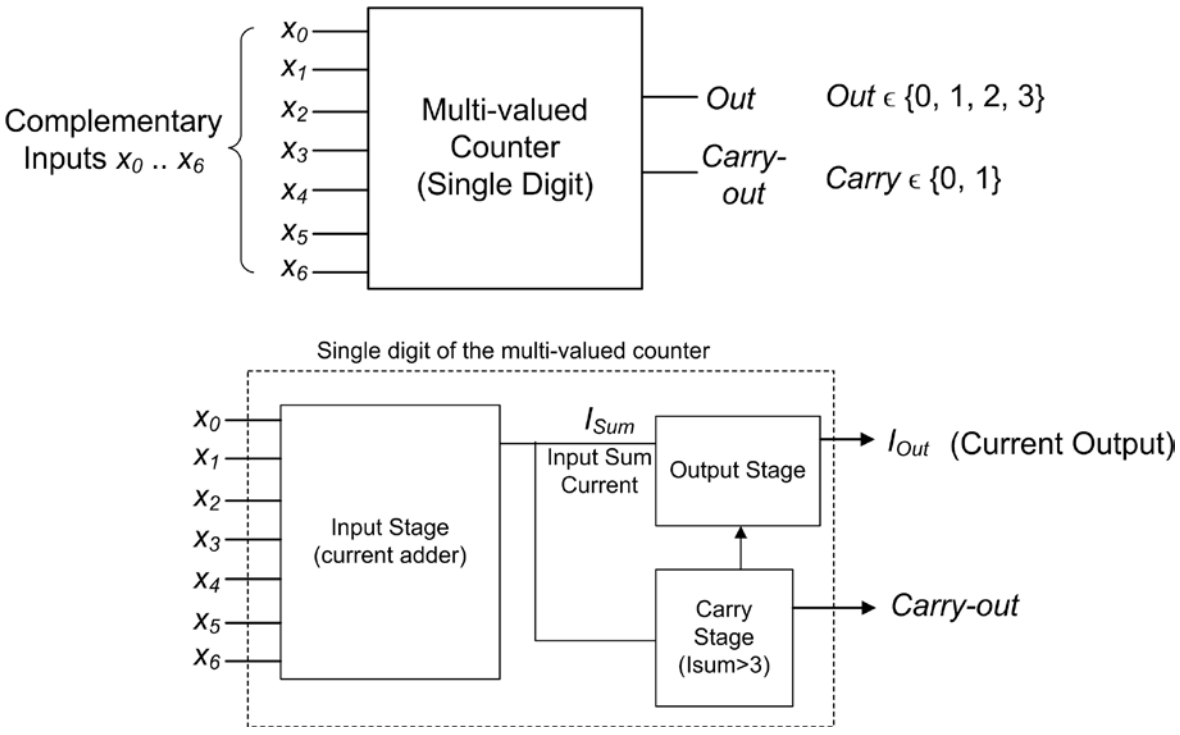



Figure 4.18. Multi-valued counter block diagram

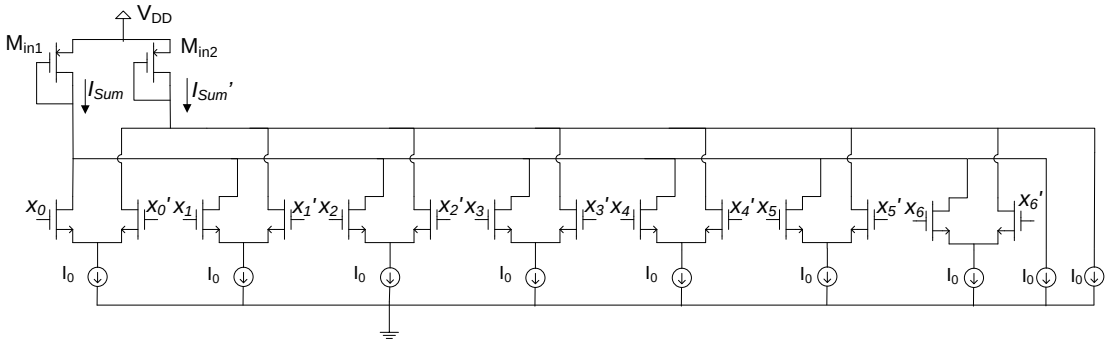



Figure 4.19. Input block of the multi-valued 7-bit addition circuit

The circuit has a logic value set of {0, 1, 2, 3} at the output and radix-2 at the input. Selection of the output levels in radix-4 i.e. in the digit set of {0, 1, 2, 3} is for compatibility of the radix with binary system. Increasing the digit set would bring difficulties in terms of analog design issues. As a result, radix-4 is selected for a good tradeoff between circuit limitations and higher radix that would have enabled higher compactness.

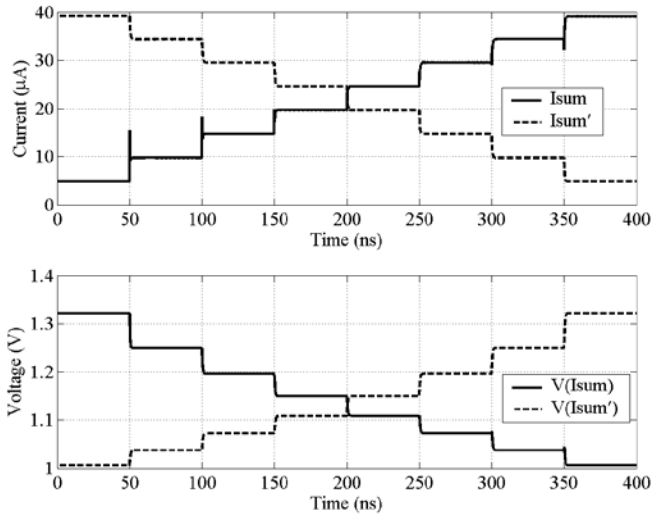



Figure 4.20. The input currents $I_{sum}$ and $I_{sum'}$ and generated voltages over the drains of the input transistors $M_{in1}$ and $M_{in2}$

Table 4.5. Logic levels of $I_{sum}$ and $I_{sum'}$ currents at the input stage

| **Logic Level** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|
| $I_{sum}$ (μA) | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 |
| $I_{sum}$' (μA) | 40 | 35 | 30 | 25 | 20 | 15 | 10 | 5 |

In this seven-operand binary input and four valued output addition circuit, there must be carry-out generated from transition from 3 to 4 since the output of the circuit must retain a value in the output digit set {0, 1, 2, 3}. The carry generation circuit is shown in Figure 4.21. The figure shows the carry-out circuit which is actually a differential comparator. $M_{CL1}$ and $M_{CL2}$ transistors are used to clamp the output voltages of the carry-out and carry-out' at certain levels. The cross-coupled load pair in the differential stage boosts the gain for carry generation [58]. The gain of the differential pair is given in the previous equations (4.8) and (4.9).

Whenever carry occurs in the circuit, '4' must be subtracted from the output. The subtraction circuit and output generation via subtraction is shown in Figure 4.22 (a). The complementary nature of the circuit provides that whenever '4' is subtracted from the $I_{sum}$, it is added into $I_{sum'}$, where complementary operation provides better signal immunity on small voltage swings of multiple-valued data. The circuit in Figure 4.22 (b) is used to simulate current mode carry-out signal where the output current $I_{out}$ and carry-out signal simulation outputs are shown in Figure 4.23. Table 4.6 shows the $I_{out}$ and $I_{out}$' signal levels in terms of current steps.

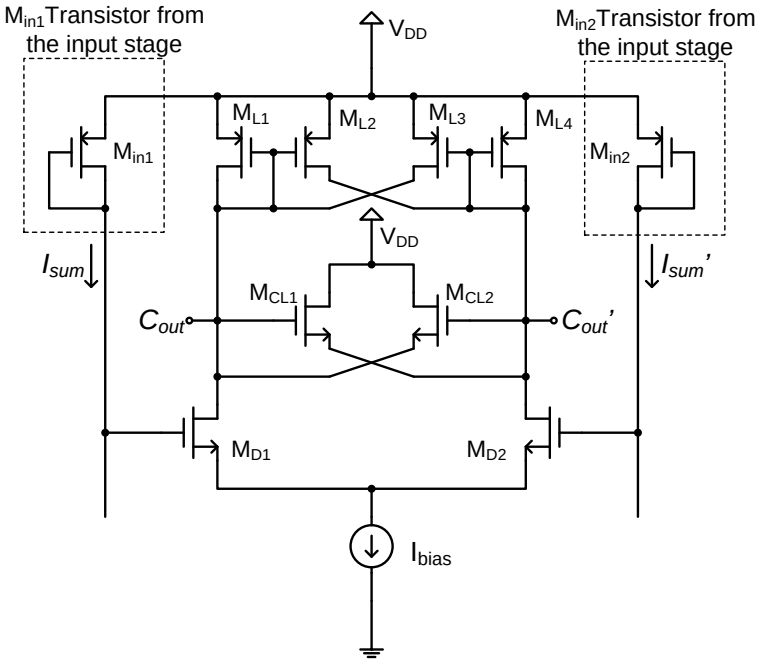
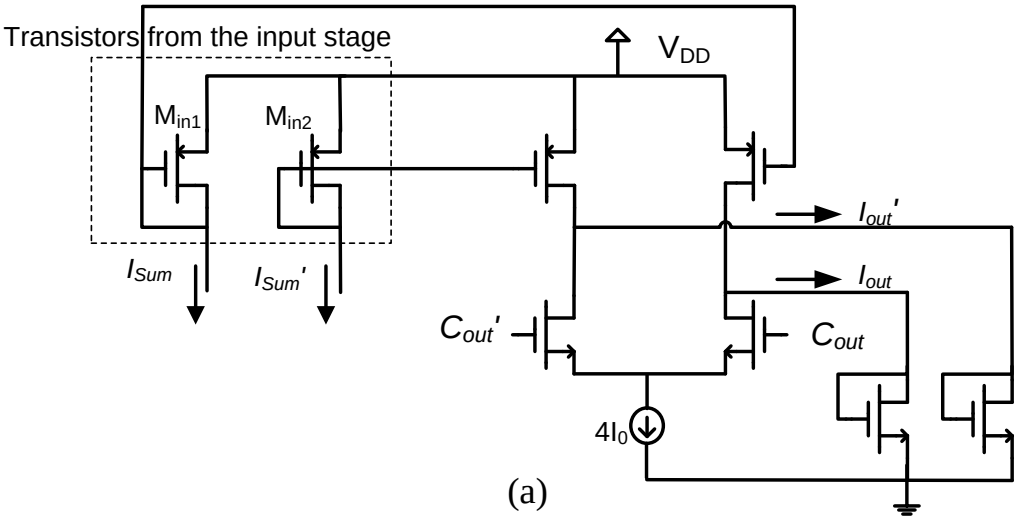



Figure 4.21. Carry generation circuit

Table 4.6. Logic levels of $I_{out}$ and $I_{out}$' currents at the output

| **Logic Level** | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| $I_{out}$ (µA) | 5 | 10 | 15 | 20 |
| $I_{out}$' (µA) | 20 | 15 | 10 | 5 |



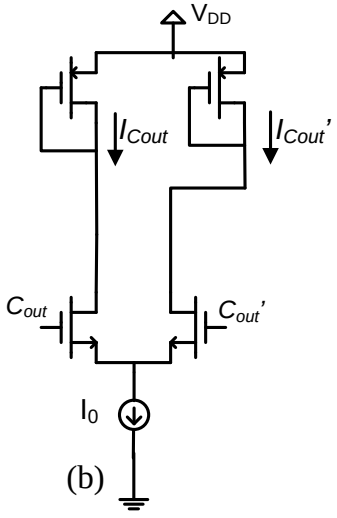




Figure 4.22. (a) Output current generation; (b) current-mode carry generation

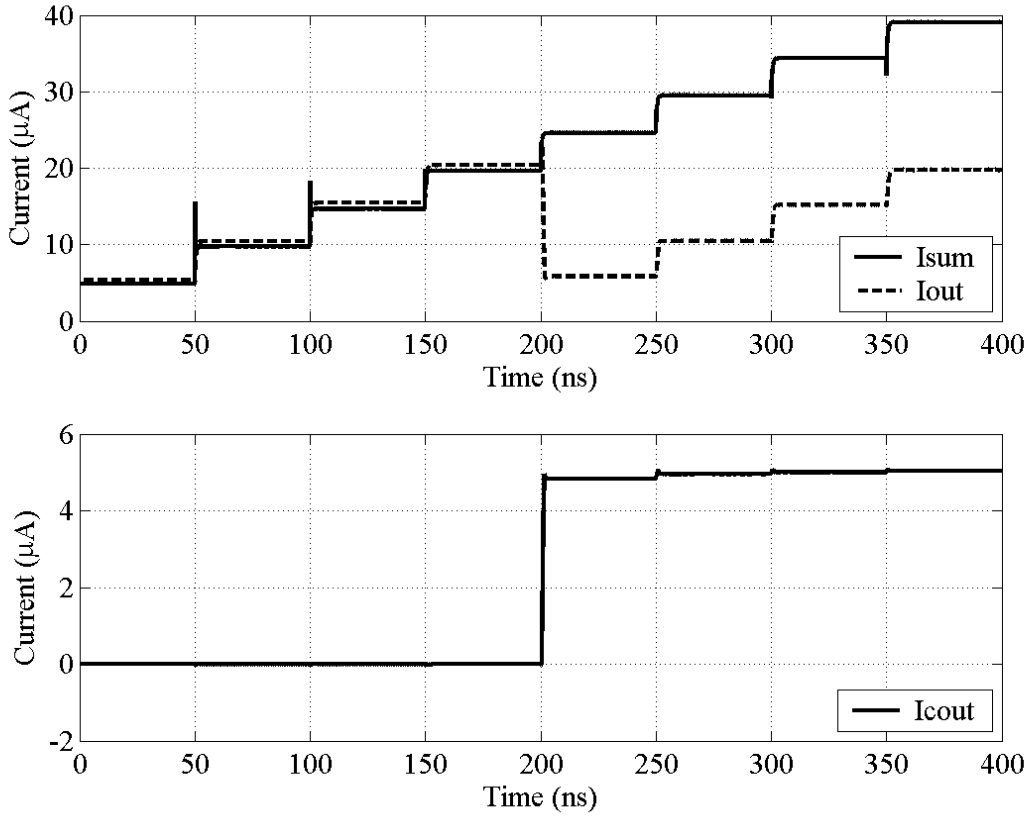



Figure 4.23. Output current ($I_{sum}$) and carry-out ($I_{Cout}$) simulation results

In the proposed circuit, the output current has multi-valued (MV) output. The output can be used as an input to a MV circuit or alternatively can be converted to binary. To convert the MV data to binary, comparator circuits should be used as shown in Figure 4.9 (a) or Figure 4.9 (b). Conversion from MV data to binary causes time delay, and reduces the performance of the circuit.

The next approach in Section 4.2.2 is to convert multi-valued data to binary in every stage, and to have totally equivalent functionality as conventional binary (7, 3) counter. The drawback of the previous designs is that, output generation requires determination of the carry-out, which limits the system performance. The design in Section 4.2.2 alleviates this problem by providing higher parallelism in the determination of the outputs since the carry generation and MV to binary conversion is done in parallel.

### 4.2.2. Binary Restored Multi-valued (7, 3) Counter

The principle operation of the counter circuits is explained at the beginning of Section 4.2. Here, a novel (7, 3) counter is proposed which is designed using multi-valued current mode design principles. The system is composed of input block, carry generation circuit, comparator circuit and output stage. General working scheme for the proposed circuit can be depicted in Figure 4.24.

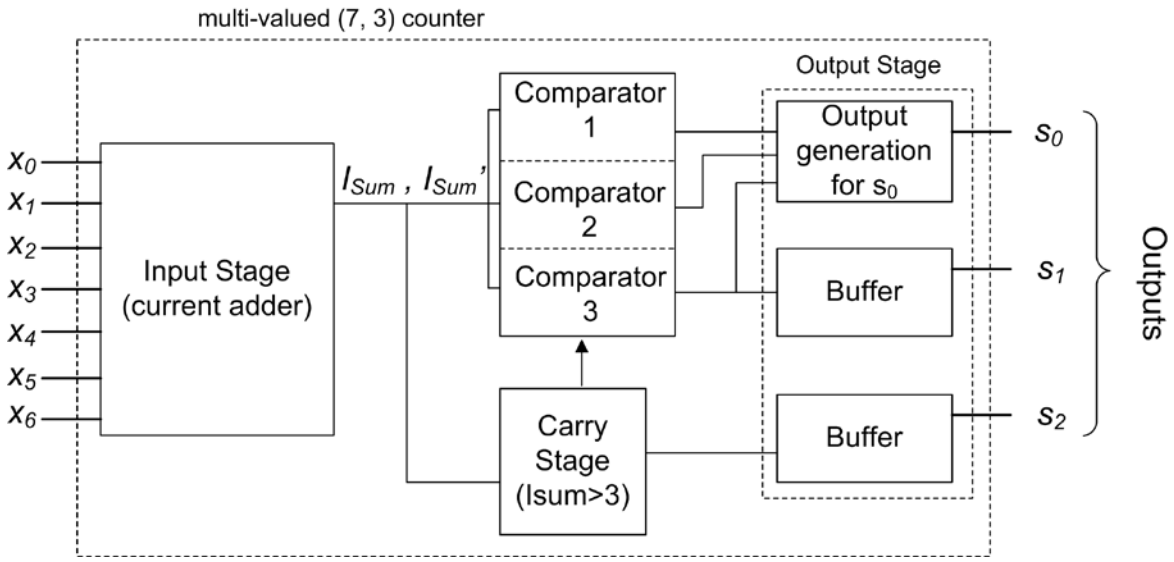



Figure 4.24. Implementation of the proposed multi-valued counter

The multi-valued counter circuit proposed here has complementary inputs and outputs. The system is biased with constant current sources at all stages. The power consumption is constant at all frequency ranges of the circuit and at any switching condition. The circuit voltage supply $V_{DD}$ is selected to be 1.8 V at all stages. The circuit is implemented in 0.18 μm UMC technology. The circuit has seven complementary input operands with three complementary outputs, as the (7, 3) counter definition implies. The input stage of the multi-valued counter circuit is shown in Figure 4.25. For the multi-valued input stage, a constant current is set up for each logic input level. In the example configuration, the input stage unit current $I_0$ is selected to be 2 μA for each bit, which can be adjusted to meet various performance requirements. In principle, the unit current defines the performance characteristics of the circuit. However, changing unit current may affect the circuit biasing and the transistor aspect ratios will be needed to be adjusted for

optimum performance. Here, $x_0$ ... $x_6$ are one bit slice of the seven numbers that are to be added up. $x_0$' … $x_6$' are the complementary inputs. $I_0$ is the current value of the simple current source connected to each of the differential pair where the value is set to 2 µA. Resistors $R_1$ and $R_2$ carry the input stage addition currents $I_{sum}$ and $I_{sum}$' respectively. These two currents are complements of each other. Basically, $I_{sum}$ and $I_{sum}$' represents the addition of seven inputs, namely $x_0$ ... $x_6$. Here, the circuit operates in fully differential mode. The corresponding current values for the summation are listed in Table 4.7. As listed in Table 4.7, $I_{sum}$ value increases by 2 µA steps whereas $I_{sum}$' decreases by 2 µA steps when the logic level increases.

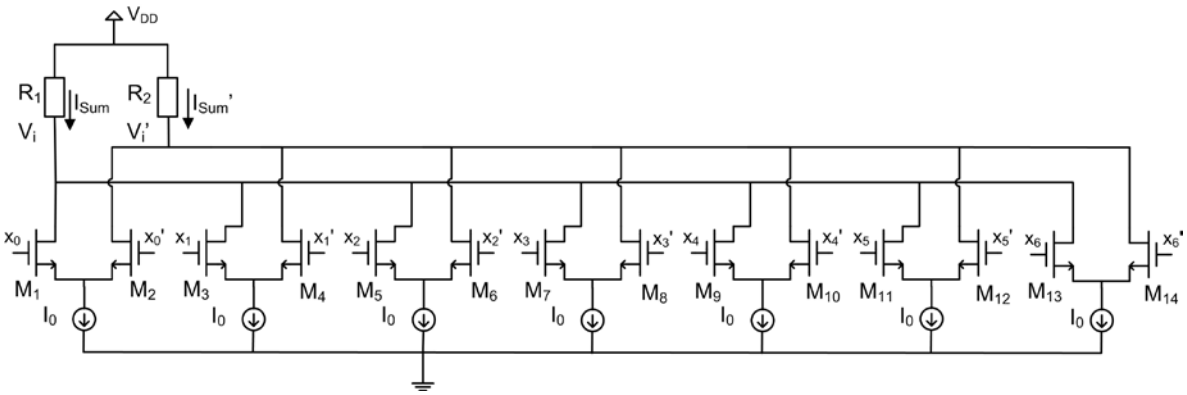



Figure 4.25. Input stage of the multi-operand counter

Table 4.7. Logic levels of $I_{sum}$ and $I_{sum}$' currents at the input stage

| **Logic Level** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|
| $I_{sum}$ (µA) | 0 | 2 | 3 | 6 | 8 | 10 | 12 | 14 |
| $I_{sum}$' (µA) | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 0 |

There are seven inputs and three outputs in the circuit. The output ($s_2s_1s_0$) will vary between (000) and (111) in binary representation which corresponds to output values between 0 and 7 in decimal. The input currents are accumulated over $R_1$ and $R_2$ resistors. The whole addition is computed over these resistors. The input stage acts as a simple digital to analog converter, where multi-valued output currents $I_{sum}$ and $I_{sum}$' are the analog outputs of the circuit. Figure 4.26 illustrates the $I_{sum}$ and $I_{sum}$' characteristics (first plot) and

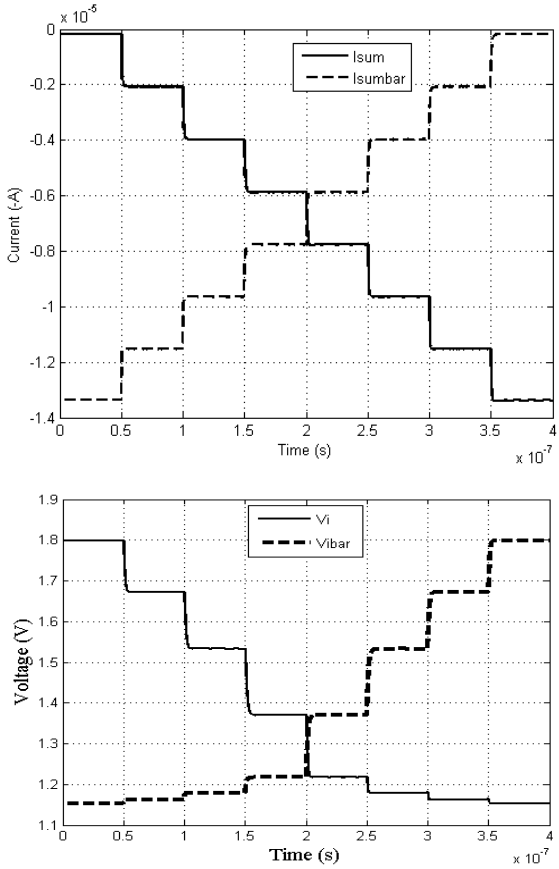



Figure 4.26. Current and voltage waveforms of the input circuit

generated voltage levels (second plot) on resistors $R_1$ and $R_2$ named as $V_i$ and $V_i$' at the input stage in Figure 4.25.

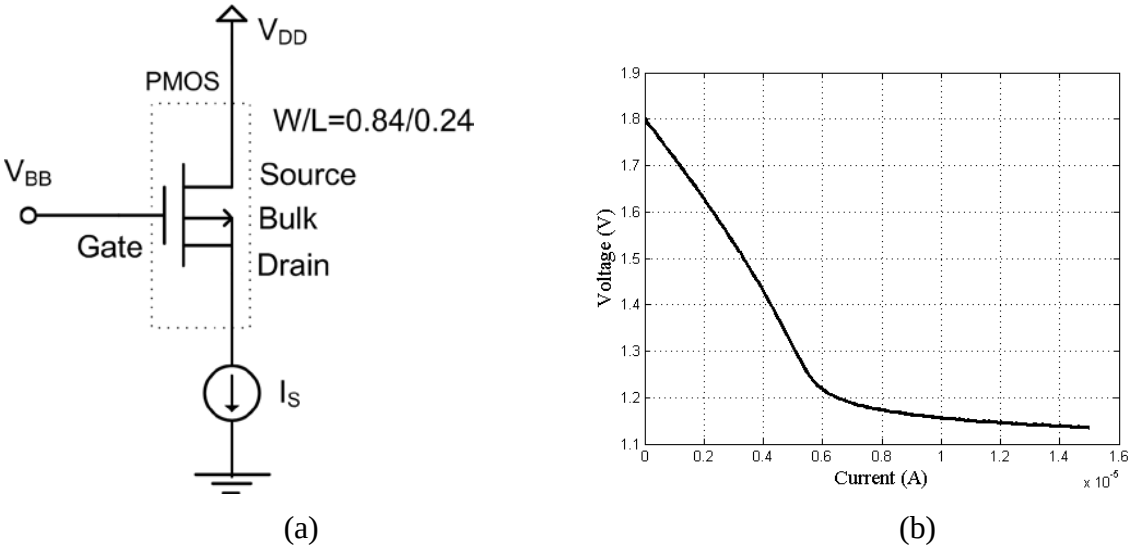
As the output characteristics show, input currents are added up linearly as logic levels of the inputs ($x_0$ ... $x_6$) are switched in the sequence (0000000), (1000000), (1100000), ... (1111111). The output currents $I_{sum}$ and $I_{sum}$' agree with the characteristics as listed in Table 4.7. At the same time, the voltage levels over $R_1$ and $R_2$ shows nonlinear characteristics when $I_{sum}$ and $I_{sum}$' values exceed a certain threshold. This property provides perfect biasing of the differential input transistors, namely $M_1$ ... $M_{14}$ in the Figure 4.25. Furthermore, it provides sufficient voltage headroom for multi-valued signals for the seven total logic levels. The nonlinear resistor implementation of $R_1$ and $R_2$ is realized using bulk to drain connection of PMOS pairs. The resistor implementation in this way provides high resistance with nonlinear characteristics [60]. Nonlinear resistance $R_1$ and $R_2$ are implemented as shown in Figure 4.27 (a). The transistor implementation is shown inside a

dotted square. The transistor bulk is connected to drain. The figure shows the whole test circuit for the resistance. The current and voltage characteristics of the nonlinear transistor are shown in Figure 4.27 (b).

Since the circuit adds up seven bits, and the result is three bits, whenever the addition value exceeds three, the third digit of the sum switches to one. From (4.14):

$$if \qquad \sum_{i=0}^{6} x_i > 3 \qquad s_2=1 \tag{4.15}$$

In Table 4.7, whenever the logic level is greater than 3, $I_{sum}$ value is greater than $I_{sum}$' and $V_i$' is greater than $V_i$ which is also shown in Figure 4.26. To sense the transition as a logic level it is fed into a source coupled comparator block which is shown in Figure 4.28. Here, transistors $M_1$ and $M_2$ are diode connected. Therefore, a diode voltage drop, which provides better biasing for the next stages, appears at the drains. The outputs $V_O$ and $V_O$' correspond to the most significant digit with its complement, $s_2$ and $s_2$'.




(a) (b)

Figure 4.27. Drain to bulk connected PMOS transistor used as a nonlinear resistor (a) electrical connection; (b) current-voltage characteristics [60]

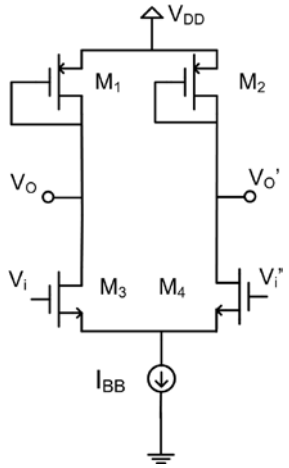



Figure 4.28. Comparator and output generator for $s_2$

In addition to the generation of $s_2$, other logic levels must be generated for proper operation. Comparator stages are required for sensing of other logic levels. Comparator stages for sensing logic levels of only 1, 2 and 3 are required. $V_i$ and $V_i$' are fed into the comparator in a multiplexed fashion regarding to the condition that $V_O$ is generated or not generated. The first comparator output $V_O$ of Figure 4.28 is fed into the comparators in Figure 4.29 as logic inputs. In Figure 4.29, *V(1), V(2)* and *V(3)* are constant comparison values and *V'(1) = V(3), V'(2) = V(2)* and *V'(3) = V(1).* Summation values of the inputs, corresponding comparator outputs and required circuit outputs are listed in Table 4.8. Using this table, $s_0$, $s_1$ and $s_2$ values are extracted by using Boolean algebra:

$$s_0 = V_{out1} \cdot V_{out2}' + V_{out3} \tag{4.16}$$

$$s_1 = V_{out2} \tag{4.17}$$

$$s_2 = V_O \tag{4.18}$$

Table 4.8. Summation and corresponding comparator outputs

| ***Sum*** | $V_O$ | $V_{out1}$ | $V_{out2}$ | $V_{out3}$ | $s_2s_1s_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 000 |
| 1 | 0 | 1 | 0 | 0 | 001 |
| 2 | 0 | 1 | 1 | 0 | 010 |
| 3 | 0 | 1 | 1 | 1 | 011 |
| 4 | 1 | 0 | 0 | 0 | 100 |
| 5 | 1 | 1 | 0 | 0 | 101 |
| 6 | 1 | 1 | 1 | 0 | 110 |
| 7 | 1 | 1 | 1 | 1 | 111 |

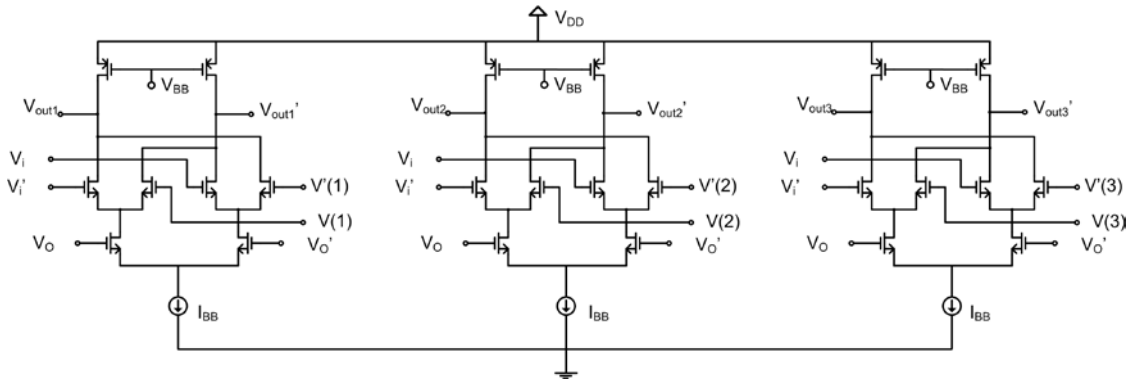



Figure 4.29. Comparator stages for sensing the logic levels

To get the summation results, regarding to (4.16), only an AND - OR circuit is required for $s_0$. The circuit implementation in SCL is shown in Figure 4.30 (a). The circuit computes the function $f = a + b \cdot c'$ . Other output bits $s_1$ and $s_2$ appear at the outputs of the comparators regarding to (4.17) and (4.18). As shown in Figure 4.30 (b), outputs $s_1$ and $s_2$ are buffered for higher driving strength at the output. The connection diagram for the whole proposed system can be depicted in Figure 4.31. HSPICE transient simulation results for the outputs are shown in Figure 4.32. Summation values from 0 to 7 are swept in the simulation. Complementary outputs are shown as in three subplots. First plot contains $s_0$ and $s_0$', second one contains $s_1$ and $s_1$', third one contains $s_2$ and $s_2$'. The output swing is set to 0.4V as complementary outputs. The system outputs can be fed into other source coupled systems (SCL) seamlessly.

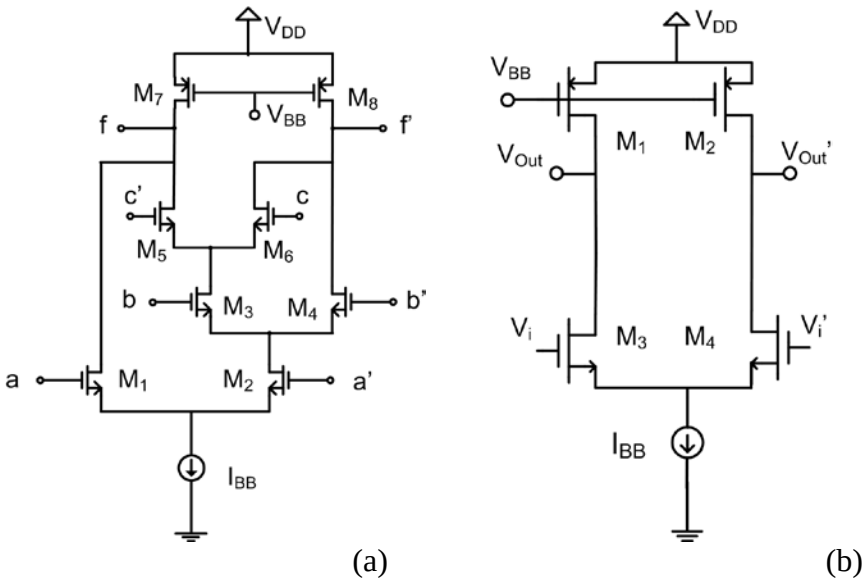



Figure 4.30. (a) Output generation for the $s_0$ and $s_0^{'}$; (b) output buffer

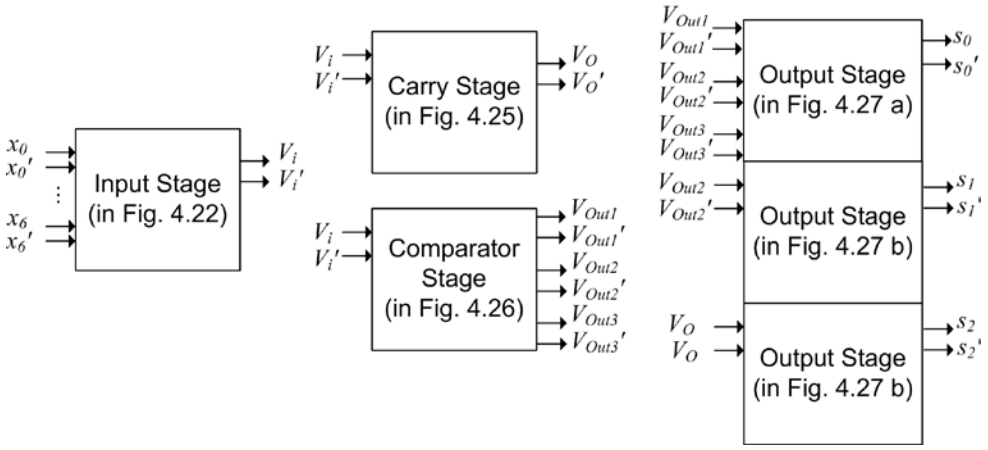



Figure 4.31. Connections of the proposed system

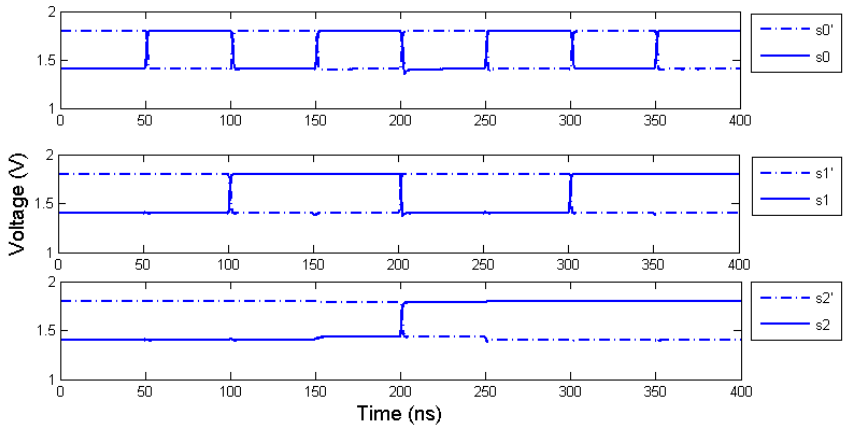


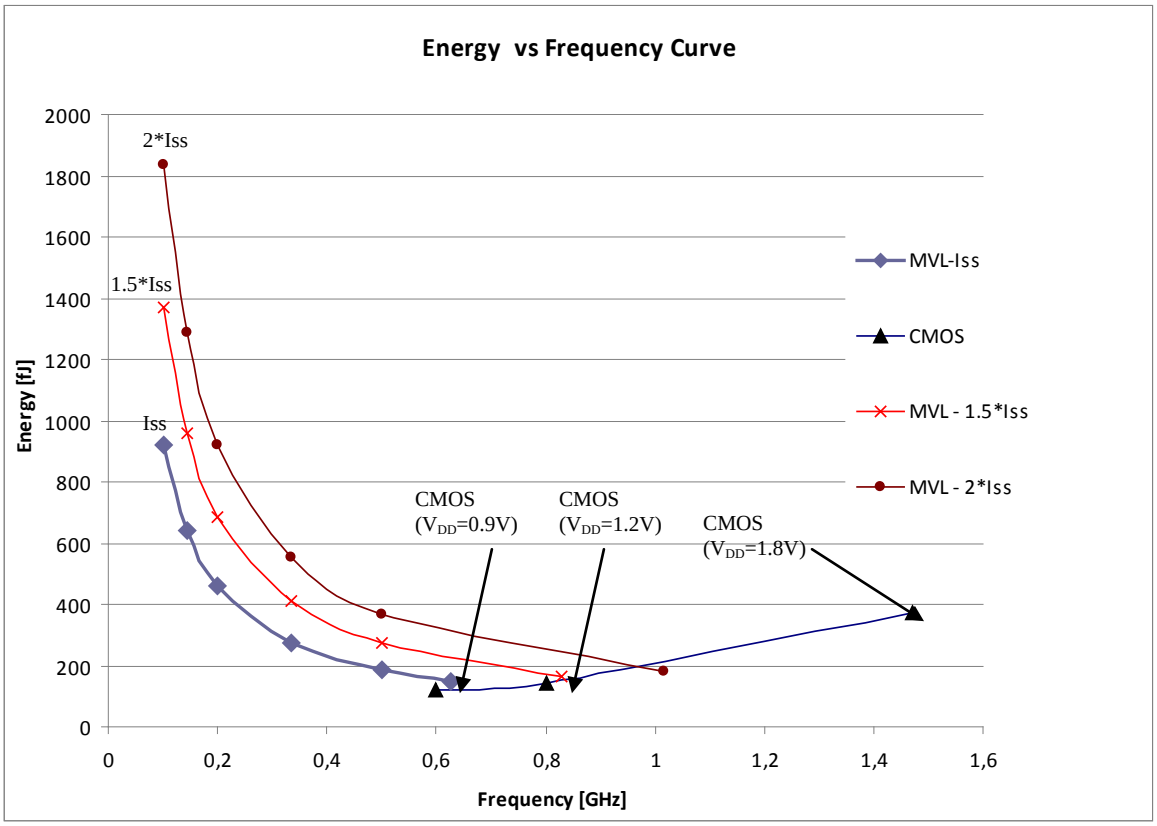

Figure 4.32. Logical outputs of the circuit

In the proposed configuration for (7, 3) counter, the transistor count is reduced. The input stage requires 23 transistors; the first comparator circuit at Figure 4.28 requires 5, three output comparators of Figure 4.29 requires 3x9 = 27 transistors. The And-Or circuit to generate $s_0$ shown in Figure 4.30 (a) requires 9 transistors. The output buffers of $s_1$ and $s_2$ in Figure 4.30 (b) is used which requires 2x5 = 10; as total, 74 transistors are needed. For binary (7, 3) counter, four full adders are needed each needs 28 transistors [12], as total of 4 x 28 =112 transistors. The transistor count is reduced by 34 per cent. Moreover, current mode design provides better power control and reduced switching power. The power consumption of the circuit is dependent on the current sources of the circuit, where it is fixed to 2 µA at the input stage and 5 µA at all of the other stages. The circuit delay is measured as 1.6 ns at a constant current consumption level of 50 µA. The standard CMOS implementation of the (7, 3) counter designed in same technology consumes 93 µA at 400

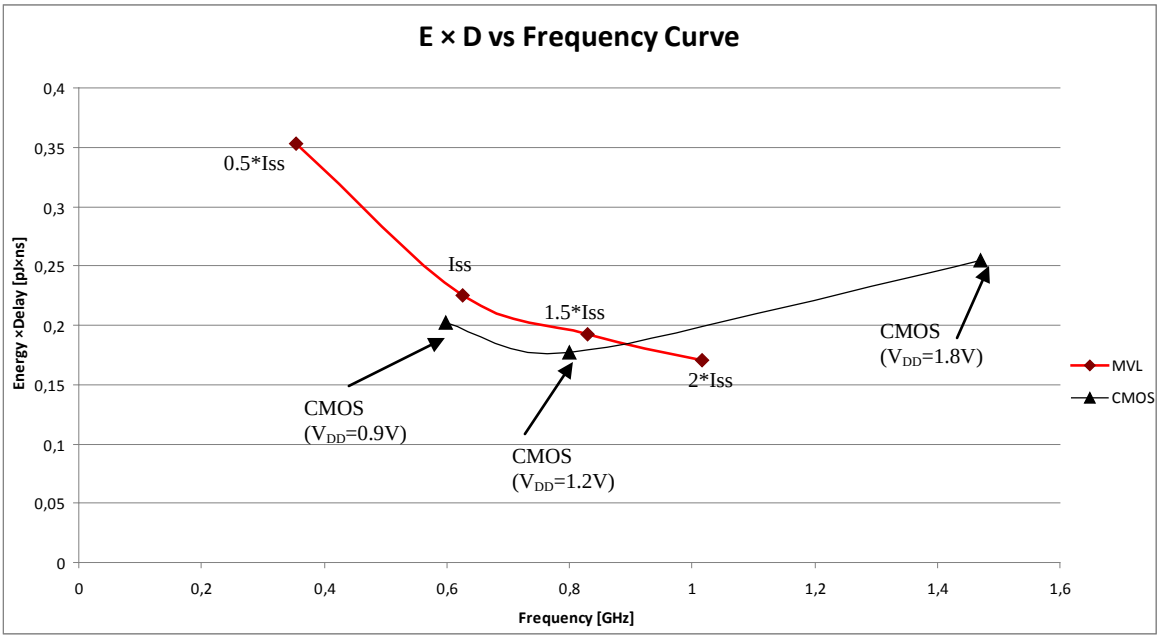
MHz. The power consumption of the standard CMOS circuit is estimated by applying random input patterns. 46 per cent power reduction is achieved at the same operating frequency. Since the power consumption is linear with frequency for the static CMOS circuits, higher than approximately 200 MHz, the proposed circuit provides better power characteristics for the same voltage supply. As a conclusion, an analog friendly current mode multi-operand addition cell is designed. The circuit is power efficient at high frequencies and can be used without any power penalty at 200 MHz or higher switching frequency compared to binary standard CMOS implementation.

For detailed analysis on power consumption and energy efficiency, Figure 4.33 (a) and (b) should be explored. The common difference between static CMOS and current-mode logic circuits is that, in static CMOS, energy consumption for each switching is constant, and the power consumption is proportional with frequency. On the other hand, in current-mode logic circuits, power consumption is always constant and it does not change with working frequency. Therefore, energy efficiency is always higher if the circuit is run as high as possible. Figure 4.33 (a) shows the energy versus frequency curve. The plot contains current scaling of the proposed current-mode circuit with constant supply voltage as 1.8 V and supply voltage scaling in standard CMOS circuit. The plot reveals that, as supply voltage for the CMOS circuit is scaled down to 0.9 V, the circuit consumes lower power. Therefore the plot shows that supply voltage scaling results in better result for lower power at low working frequencies compared to the proposed circuitry.

Figure 4.33 (b) shows Energy x Delay versus frequency curves. The plot shows that, the proposed circuit becomes energy efficient compared to standard CMOS if it is run at Gigahertz range. Another point is that, the CMOS circuit becomes most energy efficient if it is run at 1.2 V supply voltage. The SCL versus the proposed design energy and delay characteristics is almost equal, since they both have similar working characteristics. The advantage of the SCL and the proposed current-mode design is that, current consumption is constant all the time, however, the current consumption of the static CMOS circuit is variable and at its highest point at the time of starting phase at each switching activity. A plot for CMOS current consumption characteristics has been given in Figure 4.12. Since the current consumption is not constant in standard CMOS, it has peak power consumption



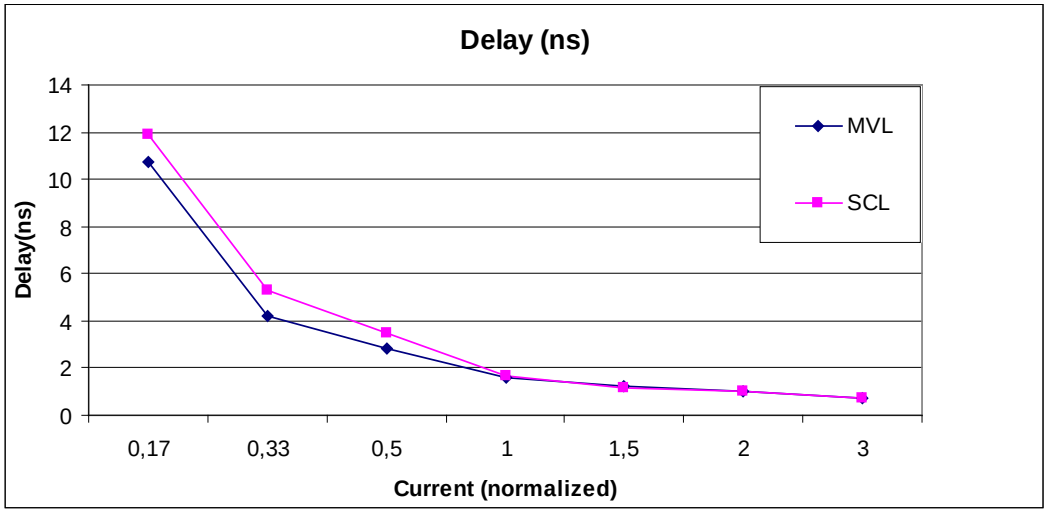

(a)



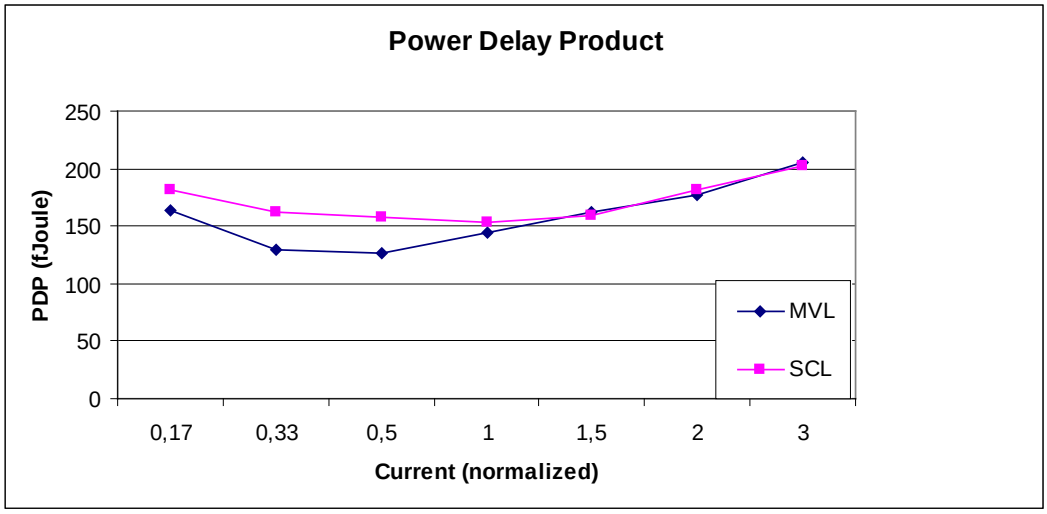

(b)

Figure 4.33. (a) Energy versus Frequency; (b) Energy x Delay versus Frequency

at random points which can be dangerous whenever the current sourcing capability is limited in a system. Moreover, in current-mode systems, much less noise is generated since the current consumption over the power supply is constant. Another point is that, voltage scaling is also possible for the SCL circuits and general current-mode circuitry. However, this time all the active devices should be resized for proper operation, since the circuits are not scalable over supply voltages or technology scale down. As a result, static CMOS circuits seem most flexible for power consumption and scalability issues and the advantages of the current-mode circuits appears only at very high frequencies and for the cases that low noise digital circuits are required.




(a)




(b)

Figure 4.34. (a) Delay vs. current consumption; (b) PDP vs. current consumption

Figure 4.34 (a) shows the delay curve of the proposed circuit and SCL equivalent under same current consumption configuration. The delay curves of the two equivalent circuits are close to each other. Fig. 4.34 (b) shows the power-delay product (PDP) of the two circuits as well. According to the figure, the proposed circuit has better PDP characteristics if the current reduced.

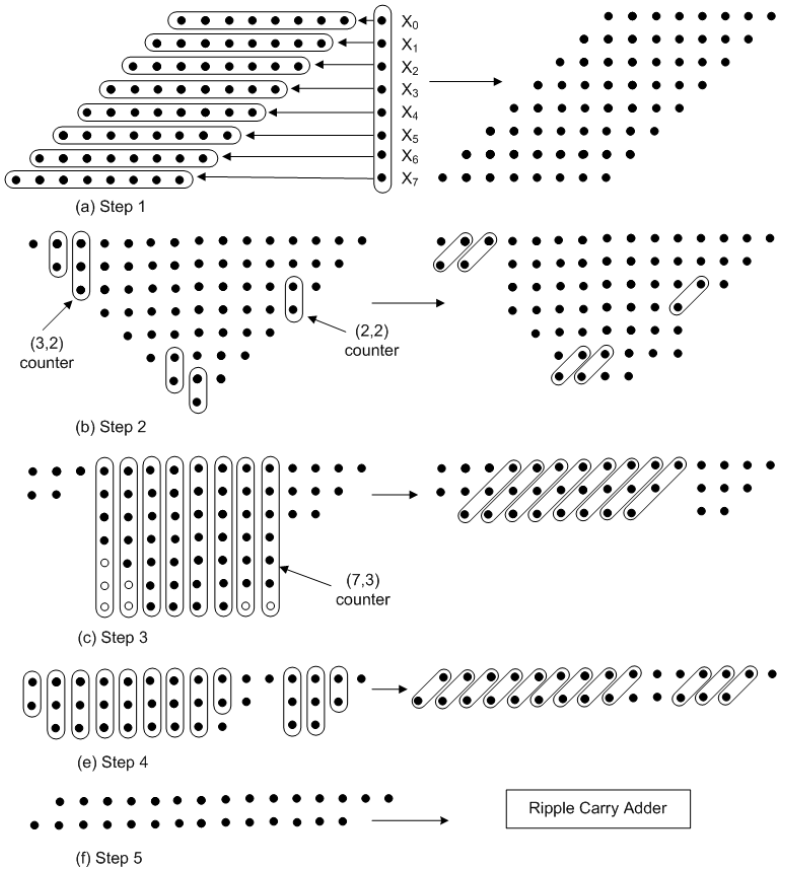



Figure 4.35. 8 x 8 bit multiplication scheme

To show an application example, an 8 x 8 bit multiplier circuit is designed using the proposed multi-operand adder. For the partial product generation, conventional source-coupled logic (SCL) AND gates are used. The multiplication scheme is shown in Figure 4.35. At step 1, partial products are generated for each array using AND gates. There are 8 partial products generated in step 1. However the a (7, 3) counter accepts 7 inputs. Therefore, in the second step partial products are reduced to 7 using (2, 2) reduction by half adder circuits. In step 3, (7, 3) reduction is done using the new multi-operand addition circuit. The empty dots in step3 are logic ‘0’ inputs, where no input operand exists. Next,

(3, 2) reduction is made using full adders. In last step, ripple-carry adder is used for the addition of the products.

The current-mode multi-valued adder circuit simultaneously adds up seven operands and is functionally equivalent to the binary (7, 3) counter circuit. The circuit in this paper is designed for the implementation of low power mixed signal systems. The adder circuit consumes 51-µA current in this current configuration. The worst-case delay of the proposed circuit is measured to be 1.6 ns. An equivalent SCL (7, 3) counter circuit is built using full adders seen in Figure 4.36. The full adders are tiled according to Figure 4.15 as tree structure and for equivalent power consumption with our circuit each current source is fixed to 6.5-µA. The delay of the circuit is measured to be 1.7 ns.

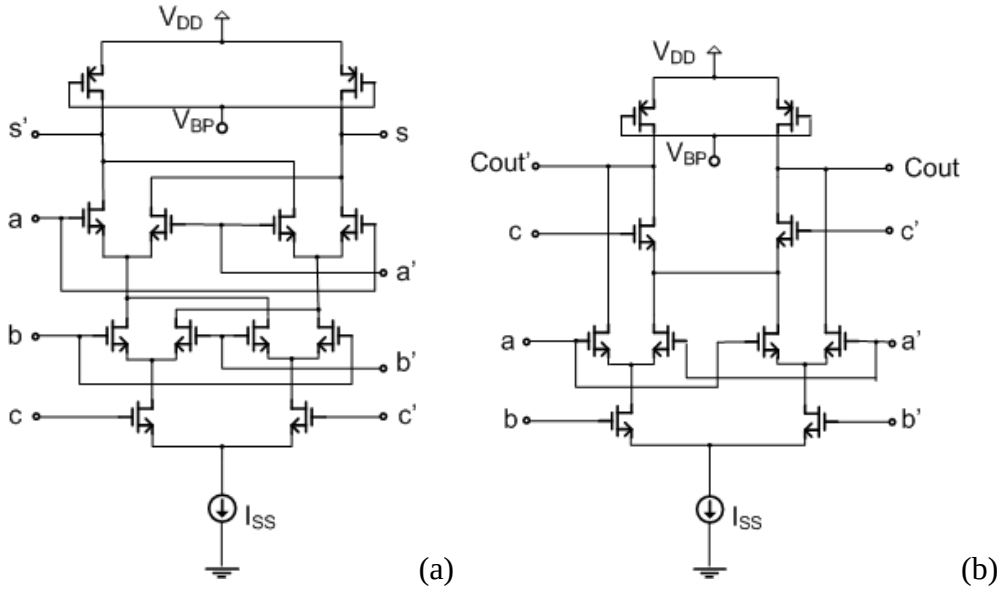


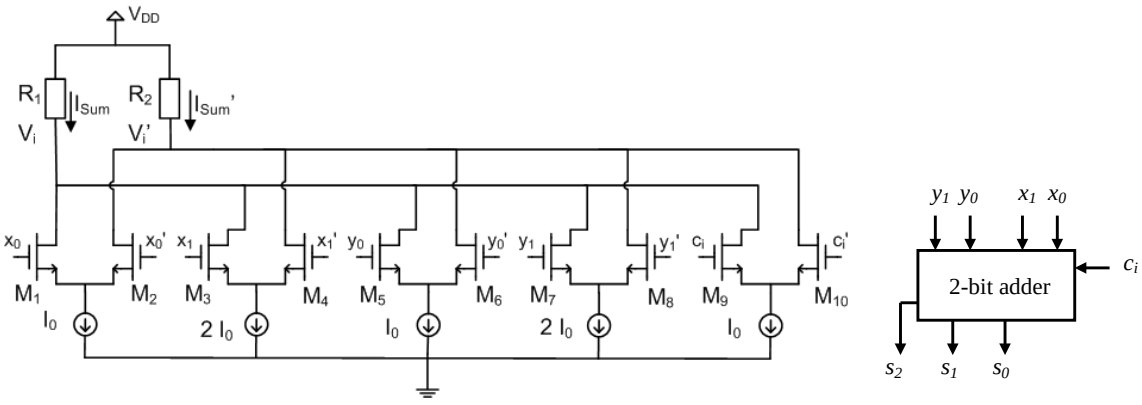

Figure 4.36. Source-coupled full adder circuit: (a) sum generation; (b) carry-out

An 8x8-multiplier circuit is built using our multi-operand addition circuit. A total number of eight of the proposed multi-operand circuit is used in the multiplier. The saving in terms of transistor count is 176. In more complex multiplier schemes, the saving in the transistor count would be higher. The delay of both implementations is nearly equivalent and the design with our multi-operand adder seems slightly faster. The comparison can be seen in Table 4.9.

Table 4.9. Comparison of the adders and multipliers

| | **Transistor count** | **Delay (ns)** | **Current consumption (µA)** |
|---|---|---|---|
| SCL (7,3) counter | 96 | 1.7 | 52 |
| Our circuit | 74 (% 23 saving) | 1.6 | 51 |
| Multiplier Circuit (normal SCL) | 1639 | 3.3 | 1520 |
| Multiplier Circuit (our circuit used) | 1463 (%11 saving) | 3.2 | 1510 |

The counter circuit proposed here can also be organized as a radix-4 adder, or equivalently two-bit binary adder by only modifying the input stage shown in Figure 4.25. The two-bit adder circuit can be realized by replacing the input stage of the proposed multi-operand (7, 3) counter by only replacing the input stage seen in Figure 4.25 by the input stage seen in Figure 4.37 (a). By cascading the comparator stage and the output stage proposed in this section, the equivalent two-bit adder can be realized that is shown in Figure 4.37 (b). However, this circuit requires 65 transistors, where a two-bit SCL adder requires two full adders which require 2x24 = 48 transistors. As a result, two-bit adder is not very efficient for multi-valued design scheme, contrary to the multi-valued (7, 3) counter proposed here. This comparison justifies our multi-valued multi-operand design approach throughout our work presented in this chapter.




(a) (b)

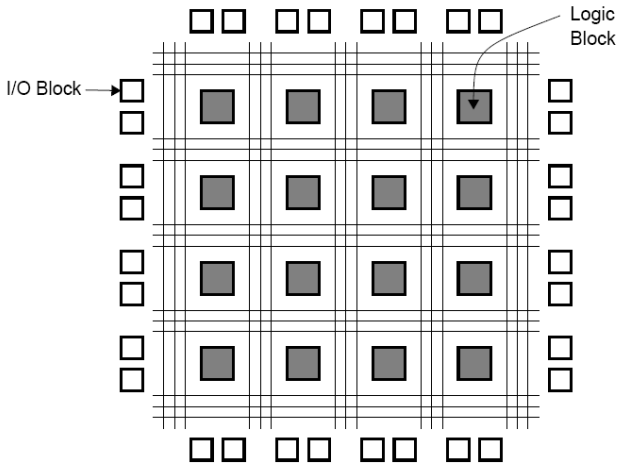
Figure 4.37. (a) Input stage of the MV two-bit adder; (b) two bit adder representation

# 5. REDUNDANT ARCHITECTURES AND FIELD PROGRAMMABLE GATE ARRAYS

Field Programmable Gate Array (FPGA) devices are becoming more and more popular in the current digital design market. Moreover, advanced logic block structures are being developed for higher performance. Redundant architectures for high performance arithmetic can be implemented efficiently on Field Programmable Gate Arrays (FPGAs) which are being used extensively in the industry nowadays.

New FPGA families support 6-input Look-up Table (LUT) devices [61, 62] that can be exploited for high performance arithmetic systems. In this chapter, especially carry-free arithmetic using various redundant number representations are analyzed. The advantages of new generation FPGA devices are analyzed and a redundant arithmetic system suitable for 6-input LUT based FPGAs is proposed.

The basic structure of an FPGA is shown in Figure 5.1. In an FPGA, each logic block is composed of logic modules. Logic modules are the smallest functional blocks that contain Look-up Tables (LUTs). They are supported with registered outputs and unregistered outputs alternatively which are used for sequential and combinational logic functions, respectively [63].




5.1. Basic FPGA architecture

The LUTs are interconnected to other logic blocks via interconnects available around the blocks. There are various logic block structures depending on the model and producer of the FPGA, a logic block is a combination of single or multiple logic modules. Figure 5.2 shows a logic module of an Altera Flex-8000 logic module [64] where it contains a 4-input look-up table, fast carry logic, registered and unregistered outputs and other structures. As the most basic element in an FPGA is the Look-up Table (LUT) Unit, LUTs can be used as a density metric for the comparison of various FPGA types.

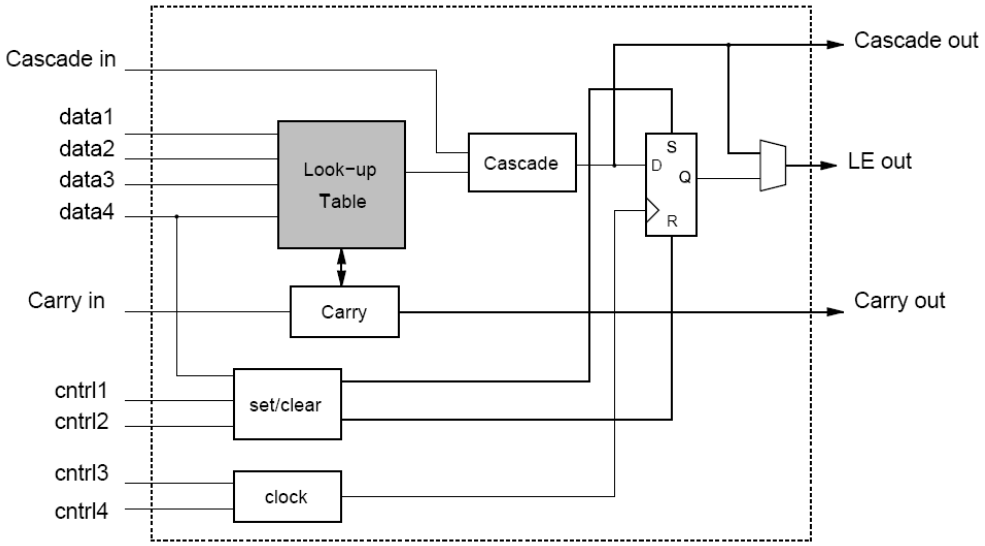



Figure 5.2. Logic module of Altera Flex 8000

New series of FPGA families such as Xilinx Virtex-5 [62] and Altera StratixII, StratixIII and StratixIV [65] devices support adaptive look-up tables (ALUT) modules. Each ALUT module can be configured as, 4-input, 5-input or 6-input LUT tables. Figure 5.3 shows two different configuration of an ALUT module which can be configured as two 4-input LUT or a single 6-input LUT module. In this chapter, especially 6-input LUT modules are explored for high performance arithmetic. 6-input LUT devices provide multi-operand addition efficiently. Generally, partial products of a multiplier or operands of a multi-operand adder can be added up together vertically using counters. The details of the counter circuits are given in Chapter 4. Especially (6, 3) counters are very suitable for the 6-input LUT structures [65].

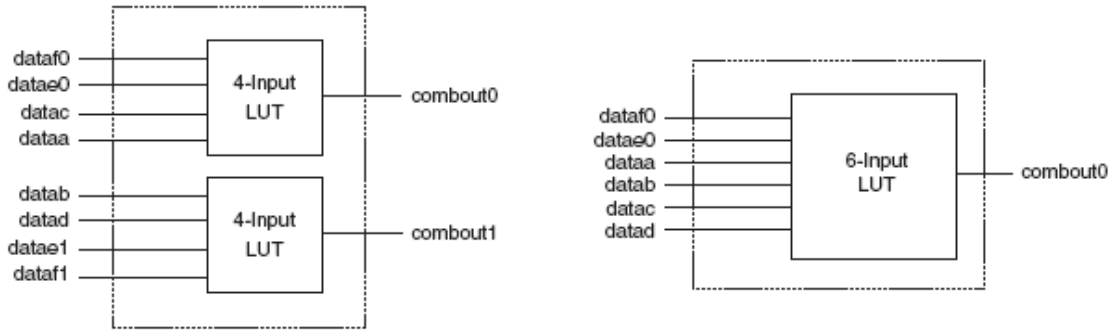



Figure 5.3. Adaptive six input LUT device of Stratix II FPGA [61]

If the architecture of the circuit is built considering the hardware structure of the system, better performance can be achieved. There are various recent papers addressing fast FPGA arithmetic appearing in [66-69]. However, they do not directly address the advantages of the new 6-input LUT devices. Signed-digit representations for carry-free arithmetic can be implemented efficiently especially with 6-input LUT structures. The work in [70] proposes radix-4 signed digit arithmetic on 6-input LUT devices, based on signed digit radix-4 addition scheme, which provides a carry-free fast addition structure. Multi-operand addition using counters can also be efficiently synthesized on 6-input LUT based devices.

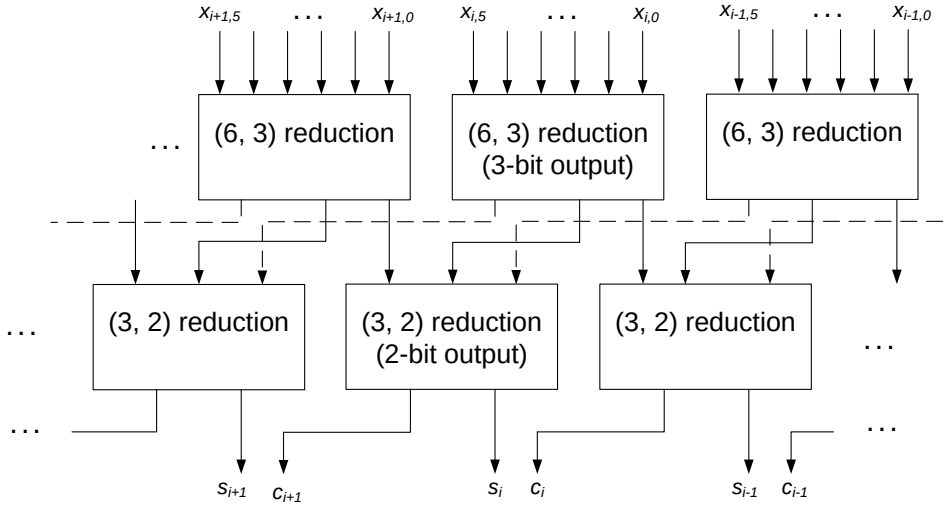



Figure 5.4. Addition of 6 operands in the LUT structure

A multi-operand addition scheme which simultaneously adds up six input operands is depicted in Figure 5.4. As can be seen in the figure, the six operand addition is realized using a two stage LUT structure. Here, six operands are reduced to three using (6, 3)

counters. Then, the redundant carry-save output is generated using a single array of (3, 2) counters, as shown in Figure 5.4. In this scheme, for each digit, one (6, 3) counter is synthesized using three LUT modules, which has three outputs. Each (3, 2) counter can be synthesized into a single 6-input adaptive LUT module, where a (3, 2) counter can be synthesized into two of 3-input LUT modules. As a result, four 6-input ALUT modules are required for the carry-save reduction for each digit in a six-operand addition. Carry-save arithmetic is explained in Section 2.1.4 and counter functionality has been explained in Section 4.2 as well.

Since 6-input LUT based FPGAs are newly introduced into the electronic market, efficient arithmetic implementations by exploiting 6-input LUT devices are not very common. In [65], the use of (6, 3) counters are suggested for multiplier designs. A redundant signed-digit arithmetic application on FPGAs appears in [70]. In [70] radix-4 signed-digit addition scheme is efficiently synthesized into 6-input LUT devices, where the application perfectly fits into this structure. The addition methodology in [70] can be explained as follows: Using the equations through (2.5) to (2.7) and selecting $r = 4$, interim sum is equal to

$$u_i = x_i + y_i - 4c_i$$

where carry is calculated as $a = (r - 1)$ :

$$c_i = \begin{cases} 1 \text{ if } (x_i + y_i) \geq 3 \\ \bar{1} \text{ if } (x_i + y_i) \leq -3 \\ 0 \text{ if } |x_i + y_i| < 3 \end{cases}$$

The summation function can be calculated as:

$$s_i = u_i + c_{i-1}$$

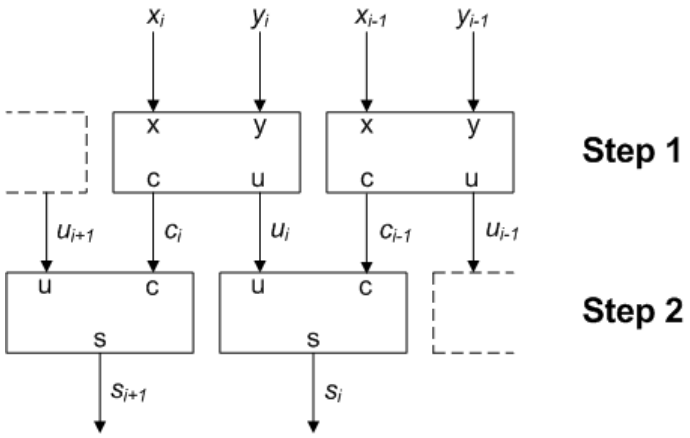



Figure 5.5. Addition of two signed digit numbers

The implementation structure is shown in Figure 5.5. Here, each variable $x_i$ and $y_i$ requires three bits, two bits for the encoding of the logic value and one digit for the sign bit (for radix-4 SD representation). As a result, six bits are required for the calculation of the interim sum and interim carry. Interim sum $u_i$ consists of three bits, two for logic value, and one for the sign bit. Interim carry $c_i$ requires two bits, one for sign and one for logic value. For the first stage, five 6-input LUT structures are required at first stage, three for interim sum, and two for interim carry generation. For the final summation, three bits are required, which corresponds to three 5-input LUTs. The output $s_i$ is represented as three bits same as $x_i$ and $y_i$. As a result, eight LUT structures are required for the generation of each digit. The proposed system in [70] is in radix-4, which is equivalent to two bits in radix-2. As a result, the addition process requires 8 / 2 = 4 LUT structures for each of the radix-2 equivalent digit. The number of stages required for SD addition is two LUT delays, as shown in Fig. 5.5.

After defining the carry-save structure in Section 2.1.4, and explaining the signed digit redundant addition scheme in here, an extra redundant arithmetic structure will be defined in the following sections. The defined structure is based on (6, 3) counters and boosts the performance for especially recursive addition operations, such as multiply-accumulate operations used in FIR filters, matrix multiplications etc. By breaking the carry chains and fitting the extra redundancy into the new generation FPGA devices, the number of LUT cascades is minimized.

Multiplying Accumulator (MAC) units are the basic building blocks of variable coefficient FIR structures, similar convolution operations and many of the signal processing blocks and matrix multiplication applications. By optimizing MAC units, higher data rates can be achieved in filtering applications. In the following sections, redundant arithmetic units for fast multiply-accumulate structures are described.

### 5.1. (6, 3) Counter and Simultaneous Addition of Six Binary Numbers

Mostly used counter circuits are (3, 2), (7, 3) and (15, 4) counters, where first operand implies the number of inputs and the second one is the number of the outputs. Here, to be most compatible with 6-input LUT devices, (6, 3) counters are used. In (6, 3) counters, six binary inputs are added up with three-bit outputs. Single bit and multi-bit addition using (6, 3) operands are shown in Figure 5.6 (a) and 5.6 (b), respectively.

In conventional carry-save structures, the carry save result is held by two digit sets, i.e. $Z = S + C$. Here, the redundant sum $Z$ is defined as:

$$\begin{aligned} S_a &= S_0 \\ S_b &= S_1 << 1 \\ S_c &= S_2 << 2 \\ Z_i &= \{S_{ai}, S_{bi}, S_{ci}\} \end{aligned} \tag{5.1}$$

$$Z = S_a + S_b + S_c \tag{5.2}$$

where "<<$a$" operator defines left shift by $a$. In this representation, each digit of the redundant sum consists of composition of three-bit set $Z_i = \{S_{ai}, S_{bi}, S_{ci}\}$. As a result each bit of $Z_i$ can have values between 0 ($S_{ai} + S_{bi} + S_{ci} = 0$) to 3 ($S_{ai} + S_{bi} + S_{ci} = 3$). In this structure, each digit is represented by three bits. In conventional carry-save system each digit is represented by two bits. Since each result is represented by 3-bit set, two results can be added up using one 6-input LUT device and computed in a single step. A single digit (6, 3) counter can be synthesized for single 6-input LUT using Verilog statements is given in Figure 5.7. Here, each definition of the (6, 3) counter defined using case statements to make compiler directly synthesize expected LUT configuration.

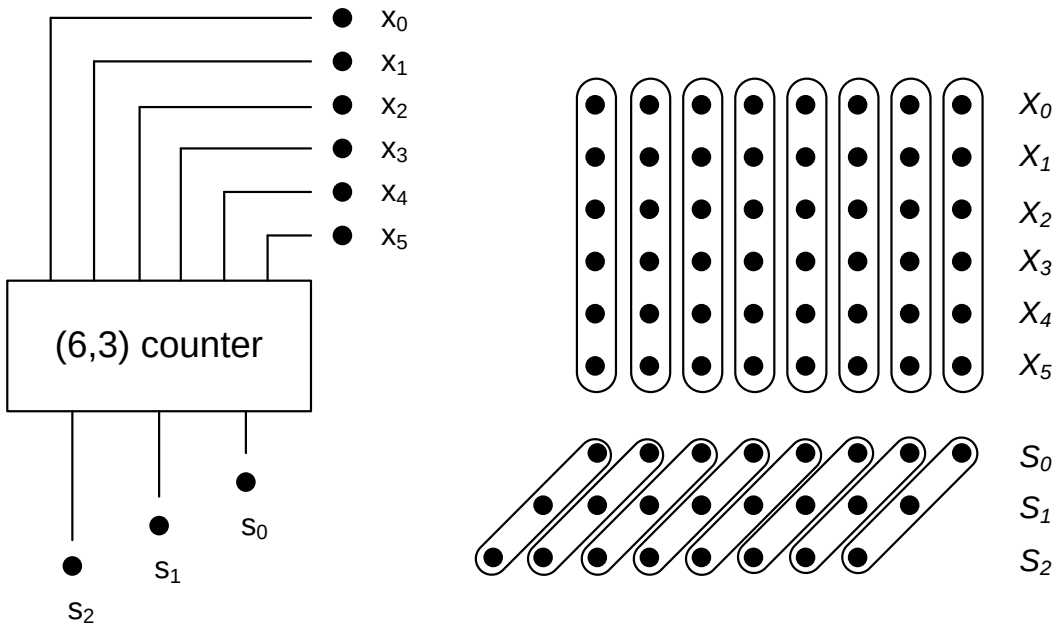
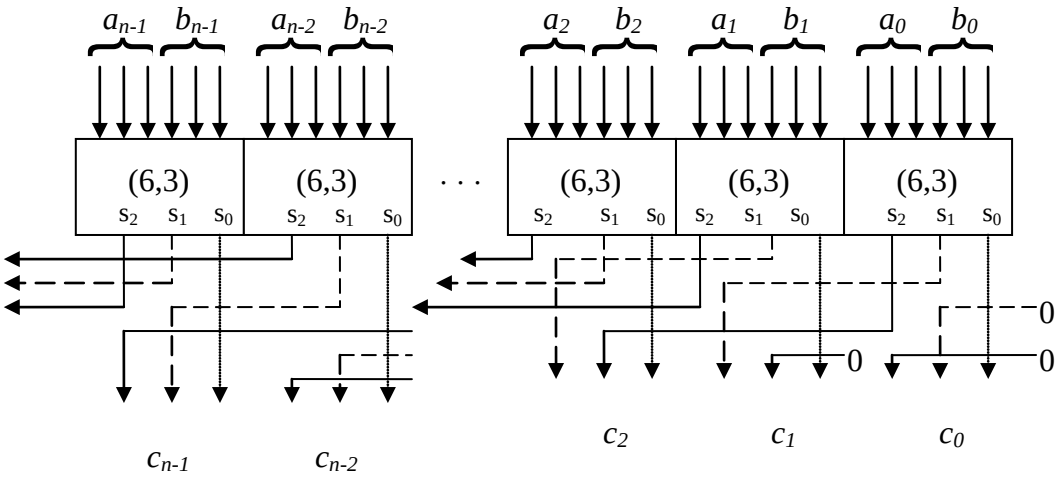



Figure 5.6. (6,3) counter: (a) single bit structure (b) addition of multiple operands using counters

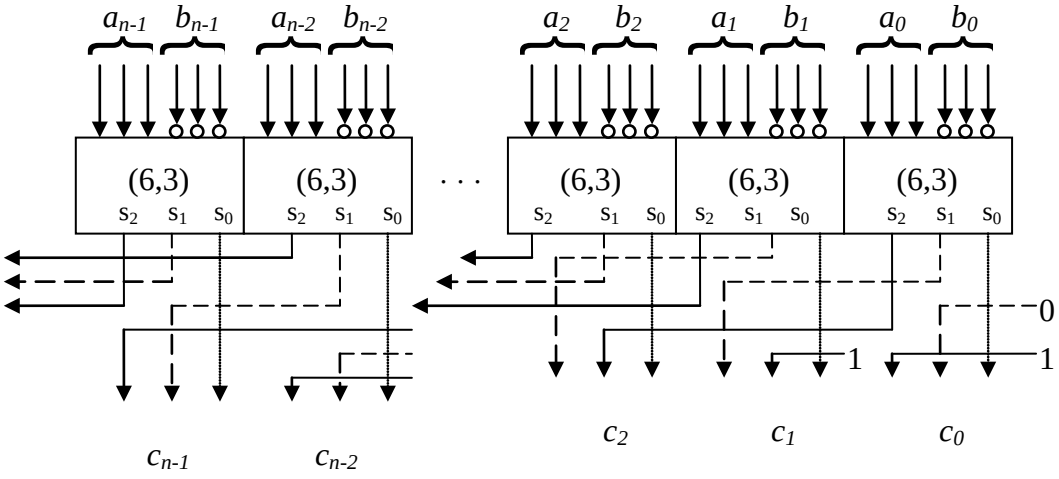
```
module six_three (x, sum);

input [5:0] x;
output [2:0] sum;
reg [2:0] temp;

always @ (x)

begin

   case (x)

   6'b000000: temp=0;        6'b001000: temp=1;
   6'b000001: temp=1;        6'b001001: temp=2;
   6'b000010: temp=1;        6'b001010: temp=2;
   6'b000011: temp=2;        6'b001011: temp=3;
   6'b000100: temp=1;        6'b001100: temp=2;
   6'b000101: temp=2;        6'b001101: temp=3;
   6'b000110: temp=2;        6'b001110: temp=3;
   6'b000111: temp=3;        6'b001111: temp=4;

   6'b010001: temp=2;        6'b011001: temp=3;
   6'b010010: temp=2;        6'b011010: temp=3;
   6'b010011: temp=3;        6'b011011: temp=4;
   6'b010100: temp=2;        6'b011100: temp=3;
   6'b010101: temp=3;        6'b011101: temp=4;
   6'b010110: temp=3;        6'b011110: temp=4;
   6'b010111: temp=4;        6'b011111: temp=5;

   6'b100000: temp=1;        6'b101000: temp=2;
   6'b100001: temp=2;        6'b101001: temp=3;
   6'b100010: temp=2;        6'b101010: temp=3;
   6'b100011: temp=3;        6'b101011: temp=4;
   6'b100100: temp=2;        6'b101100: temp=3;
   6'b100101: temp=3;        6'b101101: temp=4;
   6'b100110: temp=3;        6'b101110: temp=4;
   6'b100111: temp=4;        6'b101111: temp=5;

   6'b110000: temp=2;        6'b111000: temp=3;
   6'b110001: temp=3;        6'b111001: temp=4;
   6'b110010: temp=3;        6'b111010: temp=4;
   6'b110011: temp=4;        6'b111011: temp=5;
   6'b110100: temp=3;        6'b111100: temp=4;
   6'b110101: temp=4;        6'b111101: temp=5;
   6'b110110: temp=4;        6'b111110: temp=5;
   6'b110111: temp=5;        6'b111111: temp=6;
   default: temp=0;

endcase
end
    assign sum = temp[2:0];
endmodule
```

Figure 5.7. Verilog description of (6, 3) counter

In this redundant representation, each digit of the redundant variable is composed of three bits. Addition of two redundant variables with result $C = A + B$ can also be implemented using one stage of 6-input LUT structure as shown in Figure 5.8 (a). Each digit in this representation consists of three bits.




(a)




(b)

Figure 5.8. (a) Addition of two redundant variables; (b) subtraction

As can be seen in Figure 5.8 (a), an addition of two redundant numbers can be implemented by a single LUT sstage, consisting of three LUT units for each digit in the operation. As a result, two numbers can be added up with a single LUT delay. The subtraction operation is similar and can be depicted in Figure 5.8 (b).

## 5.2. Multiplying Accumulator Unit Design

By exploiting the 6-input LUT structure, a multiplying accumulator unit is designed. Here, a 12-bit by *M*-bit multiplying accumulator is designed. The proposed structure is suitable for many of the digital filtering and similar signal processing applications. In the multiplication phase, modified Booth encoding is employed to reduce the partial products of the multiplication [71, 72]. For modified Booth encoding, consider the multiplication of two integer numbers *A* and *B*, where *A* is multiplicand (*m*-bit) and *B* is multiplier (*n*-bit):

$$A = -a_{m-1}2^{m-1} + \sum_{j=0}^{n-2} a_j 2^j \tag{5.3}$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \tag{5.4}$$

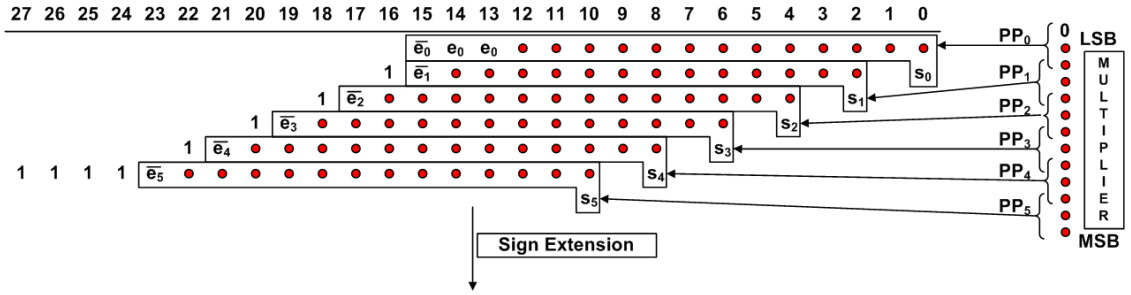
In the modified Booth encoding scheme, *B* becomes:

$$B = \sum_{i=0}^{\frac{n}{2}-1} m_i 2^{2i} = \sum_{i=0}^{\frac{n}{2}-1} (-2b_{2i+1} + b_{2i} + b_{2i-1})2^{2i} \tag{5.5}$$

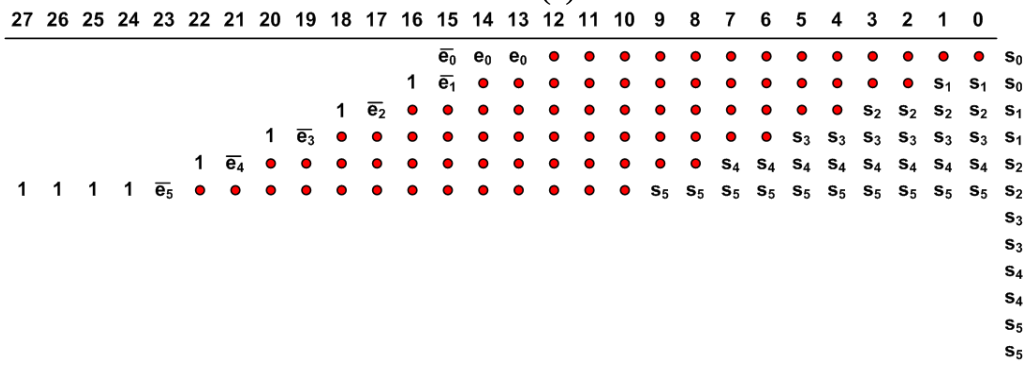
Here, $b_{-1} = 0$, $m_i \in \{-2, -1, 0, 1, 2\}$. According to the encoded results from *B*, the Booth selectors choose -2*A*, -*A*, 0, *A*, or 2*A* to generate the partial product rows. The Booth selection scheme is given in Table 5.1. When the *A* is negative, the two's complement of *A* is used (the bits of *A* are complemented the and the sign bit is added). The modified Booth encoding scheme for this work is shown in Figure 5.9 (a). The most significant bits remain all sign bits, and can be extended as shown in Figure 5.9 (a). The most significant digits are generated by $e_i = b_{2i+1} \oplus a_{m-1}$ where $b_{2i+1}$ is the sign bit of the modified Booth encoder and $a_{m-1}$ is the sign bit of the multiplicand and $\oplus$ is the exclusive-or operation [56, 73, 74]. Sign bit of each partial product, i.e. $s_i$ depending on the sign of *A* is added by the least significant digit of each partial product. In the application here, the multiplication size is 12x12 bits. The result is 24 bits where it is sign extended to 28 bits by adding extra ones in the most significant digits [75] as shown in Figure 5.9 (a) to prevent overflow in recursive multiply-accumulate operations.

Table 5.1. Modified Booth Encoding

| $b_{2i+1}$ | $b_{2i}$ | $b_{2i-1}$ | Partial Product | Booth Selector Output |
|---|---|---|---|---|
| 0 | 0 | 0 | *0* | 0 |
| 0 | 0 | 1 | *A* | $a_j$ |
| 0 | 1 | 0 | *A* | $a_j$ |
| 0 | 1 | 1 | *2A* | $a_{j-1}$ |
| 1 | 0 | 0 | *-2A* | $\overline{a_{j-1}}$ |
| 1 | 0 | 1 | *-A* | $\overline{a_j}$ |
| 1 | 1 | 0 | *-A* | $\overline{a_j}$ |
| 1 | 1 | 1 | *-0* | 0 |




(a)




(b)

Figure 5.9. (a) Partial product generating using modified Booth encoding; (b) backward sign extension

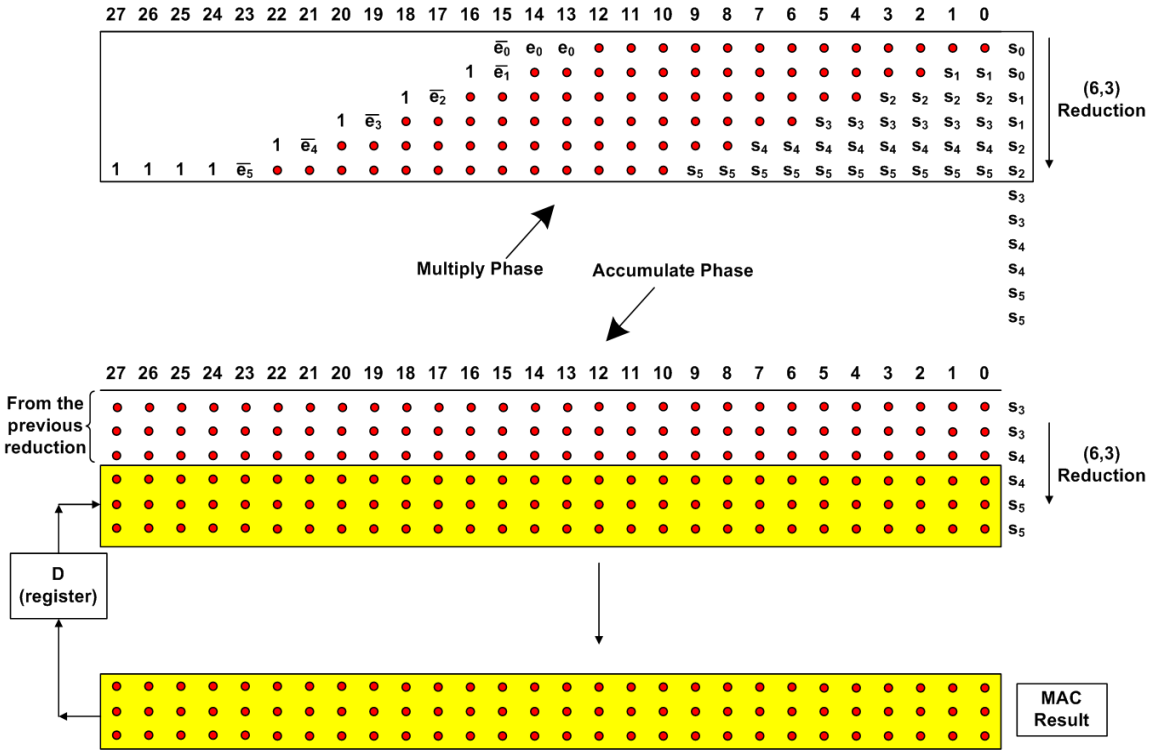



Figure 5.10. Multiply-Accumulate operation by implementing (6, 3) reduction

In modified Booth encoding, the number of partial products to be added is ($n$/2)+1. To reduce the products to (n/2) backward sign extension is employed in the implementation. A diagram for 12x12 bit multiplication scheme Booth encoding and backward sign extension is shown in Figure 5.9 (b). The sign digits are back-extended to the one right of the least significant digit. Although the digits to be reduced are doubled in the first digit, there is no product existing in this area. This is because the addition result will be even, and the LSB will be zero, i.e. empty (such as $[000]_2$, $[010]_2$, $[100]_2$, $[110]_2$). The 12 products can be added up in two phase scheme as shown in Figure 5.10.

After Booth encoding, partial products are fed into the (6, 3) counter for reduction. For the accumulation phase, again an (6, 3) counter is employed. As a result, multiplying accumulation process is completed using one Booth encoding and two stage (6, 3) counter stages each of which can be realized using single level 6-input LUT structures. The full multiply-add operation is shown in Figure 5.10.

Table 5.2. Performance comparison (6-input LUT)

| Design | # of Logic Units | # of Registers | Speed |
|---|---|---|---|
| Normal Multiply-Add | 155 | 52 | 174 MHz |
| Multiply-Add with normal Carry-Save output (*out=A+B*) | 234 | 77 | 273 MHz (57 % improvement) |
| Multiply-Add with proposed Carry-Save representation (*out=A+B+C*) | 243 | 107 | 334 MHz (92 % improvement) |

The performance comparison of the proposed design and other implementations are given in Table 5.2. The redundant output representation is converted to normal representation by using a three operand adder as *Sum = A + B + C*. In an FIR filter having *n* taps, the three operand addition is only required in the *n*-th phase and can be implemented efficiently by adding a single pipeline stage. The working diagram of the MAC unit is depicted in Figure 5.11 (a). The performance of the MAC unit can be analyzed by defining an *N* tap finite impulse response (FIR) filter where the inputs and outputs of the filter are defined as:

$$y(n) = \sum_{k=0}^{N-1} h_k \cdot x(n-k) \tag{5.6}$$

For an FIR filter implementation, the filter sampling frequency is calculated as $f_s = f_{clk}/n$. In Figure 5.11(a), redundant numbers are represented as bold lines. Multiple-MAC configuration of a system can be configured as shown in Figure 5.11(b). Here, for an *n*-tap FIR filtering scheme, the filter sampling frequency can be calculated as:

$$f_s = \frac{f_{clk}}{\lceil n/M \rceil} \tag{5.7}$$

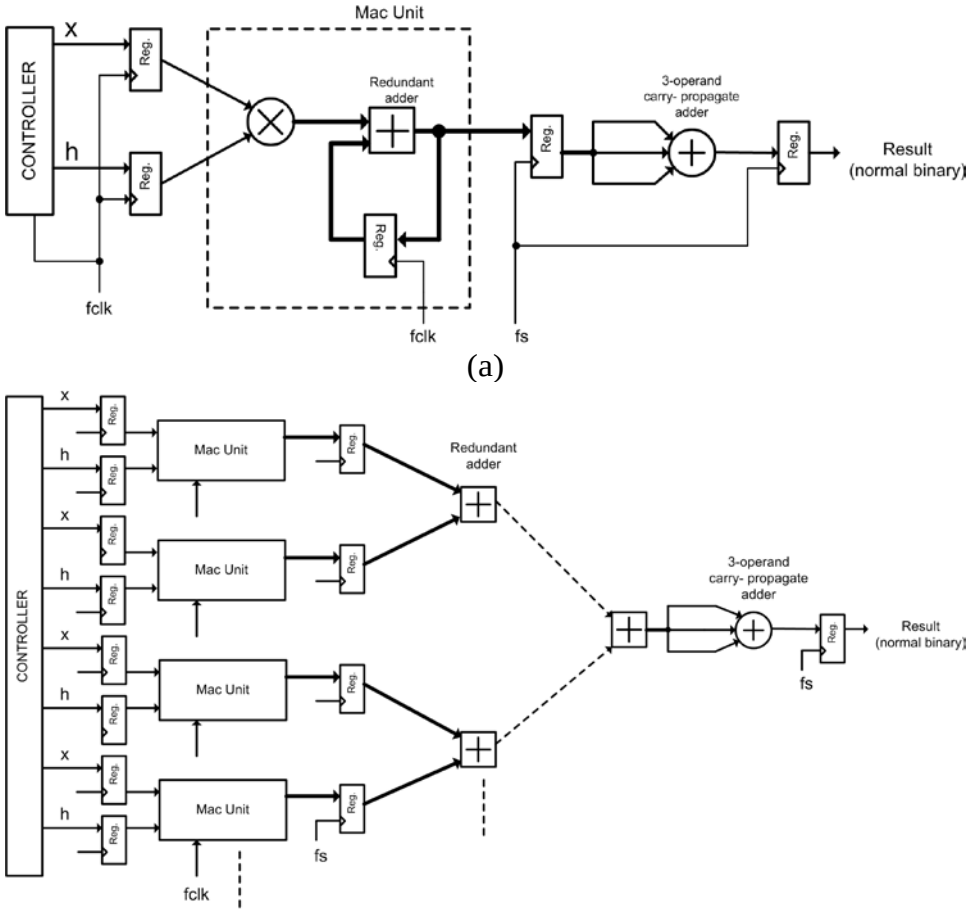



Figure 5.11. Multiply-accumulate unit application: (a) Single MAC; (b) multiple MAC

Table 5.3. Performance comparison (4-input LUT)

| Design | # of Logic Units | # of Registers | Speed |
|---|---|---|---|
| Normal Multiply-Add | 223 | 52 | 97 MHz |
| Multiply-Add with normal Carry-Save output (*out=A+B*) | 375 | 77 | 180 MHz (85 % improvement) |
| Multiply-Add with proposed Carry-Save representation (*out=A+B+C*) | 514 | 107 | 182 MHz (88 % improvement) |

where $n$ is the number of the taps in the filter, $M$ is the number of the MAC units, $f_{clk}$ is the system clock and $f_s$ is the sampling frequency of the signal. As an example, where a 15 tap filter with 4 MAC units is implemented with 300 MHz system clock, the throughput is

$$\frac{300}{\lceil 15/4 \rceil} = 75 \text{ MHz.}$$

The normal carry save and the proposed extra redundant carry save implementation are also implemented on a 4-input LUT based FPGA. The comparison results are given in Table 5.3. Both conventional and the proposed redundant implementation provides approximately the same speed improvement in this configuration. However, the proposed implementation provides dramatic performance increase when 6-input LUT devices are used, as shown in Table 5.2.

In section, a speed efficient multiplying accumulator unit is designed which is suitable for 6-input LUT based FPGA families. The multiply-add operation is handled only in three LUT critical path delay, one for Booth encoding, one for product reduction and one for addition with previous result. The implementation is suitable for high performance FIR filter and similar multiply-accumulate based signal processing blocks. Simulations show that this new implementation provides more than 90 per cent speed improvement over conventional implementations.

## 5.3. Fixed Coefficient FIR Filter Design

As mentioned in previous sections, in 6-input LUT based reconfigurable systems, (6, 3) counters are advised for the multi-operand addition schemes [65]. The counter circuits are generally used for the multi-operand addition and reduction of the partial product trees in the multiplier circuits. Here, each arithmetic element is based on (6, 3) counters and dramatic performance increase is achieved with increased hardware requirement.

In the proposed structure, the integer linear programming (ILP) model is given for the reduction of partial products for each coefficient of the filter. In each coefficient a maximum of six non-zero binary digits is allowed. In FIR filter schemes, a sparse

distribution of non-zero digits in each coefficient can be obtained via optimization. As a result, quite sharp filters can be implemented efficiently [76]. If no optimization is made over the coefficients, 12-bit coefficient word-length is also possible for the implementation. The reason is that, in canonic signed-digit (CSD) representation for any coefficient, number of non-zero digits in any number is at most half of the coefficient wordlength [77].

In the proposed system, the multiplication of each constant coefficient is made through (6, 3) counter arrays with the redundant outputs. In addition, backward sign-extension is implemented for the removal of extra sign-bit in the system. After the multiplication phase, the redundant addition operation is also implemented by a single stage (6, 3) counters with redundant outputs as well. As a result, a multiply-accumulate operation is handled in two stages.

FIR filters are used for shaping the input signal with the desired frequency response. Discrete time domain representation for an N-tap FIR filter is given as:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{5.8}$$

where $x$, $y$, $h$ are input, output and transfer function of the filter, respectively. The frequency response $H(\omega)$ of a linear-phase FIR filter with impulse response $h[n]$ and length $N$ is:

$$H(\omega) = \sum_{n=0}^{N-1} h[n]e^{-j\omega n} \tag{5.9}$$

The requirement for a linear filter is that, the filter coefficients must be symmetric or anti-symmetric [78]. As a result, the filter coefficients can also be written in terms of the amplitude $A(\omega)$ and phase terms as:

$$H(\omega) = A(\omega)e^{-j\omega(M-1)} \tag{5.10}$$

where M is approximately half the length of the filter tap count. *M* is calculated as:

$$M = \left\lfloor \frac{N}{2} \right\rfloor + 1 \tag{5.11}$$

The amplitude $A(\omega)$ is a real function of frequency given by

$$A(\omega) = \sum_{m=0}^{M-1} h[m] T_m(\omega) \tag{5.12}$$

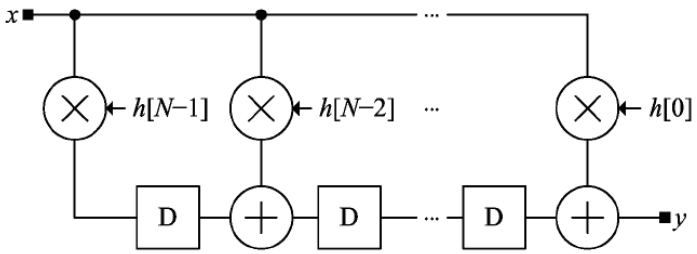
where $T_m(\omega)$ is a trigonometric function determined by the length and type of symmetry of the filter. The values of $T_m(\omega)$ for the four possible types of linear phase FIR filters are given in Table 5.4.

Table 5.4. $T_m(\omega)$ for different types of linear-phase FIR filters

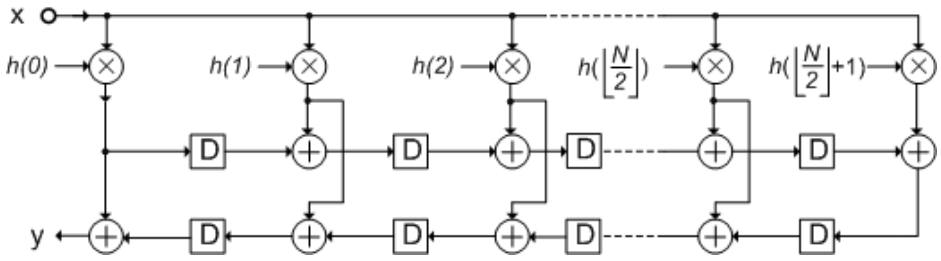
| Type | N | Symmetry | $T_m(\omega)$ |
|---|---|---|---|
| 1 | Odd | Symmetric | $\begin{cases} 1 & m = M-1 \\ 2\cos((M-m-1)\omega) & \text{otherwise} \end{cases}$ |
| 2 | Even | Symmetric | $2\cos((M-m-0.5)\omega)$ |
| 3 | Odd | anti-symmetric | $2\sin((M-m-1)\omega)$ |
| 4 | Even | anti-symmetric | $2\sin((M-m-0.5)\omega)$ |

The transposed FIR filter minimizes the critical path of the FIR operation to a single multiply-add operation as shown in Figure 5.12 (a). The linear phase implementation realized by symmetric or anti-symmetric coefficients for an odd length filter can be realized as in Fig 5.12 (b).

The filter coefficients can be optimized for better performance in terms of both frequency response and reduced hardware cost using optimization techniques. Using CSD notation for filter coefficients reduces filter implementation cost the number of non-zero digits which can be further minimized by Integer Linear Programming (ILP). If the number of non-zero digits is limited to six in each coefficient for the synthesized FIR filter, the multiplication where the coefficients are constant and data input is variable can be realized efficiently using (6, 3) counters. As stated previously, (6, 3) reduction can be done at a single stage in 6-input LUT based FPGAs.




(a)




(b)

Figure 5.12. FIR filter implementation: (a) Transposed form; (b) sharing the coefficients

For the synthesis of the desired FIR filter, let $D(\omega)$ be the desired frequency amplitude response of the filter. Let $\delta(\omega)$ be the approximation error of the synthesized filter, then the synthesized FIR filter response should satisfy:

$$|A(\omega) - D(\omega)| \leq \delta(\omega) \tag{5.13}$$

where $\omega \in [0, \pi]$ and $A(\omega)$ is the synthesized filter's frequency amplitude response, as defined previously. This formulization can be redefined for some disjoint frequency bands $\Omega_k \subset [0, \pi]$ with desired frequency response $D_k(\omega)$ and fixed error margin $\delta_k$ = 1, 2, … $K$ such that the equation can be rewritten as:

$$|A(\omega) - D_k(\omega)| \leq \delta_k(\omega) \tag{5.14}$$

Between each disjoint frequency bands, there is a transition band defined. For the transition bands, no constraints are defined. As an example, for a low-pass filter, two frequency bands for synthesis requirements can be defined such that, pass-band frequency defined as $\Omega_1 = [0, \omega_1]$ and $\Omega_2 = [\omega_2, \pi]$. There exists a transition band $\Omega^T = [\omega_1, \omega_2]$ in the low-pass example which has no constraint and left as a relaxation for the optimization realization. Frequency response characteristics of a low-pass FIR filter is given in Table 5.5. The frequency response of the filter is shown in Figure 5.13.

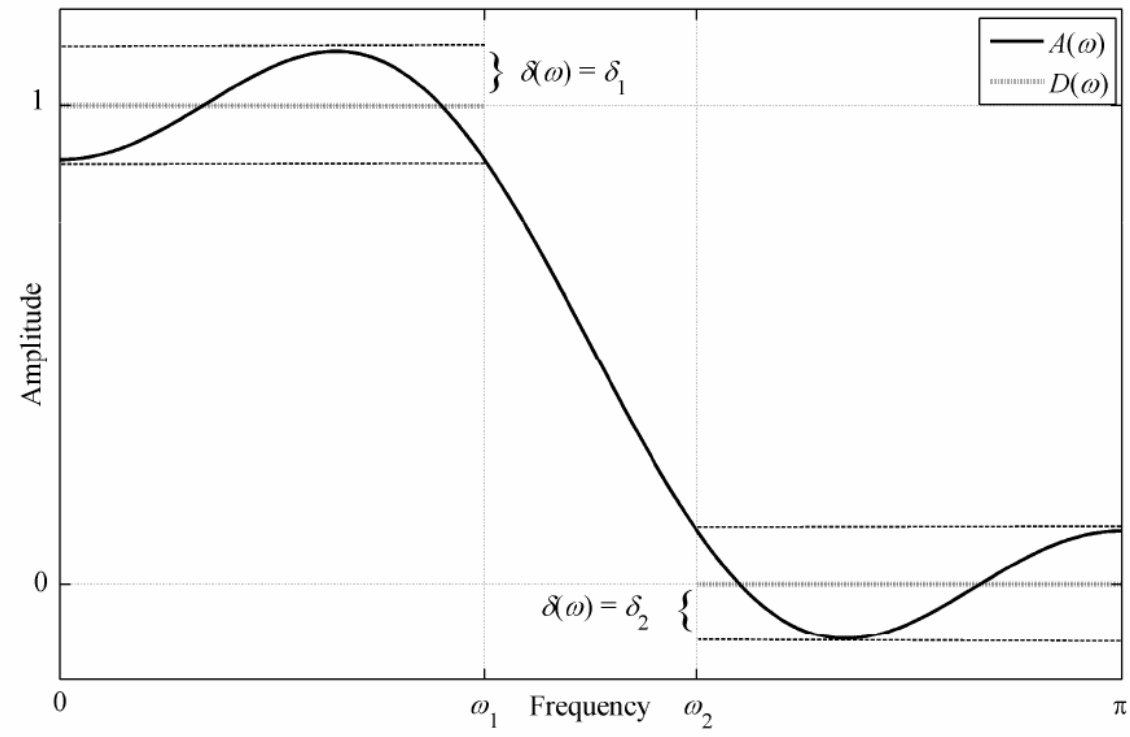



Figure 5.13. Frequency response characteristics of a low-pass FIR filter

Table 5.5. Frequency response characteristics of a low-pass FIR filter

| Band (k) | $\Omega_k$ | $D_k(\omega)$ | $\delta_k(\omega)$ |
|---|---|---|---|
| 1 | $[0, \omega_1]$ | 1 | $\delta_1$ |
| 2 | $[\omega_2, \pi]$ | 0 | $\delta_2$ |

Any type of filter realization such as low-pass band-pass, high-pass, or any combination of them can be modeled by adding proper frequency bands to the filter definition. The detailed problem formulation of the FIR filter design optimization model is given in Appendix A. The simplified FIR filter synthesis optimization definition is given as:

Minimize: $\sum_{i=0}^{M-1}\sum_{j=0}^{B-1}\left|x_{i,j}\right|$ (Number of nonzero digits in each coefficient)

Such that:

$\left|A(\omega)-D(\omega)\right| \le \delta(\omega)$

$\sum_{j=0}^{B-1}\left|x_{i,j}\right| \le 6$, $i$ = 0, 1, ... $M-1$. (Number of non-zero digits in each coefficient ≤ 6)

Here, $B$ is the wordlength, $x_{i,j}$ is $j$'th digit in $i$'th coefficient.

### 5.3.1. Realization of Multiply-add Operation

After the filter coefficients are calculated, the multiply and accumulate operations should be implemented according to the redundant arithmetic as defined previously. The fixed coefficient multiplication for the generated filter coefficients are realized by properly tiling the variables. The multiplication scheme for an example fixed 12-bit coefficient for the 12-bit variable word-length can be depicted in Figure 5.14 (a). In the example, the coefficient is given as ($0\bar{1}0101\bar{0}10\bar{1}$). Here, we rename each non-zero digit in the coefficient as $s_i$ , where $s_0 = -1$, $s_1 = 1$, $s_2 = -1$, $s_3 = 1$, $s_4 = 1$, and $s_5 = -1$ in the example. As mentioned before, according to the optimization algorithm, at most six non-

zero digits are allowed in any of the coefficients, which is always the general case for 12-bit CSD coded coefficients. The $s_i$ in each line shown in Figure 5.14 (a) also represents the sign bit of any non-zero bit for the corresponding coefficient.

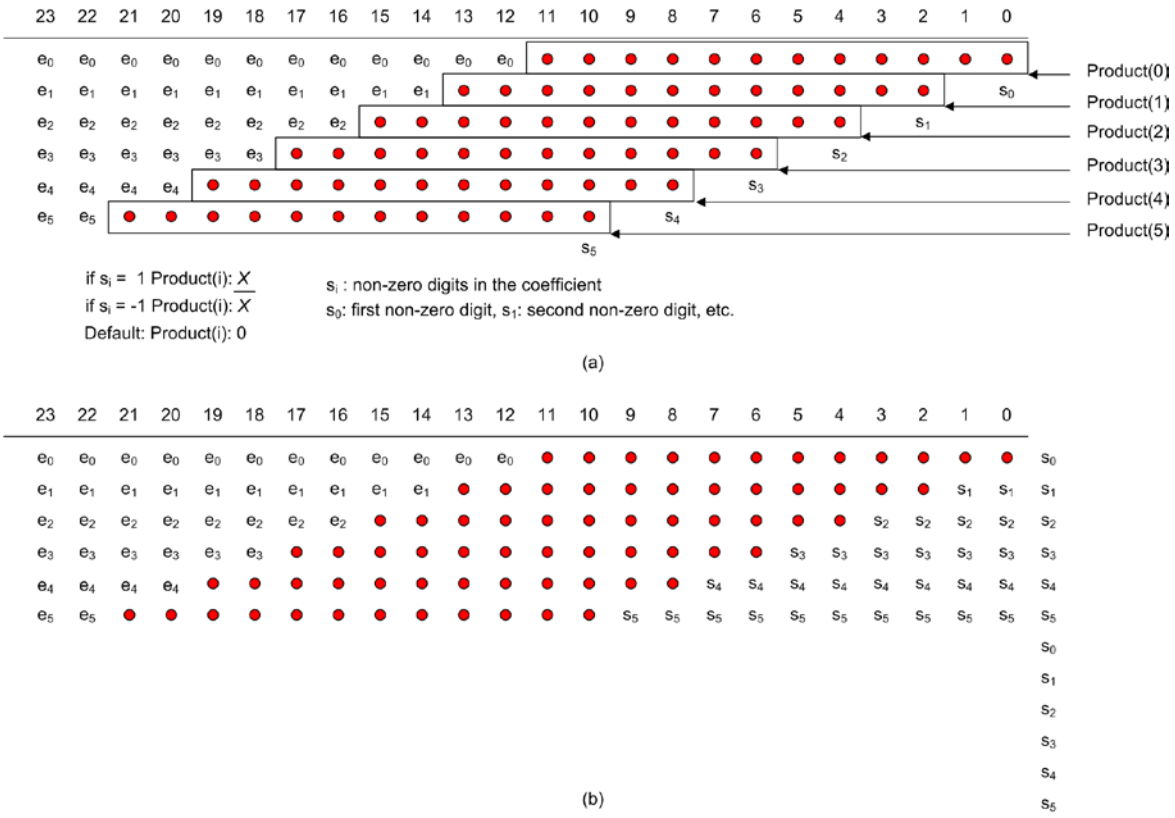



Figure 5.14. (a) Generation of the partial products; (b) backward-sign extension

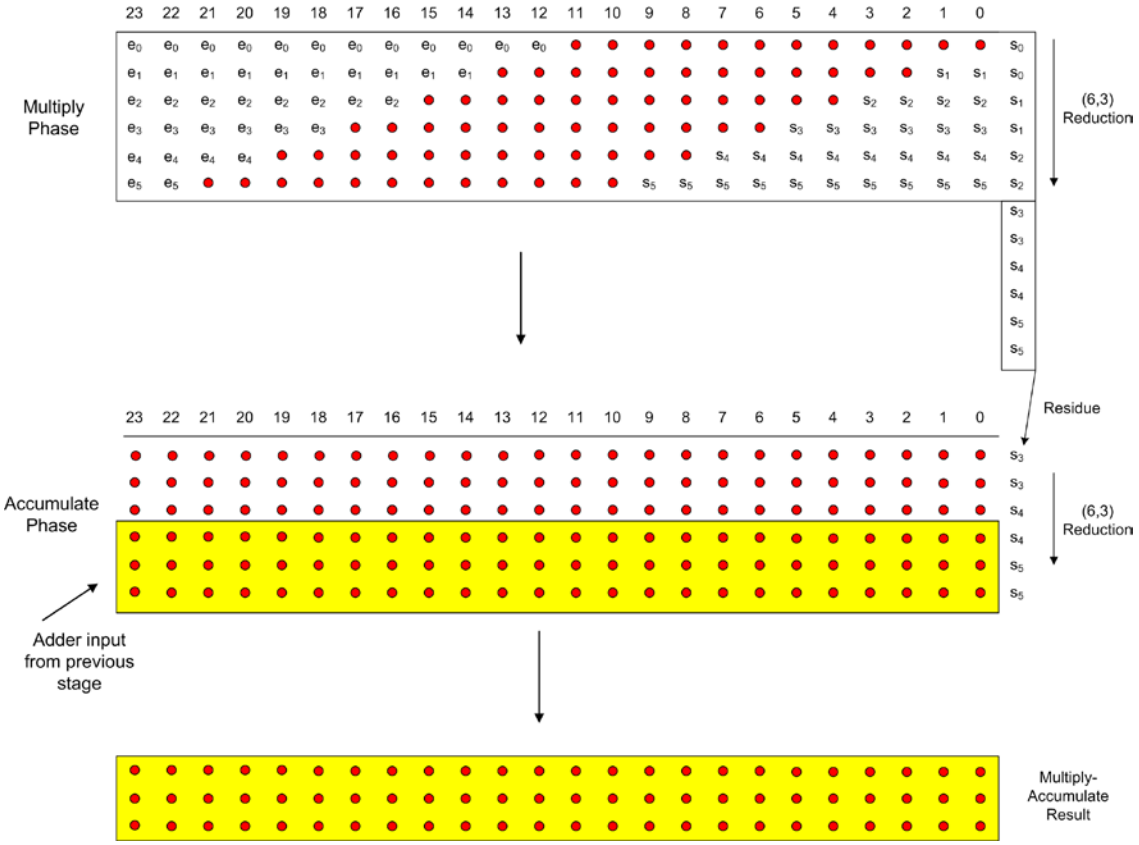
The diagram in Figure 5.14 (a) is an example and is not the generalized case. In the generalized case, the shift operations of the coefficients can be in arbitrary amounts, depending on the positions of the non-zero digits. Arithmetic shift operation is applied to the partial product by $(n - i)$ times. Here, $n$ is the coefficient word-length and $i$ is the digit number of the non-zero coefficient. In this scheme, $e_i$ is the most significant bit (MSB) of input variable $X$, if the related coefficient digit is 1. On the other hand, $e_i$ is the complement of MSB of input variable $X$ if the related coefficient digit is −1. In other words, $e_i = x_{MSB} \oplus s_i$. Here, $s_i$ is the sign of the incident non-zero coefficient digit. As shown in Figure 5.14 (a), the sign bit of the most significant digit of the coefficient increases the number of partial products by one , i.e. seven products exist in worst case and the partial products cannot be fed into a single stage (6, 3) counter. The problem is solved using backward sign extension. The (6, 3) reduction is done after this operation. Figure

5.14 (b) shows the application of backward sign extension. A verilog-like pseudo code for the generation of partial products and backward sign extension is shown in Figure 5.15. It should be noted that, coding of the representation in Figure 5.15 does not generate any extra hardware, rather it tiles the input data $X$ and $\overline{X}$ together with the sign bits of the coefficients to their proper placements. As a result, six partial products with residue output, representing the extra sign bits are generated. If there are less than six non-zero bit at the related coefficient, the remaining products left as zero, as can be analyzed from the code in Figure 5.15 as well.

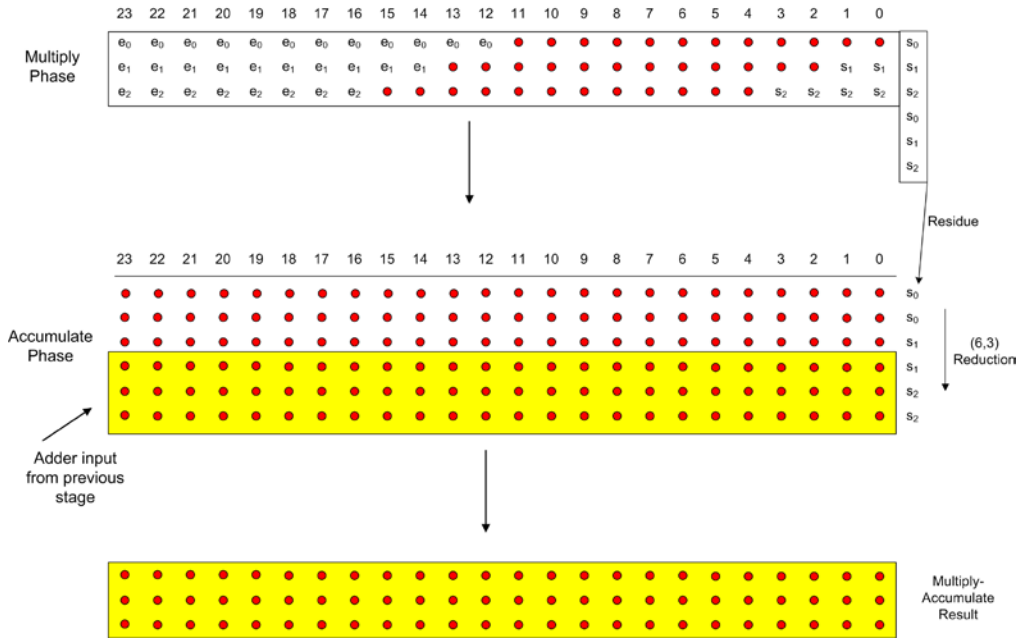
The multiplication of a fixed-coefficient with input data is accomplished by reduction of the partial products generated as shown in Figure 5.14 (b). For redundant carry-save representation, each number is a composition of three normal binary numbers. As a result, the six partial products should be reduced to three, to make the number compatible with double carry representation. Since there are six partial products generated, the multiplication with redundant outputs is accomplished as shown in Figure 5.16 (a). When the figure is explored, it can be seen that the multiplication phase consists of a single stage ($n$ + 1) digit (6, 3) reduction scheme. Here, $n$ is equal to the length of the coefficient and length of the variable, i.e. $n$ = (*coef_wordlength* + *data_wordlength*), which is 24 in the given example. The + 1 in the ($n$ + 1) definition is for the residue reduction as shown in Figure 5.14 (b) and Figure 5.16 (a). Still after the reduction, there are residue bits existing together with the multiplier result. The end result is obtained at the accumulation step. The accumulation step is also an ($n$+1) digit (6, 3) reduction scheme. The other input for the accumulation step comes from the previous tap of the designed filter. In the end, the total multiply-accumulate operation is accomplished in two stages of ($n$+1) digit (6, 3) counter arrays. At the multiply-add operation output, the result appears in double carry-save format, that is composed of three binary numbers. For the case there exists three or less than three non-zero digits in the coefficients, the multiply phase gets even simpler, as (6, 3) reduction for the multiply phase is removed. The multiplication phase for the multiply-add operation only consists of arithmetic shifts and sign bit padding operations which is shown in Figure 5.16 (b). In this case, the hardware cost for the multiply-accumulate phase is halved, which greatly reduces the hardware cost for the construction of the related filter tap. As a result, reduction of the non-zero digits in the coefficients plays an important role in the filter design procedure.

```
Module partial_product_align
    (X, Xbar, Product_0, Product_1, Product_2, Product_3, Product_4, Product_5, Residue);

// Inputs of the module
Input [data_wordlength-1:0] X;      // variable data input
Input [data_wordlength-1:0] Xbar; // inverse of variable data input
//

// Outputs of the module
Output [coef_wordlength+data_wordlength:0] Product_0;
Output [coef_wordlength+data_wordlength:0] Product_1;
Output [coef_wordlength+data_wordlength:0] Product_2;
Output [coef_wordlength+data_wordlength:0] Product_3;
Output [coef_wordlength+data_wordlength:0] Product_4;
Output [coef_wordlength+data_wordlength:0] Product_5;
Output [2:0] Residue;
//

// Parameterized inputs – coefficients, parameterized for each tap of the coefficient
Parameter [coef_wordlength-1:0] coefficient;            // one, if the related coefficient bit is non-zero
Parameter [coef_wordlength-1:0] coefficient_sign;    // coefficient bit is (-) if sign bit is 1.
//

Reg [coef_wordlength-1:0] zero_padding = 0;
Reg signed [coef_wordlength+data_wordlength-1:0] temp_product[5] = {0,0,0,0,0,0};
Reg temp_sign[5:0] = 0;

integer i, j, k;

j=0;
for (i = 0; i<coef_wordlength; i++)

{
if (coefficient[i]==1)
        {
                if (coefficient_sign[i]==1) temp_product[j] = {Xbar, zero_padding};
                else temp_product[j]= {X, zero_padding};

                temp_sign[j] = coefficient_sign[i];
                temp_product[j] = temp_product[j]>>>(coef_wordlength-i);    //arithmetic shift
                for (k=0; k<i; k++) temp_product[j][k] = coefficient_sign[i];    //sign bit padding
                j++;       // counter for the incident partial product alignment
        }

}

Product_0 = {temp_product[0], temp_sign[0]};
Product_1 = {temp_product[1], temp_sign[0]};
Product_2 = {temp_product[2], temp_sign[1]};
Product_3 = {temp_product[3], temp_sign[1]};
Product_4 = {temp_product[4], temp_sign[2]};
Product_5 = {temp_product[5], temp_sign[2]};
Residue [2:0] = {temp_sign[5], temp_sign[4], temp_sign[3]};

endmodule
```

Figure 5.15. Pseudo code representation of partial product representation




(a)




(b)

Figure 5.16. Multiply- add operation: (a) Generalized case; (b) up to 3 nonzero bits in the coefficient

### 5.3.2. Filter Implementation

For the performance measurements, two similar low-pass filters F1 and F2 are generated, the first one having tighter design requirements. The frequency response characteristics of these filters are given in Table 5.6. The passband and stopband frequencies are normalized to the sampling frequency in the example. The frequency response plots of these filters are shown in Figure 5.17.

Table 5.6. Frequency response characteristics of the example filters

| Filter | Passband Ripple (dB) | Passband Frequency | Stopband Attenuation | Stopband Frequency |
|---|---|---|---|---|
| F1 | 0.13 | 0.2 | -60 | 0.3 |
| F2 | 0.05 | 0.2 | -44 | 0.4 |

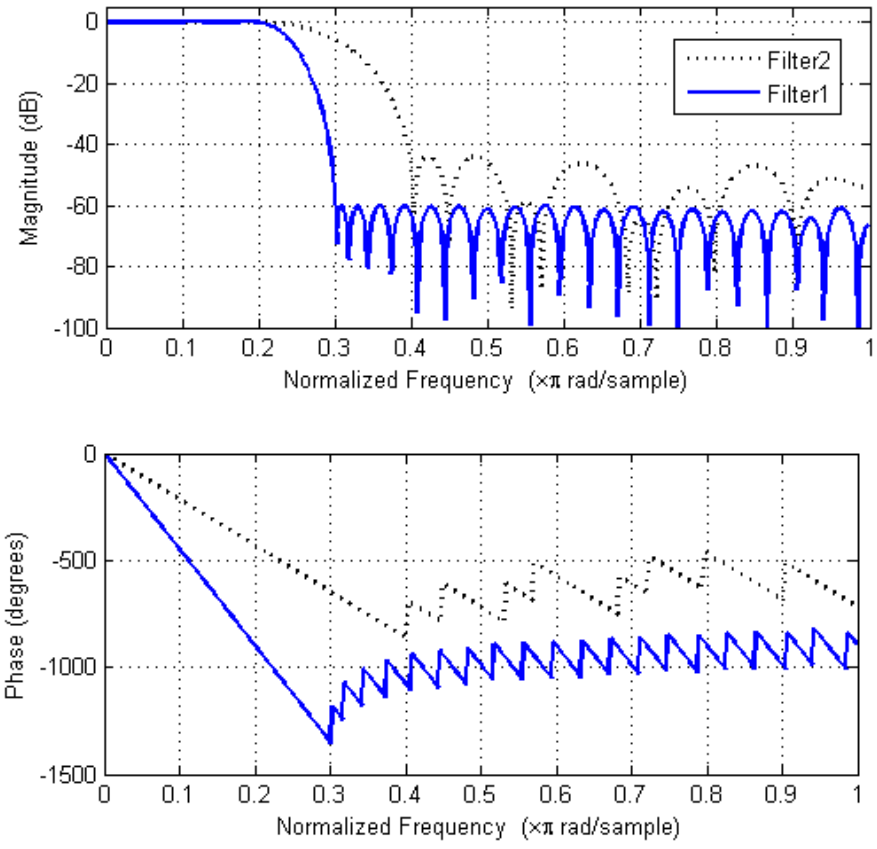



Figure 5.17. Frequency response characteristics of the example filters

The filters are synthesized according to the filter synthesis optimization rules as defined in Appendix A. The Filter 1 design workspace is tight and it is synthesized with 51 taps and 16 digit coefficients. The Filter 2 could be synthesized with 25 taps and 12 digit coefficients. The synthesized coefficients for Filter 1 and Filter 2 are given in Table 5.7 and Table 5.8, respectively.

Table 5.7. Coefficients of Filter 1

| **COEFFICIENTS OF FILTER F2 WITH *N*=25, *B*=16** | |
|---|---|
| $h(0)= -2^{-9} + 2^{-11} + 2^{-13} + 2^{-15}$ | $h(13)= -2^{-7} - 2^{-9} + 2^{-11} - 2^{-13}$ |
| $h(1)= -2^{-9} - 2^{-11}$ (*) | $h(14)= 2^{-6} + 2^{-8} – 2^{-11} - 2^{-14}$ |
| $h(2)= -2^{-8} + 2^{-10} - 2^{-13} - 2^{-15}$ | $h(15)= 2^{-5} - 2^{-7} + 2^{-12} + 2^{-15}$ |
| $h(3)= -2^{-9} - 2^{-11} + 2^{-13} - 2^{-15}$ | $h(16)= 2^{-6} - 2^{-10} – 2^{-13}$ (*) |
| $h(4)= 2^{-12}$ (*) | $h(17)= -2^{-7} + 2^{-15}$ (*) |
| $h(5)= 2^{-8} + 2^{-15}$ (*) | $h(18)= -2^{-5} - 2^{-9} – 2^{-11} - 2^{-13}$ |
| $h(6)= 2^{-7} – 2^{-10}$ (*) | $h(19)= -2^{-4} + 2^{-6} - 2^{-9} + 2^{-11}$ |
| $h(7)= 2^{-7} – 2^{-10} - 2^{-13}$ (*) | $h(20)= -2^{-5} - 2^{-7} + 2^{-9} + 2^{-11}$ |
| $h(8)= 2^{-9} + 2^{-11} + 2^{-15}$ (*) | $h(21)= 2^{-7}$ (*) |
| $h(9)= -2^{-8} - 2^{-10} + 2^{-12} - 2^{-14}$ | $h(22)= 2^{-4} + 2^{-6} + 2^{-11}$ (*) |
| $h(10)= -2^{-6} + 2^{-8} + 2^{-11}$ (*) | $h(23)= 2^{-3} + 2^{-5} + 2^{-11} + 2^{-15}$ |
| $h(11)= -2^{-6} + 2^{-8} - 2^{-10} + 2^{-15}$ | $h(24)= 2^{-2} - 2^{-5} - 2^{-10} - 2^{-15}$ |
| $h(12)= -2^{-7} + 2^{-9} - 2^{-11}$ (*) | $h(25)= 2^{-2} - 2^{-7} - 2^{-9} + 2^{-11}$ |
| $h(n)=h(50-n)$ for n=26, 27, …, 50 | |

Table 5.8. Coefficients of Filter 2

| **COEFFICIENTS OF FILTER F1 WITH *N*=25, *B*=12** | |
|---|---|
| $h(0)= -2^{-8}$ (*) | $h(7)= -2^{-4} + 2^{-6} - 2^{-8} + 2^{-10}$ |
| $h(1)= -2^{-7} + 2^{-9}$ (*) | $h(8)= -2^{-5} - 2^{-7} - 2^{-10}$ (*) |
| $h(2)= -2^{-11}$ (*) | $h(9)= 2^{-5} - 2^{-9}$ (*) |
| $h(3)= 2^{-6} – 2^{-8} - 2^{-11}$ (*) | $h(10)= 2^{-3} + 2^{-6} + 2^{-8}$ (*) |
| $h(4)= 2^{-6} + 2^{-8}$ (*) | $h(11)= 2^{-2} + 2^{-8} + 2^{-10}$ (*) |
| $h(5)= 2^{-7} + 2^{-10}$ (*) | $h(12)= 2^{-2} + 2^{-4} - 2^{-6} + 2^{-8}$ |
| $h(6)= -2^{-6} - 2^{-8} - 2^{-10}$ (*) | |
| $h(n)=h(24-n)$ for n=13, 14, …, 24 | |

The representation of the multiply-accumulate operation can be depicted in Figure 5.18. As the figure reveals, the redundant output of the system is converted to normal binary by insertion of a three operand adder circuit after the last tap of the multiply-add operation.

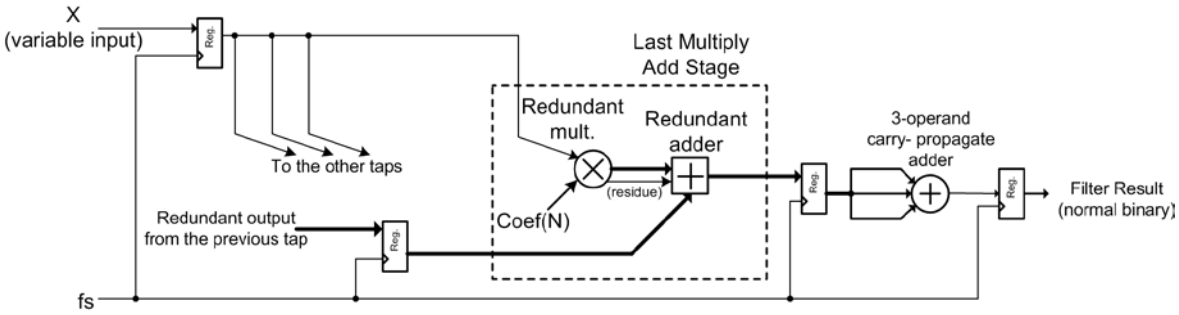



Figure 5.18. Representation of multiply-add operation and conversion to normal binary

The designed filters are realized by three methods, namely, using carry-propagate arithmetic fixed coefficient multipliers, using firm multipliers, and using the proposed method. Each of the implementations is synthesized using Altera Quartus II software. The comparison of hardware cost and maximum operating speed is given in Table 5.9. The performance comparison of the filter implementations are shown in Figure 5.19.

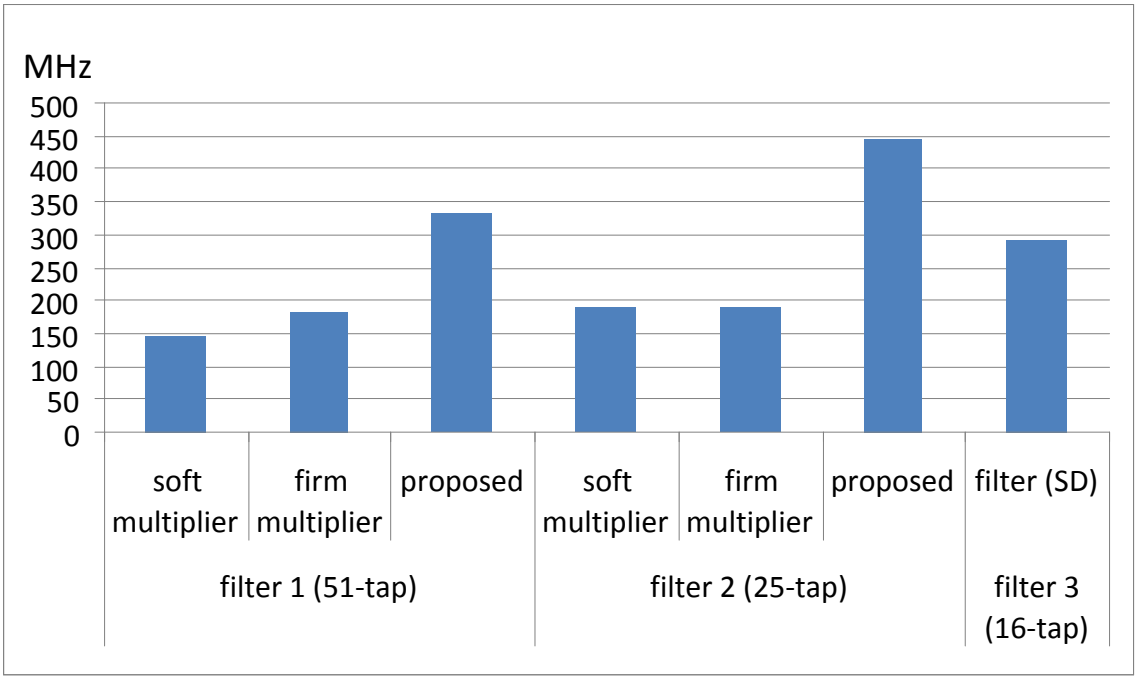



Figure 5.19. Performance comparison of the filter implementations

Table 5.9. Comparison of filter implementation schemes

| Design | Implementation | Hardware Cost | | Maximum Speed (MHz) |
|---|---|---|---|---|
| | | **ALUT + DSP Blocks** | **Register Count** | |
| Filter 1 | Fixed–coefficient soft multiplier | 1982 ALUT | 1404 | 147.99 |
| Filter 1 | Firm multiplier | 1400 ALUT + 52 DSP Blocks | 1428 | 181.26 |
| **Filter 1** | **Proposed method** | **4392 ALUT** | **3864** | **331.24** |
| Filter 2 | Fixed–coefficient soft multiplier | 814 ALUT | 576 | 188.11 |
| Filter 2 | Firm multiplier | 576 ALUT + 26 DSP Blocks | 600 | 189.54 |
| **Filter 2** | **Proposed method** | **1515 ALUT** | **1491** | **446.03** |
| Filter 3 | Method proposed in [70] | NA | NA | 293 |

In this section, a fixed coefficient filter implementation methodology suitable for 6-input LUT based FPGAs is presented. For the arithmetic operations, double carry-save arithmetic is suggested. Using the proposed scheme, the critical path for each multiply-add operation is reduced to only two LUT cascades, one for multiply and the other for the accumulate operation. The proposed implementation resulted in more than 100 per cent speed improvement over conventional soft multiplier fixed-coefficient multiplication based FIR filtering schemes.

# 6. CONCLUSION

In this thesis, a composition of multi-valued circuits and redundant number system implementations are presented. In the first part of the thesis, the focus has been multi-operand addition since it is one of the basic arithmetic blocks of multipliers and general multi-operand addition stages such as multiplier-less digital filter structures. Multi-valued circuit solutions to these structures are given for compact arithmetic structures. Current-mode design of the circuits provides analog friendly design opportunity and constant power consumption which can be adjusted for low power or high performance options. The remaining arithmetic blocks can be easily interfaced with normal binary or source-coupled binary circuits for complete design of a total digital system.

Various multi-valued logic circuit implementations for multi-operand addition are proposed in Chapter 4. First, signed-digit implementation of a multi-operand addition circuit is proposed. The signed-digit system provided for simultaneous addition of six input operands. In the end, an array of multi-valued to binary conversion is required for interfacing the circuit with binary logic. The proposed system in Section 4.1 is unique for implementing signed-digit multi-operand addition in multi-valued fashion. The circuit presented in Section 4.2.1 is an alternative implementation for multi-operand addition without signed-digit output. Another multi-valued circuit for multi-operand addition and multiplication is the multi-valued (7, 3) counter circuit presented in Section 4.2.2. The proposed circuit has binary inputs and binary outputs and the internal operations are made through multi-valued circuits. The proposed circuit in Section 4.2.2 has self restored outputs and this property provides a simpler and more compact design. The circuit has been synthesized with less active elements compared to binary implementations.

The current-mode circuits provide alternative solution for analog friendly design and lower power is achieved at high frequencies. However, constant current consumption is also a drawback since the circuit consumes power all the time regardless of switching activity. Moreover, reducing the supply voltage for static CMOS circuits also provided lower power compared to current-mode design. As a result, power efficiency is not a clear result for the current-mode circuits.

Throughout the thesis, extensive search on suitable circuit topologies for multiple-valued arithmetic circuits is accomplished. Depending on the experience of various circuit topology simulations, it can be said that, the main obstacle of the current-mode systems is current copying mechanisms. Copying currents from one node to another using current mirrors resulted slower operation as well as caused increased power consumption, since the current values are doubled in the current mirrors. The multi-operand circuit proposed in Section 4.2.2 has superior performance to other classical multi-valued circuits since current copying circuits are avoided. Nevertheless, multi-valued addition results are resolved using source-coupled comparator circuits at each level, rather than copying each current to next levels. The multi-operand addition circuit proposed in Section 4.2.2 requires less transistor count compared to SCL equivalent. However, in the whole system design, it reveals that the area reduction is not very significant. Therefore, from multi-valued circuit design perspective, not much advantage is achieved compared to the conventional circuit topologies.

The main obstacle about designing multi-valued systems in current-mode is the speed limitation of the current-mode systems, where, the performance is limited to the current source of the related block. In standard CMOS binary logic, the system is not limited to a current source, and the transistors switch as fast as possible. To relax this issue, design techniques similar to source-coupled binary logic systems is employed, where; at least the voltage swings are limited to a defined value rather than being $V_{DD}$ of the system. Limiting the logic swing increases the performance of the system. As a result, comparable results with binary source-coupled systems are achieved under same power consumption, where the proposed circuits required less active elements, because of the multi-valued design methodology.

Further study for the multi-operand multi-valued logic systems can be made in the area of very low power systems for low supply voltage levels and nanowatt designs providing small die size, suitable for body implants and similar applications of digital signal processing systems.

Although some novel topologies are proposed for multi-operand addition, MVL designs still suffer from mismatch effects, lower noise margins and complicated design

issues. Mismatch effects will be more significant for lower geometries. The future study should consider these problems when dealing with MVL design. The fabrication of the proposed architectures also remains as future study.

The second part of the work presented in Chapter 5 is related to reconfigurable computing. In this part, FPGA implementations of high performance addition and multiplication using alternative counter circuits and redundant number implementations are explored. An extra redundant addition scheme is proposed for the efficient redundant arithmetic, where an addition takes only a single LUT block delay. As conventional redundant architectures require at least two LUT delays, the proposed architecture provides an important speed advantage. The architecture is especially advantageous for the operations that require recursive addition, such as convolution operators in digital signal processing or matrix multiplication blocks. A multiply-accumulate unit has been designed using the redundant arithmetic schemes proposed. The multiply-accumulate unit provided more than 90 per cent speed improvement over conventional carry-propagate arithmetic structures. Moreover, the system provided 57 per cent speed improvement over conventional carry-save based redundant applications.

Another application of the proposed redundant arithmetic for the FPGA systems was the fixed-coefficient filter implementations. Filters coefficient are synthesized by using ILP formulation and implemented using the proposed double carry-save arithmetic. More than 100 per cent speed improvement has been achieved compared to conventional carry-propagate arithmetic implementations.

For the FPGA arithmetic, various application areas should be searched wherever redundant arithmetic is useful. As the required LUT cascades is minimized, arithmetic applications providing very high performance with least amount of pipeline stages can be realized using the proposed double carry-save arithmetic implementation on 6-input LUT based FPGA systems. Highest performance can be achieved by implementing the proposed extra redundant arithmetic design.

# APPENDIX A: PROBLEM FORMULATION OF THE FIXED-COEFFICIENT FIR FILTER SYNTHESIS

Here, the detailed problem formulation of fixed coefficient FIR filter synthesis is given, where the basic problem definition was given in Section 5.3. This Appendix should be used as a supplemental for the mentioned FIR synthesis section of the thesis and the Equations 5.8 through 5.14 should be explored for understanding the formulation presented here.

Each coefficient $h[i]$ of an $N$ tap FIR filter with $B$-bit word-length can be written for the CSD representation as:

$$h[i] = \sum_{j=0}^{B-1} x_{i,j} 2^{-j} \tag{A.1}$$

where $x_{i,j} \in \{-1,0,1\}$ corresponds to the $j$'th bit of coefficient $h[i]$. In this representation, $j$ = 0 is the most significant digit. For the ILP formulization, since in CSD representation two adjacent non-zero digits does not exist, the constraint

$$\left|x_{i,j}\right| + \left|x_{i,j+1}\right| \leq 1 \tag{A.2}$$

should be added in each coefficient. If the cost function is the number of the non-zero digits in each coefficient, the cost function for each coefficient can be modeled as:

$$\sum_{j=0}^{B-1} \left|x_{i,j}\right| \tag{A.3}$$

The filter optimization function for efficient synthesis on 6-input LUT devices can be modeled as:

Minimize $\sum_{i=0}^{M-1}\sum_{j=0}^{B-1}\left|x_{i,j}\right|$

Such that

$$\left|\sum_{i=0}^{M-1}\left(\sum_{j=0}^{B-1}x_{i,j}2^{-j}\right)T_i(\omega)-D(\omega)\right|\le\delta(\omega) \tag{A.4}$$

and

$\sum_{j=0}^{B-1}\left|x_{i,j}\right|\le 6$, $i$ = 0, 1, ... $M-1$.

and $\left|x_{i,j}\right|+\left|x_{i,j+1}\right|\le 1$ for each coefficient. For modeling the problem as a linear problem, each coefficient digit $x_{i,j}$ can be rewritten as:

$$x_{i,j}=x_{i,j}^{+}-x_{i,j}^{-} \tag{A.5}$$

since each coefficient $x_{i,j}\in\{-1,0,1\}$ and $x_{i,j}^{+},x_{i,j}^{-}\in\{0,1\}$. The optimization problem can be rewritten as:

Minimize: $\sum_{i=0}^{M-1}\sum_{j=0}^{B-1}(x_{i,j}^{+}-x_{i,j}^{-})$

Such that:

$$\left|\sum_{i=0}^{M-1}\left(\sum_{j=0}^{B-1}(x_{i,j}^{+}-x_{i,j}^{-})2^{-j}\right)T_i(\omega)-D(\omega)\right|\le\delta(\omega),\ \omega\in[0,\pi] \tag{A.6}$$

$x_{i,j}^{+}+x_{i,j}^{-}+x_{i,j+1}^{+}+x_{i,j+1}^{-}\le 1$ for $j$ = 0, 1, $B$-2.

$\sum_{j=0}^{B-1}(x_{i,j}^{+}+x_{i,j}^{-})\le 6$.

The absolute value operator does not harm the linear problem definition since the constraint can be rewritten as:

$$D(\omega)-\delta(\omega)\le\sum_{i=0}^{M-1}\left(\sum_{j=0}^{B-1}(x_{i,j}^{+}-x_{i,j}^{-})2^{-j}\right)T_i(\omega)\le D(\omega)+\delta(\omega) \tag{A.7}$$

# REFERENCES


1. Parhami, B., “Generalized Signed-digit Number Systems: a Unifying Framework for Redundant Number Representations”, *IEEE Trans. on Compute*rs, Vol. 39, pp. 89-98, 1990.

2. Koren, I., *Computer Arithmetic Algorithms*, AK Peters, Natick, MA, 2002.

3. Ercegovac, M. and T. Lang, *Digital Arithmetic*, Morgan Kauffman, 2004.

4. Noll, T. G., “Carry-Save Architectures for High-Speed Digital Signal Processing”, *Journal of VLSI Signal Processing*, Vol. 3, pp. 121-140, 1991.

5. Kawahito, S. et al., “A 32 x 32-bit Multiplier Using Multiple-Valued MOS Current-Mode Circuits”, *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 1, pp. 124– 132, Feb. 1988.

6. Current, K. W., “Current-Mode CMOS Multiple-Valued Logic Circuits”, *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 2, Feb. 1994.

7. Hanyu, T. and M. Kameyama, “A 200 MHz Pipelined Multiplier Using 1.5 V-Supply Multiple-Valued MOS Current-Mode Circuits with Dual-Rail Source-Coupled Logic”, *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 11, pp. 1239– 1245, Nov. 1995.

8. Ike, T., T. Hanyu and M. Kameyama, “Fully Source-Coupled Logic Based Multiple-Valued VLSI”, *ISMVL 2002 Proceedings 32nd IEEE International Symposium on Multiple-Valued Logic*, pp. 270–275, May 2002.

9. Temel, T., A. Morgul and N. Aydin, “Signed Higher-radix Full-adder Algorithm and Implementation with Current-mode Multi-valued Logic Circuits”, *IEE Proceedings on Circuits, Devices and Systems*, pp. 489-496, Oct. 2006.

10. De, V. and S. Borkar, “Technology and design challenges for low power and high performance”, *Int. Symp. on Low Power Electronics and Design*, pp. 163-168, Aug. 1999.

11. Narenda, S., V. De, S. Borkar, D. A. Antoniadis and A. P. Chandrakasan, “Full-Chip Subthreshold Leakage Power Prediction and Reduction Techniques for Sub-0.18-μm CMOS”, *IEEE Journal of Solid-State Circuits*, Vol. 39, No. 2, Feb. 2004.

12. Rabaey, J. M., A. Chandrakasan and B. Nikolic, Digital *Integrated Circuits: A Design Perspective*, Prentice Hall, 2003.

13. Avizienis, A., “Signed-digit Number Representations for Fast Parallel Arithmetic”, *IRE Trans. Electron. Comput.*, pp. 389-400, 1961.

14. Gonzales, A. F. and P. Mazumder, “Redundant Arithmetic, Algorithms and Implementations”, *Integration, the VLSI journal*, Vol.30, pp. 13-53, 2000.

15. Takagi, N., H. Yasuura and S. Yajima, “High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree”, *IEEE Trans. on Computers*, Vol. 34, pp. 789-796, Sept. 1985.

16. Thornton, M., “Signed Binary Addition Circuitry with Inherent Even Parity Outputs”, *IEEE Transactions on Computers*, Vol. 46, No. 7, pp. 811-816, July 1997.

17. Thornton, M., “A Signed Binary Addition Circuit Based on an Alternative Class of Addition Tables”, *Computers & Electrical Engineering*, Vol. 29, No. 2, pp. 303-315, March 2003.

18. Kuninobu, S., T. Nishiyama, H. Edamatsu, T. Taniguchi and N. Takagi, “Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation”, *Proceedings on Eighth Symposium on Computer Arithmetic*, pp. 80-86, 1987.

19. Harata, Y., Y. Nakamura, H. Nagase, M. Takigawa and N. Takagi, “A High Speed Multiplier Using a Redundant Binary Adder Tree”, *IEEE Journal of Solid-State Circuits*, Vol. SSC-22, pp. 28-34, Feb. 1987.

20. Rajashekhara, T. N. and O. Kal, “Fast Multiplier Design Using Redundant Signed-digit Number”, *International Journal of Electronics*, Vol. 69, pp. 359-368, 1990.

21. Srinivas, H. R. and K. K. Parhi, “A fast VLSI Adder Architecture”, *IEEE Journal of Solid-State Circuits*, Vol. 27, pp. 761-767, May 1992.

22. Yen, S. M., C. S. Laih, C. H. Chen and Y. Lee, “An Efficient Redundant-binary Number to Binary Number Converter”, *IEEE Journal of Solid-State Circuits,* Vol. 27, pp. 109-112, Jan. 1992.

23. Makino, H., Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara and K. Mashiko, “An 8.8-ns 54 x 54-bit Multiplier with High Speed Redundant Binary Architecture”, *IEEE Journal of Solid-State Circuits*, Vol. 31, pp. 773-783, 1996.

24. Wallace, C. S., “A Suggestion for a Fast Multiplier”, *IEEE Trans. Electron. Comput. EC-13,* 1964.

25. Zimmermann, R., *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*, PhD thesis, Swiss Federal Institute of Technology (ETH), 1997.

26. Miller, D. M. and M. A. Thornton, *Multiple Valued Logic: Concepts and Representations*, Morgan and Claypool Publishers, 2008.

27. Post, E. L., “Introduction to a General Theory of Elementary Propositons”, *Amer. J. of Math.*, Vol. 43, pp. 163-185, 1921.

28. Hassoun, S. and T. Sasao (editors), *Logic Synthesis and Verification*, Kluwer Academic Publishers, 2002.

29. Muzio, J. C. and T. C. Wesselkamper, *Multiple-Valued Switching Theory*, Adam Hilger, Bristol and Boston, 1986.

30. Jain, A. K., R. J. Bolton and M. H. Abd-El-Barr, "CMOS Multiple-Valued Logic Design – Part I: Circuit Implementation", *IEEE Trans. On Circuits and Systems I*, Vol. 40, No.8, pp. 503-514, Aug. 1993.

31. Jain, A. K., R. J. Bolton and M. H. Abd-El-Barr, "CMOS Multiple-Valued Logic Design – Part II: Functional Realization", *IEEE Trans. On Circuits and Systems I*, Vol. 40, No.8, pp. 515-522, Aug. 1993.

32. Temel, T., *Current-Mode CMOS Design of Multi-Valued Logic Circuits*, Ph.D. Thesis, Boğaziçi University, 2002.

33. Sasao, T., *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

34. Muzio, J. C. and D. M. Miller, "On the Minimization of Many-valued Functions", *Proceedings on 9th International Symposium Multiple-Valued Logic*, pp. 294-299, May 1979.

35. Rudell, R. and A. Sangiovanni-Vincentelli, "Multiple-valued Minimization for PLA Optimization", *IEEE Transactions on CAD*, Vol. CAD-5, No.9, pp. 727-750, Sept. 1987.

36. Lavagno, L., S. Malik, R. K. Brayton and A. Sangiovanni-Vincentelli, "MIS-MV: Optimization of Multi-level Logic with Multiple-valued Inputs", *Proc. Int. Conf. on Computer Aided Design*, pp. 560-563, Nov. 1990.

37. Dubrova, E. V., Y. Jiang and R. K. Brayton, "Minimization of Multiple-valued Functions in Post Algebra", Proc. *of Int. Workshop on Logic Synthesis*, pp. 132-138, June 2001.

38. Freitas, D. A. and K. W. Current, “A CMOS Current Comparator Circuit”, *Electron. Lett.*, Vol. 19, No. 17, pp. 695-697, Aug. 1983.

39. Kurisu, M. et. al., “2.8 Gb/s 176 mW byte-interleaved and 3.0 Gb/s 118 mW bit-interleaved 8:1 Multiplexers with a 0.15-μm CMOS Technology”, *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 12, Dec. 1996.

40. Jianhua, L. et. al., “Design Techniques of CMOS SCL Circuits for Gb/s Applications”, *IEEE Proceedings of 4th International Conference on ASIC*, pp. 559-562, Oct. 2001.

41. Srinivasan, V., D. S. Ha and J. B. Sulistyo, “Gigahertz-Range MCML Multiplier Architectures”, *ISCAS’04*, Vol. 2, pp. 785-788, May, 2004.

42. Musicer, J., *An Analysis of MOS Current Mode Logic for Low Power and High Performance Digital Logic*, M.S. Thesis, UC Berkeley, 2001.

43. Alioto, M. and G. Palumbo, *Modeling and Design of Bipolar and MOS Current-Mode Logic*, Kluwer Academic Publishers, 2006.

44. Freitas, D. A. and K. W. Current, “A Quaternary Logic Encoder-decoder Circuit Design Using CMOS”, *Proceedings of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 190-195, 1983.

45. Current, K. W., D. A. Freitas and F. A. Edwards, “CMOS Quaternary Threshold Logic Full Adder Circuits”, *Proceedings of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 144-151, 1985.

46. Kawahito, S., M. Kameyama and T. Higuchi, “VLSI Oriented Bi-directional Current-mode Arithmetic Circuits Based on the Radix-4 Signed-digit Number System”, *Proceedings of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 70-77, 1986.

47. Kawahito, S., M. Kameyama, T. Higuchi and H. Yamada, “A High-speed Compact Multiplier Based on Multiple-valued Bi-directional Current-mode Circuits”, *Proceedings of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 172-180, 1987.

48. Etiemble, D. and K. Navi, “A Basis for the Comparison of Binary and m-valued Current Mode Circuits: The Multioperand Addition with Redundant Number Systems”, *Proceedings of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 216-221, 1993.

49. Current, K. W., “Current-Mode CMOS Multiple-Valued Logic Circuits”, *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 2, pp. 95-107, Feb. 1994.

50. Temel, T. and A. Morgul, “Implementation of Multi-Valued Logic Gates Using Full Current-Mode CMOS Circuits”, *Analog Integrated Circuits and Signal Processing*, Kluwer Academic Publishers, Vol. 39, No.2, pp. 191-204, 2004.

51. Morgul, A. and T. Temel, “Current-mode Level Restoration: Circuit for Multi-valued Logic”, *IEE Electronic Letters*, Vol.41, No.5, pp. 230-240, March 2005.

52. Weinberger, A. and J. L. Smith, “A Logic for High-speed Addition”, *Nat. Bur. Stand., Circular 591, Sect. 1*, pp. 3-12, 1958.

53. Sklansky, J., “Conditional-Sum Addition Logic”, *IRE Trans. Electron. Comput., EC-9*, pp. 226-231, 1960.

54. Brent, R. and H. Kung, “A Regular Layout for Parallel Adders”, *IEEE Trans. on Computers*, Vol. C-31, No. 3, pp. 260-264, March 1982.

55. Kogge, P. and H. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations”, *IEEE Trans. on Computers*, Vol. C-22, No. 8, pp. 768-793, Aug. 1973.

56. Weste, N. H. E. and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Third Edition, Addison Wesley, 2005.

57. Razavi, B., *Design of CMOS Integrated Circuits*, McGraw-Hill, 2000.

58. Allstot, D. J., “A Precision Variable-Supply CMOS Comparator”, *IEEE Journal of Solid-State Circuits*, Vol. 17, Issue 6, pp. 1080-1087, Dec. 1982.

59. Uyemura, J., *CMOS Logic Circuit Design*, Springer, 1999.

60. Tajalli, A., Y. Leblebici and E.J. Brauer, ” Implementing Ultra-high-value Floating Tunable CMOS Resistors”, *IEE Electronic Letters*, Vol. 44, Issue 5, Feb. 2008.

61. Altera Corp., *Altera Stratix II Device Handbook*, http://www.altera.com, 2009.

62. Xilinx Inc., *Virtex-5 Family Overview*, http://www.xilinx.com, 2009.

63. Brown, S. and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, Second Edition, McGraw Hill, 2008.

64. Brown, S. and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, Vol. 13, No. 2, 1996.

65. Altera Corp., *Advanced Synthesis Cookbook: A Design Guide for Stratix II, Stratix III, and Stratix IV Devices*, http://www.altera.com, July 2009.

66. Hormigo, J., M. Ortiz, et. al., “Efficient Implementations of Carry-Save Adders in FPGAs”, *IEEE Int. Conference on Application-specific Systems, Architectures and Processors*, pp. 207-210, 2009.

67. Kamp, W. and A. BainbridgeSmith, “Multiply Accumulate Unit Optimized for Fast Dot-Product Evaluation”, *International Conference on Field-Programmable Technology*, pp. 349-352, 2007.

68. Parandeh-Afshar, H., P. Brisk and P. Ienne, “Improving Synthesis of Compressor Trees on FPGAs via Integer Linear Programming”, *Design, Automation and Test in Europe (DATE) Conference*, pp. 1256-1261, 2008.

69. Kamp, W., A. Bainbridge-Smith and M. Hayes, “Efficient Implementation of Fast Redundant Number Adders for Long Word-Lengths in FPGAs”, *International Conference on Field-Programmable Technology*, pp. 239-246, 2009.

70. Cardarilli, G.C., S. Pontarelli, M. Re and A. Salsano, “On the use of Signed Digit Arithmetic for the New 6-Inputs LUT Based FPGAs”, *International Conference on Electronics, Circuits and Systems*, pp.602-605, 2008.

71. Booth, A. D., “A Signed Binary Multiplication Technique”, *Quarterly J. of Mech. and Appl. Math.*, Vol. 4, pp. 236-240, 1951.

72. MacSorley, O.L., “High Speed Arithmetic in Binary Computers”, *Proc. IRE*, Vol. 49, pp. 67-91, 1961.

73. Salomon, O., J. M. Green and H. Klar, “General Algorithms for a Simplified Addition of 2’s Complement Numbers”, *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 7, pp. 839-844, July 1995.

74. Hsu, S. K., S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R. K. Krishnamurthy and S. Y. Borkar, “A 110 GOPS/W 16-bit Multiplier and Reconfigurable PLA Loop In 90-nm CMOS”, *IEEE Journal of Solid-State Circuits*, Vol. 41, No.1, Jan. 2006.

75. Gierenz, V., C. Panis and J. Nurmi, “Parameterized MAC Unit Generation for a Scalable Embedded DSP Core”, *IEEE Conference: NORCHIP 2008*, pp. 127-132, 2008.

76. Aktan, M., A. Yurdakul and G. Dündar, “An Algorithm for the Design of Low-Power Hardware Efficient FIR Filters”, *IEEE Trans. on Circuits Syst. I: Reg. Papers*, Vol. 55, No. 6, pp. 1536-1545, 2008.

77. Parhi, K., *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, 1999.

78. Mitra, S., *Digital Signal Processing: A Computer Based Approach*, Second Edition, McGraw-Hill, 2001.

# REFERENCES NOT CITED

Cini, U., A. Morgül and M. Aktan, “Current-Mode Multiple-Valued Adder Tree Design for High Performance Arithmetic Applications”, *ELECO’2007, International Conference on Electrical and Electronics Engineering*, Bursa, Dec. 2007.

Cini, U., A. Morgül and M. Aktan, “Design of a Current-Mode Multi-Operand Adder Using Signed Digit Arithmetic”, *IEEJ Intl. Workshop on Analog VLSI*, Istanbul, July 2008.

Cini, U. and A. Morgül, “A Multi-Operand Adder Circuit Design Using Novel Multi-valued Current Mode Design”, *ELECO’2009, International Conference on Electrical and Electronics Engineering*, Bursa, Dec. 2009.

Cini, U., M. Aktan M. and A. Morgül, “Constant Coefficient FIR Filtering Optimized for 6-input LUT Based FPGAs”, *Journal of Signal Processing Systems*, Submitted.

Cini U. and A. Morgül, “A Current-Mode Multi-Valued Adder Circuit for Multi-Operand Addition”, *International Journal of Electronics*, Submitted.