# High Performance Binary, Logarithmic, and BCD Multiplier Architectures

**THESIS**

Submitted in partial fulfillment

of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

by

**Syed Ershad Ahmed**

**ID No. 2009PHXF448H**

Under the Supervision of

**Prof. M. B. Srinivas**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI**

**December, 2017**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

## CERTIFICATE

This is to certify that the thesis entitled High Performance Binary, Logarithmic, and BCD Multiplier Architectures and submitted by Syed Ershad Ahmed ID No 2009PHXF448H for award of Ph.D. of the Institute embodies original work done by him under my supervision.

Signature of the Supervisor

Dr. M. B. SRINIVAS

Professor

Date:

# Acknowledgements

Accomplishment of this doctoral thesis was not possible without the support of several people directly and indirectly. It gives me great pleasure to express my deep sense of gratitude to Prof. M.B.Srinivas, Professor, Department of Electrical Engineering, BITS-Pilani, Hyderabad Campus for his invaluable guidance, cooperation and keen interest during my research work. I am indeed fortunate to work under his supervision.

I am thankful to Prof. Souvik Bhattacharyya, Vice-Chancellor and Prof. G.Sundar, Director, BITS-Pilani, Hyderabad Campus for providing a research environment to enhance my research interest and commitment.

I wish to enlist my deep sense of gratitude to Prof. Sanket Goel, Head, Electrical Engineering Department, BITS-Pilani, Hyderabad Campus for providing me full support to carry out this research work. I am extremely thankful to all the faculty, PhD scholars and staff of Electrical Engineering Department.

I take this opportunity to thank my doctoral advising committee members, Prof. Prabhakar Rao, and Prof. Subhendu Kumar Sahoo for providing suggestions during the entire thesis and sparing their valuable time in providing useful comments for my draft thesis.

It's my pleasure to thank my parents and wife who stood with me during tough times with all their valuable and encouraging words.

# Abstract

Digital multipliers form an important part of digital arithmetic circuits. Important parameters that characterize these multipliers are their precision and those related to their implementations such as area, critical path delay, and power consumption.While certain applications demand high precision, others would require optimality in terms of die-area, latency of operation and power consumed. Thus the present thesis focuses on developing novel multiplier architectures (binary, logarithmic and BCD) that lead to either improved precision or result in better implementation.

The first contribution of this thesis is the development of a reconfigurable, two-dimensional (2D) bypass multiplier architecture that is based on dynamic bypassing of partial products. The bypass elements incorporated into the multiplier reduce the power consumption by eliminating redundant signal transitions.The reconfigurable architecture offers a good trade-off between area, delay and power dissipation since it uses the same multiplier for performing one $N$ or two $N/2$ multiplications.

As the modern computing systems become increasingly embedded and portable, a growing set of applications in media processing (graphics, audio, video, and image) has evolved. Many of these applications, however, possess an inherent quality of error resilience. For example, it is a known feature of image processing that a range of image resolutions/sharpness is acceptable depending on the nature of the application. Thus arithmetic units (digital multipliers in present case), that are not very precise but return an approximate value, can be utilized in such applications. Such units, it may be anticipated, may result in area savings while also resulting in reduced power consumption. The second contribution of this thesis is the development of a novel approximate binary multiplier architecture that results in improved performance in terms of area, delay and power compared to existing architectures while the trade-off in accuracy is only marginal.

Further, in recent years, logarithmic number system has been increasingly used as an alternative to the binary number system as it converts multiplication to addition resulting in simplified hardware. While logarithmic number system cannot be compared with that of binary in terms of precision, usage of it in arithmetic operations such as multiplication certainly results in reduced area and power consumption and thus is useful in applications where precise results are not required. The third contribution of this thesis is the development of an efficient logarithmic multiplier architecture that significantly reduces the area and power consumption of the hardware while sacrificing the accuracy only marginally.

Extensive analysis of the hardware requirement of both the multipliers (approximate binary and logarithmic) has been carried out, initially using unit gate modeling, and later on using the synthesis tool. Furthermore, to quantify the advantage of the proposed architectures, both have been used in an image sharpening algorithm (that employs extensive multiplication) and benchmarked against certain standard and well known image processing applications such as Lena, Cameraman and Pirate.

Finally, while binary arithmetic is all pervasive, BCD (decimal) arithmetic is preferred in applications such as financial, scientific and commercial etc. owing to its comparatively high precision. The fourth contribution of this thesis is the development of a generalized design approach and architectural framework for decimal multiplication. In this approach, unlike the existing decimal architectures, the decimal partial product generation is achieved in parallel using fast and area efficient blocks, while the partial product reduction is achieved using hybrid multi-operand binary to decimal converters. A comprehensive analysis of the synthesis results carried out on IEEE-compliant 16-digit decimal multiplier indicates the superiority of the proposed architecture over the existing ones.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| BCD | Binary Coded Decimal |
| BD | Binary to Decimal |
| BDM | Binary Coded Decimal Digit Multiplier |
| BIM | Babic Iterative Multiplier |
| BLB | Basic Logarithmic Block |
| BM | Binary Multiplier |
| CLA | Carry Look-ahead Adder |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPA | Carry Propagate Adder |
| CSA | Carry Save Adder |
| DSP | Digital Signal Processor |
| FA | Full Adder |
| FBD | Fast Binary to Decimal |
| FP | Fractional Predictor |
| HA | Half Adder |
| HDL | Hardware Description Language |
| HPBDM | High Performance Binary Coded Decimal Digit Multiplier |
| HPPPBD | High Performance Partial Product Binary to Decimal |
| ITB | Internal Tristate Buffer |
| IUS | Incisive Unified Simulator |
| KOB | Karatsuba-Ofman |

| | |
|---|---|
| LABD | Low Area Binary to Decimal |
| LABDM | Low Area Binary Coded Decimal Digit Multiplier |
| LAPPBD | Low Area Partial Product Binary to Decimal |
| LNS | Logarithmic Number System |
| LS | Logarithmic Shifter |
| LSB | Least Significant Bit |
| LSP | Least Significant Portion |
| LUT | Look-up Table |
| MA | Mitchell Approximation |
| MBD | Multi-operand Binary to Decimal |
| MFA | Modified Full Adder |
| MFP | Modified Fraction Predictor |
| MOA | Multi-operand Adder |
| MRBA | Modified Row-Bypass Adder |
| MSB | Most Significant Bit |
| MSE | Mean Square Error |
| MSP | Most Significant Portion |
| ND | Nicoud Cell |
| PE | Priority Encoder |
| PP | Partial Product |
| PPBD | Partial Product Binary to Decimal |
| PPG | Partial Product Generation |
| PPR | Partial Product Reduction |
| PSNR | Peak Signal to Noise Ratio |
| RAM | Random Access Memory |
| RCA | Ripple Carry Adder |
| RTDBC | Reconfigurable Two Dimensional Bypass Cell |
| RTL | Register Transfer Level |
| TBLB | Truncated Basic Logarithmic Block |

| | |
|---|---|
| TDBA | Two Dimensional Bypass Adder |
| TEC | Truncated Error Correction |
| VLSI | Very Large Scale Integration |

# Chapter 1

# Introduction

## 1.1 Background

One of the most common and frequently executed operations in arithmetic computations is multiplication. Significant amount of work has been carried out to improve the performance of digital multipliers over the years and the same is expected to continue in future.The criteria that are used to quantify their performance include latency, area and power consumed. Thus any improvement made in the design/architecture of multipliers should be reflected in the improvement of these parameters.

In digital static CMOS multipliers, transition activity (due to charging and discharging of the load capacitance) dominates the total energy consumption. Thus, power saving can be achieved by lowering the switching or transition activity per operation. Earlier efforts attempted to reduce the switching activity of the binary multipliers through architectural modifications such as row and/or column bypassing. In these schemes, the redundant signal switching is eliminated by disabling the full adder circuits whose partial product is zero while forwarding the output of the previous adder rows/columns to the next row/columns. However, the extra bypass logic (mostly adder) has only limited effect in reducing the power dissipation while contributing significantly to area overhead. Thus, there is a need to develop alternate bypass multiplication architectures that can address large power consumption in multipliers.

Many of the signal and image processing applications possess an inherent quality of error resilience and thus do not require absolute accuracy in computation. Further, the final output in these applications is interpreted by human senses which are not perfect. Thus approximation in place of accuracy can be exploited that can lead to a significant improvement in area, power and performance. Based on this idea, several techniques have been proposed that focus on approximate rather than accurate computing. However, most of these techniques provide solutions that are based on trial and error and thus the accuracy achieved tends to be lower. Thus, realizing efficient multiplier units (binary and logarithmic) for approximate computing in a systematic way, which also have high precision would be of considerable interest.

The importance of error-free arithmetic is growing day-by-day and decimal (BCD) arithmetic circuits are making their way into application such as financial, scientific and commercial, etc. Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic, is multiplication. Decimal multiplication can be classified as serial multiplication, parallel ('word-by-digit') and ('digit-by-digit') multiplication. Decimal (BCD) 'digit-by-digit' multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in significant improvement in the multiplier performance. There is however a significant scope to develop more efficient architectures for 'digit-by-digit' multiplication.

## 1.2 Objectives of the Thesis

The objectives of this thesis are as follows:

- To improve the existing binary multiplier architecture to reduce the switching activity resulting in low power consumption.
- To design and implement truncated binary and iterative logarithmic multipliers targeted for error resilient applications
- To develop a BCD multiplier with improved performance for high speed (parallel) multiplication

## 1.3 Steps involved in carrying out present research

Figure 1.1 illustrates the steps involved in carrying out present work, summarized below:

- Modification of multiplier architectures to improve precision and/or performance for binary, logarithmic and decimal multiplication
- Modeling the architectures (fixed-width binary and logarithmic multiplier) in MATLAB
- Evaluation of the above using synthesis-independent unit gate level (hardware) modeling
- Verification using Verilog test benches and applying random stimuli to cover a wide input range
- Synthesis of binary, logarithmic and decimal multipliers using Cadence RTL compiler to obtain estimates of area, delay, and power
- Analysis of the above multipliers to evaluate and compare their performance with the existing designs

### 1.3.1 High Level Modeling

The multiplier schemes (fixed-width binary and logarithmic multiplier) have been modeled and verified in MATLAB environment. Metrics related to precision such as maximum error and average error have been computed for different multiplier schemes.The main purpose of carrying out high-level modeling is as follows:

1. It is a faster way of realizing optimized architectures/designs
2. It offers an easier and faster method to evaluate and compare different architectures/designs
3. A high-level model serves as an abstract model of the design to generate input stimulus and verify the result

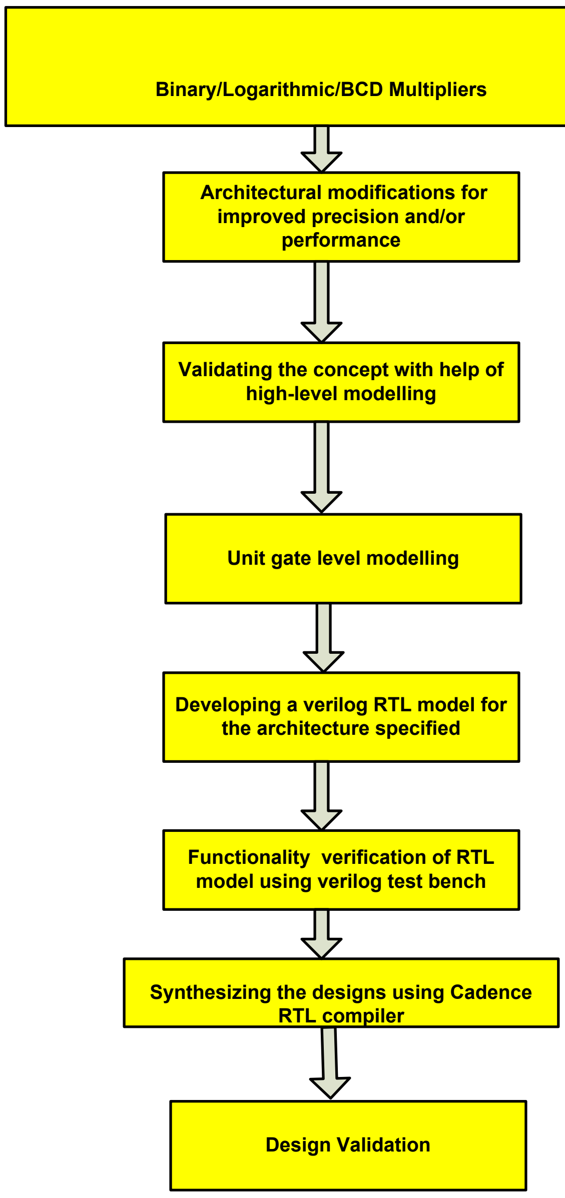

Binary/Logarithmic/BCD Multipliers
Architectural modifications for improved precision and/or performance
Validating the concept with help of high-level modelling
Unit gate level modelling
Developing a verilog RTL model for the architecture specified
Functionality verification of RTL model using verilog test bench
Synthesizing the designs using Cadence RTL compiler
Design Validation

Figure 1.1: Research Flow chart

### 1.3.2 Unit Gate Level Modeling

All the designs (fixed-width binary, logarithmic and decimal (BCD) multipliers) under consideration have been modeled using unit gate approach to obtain a rough estimate of area $(A)$ and delay $(D)$. This model is useful for high-level analysis and does not depend strongly on any one process technology, synthesis tool, or cell library. The assumptions made while performing the unit gate modeling are the following: Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for both area and delay. Further, an m-input gate is assumed to be composed of a tree of m-1 input gates while the effects of wiring, buffering and inverting costs (area and delay) are neglected.

### 1.3.3 CAD tools used in the ASIC implementation

#### 1.3.3.1 Cadence Simulator

Cadence NCSim is an RTL functional simulator that can simulate Verilog models. The functional behavior of the modules (Binary, Logarithmic and BCD multipliers) was verified in NCSim using Verilog test benches.

#### 1.3.3.2 Cadence RTL Compiler

Cadence RTL Compiler is a hardware synthesis tool. It maps an RTL hardware description model using a standard cell library into a gate-level net list. The output structural level thus obtained is composed of cells that exist in the standard-cell technology library. The synthesis tool accepts Verilog RTL code as an input and generates area, delay and power reports.

## 1.4 Organization of the Thesis

This thesis is organized as follows. Chapter 2 presents a review of various existing multiplier architectures relevant to this research and their realization in hardware. It also provides a detailed discussion of multipliers based on binary, logarithmic and BCD number systems. Keeping in mind the importance of arithmetic precision, chapters 3 and 6 develop and validate new techniques for improved precision of binary and decimal arithmetic circuits. Since precision is not as important as efficiency of implementation for error resilient applications, chapters 4 and 5 develop novel truncation schemes that lead to efficient implementation of binary and logarithmic multipliers. These schemes have also been compared for performance against the existing ones. Chapter 7 draws conclusions and provides recommendations for future work.

# Chapter 2

# Literature Review

## 2.1 Introduction

This chapter reviews a number of widely used multiplier architectures while focusing mainly on those that will be of concern in this thesis. The chapter is organized as follows: Preliminary information on the existing binary multiplier architectures is presented in Section 2.3 - 2.5, and fixed-width (truncated) multipliers are discussed in Section 2.6. An outline of the existing logarithmic multipliers is presented in Section 2.7 while decimal (BCD) multipliers are reviewed in Section 2.8.

## 2.2 Classification of Multiplier Architectures

Digital multipliers based on number system can be classified as (i) Binary multipliers (ii) Logarithmic multipliers and (iii) BCD multipliers. A pictorial representation of the same is given in Fig.2.1 and explained in detail in the following sections.

## 2.3 Binary Multipliers

It is well known that binary multipliers can be classified into two categories, viz., integer fixed-point and floating point. This thesis however focuses on integer fixed-point multiplier architectures only. In fact, floating-point multipliers consist of a fixed-point multiplier for the

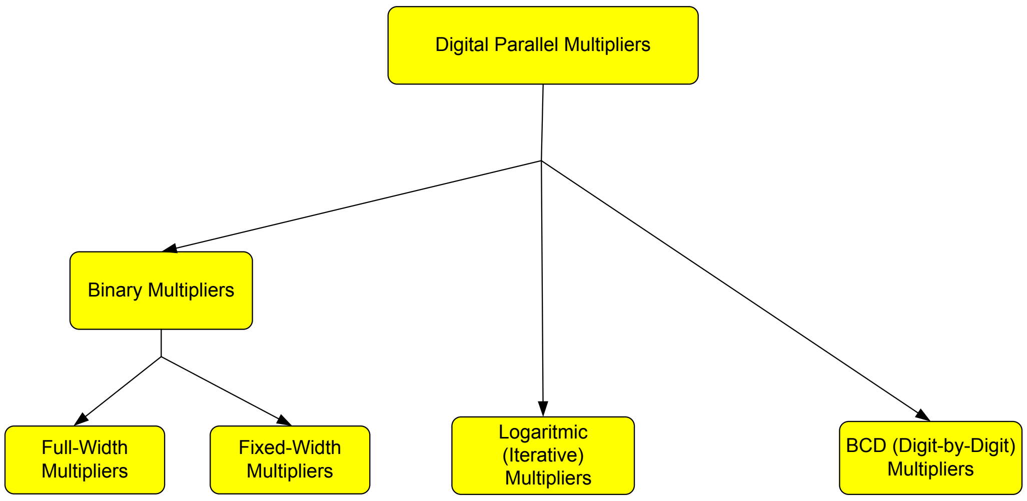

Digital Parallel Multipliers
Binary Multipliers
Full-Width Multipliers
Fixed-Width Multipliers
Logaritmic (Iterative) Multipliers
BCD (Digit-by-Digit) Multipliers

Figure 2.1: Classification of digital multiplier architectures based on number system

significant and additional circuitry to deal with the exponents and special values. Thus, techniques developed for efficient binary multiplication presented in this thesis are also applicable for floating-point multiplication.

Literature on binary computer arithmetic includes topics ranging from sequential to parallel multipliers. Today, most of the advanced digital systems include a parallel binary multiplication unit to carry-out mathematical computations. Array and Booth multipliers are a few examples of parallel multiplication in this category. As is well known, array-based multipliers [1] are ideal for very large scale integration (VLSI) and application specific integrated circuits (ASICs) due to their regular layout. On the other hand, Booth multiplier [2], although faster compared to array multipliers, has an irregular layout structure, making it not very suitable for VLSI implementations.Thus, this thesis focuses on design and validation of area and power efficient binary array multipliers. In order to provide more insight in to multiplication process, the general structure of binary multiplier is described initially and implementation of the same is illustrated using Braun array multiplier.

In general, binary multiplication involves three steps: (i) Partial product generation (PPG) (ii) Partial product reduction (PPR) and (iii) Final product computation. A typical binary multiplier accepts two binary inputs $A$ and $B$, each of $N$-bit width, as illustrated in Fig.2.2. Multiplication schemes primarily differ in the manner partial products are generated and/or accumulated. The multiplication operation can be accelerated in two ways: generating optimized number of partial products (PPs) in the first (PPG) step or accelerating their accumulation in second (PPR) step. One of the most effective ways of accumulating an array of partial products into two rows is through carry save adder structure. These two rows are eventually reduced using a final adder in the last step. A detailed explanation of PPG and PPR in an array multiplier is provided in subsequent sections.

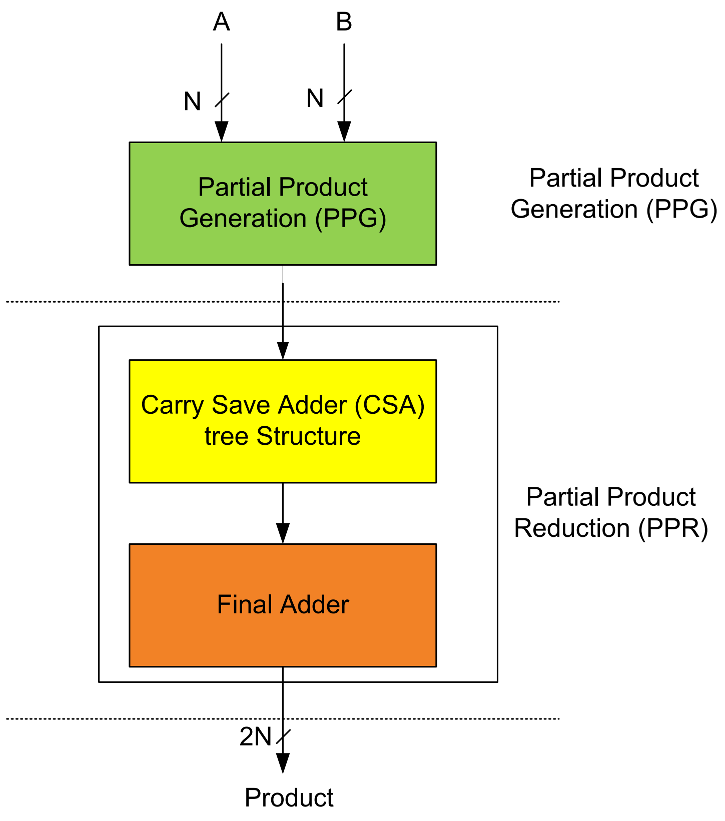

A
B
N
N
Partial Product Generation (PPG)
Partial Product Generation (PPG)
Carry Save Adder (CSA) tree Structure
Partial Product Reduction (PPR)
Final Adder
2N
Product

Figure 2.2: A general binary multiplication structure

## 2.4 Braun Multiplier

The simplest array multiplier proposed was by Braun [3], generally known as carry save multiplier, suited for unsigned operations only. The mathematical model of a N*N unsigned array multiplication is given below. Assume *A* and *B* to be two *N*-bit unsigned numbers, where *A* is the multiplicand and *B* is the multiplier.

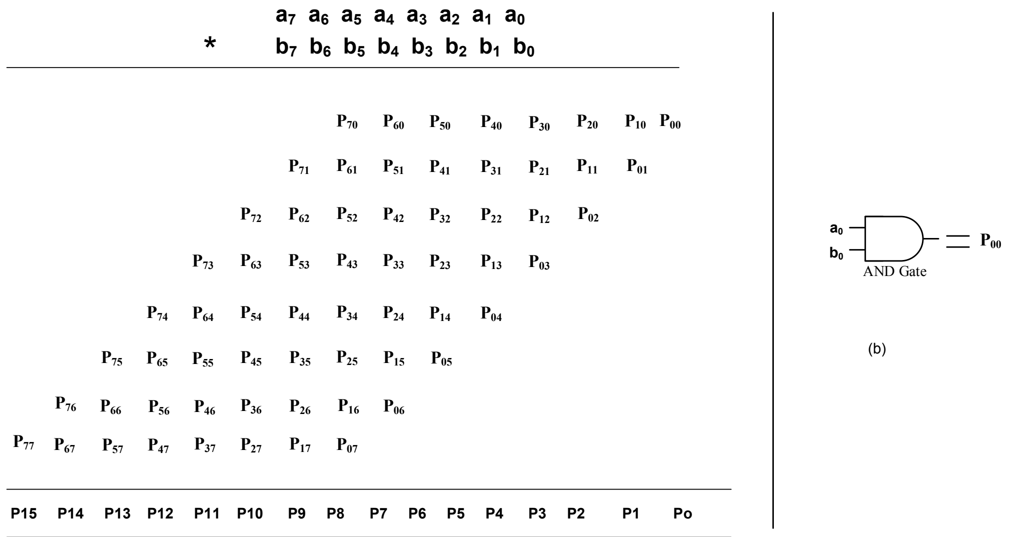
$$A = \sum_{i=0}^{n-1} a_i.2^i \tag{2.1}$$

$$B = \sum_{i=0}^{m-1} b_j.2^j \tag{2.2}$$

The product $(P)$ can be written as: $P = A * B = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} a_i b_j.2^{i+j}$

### 2.4.1 Partial Product Generation (PPG)

In general, to implement $N * N$ binary multiplier in hardware, $N^2$ partial products are required for PPG which are generated using AND gates. As an example, consider hardware implementation of 8*8 Braun multiplier illustrated in Fig.2.3. A typical multiplication of two 8-bit binary numbers results in a total of 64 PPs. Figure.2.3(a) depicts the arrangement of these partial products in a matrix form. Each of these partial products (PPs) is obtained using an AND gate as illustrated in Fig.2.3(b). Further, an alternate partial product representation of the same multiplier is shown in Figure.2.4.

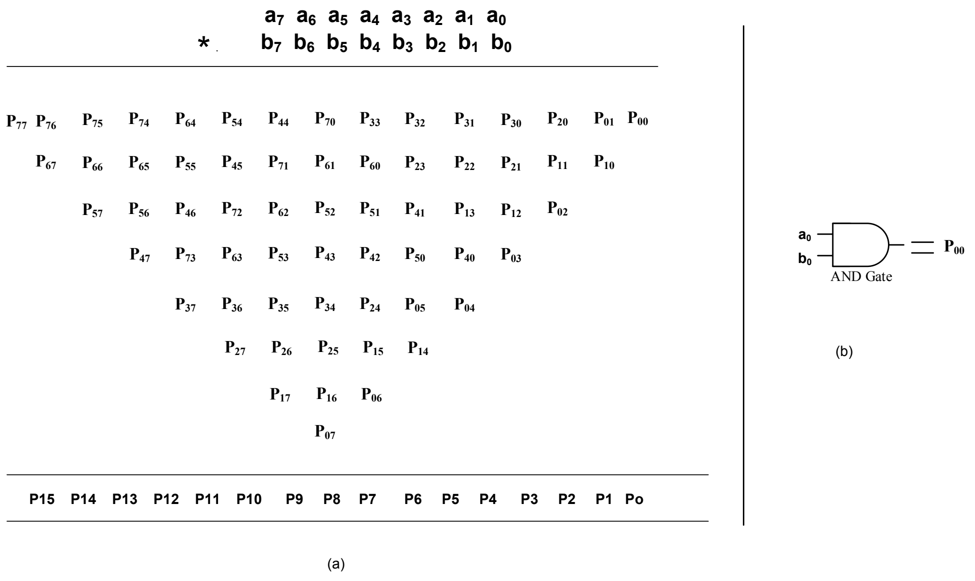


a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P70 P60 P50 P40 P30 P20 P10 P00
P71 P61 P51 P41 P31 P21 P11 P01
P72 P62 P52 P42 P32 P22 P12 P02
P73 P63 P53 P43 P33 P23 P13 P03
P74 P64 P54 P44 P34 P24 P14 P04
P75 P65 P55 P45 P35 P25 P15 P05
P76 P66 P56 P46 P36 P26 P16 P06
P77 P67 P57 P47 P37 P27 P17 P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po
a0
b0
AND Gate
P00
(b)

Figure 2.3: (a) Partial product matrix representation in a 8*8 Braun multiplier (b) Partial product computed using an AND gate

### 2.4.2 Partial Product Reduction (PPR)

The PPs generated must be accumulated to form the final product. In multiplication, accumulation of PPs, also referred to as reduction of PPs, consumes most of the time taken for multiplication. The reduction of the PPs is performed using two main methods, namely, ac-



a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P77 P76 P75 P74 P64 P54 P44 P70 P33 P32 P31 P30 P20 P01 P00
P67 P66 P65 P55 P45 P71 P61 P60 P23 P22 P21 P11 P10
P57 P56 P46 P72 P62 P52 P51 P41 P13 P12 P02
P47 P73 P63 P53 P43 P42 P50 P40 P03
P37 P36 P35 P34 P24 P05 P04
P27 P26 P25 P15 P14
P17 P16 P06
P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po
(a)
a0
b0
AND Gate
P00
(b)

Figure 2.4: (a) Alternate representation of partial products in 8*8 Braun array multiplier (b) Partial product computed using an AND gate

cumulation by rows and accumulation by columns. The building modules are referred to as adders if the accumulation is by rows and conversely, if the reduction is by columns, they are referred to as counters. A simple technique in the accumulation by rows involves multiple two-operand carry propagate adders (CPAs). However, propagation of the carry using CPAs is time-consuming and thus is slow [4]. An alternate and more efficient approach is to reduce the columns by using carry-free adders, namely, carry save adders (CSAs) as discussed below.

#### 2.4.2.1 Binary Carry Save (CSA) Adders

Carry save adders are popular structures used for partial product reduction in multiplication process. The binary partial product reduction structure uses multiple levels of carry save adders (CSAs). As illustrated in Fig.2.5, each bit-slice (group of 3-bits) of a CSA is realized using a full adder. This binary full adder generates a sum-bit and a carry-bit. The carry input is propagated from the previous bit-slice to the next most significant position in the reduction tree. The PP reduction process results in two rows (sum and carry), which are eventually converted to the final sum using a two operand adder (carry propagating adder). In short, binary PPR can be implemented via CSA tree comprising of binary half adders (HAs) and full adders (FAs) as basic elements.

As is well known, a half adder accepts two operand bits ($A$ and $B$) as inputs and computes

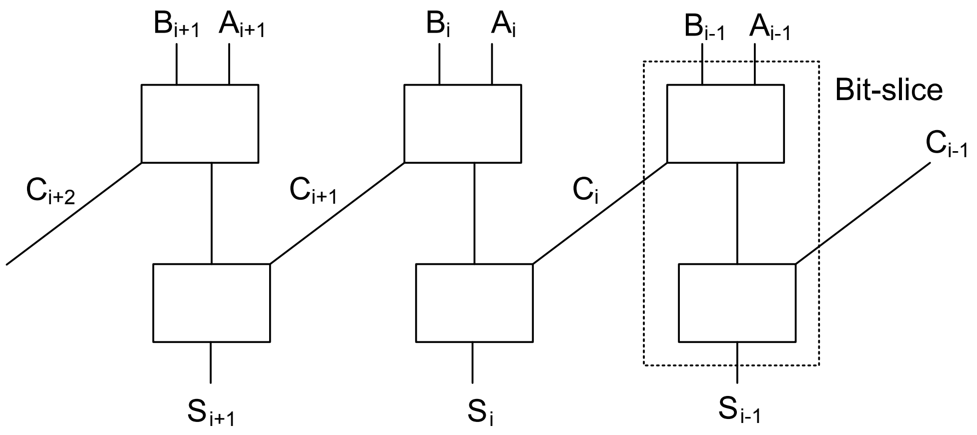


Figure 2.5: Carry- free operation using full adders

sum $(S)$ and output carry bit $(C_o)$ as outputs. The carry bit $(C_o)$ will eventually serve as input carry-in for the successive half adder. The implementation of a half adder circuit follows the Boolean equations 2.3 and 2.4 and its gate level implementation, cell notation and dot notation are shown respectively in Fig.2.6 (a-c). The notations '$\oplus$', '.' and '+' denote logical XOR, AND and OR gates respectively.

$$S = A \oplus B \tag{2.3}$$

$$C_0 = A.B \tag{2.4}$$

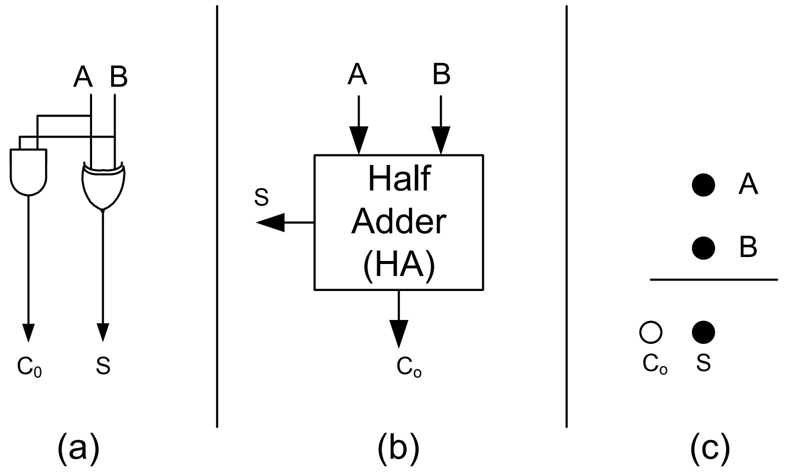


Figure 2.6: (a) Logic circuit of half adder (b) Half adder cell notation (c) Computation of Sum and Carry-out using dot notation in a half adder

Similarly, a full adder accepts three operands $(A, B$ and carry in $(C_i))$ as inputs and computes the sum $(S)$ and output carry bit $(C_o)$ as outputs. The design of a full adder circuit follows the Boolean equations 2.5 and 2.6 while the gate level implementation, cell notation and dot notation are illustrated in Fig.2.7(a-c) .

$$S = A \oplus B \oplus C_i \tag{2.5}$$

$$C_0 = (A \oplus B).C_i + AB = AB + BC_i + C_iA \tag{2.6}$$

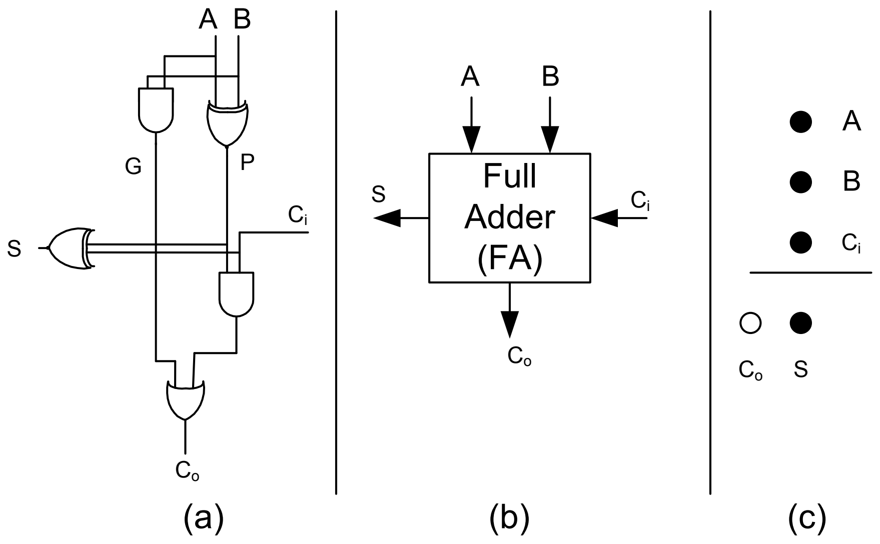


Figure 2.7: (a) Logic circuit of full adder (b) Full adder cell notation (c) Computation of Sum and Carry-out using dot notation in a full adder

A row of full-adders, represented in Fig.2.5, can be viewed as a mechanism to reduce three operands to two operands. For a CSA, each FA referred to as 3:2 counter has three dots in one column as inputs. The resulting sum output will result in a dot with the same magnitude as the inputs while carry output will result in a dot in the column to its left (one order of magnitude higher) as shown in Fig.2.7(c).

For illustration purpose, reduction of two partial product columns (c0 and c1) is shown in Fig.2.8(a) and the same is later extended to 8*8 multiplication as shown in Fig.2.9. Each column, consisting of six partial products (each denoted by solid dot(•) ), is reduced in parallel to sum (S) and carry (C). The sum is denoted by solid dot(•) with the same magnitude as the inputs while the carry, denoted with hollow dot (∘), has higher positional weight compared to input. The six partial products are reduced ( in three levels) to two rows using a tree of 3:2 and 2:2 full and half adders. These two rows of PP are eventually reduced to a binary number by using a carry propagation adder (CPA) represented with a horizontal line shown in Fig.2.8 (a) .

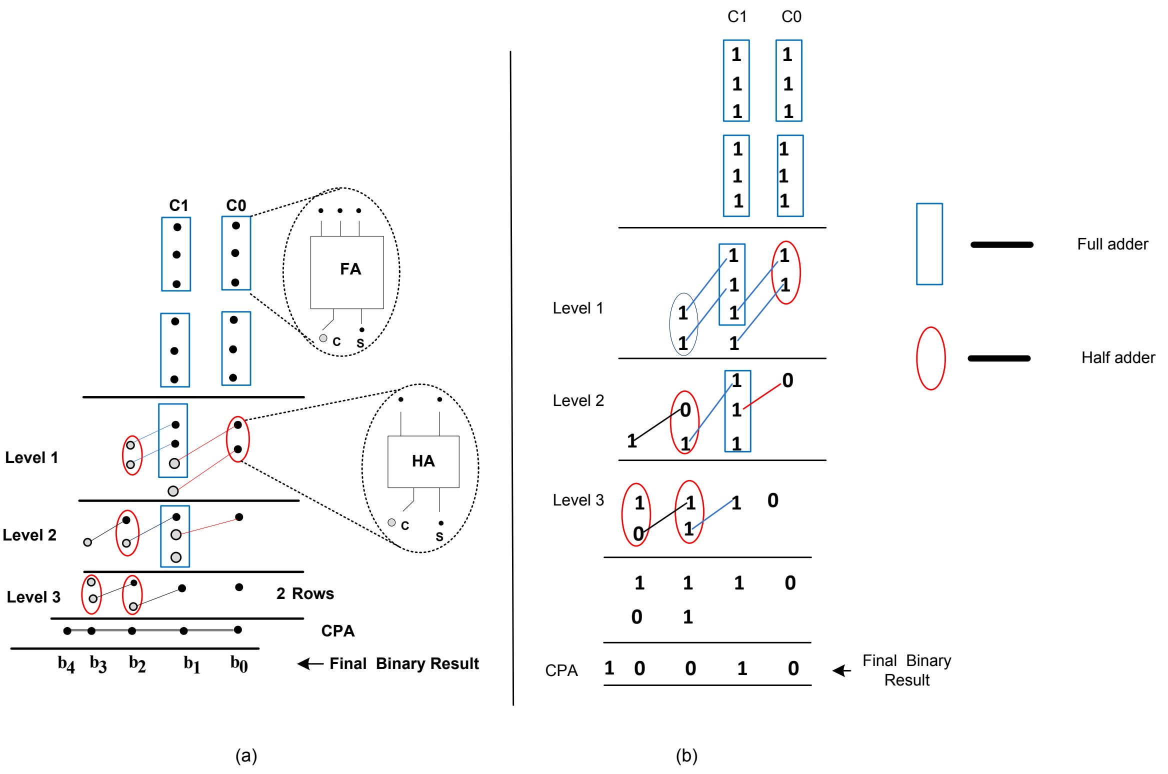


Figure 2.8: (a) Partial product reduction using CSA dot notation (b) A numerical example related to partial product reduction

The numerical example illustrated in Fig.2.8(b) describes the addition of two columns ($c0$ and $c1$ ), each consisting of six bits, where each partial product in the column is assumed to be '1'. These columns are reduced to two rows in four levels using a tree of full and half adders. These two rows of PPs are reduced to final binary result by using a carry propagation adder (CPA).

A variety of algorithms for accumulating the partial products using CSAs has been proposed [1]. The advantage of using CSAs is that they do not contribute to hardware complexity and one of the first algorithms proposed was by Wallace [5].

### 2.4.3 Wallace Reduction Tree

Wallace developed a method for reducing the columns in parallel. Figure.2.9 illustrates a Wallace-like reduction tree organization for an $8 * 8$-bit unsigned multiplier, presented in Section 2.4.1. As discussed earlier, a sum output ($S$) from a full or half-adder at one stage places a dot in the same column at the next stage. A carry output ($C_o$) from a full or half-adder at one stage places a dot one order of magnitude higher i.e., in the column to its left, in the next stage. As shown in Fig.2.9(d), three dots joined by a solid diagonal line indicates that these PPs are outputs of a (3,2) counter, while two dots joined by diagonal line indicates that these PPs are outputs of a (2,2) counter. Consequently, the PP matrix is accumulated to a height of two in four levels using a carry save adder (CSA) tree structure formed using full adder and half adder as shown in Fig.2.9(a). A total of four reduction levels with matrix heights of 6, 4, 3 and 2 is required to accumulate the PP matrix into two rows using Wallace technique. These two rows are reduced to a final sum using a carry propagate adder (CPA) or any fast adder mentioned below.

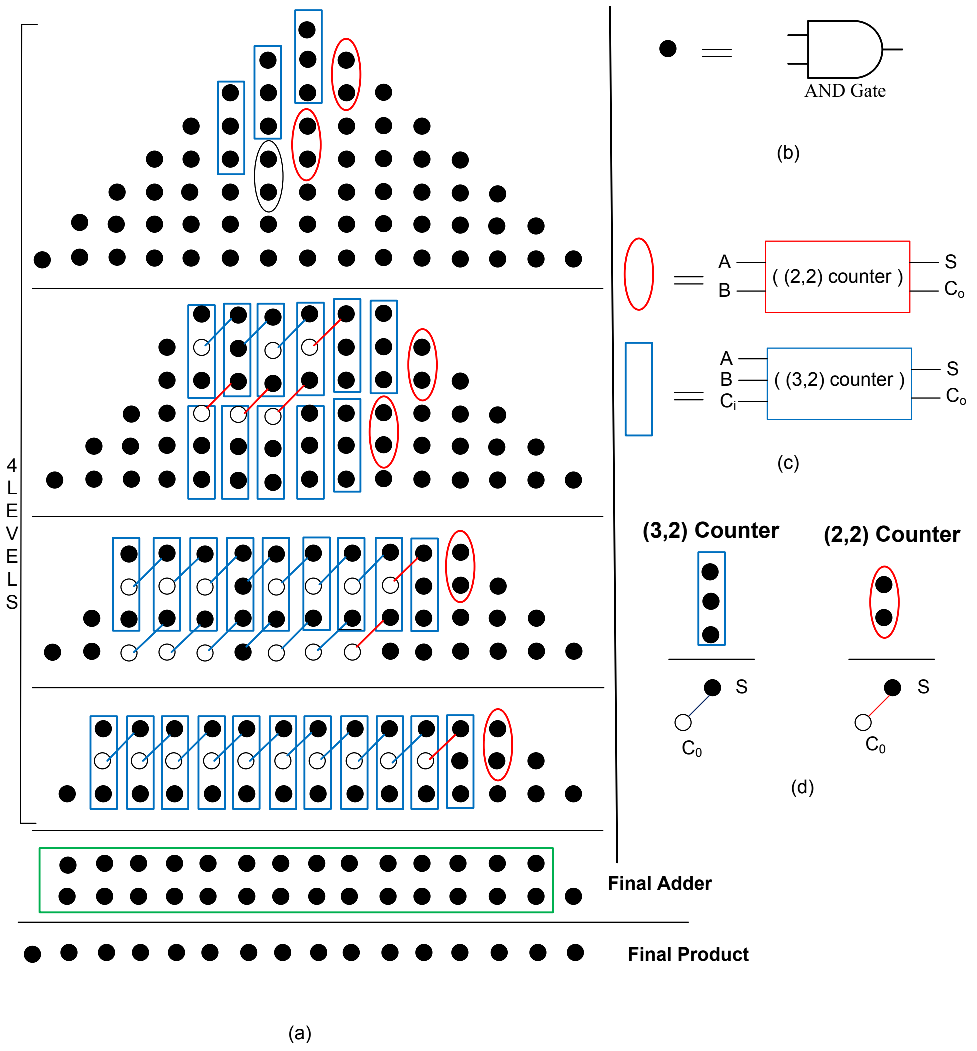


Figure 2.9: (a) Wallace tree partial product reduction structure using 3:2 and 2:2 counters (b) Partial product computed using an AND gate (c) Representation of 3:2 and 2:2 counters (d) Computation of Sum and Carry-out using dot notation in a full and half adder circuits

### 2.4.4 Final Adder

The last step in the partial product reduction process is the conversion of the redundant sum obtained from Wallace reduction tree into non-redundant representation. This step is performed using a non-redundant adder. There exist many topologies to implement the final adder namely, ripple carry, carry look-ahead, and parallel prefix (or prefix tree) [6] among many. Based on priorities (area and delay) appropriate adder topology can be selected from the available literature. For instance, a ripple carry adder has area and a delay that is proportional to the adder‘s length while prefix based adders have almost logarithmic delay but with area overhead. Thus, appropriate adder design can be chosen depending on the requirement.

#### 2.4.4.1 Ripple Carry Adder

The basic building blocks of a ripple-carry adder (RCA) are full adders. Consider two n-bit numbers, $A$ and $B$, described by equations 2.1 and 2.2. A total of n full adders are used, one for each column. The full adder in column $i$ adds the operand bits $A_i$ and $B_i$ plus the carry-in $(C_i)$, where $i = 0, 1 \ldots N-1$. The carry-out of previous stage full adder is passed down to the carry-in of the full adder in the next most significant column. The $S_i$ outputs of the $n$ full adders form the sum. Figure 2.10 illustrates a 4-bit ripple carry adder.

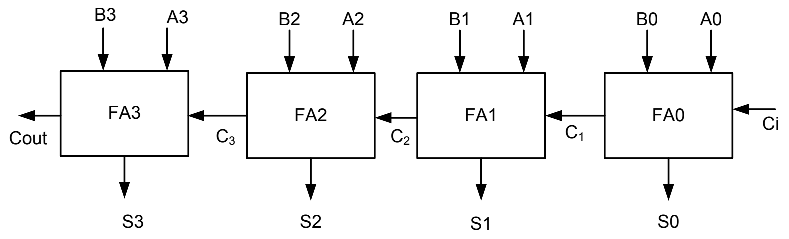


Figure 2.10: 4-bit Ripple Carry Adder

Although, ripple carry adder is simple and easy to implement, it suffers from large delay. This is because the full adder in the next stage has to wait for carry bit from the previous stage full adder (FA). By inspecting the FA shown in Fig.2.10 it can be observed that each full adder contributes to a two gate delay in the process of rippling the carry [6]. In general, critical path length of the final carry propagation adder can be deduced as follows:

$$CPA\,length = 2N-2$$

#### 2.4.4.2 Carry Look-ahead Adder (CLA)

The carry propagation delay in a ripple carry adder increases linearly with an increase in the number of input bits. Efforts to reduce this delay has resulted in novel adder architectures and Carry Look-ahead (CLA) is one such adder which improves the speed by computing the carry signals in advance that depends on the input operands.

Based on the combination of inputs $A_i$ and $B_i$, the signals, generate $(G_i)$ and propagate$(P_i)$, determine the possibility of carry generation. Generate term determines if a carry-out would be '1' independent of carry-in while propagate term determines whether carry moves to the next higher significant position. The standard carry look-ahead adder equations ($G_i$ and $P_i$) that dictate if the carry will be generated or propagated can be given as,

$$G_i = A_i.B_i \tag{2.7}$$

$$P_i = A_i \oplus B_i \tag{2.8}$$

Clearly, carry generation depends on the values of $A_i$ and $B_i$. For instance, when $A_i = B_i$ ='1', a carry of '1' is produced at the $i^{th}$ position, while a carry of '0' is generated when $A_i = B_i$ ='0'. Conversely, carry propagation happens when $A_i \neq B_i$. Hence, when $A_i \neq B_i$ and carry-in $(C_{in})$ is '1', then $C_{in}$ is said to propagate to the next position.

Accordingly, the sum and carry recurrence for the $i^{th}$ stage is as follows:

$$S_i = P_i \oplus C_i \tag{2.9}$$

$$C_{i+1} = G_i + P_i.C_i \tag{2.10}$$

Similarly, the carries in a 4-bit CLA are generated in parallel according to the following equations:

$$C_1 = g_0 + p_0 c_0 \tag{2.11}$$

$$C_2 = g_1 + p_1 g_0 + p_1 p_0 c_0 \tag{2.12}$$

$$C_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \tag{2.13}$$

$$C_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \tag{2.14}$$

The logic circuit of a 4-bit CLA is illustrated in Fig.2.11.

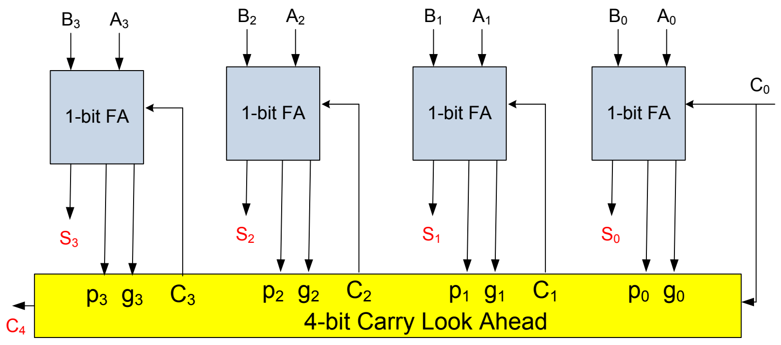


Figure 2.11: 4-bit Carry Look-ahead adder

One obvious disadvantage in CLA adder is that the carry block gets complicated for large values of N. To mitigate this, a new class of adder networks has been designed that transfers the carry through the look-ahead stage in about $log_2(N)$ stages. These networks are known as tree networks and the adder circuits that utilize these networks are called prefix-adders or tree-adders [7].

### 2.4.4.3 Carry Look-ahead (CLA) based Parallel Prefix Adder

There are numerous ways to design the parallel prefix tree adders that offer trade-offs among parameters like the number of logic stages, the maximum fan-out of each logic gate and the wiring complexity between the stages [6] etc. Based on these parameters a wide variety of prefix tree architectures, namely, Sklansky, Brent-Kung, Kogge-Stone, Ladner-Fischer, Han-Carlson and Knowles [6] have been developed.

In general, as illustrated in Fig.2.12, there are three stages in any prefix adder that can be termed as (i) pre-computation stage (ii) prefix network stage and (iii) post-computation stage [6,8,9]. The pre-computation stage determines the generate and propagate bits as per the equations 2.7 and 2.8.

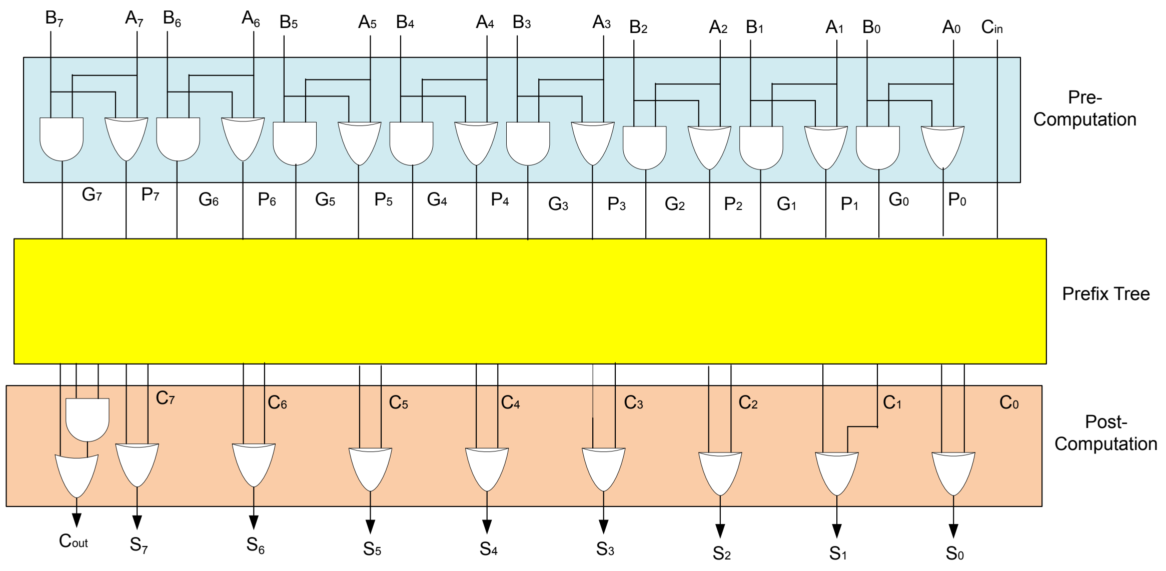


Figure 2.12: CLA based 8-bit parallel-prefix structure

The prefix network stage computes the final carries from the individual generate and propagate bits of pre-computation stage. Using associative principle, carry computation is transformed to prefix problem using the operator '∘' which associates pairs of generate and propagate as mentioned below:

$$(g, p) \circ (g', p') = (g + p.g', p.p') \tag{2.15}$$

where $g$ and $g'$ denote the generate terms and $p$ and $p'$ represent the propagate terms. Using the operator '∘' consecutive generate and propagate pairs can be grouped to generate carry as follows:

$$C_i = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ ....(g_1, p_1)(g_0, p_0) \tag{2.16}$$

The post computation stage determines the final sum from carries generated in the prefix network stage.

The graph model of prefix carry computation is obtained by representing the operator '∘' as node •, while the signal pairs $(g, p)$ are denoted as edges of a graph. Different prefix structures differ only in prefix network stage.

To illustrate a prefix structure, an 8-bit Kogge-Stone [8] prefix tree is illustrated in Fig.2.13. The dark color (•) node in the graph represents the logic module while the white color (∘) node denotes a feed through node with no logic (realized with a buffer in real hardware).

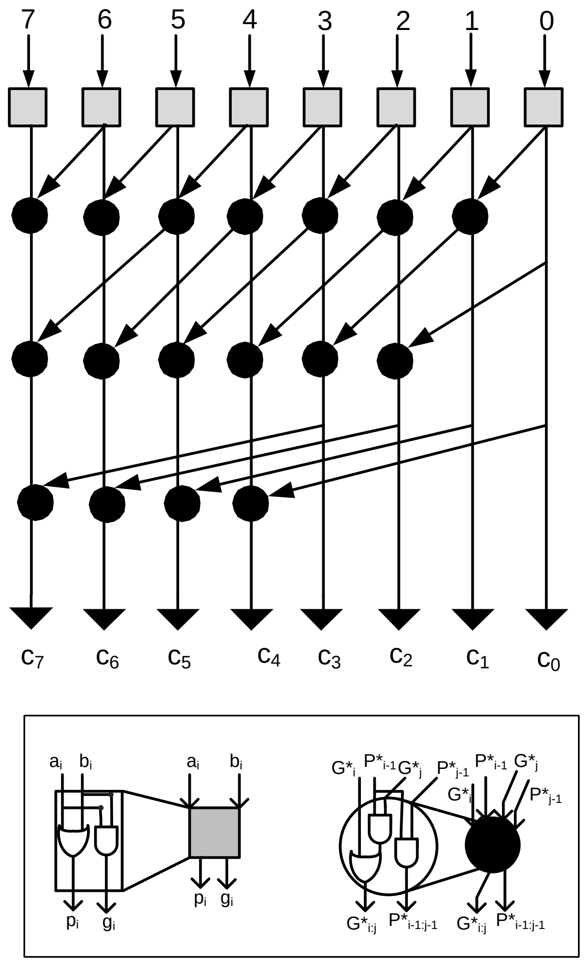


Figure 2.13: CLA based 8-bit Kogge-Stone prefix adder

## 2.5 Low Power Techniques in Binary Multiplier Design

Multipliers are logic circuits that are computationally heavy. Typically, a large number of logic gates with high transition activity are devoted to perform the multiplication operation. The logic transitions cause the logic gates to charge/discharge the load capacitance leading to dynamic power dissipation. This section provides a brief introduction to various sources of power dissipation in CMOS based designs. It is followed by preliminary information on existing reconfigurable multipliers and bypass techniques to minimize the dynamic power dissipation.

### 2.5.1 Dynamic Power Dissipation in CMOS based Circuits

The main source of power dissipation in CMOS based circuits is the dynamic power dissipation caused by switching activity of the logic circuits. Dynamic power dissipation is given by,

$$P_{av} = C_L V_{DD}^2 f_p \alpha \tag{2.17}$$

Where:

$C_L$ = charged load capacitance

$V_{DD}$ = supply voltage

$f_p$ = clock frequency

$\alpha$ = switching activity factor.

The dynamic power dissipated is thus proportional to the number of transitions occurring in a logic gate. Various power reduction methods to minimize the redundant switching$(\alpha)$ have been proposed in the literature as described later.

### 2.5.2 Power Consumption in Parallel Multipliers

In general, multipliers can be implemented as sequential or combinational circuits. However, in the current work, the focus is on parallel multipliers which are purely combinational circuits. Parallel multipliers are fairly complex circuits with a large transistor count and frequent switching of these transistors to carry out logic computations leads to large dynamic power dissipation. As elaborated in Section 3.2, parallel multipliers have the following computation steps: partial product generation, partial product reduction and vector merge addition. The partial product accumulation step, which predominantly comprises of adder units, dictates the overall computation delay, area and power consumption. An obvious technique to minimize power dissipation is to disable the unwanted computations in an adder. A number of methods to bypass the adders has been proposed and discussed in the literature [10–12].

### 2.5.3 Reconfigurable Binary Multiplier

In a binary multiplier, the die area and power consumption are largely dependent upon the word-size. Assuming that an application needs N-bit precision, then using a data path element of more than the required precision would result in wasted area and power. To overcome this problem, a twin-precision multiplier has been proposed in [13]. An attempt has been made to minimize the impact on delay and power of the N-bit multiplier by making as few modifications as possible to the conventional multiplier. This twin-precision scheme decomposes the N*N partial-product matrix into two N/2 * N/2 independent multiplications by configuring the appropriate partial products [14]. When it operates on N/2-bit operands however, large parts of the multiplier do not contribute to the final result although they may be active. Thus, the multiplier dissipates considerable dynamic power due to the switching activity involved in computing unwanted partial products. This problem is sought to be addressed in this work by using bypass computation cells that disable unnecessary computations.

### 2.5.4 A Review of Bypass Multiplier Architectures

Figure.2.14(a) illustrates the example of multiplication of two unsigned 4-bit numbers, where $A = a_3a_2a_1a_0$ is the multiplier and $B = b_3b_2b_1b_0$ is the multiplicand. In a conventional Braun array multiplier for example, the partial products are generated in parallel with the AND gates and added using a 1-bit full adder as illustrated in Fig.2.14(b) .

The adder circuits shown in Fig.2.14(b) tend to perform computation of the partial products even if their value is ‘0’ and this results in undesired signal transitions. These transitions can be avoided by disabling the respective adder cells which results in saving of power.

#### 2.5.4.1 Row-Bypass Scheme

Various techniques have been proposed from time to time to reduce the switching activity in array multipliers, of which bypass architectures are an offshoot. A simple approach to reduce the power consumption is to avoid unnecessary computations. Ohban [10] proposed a row-bypass scheme wherein some rows in the multiplier array are skipped to reduce the redundant switching activity. Figure 2.15 illustrates an implementation of a 4*4 Braun multiplier using

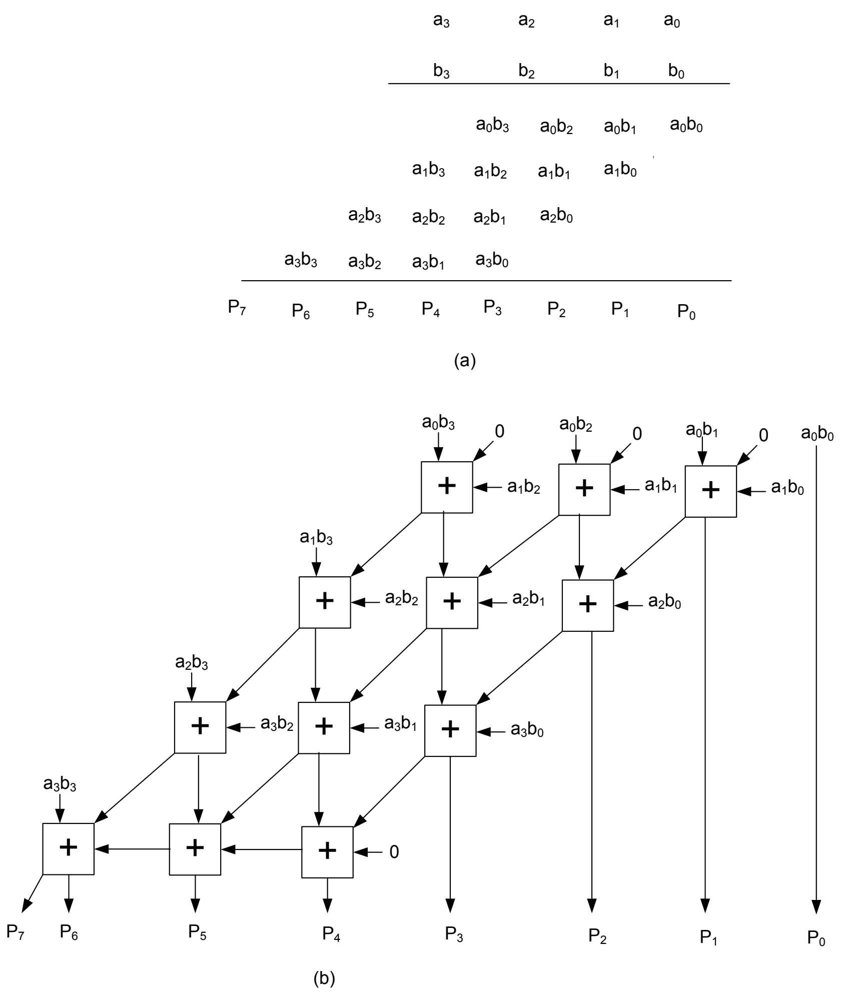


Figure 2.14: (a) An example of 4*4 array multiplication (b) Schematic diagram of 4*4 Braun multiplier

row-bypassing technique.

This scheme includes adder cells (denoted by AC and highlighted in Fig.2.15) to bypass the inputs to output whenever the row (horizontal) partial product is zero. The tri-state buffers augmented at the inputs of the adder cell disable unnecessary transitions by shutting down the full adder. The MUXes at the outputs of the adder cells automatically pass the carry-input and the sum of the previous addition to the next computational unit when the corresponding partial product is zero. A notable drawback in this scheme however is the additional logic circuitry required as highlighted in grey color in Fig.2.15.

A numerical example illustrating the multiplication of two 4-bit numbers using row-bypass scheme is shown in Fig.2.16. Since the partial products in the second row are zeroes, the corresponding computational units are turned off to save power. The partial products in the

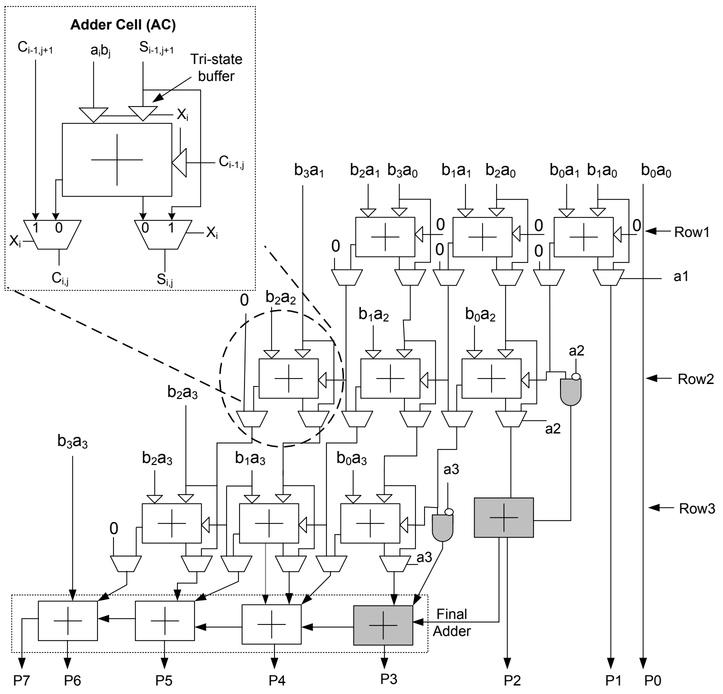


Figure 2.15: Schematic diagram of a 4*4 Braun multiplier using row-bypassing technique

first row are bypassed and added with partial products in the next level (third row). In a similar manner, the remaining partial products are reduced to form the final product.

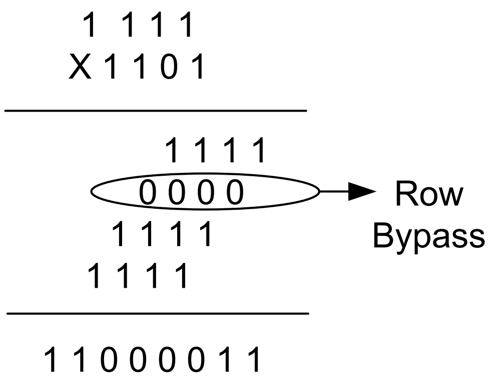


Figure 2.16: Numerical example of row-bypass scheme for 4*4 multiplier

#### 2.5.4.2 Column-Bypass Scheme

Wen [11] proposed a column-bypass scheme which avoids the adder operations in some columns instead of rows. In this approach, some columns in the partial product matrix can be skipped whenever their outputs are known. Consequently, the switching activity and therefore power dissipation are reduced. This technique has two important advantages: (i) It removes the extra compensating circuitry (ii) the modified full adder (MFA) unit is less complex than that used in the row-bypassing multiplier.

A typical 4*4 column-bypassing multiplier is illustrated in Fig.2.17 where the modified adder (MFA) cell is highlighted. The MFA cell skips the full adder whenever the partial product in the corresponding column is zero. This multiplier has less hardware complexity compared to the row-bypassing scheme also because it does not need to consider bypassing of the carry bit.

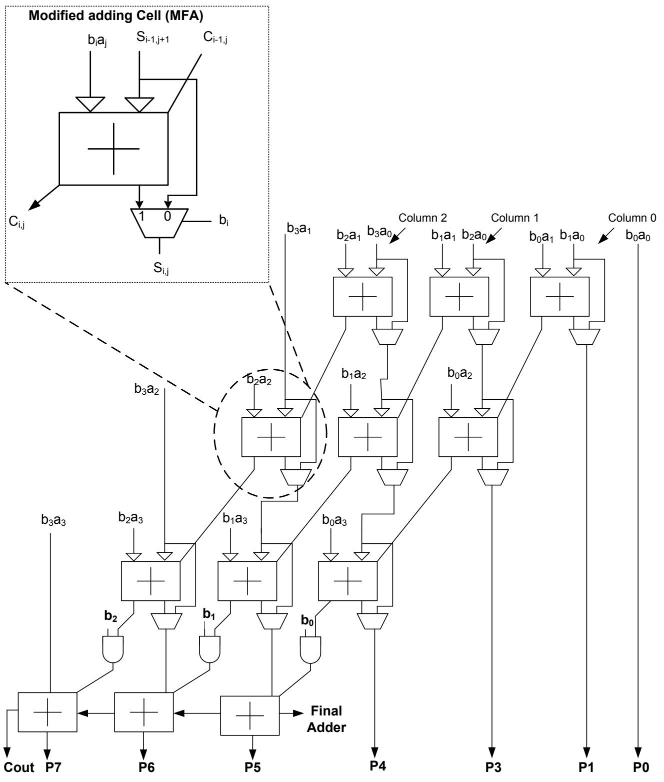


Figure 2.17: Schematic diagram of a 4*4 Braun multiplier with column-bypassing scheme

#### 2.5.4.3 Two-Dimensional Bypass Scheme

In a 2-dimensional bypassing multiplier, the computing logic cells skip the corresponding row and column depending on nullity of the partial products [12, 15]. Figure 2.18 shows the structure of the 4*4 Braun multiplier with 2-dimensional bypassing scheme [12].

To overcome the conflict that occurs when both row-bypassing and column-bypassing appear simultaneously, bypass adder cells (AC) incorporate additional logic. These bypass cells have the capability to bypass when either row and/or column element is zero, however with large circuit overhead. In view of this additional complexity, power saving tend to get re-

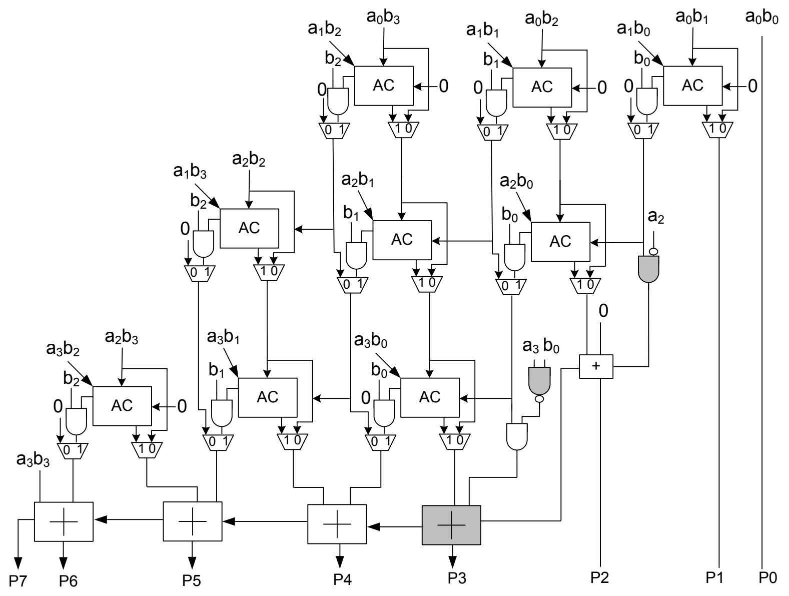


Figure 2.18: Schematic diagram of a 4*4 Braun multiplier with two-dimensional-bypassing scheme

duced. To overcome this, Hong [15] introduced two kinds of adder cells, namely, modified row-bypassing adder (MRBA) and two-dimensional bypassing adder (TDBA). The MRBA cells have row-bypassing capability while the TDBA cells are deactivated when either row or column partial product becomes zero.

## 2.6 A Review of Recursive Binary Multipliers

This section presents the mathematical modeling of the recursive binary multiplier. This is followed by various truncation schemes that have been used in the existing multiplier architectures.

### 2.6.1 Mathematical Analysis of Recursive Multiplier

Recursive multipliers based on Karatsuba-Ofman Algorithm (KOA) [16] are found to have a hierarchical architecture consisting of several sub-multipliers making them ideal for fixed-width multiplication.

Assume $A$ and $B$ to be two $2n$-bit unsigned numbers, where $A$ is the multiplicand and $B$ is the multiplier. $A$ and $B$ can be written as:

$$A = \sum_{i=0}^{2n-1} a_i.2^i \tag{2.18}$$

$$B = \sum_{j=0}^{2n-1} b_j.2^j \tag{2.19}$$

The recursive multiplication is performed by partitioning each of the operands into two equal portions of $n$-bit width. Based on this, the multiplicand $(A)$ is split into $A_H$ and $A_L$, while multiplier $(B)$ is divided into $B_H$ and $B_L$ respectively as mentioned in equations 2.20 and 2.21 given below:

$$A = A_H * 2^n + A_L \tag{2.20}$$
$$B = B_H * 2^n + B_L \tag{2.21}$$

The subscript $H$ denotes the most significant portion while $L$ denotes the lower significant portion of the corresponding binary numbers.

The product $(P)$ is written as follows:

$$P = A * B = (A_H * B_H) * 2^{2n} + (A_L * B_H + A_H * B_L) * 2^n + A_L * B_L \tag{2.22}$$

Thus, multiplication can be performed using four $n * n$ binary sub-multipliers, namely, $A_H * B_H$, $A_H * B_L$, $A_L * B_H$, and $A_L * B_L$, in parallel as shown in Fig.2.19. The partial products of all the individual sub-multipliers are reduced to product $(P)$ of $2n$-bit width using a reduction structure.

### 2.6.2 Truncation Schemes for Binary Multipliers

Several techniques [17–20] have been proposed in the past to achieve fixed-width multiplication. Among these, truncation techniques developed for recursive multipliers have been proven to be efficient as opposed to the array multipliers in terms of die area and power dissipation [17].

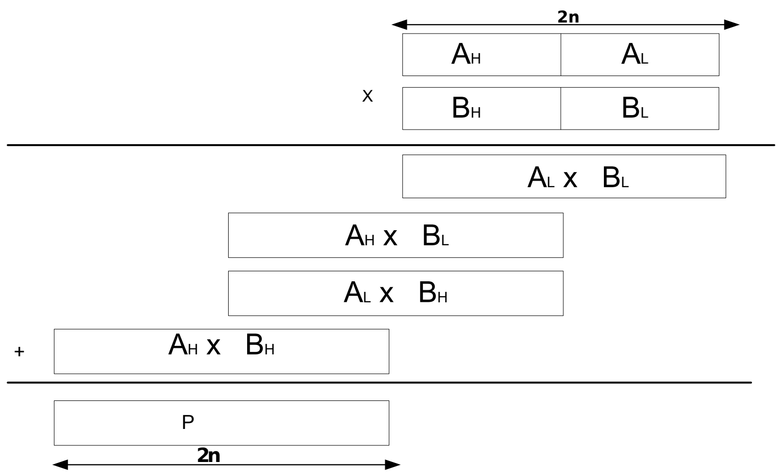


Figure 2.19: Schematic diagram of the original recursive multiplication scheme

#### 2.6.2.1 Truncation Schemes for Array Multipliers

In truncation schemes for array multipliers, the least significant bits of the partial product matrix are removed and a correction function, which is either constant or data-dependent, is added to compensate for the error [21–24].

Authors in [21, 23] present a constant correction technique where compensation function is based on the average value of the partial product bits which are not formed. This technique results in simple hardware which in turn leads to higher power savings. However, the error bounds obtained are high. To overcome this, a data-dependent correction technique proposed in [22, 25] adds a correction value based on the partial products corresponding to least significant column that are not formed. This technique, also referred to as variable correction, achieves a lower error bound compared to the constant correction schemes, though at the cost of the hardware complexity.

#### 2.6.2.2 Truncation Schemes for the Recursive Multiplier

Most of the truncation techniques targeted at array multipliers focus on modifying the multiplier structure. However, truncation schemes applied to recursive multipliers simply get rid of a least sub-multiplier $(A_L B_L)$ as shown in Fig.2.20 and replace it with a correction function which is data dependent.

Three correction schemes can be found in the literature. In scheme 1 [18], $A_H * B_L$ or $A_L * B_H$ sub-multiplier replaces $A_L * B_L$ while in scheme 2, the average value of $A_H * B_L$ and

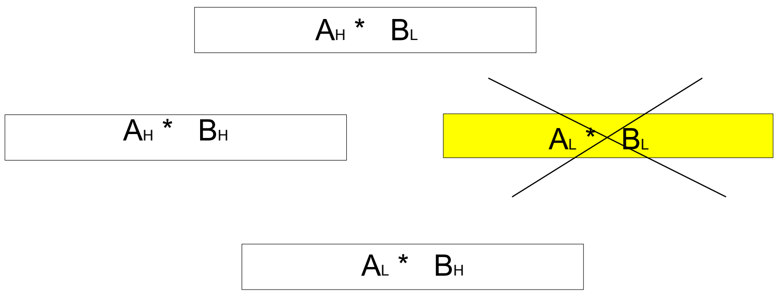


Figure 2.20: Sub-multipliers in a recursive multipliers

$A_L * B_H$ is used. In scheme 3, the most significant partial product bit of $A_L * B_L$, namely, $a_{n-1} * b_{n-1}$ forms the correction function. All these schemes are based on trial and error and the precision achieved is also fixed. In this work, a tunable correction function is proposed using a systematic approach and it‘s performance is compared with the existing ones.

## 2.7 Multipliers based on Logarithmic Number System

Most of the logarithmic multiplier schemes can be classified as iterative [26,27] or non-iterative [28–32]. Non-iterative multipliers have limited precision due to the usage of techniques such as piecewise linear approximation [33], memory look-up [34] or a combination of both [35] making them limited to only a few applications. On the other hand, iterative multipliers tend to improve the precision of the result with each successive iteration.

Mitchell [36] introduced the first iterative multiplier that was simple and flexible to meet the requirements of a wide range of applications. This multiplier however suffered from large relative error in the final result. Further, Mitchell approach cannot initiate next iteration until the completion of the present one. Babic [26] modified the Mitchell design by introducing greater pipeline-level parallelism with an objective to reduce the latency. However, his approach lead to reduced precision in each iteration due to the neglect of carry. Babic iterative multiplier (BIM) design was further improved by truncated error correction (TEC) method [27] which has an additional capability for speculative carry, thereby improving the precision. This however comes at the cost of area overhead.

To overcome these shortcomings, the present work combines carry speculation with an improved fractional predictor leading to a better precision when compared to the existing work.

The fractional predictor logic and its efficient precomputation contribute to the improved overall precision due to a fewer number of iterations required compared to the existing techniques. Further, precision of the multiplier improves as the number of iterations increases. Also, savings in hardware are achieved using the truncation scheme proposed in this work. The proposed and the existing logarithmic multipliers have been applied on an image sharpening algorithm and compared in the context of certain well-known image processing benchmarks such as Lena and Cameraman for performance.

### 2.7.1 Mathematical Analysis of MA Based Multiplier

This section presents the mathematical approach common for MA based multipliers [26,27,36] described below:

According to Mitchell, the binary representation of two n-bit input numbers $N_1$ and $N_2$ is given as :

$$\begin{cases} N_1 = 2^{k_1}(1+x_1) \\ N_2 = 2^{k_2}(1+x_2) \end{cases} \tag{2.23}$$

The characteristics of $N_1$ and $N_2$ are $k_1$ and $k_2$ respectively, representing the most significant operand bits with the value of '1'. Further, $x_1$ and $x_2$ denote fractional portions whose values lie in the range [0,1].

The base-2 logarithm of the product, $N_1$ and $N_2$ is written as

$$log_2(N_1 * N_2) = k_1 + k_2 + log_2(1+x_1) + log_2(1+x_2) \tag{2.24}$$

To compute the antilogarithm of equation (2.24), Mitchell proposed the following analytical expressions based on carry information from the fractional portion

$$N_1 * N_2 = 2^{k_1+k_2}(1+x_1+x_2) + 2^{k_1+k_2}(x_1 * x_2), \qquad x_1 + x_2 < 1 \tag{2.25}$$

and

$$N_1 * N_2 = 2^{k_1+k_2+1}(x_1 + x_2) + 2^{k_1+k_2}(x_1' * x_2'),$$
$$x_1 + x_2 \geq 1 \tag{2.26}$$

Where $2^{k_1+k_2}(x_1 * x_2)$ and $2^{k_1+k_2}(x_1' * x_2')$ are the correction terms.

Babic [26] ignored the carry altogether resulting in a simple and faster design with trade-off, however, in precision. Accordingly, Babic used the above expression (2.25).

Further, error due to the approximation was avoided by considering the relation given in equation (2.23) :

$$\begin{cases} x_1 * 2^{k_1} = N_1 - 2^{k_1} \\ x_2 * 2^{k_2} = N_2 - 2^{k_2} \end{cases} \tag{2.27}$$

Combining equations 2.25 and 2.27 results in,

$$N_1 * N_2 = 2^{k_1+k_2} + f_1 * 2^{k_2} + f_2 * 2^{k_1} + f_1 * f_2$$

Where $N_1 - 2^{k_1} = f_1$ ; $N_2 - 2^{k_2} = f_2$

The above equation is represented as

$$N_1 * N_2 = A^0 + f_1 * f_2 \tag{2.28}$$

where approximate product term $A^0 = 2^{k_1+k_2} + f_1 * 2^{k_2} + f_2 * 2^{k_1}$

The computation of term $f_1 * f_2$ given in equation (2.28) requires multiplication. Evidently, the product $N_1 * N_2$ gets simplified, if these terms are ignored which leads to sacrificing the precision. This was the approach adopted by Babic and TEC designs. Nevertheless, the correction term $(f_1 * f_2)$ can be computed in parallel with $A^0$, which however results in area overhead.

### 2.7.2 Hardware Architectures

The architecture of Babic multiplier [26] for one iteration is illustrated in Fig.2.21. It consists of components such as basic logarithmic converter blocks (BLBs), decoder and adders.

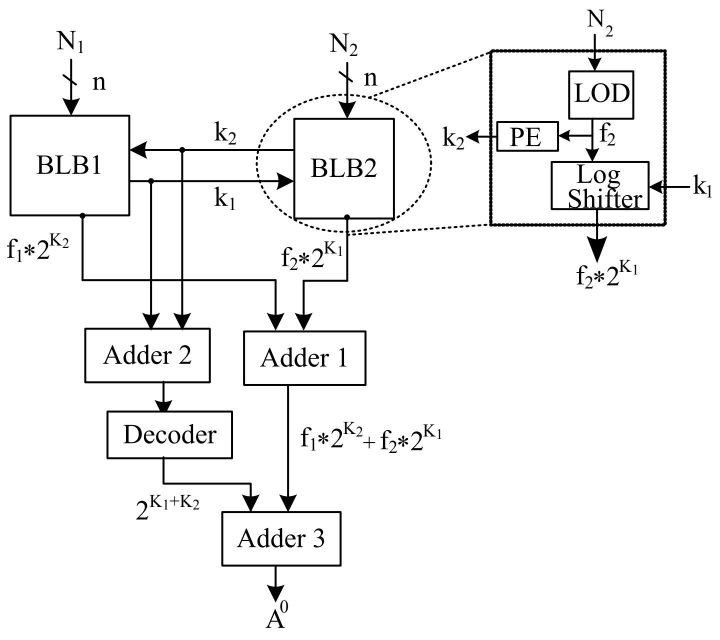


Figure 2.21: Functional diagram of Babic Iterative Multiplier (BIM)

A typical BLB includes a leading one detector (LOD), priority encoder (PE) and logarithmic shifter modules. It forms the fundamental module in the design of iterative multipliers and provides the characteristic $(k)$ and fractional portions $(f)$.

Based on the binary number $(N_2)$ and characteristic $(k_1)$, the BLB2 block highlighted in Fig.2.21 calculates the shifted fractional portion, $f_2 * 2^{k_1}$ and characteristic, $k_2$. Similarly, BLB1 block computes the fractional portion, $f_1 * 2^{k_2}$ and characteristic, $k_1$. The Adder 2 and Decoder logic calculate the integer portion of the product $(2^{k_1+k_2})$ while the summation of fraction portions $(f_1 * 2^{k_2}$ and $f_2 * 2^{k_1})$ computed using Adder 1 provides fractional portion $(f)$. The computation of the product $(A^0)$ is achieved by the addition of fractional portion $(f)$ and output of the Decoder using Adder 3 block. The inputs to next iteration are $f_1$ and $f_2$ which are obtained from the respective LOD circuits.

The truncated error correction approach suggested in [27] extends the BIM scheme with addition of fractional predictor (FP), shared logic, multi-operand addition (MOA) and mask as illustrated in Fig.2.22.

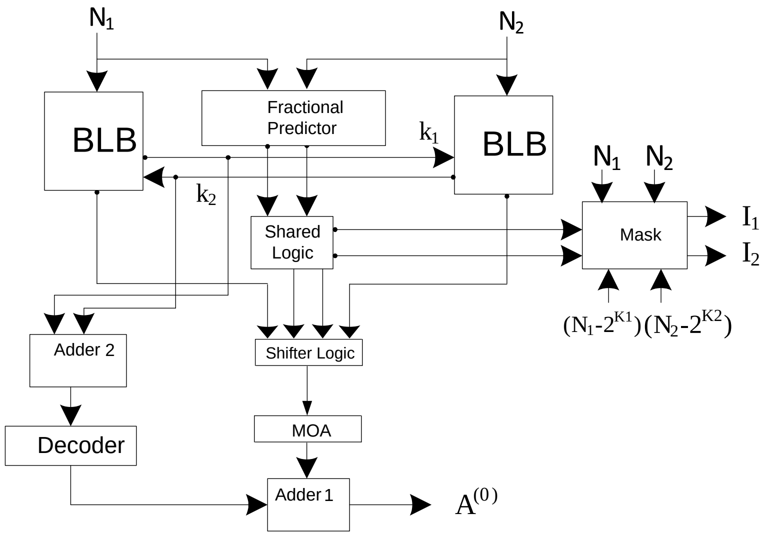


Figure 2.22: Functional diagram of truncated error correction (TEC) Scheme

The speculation of carry from fractional portion is carried out by a variable size FP while the shared logic gives the position of fractional bits that require error correction. The error correction itself is accomplished using shifter and multi-operand adder (MOA) and the inputs for next iteration are computed via mask logic. The shortcomings in TEC multiplier include extra hardware circuitry and lower error reduction rate for successive iterations.

## 2.8 Decimal Multiplication

Decimal multiplication typically have the following stages: (i) partial product generation (ii) partial product reduction and (iii) final product computation. A general architecture of 'digit-by-digit' multiplier is shown in Fig.2.23. The decimal multiplier accepts two BCD inputs A and B of m-bit width. In the partial product generation stage, the individual digits of multiplier and multiplicand are multiplied using the BDMs.

The reduction of partial products is accomplished using ripple-free binary CSA tree and conversion to decimal is achieved using the multi-operand BD converter. The final product is obtained after the addition of the decimal digits using decimal adder.

The 'digit-by-digit' multiplication is presented in the following Subsection 2.8.1 while various existing partial product generation and reduction schemes adapted for present designs are discussed in Subsection 2.8.2.

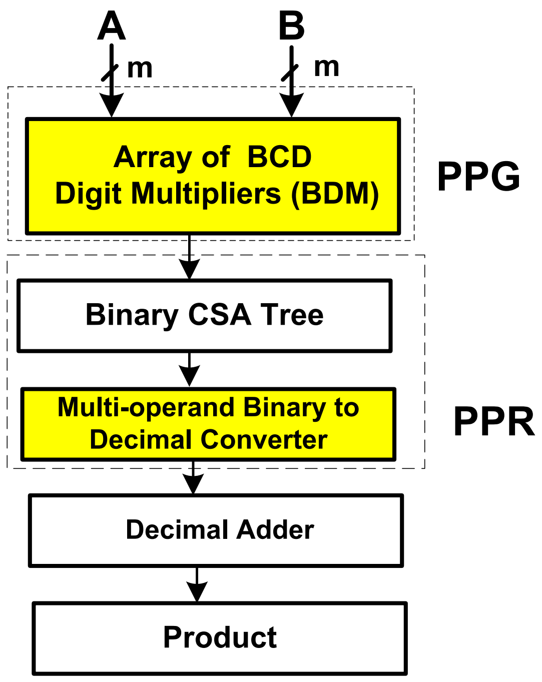

A
B
m
m
Array of BCD Digit Multipliers (BDM)
PPG
Binary CSA Tree
Multi-operand Binary to Decimal Converter
PPR
Decimal Adder
Product

Figure 2.23: A top-level architecture of 'digit-by-digit' multiplication

### 2.8.1 'Digit-by-Digit' Multiplier

Decimal (BCD) arithmetic computations are generally sluggish (slow) and tend to occupy more silicon area. This has led to efforts to improve decimal architectures that result in high performance and compact arithmetic circuits [37]. For example, microprocessors such as IBM Power PC [38] and IBM z10 [39] include dedicated decimal hardware units.

Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic is multiplication. While a large body of literature on decimal arithmetic covers serial multiplication, parallel ('word-by-digit') [40–43] and ('digit-by-digit') [44, 45] multiplication has also been reported recently. Decimal (BCD) 'digit-by-digit' multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in a significant improvement in the multiplier performance [46].

A step by step implementation of 4*4 'digit-by-digit' multiplication [44] is illustrated in Fig.2.24. Multiplication of each digit of the multiplicand with the digit of multiplier is performed using the BDM.

For example, multiplication of $A_1$ and $B_1$ is highlighted in the dotted circle of the figure.

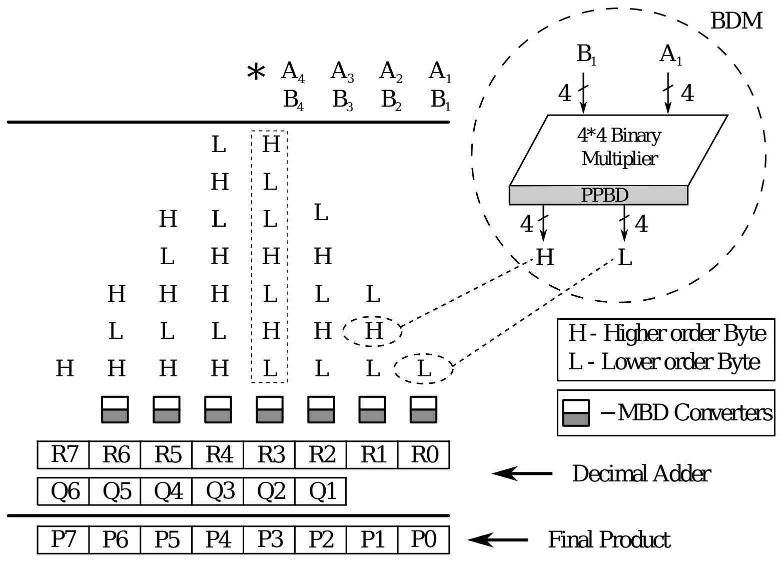


Figure 2.24: Example of 4*4 ‘digit-by-digit’ multiplication using BDMs

The output of the BDM results in most significant digit and least significant digit denoted by H and L respectively. A typical BDM is composed of a 4*4 binary multiplier and a partial product binary to decimal (PPBD) converter. Most of the previous work available in literature is focused on PPBD converters at partial product generation stage which is discussed in Section 2.8.2.

The individual decimal partial product columns (one such column is highlighted with dotted rectangle in Fig.2.24) are compressed in parallel by using a tree of binary carry save adders (CSAs) resulting in a binary number as output of each column. The conversion from binary to decimal is carried out using multi-operand BD (MBD) converters resulting in rows of decimal digits R0-R7 and Q1-Q6 which are eventually compressed using a decimal adder to obtain the final product (P0-P7).

## 2.8.2 A Review of Partial Product Generation and Reduction Schemes

### 2.8.2.1 Partial Product Generation (Binary Product to BCD conversion)

The algorithm proposed in [44] converts a 7-bit binary number ($p_6p_5p_4p_3p_2p_1p_0$) to a 2-digit BCD number ($D_H$ and $D_L$ ) to support high performance decimal multiplication. This algorithm calculates the contributions for lower BCD digit $(D_L)$ and the higher BCD digit $(D_H)$ from each of the input binary bits as shown in Table 2.1.

Table 2.1: Principle of Binary to BCD conversion

| 80 | 40 | 20 | 10 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_2$ | $p_1$ | $p_0$ |
| 0 | 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_4$ | 0 |
| | | | | $p_6$ | 0 | $p_5$ | 0 |
| | | | | $p_3$ | 0 | 0 | 0 |
| $d_{h3}$ | $d_{h2}$ | $d_{h1}$ | $d_{h0}$ | $d_{l3}$ | $d_{l2}$ | $d_{l1}$ | $d_{l0}$ |

The first row in the Table shows the BCD weights. The binary numbers $p_3, p_2, p1$ and $p_0$ are retained in their position as their weights are same as the corresponding weights in the original binary number. However, the weights 16, 32 and 64 corresponding to $p_4$, $p_5$ and $p_6$ are decomposed into $(10,4,2)$, $(20,10,2)$ and $(40,20,4)$, respectively. The four columns in the right consisting of BCD digits are summed using BCD adder leading to the BCD digit $D_L$ $(d_{l3}d_{l2}d_{l1}d_{l0})$ while the resulting carry is added to the BCD digit that is present in the left three columns leading to $D_H$ $(d_{h3}d_{h2}d_{h1}d_{h0})$.

Work in [47] modifies the architecture in [44] by adding the contributions in a BCD fashion. This design partitions or splits the binary input into two sub-parts, three MSBs and four LSBs. It calculates the contributions for the two BCD digits and adds them in a BCD fashion to get the final result.

Work presented in [48] proposes two schemes, 'three-four split' and 'four-three split' binary to BCD converters. The 'three-four split' algorithms have optimized $D_L$ and $D_H$ generator blocks resulting in better performance in terms of area, delay and power. An illustration of the 'three-four' split algorithm is provided in Fig.2.25.

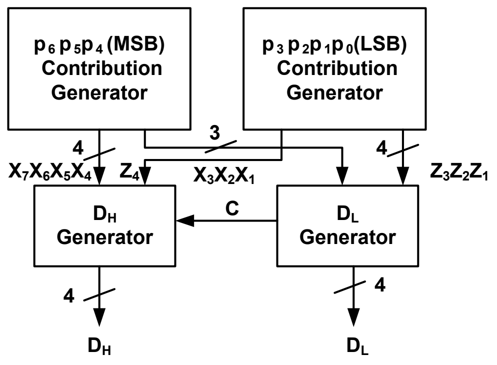

p6p5p4(MSB) Contribution Generator
p3p2p1p0(LSB) Contribution Generator
3
4
4
X7X6X5X4
Z4
X3X2X1
Z3Z2Z1
DH Generator
C
DL Generator
4
4
DH
DL

Figure 2.25: Block diagram of 'three-four split' binary to BCD converter

The 'four-three split' design partitions the 7-bit binary input into four MSBs and three

LSBs. Since the LSBs do not contribute to the higher BCD digit the LSB contribution generator is removed resulting in area savings. However, this comes at the cost of increased complexity of MSB contribution generator as shown in Fig.2.26. The ‘three-four split’ is faster than the ‘four- three split’ whereas the ‘four-three split’ results in a more area efficient design.

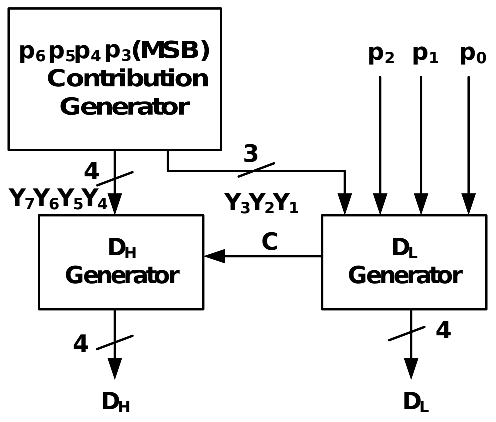


Figure 2.26: Block diagram of ‘four-three split’ binary to BCD converter

Work published in [49] adapted a binary-to-BCD conversion cell proposed by Nicoud [50]. Although, it was Dadda [51] who first showed that an iterative array of Nicoud’s cells can be used to design multi-operand BD converters at PPR level mentioned in later Section. This idea was used in [49] however at PPG level to design PPBD converter.

Certain shortcomings, however, have been recognized in these methods such as (i) redundant contribution blocks [47,48] and (ii) large area consumption [49]. To alleviate these, two partial product BD converters, namely ‘high performance’ PPBD (HPPPBD) and ‘low area’ PPBD (LAPPBD) converters, are proposed in this work (Section 6.2).

#### 2.8.2.2 Partial Product Reduction

Partial product reduction in the first stage of ‘digit-by-digit’ multiplier is achieved using a binary CSA structure [52]. The binary result of each partial product column is subsequently converted to decimal (BCD) using a multi-operand BD converter consisting of iterative connection of Nicoud cells [50] suggested by Dadda [51]. A typical Nicoud (ND) cell would accept a 4-bit binary input $(b_j)$, multiplies it by two, and then adds it to $b_i$ as depicted in Fig. 2.27(a). Thus the computation of BCD (decimal) outputs, $b_0$ (higher digit) and $D_0$ (lower digit) is carried out using the relation $\{b_0, D_0\} = 2.b_j + b_i$ where the maximum values of $b_0$ and $D_0$

are $(1)_{10}$ and $(9)_{10}$ respectively.

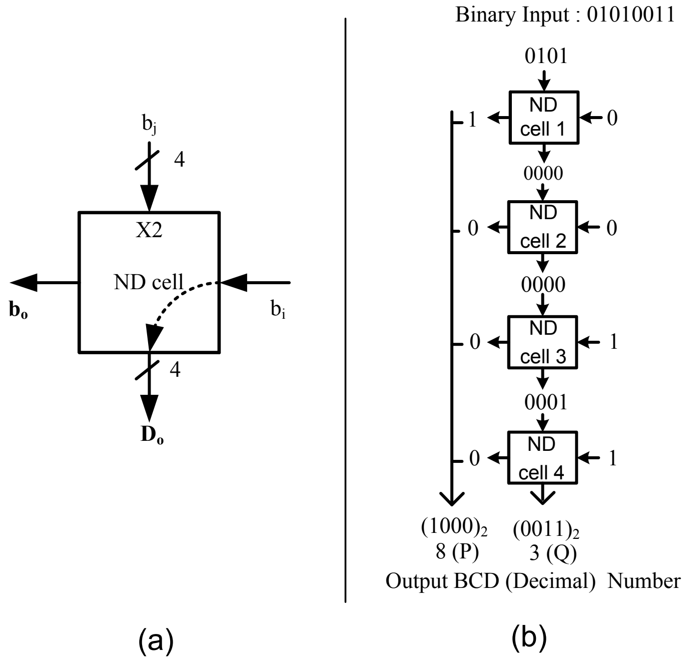


Figure 2.27: (a) Compact notation of Nicoud cell (b) Linear array of Nicoud cells to form Dadda multi-operand BD converter

An example to convert a binary number to two BCD digits ($b_0$ and $D_0$) is illustrated in Fig.2.27(b). Since the binary number $(1010011)_2$ to be converted here is larger than $(19)_{10}$, four Nicoud cells are required to realize the converter. As illustrated in Fig.2.27(b), the input to the Nicoud cell is restricted to $(1001)_2$. Hence the 3 MSBs of binary input along with ‘0’ prepended $(0101)_2$ is accepted as $b_j$ and the next significant binary input ‘0’ as $b_i$ resulting in the outputs $(1)_2$ and $(0000)_2$. The 4-bit output $(0000)_2$ of cell 1 along with the next significant binary input ‘0’ form input to cell 2, resulting in $(0)_2$ and $(0000)_2$. Similarly, the 4-bit output of each subsequent cell along with residual 1-bit binary input feeds the decimal input of the following cell resulting in higher (P) digit $(1000)_2$ and lower (Q) digit $(0011)_2$. In general, binary number of any operand width can be converted to decimal by a linear arrangement of Nicoud cells.

The limitation of Nicoud cells however is their latency and thus the delay of multi-operand BD converter increases with the size of the binary number. To mitigate this, a hybrid multi-operand BD converter is proposed in this work. A detailed discussion of the partial product reduction scheme is presented in Section 6.3 .

## 2.9 Conclusions

In this chapter, the necessary background material about the multipliers based on different number systems is presented. This knowledge is required to understand the subsequent chapters included in the thesis. The objective of this chapter was to provide a quick introduction to various architectures such as fixed width binary multipliers, logarithmic and BCD multipliers. The multipliers based on binary and logarithmic number offers a low power alternative solution in error resilience applications. On the other hand decimal arithmetic has been increasing used in the financial applications where precision is very important. Finally, the overall research approach followed in this thesis is presented.

# Chapter 3

# An Efficient Reconfigurable Binary Multiplier with 2-Dimensional bypassing

## 3.1 Introduction

Recent developments in digital signal processing (DSP) have necessitated development of reconfigurable binary multiplier architectures that can dynamically adapt to varying application needs [53]. For example, a typical digital system may need to switch between one application that requires 4-bit accuracy to another application that needs 8-bit accuracy. This could partially be compensated by having two multipliers, each of precise bit-width, and having smallest bit-width multiplier that is adequate for current multiplication. Though, this approach optimizes the multiplier in terms of delay, it results in area and power overhead in view of multiple multiplier instances. Thus, one of the objectives in this chapter is to design a data path component that can be configured to perform either one $N$ or two independent $N/2$ multiplication operations.

A wide variety of low-power array multiplier architectures exist in literature that are listed in [10,11,54]. A simple and straightforward method to save power in these multipliers is to use bypassing technique [10] which reduces the switching activity by avoiding unnecessary computations. The second objective of this chapter is to design novel two-dimensional bypassing computational cells and incorporate them into the reconfigurable multiplier mentioned above to reduce the switching activity further. Also, a reconfigurable Ladner-Fisher prefix adder that simplifies the final product computation is included in the multiplier.

The proposed multiplier architecture is described in section 3.2 while synthesis results of the performance of various multipliers are compared in section 3.3.

## 3.2 Proposed Reconfigurable Binary Multiplier Architecture

In this section, a bit-width aware reconfigurable multiplier architecture with two-dimensional bypassing is proposed. In normal operation mode, this reconfigurable multiplier performs 8-bit multiplication while for applications where accuracy can be relaxed, it can perform 4-bit multiplication with only a fraction of the energy of 8-bit multiplication being expended. Also, when performing two 4-bit parallel multiplication within a 8-bit multiplier, only one half of the logic is used.

Further, to reduce the dynamic power, a new 2-dimensional bypassing technique is incorporated into the multiplier architecture. The bypass technique uses selective disabling of computation cells when the column and/or row partial products are zero. This is achieved by incorporating the new bypassing computational cells that improve the power efficiency and also facilitate reconfigurability of the multiplier.

A block diagram of the proposed reconfigurable multiplier with bypass cells is shown in Fig.3.1 below. As can be seen, it has two inputs each of m-bit width. The partial product (PP) generation is accomplished using AND gates while the PP matrix reduction is achieved using novel 2-dimensional bypass adder cells. The bypass adder logic, while reducing the PPs into two rows, also helps in minimizing the latency and power dissipation by disabling the unnecessary PPs. In addition, capability to reconfigure is built into the bypass computation cells. To further improve the speed of operation, two rows are reduced to a final product using a scalable Ladner-Fisher prefix adder.

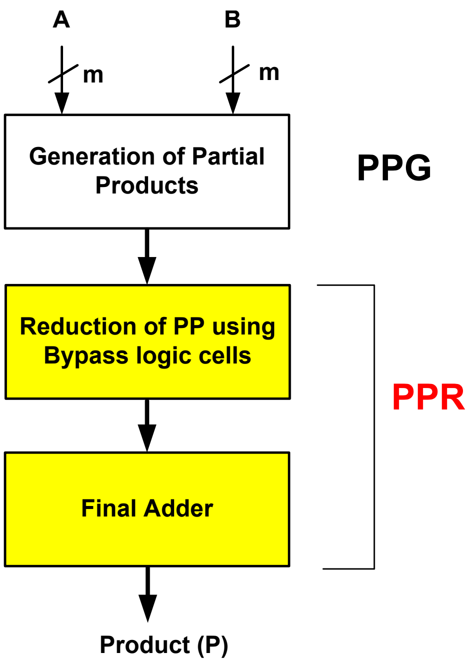

A
B
m
m
Generation of Partial Products
PPG
Reduction of PP using Bypass logic cells
PPR
Final Adder
Product (P)

Figure 3.1: Block diagram of proposed reconfigurable binary multiplier

### 3.2.1 Partial Product Arrangement

Figure.3.2 illustrates the partial product arrangement in a 8*8 array multiplier while their reduction is shown in Section 3.2.2. Based on the configuration mode given in Table 3.1, the partial product matrix in Fig.3.2 can be configured to perform either as one $8*8$ multiplier or two independent $4*4$ multipliers.

Table 3.1: Proposed configuration modes of an array multiplier

| Configuration Mode, CM | Function Description |
|---|---|
| $CM1$ | one 8*8 full-width multiplier |
| $CM0$ | two independent 4 * 4 full-width multipliers |

#### 3.2.1.1 Configuration Mode 1 (CM1) :

In configuration mode 1, one $8*8$ multiplication can be performed as illustrated in Fig.3.2. The PPG operation generates a total of 64 PPs, which are arranged in a matrix form as shown in Fig.3.2. To eliminate the redundant switching activity, all the PPs are reduced using bypass computation cells as discussed in section 3.2.2.

$$
\begin{array}{cccccccccccccccc}
 & & & & & & & & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & & & * & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
\hline
 & P_{77} & P_{76} & P_{75} & P_{74} & P_{64} & P_{54} & P_{44} & P_{70} & P_{33} & P_{32} & P_{31} & P_{30} & P_{20} & P_{01} & P_{00} \\
 & & P_{67} & P_{66} & P_{65} & P_{55} & P_{45} & P_{71} & P_{61} & P_{60} & P_{23} & P_{22} & P_{21} & P_{11} & P_{10} & \\
 & & & P_{57} & P_{56} & P_{46} & P_{72} & P_{62} & P_{52} & P_{51} & P_{41} & P_{13} & P_{12} & P_{02} & & \\
 & & & & P_{47} & P_{73} & P_{63} & P_{53} & P_{43} & P_{42} & P_{50} & P_{40} & P_{03} & & & \\
 & & & & & P_{37} & P_{36} & P_{35} & P_{34} & P_{24} & P_{05} & P_{04} & & & & \\
 & & & & & & P_{27} & P_{26} & P_{25} & P_{15} & P_{14} & & & & & \\
 & & & & & & & P_{17} & P_{16} & P_{06} & & & & & & \\
 & & & & & & & & P_{07} & & & & & & & \\
\hline
P15 & P14 & P13 & P12 & P11 & P10 & P9 & P8 & P7 & P6 & P5 & P4 & P3 & P2 & P1 & Po \\
\hline
\end{array}
$$

Figure 3.2: Partial product matrix in configuration mode 1

#### 3.2.1.2 Configuration Mode 0 (CM0) :

In configuration mode 0, two parallel $4*4$ multiplications can be performed as illustrated in Fig.3.3. The partial products (colored in gray and black) that contribute to independent $4*4$ multiplications are given in the upper half of the PP matrix while those (enclosed in dotted box in Fig.3.3) that do not contribute to the product are shown in the lower half of the PP matrix. The computation cells corresponding to these PPs are turned-off, thus leading to power saving. This has been achieved using new bypass computation cells discussed below.

### 3.2.2 Partial Product Reduction using 2-Dimensional Bypass Cells

Figure 3.4 illustrates the proposed bypassing architecture for an 8*8 array multiplier. The reduction of partial products generated (shown in Fig.3.3) is carried out using the new adder bypass logic cells besides the existing cells. The bypassing cells incorporated into the multiplier skip the redundant computations whenever the row (horizontal) or the column (vertical) partial product is zero. Two adder cells namely, TDBA and MRBA with bypassing capability, are adapted from previous design [15] for this purpose and used. The TDBA cell has the capability to bypass when either row or column element is zero while MRBA bypasses when only the row element is zero.

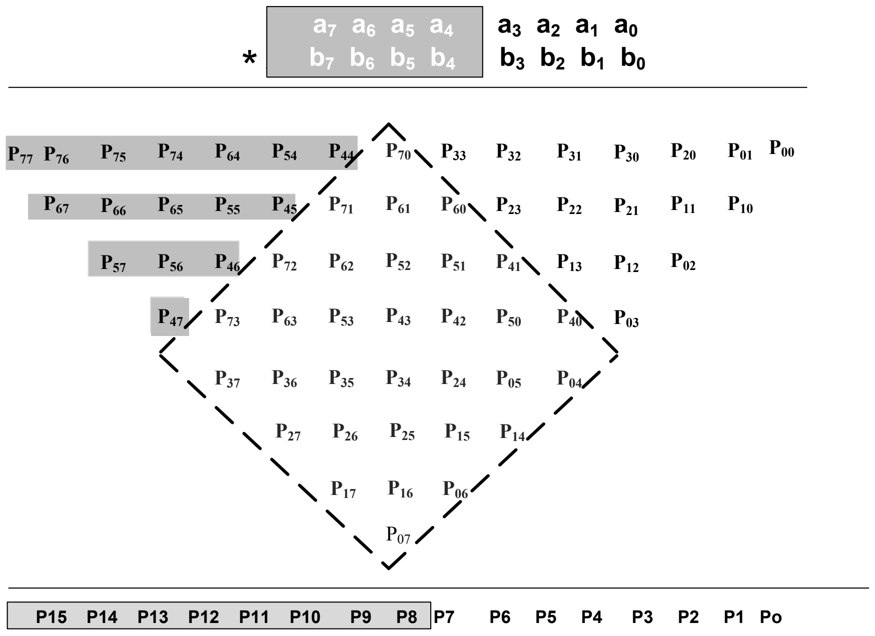

a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P77 P76 P75 P74 P64 P54 P44 P70 P33 P32 P31 P30 P20 P01 P00
P67 P66 P65 P55 P45 P71 P61 P60 P23 P22 P21 P11 P10
P57 P56 P46 P72 P62 P52 P51 P41 P13 P12 P02
P47 P73 P63 P53 P43 P42 P50 P40 P03
P37 P36 P35 P34 P24 P05 P04
P27 P26 P25 P15 P14
P17 P16 P06
P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po

Figure 3.3: Partial product matrix in configuration mode 0

In mode 0 (CM 0), the bypass cells are intended to provide reconfigurability to the multiplier besides minimizing the redundant switching activity. Therefore, two new adder cells, reconfigurable two-dimensional bypass cell (RTDBC) and reconfigurable row-bypass cell (RRBC) are proposed in this work. These reconfigurable computation cells can be configured according to the mode of operation using configuration mode bit, *CM*. The RTDBC cell is deactivated when the corresponding row or column partial product is zero. Conversely, RRBC skips the computation unit when the corresponding row partial product is zero. A detailed implementation of these cells is provided in Section 3.2.2.1 & 3.2.2.2. With the proposed RTDBC and RRBC, the power saving of the multiplier is higher than the conventional bypassing architectures as will be demonstrated later.

The operation of the multiplier that depends on the configuration mode can be described as follows: When *CM* = ‘1’, one 8*8 full-width multiplication is carried out. Since in this mode all the PPs contribute to the final result $(P0 - P15)$ it can be said that RTDBC and RRBC work as normal bypass logic cells. The reconfigurable cells along with TDBA and MRBA are turned off that depends on the nullity of partial products.

Conversely, when *CM* = ‘0’, two independent 4*4 multiplications can be performed. In this mode, the partial products enclosed in dotted box in Fig.3.3 do not contribute to the final result.

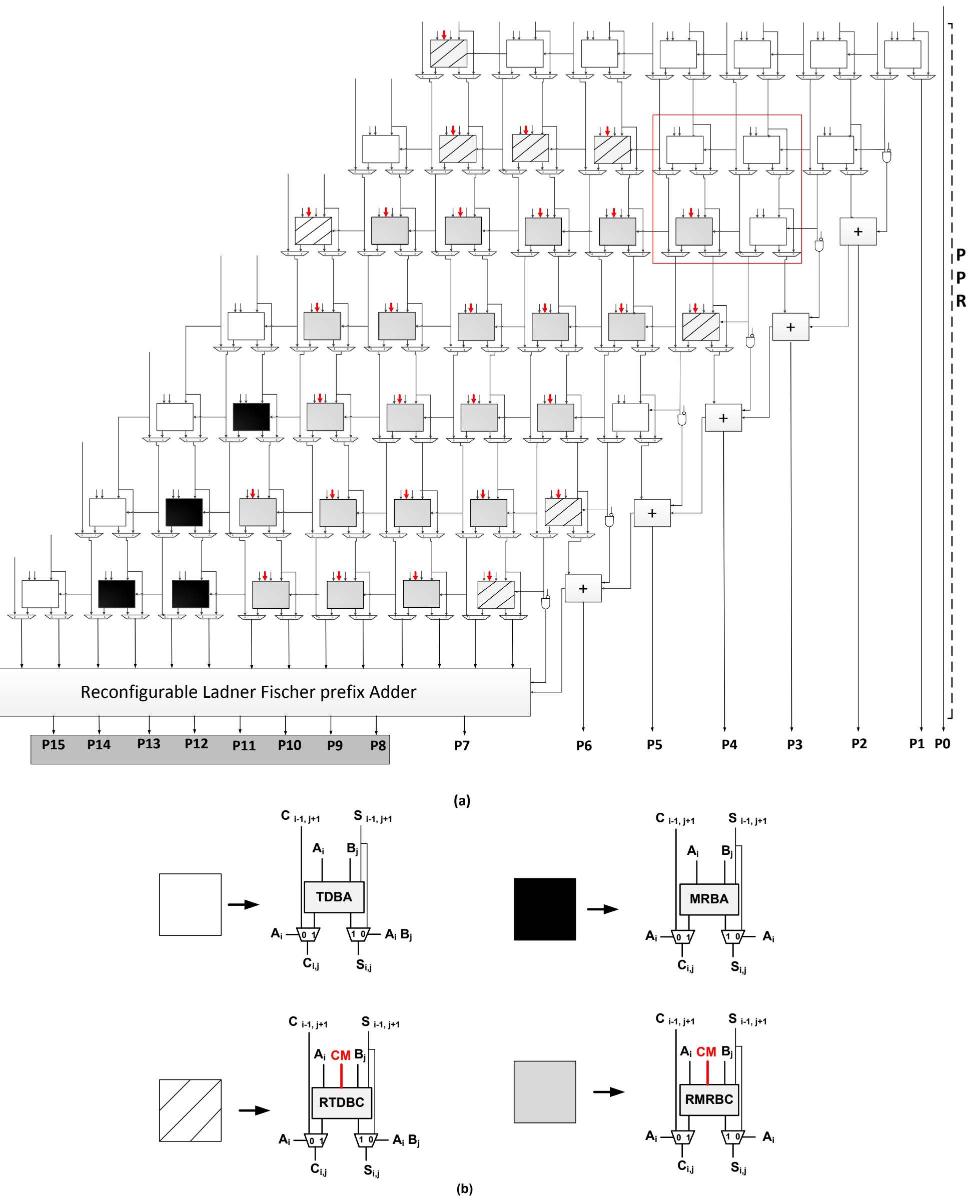


Figure 3.4: (a) Proposed reconfigurable multiplier architecture with bypass computation cells (b) Reconfigurable row and column bypass cells with mode bit, CM

Therefore, RTDBC and RRBC computation cells corresponding to these PP bits are disabled in Fig.3.4. This helps in minimizing the unnecessary transition activity in the multiplier.

Finally, the PPs out of the bypass cells are accumulated to two rows regardless of the mode of operation. These rows are eventually reduced to final product using a scalable Ladner-Fisher prefix adder discussed in section 3.2.3.

The reason for implementing the proposed multiplier with different cells (two dimensional and row bypassing) is to avoid carry problem that occurs when both row and column-bypassing are applied simultaneously. For example, in Fig.3.5 (a section of proposed multiplier high-

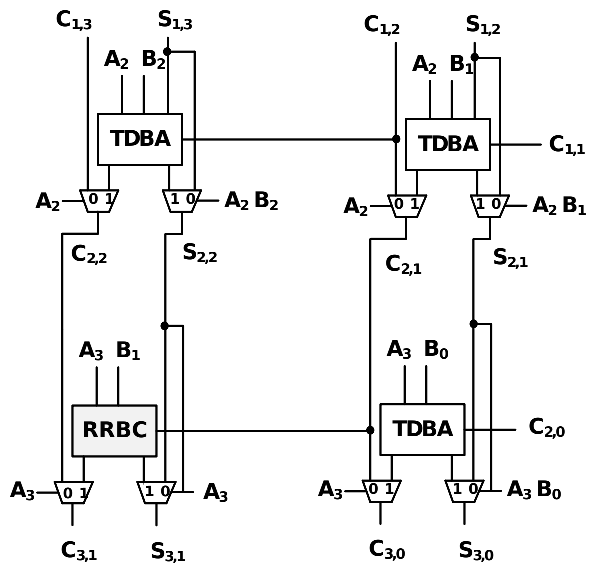


Figure 3.5: A section of the proposed reconfigurable multiplier

lighted in dotted box in Fig.3.4), assume both column input $B_1$ and row input $A_2$ are zero and the carry-input $C_{1,2}$ is '1'. In such a case, the respective computation cell will be disabled and carry output $C_{2,1}$ is '1' due to bypassing. If $A_3$ is '1', $C_{2,1}$ is lost due to column-bypassing. This introduces errors in the multiplication. To overcome this problem, the proposed multiplier has 2-dimensional cells (TDBC / RTDBC) only in the first two rows and the first and the last columns in which the carry problem does not occur, as illustrated in Fig.3.4. The remaining portion of the multiplier should have different types of logic cells which are different from RT-DBCs. The carry problem is solved using row-bypassing cells (MRBA/RRBC) built with less number of logic gates.

#### 3.2.2.1 Reconfigurable Two-Dimensional Bypass Cell (RTDBC)

The bypassing scheme in existing two-dimensional bypass approaches [12,54] consists of a full adder and additional logic circuitry. The reconfigurable two-dimensional bypass cell (RTDBC) in this work however is designed as per the Table 3.2. Clearly, the truth table of the RTDBC cell is simple and can be implemented only with a few logic gates. In RTDBC, the unnecessary logic computations are disabled using internal tri-state buffers (ITBs). The EX-OR used in this cell is a 4-transistor type with cascaded inverter for driving the output [55]. As a result, the area and power overhead are reduced.

Figure 3.6 shows a schematic of the reconfigurable two-dimensional bypass cell. As illus-

Table 3.2: Truth Table of reconfigurable two-dimensional bypass adder cell (RTDBC)

| $CM$ | A | B | $C_{out}$ | $S_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | $C$ | $S_{in}$ |
|  | 0 | 1 | $C$ | $S_{in}$ |
|  | 1 | 0 | $C$ | $S_{in}$ |
|  | 1 | 1 | $C$ | $S_{in}$ |
| 1 | 0 | 0 | $C$ | $S_{in}$ |
|  | 0 | 1 | $C$ | $S_{in}$ |
|  | 1 | 0 | 0 | $S_{in}$ |
|  | 1 | 1 | $C_{in} + S_{in}$ | $C_{in} \odot S_{in}$ |

trated in Fig. 3.6, ITBs that are augmented with 2-input NAND gate and inverters form the input to the MUXs. When the input $CM$ is ‘0’, both the NAND and inverter logic are disabled and the MUX logic passes $C$ and $S_{in}$ to the outputs $C_{out}$ and $S_{out}$ respectively. For the case when $CM$ and the row input A is ‘1’, the output $C_{out}$ depends on the column input $B$ as shown in Fig. 3.6 . However, when row input $A$ is ‘0’, the inverter logic gates are disabled and the output $C_{out}$ depends on input $C$ while $S_{out}$ follows the input $S_{in}$. Hence, using RTDBC, power consumption can be reduced.

#### 3.2.2.2 Reconfigurable Row Bypass Cell (RRBC)

The reconfigurable row bypass cell (RRBC) has the capability to bypass when the row element is zero. The truth table of the RRBC illustrated in Table 3.3 is simple and can be implemented with a few logic gates instead of using a full adder and additional logic. Further, RRBC does not have the carry problem which is unlike RTDBC. To reduce the power consumption in RRBC, ITBs are used to disable computation in cases where it is unnecessary. Figure 3.7 illustrates the schematic of RRBC.

Table 3.3: Truth Table of reconfigurable row-bypass cell (RRBC)

| $CM$ | $A$ | $B$ | $C_{out}$ | $S_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | $C$ | $S_{in}$ |
| 0 | 0 | 1 | $C$ | $S_{in}$ |
| 0 | 1 | 0 | $C$ | $S_{in}$ |
| 0 | 1 | 1 | $C$ | $S_{in}$ |
| 1 | 0 | 0 | $C$ | $S_{in}$ |
| 1 | 0 | 1 | $C$ | $S_{in}$ |
| 1 | 1 | 0 | $C_{in}.S_{in}$ | $C_{in} \oplus S_{in}$ |
| 1 | 1 | 1 | $C_{in} + S_{in}$ | $C_{in} \odot S_{in}$ |

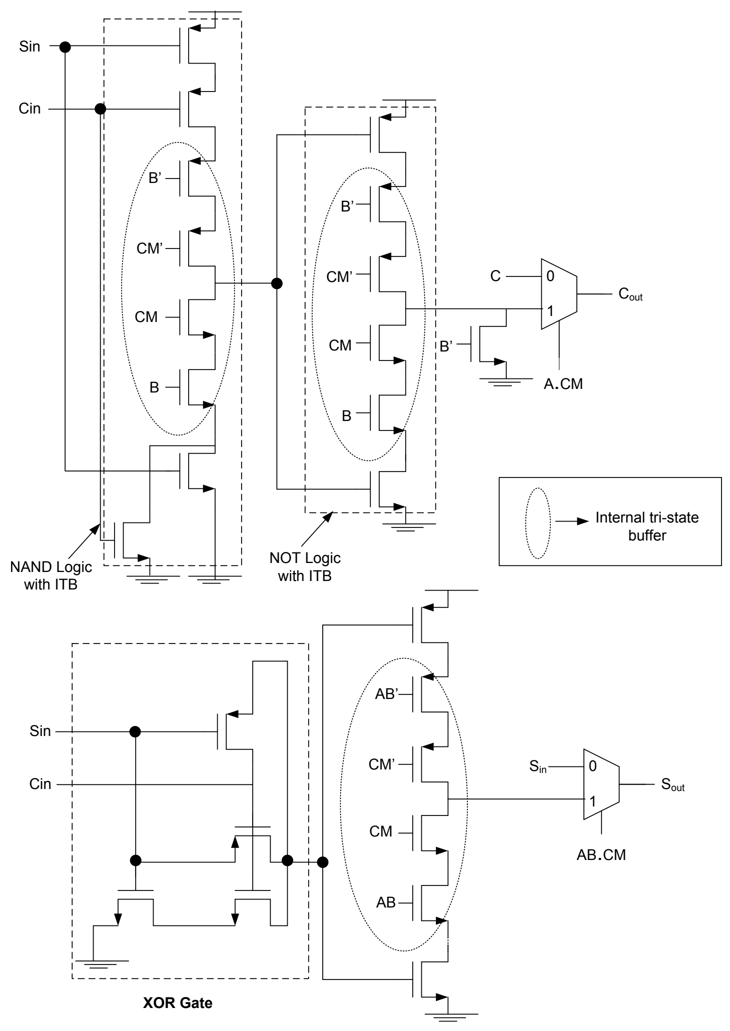


Figure 3.6: Logic schematic of RTDBC

A typical RRBC consists of ITBs, 2-input NOR gate, 2-input NAND gate, and inverters in front of MUXs. The EX-OR and EX-NOR gates used in this cell are 4-transistor type with cascaded inverter for driving the output [55]. When the input *CM* is '0', the ITB disables the logic gates and the MUX logic passes *C* and $S_{in}$ to the outputs $C_{out}$ and $S_{out}$ respectively. For the case when *CM* is '1' and the row input *A*, the output $C_{out}$ depend on the column input *B* as shown in Fig.3.7. However, when row input *A* is '0', the inverter logic gates are disabled and the output $C_{out}$ depends on input *C* while $S_{out}$ follows the input $S_{in}$.

### 3.2.3 Reconfigurable Ladner-Fisher Prefix Adder

The partial products are reduced to two rows using bypass computation cells at PPR level and are converted to a product using a final adder. This adder implementation is based on Ladner-

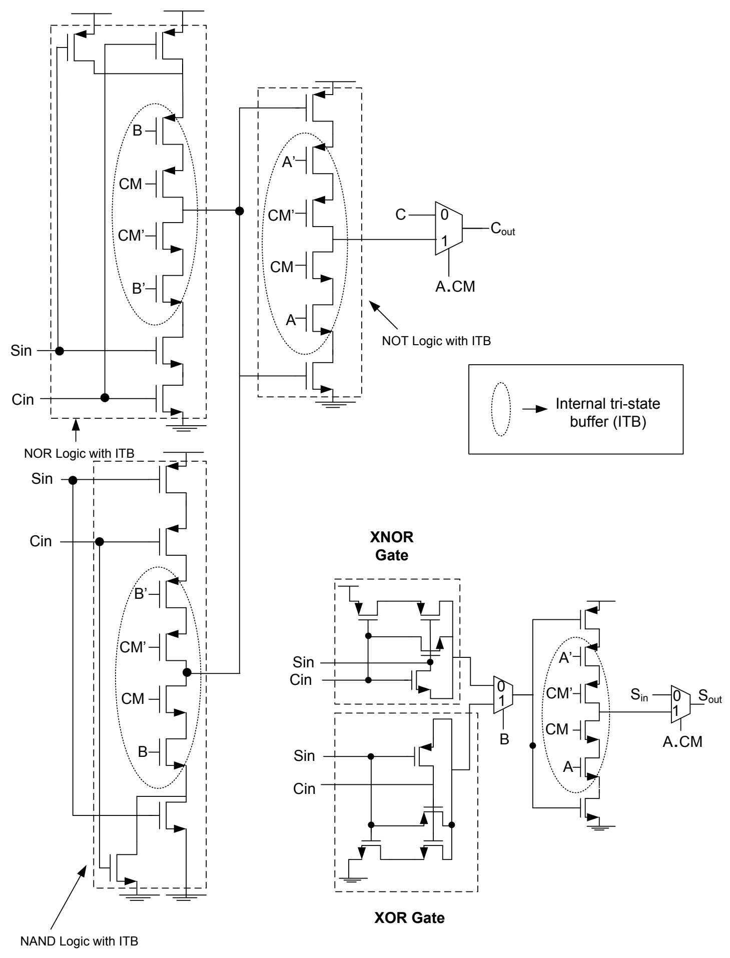


Figure 3.7: Logic schematic of RRBC

Fischer [9] adder architecture that is known to have a good trade-off between area and the performance .

The Ladner-Fischer architecture illustrated in Fig.3.8 is modified to support either a one 8-bit addition or two 4-bit additions. Since the focus in this work is reconfigurability, an AND gate is inserted in the carry propagation path and one of its inputs is connected to configuration mode $(CM)$. If the control bit $CM$ is '1', the adder will operate as a one 8-bit adder while if it is '0', the carry propagation is broken and the adder operates as two independent 4-bit adders. This adder facilitates the proposed multiplier to carry out either one 8-bit multiplication or two 4-bit multiplications.

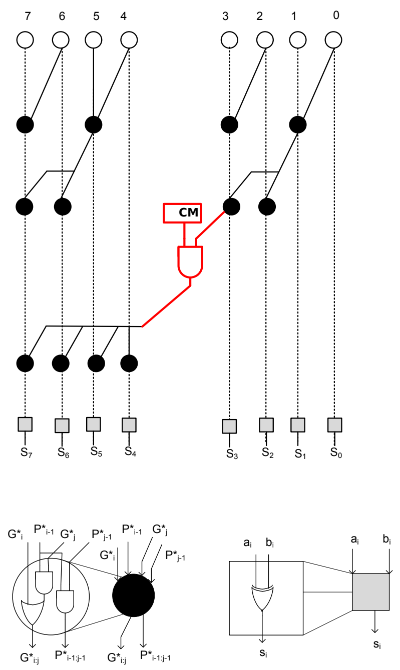


Figure 3.8: A reconfigurable 8-bit Ladner-Fisher prefix adder

## 3.3 Simulation and Synthesis

Detailed simulations of the proposed multiplier have been carried out and a comparison with similar designs existing in the literature has been made. For a fair comparison, existing row-bypass [10] and column-bypass [11] designs are independently combined with twin-precision [14] to form a multiplier. For example, twin row-bypass multiplier is obtained by incorporating row-bypass logic into twin-precision multiplier. Similarly, twin column-bypass is formed using column bypass logic with twin-precision multiplier.

All the multiplier designs with bypass schemes given in Table 3.4 including the proposed one have been described structurally using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1. These multipliers have been mapped on to TSMC 180 nm technology slow-normal library (operating conditions 1.8 V, 25°C) using cadence RTL compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating dynamic power.

Table 3.4 presents performance metrics such as area, delay, power, and power-delay product for the 8-bit reconfigurable multiplier with different bypass schemes including the proposed one. Figures 3.9-3.12 provide a graphical comparison of the same.

Table 3.4: Area, delay and power of various multipliers with various bypassing schemes

| Mode | Multiplier | Area ($\mu m^2$) | % change | Delay (ps) | % change | Power (nW) | % change | Power -Delay product ($10^6$)J | % change |
|---|---|---|---|---|---|---|---|---|---|
| CM1 | Twin precision [14] | 586 | 94% | 2094 | 134% | 13454 | 105% | 28.1 | 140% |
| | Twin Row [10] | 660 | 105% | 2013 | 127% | 13254 | 103% | 26.6 | 132% |
| | Twin Column [11] | 630 | 101% | 1960 | 123% | 13142 | 102% | 25.7 | 128% |
| | Proposed | 625 | 100% | 1560 | 100% | 12838 | 100% | 20 | 100% |
| CM0 | Twin precision [14] | 586 | 94% | 2094 | 140% | 13454 | 129% | 28.1 | 181% |
| | Twin Row [10] | 660 | 105% | 1905 | 127% | 11073 | 106% | 21 | 136% |
| | Twin Column [11] | 630 | 101% | 1844 | 123% | 10867 | 104% | 20 | 129% |
| | Proposed | 625 | 100% | 1492 | 100% | 10379 | 100% | 15.48 | 100% |

It is clear from the Table 3.4 and Fig.3.9 that the proposed multiplier in mode 1 (CM1) achieves 25 to 34% lesser delay compared to the existing designs.

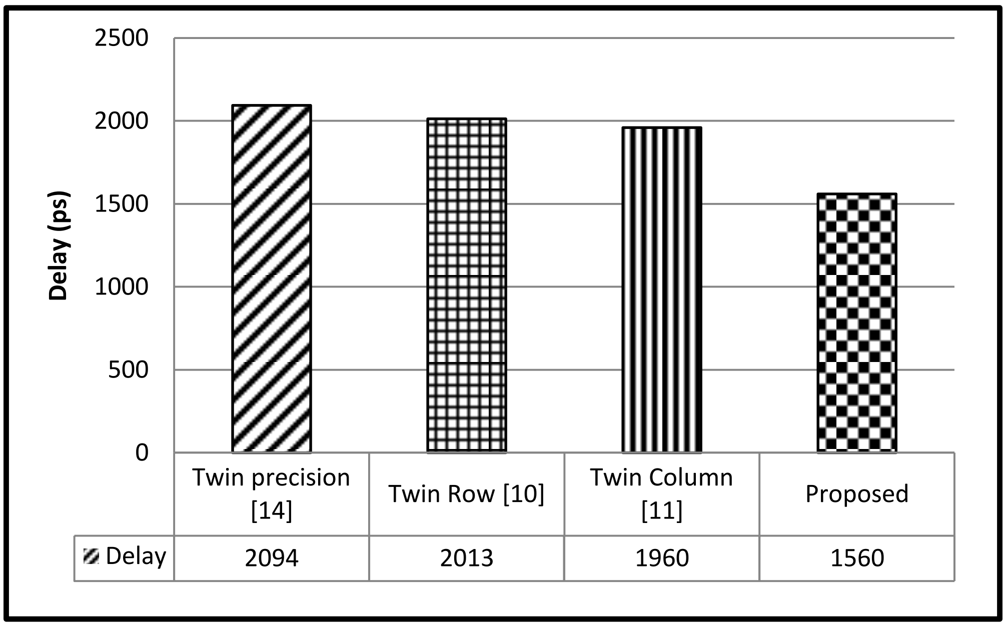
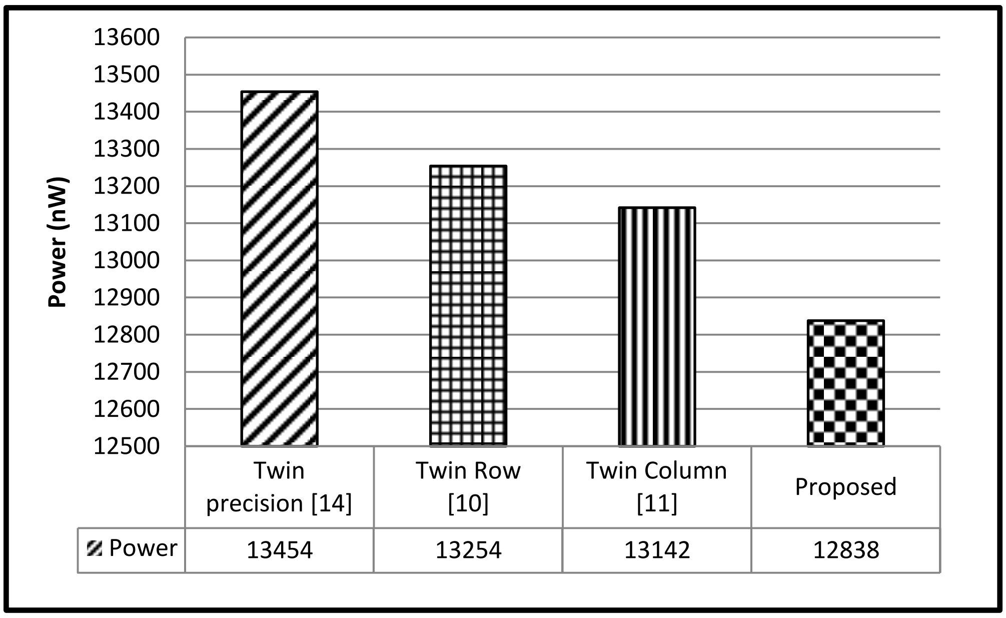


Figure 3.9: Latency of various bypass multiplier schemes

It can also be seen from the Table 3.4 (CM1) and Fig.3.10 that the proposed design consumes 2 to 5% lesser power than the existing ones. This reduction in power dissipation is due to the reduction in switching activity arising out of disabling the computational units.

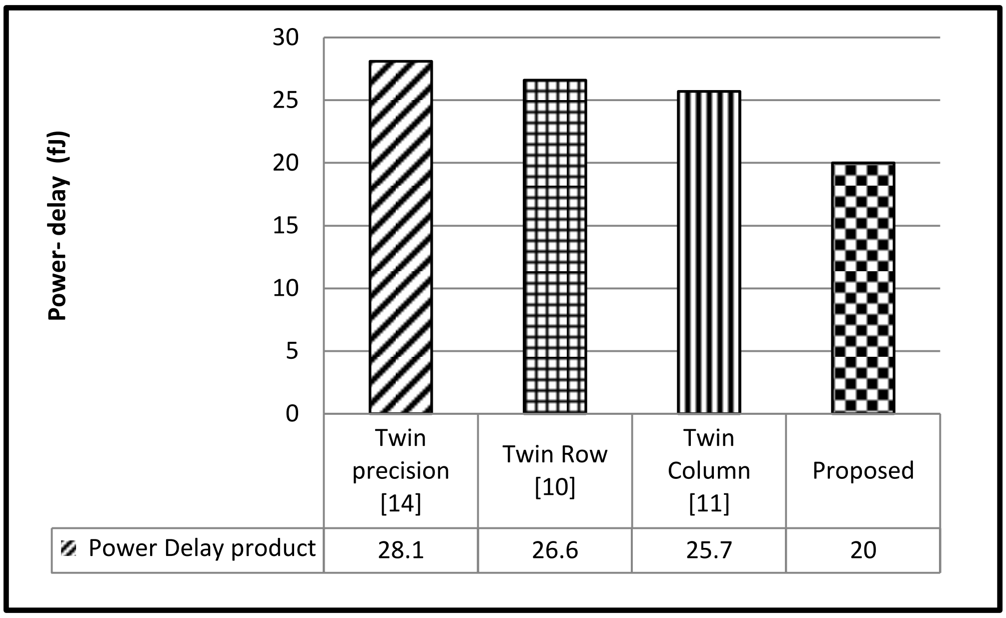
From Table 3.4 (CM1) and Fig.3.11, it can be observed that there is also an improvement of 28 to 40% in power-delay product. This is due to the saving achieved in both power and delay



| | Twin precision [14] | Twin Row [10] | Twin Column [11] | Proposed |
|---|---|---|---|---|
| ▨ Power | 13454 | 13254 | 13142 | 12838 |

Figure 3.10: Comparison of various bypass multiplier schemes in terms of power

mentioned above.Thus the proposed multiplier with a new 2-dimensional bypass scheme is found to be better in terms of power, delay and power-delay product when compared to earlier designs.



| | Twin precision [14] | Twin Row [10] | Twin Column [11] | Proposed |
|---|---|---|---|---|
| ▨ Power Delay product | 28.1 | 26.6 | 25.7 | 20 |

Figure 3.11: Power-delay product of various bypass multiplier schemes

It is evident from the Table 3.4 (CM1) and Fig.3.12 that the area overhead of row bypass scheme alone is 9% while this increase is only 6% in the proposed design even after incorporating both row and column bypassing logic. This is because of the usage of efficient adder bypass cells which take up less area and are also fast.

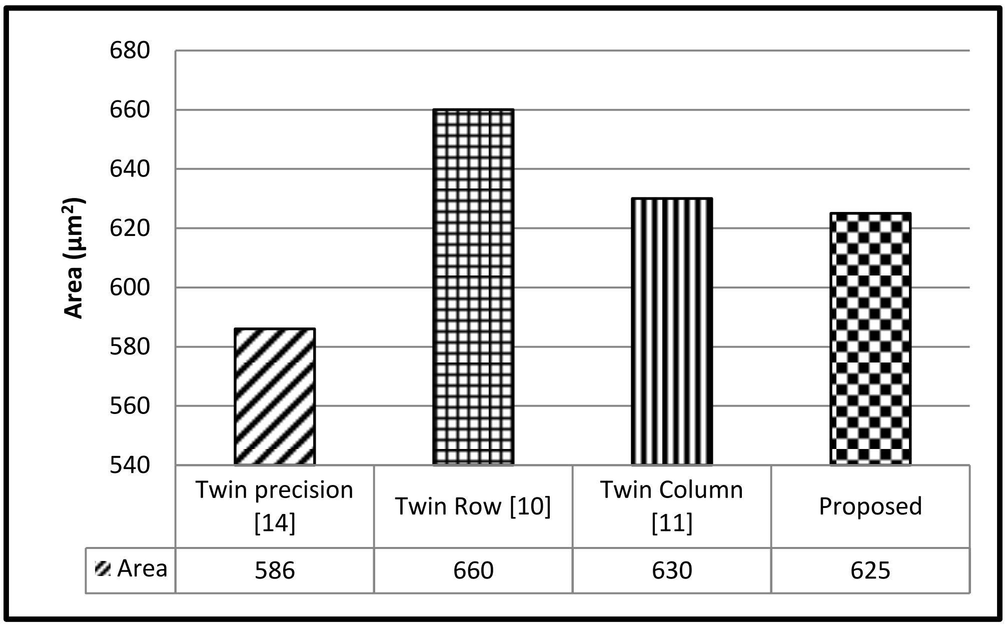


Figure 3.12: Comparison of various bypass multiplier schemes in terms of area

It can also be seen from the Table 3.4 (CM0) that the proposed design achieves 23-40% and 4-29% lesser delay and power respectively compared to the existing designs in mode 0 (CM0). Also, an improvement of 29 to 80% in power-delay product is achieved. This reduction of delay is because of the new bypass adder cells that have shorter critical path compared to the conventional bypass cells. Another reason for the improvement in delay is due to bypassing of the adder cells whenever the partial products in the PP matrix are zero. Accordingly, the more the number of zeroes are in the PP matrix, the faster will be the design. For this reason, the power improvement achieved in mode 0 is much larger compared to mode 1.

## 3.4 Conclusions

In this chapter, a reconfigurable multiplier with two dimensional bypassing scheme has been proposed. In normal operation mode, the reconfigurable multiplier performs 8-bit multiplication. For applications where accuracy can be relaxed, the multiplier can perform 4-bit multiplication while expending only a fraction of the energy of a conventional 8-bit array multiplier. Also, when performing two 4-bit parallel multiplications within an 8-bit multiplier only one half of the logic is used.

Further, to reduce the dynamic power, a new 2-dimensional bypassing technique is incorporated into the multiplier architecture. The bypass technique uses selective disabling of computation cells when the column and/or row partial products are zero. This is achieved by incorporating the new bypassing computational cells ( RTDBC and RMRBC) that improve the power efficiency and also facilitate reconfigurability of the multiplier.

The proposed multiplier with the new 2-dimensional bypass scheme performs better than other designs in terms of delay (a reduction of 34%) and power (a reduction of 5% ) resulting in a overall reduction of 40% in power-delay product. The delay advantage in the proposed design is due to the RTDBC and RMRBC cells that are simple and take up less logic unlike in the existing designs.

# Chapter 4

# An Improved Fixed-Width Recursive Binary Multiplier

## 4.1 Introduction

Most of the signal and image processing applications possess an inherent quality of error resilience and hence can tolerate error up to a certain limit in computations [56]. In such applications, savings in power are achieved by pruning the data path units [57] such as truncating a multiplier. Truncation however may lead to errors in computing and therefore it‘s always a challenge between the amount of error that can be tolerated in an application and the advantage that can be obtained in terms of it‘s implementation as reflected by the parameters such as area, latency and power. Thus, the focus of this chapter is to implement and validate a fixed-width multiplier with improved efficiency for error resilient applications.

A large variety of fixed-width (or truncated) multiplier designs has been proposed in the literature and found to be a potential solution for efficient implementation. In these designs, most of which are either array or recursive-based [17,18], the least significant bits (LSBs) of the partial product matrix are removed and an error compensation function is added in their place [21–23]. Multiplier designs based on recursive technique are faster and have the regularity of array multipliers making them the most suitable candidate for VLSI implementation [19].

In this work, a recursive binary multiplier architecture based on Karatsuba algorithm [16]

is chosen. This architecture possesses an inherent hierarchical structure that consists of several sub-multipliers making it suitable for fixed-width applications [18]. Thus, rather than modifying the entire multiplier structure, the sub-multiplier in the least significant position can be removed and replaced with a data-dependent correction term. Following this, a new correction scheme is developed in this work for fixed-width recursive multipliers. This scheme enables tuning of the accuracy through a new error compensating function achieved in a systematic manner. The proposed and the existing truncation schemes have been applied on an image sharpening algorithm and compared in the context of certain well-known image processing benchmarks such as Lena, Cameraman and Pirate for performance.

The rest of the chapter is organized as follows. The proposed fixed-width recursive multiplier architecture is described in Section 4.2. Performance analysis of both the proposed and the existing multiplier architectures is carried out and compared in Section 4.3. They are used in an image sharpening algorithm to quantify their performance, results of which are provided in Section 4.4.

## 4.2 A New Approach to Error Correction in Fixed-Width Recursive Multipliers

The proposed architecture is based on the fact that the error after removing the least significant sub-multiplier $A_LB_L$ (which contributes minimum to the final result) is always positive and hence the multiplication product obtained is less than the exact value. Error due to this approximation is further reduced by using a data-dependent correction function that is based on the least significant partial product columns within $A_LB_L$ that are not formed.

A typical recursive multiplication of two inputs $A$ and $B$, each of $2n$-bit width, results in four sub-multipliers as illustrated in Fig.4.1. The PPs generated in these sub-multipliers are reduced using the partial product reduction structure [5] as mentioned earlier.

The product ($P_{exact}$) after multiplication is given by,

$$P_{exact} = P_{H1} + P_{H2} + P_{H3} + P_L \tag{4.1}$$

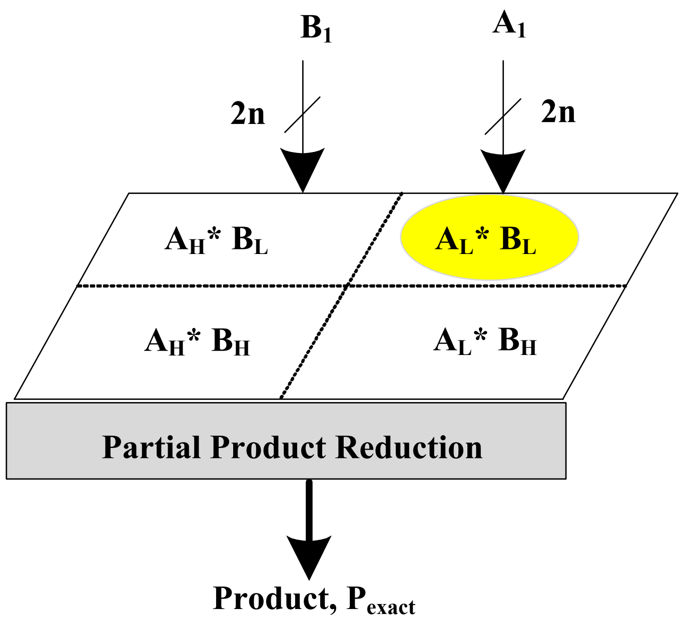


Figure 4.1: A 2n*2n recursive multiplication structure illustrating sub-multipliers

Where $P_{H1} = A_H B_H$, $P_{H2} = A_H B_L$, $P_{H3} = A_L B_H$ and $P_L = A_L B_L$

In the proposed scheme, as in the other scheme, the sub-multiplier $(A_L B_L)$ that contributes minimum to the final result is removed. Further, error due to this approximation is reduced using a data-dependent correction function $(CF_0)$ which is based on the least significant partial product columns within $A_L B_L$ that are not formed.

The expression for approximate product $(P_{approx})$ after removing $A_L B_L$ and subsequent correction is as follows:

$$P_{approx} = P_{H1} + P_{H2} + P_{H3} + CF_0 \tag{4.2}$$

Since the correction function $(CF_0)$ does not include all the PP columns of $A_L B_L$, it leads to an error. Further, the error magnitude and the hardware complexity depend on the function selected.

The error due to replacement of $A_L B_L$ using the correction function is determined as,

$$Error, e_0 = A_L B_L - CF_0 \tag{4.3}$$

Where

$$A_L B_L = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} a_i b_j 2^{i+j} \tag{4.4}$$

The operation of the proposed approach is demonstrated on a 8*8 recursive multiplier and error analysis carried out. Figure.4.2 illustrates four sub-multipliers and their corresponding partial products in a 8*8 recursive multiplier. As mentioned earlier, the sub-multiplier $A_L B_L$ is removed and replaced with the correction function $(CF_0)$.

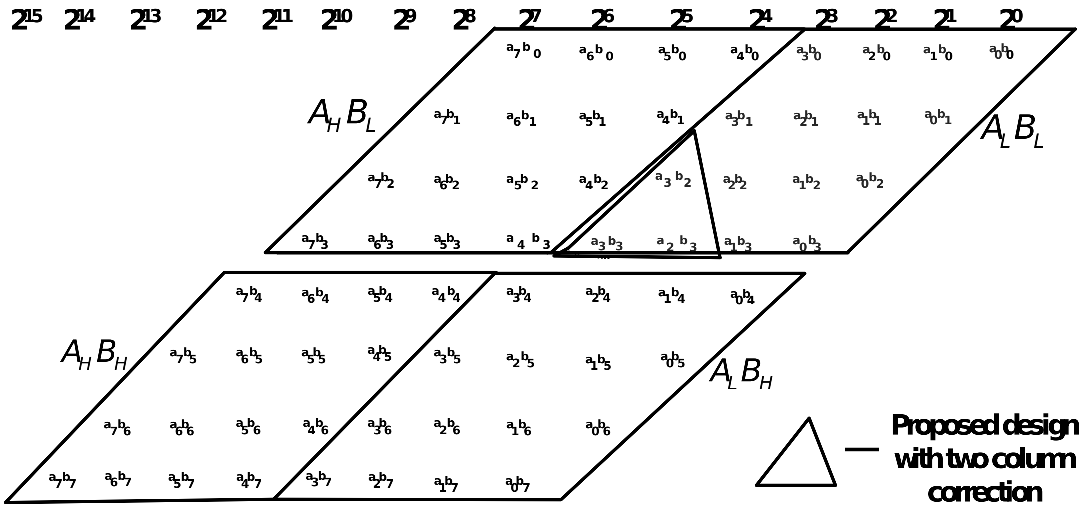


Figure 4.2: Partial product matrix of a fixed-width recursive multiplier with two most significant columns considered for correction

The proposed approach considers the most significant partial product columns of $A_L B_L$ for the correction function. Based on the number of columns considered, the accuracy and hardware requirements vary. The variation of average error with number of columns (considered for correction) is carried out on a 8*8 recursive binary multiplier mentioned above, as illustrated in Fig.4.3. The plot shows the variation of average error as a function of the number of most significant partial product columns of $A_L B_L$. It can be observed from the figure that the average error reduces with the increase in number of partial product columns, though with an area overhead. Further, it can be noted that the average error drops very sharply with only two most significant columns considered. Beyond this point, even with the inclusion of more partial product columns the rate at which the average error decreases is less.

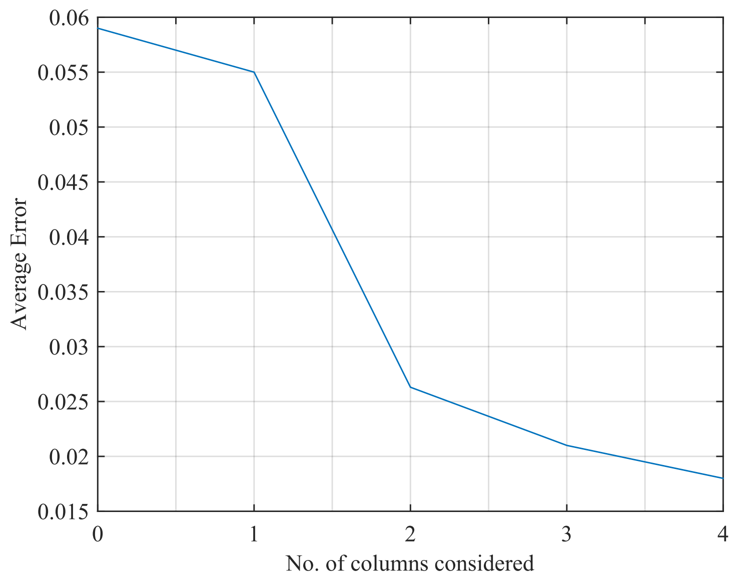


Figure 4.3: Average error Vs Number of most significant partial products columns considered for correction in a 8 * 8 fixed-width recursive multiplier

### 4.2.1 Error Analysis for the proposed correction function

For the sake of understanding, error analysis when two most significant columns of $A_LB_L$ (highlighted in a triangle in Fig.4.2) are considered for correction, is given below. The advantage of the proposed method is that the partial product columns selected for correction have higher weight when compared to those in the existing schemes. The correction function proposed in this work thus tends to achieve better accuracy. The same is proved in Section 4.3.

A step-by-step implementation of unsigned multiplication using existing and the proposed correction (where two most significant partial product columns are considered) schemes is given in the numerical example 4.1 below .

**Example 4.1.**

$A = (25)_{10} = (00011001)_2$

$B = (20)_{10} = (00010100)_2$

Actual Product, $P_{exact} = A * B = (500)_{10} = (0000000111110100)_2$

Product using recursive multiplier with average value correction terms, Scheme 2 [18] :

$P_{approx2} = (475)_{10} = (0000000111011011)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx2})}{P_{exact}}$ = $\frac{(500-475)}{500} * 100 = 5$

Product using recursive multiplier with 1-bit correction, Scheme 3 [18] :

$P_{approx1} = (464)_{10} = (0000000111010000)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx1})}{P_{exact}} = \frac{(500-464)}{500} * 100 = 7.2$

Product using recursive multiplier with proposed correction (two most significant columns considered for correction) scheme :

$P_{approx} = (0000000111110000)_2 = (496)_{10}$

Average error (%) = $\frac{(P_{actual} - P_{approx})}{P_{exact2}} = \frac{(500-496)}{500} * 100 = 0.8$

It can be observed from the above analysis that the proposed correction with two most significant columns considered for correction achieves better accuracy when compared to the existing schemes.

#### 4.2.1.1 Hardware Implementation of the Proposed Fixed-Width Recursive Multiplier

The hardware block diagram of a $8*8$ fixed-width recursive multiplier is illustrated in Fig.4.4. The sub-multipliers $A_H B_H, A_L B_H, A_H B_L$ are implemented using 4*4 binary array multipliers [3] while the compensation function essentially includes the two most significant partial product columns highlighted in triangle in Fig.4.2. The PPs corresponding to sub-multipliers along with the correction function are reduced using a tree carry save adders and final adder. A detailed PP reduction mechanism is explained below.

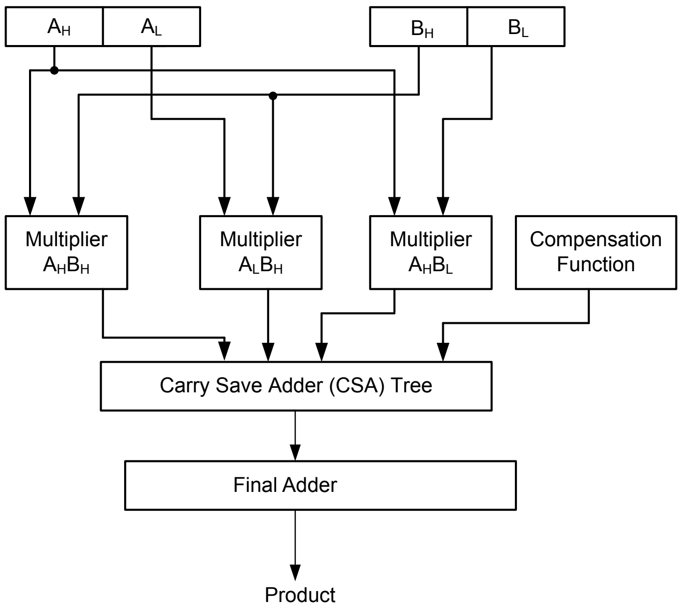


Figure 4.4: A new fixed-width recursive multiplier hardware

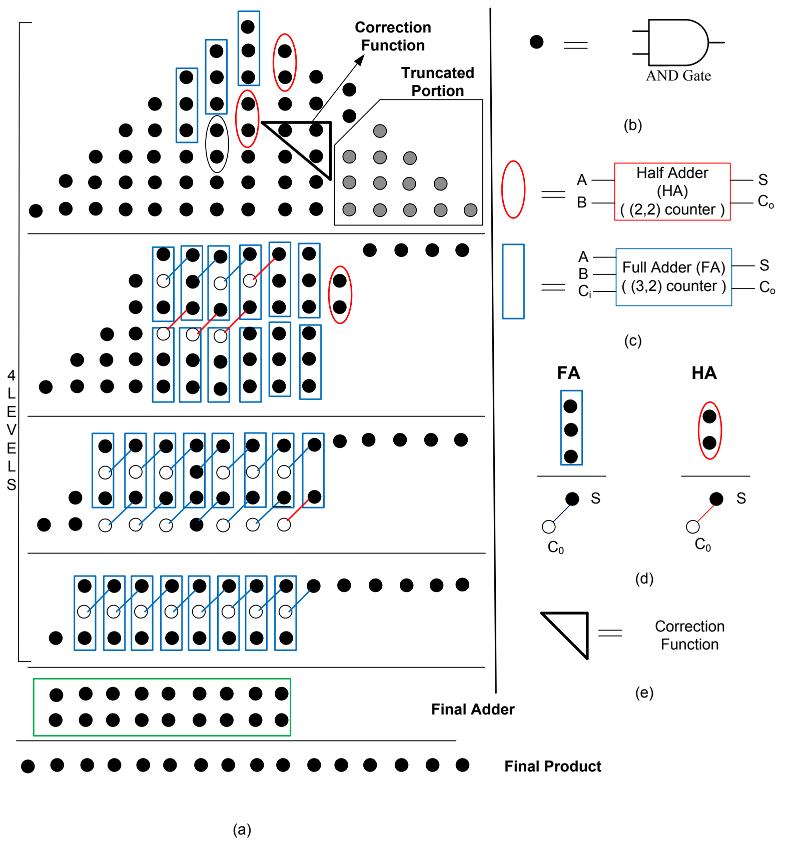


Figure 4.5: (a) Partial product reduction structure of a fixed-width recursive multiplier with two most significant columns considered for correction (b) Partial product computed using an AND gate (c) Representation of 3:2 and 2:2 counters (d) Computation of Sum and Carry-out using dot notation in a full and half adder circuits (e) Notation for correction function with two most significant columns considered

Figure 4.5 illustrates the reduction of PPs generated from individual sub-multipliers and compensation function using a CSA tree structure. The partial products shown in grey color corresponding to $A_L B_L$ are discarded and a correction function highlighted in a triangle is added. As mentioned earlier, the CSA structure is formed using full adder (3:2 counter) and half adder (2:2 counter) circuits as shown in Fig.4.5(c). The sum output $(S)$, from a full or half-adder at one stage places a dot in the same column at the next stage. A carry output $(C_o)$ from a full or half-adder at one stage places a dot one order of magnitude higher i.e., in the column to its left, at the next stage. As shown in Fig.4.5(d), three dots joined by a solid diagonal line indicates that these PPs are outputs of (3,2) counter while two dots joined by diagonal line indicates that these PPs are outputs of (2,2) counter. Consequently, the PP matrix is accumulated to a height of two in four levels using a carry save adder (CSA) tree structure, formed using full adder and half adder as shown in Fig.4.5(a), that are eventually reduced to a product using a final ripple carry adder [6] .

## 4.3 Results

In order to compare the proposed fixed-width multiplication scheme with the existing ones, error analysis has been carried out using MATLAB to check the improvement in precision. This is followed by unit gate level modeling and synthesis based analysis to understand the hardware savings achieved.

### 4.3.1 Error Analysis

MATLAB program has been used to simulate various multiplier architectures including the proposed one. For the simulation purpose, we randomly selected 10,000 inputs from all possible input patterns (i.e., 0–65 535). Error analysis has been carried out to compute the average and maximum error in the proposed and the existing schemes [18, 21–23]. An example of calculating the average error has already been illustrated in example 4.1 given earlier. A comparison of the errors is provided in Table 4.1. It may be noted that the proposed multiplier with two most significant columns is taken as the reference (ratio of ‘1’). For example, constant correction [23] scheme has 15.2X error in comparison with the reference design. Also, lesser ratio implies better precision.

For a fair comparison, the following have been taken into consideration. The existing schemes have correction function which is fixed and hence the maximum and average error shown in Table 4.1 do not mention the number of most significant columns considered for correction unlike the proposed multiplier.

The fixed-width recursive multiplier schemes [18] including the proposed one perform better in terms of precision compared to the fixed-width array multipliers [21, 22]. This is due to the fact that in all the recursive architectures the sub-multiplier $A_LB_L$ that contributes marginally to final result is discarded.

It can be seen that in the proposed scheme when more number of columns is considered, the average error reduces and hence the accuracy of multiplier improves. Results in Table 4.1

Table 4.1: An error comparison of various multiplier architectures

| Multiplier scheme | Most significant columns considered | Average error | Ratio | Maximum error | Ratio |
|---|---|---|---|---|---|
| Constant correction [23] | - | 0.4 | 15.2 | 4 | 4.77 |
| Constant correction [21] | - | 0.06 | 2.28 | 3 | 3.58 |
| Variable correction [22] | - | 0.06 | 2.28 | 1.4 | 1.67 |
| Direct truncated recursive multiplier [58] (Without correction) | - | 0.059 | 2.1 | 1 | 1.19 |
| Fixed-width recursive multiplier with average value correction (Scheme 2) [18] | - | 0.037 | 1.4 | 0.875 | 1.04 |
| Fixed-width recursive multiplier with 1-bit correction (Scheme 3) [18] | - | 0.055 | 2.24 | 1.25 | 1.49 |
| Proposed recursive fixed-width recursive multiplier | 1 | 0.055 | 2.24 | 1.25 | 1.49 |
| | 2 | 0.0263 | 1 | 0.837 | 1 |
| | 3 | 0.0229 | 0.87 | 0.810 | 0.96 |
| | 4 | 0.0189 | 0.71 | 0.782 | 0.93 |

show that the new recursive scheme with two most significant columns as correction has an improvement in average error of 1.4 to 15.2X compared to the existing designs. Similarly, an improvement of 1.04 to 4.77X in maximum error is achieved. It can also be seen that an improvement of 1.5 to 15.3X and 1.7 to 15.5X in average error is achieved, if column sizes of ‘3’ and ‘4’ are included for correction respectively. It can be observed that significant reduction in error is obtained when only two most significant columns are considered for correction. Although, there is an increase in the accuracy when the number of columns considered is more than two, it is only nominal which comes at the cost however of increased hardware, as shown in the next subsection.

### 4.3.2 Area and Delay Comparison of Various Multipliers using Unit Gate Analysis

Different correction schemes of various multipliers have been analyzed using unit gate modeling as this approach provides a decent model for computing the real cost of each component. Further, it does not depend on any synthesis tool [59]. Design metrics such as area (A) and delay (D), have been considered and compared for all the designs. Assumptions made while calculating the area of components are shown in Table 4.2.

Table 4.2: Assumptions made for unit gate modeling

| Gates | Count |
|---|---|
| 2- INPUT AND, OR, NAND, NOR | 1 |
| M- INPUT AND, OR, NAND, NOR | M-1 |
| 2- INPUT XOR, XNOR, MUX | 2 |

Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for area [59]. Moreover, an m-input gate is assumed to be composed of a tree of 2-input gates and the effects of wiring, buffering and inverting costs are neglected.

Table 4.3: Unit gate modeling analysis of various multiplier architectures

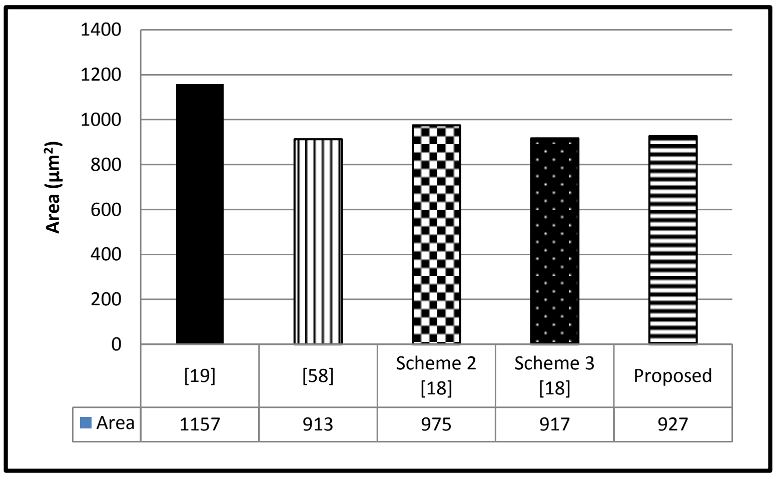
| Multiplier scheme | Most significant columns considered | Area | Percentage |
|---|---|---|---|
| Accurate recursive multiplier [19] | - | 470 | 124% |
| Direct truncated recursive multiplier (Without correction) [58] | - | 364 | 96% |
| Fixed-width recursive with average value correction (Scheme 2) [18] | - | 396 | 104% |
| Fixed-width recursive with 1-bit correction (Scheme 3) [18] | - | 367 | 97% |
| Proposed recursive fixed-width recursive multiplier | 1 | 367 | 97% |
| | 2 | 379 | 100% |
| | 3 | 401 | 106% |
| | 4 | 421 | 111% |

The results of unit gate modeling of various 8-bit multipliers, including the proposed multiplier, (with various number of columns considered) have been compared and given in Table

4.3. The proposed multiplier with two most significant columns as correction is taken as the reference. It can be seen that the hardware of the proposed design increases with the number of columns considered for correction. It may be observed that there is an area overhead of 4% of the proposed multiplier compared to the best performing recursive multiplier when only two columns are considered for correction while it increases to 10% and 15% for column sizes of three and four respectively. Thus, taking into account the precision improvement and hardware required, it can be concluded that including two most significant columns for correction results in a good trade-off between the precision and area of the hardware.

### 4.3.3 Hardware Synthesis Results

For a fair comparison, all the multiplier designs of 8-bit width have been modeled with Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1 and mapped on to TSMC 180nm technology, slow-normal library using cadence RTL compiler v7.1. Hardware synthesis has been carried out to compare important implementation metrics such as area, delay and power.



| | [19] | [58] | Scheme 2 [18] | Scheme 3 [18] | Proposed |
|---|---|---|---|---|---|
| ■ Area | 1157 | 913 | 975 | 917 | 927 |

Figure 4.6: A Comparison of the proposed fixed-width recursive multiplier with various existing multipliers in terms of area

Table 4.4 presents performance metrics such as area, delay and power for the 8-bit multiplier with different correction schemes including the proposed one. Also included in the Table is performance in percentage of various designs in comparison with the proposed designs. Further, Figs.4.6-4.8 provide a graphical comparison of area, delay and power of various

Table 4.4: Area, delay and power of various multipliers with correction schemes

| Multiplier scheme | Area ($\mu m^2$) | Percentage (%) | Delay ($ps$) | Percentage (%) | Power ($nW$) | Percentage (%) |
|---|---|---|---|---|---|---|
| Accurate recursive multiplier [19] | 1157 | 124% | 2120 | 115% | 40834 | 114% |
| Direct truncated recursive multiplier (Without correction) [58] | 913 | 98% | 1715 | 93% | 35196 | 98% |
| Fixed-width recursive with average value correction (Scheme 2) [18] | 975 | 105% | 1952 | 106% | 36405 | 102% |
| Fixed-width recursive with 1-bit correction (Scheme 3) [18] | 917 | 99% | 1795 | 97% | 35245 | 98% |
| Proposed fixed-width recursive multiplier with two column correction scheme | 927 | 100% | 1843 | 100% | 35849 | 100% |

multipliers including the proposed one.

It is evident from Table 4.4 and Figs.4.6 & 4.7, that an improvement of up to 5% and 6% in area and delay respectively is achieved by the proposed scheme compared to scheme 2 [18] which comes closest in terms of the precision. It should however be remembered that scheme 2 has a precision that is 40% less than that of the proposed one. While other schemes may marginally perform better in terms of area and delay, they are way off in terms of the precision compared to the existing one. Thus, it can be concluded that the proposed fixed-width multiplier scheme outperforms all other existing schemes in the literature.

## 4.4 Benchmarking Various Multiplication Schemes-Application to Image processing

Image sharpening is an important image enhancement technique employed in image processing applications. The computational process of sharpening an image involves a number of fixed

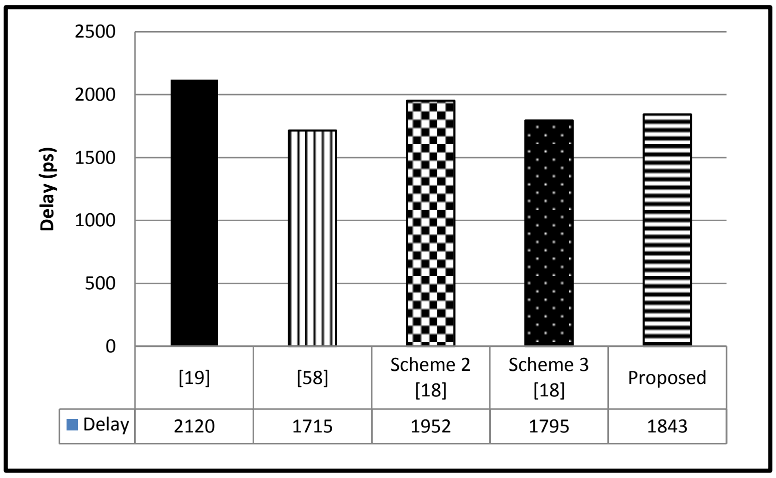


Figure 4.7: A Comparison of proposed fixed-width recursive multiplier scheme with various existing multipliers in terms of delay

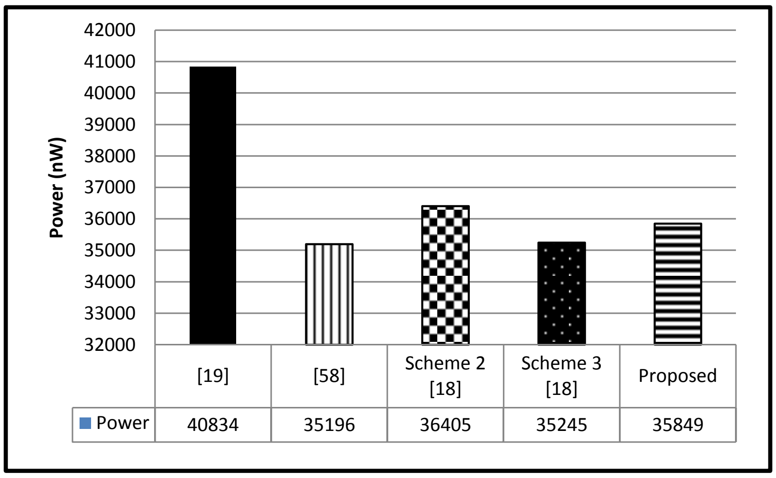


Figure 4.8: A Comparison of proposed fixed-width recursive multiplier scheme with various existing multipliers in terms of power

point multiplications. It is therefore a good application to prove the efficacy of the proposed fixed-width recursive multiplier.

### 4.4.1 Image Sharpening Algorithm

Human perception is highly sensitive to edges and fine details of an image. Since images essentially consist of high-frequency components their visual quality is corrupted if these high frequencies are removed. Conversely, increasing the high-frequency components of an image improves the image quality. Image sharpening algorithm described in [60] is one such enhancement technique which highlights the edges and fine details in an image.

This algorithm described below, accepts an image, processes it, and produces an image of high quality. Suppose $I$ is the original image, the processed image $S$ is described using the expression

$$S(x,y) = 2I(x,y) - M \tag{4.5}$$

where $M = \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} H(i+3,\, j+3)\, I(x-i, y-j)$

and $H$ is a matrix defined as

$$H = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Since this expression involves a number of multiplications, an exact multiplier such as an array multiplier can perform these operations accurately thereby producing an image of high quality. On the other hand, using an approximate multiplier would result in an image of certain quality which is quantified using established metrics such as mean square error (MSE) and peak signal to noise ratio (PSNR).

The MSE represents the loss of information in the image and is expressed as,

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,\, j) - K(i, j)]^2 \tag{4.6}$$

where $MAX_I$ represents the maximum possible pixel value of the image.

The peak signal to noise ratio (PSNR) in dB is expressed using $MSE$ as follows:

$$PSNR \text{ in dB} = 10.log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{4.7}$$

While the use of approximate multiplier may affect the image quality, it has the advantage of savings in terms of area and delay as compared to an accurate multiplier. In what follows, the performance of the existing recursive multipliers (Schemes 2 and 3) [18] and the proposed multiplier with two column correction is studied and compared with reference to the image sharpening algorithm. The algorithm is applied to blocks of 5*5 pixels on a set of standard images ( Cameraman, Lena and pirate). The exact multiplications are replaced by approximate multiplications using existing multipliers and the proposed multiplier, while addition, subtraction and division operations are carried out using accurate techniques. The metric MSE is computed by finding the mean of squares of difference in pixel values between original image and the processed image using approximate multipliers and these values are substituted in equation (4.7) to calculate PSNR values.

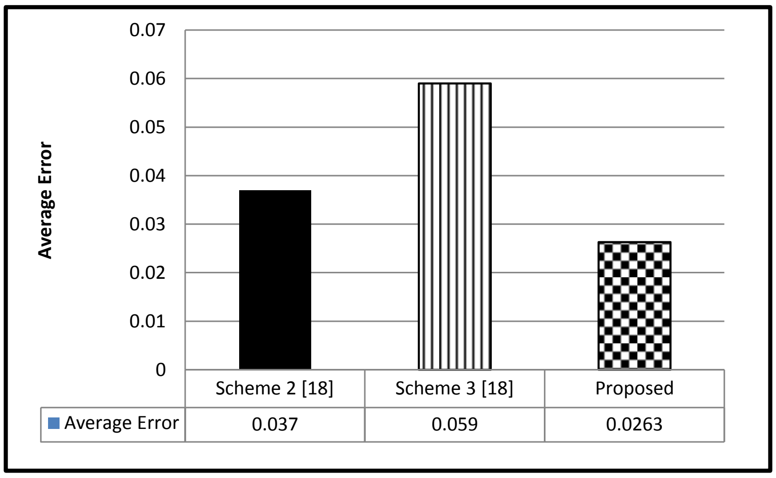
Table 4.5 provides a comparison of these metrics on a set of standard images ( Cameraman, Lena and pirate). As is well known, the quality of image is decided by the magnitude of MSE and PSNR values. It is evident from Table 4.5 that images processed with the proposed multiplier have better MSE and PSNR compared to the images processed using the schemes mentioned in [18].

Table 4.5: A comparison of values of MSE and PSNR for benchmark images using various multiplier schemes

| Image | Metric | Scheme 3 [18] | Scheme 2 [18] | Proposed |
|---|---|---|---|---|
| Cameraman | PSNR (dB) | 43.2 | 43.6 | 44.1 |
| Lena | | 39.2 | 39.3 | 39.7 |
| Pirate | | 42.7 | 43.2 | 43.8 |
| Cameraman | MSE | 3.1 | 2.8 | 2.5 |
| Lena | | 7.8 | 7.5 | 7 |
| Pirate | | 3.5 | 3.1 | 2.7 |

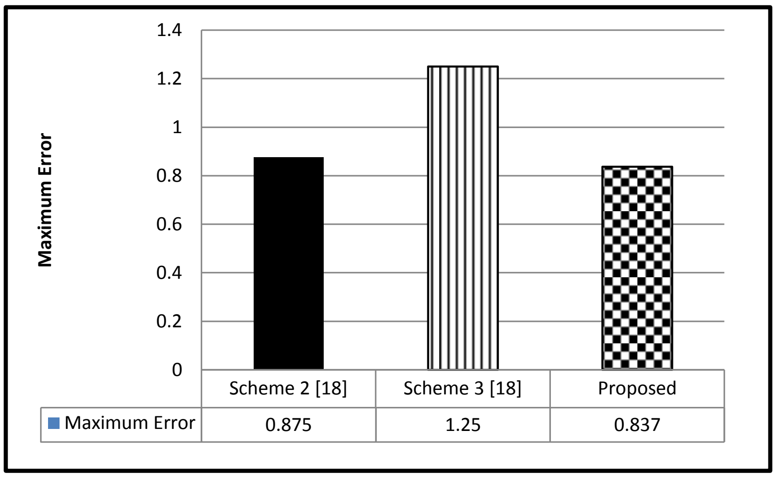
The results are only expected since the proposed scheme has less average and maximum error compared to schemes mentioned in [18] as illustrated in Figs.4.9 & 4.10.

Further, the performance of the proposed multiplier, in terms of both PSNR and MSE, may be understood by observing the images processed by it. Figure.4.11 illustrates the images that are processed using exact (array) and the proposed multiplier. It can be observed that the images of Cameraman, Lena and Pirate processed with the proposed multiplier look very similar to the original ones. Thus, it can be concluded that the proposed fixed-width recursive multiplier has a better performance compared to all other similar and existing multipliers.



| | Scheme 2 [18] | Scheme 3 [18] | Proposed |
|---|---|---|---|
| ■ Average Error | 0.037 | 0.059 | 0.0263 |

Figure 4.9: Average errors of various multipliers



| | Scheme 2 [18] | Scheme 3 [18] | Proposed |
|---|---|---|---|
| ■ Maximum Error | 0.875 | 1.25 | 0.837 |

Figure 4.10: Maximum errors of various multipliers

## 4.5 Conclusions

In this chapter, a reconfigurable multiplier design with a new truncation technique targeted for error resilience applications has been proposed. In this scheme, the sub-multiplier is replaced $A_LB_L$ with a data-dependent correction term. The heuristic correction function retains the portion of partial product matrix that is more relevant and thus helps to achieve higher accuracy compared to the earlier works. Also, the hardware overhead due to correction scheme is nominal.

Exhaustive analysis was carried out to compute the average and maximum error bound in the proposed and existing recursive multiplier schemes. Results show 15.2X times and 3.7X times improvement in average error and maximum error respectively. The error analysis

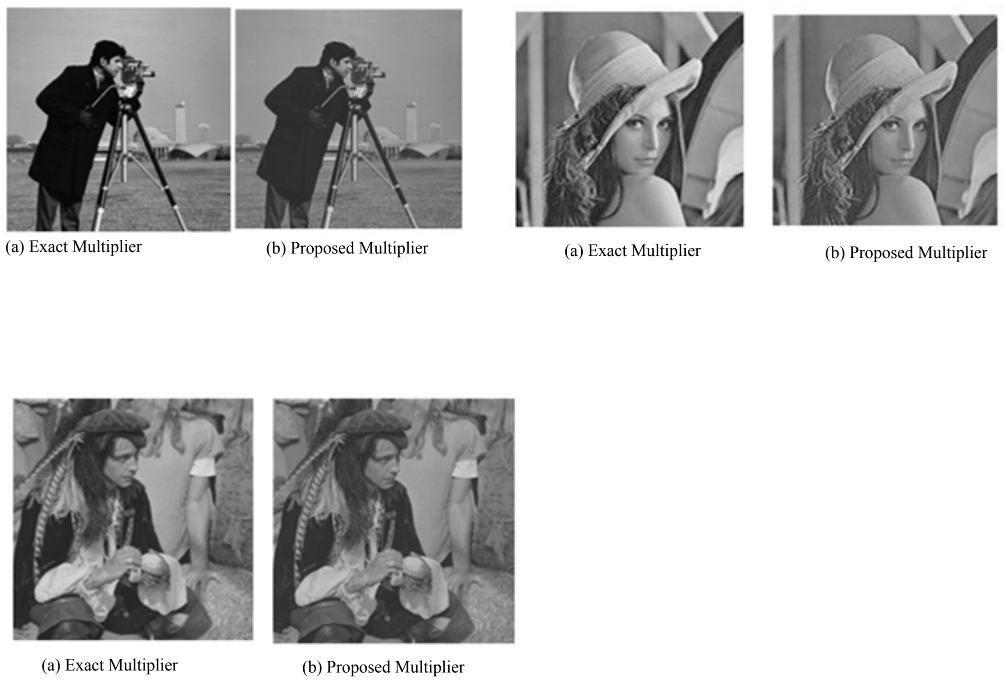


Figure 4.11: Cameraman, Lena and pirate Images obtained using exact and the proposed multiplier

show that the proposed multiplier scheme represent, in most cases, a better trade-off between accuracy and complexity.

Further, the proposed architecture efficiently performs N-bit fixed-width multiplications. For applications with high demands on precision, the multiplier is capable of performing two independent N/2-bit full precision multiplications in parallel.

# Chapter 5

# An Iterative Logarithmic Multiplier with Improved Precision

## 5.1 Introduction

In recent years, logarithmic number system (LNS) has been increasingly used as an alternative to the binary number system as it converts multiplication to addition resulting in simplified hardware [61]. However, they suffer from inherent error and any efforts in improving their accuracy would help find their increased usage in arithmetic computations with efficient hardware. In this chapter, a novel binary logarithmic multiplier with improved precision is designed and demonstrated to perform better than the existing designs. Also, the multiplier has been synthesized and it‘s performance metrics such as area, delay and power have been shown to be better than those of the existing designs.

Throughout this chapter, the symbols ‘ ~ ’ and ‘ ’ ’ are used to denote 1‘s and 2‘s complement while the symbols ‘ & ’ and ‘ | ’ denote logic AND and logic OR operations respectively. Rest of the chapter is organized as follows: Mathematical modeling and hardware architecture of the proposed scheme are presented in Section 5.2. Detailed error analysis and hardware synthesis results of various iterative logarithmic multipliers including the proposed one are presented and compared in Section 5.3. Proposed and the existing schemes have been applied on an image sharpening algorithm to quantify their performance, results of which are provided in

Section 5.4.

## 5.2 Proposed Approach

The contributions of this work include speculation of carry which results in better precision when compared to BIM and TEC approaches. Further, savings in hardware is achieved using the fractional predictor and adapted truncation scheme proposed in this work. This section presents the mathematical modeling while the hardware implementation details are discussed in the subsequent sections.

### 5.2.1 Mathematical Analysis of the Proposed Scheme

Similar to Babic's approach, equation (2.26) is expressed when carry = 1 ($x_1 + x_2 \geq 1$) as

$$\begin{aligned} N_1 * N_2 &= 2^{k_1+k_2+1}(x_1 + x_2) + 2^{k_1+k_2}(x_1^{'} * x_2^{'}) \\ &= 2^{k_2+1} * f_1 + 2^{k_1+1} * f_2 + f_1^{'} * f_2^{'} \\ &= A^1 + f_1^{'} * f_2^{'} \end{aligned} \tag{5.1}$$

$$\text{Where approximate product, } A^1 = f_1 * 2^{k_2+1} + f_2 * 2^{k_1+1} \tag{5.2}$$

Combining the equations 2.28 and 5.1, the final product terms based on carry information can be obtained as,

$$N_1 * N_2 = A^0 + (f_1 * f_2), \quad x_1 + x_2 < 1 \tag{5.3}$$

$$N_1 * N_2 = A^1 + (f_1^{'} * f_2^{'}), \quad x_1 + x_2 \geq 1 \tag{5.4}$$

where $A^0$ and $A^1$ are the approximate product terms, $f_1 * f_2$ and $f_1^{'} * f_2^{'}$ are the correction terms that form the input to the next iteration based on carry condition. Since the carry information is available prior to the completion of an iteration, error correction can be performed concurrently in hardware.

While this approach leads to improved precision, it also results in area overhead. Thus, a new truncation scheme for logarithmic multiplication is proposed which when combined with the above approach results in area savings.

### 5.2.2 A New Approach to the Approximation of Logarithmic Multiplier

The proposed approximation of logarithmic multiplier is based on the fact that the error due to Mitchell approach is always positive [59] and hence the multiplication product obtained is less than the exact value. Further, the least significant bits of these fractional portions ($f_1 * 2^{k_2}$, $f_2 * 2^{k_1}$, $f_1 * 2^{k_2+1}$ and $f_2 * 2^{k_1+1}$) are inaccurate. Thus, rounding off and manipulating these inexact terms aid in achieving substantial hardware savings without compromising on precision. On the other hand, the term $2^{k_1+k_2}$ (leading one) contributes to integer portion which is an exact value and hence need not be approximated.

The product terms in equations 5.3 and 5.4 after truncation ($T_t$) of fractional portions are written as follows:

$$N_1 * N_2 = 2^{k_1+k_2} + T_t(f_1 * 2^{k_2}) + T_t(f_2 * 2^{k_1}) + f_1 * f_2, \quad x_1 + x_2 < 1 \tag{5.5}$$

$$N_1 * N_2 = T_t(f_1 * 2^{k_2+1}) + T_t(f_2 * 2^{k_1+1}) + f_1^{'} * f_2^{'}, \quad x_1 + x_2 \geq 1 \tag{5.6}$$

where approximate product terms without and with carry are,

$A_t^0 = 2^{k_1+k_2} + T_t(f_1 * 2^{k_2}) + T_t(f_2 * 2^{k_1})$ and

$A_t^1 = T_t(f_1 * 2^{k_2+1}) + T_t(f_2 * 2^{k_1+1})$ respectively. Here $t$ denotes the truncation width.

A typical logarithmic multiplication of two inputs $N_1$ and $N_2$, each of $n$-bit width, results in $2n$ product bits. In the proposed truncated approach however, from these $2n$ fractional bits, only $t$ bits are retained. The remaining $(2n-t)$ bits are replaced either by ‘1’ (rounding up) or ‘0‘ (rounding down), denoted as $T_{t,1}$ and $T_{t,0}$ respectively. In order to arrive at an optimal value of

$t$, error analysis of an 8*8 logarithmic multiplier is carried out for one iteration as illustrated in Fig.5.1. The plot shows the variation of average error as a function of the number of fractional bits. The horizontal line gives the error without truncation $(T_{wt})$, while the curves $T_{t,1}$ and $T_{t,0}$ provide the error due to truncation.

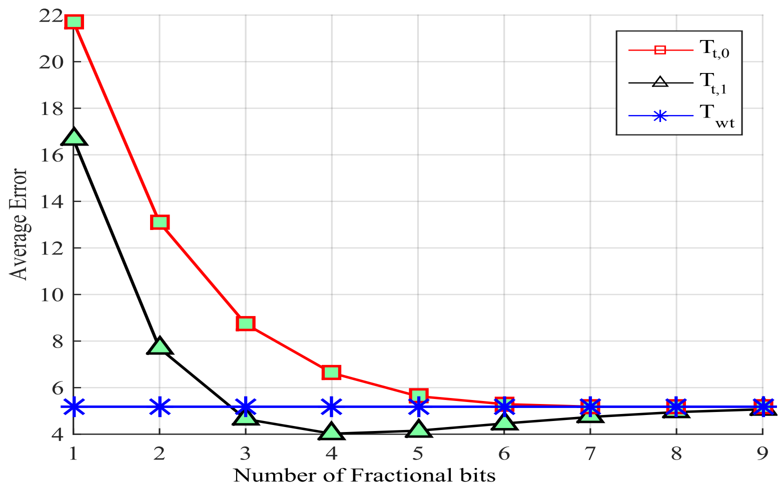


Figure 5.1: Average error Vs Number of fractional bits in 8 * 8 multiplication for one iteration

It can be seen from the figure that the average error without truncation $(T_{wt})$ (considering 2n fractional bits) remains constant and is independent of truncation width. Also, it is evident from $T_{t,0}$ plot that with the increase in truncation width$(t)$, the average error decreases rapidly and from $t = 6$ onwards, attains a value almost equal to that without truncation. Therefore, it can be inferred that the first 6 MSB fractional bits alone are contributing to the precision of the multiplication product in most cases. Similarly, from $T_{t,1}$ plot it can be inferred that for $t = 3$ error is almost same as that without truncation with precision being the best for $t = 4$. Therefore, it can be concluded that the average error obtained is minimum for $T_{4,1}$ scheme compared to all other cases.

Since the error in multiplication due to approximation should decrease with each iteration, the viability of $T_{4,1}$ truncation scheme alone for multiple iterations has been investigated. It is however found that the result obtained using $T_{4,1}$ method for first iteration is more than the exact value and further with each successive iteration there is a probability for the error to increase. Thus, utilizing $T_{4,1}$ scheme alone for multiple iterations may not be desirable. Based on the error simulations, it is observed that truncating with $T_{6,0}$ in initial iterations and $T_{4,1}$ in the final iteration results in improved error and the same is proved in the example given below.

Further, it should be noted that error plots obtained for 16* 16 and 32*32 multipliers will have characteristics similar to that in Fig.5.1.

The step-by-step procedure of the proposed truncation approach for two iterations is illustrated in example 5.1. Initially, for both 1st and 2nd iterations, truncation method $T_{6,0}$ is applied. Next, $T_{6,0}$ is applied in the 1st iteration with $T_{4,1}$ in the 2nd iteration.

**Example 5.1.** Let the two binary numbers $N_1$ and $N_2$ be

$N_1 = (00110011)_2 = (51)_{10}; N_2 = (01110111)_2 = (119)_{10}$

Exact product, $A_{exact}$ is given by $N_1 * N_2 = (6069)_{10}$

**Case 1**: $T_{6,0}$ is considered in both iterations

**Iteration 1:**

1. Initialization:

$(N_1) = (110011)_2$ ; $(N_2) = (1110111)_2$

$(f_1) = (10011)_2$ ; $(f_2) = (110111)_2$

$(k_1) = (101)_2 = (5)_{10}$ ; $(k_2) = (110)_2 = (6)_{10}$

The addition of $f_1$ and $f_2$ generates a carry satisfying the condition $x_1 + x_2 \geq 1$ and hence equation (5.6) is considered.

2. Left shift the fractional portions,

$f_1 * 2^{k_2+1} = (2432)_{10}$ ; $f_2 * 2^{k_1+1} = (3520)_{10}$

3. Truncate the fractional portion and compute the approximate product :

$T_{6,0}(f_1 * 2^{k_2+1}) = (2432)_{10}; T_{6,0}(f_2 * 2^{k_1+1}) = (3520)_{10}$

which after the 1st iteration is,

$$A^1 = (5952)_{10} \tag{5.7}$$

4. Percentage average error accumulated after 1st iteration:

$$\text{Average Error} = \left(\frac{A_{exact} - A^1}{A_{exact}}\right)$$

$$= 1.92\%$$

**Iteration 2:**

Since $x_1 + x_2 \geq 1$ in the 1st iteration,

the inputs ($N_1$ and $N_2$) to this iteration are $f_1^{'}$ and $f_2^{'}$ of 1st iteration

1. Initialization:

$(N_1) = (01101)_2$ ; $(N_2) = (001001)_2$

$(f_1) = (101)_2 = (5)_{10}$ ; $(f_2) = (001)_2 = (1)_{10}$

$(k_1) = (11)_2 = (3)_{10}$ ; $(k_2) = (11)_2 = (3)_{10}$

For the proposed design with truncation ($T_{6,0}$), condition $x_1 + x_2 < 1$ is satisfied as addition of $f_1$ and $f_2$ does not generate a carry and hence equation (5.5) is considered.

2. Left shift the fractional portions:

$f_1 * 2^{k_2} = (00101000)_2 = (40)_{10}$; $f_2 * 2^{k_1} = (1000)_2 = (8)_{10}$

3. Truncate the fractional portion and compute $\left(A^0\right)$:

$T_{6,0}\left(f_1 * 2^{k_2}\right) = (00101000)_2 = (40)_{10}$,

$T_{6,0}\left(f_2 * 2^{k_1}\right) = (1000)_2 = (8)_{10}$; $2^{k_1+k_2} = (64)_{10}$

Approximate product after the 2nd iteration is,

$A^0 = (112)_{10}$

4. Percentage average error accumulated after two iterations:

$A_{total} = A^1 + A^0 = (6064)_{10}$

$\text{Average Error} = \left(\frac{A_{exact} - A_{total}}{P_{exact}}\right)$

$\text{Average Error} = 0.082\%$

**Case 2:**

The product in the first iteration is computed using the $T_{6,0}$ scheme while in the second iteration $T_{4,1}$ scheme is adopted

**Iteration 1 :**

From equation (5.7) we obtain the approximate product in the 1st iteration as,

$\text{Approximate Product}\left(A^1\right) = (5952)_{10}$

**Iteration 2 :**

$T_{4,1}(f_1 * 2^{k_2}) = (00101011)_2 = (43)_{10},$

$T_{4,1}(f_2 * 2^{k_1}) = (1000)_2 = (8)_{10};\ 2^{k_1+k_2} = (64)_{10}$

$A^0 = (115)_{10}$

Combining the products obtained in the first and second iteration with $T_{6,0}$ and $T_{4,1}$ schemes respectively, we get the product:

$A_{total} = A^1 + A^0 = (6067)_{10}$

The corresponding average error is given by,

$\text{Average Error} = 0.032\%$

Thus, it is observed that lower error is achieved if $T_{6,0}$ is used for initial iterations followed by $T_{4,1}$ in the final iteration.

### 5.2.3 Truncated Iterative Multiplier (TIM) Hardware Implementation

The iterative multiplier scheme illustrated in Fig.5.2 is essentially an implementation of equations 5.5 and 5.6. It comprises of truncated basic logarithm blocks (TBLBs), decoder logic, adder blocks and mask logic. Based on the binary numbers ($N_1$ and $N_2$), the TBLBs generate the characteristics and truncated fractional portions. The addition of characteristics $k_1$ and $k_2$ obtained from the respective TBLBs is accomplished using the Adder 2 block. Likewise, the sum of inexact fractional portions $(f_1 * 2^{k_2})_T$ and $(f_2 * 2^{k_1})_T$ obtained from the respective TBLBs is found using the Adder 1 module. Based on carry information, the control signal (m_fp) is activated by the modified fractional predictor (MFP). The truncated BLB and MFP are presented later in Sections. 5.2.3.1 & 5.2.3.2.

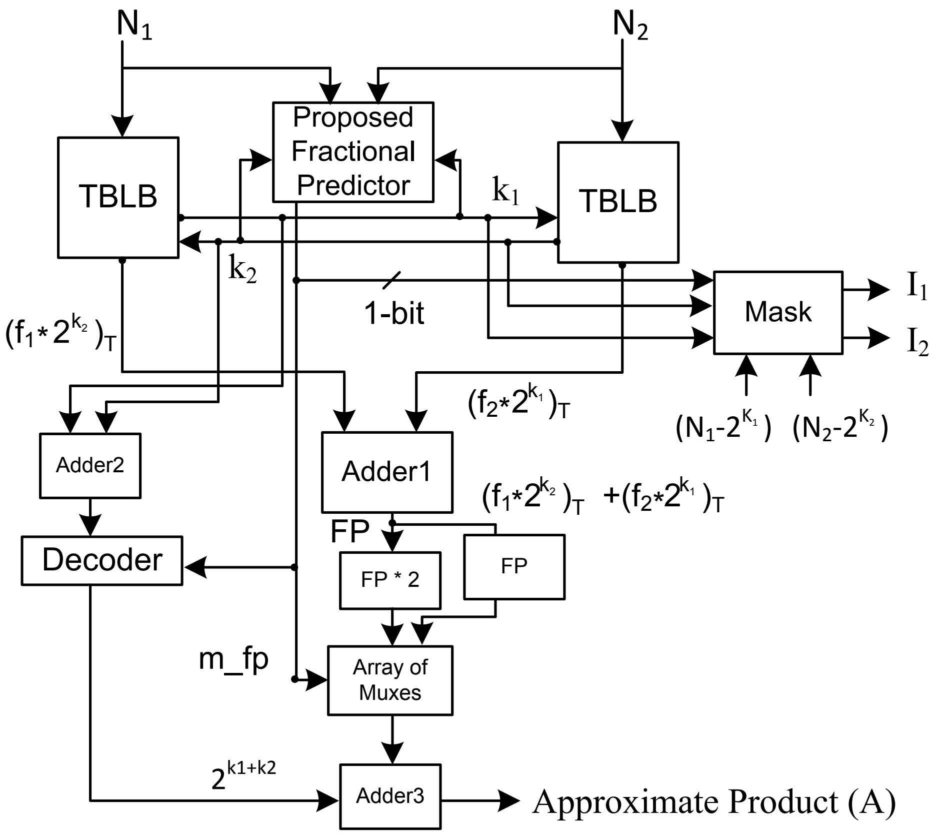


Figure 5.2: Block diagram of the proposed truncated iterative multiplier (TIM)

When the control signal m_fp is '0', meaning no carry generation, the leading one $\left(2^{k_1+k_2}\right)$ obtained from the Decoder and the truncated fractional portions are added using Adder 3 to form the approximate product $(A_t^0)$ according to equation (5.5). On the other hand, if m_fp is '1', the Decoder is disabled and the fractional portions are left-shifted by 1-bit as per equation

(5.6) to form the approximate product $\left(A_t^1\right)$. The MUX structure based on the carry signal facilitates the selection of appropriate truncated fractional portions.

#### 5.2.3.1 Truncated Basic Logarithmic Block (TBLB)

The Truncated BLB (TBLB) shown in Fig.5.3 used in this work is similar to the BLB presented in [26] except for the truncated logarithmic shifter.

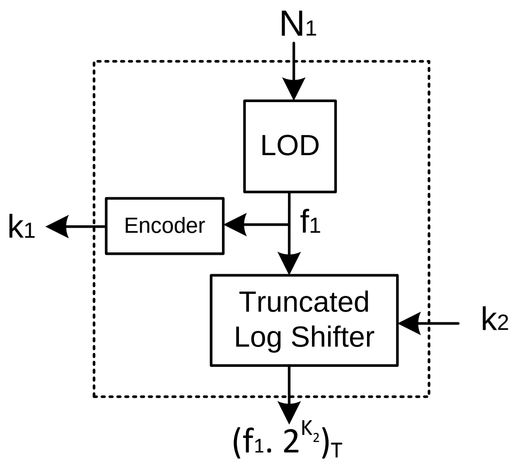


Figure 5.3: Block diagram of the truncated basic logarithmic block (TBLB)

The fact that the fractional portion $\left(f * 2^k\right)$ computed by the shifter is always a positive approximate term provides the possibility for truncating it without a significant loss of precision. Based on this idea, a method proposed in [59] implements a truncated logarithmic shifter which is used in this work. The operation of the shifter is such that it retains only those most significant bits as dictated by the truncation width. The truncated LSB section is manipulated with either ‘1’ or ‘0’ that depends on rounding up or down.

#### 5.2.3.2 Modified Fractional Predictor (MFP)

The TEC scheme [27] involves prohibitively large hardware circuitry in the form of fractional predictor, shared logic and multi-operand adder to speculate the carry out of the fractional portion. In order to overcome this, a simple and flexible carry prediction logic namely, modified fraction predictor (MFP) is proposed in this work. The output of MFP logic is 1-bit (‘0’ or ‘1’), irrespective of the number of input bits unlike in TEC approach.

In this work, MFP is preceded by a number that refers to the number of fractional bits considered for prediction. For instance, if number of fractional bits is one then it is termed as 1-bit MFP and so on. A 1-bit MFP is explained in detail using Example 5.2.

**Example 5.2.** Suppose two input binary numbers $N_1$ and $N_2$ are given by,

$N_1 = (00110011)_2 = (51)_{10}$; $N_2 = (01110111)_2 = (119)_{10}$

The characteristics ($k_1$ and $k_2$) and their 1‘s complement ($S_1$ and $S_2$) are :

$k_1 = (101)_2 \; = (5)_{10}$ ; $k_2 = (110)_2 \; = (6)_{10}$

$S_1 = \sim k_1 = (010)_2$ ; $S_2 = \sim k_2 = (001)_2$

The leading one of these two numbers is found by shifting $N_1$ and $N_2$ by an amount of $S_1$ and $S_2$ respectively resulting in $A$ and $B$ as depicted in Fig.5.4. .

$A = (11001100)_2$ ; $B = (11101110)_2$

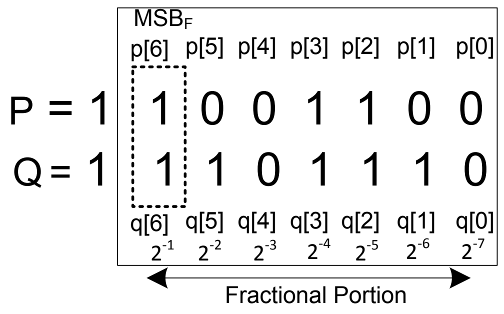


Figure 5.4: Proposed method to detect leading one in fractional portion

The $MSB_F$ in A and B has a weight of $2^{-1}$ or 0.5 and carry will be generated to next stage when $a[6]$ and $b[6]$ is ‘1’. Conversely, if either $a[6]$ or $b[6]$ is logic ‘1’, then generation of carry depends on the LSB portion of $A[5:0]$ and $B[5:0]$. Thus, carry detection is performed without waiting for actual addition to happen leading to implementation of error correction parallely.

Similarly, the logic for 2-bit and 3-bit MFP is deduced. The carry prediction logic corresponding to 1-bit, 2-bit and 3-bit MFP are deduced as specified in equations (5.8-5.10) as

$$sel1 = a[6] \, \& b[6] \tag{5.8}$$

$$sel2 = sel1 \,|\, ((a[6]|b[6])\&(a[5]\&b[5])) \tag{5.9}$$

$$sel3 = sel2 \,|\, sel \tag{5.10}$$

Where $sel = ((a[6]|b[6])\&(a[5]|b[5])\&(a[4]\&b[4]))$

To analyze the variation of average error with the number of fractional input bits, a graph is plotted for one iteration as illustrated in Fig.5.5. It is evident from the graph that the error is minimum for bit size of 4 but thereafter remains almost constant. Hence, MFP with four input fractional bits is sufficient for accurate carry prediction.

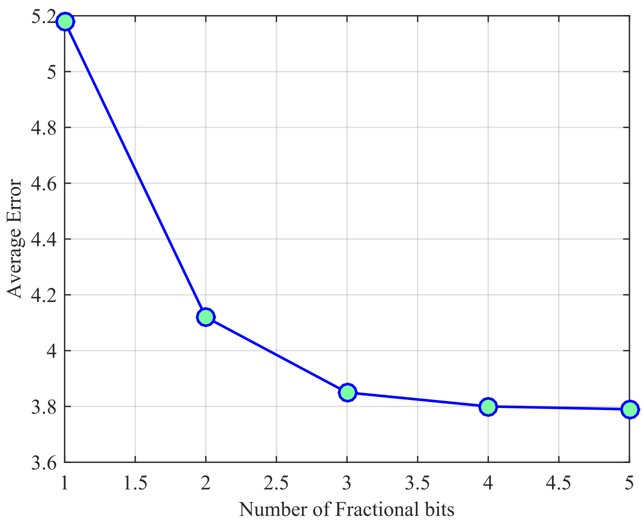


Figure 5.5: Variation of average error based on number of fractional bits

### 5.2.4 TIM Hardware for two Iterations

Figure.5.6 illustrates the cascaded TIM multipliers. While the first one computes the approximate product (A), the second one calculates the correction term. The input to the second TIM is determined by the mask circuit depending on m_fp. If m_fp is ‘1’, the inputs to next iteration $I_1$ and $I_2$ are $f_1^{'}$ and $f_2^{'}$ otherwise $f_1$ and $f_2$ respectively as per equations 5.5 and 5.6. The second TIM computes the correction term (C) based on $I_1$ and $I_2$. Similarly, in case of three iterations there will be an approximate product (A) and two correction terms. Evidently, with each successive iteration the error gets reduced.

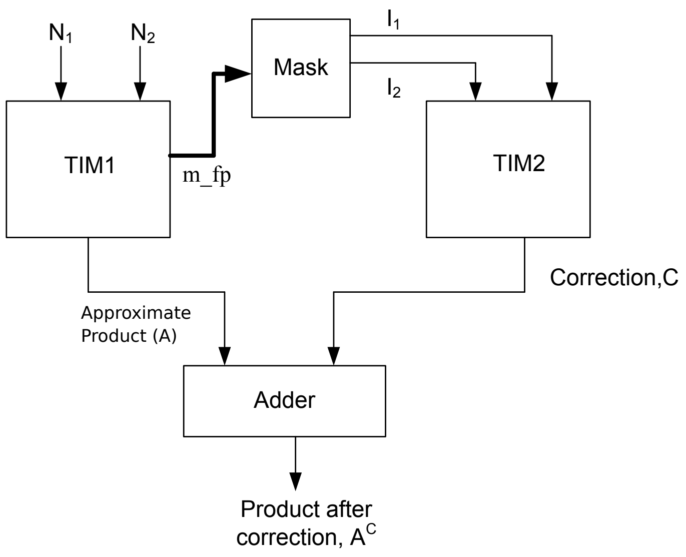


Figure 5.6: Truncated iterative multiplier implementation for two iterations

## 5.3 Error Analysis

Exhaustive error analysis has been carried out using MATLAB to compare the accuracy of the proposed multiplier with the existing iterative multiplier [26, 27].

The comparison of maximum and average error of existing designs with $T_{6,0}$ and $T_{4,1}$ schemes for three iterations is illustrated in Figs.5.7 & 5.8 respectively. Clearly, it can be observed from both the graphs that except for the first iteration, truncated TIM has least maximum and average error. A more elaborate analysis corresponding to maximum and average error for three iterations is presented in Table.5.1.

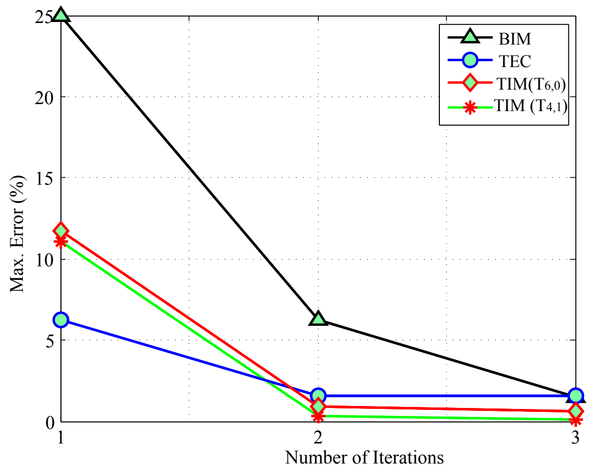


Figure 5.7: A comparison of maximum error of existing iterative multiplier designs with TIM ($T_{6,0}$ and $T_{4,1}$) approach

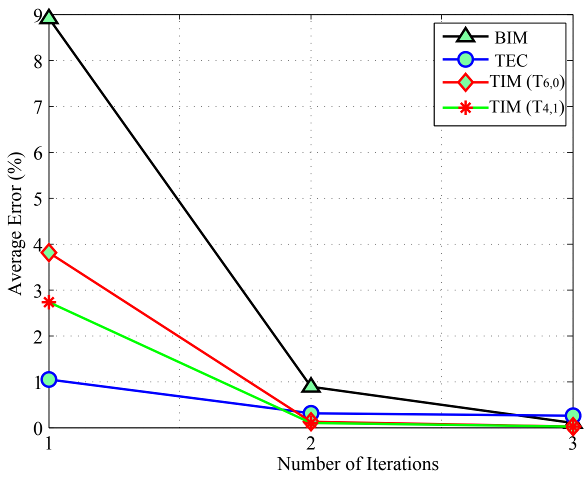


Figure 5.8: A comparison of average error of existing iterative multiplier designs with TIM ($T_{6,0}$ and $T_{4,1}$) approach

For a fair comparison, the following have been taken into consideration. The Babic iterative multiplier (BIM) [26] is devoid of error correction logic and hence the maximum and average error shown in Table.5.1 do not mention the number of mantissa fractional bits. The error analysis carried out in TEC [27] corresponds to 3-bit and 4-bit MFP hence for proper comparison the proposed TIM scheme with 3-bit and 4-bit MFP respectively is considered. Moreover, analysis for 1-bit and 2-bit MFP proposed in this work has been carried out to prove the improvement in precision achieved compared to higher bit TEC. Error metrics (maximum and average error) have been considered and compared with that of existing designs for each iteration.

A comparison of the proposed TIM scheme ($T_{6,0}$) (as per the equations 5.5 and 5.6 ) with BIM and TEC for maximum and average error for three iterations is shown in Table.5.1. It can be observed that BIM multiplier is taken as a reference (ratio of 1) corresponding to each iteration. The TIM ($T_{6,0}$) performs as well as BIM with no MFP while it performs better even with 1-bit MFP. Although, TIM approach achieves less precision compared to TEC [27] in the first iteration, the same gets better with further iterations. Maximum error with 1-bit and 2-bit MFP design has an improvement of at least 64% compared to both TEC of 3-bit MFP and BIM in second iteration. Also, the TIM design with 3-bit and 4-bit MFP has 5X and 1.7X better precision compared to TEC (3-bit and 4-bit) respectively in second iteration. In third iteration, the maximum error of the proposed scheme with 3-bit and 4-bit MFP is 9X and 2.5X better than TEC (3-bit and 4-bit) MFP respectively.

Table 5.1: A comparison of maximum and average error (%) in the proposed TIM ($T_{6,0}$) and existing schemes for 3 iterations

| | | Maximum Error (%) | | | | | | Average Error (%) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme | No. of MFP bits | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio |
| BIM [26] | - | 25 | 1 | 6.25 | 1 | 1.5 | 1 | 8.9 | 1 | 0.89 | 1 | 0.083 | 1 |
| TIM | 0-bit | 25 | 1 | 6.25 | 1 | 1.5 | 1 | 8.9 | 1 | 0.89 | 1 | 0.083 | 1 |
| TIM | 1-bit | 16 | 0.64 | 2.25 | 0.36 | 1.11 | 0.74 | 5.18 | 0.58 | 0.24 | 0.27 | 0.011 | 0.13 |
| TIM | 2-bit | 14 | 0.56 | 1.75 | 0.28 | 0.82 | 0.55 | 4.12 | 0.46 | 0.16 | 0.18 | 0.006 | 0.07 |
| TIM | | 12.5 | 0.5 | 1.19 | 0.19 | 0.67 | 0.45 | 3.85 | 0.43 | 0.13 | 0.14 | 0.005 | 0.06 |
| TEC [27] | 3-bit | 6.25 | 0.25 | 6.25 | 1 | 6.25 | 4.16 | 1.36 | 0.15 | 0.77 | 0.86 | 0.76 | 9.15 |
| TIM | | 11.75 | 0.47 | 0.92 | 0.14 | 0.6 | 0.4 | 3.8 | 0.42 | 0.12 | 0.13 | 0.004 | 0.05 |
| TEC [27] | 4-bit | 6.25 | 0.25 | 1.56 | 0.25 | 1.56 | 1.04 | 1.05 | 0.11 | 0.304 | 0.34 | 0.26 | 3.13 |

Further, Table.5.1 provides a comparison of the proposed TIM scheme ($T_{6,0}$) (as per the equations 5.5 and 5.6 ) with BIM and TEC for average error. TIM scheme with 1-bit and 2-bit MFP design has at least 54% improvement in error compared to both TEC of 3-bit MFP and BIM in second iteration. Moreover, TIM design with 3-bit and 4-bit MFP has 6X and 2.5X better precision compared to TEC, 3-bit and 4-bit, respectively in second iteration. In third iteration, the average error of the proposed scheme with 3-bit and 4-bit MFP is 152X and 65X better compared to TEC, 3-bit and 4-bit, MFP respectively.

Having established the superiority of the $T_{6,0}$ technique over BIM and TEC, the following Table 5.2 provides a comparison of the same with TIM ($T_{4,1}$). An improvement in maximum and average error in truncated multiplier $(T_{4,1})$ compared to $T_{6,0}$ is evident from this Table. Finally, as mentioned earlier in Section 5.2.2, irrespective of the number of iterations, $T_{6,0}$ is used for initial iterations while the final iteration is carried out using $T_{4,1}$. For instance, if total number of iterations is three, the initial two iterations are performed using $T_{6,0}$ while the third iteration is carried out using $T_{4,1}$.

### 5.3.1 Area and Delay Comparison of Various Multipliers using Unit Gate Level Modeling

All the logarithmic multipliers have been analyzed using unit gate modeling as this approach provides a decent model for estimating the real cost of each component and does not depend strongly on any synthesis tool. Design metrics, area $(A)$ and delay $(D)$, have been considered

Table 5.2: A comparison of maximum and average error (%) in the proposed TIM ($T_{4,1}$) for 3 iterations

| | No. of MFP | Maximum Error (%) | | | | | | Average Error (%) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme | bits | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio |
| TIM ($T_{6,0}$) | | 16 | 1 | 2.25 | 1 | 1.11 | 1 | 5.18 | 1 | 0.24 | 1 | 0.011 | 1 |
| TIM ($T_{4,1}$) | 1-bit | 15.91 | 0.99 | 2.2 | 0.97 | 1.01 | 0.9 | 4.02 | 0.77 | 0.2 | 0.83 | 0.0107 | 0.97 |
| TIM ($T_{6,0}$) | | 14 | 1 | 1.75 | 1 | 0.82 | 1 | 4.12 | 1 | 0.16 | 1 | 0.006 | 1 |
| TIM ($T_{4,1}$) | 2-bit | 13.46 | 0.96 | 1.52 | 0.86 | 0.5 | 0.6 | 3 | 0.72 | 0.12 | 0.75 | 0.005 | 0.83 |
| TIM ($T_{6,0}$) | | 12.5 | 1 | 1.19 | 1 | 0.67 | 1 | 3.85 | 1 | 0.13 | 1 | 0.005 | 1 |
| TIM ($T_{4,1}$) | 3-bit | 11.95 | 0.95 | 0.8 | 0.67 | 0.34 | 0.5 | 2.81 | 0.7 | 0.09 | 0.69 | 0.004 | 0.8 |
| TIM ($T_{6,0}$) | | 11.75 | 1 | 0.92 | 1 | 0.6 | 1 | 3.8 | 1 | 0.12 | 1 | 0.004 | 1 |
| TIM ($T_{4,1}$) | 4-bit | 11.11 | 0.94 | 0.56 | 0.68 | 0.3 | 0.5 | 2.72 | 0.71 | 0.08 | 0.67 | 0.002 | 0.5 |

and compared for all the designs. Assumptions made while calculating them are shown in Table 5.3. Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates, for both area and delay. Moreover, a m-input gate is assumed to be composed of a tree of 2 input gates and the effects of wiring, buffering and inverting costs are neglected [59]. First, the unit gate modeling of 8-bit, 16-bit and 32-bit truncated logarithmic shifter in comparison with the logarithmic shifter is carried out and later it is extended to multiplier designs.

Table 5.3: Area and Delay metrics of basic design components

| Design Component (2-input) | Area (A) | Delay (D) |
|---|---|---|
| AND, OR, NAND, NOR Gates | 1 | 1 |
| EX-OR, EX-NOR, MUXes and Half Adder | 2 | 2 |

The area and delay of leading one detector (LOD), encoder, decoder and adder are computed as mentioned in [59] which are constant in the design. Similarly, the area $(A_{LS})$ and delay $(D_{LS})$ of the logarithmic shifter (LS) without truncation is computed based on equations 5.11 and 5.12 shown below. The area of truncated logarithmic shifter with t = 4 is calculated and shown in Table.5.4.

$$A_{LS}(n) = A_{mux}\left(n + \sum_{0}^{t} 2^{T}\right) \tag{5.11}$$

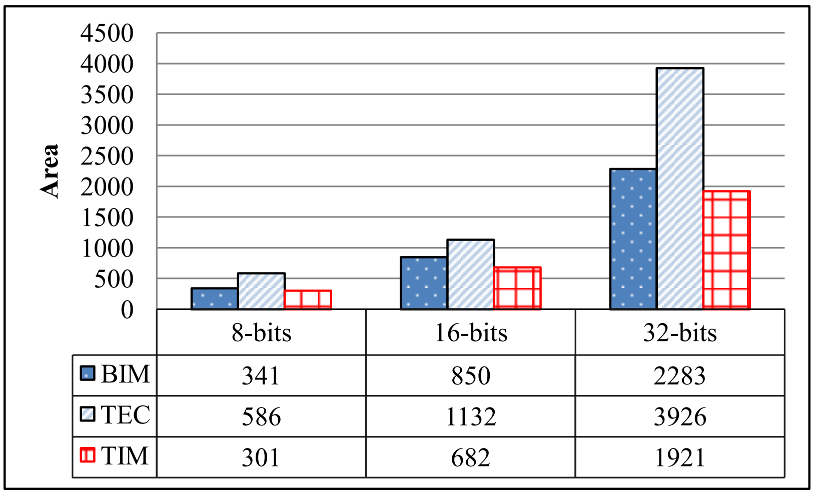
$$D_{LS}(n) = D_{mux} * log_2(n) \tag{5.12}$$

From Table.5.4, it is clear that the truncated logarithmic shifter is more area efficient compared to the logarithmic shifter. For example, the 16-bit truncated shifter occupies 35% less area compared to the logarithmic shifter. However, the delay of the truncated shifter is same as that of normal shifter since the number of logic levels of computation does not change.

Table 5.4: Area of logarithmic and truncated logarithmic shifter computed using unit gate analysis

| Logarithmic Shifter | No. of bits | Area | Percentage |
|---|---|---|---|
| logarithmic Shifter [29] | 8-bit | 84 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 60 | 71% |
| Logarithmic Shifter [29] | 16-bit | 240 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 156 | 65% |
| Logarithmic Shifter [29] | 32-bit | 420 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 348 | 83% |

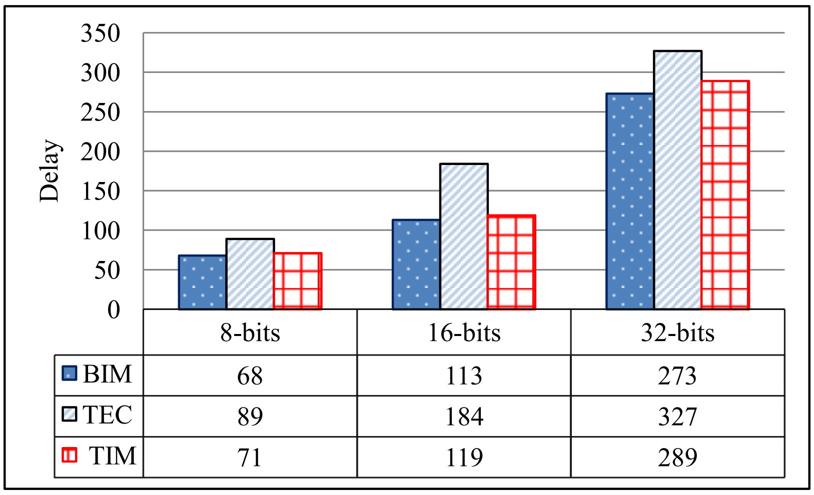
The unit gate area and delay analysis is carried out for various multiplier schemes in [26,27], and the proposed TIM designs for 8-bit, 16-bit and 32-bit. Results have been compiled and show in graph in Fig.5.9, that highlight the area savings achieved by TIM design compared to BIM. For example, the 16-bit TIM scheme occupies 20% less area compared to the BIM of same bit-width. The reason behind the area savings are attributed to the new truncation scheme proposed in this work. Similarly, it can be concluded from Fig.5.10 that the 16-bit TIM is faster compared to TEC however having a delay overhead of 5% compared to BIM of 16-bit width.



| | 8-bits | 16-bits | 32-bits |
|---|---|---|---|
| ■ BIM | 341 | 850 | 2283 |
| ▨ TEC | 586 | 1132 | 3926 |
| ⊞ TIM | 301 | 682 | 1921 |

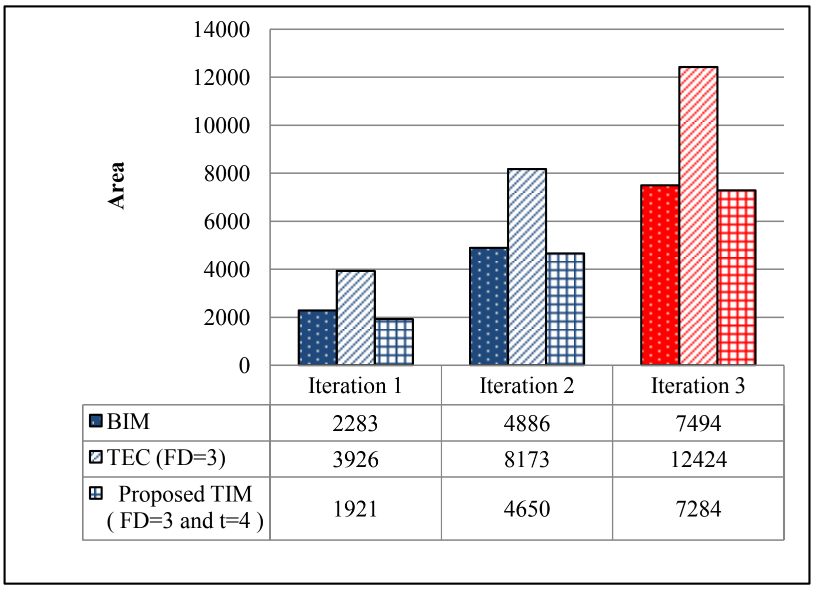
Figure 5.9: Area comparison of various multipliers for different bit-widths

Figures 5.11 & 5.12 present the area and delay improvement achieved by different 32-bit iterative multiplier schemes for three iterations. It is observed from Fig.5.11 that the area occupied by TIM is less compared to BIM and TEC in all iterations. For example, 32-bit TIM scheme in the first iteration occupies at least 19% less area compared to BIM and TEC designs.



| | 8-bits | 16-bits | 32-bits |
|---|---|---|---|
| ▦ BIM | 68 | 113 | 273 |
| ▨ TEC | 89 | 184 | 327 |
| ▦ TIM | 71 | 119 | 289 |

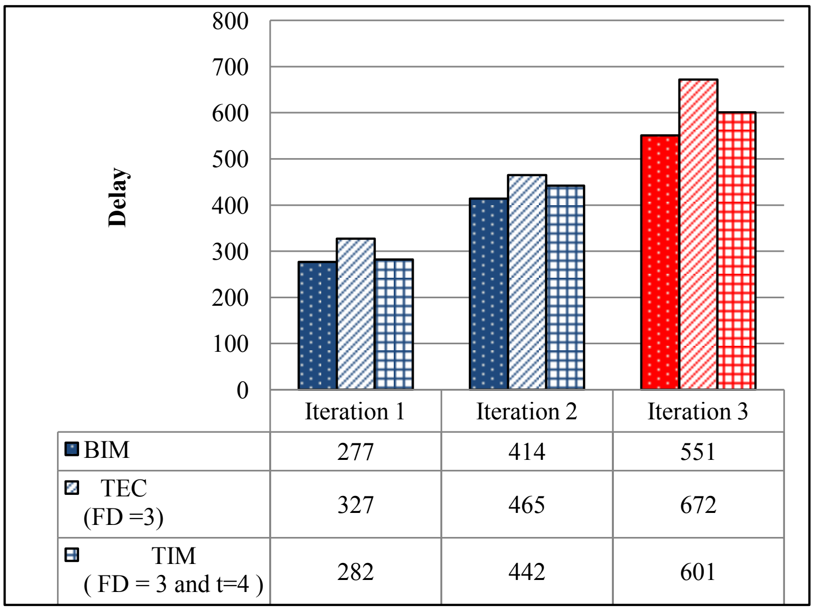
Figure 5.10: Delay comparison of various multipliers for different bit-widths



| | Iteration 1 | Iteration 2 | Iteration 3 |
|---|---|---|---|
| ▦ BIM | 2283 | 4886 | 7494 |
| ▨ TEC (FD=3) | 3926 | 8173 | 12424 |
| ⊞ Proposed TIM ( FD=3 and t=4 ) | 1921 | 4650 | 7284 |

Figure 5.11: Area comparison of various multipliers for three iterations

The delay comparison of various multipliers for three iterations is presented in Fig.5.12. It is observed that TIM method performs better than TEC in terms of delay for same precision. However, it appears to have more delay compared to BIM which is not surprising because the precision provided by BIM is much less compared to TIM and thus needs more iterations leading to more delay if a precision similar to that of TIM is to be achieved. The delay overhead in TEC is due to carry speculation. The proposed design overcomes this problem by successfully replacing the complex fractional predictor design with a simple one (MFP) as discussed in Section 5.2.3.2.



| | Iteration 1 | Iteration 2 | Iteration 3 |
|---|---|---|---|
| ■ BIM | 277 | 414 | 551 |
| ▨ TEC (FD =3) | 327 | 465 | 672 |
| ⊞ TIM ( FD = 3 and t=4 ) | 282 | 442 | 601 |

Figure 5.12: Delay comparison of various multipliers for three iterations

### 5.3.2 Synthesis Results of Various Multipliers

For a fair comparison, TEC and BIM multipliers and the TIM have been modeled using Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1. They are mapped on to TSMC 180nm technology slow-normal library using cadence RTL compiler v7.1.

Table 5.5: Synthesis results of various 32*32 multipliers for one iteration

| 32 * 32 Multiplier | Area ($\mu m^2$) | % change | Delay($ps$) | % change | Area-Delay Product ($*10^5$) ($\mu m^2 - ps$) | % change |
|---|---|---|---|---|---|---|
| BIM [26] | 3675 | 100% | 15454 | 100% | 568 | 100% |
| TEC [27] | 5696 | 155% | 18235 | 117% | 1038 | 182% |
| Proposed TIM ($T_{6,0}$) | 3285 | 90% | 16088 | 104% | 528 | 93% |

Hardware synthesis has been carried out to compare the important metrics such as area and delay. Table.5.5 provides a performance comparison of 32*32 multiplier designed with above three schemes for one iteration. As seen from the Table, the area consumed by the proposed TIM scheme is 10% less compared to BIM and 65% less compared to TEC. While TIM has a delay overhead of 4% compared to BIM, it has a 64% better precision than BIM. Although, BIM performs better in delay with more iterations, TIM achieves a much better precision compared to BIM with only a marginal increase in delay. The reason for increase in delay in TIM scheme is due to the inclusion of mask from second iteration onwards. Nevertheless, TIM performs much better in terms of precision, area and delay compared to TEC. It is also evident from Table.5.5 that TIM scheme has improved area-delay product (7 to 89%) compared to TEC and BIM respectively. Overall, TIM scheme outperforms all other similar designs that exist in the literature.

## 5.4 Benchmarking Various Multiplication Schemes-Application to Image processing

Image sharpening is an important image enhancement technique employed in image processing applications. The computational process of sharpening an image involves a number of fixed point multiplications. It is therefore a good application to prove the efficacy of the proposed truncated iterative multiplier.

### 5.4.1 Image Sharpening Algorithm

Human perception is highly sensitive to edges and fine details of an image. Since images essentially consist of high-frequency components their visual quality is corrupted if these high frequencies are removed. Conversely, increasing the high-frequency components of an image improves the image quality. Image sharpening algorithm described in [60] is one such enhancement technique which highlights the edges and fine details in an image.

This algorithm described below, accepts an image, processes it, and produces an image of high quality. Suppose $I$ is the original image, the processed image $S$ is described using the expression

$$S(x,y) = 2I(x,y) - M \tag{5.13}$$

where $M = \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} H(i+3,\, j+3)\, I(x-i, y-j)$

and $H$ is a matrix defined as

$$H = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Since this expression involves a number of multiplications, an exact multiplier such as an array multiplier can perform these operations accurately thereby producing an image of high quality. On the other hand, using an approximate multiplier would result in an image of certain quality which is quantified using established metrics such as mean square error (MSE) and peak signal to noise ratio (PSNR).

The MSE represents the loss of information in the image and is expressed as,

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{5.14}$$

The peak signal to noise ratio (PSNR) in dB is expressed using $MSE$ as follows:

$$PSNR \text{ in dB} = 10 log_{10} \left( \frac{MAX_I^2}{MSE} \right) \tag{5.15}$$

where $MAX_I$ represents the maximum possible pixel value of the image.

While the use of approximate multiplier affects the image quality, it has the advantage of savings in terms of area and delay as compared to an accurate multiplier. In what follows, the performance of the existing multipliers such as BIM and TEC and the proposed TIM is studied and compared with reference to the image sharpening algorithm. The algorithm is applied to blocks of 5*5 pixels on a set of standard images ( Cameraman and Lena). The exact multiplications are replaced by approximate multiplications using BIM, TEC and TIM, while addition, subtraction and division operations are carried out using accurate techniques. The metric MSE is computed by finding the mean of squares of difference in pixel values between original image and the processed image using approximate multipliers and these values are substituted in equation (5.15) to calculate PSNR values.

Table 5.6 provides a comparison of these metrics on a set of standard images (Lena and

Cameraman). While PSNR of the proposed TIM is less compared to that of TEC in the first iteration, the same gets better with higher number of iterations. This is due to the increase in number of iterations, the accuracy of TIM improves over that of TEC. It can also been seen from Fig.5.13 where maximum and average error for all the techniques is compared.

Table 5.6: A comparison of values of MSE and PSNR for benchmark images using various multiplier schemes

| | | 1st Iteration | | | 2nd Iteration | | | 3rd Iteration | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Image | Metric | BIM | TEC | TIM | BIM | TEC | TIM | BIM | TEC | TIM |
| Cameraman | | 43.2 | 43.4 | 43.4 | 43.46 | 43.6 | 43.8 | 43.7 | 44.2 | 44.45 |
| Lena | | 39 | 39.4 | 39.2 | 39.2 | 39.5 | 39.53 | 39.4 | 39.5 | 39.7 |
| Pirate | PSNR ( dB) | 42.8 | 43 | 42.8 | 43.1 | 43.2 | 43.4 | 43.2 | 43.4 | 43.7 |
| Cameraman | | 3.1 | 3 | 3 | 2.95 | 2.8 | 2.7 | 2.8 | 2.5 | 2.35 |
| Lena | | 7.9 | 7.5 | 7.7 | 7.75 | 7.35 | 7.3 | 7.5 | 7.2 | 7.05 |
| Pirate | MSE | 3.42 | 3.3 | 3.39 | 3.2 | 3.15 | 3 | 3.1 | 3 | 2.8 |

Further, BIM has the highest error for all iterations while between TEC and TIM, the latter has relatively larger error in the first iteration which however falls sharply from the second iteration onwards. While the difference between TEC and TIM for higher number of iterations is not apparent from the graph, it is clear from Table 5.1 that TIM technique has a maximum error and an average error that is less than that of TEC.

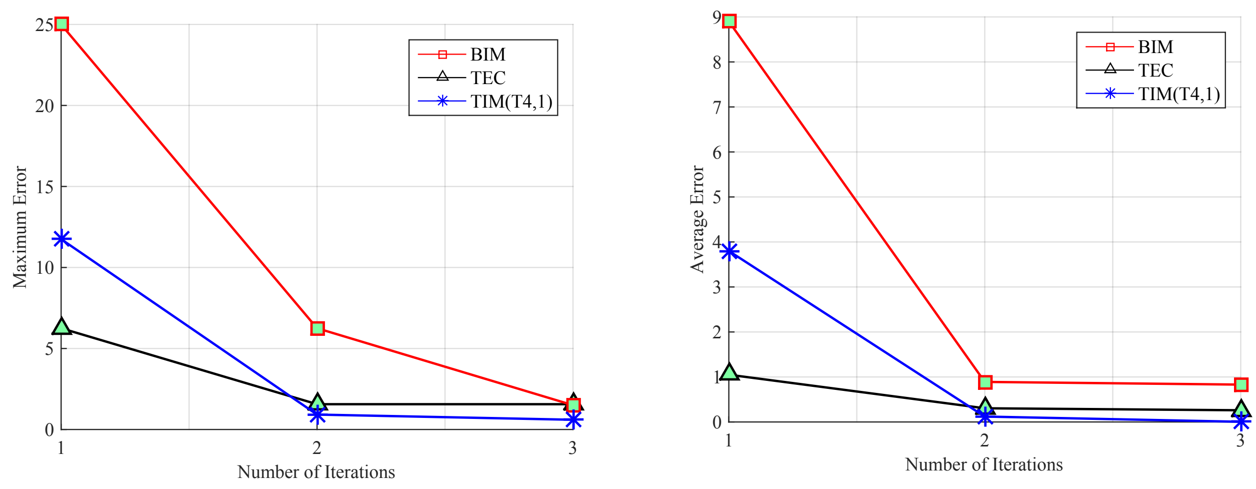


Figure 5.13: Maximum and Average errors of various multipliers for three iterations

In what follows, BIM, TEC, and TIM are compared for precision while keeping in mind the area and delay performance of the hardware. Figures.5.14 and 5.15 provide a comparison of area and delay performance of various multiplier schemes. It is evident from these figures that

TIM performs better than TEC. Although, as stated earlier, TEC has better precision for first iteration, TIM performs better from the second iteration onwards. Also, TIM performs better in terms of both area and delay for any number of iterations.

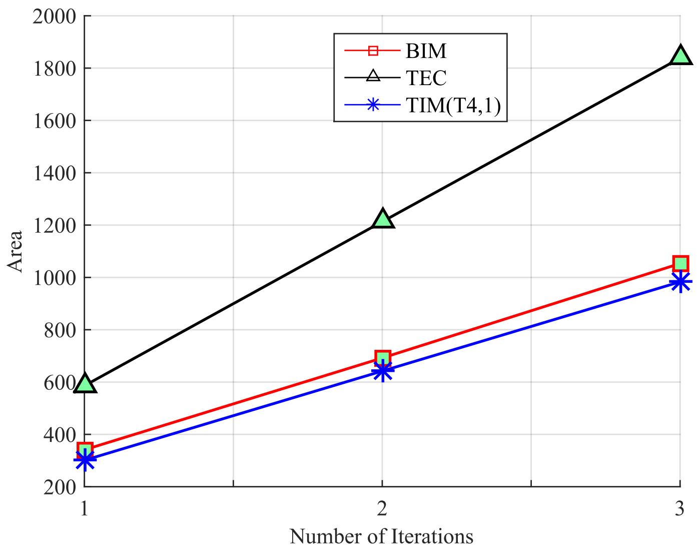


Figure 5.14: Unit gate area statistics of various multiplier for three iterations

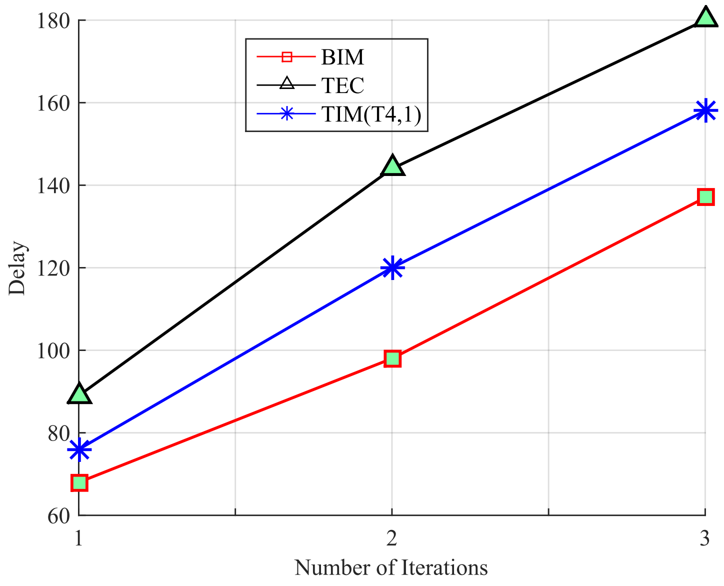


Figure 5.15: Unit gate delay statistics of various multiplier for three iterations

Further, while TIM has a slightly better performance than BIM in terms of area, BIM does well in terms of delay (Fig.5.15). However, this is not considered significant in the current context since TIM has better precision as well as better PSNR and MSE for the benchmark images considered, compared to BIM.

Also, the performance of the proposed multiplier, in terms of both PSNR and MSE, may be understood by observing the images processed by it. Figure.5.16 illustrates the images that are processed using exact (array) and the TIM. It can be observed that the images of Lena and Cameraman processed with the proposed multiplier look very similar to the original ones.

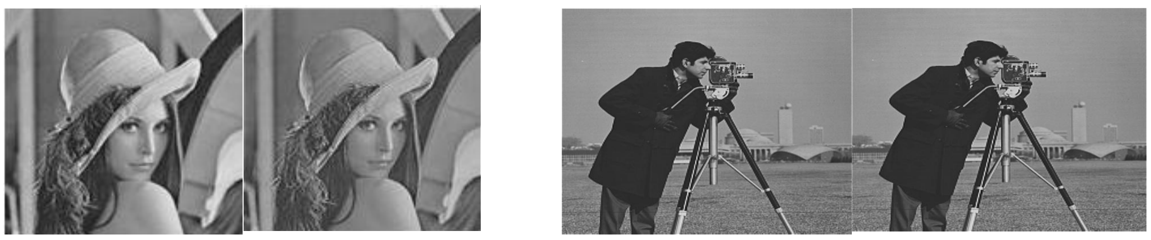

(a) Original Image (b) Processed Image (a) Original Image (b) Processed Image

Figure 5.16: Lena and Cameraman images obtained using exact and the proposed multiplier

## 5.5 Conclusions

In this work, an improved iterative multiplier has been proposed based on Mitchel algorithm with enhanced precision. A new fractional detector scheme and a modified truncation method presented significantly reduce the area of the related hardware. The fractional detector logic and its efficient precomputation contribute to improved overall accuracy due to fewer number of iterations required compared to the existing ones. Further, the precision of the multiplier improves as the number of iterations increases. Performance improvement has been achieved through the use of truncated logarithmic shifter and a fractional predictor.

Extensive analysis of the existing (TEC and BIM) and the proposed (TIM) schemes has been carried out using unit gate modeling and compared with that obtained using synthesis tool. Results of accuracy and hardware performance prove the superiority of the TIM technique over TEC and BIM. The same has also been validated using image processing benchmarks, Lena, cameraman, and pirate.

# Chapter 6

# An Improved ‘Digit-by-Digit’ Decimal Multiplier

## 6.1 Introduction

The previous chapters presented improved designs of binary and logarithmic multipliers. However, it may be noted that decimal arithmetic is preferred in applications such as financial, scientific and commercial etc. owing to their higher precision compared to binary arithmetic. However, these computations are generally sluggish (slow) and tend to occupy more silicon area [48]. This has led to efforts in improving decimal architectures to enable high performance and compact arithmetic circuits. Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic, is multiplication. While a large body of literature on decimal arithmetic covers serial multiplication [62, 63], parallel (‘word-by-digit’) [40–43] and (‘digit-by-digit’) [44, 45] multiplication has also been reported recently. Decimal (BCD) ‘digit-by-digit’ multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in improvement in the multiplier performance [46]. In this work, we focus on developing efficient architectures for decimal ‘digit-by-digit’ multiplication.

The partial products in ‘digit-by-digit’ multiplication scheme are generated using BCD digit-multiplier (BDM) and their reduction is accomplished using carry-free binary adders, multi-operand binary to decimal (BD) converters and decimal adder. Since BDM is an important component in partial product generation, we focus on new and improved designs for BDM cells in this chapter. Besides, novel designs of multi-operand BD converters are proposed to convert the column binary sum to decimal in partial product reduction. Further, a hybrid multi-operand BD converter algorithm is proposed and analyzed for its performance. It is expected that these improvisations would result in significant savings in terms of area and latency.

Throughout this chapter, upper and lower case letters are used to signify decimal digits and binary bits respectively, where a digit represents a 4-bit BCD number. The symbols ‘ . ’ and ‘ + ’ are used to denote AND and OR gates while the symbols ‘$\oplus$’ and ‘$\odot$’ denote XOR and XNOR operations respectively. Further, the term binary coded decimal (BCD) is used interchangeably with decimal.

Rest of the chapter is organized as follows: An outline of the proposed partial product generation and reduction schemes in 16*16 ‘digit-by-digit’ multiplier is provided and discussed in Sections 6.2 and 6.3. In addition, design of hybrid multi-operand BD converters is described in Section 6.3.2. A detailed performance analysis of 16*16 ‘digit-by-digit’ multiplier is carried out and compared in Section 6.4.

## 6.2 A New Partial Product Generation Scheme in ‘Digit-by-Digit’ Multiplier

This section presents two new partial product generation (PPG) schemes for improved area and performance of BDM cell in ‘digit-by-digit’ multiplication. This is achieved through novel designs of PPBD converter which forms a part of the BDM cell.

### 6.2.1 High Performance Partial Product Binary to Decimal (PPBD) Converter

The first of the proposed two PPBD converters, the ‘high performance’ PPBD converter, is designed using the fast BD (FBD) converter cells. A typical FBD cell, would accept a 4-

bit binary input ($b_j$), multiplies it by four, and then adds it to $b_i$ of 2-bits as illustrated in Fig.6.1(a). Thus the computation of BCD (decimal) outputs, $H_0$ (higher digit) and $L_0$ (lower digit) is carried out by using the relation $\{H_0, L_0\} = 4\,.\,b_j + b_i$ where the maximum values of $H_0$ and $L_0$ are $(3)_{10}$ and $(9)_{10}$ respectively. Therefore, the output of FBD cell is limited at most to $(39)_{10}$ unlike $(19)_{10}$ in Nicoud cell [50]. The binary inputs, $b_j$ and $b_i$ comprise of $\{b_3, b_2, b_1, b_0\}$ and $\{b_{i1}, b_{i0}\}$ bits respectively.

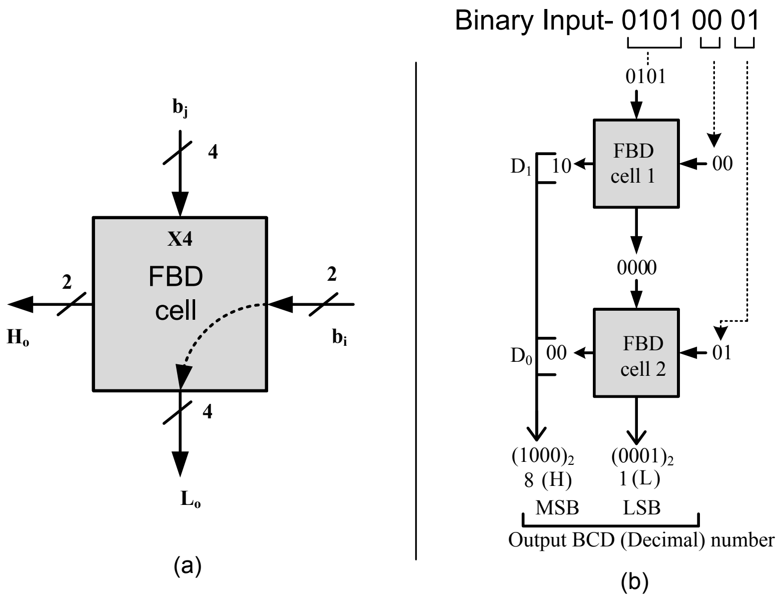


Figure 6.1: (a) Compact notation of FBD cell (b) Linear array of FBD cells to form PPBD converter

The FBD cell is shown in Fig.6.1(a) and is described by the following equations derived using truth tables :

$$\begin{cases} H_0\,[1] = b_3 + b_2.b_1^{'}.b_0^{'} + b_2^{'}.b_1.b_0 + b_2^{'}.b_1.b_{i0} \\ H_0\,[0] = b_3 + b_2.(b_1 + b_0) \\ L_0\,[3] = b_{i0}.b_3.b_0 + b_1.(b_{i0}^{\prime}(b_2 \circledcirc b_0) + b_1^{'}.(b_2 b_{i0} b_0^{'}) \\ L_0\,[2] = b_3.b_0^{'}.b_{i0} + b_2.b_0^{'}.(b_1 + b_{i0}^{'}) + b_2^{'}.b_0.(b_1^{'} + b_3^{'}.b_{i0}) \\ L_0\,[1] = b_2^{'}.(b_1^{'}.(b_3 \oplus b_{i0}) + b_2.(b_0^{'}.(b_1 \circledcirc b_{i0}) + \emptyset \\ L_0\,[0] = b_{i1} \end{cases} \qquad (6.1)$$

where $\emptyset$ $= b_0(b_2^{\prime}.b_1.b_{i0}^{\prime} + b_2.b_1^{\prime}.b_{i0})$

An example to convert a 7-bit binary number to two BCD digits (H and L) is illustrated in Fig.6.1(b). Since the binary number $(01010001)_2$ to be converted is larger than $(39)_{10}$, two FBD cells are required to realize the converter. As illustrated in Fig.6.1(b), the input to the FBD cell 1 is restricted to $(1001)_2$. Hence the 3 MSBs of binary input along with a '0' prepended $(0101)_2$ are accepted as $b_j$ and the next significant two binary input bits "00" as $b_i$ results in the outputs $(10)_2$ and $(0000)_2$. The 4-bit output $(0000)_2$ of cell 1 along with residual binary input "01" form inputs to cell 2, resulting in higher (H) digit $(1000)_2$ formed by $\{D_1, D_0\}$ and lower (L) digit $(0001)_2$. In general, binary number of any operand width can be converted to BCD (decimal) by a linear arrangement of FBD cells.

Since a BCD digit can take values only between $(0)_{10}$ and $(9)_{10}$, the output of the 4*4 binary multiplier in BDM illustrated in Fig.2.24 is restricted to $(81)_{10}$, that is equivalent to $(1010001)_2$. Thus, the input binary number to a PPBD converter is limited to $(1010001)_2$. This gives a possibility for optimization of FBD cell 1 shown in Fig.6.1(b), which eventually results in simplified PPBD converter. As a consequence, the input to the FBD cell 1 is restricted to $(0101)_2$ as depicted in Fig.6.1(b). In view of this, the FBD cell 1 in Fig.6.1(a) gets simplified resulting in 'low area' BD (LABD) cell as shown in Fig.6.2(a). The binary inputs $\{b_2, b_1, b_0\}$ and $\{b_{i1}, b_{i0}\}$ are applied to this cell which converts them into equivalent decimal number $H_0\,[1:0]$ and $L_0[3:0]$.

The resulting simplified equations for LABD cell which can be obtained from truth table can be written as :

$$\begin{cases} H_1\,[1] = b_2.b_0' + b_1.(b_0 + b_{i0}) \\ H_0\,[0] = b_2.b_0 \\ L_0\,[3] = b_1.b_0'.b_{i0}' + b_2.b_{i0} \\ L_0\,[2] = b_0.(b_2'.b_1 + b_{i0}) + b_2.b_0'.b_{i0}' \\ L_0\,[1] = b_{i0}'.(b_2.b_0' + b_1.b_0) + b_2'.b_0'.b_{i0} \\ L_0\,[0] = b_{i1} \end{cases} \qquad (6.2)$$

Figure.6.2(b) depicts the ‘high performance’ PPBD (HPPPBD) converter scheme achieved by a linear arrangement of LABD and FBD cells. This converter has improved performance in terms of delay as illustrated later.

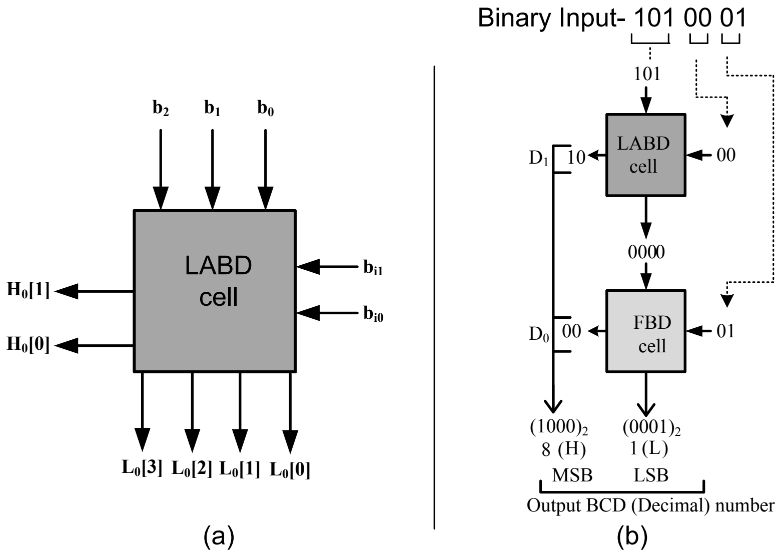


Figure 6.2: (a) Compact notation of LABD cell (b) Linear array of LABD and FBD cells to form ‘high performance’ PPBD (HPPPBD) converter

### 6.2.2 Low Area Partial Product Binary to Decimal Converter (LAPPBD)

Work in [51] presents partial product reduction using a multi-operand BD converter consisting of Nicoud cells as illustrated in Fig.6.3(a). We propose a design of PPBD converter using Nicoud cells and FBD cells which however are used for partial product generation to achieve area reduction as well as improve the performance. This design comprises of a linear array of LABD cells (shown earlier in Fig.6.2(a)) and Nicoud cells as illustrated in Fig.6.3(b). Since Nicoud cells have more latency, achieving a PPBD converter with these cells alone results in a slower design. An alternative method is to use area optimized and faster LABD cell so as to improve the performance of PPBD converter.

Referring to Fig.6.3(a), Nicoud cells (circled with dotted lines) are replaced with LABD cell resulting in a PPBD converter shown in Fig.6.3(b) which has the advantage of lower area as compared to all Nicoud cell converter.

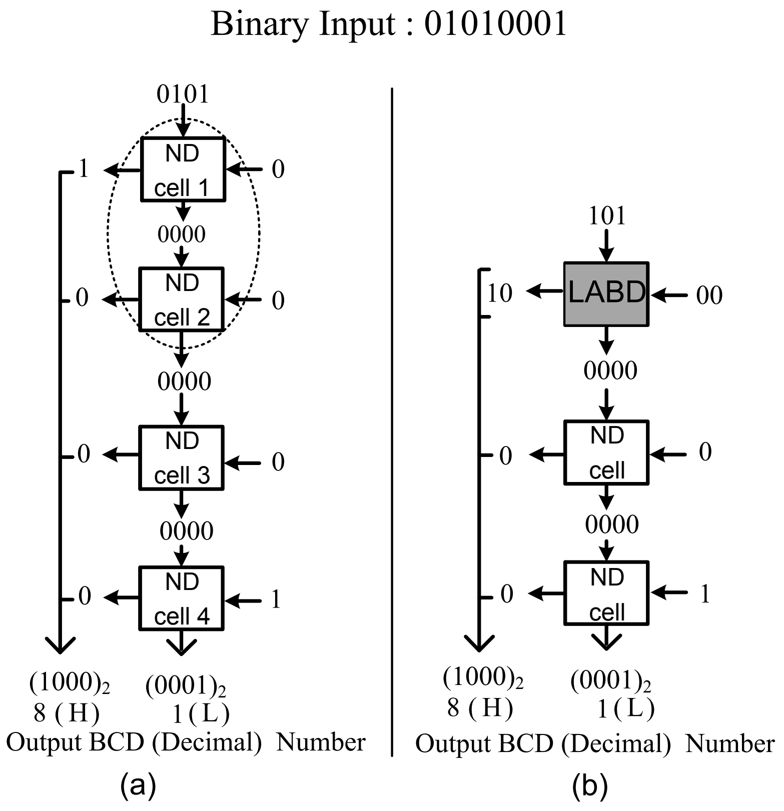


Figure 6.3: (a) Iterative array of Nicoud cells to form BDM (b) Linear array of Nicoud and LABD cells to form LAPPBD

## 6.3 Partial Product Reduction (PPR) in ‘Digit-by-Digit’ Multiplier

This section discusses the decimal (BCD) partial product reduction scheme in ‘digit-by-digit’ multiplier using the proposed multi-operand BD converters. An example of this approach is illustrated in Fig.6.4. Referring to decimal partial product of column length C=6 with six partial products, each partial product consists of 4-bits, denoted with solid dots. The columns c0, c1, c2 and c3 are reduced to two rows using tree of 3:2 binary Carry Save Adders (CSA) which requires 2 levels to compress all the columns into two rows. These two rows are reduced to further obtain the final binary result using the carry propagation adder represented with straight line in grey color as shown in Fig.6.4.

A numerical example illustrates the addition of six BCD digits, each of 4-bits $[(9)_{10} + (9)_{10} + (9)_{10} + (9)_{10} + (9)_{10} + (9)_{10}]$, as mentioned in Fig.6.5 (a). The maximum value of each digit is $(9)_{10}$. Addition of these digits is performed by a tree of CSAs and a carry propaga-

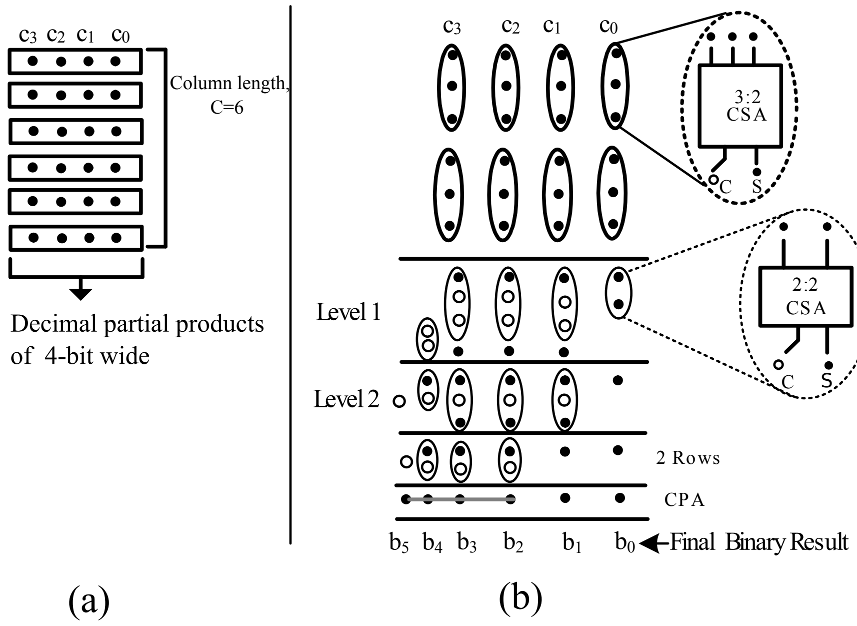


Figure 6.4: (a) Decimal partial products of six columns each of 4-bit (b) Example of partial product, denoted using dot, reduction of column size C= 6

tion adder resulting in final binary result $(110110)_2$. The conversion of this binary number to decimal (BCD) is accomplished by using the proposed multi-operand binary to decimal converter. The two FBD cells are connected in a linear array to convert binary number $(110110)_2$ to decimal $(54)_{10}$ as illustrated in Fig.6.5(b).

The implementation details of partial product reduction in 16*16 ‘digit-by-digit’ multiplier is presented below.

### 6.3.1 Implementation of 16*16 ‘Digit-by-Digit’ Multiplier

In a typical $N*N$ ‘digit-by-digit’ multiplication, $2N^2$ decimal partial products with a maximum column size of $2N-1$ are generated using BDM cells. As mentioned in Section 2.8.1, multi-operand BD (MBD) converters are used at the partial product reduction stage. The reduction scheme using these converters for $16*16$ ‘digit-by-digit’ multiplier is illustrated in Fig.6.6. It can be seen here that the layout of decimal partial products ($H$ and $L$) is in columns of varied length as illustrated in Fig.6.6(a). The reduction of decimal partial products generated happens in two steps: First, all the partial product columns are compressed in parallel using CSAs . Next, the binary number obtained from respective columns is converted in to a decimal number using the multi-operand binary to decimal (MBD) converters.

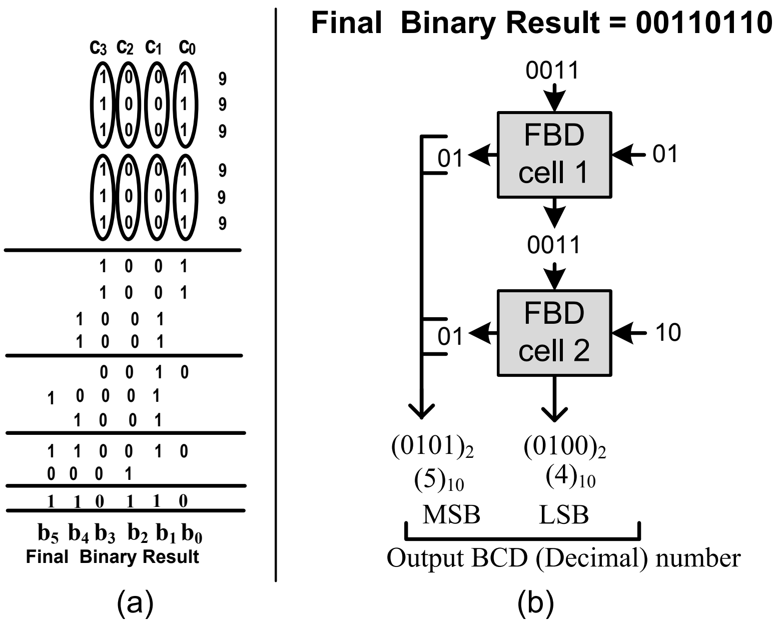
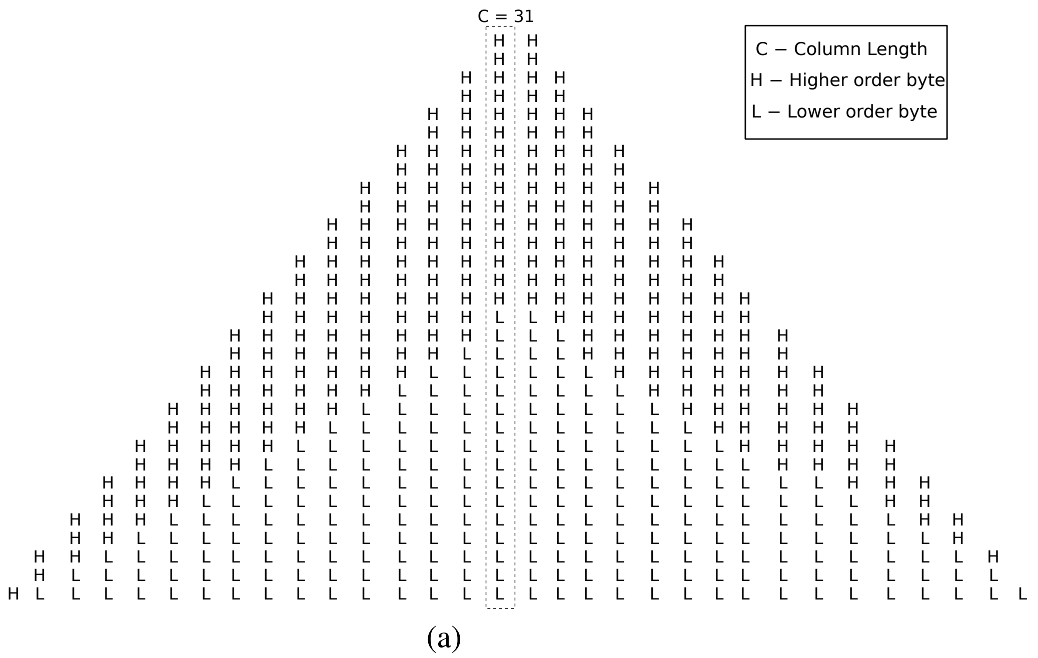


Figure 6.5: (a) Numerical example illustrating the reduction of decimal (BCD) partial products using CSAs (b) MBD converter formed using linear array of FBD cells to convert binary number to decimal (BCD)

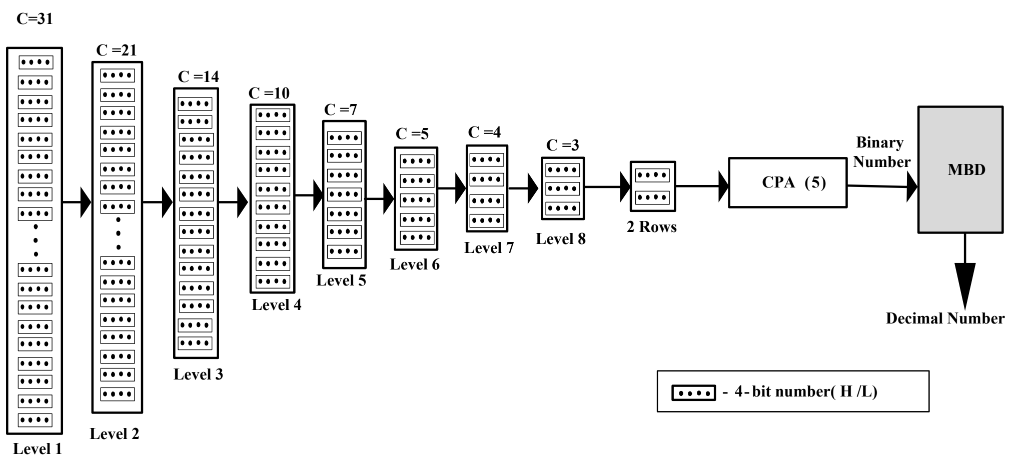
In this work, the existing MBD converter [51] is modified by replacing Nicoud cell with FBD cell resulting in improved performance. The operation of the proposed MBD converter for the largest column, $C = 31$ (highlighted by using dotted line in the Fig.6.6 (a)), is illustrated in Fig.6.6(b) as an example. The binary addition of 31-digits column comprising of $H$ and $L$ (whose maximum values, mentioned earlier, are considered) is performed using a tree of 3:2 binary CSAs [52]. It requires eight reduction levels to compress this column into 2 rows, which are eventually reduced to a binary number using the carry propagation adder (CPA). For instance, considering the case when output of the column size (C=31) results in a binary number $(0100001000)_2$, it is converted into decimal in two stages using a linear connection of FBD cells (MBD converter) as shown in Fig.6.7.

In a similar manner, binary result from respective columns are compressed in parallel to obtain decimal numbers which are then aligned according to their decimal weight and reduced to a product using the decimal adder [64].

While this approach improves the performance by reducing the latency of the MBD converter, it also results in area overhead. Further, decimal conversion of a column using Nicoud cells [51] alone, while resulting in smaller area as illustrated in Fig.6.8(a) it increases the converter latency significantly. To address these competing requirements, a hybrid multi-operand



(a)



(b)

Figure 6.6: (a) Partial product matrix in a 16*16 ‘digit-by-digit’ multiplier (b) Partial product reduction of column of largest size C=31 using CSA structure

BD converter using a mix of FBD and Nicoud cells is proposed that results in small area but high speed of operation. For instance, Fig.6.8(b) shows a hybrid MBD converter consisting of fast FBD cells in the critical path (stage1) while stage 2 comprises of (small area) Nicoud cells resulting in a hybrid MBD converter that consumes less area without compromising on speed when compared to the converter shown in Fig.6.7. Clearly, the hybrid converter is faster with only a marginal increase in area when compared with that in Fig.6.8(a). As the column size increases, selection of optimal number of FBD and Nicoud cells becomes tedious. Hence, a hybrid multi-operand BD algorithm which helps in selecting the best combination of FBD and

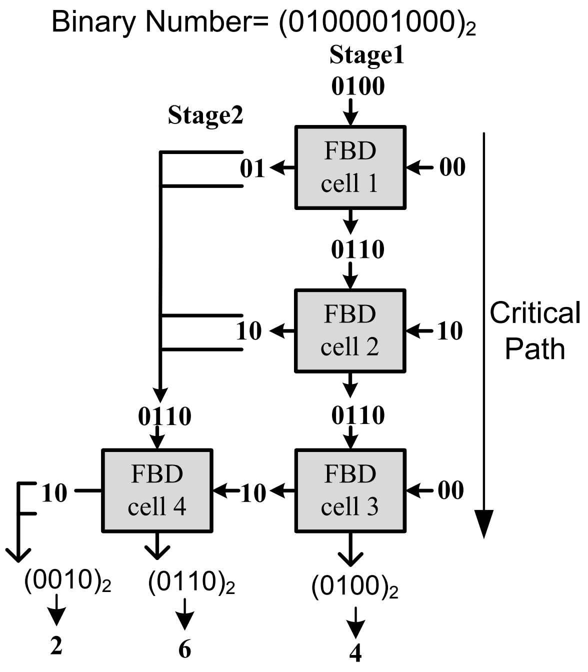


Figure 6.7: MBD converter for column size, C=31 in 16*16 ‘digit-by-digit’ multiplier

Nicoud cells is also proposed in this work and is discussed below.

### 6.3.2 Algorithm for Hybrid Multi-operand Binary to Decimal Converter

The algorithm for the design of hybrid multi-operand BD converter uses number of bits (*nob*) to compute all possible combinations of Nicoud and FBD cells. The pseudo code of the algorithm (given below) accepts the column size (C) and converts it into ‘*nob*’ using the relation $nob = ceil(log_2(C*P))$, where P is the maximum decimal weight of partial product $((9)_{10})$. For instance, given a value of $C = 31$ and $P = 9$, the value of ‘*nob*’ turns out to be 9. Depending on the number of bits, the algorithm recursively computes all possible combinations of Nicoud (ND) and fast binary to decimal (FBD) cells and their respective area and delay. The selection of appropriate multi-operand BD converter itself is based on the area and delay requirements of the design on hand. The area of individual FBD and ND cells is denoted by $A_1$ and $A_2$ respectively.

The algorithm, which calls three sub-functions Max_ND_Cell, Max_FBD_Cell and CAL is explained as follows :

1. The input to the algorithm is the binary number obtained by reducing the partial product column of length C.

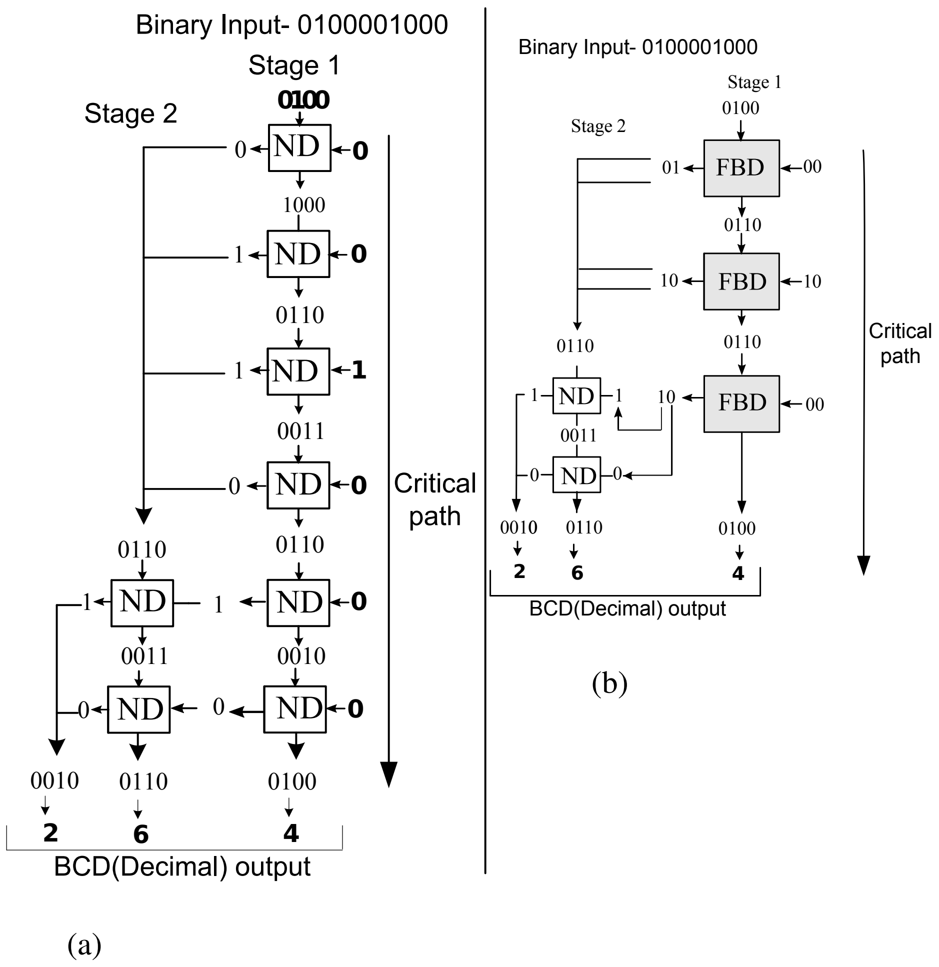


Figure 6.8: MBD converter for column size, C=31 in 16*16 ‘digit-by-digit’ multiplier (a) Using Nicoud cells (b) Hybrid converter using Nicoud and FBD cells

2. The sub-function Max_ND_Cell computes the maximum number of ND cells (mndc) required to design the converter.

3. Similarly, the sub-function Max_FBD_Cell determines the maximum number of FBD cells (mfbdc) necessary to design the converter.

4. Higher priority is given for selection of FBD cells if the objective is to obtain a low latency design.

5. The sub-function CAL calculates the number of unused FBD $(m1)$ and ND $(m2)$ cells for various valid combinations.

6. For a given set of values $m_1$ and $m_2$, the algorithm computes the total number of utilized FBD $(n_1)$ and ND $(n_2)$ cells and their area as well as the critical path delay for that combination.

7. The area and delay for all valid combinations are calculated and stored in a list as a tuple $<n_1, n_2, delay, area>$.

8. The above steps are repeated till all valid combinations are exhausted.

**Algorithm 6.1** Pseudo code of hybrid multi-operand converter

**Input**: *nob*

**Output**: utilized FBD cells $(n_1)$, utilized ND cells $(n_2)$, area, delay

```
1.  function Hybrid_MBD
2.   mndc = Max_ND_Cell
3.   mfbdc = Max_FBD_Cell
4.    for i = 0; i < mndc + 1; i++
5.     for j = mfbdc + 1; j > -1; j–
6.      $m_1$, $m_2$= CAL
7.       if $delay! = 0$
8.        if $m_1 >= 0$ and $m_2 >= 0$
9.         set $area$ to $((|i-m_1|)*A_1)+((|j-m_2|)*A_2)$
10.          if $[|i-m_1|, |j-m_2|, delay, area]$ not in list
11.           add $([|i-m_1|, |j-m_2|, delay, area])$ to list
12.          end if
13.        end if
14.       end if
15.     end for
16.    end for
17. return $(delay, area, n_1, n_2)$
```

It can be noted from Fig.6.8(b) that the critical path of the MBD converter is dictated by the first stage wherein the delay of individual cells chosen gets added up. However, in the subsequent stages the delay of final cell alone is added [51]. Therefore, to achieve a faster hybrid converter, delay per bit in the first stage has to be reduced by exclusively using FBD cells.

The algorithm terminates after storing all the values in a data set (list). No need to mention, area efficient converter can be designed using more number of ND cells. Conversely, if high speed converter is the objective, then more number of FBD cells can be used. Thus, the hybrid algorithm provides flexibility in choosing appropriate multi-operand converter.

## 6.4 Results and Discussion

In order to compare the proposed multiplier with existing designs, initially unit gate level modeling is carried out and later synthesis based analysis is performed.

### 6.4.1 Area and Delay Comparison using Unit Gate based Modeling

All the designs under consideration have been modeled using unit gate approach [59] to obtain a synthesis-independent estimate of area $(A)$ and delay $(D)$. Further, each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for both area and delay [59]. Moreover, an m-input gate is assumed to be composed of a tree of m-1 input gates while the effects of wiring, buffering and inverting costs (area and delay) are neglected.

As discussed earlier, a BDM typically consists of a binary multiplier and a partial product binary to decimal (PPBD) converter. In this work, the binary multiplier proposed by Jaberipur 2 [44], being the most efficient design in literature, is adopted for all the designs, shown in Table below, together with respective PPBD schemes to form the BDM. For instance, the HPBDM comprises of a Jaberipur binary multiplier and a HPPPBD.

Table 6.1 compares the area and delay performance of various BDM converters including the two proposed schemes (HPBDM and LABDM) mentioned earlier. Besides each value of area and delay, the percentage mentioned signifies how the proposed (HPBDM and LABDM) designs perform compared to the existing designs. It can be observed that an improvement of 5% in delay is achieved in proposed HPBDM compared to the best performing Split_4_3 BDM. Further, it can be seen that area-delay product of proposed LABDM design is better than every other design. Also, the area occupied by LABDM and HPBDM is less in comparison with

Table 6.1: Comparison of area and delay of various BDMs

| Scheme | Area | % change | Delay | % change | Area-Delay Product | % change |
|---|---|---|---|---|---|---|
| **HPBDM** | **130** | **107%** | **20** | **95%** | **2600** | **102%** |
| **LABDM** | **121** | **100%** | **21** | **100%** | **2541** | **100%** |
| N2BDM [65] | 119 | 98% | 22 | 105% | 2618 | 103% |
| Split_4_3 BDM [48] | 173 | 143% | 21 | 100% | 3633 | 143% |
| Bhatt BDM [47] | 175 | 144% | 22 | 105% | 3850 | 151% |
| Sree BDM [49] | 187 | 154% | 37 | 176% | 6919 | 272% |

other designs except N2BDM scheme.

As mentioned earlier, the decimal partial products obtained from BDMs are compressed using binary CSA tree, multi-operand converters and decimal adders. The main focus at PPR stage is the efficient design of multi-operand binary to decimal (MBD) converter. The PPR stage of present work is compared with that of Dadda [51] which provides an efficient implementation in ‘digit-by-digit’ multipliers. The main difference between the Dadda and proposed hybrid PPR schemes lies in MBD converters, which were realized using ND cells in Dadda scheme, while hybrid (ND and FBD) cells are used in the present scheme. Hence, to prove the efficacy of proposed hybrid MBD design, exhaustive comparisons are done at the multi-operand binary to decimal converter stage.

The unit gate area and delay statistics of Dadda, FBD and proposed hybrid MBD converters for various column (C) sizes in a 16* 16 multiplier illustrated in Fig.6.6 are shown in Table 6.2.

It can be noted that the number of cells (Dadda and/or FBD) required in each of the MBD designs depends on the number of bits as well as the binary number. Hence for the same ‘*nob*’ the number of cells required varies. From the results obtained it can be concluded that MBD converters formed using FBD cells alone have the speed advantage with, however, area overhead while employing Nicoud (ND) cells alone results in area efficient design with increased latency. On the other hand, MBD converters obtained from the hybrid converter algorithm (discussed in Section 6.3.2) have lesser delay compared to Dadda converters with marginal increase in area. Further, hybrid MBD converters are more area efficient compared to FBD based, that

Table 6.2: Performance comparison of different multi-operand converter designs using Dadda and proposed cells for different column size

| | | Dadda [51] | | | FBD | | | Hybrid | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Column Size (C) | No. of bits (nob) | No. of ND Cells [50] | Area | Delay | No. of FBD Cells | Area | Delay | No. of ND Cells [50] | No. of FBD Cells | Area | Delay |
| 31 | 9 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 29 | 9 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 27 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 25 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 23 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 21 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 19 | 8 | 6 | 90 | 24 | 3 | 138 | 18 | 2 | 2 | 122 | 20 |
| 17 | 8 | 6 | 90 | 24 | 3 | 138 | 18 | 2 | 2 | 122 | 20 |
| 15 | 8 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 13 | 7 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 11 | 7 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 9 | 7 | 4 | 60 | 16 | 2 | 92 | 12 | 0 | 2 | 92 | 12 |
| 7 | 6 | 3 | 45 | 12 | 2 | 92 | 12 | 1 | 1 | 61 | 10 |
| 5 | 6 | 3 | 45 | 12 | 2 | 92 | 12 | 1 | 1 | 61 | 10 |
| 3 | 5 | 2 | 30 | 8 | 1 | 46 | 6 | 2 | 0 | 30 | 8 |

too without any compromise on latency.

The total area and delay statistics of hybrid and Dadda MBD schemes for a 16*16 digit by digit multiplier are shown in Table 6.3. The total area of MBD converter is decided by the number of partial product columns while the critical path is decided by the largest column size. It can be observed from Table 6.3 that hybrid MBD design is 17% faster compared to Dadda however with 5% overhead in area.

Table 6.3: Performance comparison of Dadda and Hybrid multi-operand converter in 16*16 multiplier

| Design | Area | % change | Delay | % change |
|---|---|---|---|---|
| **Proposed Hybrid MBD Converter** | **14138** | **100%** | **24** | **100%** |
| Dadda MBD Converter [51] | 13490 | 95% | 28 | 117% |

### 6.4.2 Synthesis based Comparison

To obtain more exact comparisons, Verilog-HDL models of various multiplier designs are synthesized across different technology nodes. For a fair comparison, all the existing and proposed multiplier schemes based on ‘digit-by-digit’ algorithm are extended to perform 16-digit parallel decimal multiplication, modeled using Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1. The performance evaluation of each design has been carried out at 180nm,130nm, 90nm, 65nm and 28nm process technology nodes using TSMC library. Low power standard libraries of 13 track for 180nm/130nm, 11 track for 90nm/65nm and 9 track for 28nm provided by foundry are adopted. Synopsys 'Prime time' tool is used for delay calculation for all the BDM topologies on gate level net list with back annotated RC values.

In this section, initially the synthesized area and delay statistics of various multiplier schemes at partial product generation (PPG) stage are presented. Next, the area-delay figures of hybrid converters and partial product reduction (PPR) stage are detailed. Towards the end, a comprehensive performance analysis of $16 * 16$ 'digit-by-digit' multiplier based approaches is presented and discussed.

#### 6.4.2.1 Partial Product Generation (PPG)

An area comparison of six BCD digit multipliers (BDMs) in 'digit-by-digit' scheme, including that of two new schemes (depicted using solid lines), is provided in Fig.6.9. The BDM cells considered are HPBDM, LABDM, N2BDM, split_4_3 BDM, Bhatt BDM and Sree BDM. It can be observed from Fig.6.9 that the proposed LABDM designs performs well at all technology nodes compared to the existing schemes and also conform to the gate level analysis carried out in Section 6.4.1. The area consumed by LABDM is least in comparison to designs considered across various technology nodes, which is due to the LABD converter cells designed as a part of this work except N2BDM reported in [65].

From the graphs shown in Fig.6.10, it can be observed that the proposed HPBDM is the fastest among all the BDM cells compared. The HPBDM achieves a 15% reduction in delay across various technology nodes compared to other implementations. Further, it can be noted that proposed LABDM, though optimized for area, performs better compared to all existing BDM cells.

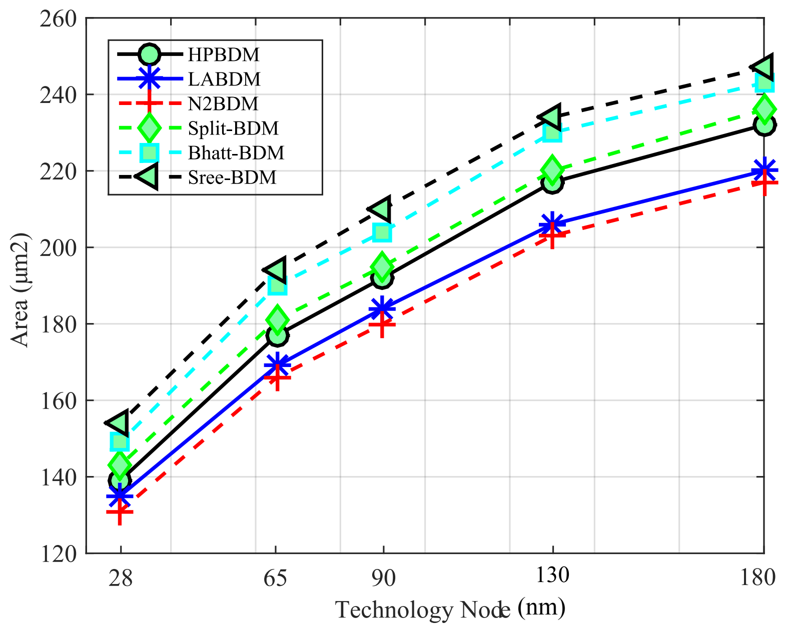


Figure 6.9: A Comparison of area consumption by various BDM at different technology nodes

#### 6.4.2.2 Partial Product Reduction (PPR)

The area and delay statistics across various technology nodes in case of Dadda [51] and proposed hybrid MBD converters are shown in Table 6.4. It can be noted that hybrid MBD converters perform better in terms of delay compared to Dadda MBD converters with only a marginal increase in area.

Table 6.4: A Comparison of Dadda and proposed hybrid MBD converters for area and delay performance across various technology nodes

| Technology (nm) | Dadda MBD Converter | | | | Hybrid MBD Converter | | | |
|---|---|---|---|---|---|---|---|---|
| | Area ($\mu m^2$) | Percent | Delay (ps) | Percent | Area ($\mu m^2$) | Percent | Delay (ps) | Percent |
| 180 | 1268 | 98% | 175 | 113% | 1295 | 100% | 155 | 100% |
| 130 | 1214 | 97.70% | 161 | 116% | 1242 | 100% | 138 | 100% |
| 90 | 1092 | 93% | 143 | 116% | 1174 | 100% | 123 | 100% |
| 65 | 1024 | 92.70% | 127 | 117% | 1104 | 100% | 108 | 100% |
| 28 | 804 | 92% | 98 | 119% | 871 | 100% | 82 | 100% |

As pointed out earlier, PPR stage in Dadda and hybrid PPR schemes consists of binary CSA [51], MBD converters and decimal adders [64]. While, Dadda PPR scheme uses the MBD converters based on Nicoud cells, hybrid MBD converters are used in the proposed PPR scheme. Figures 6.11 and 6.12 provide a comparison of the area and latencies of these two approaches respectively at various technology nodes. It can be seen from these figures that the

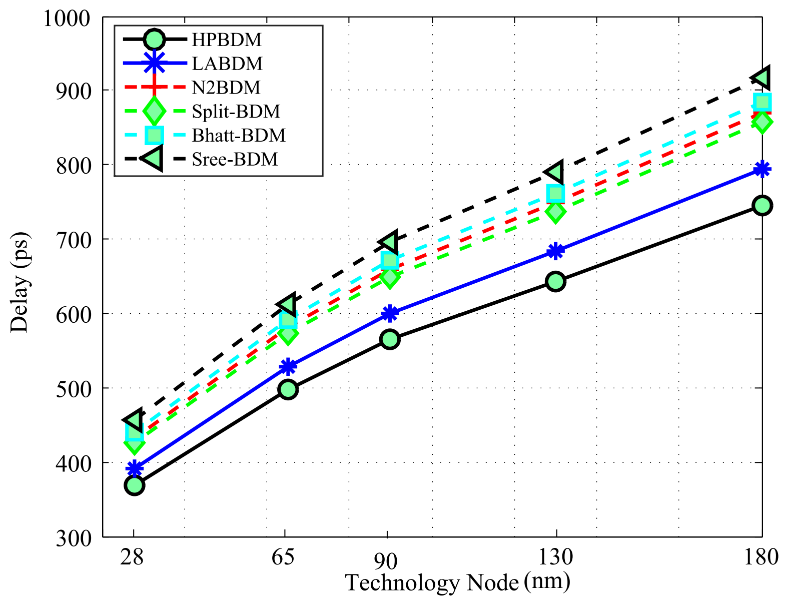


Figure 6.10: A Comparison of delay in various BDM at different technology nodes

proposed scheme results in a 11% improvement in delay with only a small increase in area. Similar pattern of behavior can be observed across all technology nodes.

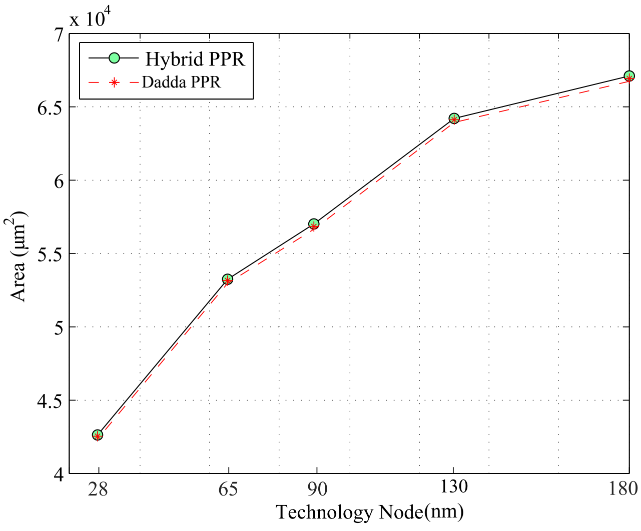


Figure 6.11: A Comparison of area consumption at PPR level in 16*16 ‘digit-by-digit’ multiplier at different technology nodes

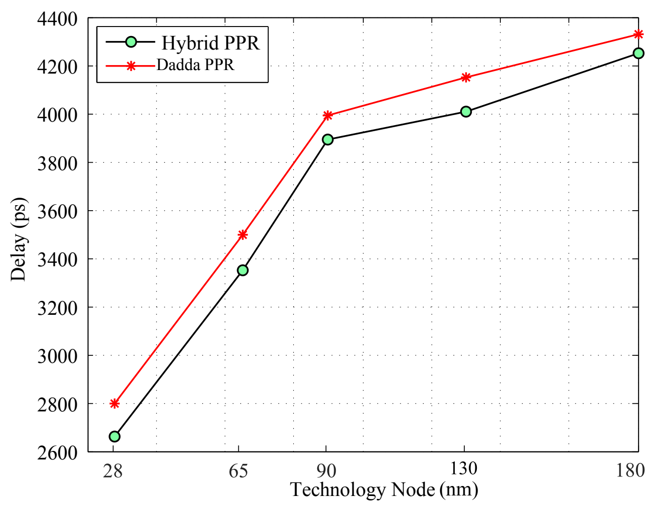


Figure 6.12: A Comparison of delay at PPR level in 16*16 ‘digit-by-digit’ multiplier at different technology nodes

#### 6.4.2.3 Synthesis Results of 16*16 ‘Digit-by-Digit’ Multiplier

The area and delay of different multiplier schemes based on ‘digit-by-digit’multiplication algorithm have been investigated at various technology nodes 180nm,130nm, 90nm, 65nm and 28nm using TSMC library and are presented in Table 6.5. The PPG is carried out using HPBDM, LABDM, N2BDM, split_4_3 BDM, Bhatt BDM and Sree BDM schemes. In PPR stage, while HPBDM and LABDM schemes use the proposed hybrid PPR technique, rest of the BDM schemes use Dadda PPR [51] approach. The proposed schemes have been found to be functional across all the technology nodes. A detailed comparison of these multipliers in terms of area, delay and area-delay product is provided in Table 6.5 and Figs.6.13-6.15. It is evident that the HPBDM with Hybrid PPR performs better in terms of delay (10 to 29%) and area-delay product (4 to 38%) while LABDM with Hybrid PPR achieves an improvement of 9 to 28% in delay and 5 to 39% in area-delay product across various technology nodes. The improvements in delay and area-delay product are bound to increase further in multipliers involving larger operand width. This is due to the efficient PPBD converters and multi-operand converter realized using hybrid converter algorithm.

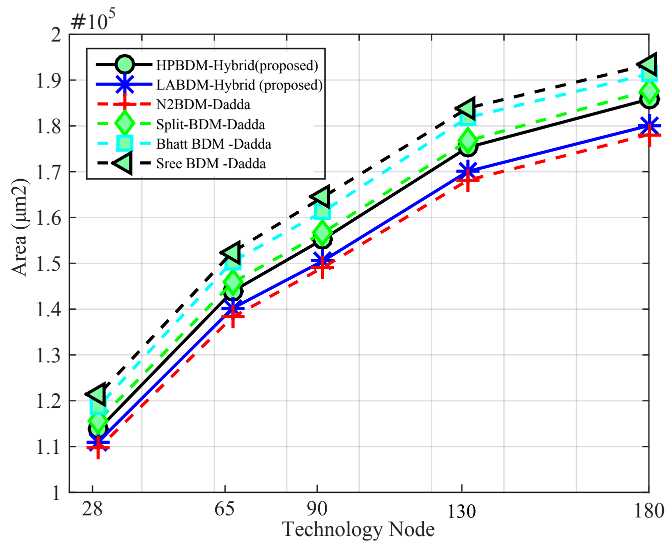


Figure 6.13: A Comparison of area consumption by various multiplier at different technology nodes

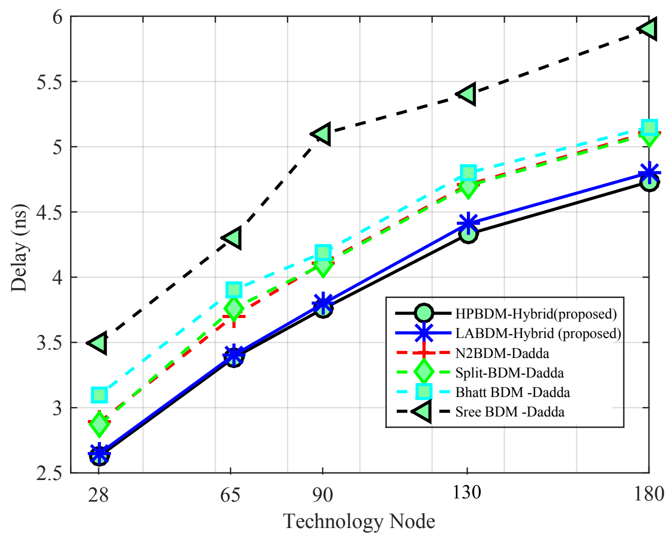


Figure 6.14: A Comparison of delay of various multiplier at different technology nodes

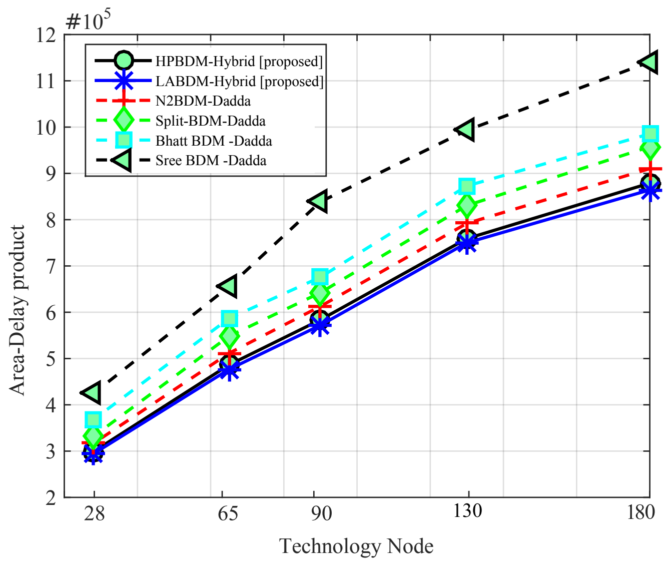


Figure 6.15: A Comparison of area-delay product of various multiplier at different technology nodes

## 6.5 Conclusions

Digital multipliers (binary/decimal) have partial product generation and reduction stages and any performance improvement in these stages will contribute to overall performance of the multiplier. In this work, new schemes have been proposed to improve the partial product generation and reduction stages in decimal multipliers. Decimal partial products are generated in parallel using fast and area efficient BCD digit multipliers and their reduction is achieved using hybrid multi-operand binary to decimal converters. Also, a new hybrid algorithm to design a multi-operand binary to decimal converter based on area and delay requirements has been proposed. In contrast to most of the previous implementations, which propose changes either in partial product generation or reduction, this work proposes modifications at both partial product generation and reduction stages resulting in an improved performance. Results obtained for a 16*16 'digit-by-digit' multiplier clearly show that a performance improvement, in terms of delay of upto 8 to 24%, and an area-delay product of upto 4 to 32%.

Table 6.5: Area and Delay comparison in 16*16 ‘Digit-by-Digit’ multipliers

| Technology Node (nm) | Scheme | Area ($\mu m^2$) | % | Delay(ns) | % | Product (µm2- ns) | % |
|---|---|---|---|---|---|---|---|
| 180 | **HPBDM-Hybrid** | 185806 | 103% | 4.73 | 98% | 878862 | 101% |
| | **LABDM-Hybrid** | 180065 | 100% | 4.8 | 100% | 864312 | 100% |
| | N2 BDM - Dadda | 178126 | 99% | 5.11 | 106% | 910224 | 105% |
| | Split_4_3BDM - Dadda | 187745 | 104% | 5.1 | 106% | 957500 | 110% |
| | Bhatt BDM - Dadda | 191329 | 106% | 5.15 | 107% | 985344 | 114% |
| | Sree BDM - Dadda | 193377 | 107% | 5.9 | 122% | 1140924 | 132% |
| 130 | **HPBDM-Hybrid** | 175306 | 103% | 4.33 | 98% | 759075 | 101% |
| | **LABDM-Hybrid** | 170066 | 100% | 4.41 | 100% | 749991 | 100% |
| | N2 BDM - Dadda | 168138 | 99% | 4.7 | 106% | 790249 | 105% |
| | Split_4_3BDM - Dadda | 176722 | 104% | 4.73 | 107% | 835895 | 111% |
| | Bhatt BDM - Dadda | 181842 | 107% | 4.8 | 108% | 872842 | 116% |
| | Sree BDM - Dadda | 183890 | 108% | 5.4 | 122% | 993006 | 132% |
| 90 | **HPBDM-Hybrid** | 155240 | 103% | 3.76 | 98% | 583702 | 102% |
| | **LABDM-Hybrid** | 150515 | 100% | 3.8 | 100% | 571957 | 100% |
| | N2 BDM - Dadda | 149096 | 99% | 4.11 | 108% | 612785 | 107% |
| | Split_4_3BDM - Dadda | 156659 | 104% | 4.1 | 108% | 642302 | 112% |
| | Bhatt BDM - Dadda | 161267 | 107% | 4.19 | 110% | 675709 | 118% |
| | Sree BDM - Dadda | 166456 | 110% | 4.7 | 123% | 782343 | 136% |
| 65 | **HPBDM-Hybrid** | 143151 | 102% | 3.38 | 99% | 483850 | 102% |
| | **LABDM-Hybrid** | 140157 | 100% | 3.4 | 100% | 476534 | 100% |
| | N2 BDM - Dadda | 136219 | 97% | 3.7 | 109% | 504010 | 106% |
| | Split_4_3BDM - Dadda | 145789 | 104% | 3.76 | 110% | 548167 | 115% |
| | Bhatt BDM - Dadda | 150397 | 107% | 3.9 | 114% | 586548 | 123% |
| | Sree BDM - Dadda | 152445 | 109% | 4.3 | 126% | 655514 | 137% |
| 28 | **HPBDM-Hybrid** | 113718 | 102% | 2.63 | 99% | 299078 | 101% |
| | **LABDM-Hybrid** | 111067 | 100% | 2.65 | 100% | 294328 | 100% |
| | N2 BDM - Dadda | 107922 | 97% | 2.89 | 109% | 311895 | 105% |
| | Split_4_3BDM - Dadda | 115675 | 104% | 2.87 | 108% | 331987 | 112% |
| | Bhatt BDM - Dadda | 118747 | 106% | 3.1 | 117% | 368116 | 125% |
| | Sree BDM - Dadda | 121307 | 109% | 3.4 | 128% | 412444 | 139% |

# Chapter 7

# Conclusions and Future Work

This thesis focused on designing new multiplier architectures in binary, logarithmic and BCD number systems customized for different requirements (accuracy, speed, area and power) to meet the diverse needs of practical applications. The architectures included that of a two-dimensional binary bypass multiplier, a fixed-width binary multiplier, an iterative logarithmic multiplier, and a ‘digit-by-digit’ BCD multiplier. Various schemes developed to improve these architectures have been detailed, discussed and superiority of their performance has been demonstrated.

The first contribution of this thesis is the development of a reconfigurable two-dimensional bypass multiplier based on dynamic bypassing of partial products. The bypass elements incorporated into the multiplier reduce the power consumption by eliminating redundant signal transitions. Further, the reconfigurable multiplier offers a good trade-off between area, delay and power dissipation by using the same multiplier for performing one $N$ or two $N/2$ multiplications.

The second contribution of this thesis is the development of a novel fixed-width binary multiplier with a target to deploy in error resilient applications where the focus is less on accuracy and more in terms of improved hardware performance. Such units result in area savings while also resulting in reduced power consumption. Further, to quantify the benefits achieved, the performance of this multiplier has been validated using the image sharpening algorithm applied on image processing benchmarks such as Lena, Cameraman, and Pirate.

The third contribution of this thesis is the development of a novel binary logarithmic multiplier with improved precision. The hardware implementation of this multiplier shows that it also has an improved performance in terms of parameters area and delay compared to other designs existing in the literature.

The fourth contribution of this thesis is the development of a generalized design approach and architectural framework for decimal multiplication. Decimal partial products have been generated in parallel using fast and area efficient BCD digit multipliers and their reduction is achieved using new hybrid multi-operand binary to decimal converters. The resulting multiplier has been shown to perform better than the existing BCD multipliers in terms of delay and area-delay product.

## 7.1 Future Work

The binary multiplier units designed and implemented in this work targeted at 180 nm technology and hence dynamic power dominates the overall power consumption. As process technologies shrink, leakage power dominates the overall power dissipation. Hence, it would be interesting to understand the performance of the binary designs proposed in this work at lower technology nodes.

Approximate computing offers potential benefits in terms of area, power and performance. However, its impact on applications is difficult to measure. Researchers and practitioners alike need tools to automate the process of carrying out approximate computing. Hence, developing new approaches that are capable of generating and synthesizing circuits with reasonable error tolerance and significantly less area consumption and power dissipation is an immediate need.

# Bibliography

[1] B. Parhami, *Computer arithmetic*, vol. 20. Oxford university press, 1999.

[2] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[3] E. L. Braun, *Digital computer design: logic, circuitry, and synthesis*. Academic Press, 2014.

[4] P. Behrooz, "Computer arithmetic: Algorithms and hardware designs," *Oxford University Press*, vol. 19, pp. 512583–512585, 2000.

[5] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.

[6] R. Zimmermann, *Binary adder architectures for cell-based VLSI and their synthesis*. Citeseer, 1998.

[7] W. N. HE *et al.*, *Cmos Vlsi Design: A Circuits And Systems Perspective, 3/E*. Pearson Education India, 2006.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE transactions on computers*, vol. 100, no. 8, pp. 786–793, 1973.

[9] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 831–838, 1980.

[10] J.-n. Ohban, V. G. Moshnyaga, and K. Inoue, "Multiplier energy reduction through bypassing of partial products," in *Circuits and Systems, 2002. APCCAS'02. 2002 Asia-Pacific Conference on*, vol. 2, pp. 13–17, IEEE, 2002.

[11] M.-C. Wen, S.-J. Wang, and Y.-N. Lin, "Low power parallel multiplier with column bypassing," in *2005 IEEE International Symposium on Circuits and Systems*, pp. 1638–1641, IEEE, 2005.

[12] G.-N. Sung, Y.-J. Ciou, and C.-C. Wang, "A power-aware 2-dimensional bypassing multiplier using cell-based design flow," in *2008 IEEE International Symposium on Circuits and Systems*, pp. 3338–3341, IEEE, 2008.

[13] M. Själander, M. Draždžiulis, P. Larsson-Edefors, and H. Eriksson, "A low-leakage twin-precision multiplier using reconfigurable power gating," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 1654–1657, IEEE, 2005.

[14] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin-precision multiplier," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 30–33, IEEE, 2004.

[15] S. Hong, T. Roh, and H.-J. Yoo, "A 145$\mu$w 8$\times$ 8 parallel multiplier based on optimized bypassing architecture," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 1175–1178, IEEE, 2011.

[16] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, p. 595, 1963.

[17] M. J. Schulte, J. E. Stine, and J. G. Jansen, "Reduced power dissipation through truncated multiplication," in *Low-Power Design, 1999. Proceedings. IEEE Alessandro Volta Memorial Workshop on*, pp. 61–69, IEEE, 1999.

[18] K. Biswas, P. Mokrian, H. Wu, and M. Ahmadi, "Truncation schemes for recursive multipliers," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, pp. 1177–1180, IEEE, 2005.

[19] A. N. Danysh and E. E. Swartzlander Jr, "A recursive fast multiplier," in *Signals, Systems &amp; Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference on*, vol. 1, pp. 197–201, IEEE, 1998.

[20] K. Biswas, H. Wu, and M. Ahmadi, "Fixed-width multi-level recursive multipliers," in *Signals, Systems and Computers, 2006. ACSSC'06. Fortieth Asilomar Conference on*, pp. 935–938, IEEE, 2006.

[21] E. E. Swartzlander Jr, "Truncated multiplication with approximate rounding," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, vol. 2, pp. 1480–1483, IEEE, 1999.

[22] E. J. King and E. E. Swartzlander Jr, "Data-dependent truncation scheme for parallel multipliers," in *Signals, Systems &amp; Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 2, pp. 1178–1182, IEEE, 1997.

[23] Y. Lim, "Single-precision multiplier with reduced circuit complexity for signal processing applications," *Computers, IEEE Transactions on*, vol. 41, no. 10, pp. 1333–1336, 1992.

[24] S. S. Kidambi, F. El-Guibaly, and A. Antoniou, "Area-efficient multipliers for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 2, pp. 90–95, 1996.

[25] J. M. Jou, S. R. Kuang, and R. Der Chen, "Design of low-error fixed-width multipliers for dsp applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 6, pp. 836–842, 1999.

[26] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[27] M. Sullivan and E. Swartzlander, "Truncated error correction for flexible approximate multiplication," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, pp. 355–359, Nov 2012.

[28] K. Abed and R. Siferd, "Cmos vlsi implementation of a low-power logarithmic converter," *Computers, IEEE Transactions on*, vol. 52, pp. 1421–1433, Nov 2003.

[29] K. Abed and R. Siferd, "Vlsi implementation of a low-power antilogarithmic converter," *Computers, IEEE Transactions on*, vol. 52, pp. 1221–1228, Sept 2003.

[30] M. Combet, H. Van Zonneveld, and L. Verbeek, "Computation of the base two logarithm of binary numbers," *Electronic Computers, IEEE Transactions on*, vol. EC-14, pp. 863–867, Dec 1965.

[31] E. L. Hall, D. Lynch, and I. Dwyer, S.J., "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *Computers, IEEE Transactions on*, vol. C-19, pp. 97–105, Feb 1970.

[32] S. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, "A fast, low-power logarithm approximation with cmos vlsi implementation," in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, vol. 1, pp. 388–391 vol. 1, 1999.

[33] V. Mahalingam and N. Ranganathan, "Improving accuracy in mitchell's logarithmic multiplication using operand decomposition," *Computers, IEEE Transactions on*, vol. 55, pp. 1523–1535, Dec 2006.

[34] T. Brubaker and J. Becker, "Multiplication using logarithms implemented with read-only memory," *Computers, IEEE Transactions on*, vol. C-24, pp. 761–765, Aug 1975.

[35] D. Mclaren, "Improved mitchell-based logarithmic multiplier for low-power dsp applications," in *SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip]*, pp. 53–56, Sept 2003.

[36] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *Electronic Computers, IRE Transactions on*, vol. EC-11, pp. 512–517, Aug 1962.

[37] M. Cowlishaw, "Decimal floating-point: algorism for computers," in *Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on*, pp. 104–111, June 2003.

[38] S. Shankland, “IbmŠs power6 gets help with math, multimedia.,” 2006.

[39] C. Webb, “Ibm z10: The next-generation mainframe microprocessor,” *Micro, IEEE*, vol. 28, pp. 19–29, March 2008.

[40] L. Dadda and A. Nannarelli, “A variant of a radix-10 combinational multiplier,” in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 3370–3373, May 2008.

[41] L. Han and S.-B. Ko, “High-speed parallel decimal multiplication with redundant internal encodings,” *Computers, IEEE Transactions on*, vol. 62, pp. 956–968, May 2013.

[42] G. Jaberipur and A. Kaivani, “Improving the speed of parallel decimal multiplication,” *Computers, IEEE Transactions on*, vol. 58, pp. 1539–1552, Nov 2009.

[43] A. Vazquez, E. Antelo, and P. Montuschi, “Improved design of high-performance parallel decimal multipliers,” *Computers, IEEE Transactions on*, vol. 59, pp. 679–693, May 2010.

[44] G. Jaberipur and A. Kaivani, “Binary-coded decimal digit multipliers,” *Computers Digital Techniques, IET*, vol. 1, pp. 377–381, July 2007.

[45] R. James, T. Shahana, K. Jacob, and S. Sasi, “Decimal multiplication using compact bcd multiplier,” in *Electronic Design, 2008. ICED 2008. International Conference on*, pp. 1–6, Dec 2008.

[46] S. Gorgin, G. Jaberipur, and B. Parhami, “Design and evaluation of decimal array multipliers,” in *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pp. 1782–1786, Nov 2009.

[47] J. Bhattacharya, A. Gupta, and A. Singh, “A high performance binary to bcd converter for decimal multiplication,” in *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pp. 315–318, April 2010.

[48] O. Al-Khaleel, Z. Al-QudahJ, M. Al-Khaleel, C. A. Papachristou, and F. Wolff, “Fast and compact binary-to-bcd conversion circuits for decimal multiplication,” in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pp. 226–231, Oct 2011.

[49] S. Veeramachaneni and M. Srinivas, “Novel high-speed architecture for 32-bit binary coded decimal (bcd) multiplier,” in *Communications and Information Technologies, 2008. ISCIT 2008. International Symposium on*, pp. 543–546, Oct 2008.

[50] J.-D. Nicoud, “Iterative arrays ror radix conversion,” *Computers, IEEE Transactions on*, vol. C-20, pp. 1479–1489, Dec 1971.

[51] L. Dadda, “Multioperand parallel decimal adder: A mixed binary and bcd approach,” *Computers, IEEE Transactions on*, vol. 56, pp. 1320–1328, Oct 2007.

[52] L. Dadda, “Some schemes for parallel multipliers,” *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, 1965.

[53] M. Sjalander and P. Larsson-Edefors, “Multiplication acceleration through twin precision,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 9, pp. 1233–1246, 2009.

[54] C.-C. Wang and G.-N. Sung, “Low-power multiplier design using a bypassing technique,” *Journal of Signal Processing Systems*, vol. 57, no. 3, pp. 331–338, 2009.

[55] J.-M. Wang, S.-C. Fang, and W.-S. Feng, “New efficient designs for xor and xnor functions on the transistor level,” *IEEE Journal of solid-state Circuits*, vol. 29, no. 7, pp. 780–786, 1994.

[56] K. V. Palem, “Energy aware computing through probabilistic switching: A study of limits,” *IEEE Trans. Computers*, vol. 54, no. 9, pp. 1123–1137, 2005.

[57] M. Hasan, T. Arslan, and J. S. Thompson, “A novel coefficient ordering based low power pipelined radix-4 fft processor for wireless lan applications,” *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 1, pp. 128–134, 2003.

[58] J.-H. Tu and L.-D. Van, “Power-efficient pipelined reconfigurable fixed-width baugh-wooley multipliers,” *IEEE transactions on computers*, vol. 58, no. 10, pp. 1346–1355, 2009.

[59] M. Sullivan and E. Swartzlander, “Truncated logarithmic approximation,” in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, pp. 191–198, April 2013.

[60] M. S. Lau, K.-V. Ling, and Y.-C. Chu, “Energy-aware probabilistic multiplier: Design and analysis,” in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES ’09, (New York, NY, USA), pp. 281–290, ACM, 2009.

[61] V. Paliouras and T. Stouraitis, “Low-power properties of the logarithmic number system,” in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, pp. 229–236, 2001.

[62] M. Erle and M. Schulte, “Decimal multiplication via carry-save addition,” in *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, pp. 348–358, June 2003.

[63] M. Erle, E. Schwarz, and M. Schulte, “Decimal multiplication with efficient partial product generation,” in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pp. 21–28, June 2005.

[64] M. S. Schmookler and A. Weinberger, “High speed decimal addition,” *IEEE Transactions on Computers*, vol. 20, no. 8, pp. 862–866, 1971.

[65] S. Gorgin, G. Jaberipur, and R. Hashemi Asl, “Efficient asic and fpga implementation of binary-coded decimal digit multipliers,” *Circuits, Systems, and Signal Processing*, vol. 33, no. 12, pp. 3883–3899, 2014.

# List of My Publications

## Publications Relevant to this Thesis

### Journals

1. **S. E. Ahmed**, Santhosh, and M. B. Srinivas, "Improved designs of a digit- by- digit decimal multiplier," *Integration, the VLSI Journal*, Elsevier, 2017 (In press).

2. **S. E. Ahmed** M. B. Srinivas, "An Improved logarithmic multiplier for media processing ," *Journal of Signal Processing Systems*, Springer, 2017, (Submitted after revising manuscript).

### Conferences

1. **S. E. Ahmed**, S. Kadam, and M. B. Srinivas, "An iterative logarithmic multiplier with improved precision," in *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, pp. 104–111, July 2016, San Francisco, USA.

2. **S. E. Ahmed**, S. Abraham, S. Veeramanchaneni, M. B. Srinivas, *et al.*, "A modified twin precision multiplier with 2d bypassing technique," in *Electronic System Design (ISED), 2012 International Symposium on*, pp. 102–106, IEEE, 2012, Kolkata, India.

3. K. V. S. Sashank and **S. E. Ahmed**, "A fixed width scheme for reconfigurable recursive multipliers," in *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)* pp. 126–130, Dec 2013, Visakhapatnam, India.

## Other Conference Publications

1. **S. E. Ahmed**, S. S. Srinivas, and M. B. Srinivas, "A hybrid energy efficient digital comparator," in *VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on*, pp. 567–568, IEEE, 2016, Kolkata, India.

2. **S. E. Ahmed**, S. Veeramanchaneni, N. M. Muthukrishnan, and M. B. Srinivas, "Reconfigurable adders for binary/bcd addition/subtraction," in *Microelectronics and Electronics (PrimeAsia), 2011 Asia Pacific Conference on Postgraduate Research in*, pp. 106–109 IEEE, 2011, Macau, China.

3. S. Ganguly, A. Mittal, **S. E. Ahmed**, and M. B. Srinivas, "A unified flagged prefix constant addition-subtraction scheme for design of area and power efficient binary floating-point and constant integer arithmetic circuits," in *Circuits and Systems (APCCAS), 2014 IEEE Asia Pacific Conference on*, pp. 69–72, IEEE, 2014, Ishigaki, Japan.

4. C. S. Varma, **S. E. Ahmed**, and M. B. Srinivas, "A decimal/binary multi-operand adder using a fast binary to decimal converter," in *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pp. 365–368, IEEE, 2014, Mumbai, India.

5. S. Ganguly, A. Mittal, and **S. E. Ahmed**, "A reconfigurable parallel prefix ling adder with modified enhanced flagged binary logic," in *Microelectronics and Electronics (PrimeAsia), 2012 Asia Pacific Conference on Postgraduate Research in*, pp. 1–6, IEEE, 2012, Hyderabad, India.

6. V. C. Kumar, P. S. Phaneendra, **S. E. Ahmed**, V. Sreehari, N. M. Muthukrishnan, and M. B. Srinivas, "Higher radix sparse-2 adders with improved grouping technique," in *TENCON 2011-2011 IEEE Region 10 Conference*, pp. 676–679, IEEE, 2011, Bali, Indonesia.

7. C. Kumar, **S. E. Ahmed**, S. Veeramachaneni, N. M. Muthukrishnan, M. B. Srinivas,

*et al.*, "A prefix based reconfigurable adder," in *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pp. 349–350, IEEE, 2011, Chennai, India.

8. V. C. Kumar, P. S. Phaneendra, **S. E. Ahmed**, V. Sreehari, N. M. Muthukrishnan, and M. B. Srinivas, "A reconfigurable inc/dec/2's complement/priority encoder circuit with improved decision block," in *Electronic System Design (ISED), 2011 International Symposium on*, pp. 100–105, IEEE, 2011, Kochi, India.

9. V. C. Kumar, P. S. Phaneendra, **S. E. Ahmed**, S. Veeramachaneni, N. M. Muthukrishnan, and M. B. Srinivas, "A unified architecture for bcd and binary adder/subtractor," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp. 426–429, IEEE, 2011, Oulu, Finland.

10. P. S. Phaneendra, C. Vudadha, **S. E. Ahmed**, V. Sreehari, N. M. Muthukrishnan, and M. B. Srinivas, "Increment/decrement/2's complement/priority encoder circuit for varying operand lengths," in *Communications and Information Technologies (ISCIT), 2011 11th International Symposium on*, pp. 472–477, IEEE, 2011, Hangzhou, China.

11. C. Vudadha, G. Makkena, M. V. S. Nayudu, P. S. Phaneendra, **S. E. Ahmed**, S. Veeramachaneni, N. M. Muthukrishnan, and M. B. Srinivas, "Low-power self reconfigurable multiplexer based decoder for adaptive resolution flash adcs," in *VLSI Design (VLSID), 2012 25th International Conference on*, pp. 280–285, IEEE, 2012, Hyderabad, India.

# Biography of the Candidate

Syed Ershad Ahmed obtained M.Tech from Vishweshwaraiah Technological University, Belgaum in Microelectronics and Control System Engineering and he is currently pursuing Ph.D in the department of Electrical Engineering, BITS, Pilani, Hyderabad Campus.

He is also working as a Lecturer in the Electrical Engineering department at BITS, Pilani, Hyderabad Campus. His research interests include low power VLSI design and Approximate Computing.

# Biography of the Supervisor

Prof.M.B.Srinivas obtained Ph.D. from IISc Bangalore in Electrical Engineering. He is currently a Professor in the Electrical Engineering department and Dean, Administration at BITS, Pilani, Hyderabad Campus.

Prof.Srinivas's research interests include High Performance Logic Design, VLSI Arithmetic, Data Converters and Reversible Computing, areas in which he has more than 150 publications, in journals as well as conferences.

Prof.Srinivas is a recipient of AIF/Stanford Medicine 'Med-Tech Innovation Award' in 2016 and Microsoft Research Digital Inclusion Award in 2006. He was an Invited Speaker at the 'Wide-Open Access' Workshop at Stockholm, Sweden in 2004, Microsoft Research Faculty Summit, Redmond in 2007 and Microsoft Research Latin-American Faculty Summit, Bina Del-Mar, Chile in 2008. He also delivered an Invited Tutorial on 'Reconfigurable ADCs' at the NASA Conference on 'Adaptive Hardware Systems' in San Diego, USA in 2011.

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

## CERTIFICATE

This is to certify that the thesis entitled High Performance Binary, Logarithmic, and BCD Multiplier Architectures and submitted by Syed Ershad Ahmed ID No 2009PHXF448H for award of Ph.D. of the Institute embodies original work done by him under my supervision.

Signature of the Supervisor

Dr. M. B. SRINIVAS

Professor

Date:

# Acknowledgements

Accomplishment of this doctoral thesis was not possible without the support of several people directly and indirectly. It gives me great pleasure to express my deep sense of gratitude to Prof. M.B.Srinivas, Professor, Department of Electrical Engineering, BITS-Pilani, Hyderabad Campus for his invaluable guidance, cooperation and keen interest during my research work. I am indeed fortunate to work under his supervision.

I am thankful to Prof. Souvik Bhattacharyya, Vice-Chancellor and Prof. G.Sundar, Director, BITS-Pilani, Hyderabad Campus for providing a research environment to enhance my research interest and commitment.

I wish to enlist my deep sense of gratitude to Prof. Sanket Goel, Head, Electrical Engineering Department, BITS-Pilani, Hyderabad Campus for providing me full support to carry out this research work. I am extremely thankful to all the faculty, PhD scholars and staff of Electrical Engineering Department.

I take this opportunity to thank my doctoral advising committee members, Prof. Prabhakar Rao, and Prof. Subhendu Kumar Sahoo for providing suggestions during the entire thesis and sparing their valuable time in providing useful comments for my draft thesis.

It's my pleasure to thank my parents and wife who stood with me during tough times with all their valuable and encouraging words.

# Abstract

Digital multipliers form an important part of digital arithmetic circuits. Important parameters that characterize these multipliers are their precision and those related to their implementations such as area, critical path delay, and power consumption.While certain applications demand high precision, others would require optimality in terms of die-area, latency of operation and power consumed. Thus the present thesis focuses on developing novel multiplier architectures (binary, logarithmic and BCD) that lead to either improved precision or result in better implementation.

The first contribution of this thesis is the development of a reconfigurable, two-dimensional (2D) bypass multiplier architecture that is based on dynamic bypassing of partial products. The bypass elements incorporated into the multiplier reduce the power consumption by eliminating redundant signal transitions.The reconfigurable architecture offers a good trade-off between area, delay and power dissipation since it uses the same multiplier for performing one $N$ or two $N/2$ multiplications.

As the modern computing systems become increasingly embedded and portable, a growing set of applications in media processing (graphics, audio, video, and image) has evolved. Many of these applications, however, possess an inherent quality of error resilience. For example, it is a known feature of image processing that a range of image resolutions/sharpness is acceptable depending on the nature of the application. Thus arithmetic units (digital multipliers in present case), that are not very precise but return an approximate value, can be utilized in such applications. Such units, it may be anticipated, may result in area savings while also resulting in reduced power consumption. The second contribution of this thesis is the development of a novel approximate binary multiplier architecture that results in improved performance in terms of area, delay and power compared to existing architectures while the trade-off in accuracy is only marginal.

Further, in recent years, logarithmic number system has been increasingly used as an alternative to the binary number system as it converts multiplication to addition resulting in simplified hardware. While logarithmic number system cannot be compared with that of binary in terms of precision, usage of it in arithmetic operations such as multiplication certainly results in reduced area and power consumption and thus is useful in applications where precise results are not required. The third contribution of this thesis is the development of an efficient logarithmic multiplier architecture that significantly reduces the area and power consumption of the hardware while sacrificing the accuracy only marginally.

Extensive analysis of the hardware requirement of both the multipliers (approximate binary and logarithmic) has been carried out, initially using unit gate modeling, and later on using the synthesis tool. Furthermore, to quantify the advantage of the proposed architectures, both have been used in an image sharpening algorithm (that employs extensive multiplication) and benchmarked against certain standard and well known image processing applications such as Lena, Cameraman and Pirate.

Finally, while binary arithmetic is all pervasive, BCD (decimal) arithmetic is preferred in applications such as financial, scientific and commercial etc. owing to its comparatively high precision. The fourth contribution of this thesis is the development of a generalized design approach and architectural framework for decimal multiplication. In this approach, unlike the existing decimal architectures, the decimal partial product generation is achieved in parallel using fast and area efficient blocks, while the partial product reduction is achieved using hybrid multi-operand binary to decimal converters. A comprehensive analysis of the synthesis results carried out on IEEE-compliant 16-digit decimal multiplier indicates the superiority of the proposed architecture over the existing ones.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| BCD | Binary Coded Decimal |
| BD | Binary to Decimal |
| BDM | Binary Coded Decimal Digit Multiplier |
| BIM | Babic Iterative Multiplier |
| BLB | Basic Logarithmic Block |
| BM | Binary Multiplier |
| CLA | Carry Look-ahead Adder |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPA | Carry Propagate Adder |
| CSA | Carry Save Adder |
| DSP | Digital Signal Processor |
| FA | Full Adder |
| FBD | Fast Binary to Decimal |
| FP | Fractional Predictor |
| HA | Half Adder |
| HDL | Hardware Description Language |
| HPBDM | High Performance Binary Coded Decimal Digit Multiplier |
| HPPPBD | High Performance Partial Product Binary to Decimal |
| ITB | Internal Tristate Buffer |
| IUS | Incisive Unified Simulator |
| KOB | Karatsuba-Ofman |

| | |
|---|---|
| LABD | Low Area Binary to Decimal |
| LABDM | Low Area Binary Coded Decimal Digit Multiplier |
| LAPPBD | Low Area Partial Product Binary to Decimal |
| LNS | Logarithmic Number System |
| LS | Logarithmic Shifter |
| LSB | Least Significant Bit |
| LSP | Least Significant Portion |
| LUT | Look-up Table |
| MA | Mitchell Approximation |
| MBD | Multi-operand Binary to Decimal |
| MFA | Modified Full Adder |
| MFP | Modified Fraction Predictor |
| MOA | Multi-operand Adder |
| MRBA | Modified Row-Bypass Adder |
| MSB | Most Significant Bit |
| MSE | Mean Square Error |
| MSP | Most Significant Portion |
| ND | Nicoud Cell |
| PE | Priority Encoder |
| PP | Partial Product |
| PPBD | Partial Product Binary to Decimal |
| PPG | Partial Product Generation |
| PPR | Partial Product Reduction |
| PSNR | Peak Signal to Noise Ratio |
| RAM | Random Access Memory |
| RCA | Ripple Carry Adder |
| RTDBC | Reconfigurable Two Dimensional Bypass Cell |
| RTL | Register Transfer Level |
| TBLB | Truncated Basic Logarithmic Block |

| | |
|---|---|
| TDBA | Two Dimensional Bypass Adder |
| TEC | Truncated Error Correction |
| VLSI | Very Large Scale Integration |

# Chapter 1

# Introduction

## 1.1 Background

One of the most common and frequently executed operations in arithmetic computations is multiplication. Significant amount of work has been carried out to improve the performance of digital multipliers over the years and the same is expected to continue in future.The criteria that are used to quantify their performance include latency, area and power consumed. Thus any improvement made in the design/architecture of multipliers should be reflected in the improvement of these parameters.

In digital static CMOS multipliers, transition activity (due to charging and discharging of the load capacitance) dominates the total energy consumption. Thus, power saving can be achieved by lowering the switching or transition activity per operation. Earlier efforts attempted to reduce the switching activity of the binary multipliers through architectural modifications such as row and/or column bypassing. In these schemes, the redundant signal switching is eliminated by disabling the full adder circuits whose partial product is zero while forwarding the output of the previous adder rows/columns to the next row/columns. However, the extra bypass logic (mostly adder) has only limited effect in reducing the power dissipation while contributing significantly to area overhead. Thus, there is a need to develop alternate bypass multiplication architectures that can address large power consumption in multipliers.

Many of the signal and image processing applications possess an inherent quality of error resilience and thus do not require absolute accuracy in computation. Further, the final output in these applications is interpreted by human senses which are not perfect. Thus approximation in place of accuracy can be exploited that can lead to a significant improvement in area, power and performance. Based on this idea, several techniques have been proposed that focus on approximate rather than accurate computing. However, most of these techniques provide solutions that are based on trial and error and thus the accuracy achieved tends to be lower. Thus, realizing efficient multiplier units (binary and logarithmic) for approximate computing in a systematic way, which also have high precision would be of considerable interest.

The importance of error-free arithmetic is growing day-by-day and decimal (BCD) arithmetic circuits are making their way into application such as financial, scientific and commercial, etc. Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic, is multiplication. Decimal multiplication can be classified as serial multiplication, parallel (‘word-by-digit’) and (‘digit-by-digit’) multiplication. Decimal (BCD) ‘digit-by-digit’ multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in significant improvement in the multiplier performance. There is however a significant scope to develop more efficient architectures for ‘digit-by-digit’ multiplication.

## 1.2 Objectives of the Thesis

The objectives of this thesis are as follows:

- To improve the existing binary multiplier architecture to reduce the switching activity resulting in low power consumption.
- To design and implement truncated binary and iterative logarithmic multipliers targeted for error resilient applications
- To develop a BCD multiplier with improved performance for high speed (parallel) multiplication

## 1.3 Steps involved in carrying out present research

Figure 1.1 illustrates the steps involved in carrying out present work, summarized below:

- Modification of multiplier architectures to improve precision and/or performance for binary, logarithmic and decimal multiplication
- Modeling the architectures (fixed-width binary and logarithmic multiplier) in MATLAB
- Evaluation of the above using synthesis-independent unit gate level (hardware) modeling
- Verification using Verilog test benches and applying random stimuli to cover a wide input range
- Synthesis of binary, logarithmic and decimal multipliers using Cadence RTL compiler to obtain estimates of area, delay, and power
- Analysis of the above multipliers to evaluate and compare their performance with the existing designs

### 1.3.1 High Level Modeling

The multiplier schemes (fixed-width binary and logarithmic multiplier) have been modeled and verified in MATLAB environment. Metrics related to precision such as maximum error and average error have been computed for different multiplier schemes.The main purpose of carrying out high-level modeling is as follows:

1. It is a faster way of realizing optimized architectures/designs
2. It offers an easier and faster method to evaluate and compare different architectures/designs
3. A high-level model serves as an abstract model of the design to generate input stimulus and verify the result

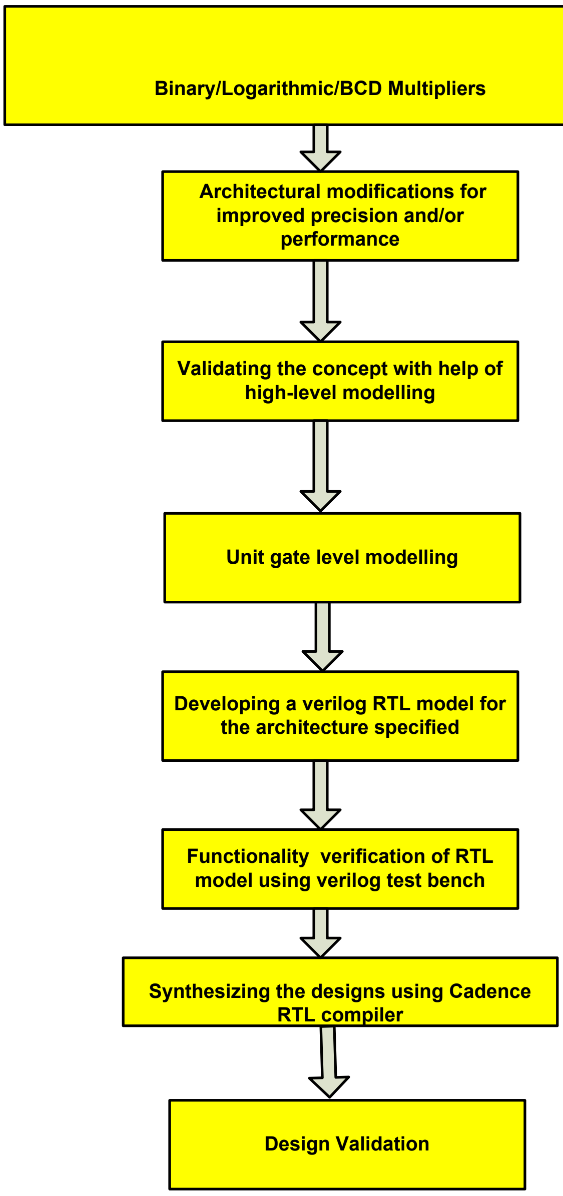

Binary/Logarithmic/BCD Multipliers
Architectural modifications for improved precision and/or performance
Validating the concept with help of high-level modelling
Unit gate level modelling
Developing a verilog RTL model for the architecture specified
Functionality verification of RTL model using verilog test bench
Synthesizing the designs using Cadence RTL compiler
Design Validation

Figure 1.1: Research Flow chart

### 1.3.2 Unit Gate Level Modeling

All the designs (fixed-width binary, logarithmic and decimal (BCD) multipliers) under consideration have been modeled using unit gate approach to obtain a rough estimate of area $(A)$ and delay $(D)$. This model is useful for high-level analysis and does not depend strongly on any one process technology, synthesis tool, or cell library. The assumptions made while performing the unit gate modeling are the following: Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for both area and delay. Further, an m-input gate is assumed to be composed of a tree of m-1 input gates while the effects of wiring, buffering and inverting costs (area and delay) are neglected.

### 1.3.3 CAD tools used in the ASIC implementation

#### 1.3.3.1 Cadence Simulator

Cadence NCSim is an RTL functional simulator that can simulate Verilog models. The functional behavior of the modules (Binary, Logarithmic and BCD multipliers) was verified in NC-Sim using Verilog test benches.

#### 1.3.3.2 Cadence RTL Compiler

Cadence RTL Compiler is a hardware synthesis tool. It maps an RTL hardware description model using a standard cell library into a gate-level net list. The output structural level thus obtained is composed of cells that exist in the standard-cell technology library. The synthesis tool accepts Verilog RTL code as an input and generates area, delay and power reports.

## 1.4 Organization of the Thesis

This thesis is organized as follows. Chapter 2 presents a review of various existing multiplier architectures relevant to this research and their realization in hardware. It also provides a detailed discussion of multipliers based on binary, logarithmic and BCD number systems. Keeping in mind the importance of arithmetic precision, chapters 3 and 6 develop and validate new techniques for improved precision of binary and decimal arithmetic circuits. Since precision is not as important as efficiency of implementation for error resilient applications, chapters 4 and 5 develop novel truncation schemes that lead to efficient implementation of binary and logarithmic multipliers. These schemes have also been compared for performance against the existing ones. Chapter 7 draws conclusions and provides recommendations for future work.

# Chapter 2

# Literature Review

## 2.1 Introduction

This chapter reviews a number of widely used multiplier architectures while focusing mainly on those that will be of concern in this thesis. The chapter is organized as follows: Preliminary information on the existing binary multiplier architectures is presented in Section 2.3 - 2.5, and fixed-width (truncated) multipliers are discussed in Section 2.6. An outline of the existing logarithmic multipliers is presented in Section 2.7 while decimal (BCD) multipliers are reviewed in Section 2.8.

## 2.2 Classification of Multiplier Architectures

Digital multipliers based on number system can be classified as (i) Binary multipliers (ii) Logarithmic multipliers and (iii) BCD multipliers. A pictorial representation of the same is given in Fig.2.1 and explained in detail in the following sections.

## 2.3 Binary Multipliers

It is well known that binary multipliers can be classified into two categories, viz., integer fixed-point and floating point. This thesis however focuses on integer fixed-point multiplier architectures only. In fact, floating-point multipliers consist of a fixed-point multiplier for the

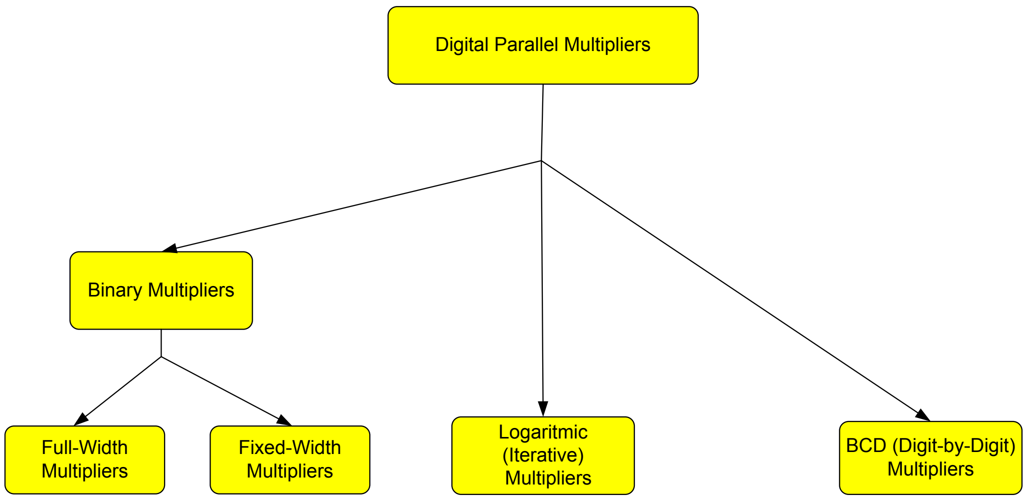

Digital Parallel Multipliers
Binary Multipliers
Full-Width Multipliers
Fixed-Width Multipliers
Logaritmic (Iterative) Multipliers
BCD (Digit-by-Digit) Multipliers

Figure 2.1: Classification of digital multiplier architectures based on number system

significant and additional circuitry to deal with the exponents and special values. Thus, techniques developed for efficient binary multiplication presented in this thesis are also applicable for floating-point multiplication.

Literature on binary computer arithmetic includes topics ranging from sequential to parallel multipliers. Today, most of the advanced digital systems include a parallel binary multiplication unit to carry-out mathematical computations. Array and Booth multipliers are a few examples of parallel multiplication in this category. As is well known, array-based multipliers [1] are ideal for very large scale integration (VLSI) and application specific integrated circuits (ASICs) due to their regular layout. On the other hand, Booth multiplier [2], although faster compared to array multipliers, has an irregular layout structure, making it not very suitable for VLSI implementations.Thus, this thesis focuses on design and validation of area and power efficient binary array multipliers. In order to provide more insight in to multiplication process, the general structure of binary multiplier is described initially and implementation of the same is illustrated using Braun array multiplier.

In general, binary multiplication involves three steps: (i) Partial product generation (PPG) (ii) Partial product reduction (PPR) and (iii) Final product computation. A typical binary multiplier accepts two binary inputs $A$ and $B$, each of $N$-bit width, as illustrated in Fig.2.2. Multiplication schemes primarily differ in the manner partial products are generated and/or accumulated. The multiplication operation can be accelerated in two ways: generating optimized number of partial products (PPs) in the first (PPG) step or accelerating their accumulation in second (PPR) step. One of the most effective ways of accumulating an array of partial products into two rows is through carry save adder structure. These two rows are eventually reduced using a final adder in the last step. A detailed explanation of PPG and PPR in an array multiplier is provided in subsequent sections.

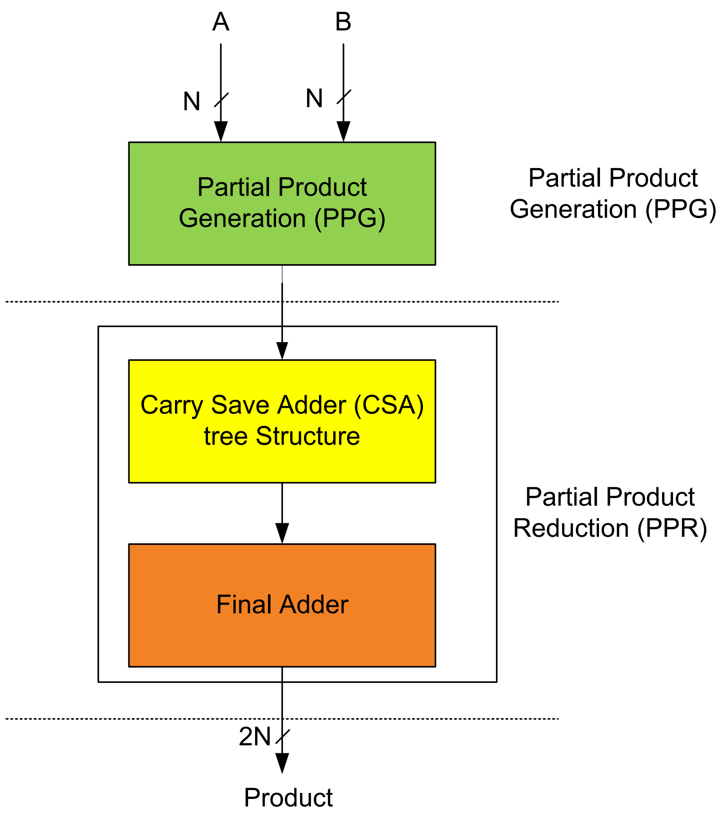

A
B
N
N
Partial Product Generation (PPG)
Partial Product Generation (PPG)
Carry Save Adder (CSA) tree Structure
Partial Product Reduction (PPR)
Final Adder
2N
Product

Figure 2.2: A general binary multiplication structure

## 2.4 Braun Multiplier

The simplest array multiplier proposed was by Braun [3], generally known as carry save multiplier, suited for unsigned operations only. The mathematical model of a N*N unsigned array multiplication is given below. Assume $A$ and $B$ to be two $N$-bit unsigned numbers, where $A$ is the multiplicand and $B$ is the multiplier.

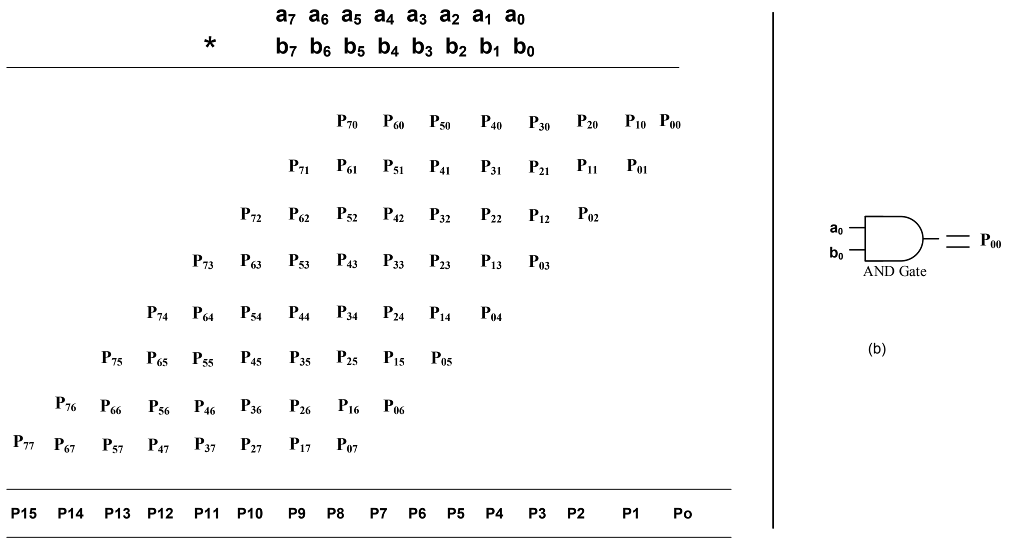
$$A = \sum_{i=0}^{n-1} a_i.2^i \tag{2.1}$$

$$B = \sum_{i=0}^{m-1} b_j.2^j \tag{2.2}$$

The product $(P)$ can be written as: $P = A * B = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} a_i b_j.2^{i+j}$

### 2.4.1 Partial Product Generation (PPG)

In general, to implement $N * N$ binary multiplier in hardware, $N^2$ partial products are required for PPG which are generated using AND gates. As an example, consider hardware implementation of 8*8 Braun multiplier illustrated in Fig.2.3. A typical multiplication of two 8-bit binary numbers results in a total of 64 PPs. Figure.2.3(a) depicts the arrangement of these partial products in a matrix form. Each of these partial products (PPs) is obtained using an AND gate as illustrated in Fig.2.3(b). Further, an alternate partial product representation of the same multiplier is shown in Figure.2.4.



a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P70 P60 P50 P40 P30 P20 P10 P00
P71 P61 P51 P41 P31 P21 P11 P01
P72 P62 P52 P42 P32 P22 P12 P02
P73 P63 P53 P43 P33 P23 P13 P03
P74 P64 P54 P44 P34 P24 P14 P04
P75 P65 P55 P45 P35 P25 P15 P05
P76 P66 P56 P46 P36 P26 P16 P06
P77 P67 P57 P47 P37 P27 P17 P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po
a0
b0
AND Gate
P00
(b)

Figure 2.3: (a) Partial product matrix representation in a 8*8 Braun multiplier (b) Partial product computed using an AND gate

### 2.4.2 Partial Product Reduction (PPR)

The PPs generated must be accumulated to form the final product. In multiplication, accumulation of PPs, also referred to as reduction of PPs, consumes most of the time taken for multiplication. The reduction of the PPs is performed using two main methods, namely, ac-

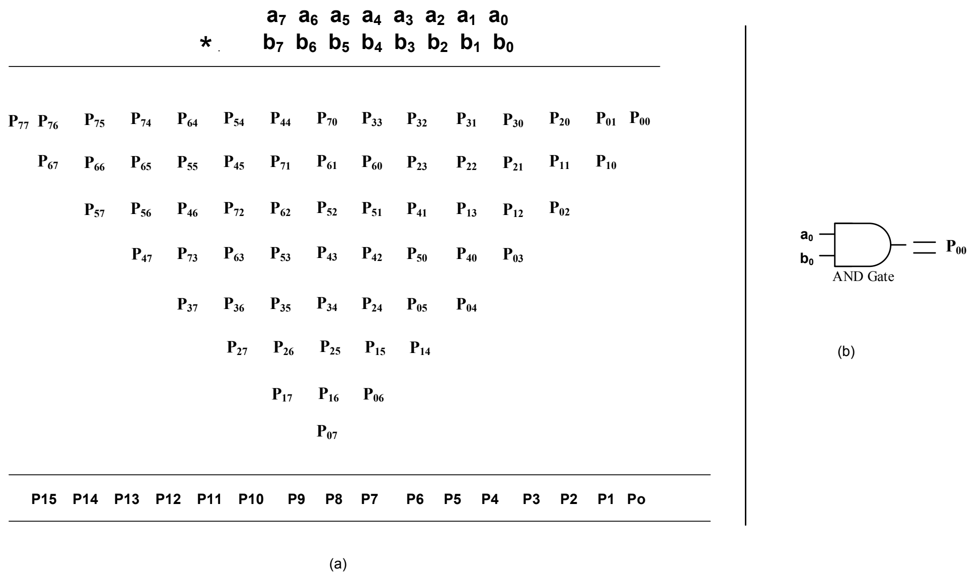

a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P77 P76 P75 P74 P64 P54 P44 P70 P33 P32 P31 P30 P20 P01 P00
P67 P66 P65 P55 P45 P71 P61 P60 P23 P22 P21 P11 P10
P57 P56 P46 P72 P62 P52 P51 P41 P13 P12 P02
P47 P73 P63 P53 P43 P42 P50 P40 P03
P37 P36 P35 P34 P24 P05 P04
P27 P26 P25 P15 P14
P17 P16 P06
P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po
(a)
a0
b0
AND Gate
P00
(b)

Figure 2.4: (a) Alternate representation of partial products in 8*8 Braun array multiplier (b) Partial product computed using an AND gate

cumulation by rows and accumulation by columns. The building modules are referred to as adders if the accumulation is by rows and conversely, if the reduction is by columns, they are referred to as counters. A simple technique in the accumulation by rows involves multiple two-operand carry propagate adders (CPAs). However, propagation of the carry using CPAs is time-consuming and thus is slow [4]. An alternate and more efficient approach is to reduce the columns by using carry-free adders, namely, carry save adders (CSAs) as discussed below.

#### 2.4.2.1 Binary Carry Save (CSA) Adders

Carry save adders are popular structures used for partial product reduction in multiplication process. The binary partial product reduction structure uses multiple levels of carry save adders (CSAs). As illustrated in Fig.2.5, each bit-slice (group of 3-bits) of a CSA is realized using a full adder. This binary full adder generates a sum-bit and a carry-bit. The carry input is propagated from the previous bit-slice to the next most significant position in the reduction tree. The PP reduction process results in two rows (sum and carry), which are eventually converted to the final sum using a two operand adder (carry propagating adder). In short, binary PPR can be implemented via CSA tree comprising of binary half adders (HAs) and full adders (FAs) as basic elements.

As is well known, a half adder accepts two operand bits ($A$ and $B$) as inputs and computes

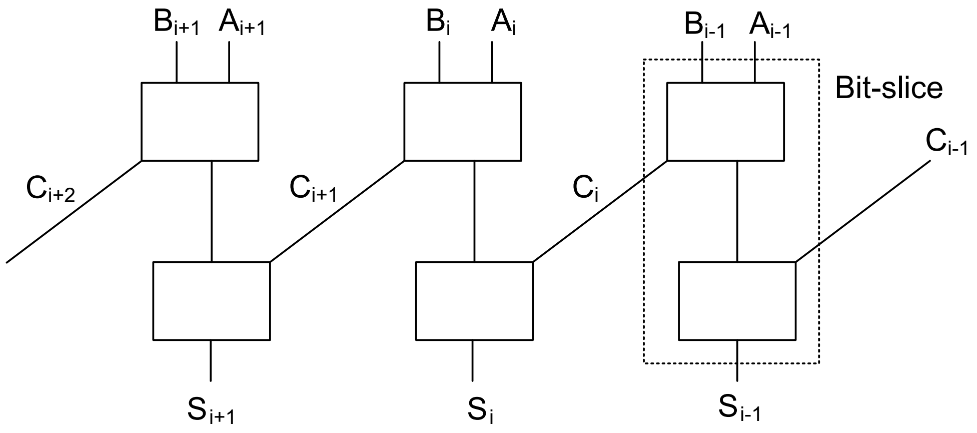


Figure 2.5: Carry- free operation using full adders

sum $(S)$ and output carry bit $(C_o)$ as outputs. The carry bit $(C_o)$ will eventually serve as input carry-in for the successive half adder. The implementation of a half adder circuit follows the Boolean equations 2.3 and 2.4 and its gate level implementation, cell notation and dot notation are shown respectively in Fig.2.6 (a-c). The notations '$\oplus$', '.' and '$+$' denote logical XOR, AND and OR gates respectively.

$$S = A \oplus B \tag{2.3}$$

$$C_0 = A.B \tag{2.4}$$

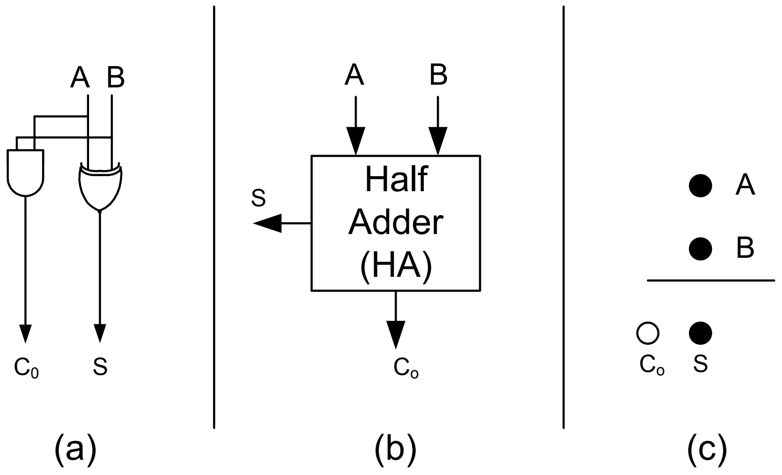


Figure 2.6: (a) Logic circuit of half adder (b) Half adder cell notation (c) Computation of Sum and Carry-out using dot notation in a half adder

Similarly, a full adder accepts three operands $(A, B$ and carry in $(C_i))$ as inputs and computes the sum $(S)$ and output carry bit $(C_o)$ as outputs. The design of a full adder circuit follows the Boolean equations 2.5 and 2.6 while the gate level implementation, cell notation and dot notation are illustrated in Fig.2.7(a-c) .

$$S = A \oplus B \oplus C_i \tag{2.5}$$

$$C_0 = (A \oplus B).C_i + AB = AB + BC_i + C_iA \tag{2.6}$$

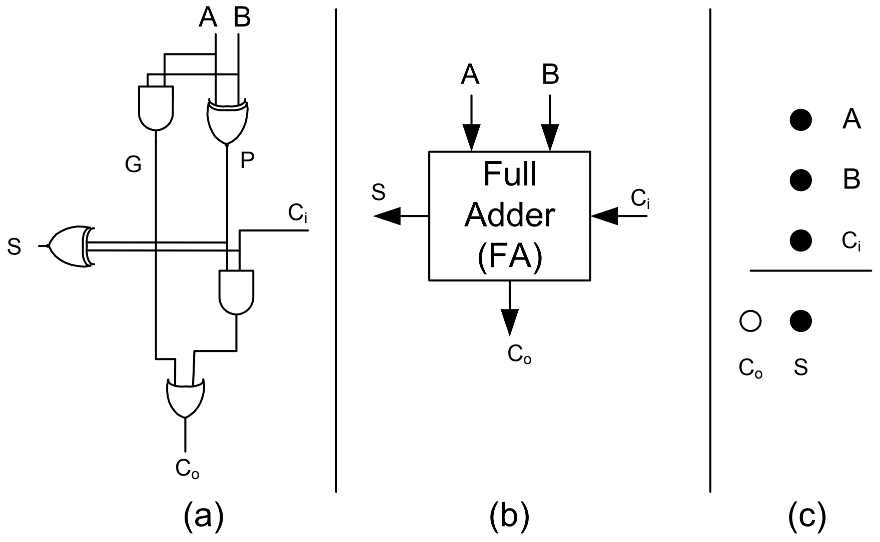


Figure 2.7: (a) Logic circuit of full adder (b) Full adder cell notation (c) Computation of Sum and Carry-out using dot notation in a full adder

A row of full-adders, represented in Fig.2.5, can be viewed as a mechanism to reduce three operands to two operands. For a CSA, each FA referred to as 3:2 counter has three dots in one column as inputs. The resulting sum output will result in a dot with the same magnitude as the inputs while carry output will result in a dot in the column to its left (one order of magnitude higher) as shown in Fig.2.7(c).

For illustration purpose, reduction of two partial product columns (c0 and c1) is shown in Fig.2.8(a) and the same is later extended to 8*8 multiplication as shown in Fig.2.9. Each column, consisting of six partial products (each denoted by solid dot(•) ), is reduced in parallel to sum (S) and carry (C). The sum is denoted by solid dot(•) with the same magnitude as the inputs while the carry, denoted with hollow dot (◦), has higher positional weight compared to input. The six partial products are reduced ( in three levels) to two rows using a tree of 3:2 and 2:2 full and half adders. These two rows of PP are eventually reduced to a binary number by using a carry propagation adder (CPA) represented with a horizontal line shown in Fig.2.8 (a) .

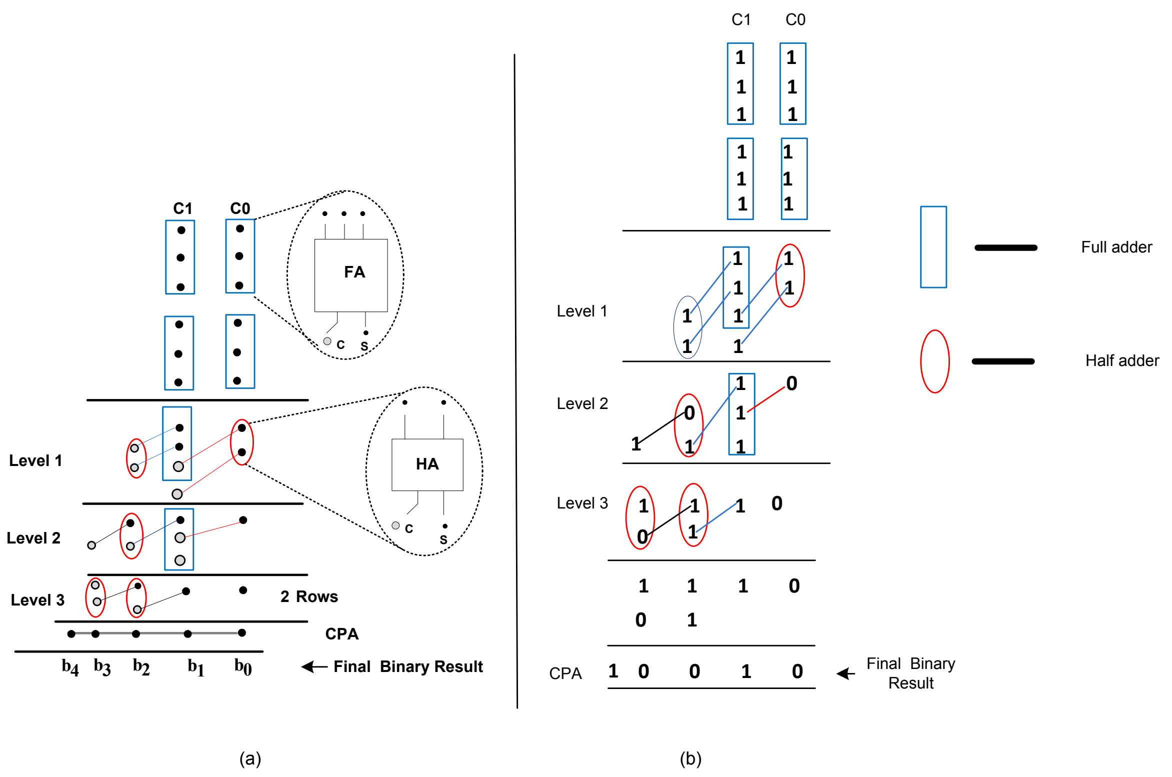


Figure 2.8: (a) Partial product reduction using CSA dot notation (b) A numerical example related to partial product reduction

The numerical example illustrated in Fig.2.8(b) describes the addition of two columns ($c0$ and $c1$ ), each consisting of six bits, where each partial product in the column is assumed to be '1'. These columns are reduced to two rows in four levels using a tree of full and half adders. These two rows of PPs are reduced to final binary result by using a carry propagation adder (CPA).

A variety of algorithms for accumulating the partial products using CSAs has been proposed [1]. The advantage of using CSAs is that they do not contribute to hardware complexity and one of the first algorithms proposed was by Wallace [5].

### 2.4.3 Wallace Reduction Tree

Wallace developed a method for reducing the columns in parallel. Figure.2.9 illustrates a Wallace-like reduction tree organization for an $8*8$-bit unsigned multiplier, presented in Section 2.4.1. As discussed earlier, a sum output ($S$) from a full or half-adder at one stage places a dot in the same column at the next stage. A carry output ($C_o$) from a full or half-adder at one stage places a dot one order of magnitude higher i.e., in the column to its left, in the next stage. As shown in Fig.2.9(d), three dots joined by a solid diagonal line indicates that these PPs are outputs of a (3,2) counter, while two dots joined by diagonal line indicates that these PPs are outputs of a (2,2) counter. Consequently, the PP matrix is accumulated to a height of two in four levels using a carry save adder (CSA) tree structure formed using full adder and half adder as shown in Fig.2.9(a). A total of four reduction levels with matrix heights of 6, 4, 3 and 2 is required to accumulate the PP matrix into two rows using Wallace technique. These two rows are reduced to a final sum using a carry propagate adder (CPA) or any fast adder mentioned below.

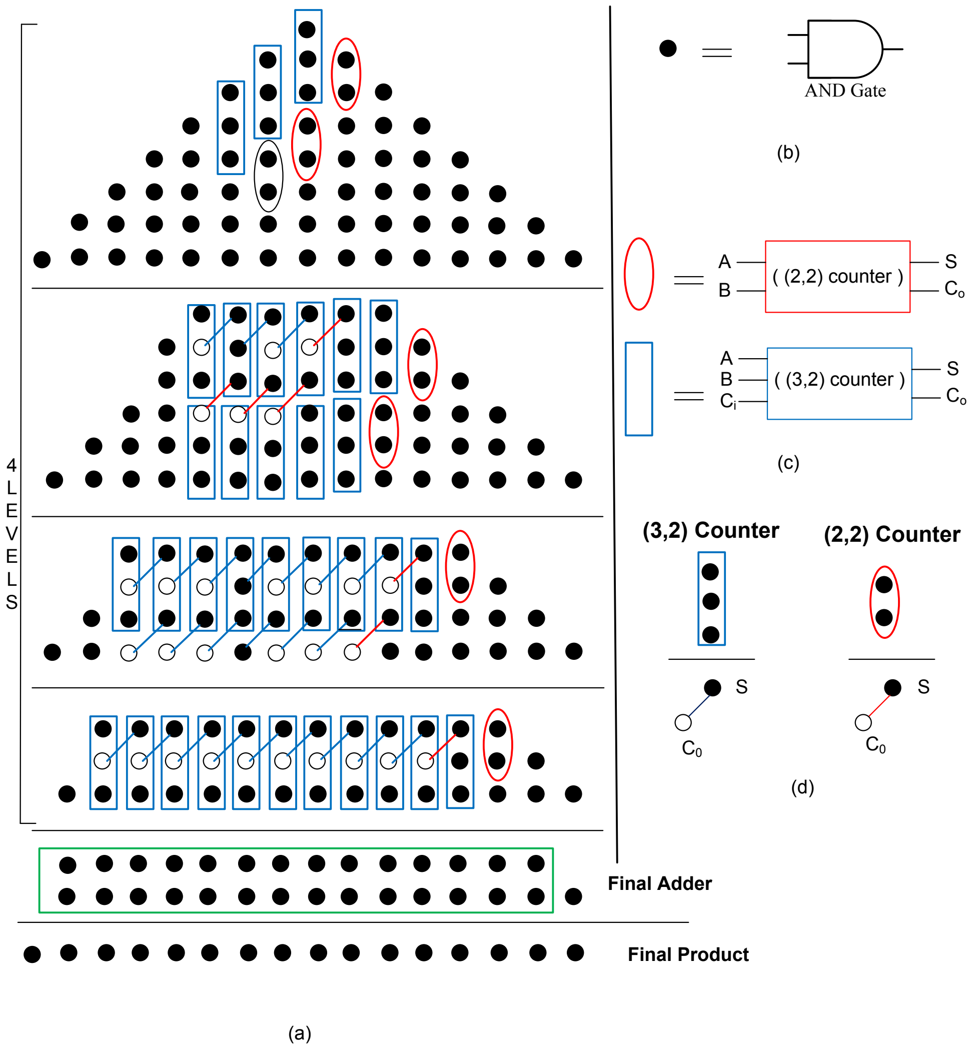


Figure 2.9: (a) Wallace tree partial product reduction structure using 3:2 and 2:2 counters (b) Partial product computed using an AND gate (c) Representation of 3:2 and 2:2 counters (d) Computation of Sum and Carry-out using dot notation in a full and half adder circuits

### 2.4.4 Final Adder

The last step in the partial product reduction process is the conversion of the redundant sum obtained from Wallace reduction tree into non-redundant representation. This step is performed using a non-redundant adder. There exist many topologies to implement the final adder namely, ripple carry, carry look-ahead, and parallel prefix (or prefix tree) [6] among many. Based on priorities (area and delay) appropriate adder topology can be selected from the available literature. For instance, a ripple carry adder has area and a delay that is proportional to the adder‘s length while prefix based adders have almost logarithmic delay but with area overhead. Thus, appropriate adder design can be chosen depending on the requirement.

#### 2.4.4.1 Ripple Carry Adder

The basic building blocks of a ripple-carry adder (RCA) are full adders. Consider two n-bit numbers, $A$ and $B$, described by equations 2.1 and 2.2. A total of n full adders are used, one for each column. The full adder in column $i$ adds the operand bits $A_i$ and $B_i$ plus the carry-in $(C_i)$, where $i = 0, 1 \ldots N-1$. The carry-out of previous stage full adder is passed down to the carry-in of the full adder in the next most significant column. The $S_i$ outputs of the $n$ full adders form the sum. Figure 2.10 illustrates a 4-bit ripple carry adder.

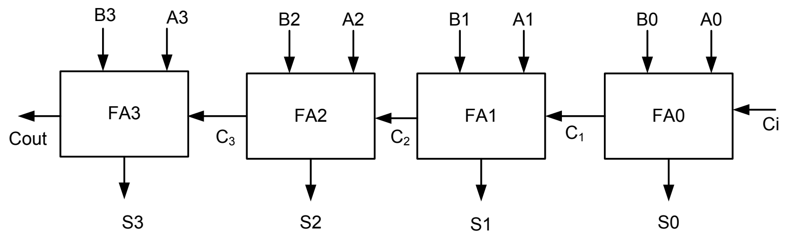


Figure 2.10: 4-bit Ripple Carry Adder

Although, ripple carry adder is simple and easy to implement, it suffers from large delay. This is because the full adder in the next stage has to wait for carry bit from the previous stage full adder (FA). By inspecting the FA shown in Fig.2.10 it can be observed that each full adder contributes to a two gate delay in the process of rippling the carry [6]. In general, critical path length of the final carry propagation adder can be deduced as follows:

$$CPA\,length = 2N - 2$$

#### 2.4.4.2 Carry Look-ahead Adder (CLA)

The carry propagation delay in a ripple carry adder increases linearly with an increase in the number of input bits. Efforts to reduce this delay has resulted in novel adder architectures and Carry Look-ahead (CLA) is one such adder which improves the speed by computing the carry signals in advance that depends on the input operands.

Based on the combination of inputs $A_i$ and $B_i$, the signals, generate $(G_i)$ and propagate$(P_i)$, determine the possibility of carry generation. Generate term determines if a carry-out would be ‘1’ independent of carry-in while propagate term determines whether carry moves to the next higher significant position. The standard carry look-ahead adder equations ($G_i$ and $P_i$) that dictate if the carry will be generated or propagated can be given as,

$$G_i = A_i.B_i \tag{2.7}$$

$$P_i = A_i \oplus B_i \tag{2.8}$$

Clearly, carry generation depends on the values of $A_i$ and $B_i$. For instance, when $A_i = B_i$ =‘1’, a carry of ‘1’ is produced at the $i^{th}$ position, while a carry of ‘0’ is generated when $A_i = B_i$ =‘0’. Conversely, carry propagation happens when $A_i \neq B_i$. Hence, when $A_i \neq B_i$ and carry-in $(C_{in})$ is ‘1’, then $C_{in}$ is said to propagate to the next position.

Accordingly, the sum and carry recurrence for the $i^{th}$ stage is as follows:

$$S_i = P_i \oplus C_i \tag{2.9}$$

$$C_{i+1} = G_i + P_i.C_i \tag{2.10}$$

Similarly, the carries in a 4-bit CLA are generated in parallel according to the following equations:

$$C_1 = g_0 + p_0c_0 \tag{2.11}$$

$$C_2 = g_1 + p_1 g_0 + p_1 p_0 c_0 \tag{2.12}$$

$$C_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \tag{2.13}$$

$$C_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \tag{2.14}$$

The logic circuit of a 4-bit CLA is illustrated in Fig.2.11.

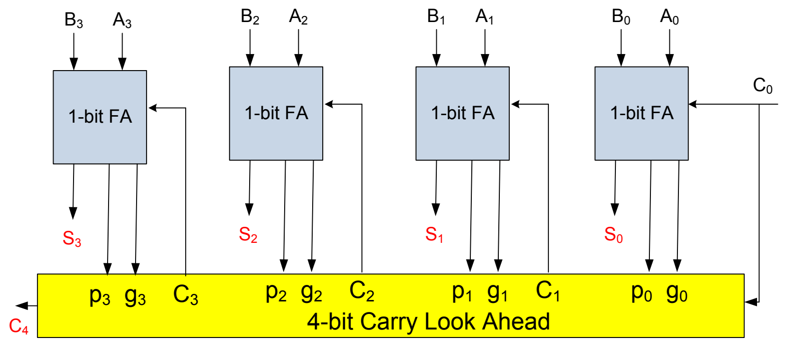


Figure 2.11: 4-bit Carry Look-ahead adder

One obvious disadvantage in CLA adder is that the carry block gets complicated for large values of N. To mitigate this, a new class of adder networks has been designed that transfers the carry through the look-ahead stage in about $log_2(N)$ stages. These networks are known as tree networks and the adder circuits that utilize these networks are called prefix-adders or tree-adders [7].

#### 2.4.4.3 Carry Look-ahead (CLA) based Parallel Prefix Adder

There are numerous ways to design the parallel prefix tree adders that offer trade-offs among parameters like the number of logic stages, the maximum fan-out of each logic gate and the wiring complexity between the stages [6] etc. Based on these parameters a wide variety of prefix tree architectures, namely, Sklansky, Brent-Kung, Kogge-Stone, Ladner-Fischer, Han-Carlson and Knowles [6] have been developed.

In general, as illustrated in Fig.2.12, there are three stages in any prefix adder that can be termed as (i) pre-computation stage (ii) prefix network stage and (iii) post-computation stage [6,8,9]. The pre-computation stage determines the generate and propagate bits as per the equations 2.7 and 2.8.

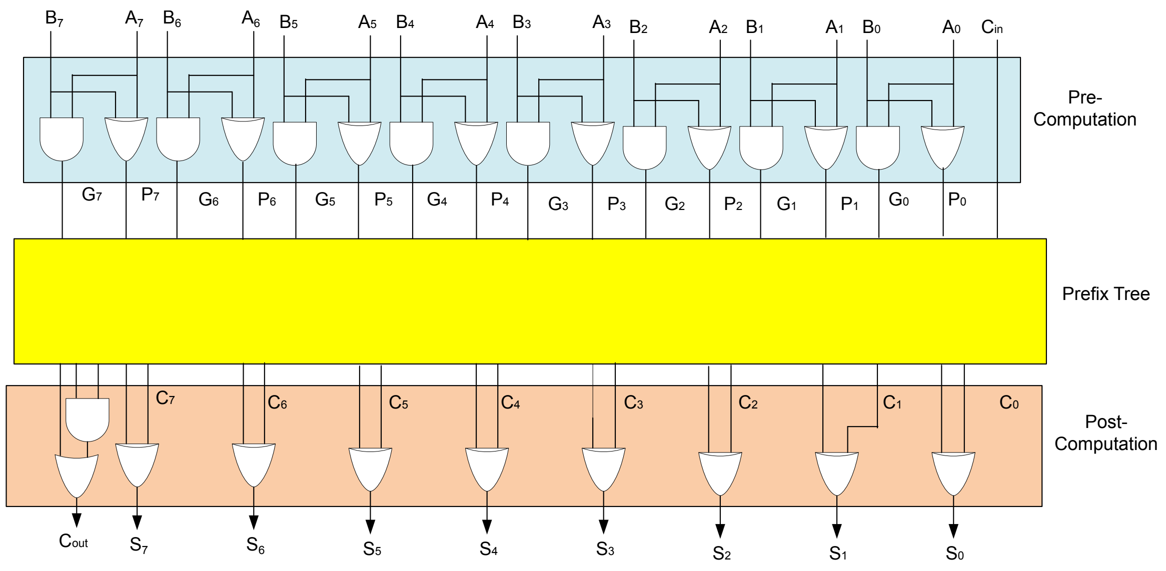


Figure 2.12: CLA based 8-bit parallel-prefix structure

The prefix network stage computes the final carries from the individual generate and propagate bits of pre-computation stage. Using associative principle, carry computation is transformed to prefix problem using the operator '∘' which associates pairs of generate and propagate as mentioned below:

$$(g,p)\circ(g',p') = (g+p.g',p.p') \tag{2.15}$$

where $g$ and $g'$ denote the generate terms and $p$ and $p'$ represent the propagate terms. Using the operator '∘' consecutive generate and propagate pairs can be grouped to generate carry as follows:

$$C_i = (g_i, p_i)\circ(g_{i-1}, p_{i-1})\circ....(g_1, p_1)(g_0, p_0) \tag{2.16}$$

The post computation stage determines the final sum from carries generated in the prefix network stage.

The graph model of prefix carry computation is obtained by representing the operator '∘' as node •, while the signal pairs $(g, p)$ are denoted as edges of a graph. Different prefix structures differ only in prefix network stage.

To illustrate a prefix structure, an 8-bit Kogge-Stone [8] prefix tree is illustrated in Fig.2.13. The dark color (•) node in the graph represents the logic module while the white color (∘) node denotes a feed through node with no logic (realized with a buffer in real hardware).

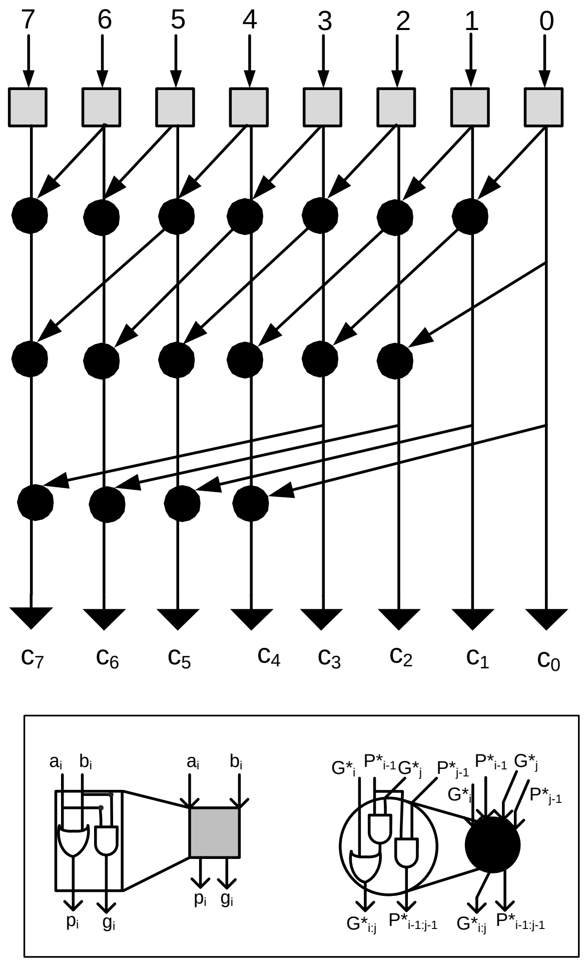


Figure 2.13: CLA based 8-bit Kogge-Stone prefix adder

## 2.5 Low Power Techniques in Binary Multiplier Design

Multipliers are logic circuits that are computationally heavy. Typically, a large number of logic gates with high transition activity are devoted to perform the multiplication operation. The logic transitions cause the logic gates to charge/discharge the load capacitance leading to dynamic power dissipation. This section provides a brief introduction to various sources of power dissipation in CMOS based designs. It is followed by preliminary information on existing reconfigurable multipliers and bypass techniques to minimize the dynamic power dissipation.

### 2.5.1 Dynamic Power Dissipation in CMOS based Circuits

The main source of power dissipation in CMOS based circuits is the dynamic power dissipation caused by switching activity of the logic circuits. Dynamic power dissipation is given by,

$$P_{av} = C_L V_{DD}^2 f_p \alpha \tag{2.17}$$

Where:

$C_L$ = charged load capacitance

$V_{DD}$ = supply voltage

$f_p$ = clock frequency

$\alpha$ = switching activity factor.

The dynamic power dissipated is thus proportional to the number of transitions occurring in a logic gate. Various power reduction methods to minimize the redundant switching$(\alpha)$ have been proposed in the literature as described later.

### 2.5.2 Power Consumption in Parallel Multipliers

In general, multipliers can be implemented as sequential or combinational circuits. However, in the current work, the focus is on parallel multipliers which are purely combinational circuits. Parallel multipliers are fairly complex circuits with a large transistor count and frequent switching of these transistors to carry out logic computations leads to large dynamic power dissipation. As elaborated in Section 3.2, parallel multipliers have the following computation steps: partial product generation, partial product reduction and vector merge addition. The partial product accumulation step, which predominantly comprises of adder units, dictates the overall computation delay, area and power consumption. An obvious technique to minimize power dissipation is to disable the unwanted computations in an adder. A number of methods to bypass the adders has been proposed and discussed in the literature [10–12].

### 2.5.3 Reconfigurable Binary Multiplier

In a binary multiplier, the die area and power consumption are largely dependent upon the word-size. Assuming that an application needs N-bit precision, then using a data path element of more than the required precision would result in wasted area and power. To overcome this problem, a twin-precision multiplier has been proposed in [13]. An attempt has been made to minimize the impact on delay and power of the N-bit multiplier by making as few modifications as possible to the conventional multiplier. This twin-precision scheme decomposes the N*N partial-product matrix into two N/2 * N/2 independent multiplications by configuring the appropriate partial products [14]. When it operates on N/2-bit operands however, large parts of the multiplier do not contribute to the final result although they may be active. Thus, the multiplier dissipates considerable dynamic power due to the switching activity involved in computing unwanted partial products. This problem is sought to be addressed in this work by using bypass computation cells that disable unnecessary computations.

### 2.5.4 A Review of Bypass Multiplier Architectures

Figure.2.14(a) illustrates the example of multiplication of two unsigned 4-bit numbers, where $A = a_3a_2a_1a_0$ is the multiplier and $B = b_3b_2b_1b_0$ is the multiplicand. In a conventional Braun array multiplier for example, the partial products are generated in parallel with the AND gates and added using a 1-bit full adder as illustrated in Fig.2.14(b) .

The adder circuits shown in Fig.2.14(b) tend to perform computation of the partial products even if their value is ‘0’ and this results in undesired signal transitions. These transitions can be avoided by disabling the respective adder cells which results in saving of power.

#### 2.5.4.1 Row-Bypass Scheme

Various techniques have been proposed from time to time to reduce the switching activity in array multipliers, of which bypass architectures are an offshoot. A simple approach to reduce the power consumption is to avoid unnecessary computations. Ohban [10] proposed a row-bypass scheme wherein some rows in the multiplier array are skipped to reduce the redundant switching activity. Figure 2.15 illustrates an implementation of a 4*4 Braun multiplier using

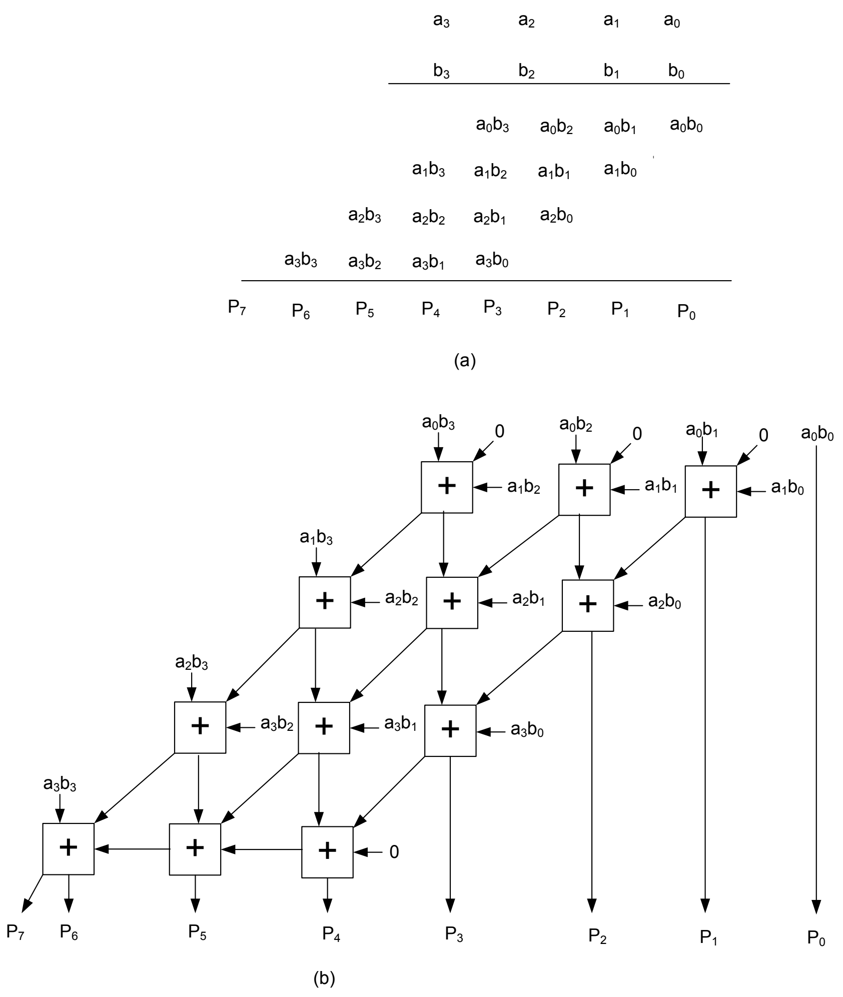


Figure 2.14: (a) An example of 4*4 array multiplication (b) Schematic diagram of 4*4 Braun multiplier

row-bypassing technique.

This scheme includes adder cells (denoted by AC and highlighted in Fig.2.15) to bypass the inputs to output whenever the row (horizontal) partial product is zero. The tri-state buffers augmented at the inputs of the adder cell disable unnecessary transitions by shutting down the full adder. The MUXes at the outputs of the adder cells automatically pass the carry-input and the sum of the previous addition to the next computational unit when the corresponding partial product is zero. A notable drawback in this scheme however is the additional logic circuitry required as highlighted in grey color in Fig.2.15.

A numerical example illustrating the multiplication of two 4-bit numbers using row-bypass scheme is shown in Fig.2.16. Since the partial products in the second row are zeroes, the corresponding computational units are turned off to save power. The partial products in the

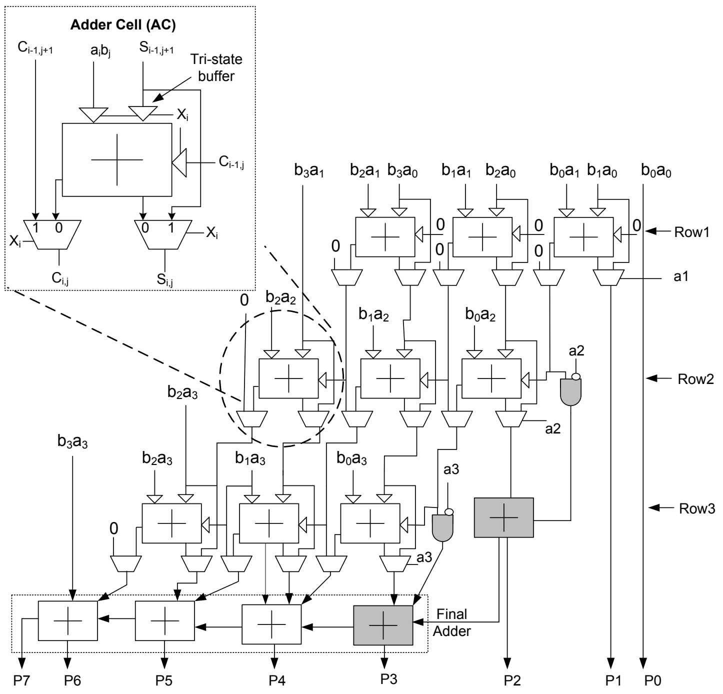


Figure 2.15: Schematic diagram of a 4*4 Braun multiplier using row-bypassing technique

first row are bypassed and added with partial products in the next level (third row). In a similar manner, the remaining partial products are reduced to form the final product.

```
     1 1 1 1
    X 1 1 0 1
 ____________________
        1 1 1 1
      0 0 0 0      → Row
     1 1 1 1         Bypass
   1 1 1 1
 ____________________
  1 1 0 0 0 0 1 1
```

Figure 2.16: Numerical example of row-bypass scheme for 4*4 multiplier

#### 2.5.4.2 Column-Bypass Scheme

Wen [11] proposed a column-bypass scheme which avoids the adder operations in some columns instead of rows. In this approach, some columns in the partial product matrix can be skipped whenever their outputs are known. Consequently, the switching activity and therefore power dissipation are reduced. This technique has two important advantages: (i) It removes the extra compensating circuitry (ii) the modified full adder (MFA) unit is less complex than that used in the row-bypassing multiplier.

A typical 4*4 column-bypassing multiplier is illustrated in Fig.2.17 where the modified adder (MFA) cell is highlighted. The MFA cell skips the full adder whenever the partial product in the corresponding column is zero. This multiplier has less hardware complexity compared to the row-bypassing scheme also because it does not need to consider bypassing of the carry bit.

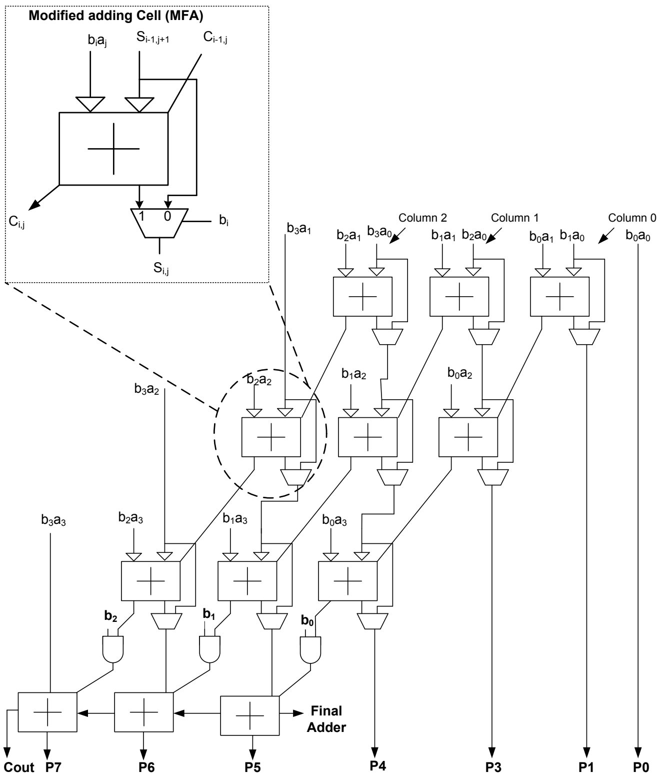


Figure 2.17: Schematic diagram of a 4*4 Braun multiplier with column-bypassing scheme

#### 2.5.4.3 Two-Dimensional Bypass Scheme

In a 2-dimensional bypassing multiplier, the computing logic cells skip the corresponding row and column depending on nullity of the partial products [12, 15]. Figure 2.18 shows the structure of the 4*4 Braun multiplier with 2-dimensional bypassing scheme [12].

To overcome the conflict that occurs when both row-bypassing and column-bypassing appear simultaneously, bypass adder cells (AC) incorporate additional logic. These bypass cells have the capability to bypass when either row and/or column element is zero, however with large circuit overhead. In view of this additional complexity, power saving tend to get re-

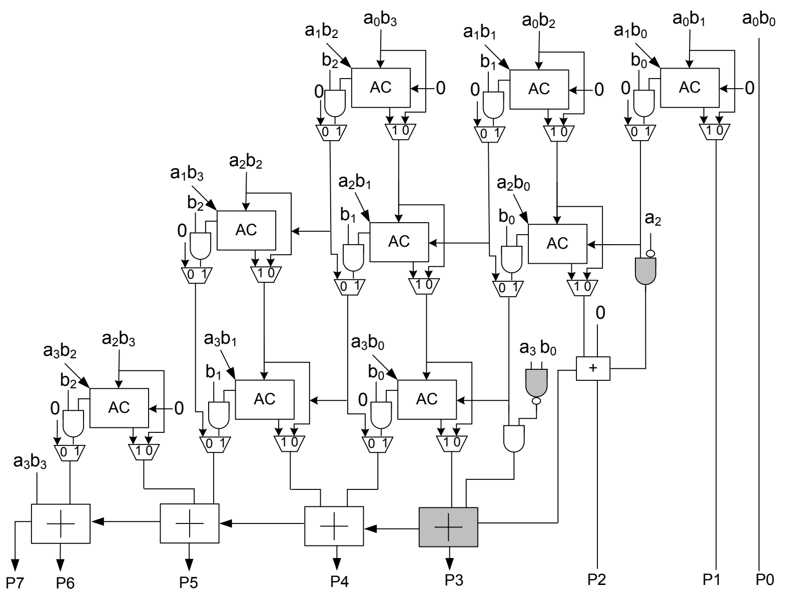


Figure 2.18: Schematic diagram of a 4*4 Braun multiplier with two-dimensional-bypassing scheme

duced. To overcome this, Hong [15] introduced two kinds of adder cells, namely, modified row-bypassing adder (MRBA) and two-dimensional bypassing adder (TDBA). The MRBA cells have row-bypassing capability while the TDBA cells are deactivated when either row or column partial product becomes zero.

## 2.6 A Review of Recursive Binary Multipliers

This section presents the mathematical modeling of the recursive binary multiplier. This is followed by various truncation schemes that have been used in the existing multiplier architectures.

### 2.6.1 Mathematical Analysis of Recursive Multiplier

Recursive multipliers based on Karatsuba-Ofman Algorithm (KOA) [16] are found to have a hierarchical architecture consisting of several sub-multipliers making them ideal for fixed-width multiplication.

Assume $A$ and $B$ to be two $2n$-bit unsigned numbers, where $A$ is the multiplicand and $B$ is the multiplier. $A$ and $B$ can be written as:

$$A = \sum_{i=0}^{2n-1} a_i.2^i \tag{2.18}$$

$$B = \sum_{j=0}^{2n-1} b_j.2^j \tag{2.19}$$

The recursive multiplication is performed by partitioning each of the operands into two equal portions of $n$-bit width. Based on this, the multiplicand $(A)$ is split into $A_H$ and $A_L$, while multiplier $(B)$ is divided into $B_H$ and $B_L$ respectively as mentioned in equations 2.20 and 2.21 given below:

$$A = A_H * 2^n + A_L \tag{2.20}$$

$$B = B_H * 2^n + B_L \tag{2.21}$$

The subscript $H$ denotes the most significant portion while $L$ denotes the lower significant portion of the corresponding binary numbers.

The product $(P)$ is written as follows:

$$P = A * B = (A_H * B_H) * 2^{2n} + (A_L * B_H + A_H * B_L) * 2^n + A_L * B_L \tag{2.22}$$

Thus, multiplication can be performed using four $n * n$ binary sub-multipliers, namely, $A_H * B_H$, $A_H * B_L$, $A_L * B_H$, and $A_L * B_L$, in parallel as shown in Fig.2.19. The partial products of all the individual sub-multipliers are reduced to product $(P)$ of $2n$-bit width using a reduction structure.

## 2.6.2 Truncation Schemes for Binary Multipliers

Several techniques [17–20] have been proposed in the past to achieve fixed-width multiplication. Among these, truncation techniques developed for recursive multipliers have been proven to be efficient as opposed to the array multipliers in terms of die area and power dissipation [17].

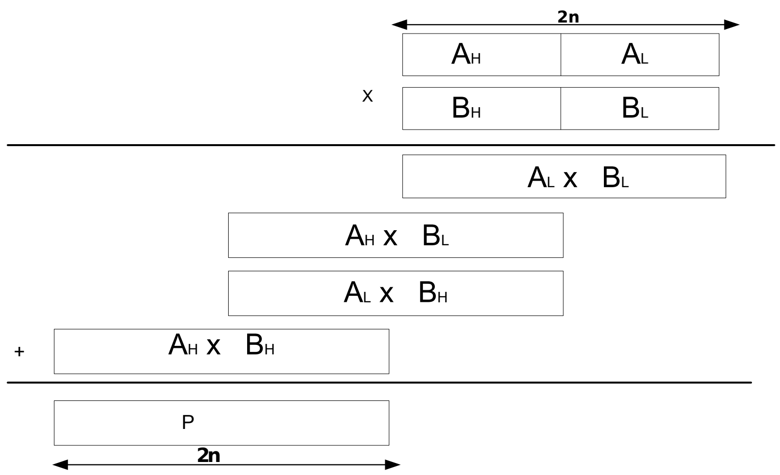


Figure 2.19: Schematic diagram of the original recursive multiplication scheme

#### 2.6.2.1 Truncation Schemes for Array Multipliers

In truncation schemes for array multipliers, the least significant bits of the partial product matrix are removed and a correction function, which is either constant or data-dependent, is added to compensate for the error [21–24].

Authors in [21, 23] present a constant correction technique where compensation function is based on the average value of the partial product bits which are not formed. This technique results in simple hardware which in turn leads to higher power savings. However, the error bounds obtained are high. To overcome this, a data-dependent correction technique proposed in [22, 25] adds a correction value based on the partial products corresponding to least significant column that are not formed. This technique, also referred to as variable correction, achieves a lower error bound compared to the constant correction schemes, though at the cost of the hardware complexity.

#### 2.6.2.2 Truncation Schemes for the Recursive Multiplier

Most of the truncation techniques targeted at array multipliers focus on modifying the multiplier structure. However, truncation schemes applied to recursive multipliers simply get rid of a least sub-multiplier $(A_L B_L)$ as shown in Fig.2.20 and replace it with a correction function which is data dependent.

Three correction schemes can be found in the literature. In scheme 1 [18], $A_H * B_L$ or $A_L * B_H$ sub-multiplier replaces $A_L * B_L$ while in scheme 2, the average value of $A_H * B_L$ and

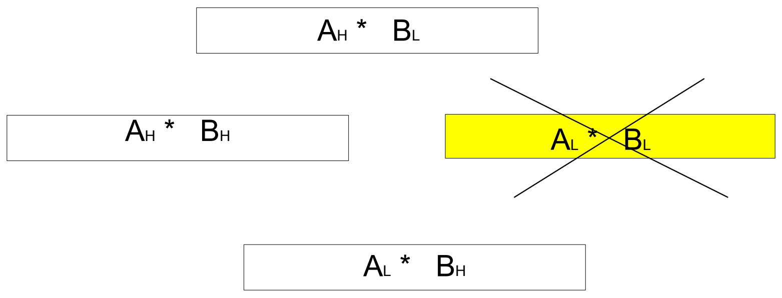


Figure 2.20: Sub-multipliers in a recursive multipliers

$A_L * B_H$ is used. In scheme 3, the most significant partial product bit of $A_L * B_L$, namely, $a_{n-1} * b_{n-1}$ forms the correction function. All these schemes are based on trial and error and the precision achieved is also fixed. In this work, a tunable correction function is proposed using a systematic approach and it‘s performance is compared with the existing ones.

## 2.7 Multipliers based on Logarithmic Number System

Most of the logarithmic multiplier schemes can be classified as iterative [26,27] or non-iterative [28–32]. Non-iterative multipliers have limited precision due to the usage of techniques such as piecewise linear approximation [33], memory look-up [34] or a combination of both [35] making them limited to only a few applications. On the other hand, iterative multipliers tend to improve the precision of the result with each successive iteration.

Mitchell [36] introduced the first iterative multiplier that was simple and flexible to meet the requirements of a wide range of applications. This multiplier however suffered from large relative error in the final result. Further, Mitchell approach cannot initiate next iteration until the completion of the present one. Babic [26] modified the Mitchell design by introducing greater pipeline-level parallelism with an objective to reduce the latency. However, his approach lead to reduced precision in each iteration due to the neglect of carry. Babic iterative multiplier (BIM) design was further improved by truncated error correction (TEC) method [27] which has an additional capability for speculative carry, thereby improving the precision. This however comes at the cost of area overhead.

To overcome these shortcomings, the present work combines carry speculation with an improved fractional predictor leading to a better precision when compared to the existing work.

The fractional predictor logic and its efficient precomputation contribute to the improved overall precision due to a fewer number of iterations required compared to the existing techniques. Further, precision of the multiplier improves as the number of iterations increases. Also, savings in hardware are achieved using the truncation scheme proposed in this work. The proposed and the existing logarithmic multipliers have been applied on an image sharpening algorithm and compared in the context of certain well-known image processing benchmarks such as Lena and Cameraman for performance.

### 2.7.1 Mathematical Analysis of MA Based Multiplier

This section presents the mathematical approach common for MA based multipliers [26,27,36] described below:

According to Mitchell, the binary representation of two n-bit input numbers $N_1$ and $N_2$ is given as :

$$\begin{cases} N_1 = 2^{k_1}(1+x_1) \\ N_2 = 2^{k_2}(1+x_2) \end{cases} \tag{2.23}$$

The characteristics of $N_1$ and $N_2$ are $k_1$ and $k_2$ respectively, representing the most significant operand bits with the value of '1'. Further, $x_1$ and $x_2$ denote fractional portions whose values lie in the range [0,1].

The base-2 logarithm of the product, $N_1$ and $N_2$ is written as

$$log_2(N_1 * N_2) = k_1 + k_2 + log_2(1+x_1) + log_2(1+x_2) \tag{2.24}$$

To compute the antilogarithm of equation (2.24), Mitchell proposed the following analytical expressions based on carry information from the fractional portion

$$N_1 * N_2 = 2^{k_1+k_2}(1+x_1+x_2) + 2^{k_1+k_2}(x_1 * x_2), \qquad x_1 + x_2 < 1 \tag{2.25}$$

and

$$N_1 * N_2 = 2^{k_1+k_2+1}(x_1 + x_2) + 2^{k_1+k_2}(x_1' * x_2'),$$
$$x_1 + x_2 \geq 1 \tag{2.26}$$

Where $2^{k_1+k_2}(x_1 * x_2)$ and $2^{k_1+k_2}(x_1' * x_2')$ are the correction terms.

Babic [26] ignored the carry altogether resulting in a simple and faster design with trade-off, however, in precision. Accordingly, Babic used the above expression (2.25).

Further, error due to the approximation was avoided by considering the relation given in equation (2.23) :

$$\begin{cases} x_1 * 2^{k_1} = N_1 - 2^{k_1} \\ x_2 * 2^{k_2} = N_2 - 2^{k_2} \end{cases} \tag{2.27}$$

Combining equations 2.25 and 2.27 results in,

$$N_1 * N_2 = 2^{k_1+k_2} + f_1 * 2^{k_2} + f_2 * 2^{k_1} + f_1 * f_2$$

Where $N_1 - 2^{k_1} = f_1$ ; $N_2 - 2^{k_2} = f_2$

The above equation is represented as

$$N_1 * N_2 = A^0 + f_1 * f_2 \tag{2.28}$$

where approximate product term $A^0 = 2^{k_1+k_2} + f_1 * 2^{k_2} + f_2 * 2^{k_1}$

The computation of term $f_1 * f_2$ given in equation (2.28) requires multiplication. Evidently, the product $N_1 * N_2$ gets simplified, if these terms are ignored which leads to sacrificing the precision. This was the approach adopted by Babic and TEC designs. Nevertheless, the correction term $(f_1 * f_2)$ can be computed in parallel with $A^0$, which however results in area overhead.

## 2.7.2 Hardware Architectures

The architecture of Babic multiplier [26] for one iteration is illustrated in Fig.2.21. It consists of components such as basic logarithmic converter blocks (BLBs), decoder and adders.

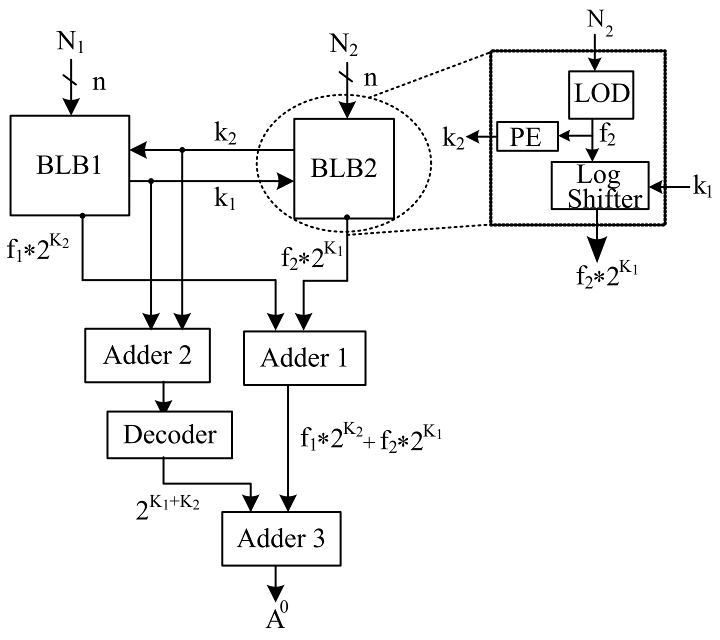


Figure 2.21: Functional diagram of Babic Iterative Multiplier (BIM)

A typical BLB includes a leading one detector (LOD), priority encoder (PE) and logarithmic shifter modules. It forms the fundamental module in the design of iterative multipliers and provides the characteristic $(k)$ and fractional portions $(f)$.

Based on the binary number $(N_2)$ and characteristic $(k_1)$, the BLB2 block highlighted in Fig.2.21 calculates the shifted fractional portion, $f_2 * 2^{k_1}$ and characteristic, $k_2$. Similarly, BLB1 block computes the fractional portion, $f_1 * 2^{k_2}$ and characteristic, $k_1$. The Adder 2 and Decoder logic calculate the integer portion of the product $(2^{k_1+k_2})$ while the summation of fraction portions $(f_1 * 2^{k_2}$ and $f_2 * 2^{k_1})$ computed using Adder 1 provides fractional portion $(f)$. The computation of the product $(A^0)$ is achieved by the addition of fractional portion $(f)$ and output of the Decoder using Adder 3 block. The inputs to next iteration are $f_1$ and $f_2$ which are obtained from the respective LOD circuits.

The truncated error correction approach suggested in [27] extends the BIM scheme with addition of fractional predictor (FP), shared logic, multi-operand addition (MOA) and mask as illustrated in Fig.2.22.

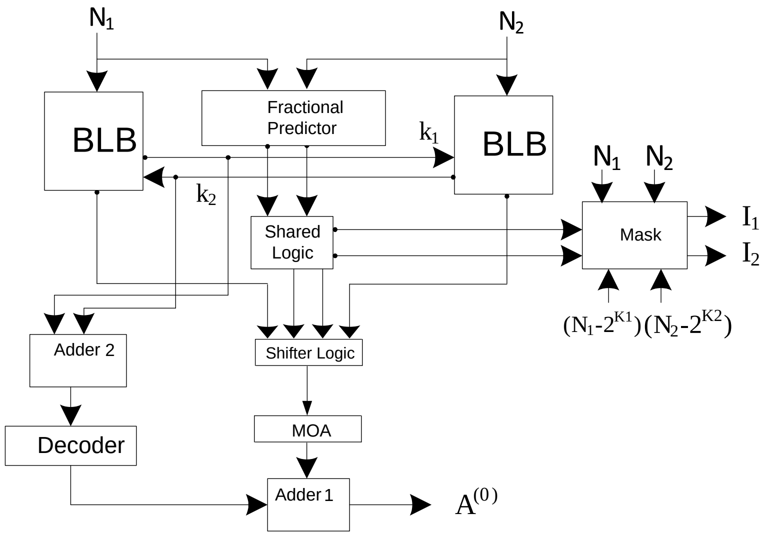


Figure 2.22: Functional diagram of truncated error correction (TEC) Scheme

The speculation of carry from fractional portion is carried out by a variable size FP while the shared logic gives the position of fractional bits that require error correction. The error correction itself is accomplished using shifter and multi-operand adder (MOA) and the inputs for next iteration are computed via mask logic. The shortcomings in TEC multiplier include extra hardware circuitry and lower error reduction rate for successive iterations.

## 2.8 Decimal Multiplication

Decimal multiplication typically have the following stages: (i) partial product generation (ii) partial product reduction and (iii) final product computation. A general architecture of 'digit-by-digit' multiplier is shown in Fig.2.23. The decimal multiplier accepts two BCD inputs A and B of m-bit width. In the partial product generation stage, the individual digits of multiplier and multiplicand are multiplied using the BDMs.

The reduction of partial products is accomplished using ripple-free binary CSA tree and conversion to decimal is achieved using the multi-operand BD converter. The final product is obtained after the addition of the decimal digits using decimal adder.

The 'digit-by-digit' multiplication is presented in the following Subsection 2.8.1 while various existing partial product generation and reduction schemes adapted for present designs are discussed in Subsection 2.8.2.

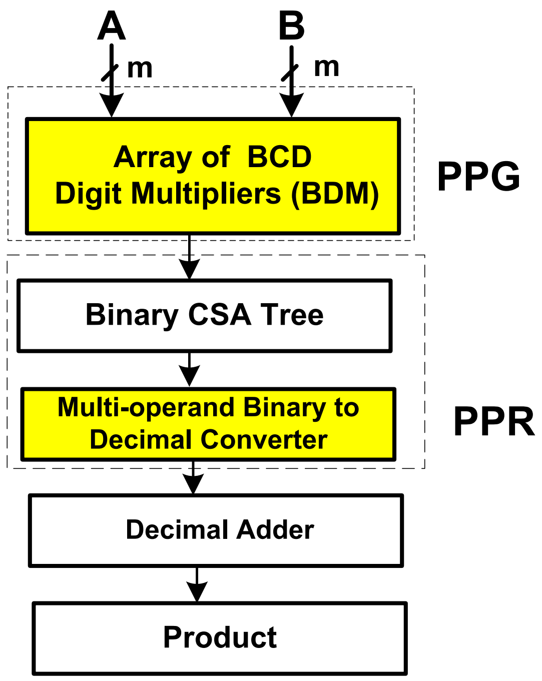

A
m
B
m
Array of BCD Digit Multipliers (BDM)
PPG
Binary CSA Tree
Multi-operand Binary to Decimal Converter
PPR
Decimal Adder
Product

Figure 2.23: A top-level architecture of ‘digit-by-digit’ multiplication

## 2.8.1 ‘Digit-by-Digit’ Multiplier

Decimal (BCD) arithmetic computations are generally sluggish (slow) and tend to occupy more silicon area. This has led to efforts to improve decimal architectures that result in high performance and compact arithmetic circuits [37]. For example, microprocessors such as IBM Power PC [38] and IBM z10 [39] include dedicated decimal hardware units.

Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic is multiplication. While a large body of literature on decimal arithmetic covers serial multiplication, parallel (‘word-by-digit’) [40–43] and (‘digit-by-digit’) [44, 45] multiplication has also been reported recently. Decimal (BCD) ‘digit-by-digit’ multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in a significant improvement in the multiplier performance [46].

A step by step implementation of 4*4 ‘digit-by-digit’ multiplication [44] is illustrated in Fig.2.24. Multiplication of each digit of the multiplicand with the digit of multiplier is performed using the BDM.

For example, multiplication of $A_1$ and $B_1$ is highlighted in the dotted circle of the figure.

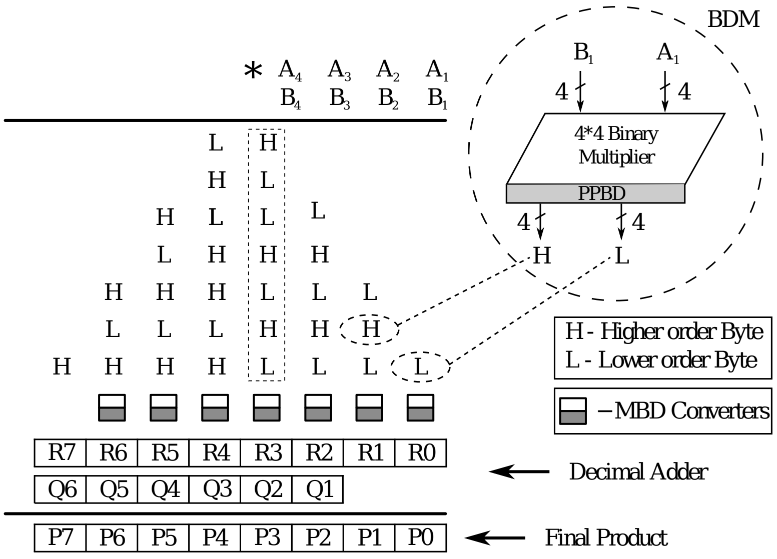


Figure 2.24: Example of 4*4 ‘digit-by-digit’ multiplication using BDMs

The output of the BDM results in most significant digit and least significant digit denoted by H and L respectively. A typical BDM is composed of a 4*4 binary multiplier and a partial product binary to decimal (PPBD) converter. Most of the previous work available in literature is focused on PPBD converters at partial product generation stage which is discussed in Section 2.8.2.

The individual decimal partial product columns (one such column is highlighted with dotted rectangle in Fig.2.24) are compressed in parallel by using a tree of binary carry save adders (CSAs) resulting in a binary number as output of each column. The conversion from binary to decimal is carried out using multi-operand BD (MBD) converters resulting in rows of decimal digits R0-R7 and Q1-Q6 which are eventually compressed using a decimal adder to obtain the final product (P0-P7).

## 2.8.2 A Review of Partial Product Generation and Reduction Schemes

### 2.8.2.1 Partial Product Generation (Binary Product to BCD conversion)

The algorithm proposed in [44] converts a 7-bit binary number ($p_6p_5p_4p_3p_2p_1p_0$) to a 2-digit BCD number ($D_H$ and $D_L$ ) to support high performance decimal multiplication. This algorithm calculates the contributions for lower BCD digit ($D_L$) and the higher BCD digit ($D_H$) from each of the input binary bits as shown in Table 2.1.

Table 2.1: Principle of Binary to BCD conversion

| 80 | 40 | 20 | 10 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_2$ | $p_1$ | $p_0$ |
| 0 | 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_4$ | 0 |
|  |  |  |  | $p_6$ | 0 | $p_5$ | 0 |
|  |  |  |  | $p_3$ | 0 | 0 | 0 |
| $d_{h3}$ | $d_{h2}$ | $d_{h1}$ | $d_{h0}$ | $d_{l3}$ | $d_{l2}$ | $d_{l1}$ | $d_{l0}$ |

The first row in the Table shows the BCD weights. The binary numbers $p_3, p_2, p1$ and $p_0$ are retained in their position as their weights are same as the corresponding weights in the original binary number. However, the weights 16, 32 and 64 corresponding to $p_4$, $p_5$ and $p_6$ are decomposed into $(10,4,2)$, $(20,10,2)$ and $(40,20,4)$, respectively. The four columns in the right consisting of BCD digits are summed using BCD adder leading to the BCD digit $D_L$ $(d_{l3}d_{l2}d_{l1}d_{l0})$ while the resulting carry is added to the BCD digit that is present in the left three columns leading to $D_H$ $(d_{h3}d_{h2}d_{h1}d_{h0})$.

Work in [47] modifies the architecture in [44] by adding the contributions in a BCD fashion. This design partitions or splits the binary input into two sub-parts, three MSBs and four LSBs. It calculates the contributions for the two BCD digits and adds them in a BCD fashion to get the final result.

Work presented in [48] proposes two schemes, 'three-four split' and 'four-three split' binary to BCD converters. The 'three-four split' algorithms have optimized $D_L$ and $D_H$ generator blocks resulting in better performance in terms of area, delay and power. An illustration of the 'three-four' split algorithm is provided in Fig.2.25.

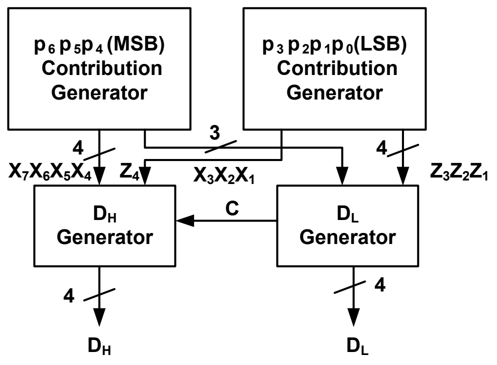

p₆p₅p₄(MSB) Contribution Generator
p₃p₂p₁p₀(LSB) Contribution Generator
4
3
4
X₇X₆X₅X₄
Z₄
X₃X₂X₁
Z₃Z₂Z₁
C
D_H Generator
D_L Generator
4
4
D_H
D_L

Figure 2.25: Block diagram of 'three-four split' binary to BCD converter

The 'four-three split' design partitions the 7-bit binary input into four MSBs and three

LSBs. Since the LSBs do not contribute to the higher BCD digit the LSB contribution generator is removed resulting in area savings. However, this comes at the cost of increased complexity of MSB contribution generator as shown in Fig.2.26. The 'three-four split' is faster than the 'four- three split' whereas the 'four-three split' results in a more area efficient design.

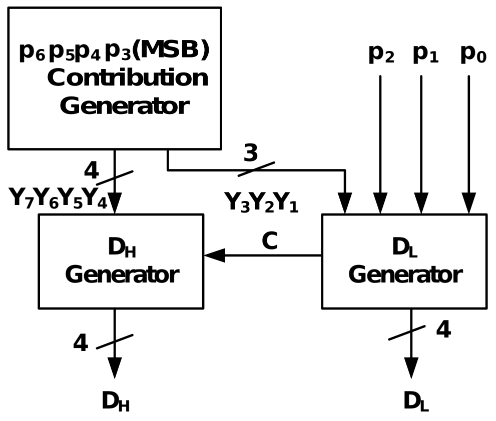


Figure 2.26: Block diagram of 'four-three split' binary to BCD converter

Work published in [49] adapted a binary-to-BCD conversion cell proposed by Nicoud [50]. Although, it was Dadda [51] who first showed that an iterative array of Nicoud's cells can be used to design multi-operand BD converters at PPR level mentioned in later Section. This idea was used in [49] however at PPG level to design PPBD converter.

Certain shortcomings, however, have been recognized in these methods such as (i) redundant contribution blocks [47,48] and (ii) large area consumption [49]. To alleviate these, two partial product BD converters, namely 'high performance' PPBD (HPPPBD) and 'low area' PPBD (LAPPBD) converters, are proposed in this work (Section 6.2).

#### 2.8.2.2 Partial Product Reduction

Partial product reduction in the first stage of 'digit-by-digit' multiplier is achieved using a binary CSA structure [52]. The binary result of each partial product column is subsequently converted to decimal (BCD) using a multi-operand BD converter consisting of iterative connection of Nicoud cells [50] suggested by Dadda [51]. A typical Nicoud (ND) cell would accept a 4-bit binary input $(b_j)$, multiplies it by two, and then adds it to $b_i$ as depicted in Fig. 2.27(a). Thus the computation of BCD (decimal) outputs, $b_0$ (higher digit) and $D_0$ (lower digit) is carried out using the relation $\{b_0, D_0\} = 2.b_j + b_i$ where the maximum values of $b_0$ and $D_0$

are $(1)_{10}$ and $(9)_{10}$ respectively.

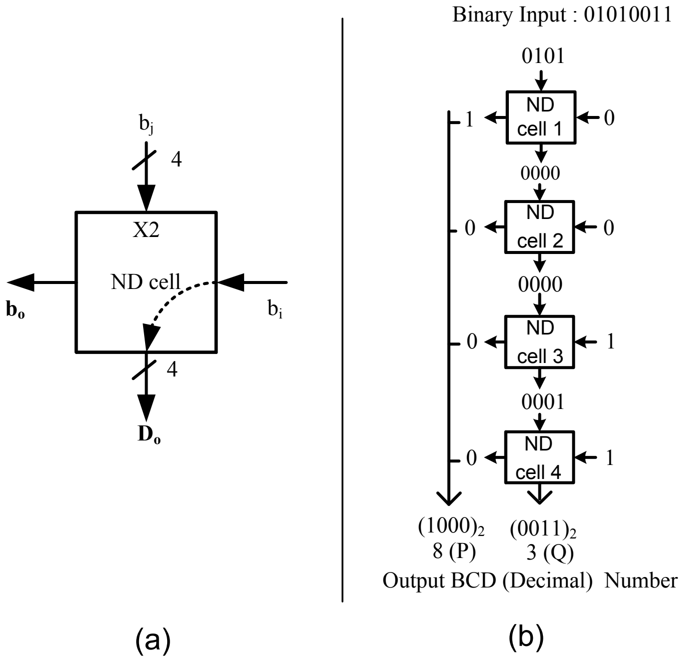


Figure 2.27: (a) Compact notation of Nicoud cell (b) Linear array of Nicoud cells to form Dadda multi-operand BD converter

An example to convert a binary number to two BCD digits ($b_0$ and $D_0$) is illustrated in Fig.2.27(b). Since the binary number $(1010011)_2$ to be converted here is larger than $(19)_{10}$, four Nicoud cells are required to realize the converter. As illustrated in Fig.2.27(b), the input to the Nicoud cell is restricted to $(1001)_2$. Hence the 3 MSBs of binary input along with '0' prepended $(0101)_2$ is accepted as $b_j$ and the next significant binary input '0' as $b_i$ resulting in the outputs $(1)_2$ and $(0000)_2$. The 4-bit output $(0000)_2$ of cell 1 along with the next significant binary input '0' form input to cell 2, resulting in $(0)_2$ and $(0000)_2$. Similarly, the 4-bit output of each subsequent cell along with residual 1-bit binary input feeds the decimal input of the following cell resulting in higher (P) digit $(1000)_2$ and lower (Q) digit $(0011)_2$. In general, binary number of any operand width can be converted to decimal by a linear arrangement of Nicoud cells.

The limitation of Nicoud cells however is their latency and thus the delay of multi-operand BD converter increases with the size of the binary number. To mitigate this, a hybrid multi-operand BD converter is proposed in this work. A detailed discussion of the partial product reduction scheme is presented in Section 6.3 .

## 2.9 Conclusions

In this chapter, the necessary background material about the multipliers based on different number systems is presented. This knowledge is required to understand the subsequent chapters included in the thesis. The objective of this chapter was to provide a quick introduction to various architectures such as fixed width binary multipliers, logarithmic and BCD multipliers. The multipliers based on binary and logarithmic number offers a low power alternative solution in error resilience applications. On the other hand decimal arithmetic has been increasing used in the financial applications where precision is very important. Finally, the overall research approach followed in this thesis is presented.

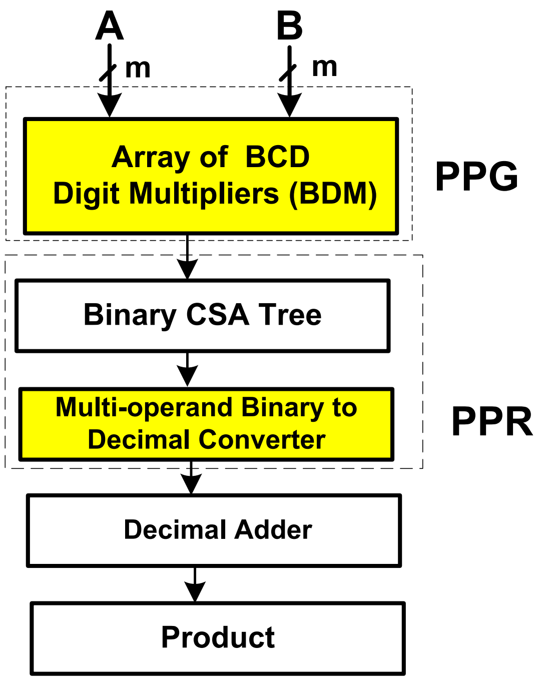

A
B
m
m
Array of BCD Digit Multipliers (BDM)
PPG
Binary CSA Tree
Multi-operand Binary to Decimal Converter
PPR
Decimal Adder
Product

Figure 2.23: A top-level architecture of ‘digit-by-digit’ multiplication

### 2.8.1 ‘Digit-by-Digit’ Multiplier

Decimal (BCD) arithmetic computations are generally sluggish (slow) and tend to occupy more silicon area. This has led to efforts to improve decimal architectures that result in high performance and compact arithmetic circuits [37]. For example, microprocessors such as IBM Power PC [38] and IBM z10 [39] include dedicated decimal hardware units.

Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic is multiplication. While a large body of literature on decimal arithmetic covers serial multiplication, parallel (‘word-by-digit’) [40–43] and (‘digit-by-digit’) [44, 45] multiplication has also been reported recently. Decimal (BCD) ‘digit-by-digit’ multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in a significant improvement in the multiplier performance [46].

A step by step implementation of 4*4 ‘digit-by-digit’ multiplication [44] is illustrated in Fig.2.24. Multiplication of each digit of the multiplicand with the digit of multiplier is performed using the BDM.

For example, multiplication of $A_1$ and $B_1$ is highlighted in the dotted circle of the figure.

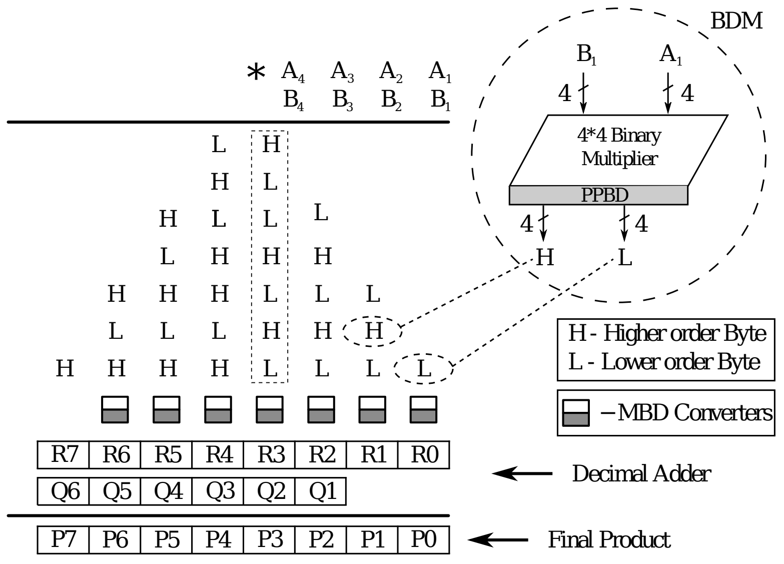


Figure 2.24: Example of 4*4 'digit-by-digit' multiplication using BDMs

The output of the BDM results in most significant digit and least significant digit denoted by H and L respectively. A typical BDM is composed of a 4*4 binary multiplier and a partial product binary to decimal (PPBD) converter. Most of the previous work available in literature is focused on PPBD converters at partial product generation stage which is discussed in Section 2.8.2.

The individual decimal partial product columns (one such column is highlighted with dotted rectangle in Fig.2.24) are compressed in parallel by using a tree of binary carry save adders (CSAs) resulting in a binary number as output of each column. The conversion from binary to decimal is carried out using multi-operand BD (MBD) converters resulting in rows of decimal digits R0-R7 and Q1-Q6 which are eventually compressed using a decimal adder to obtain the final product (P0-P7).

## 2.8.2 A Review of Partial Product Generation and Reduction Schemes

### 2.8.2.1 Partial Product Generation (Binary Product to BCD conversion)

The algorithm proposed in [44] converts a 7-bit binary number ($p_6p_5p_4p_3p_2p_1p_0$) to a 2-digit BCD number ($D_H$ and $D_L$ ) to support high performance decimal multiplication. This algorithm calculates the contributions for lower BCD digit ($D_L$) and the higher BCD digit ($D_H$) from each of the input binary bits as shown in Table 2.1.

Table 2.1: Principle of Binary to BCD conversion

| 80 | 40 | 20 | 10 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_2$ | $p_1$ | $p_0$ |
| 0 | 0 | $p_6$ | $p_5$ | $p_4$ | 0 | $p_4$ | 0 |
|  |  |  |  | $p_6$ | 0 | $p_5$ | 0 |
|  |  |  |  | $p_3$ | 0 | 0 | 0 |
| $d_{h3}$ | $d_{h2}$ | $d_{h1}$ | $d_{h0}$ | $d_{l3}$ | $d_{l2}$ | $d_{l1}$ | $d_{l0}$ |

The first row in the Table shows the BCD weights. The binary numbers $p_3, p_2, p1$ and $p_0$ are retained in their position as their weights are same as the corresponding weights in the original binary number. However, the weights 16, 32 and 64 corresponding to $p_4$, $p_5$ and $p_6$ are decomposed into $(10,4,2)$, $(20,10,2)$ and $(40,20,4)$, respectively. The four columns in the right consisting of BCD digits are summed using BCD adder leading to the BCD digit $D_L$ $(d_{l3}d_{l2}d_{l1}d_{l0})$ while the resulting carry is added to the BCD digit that is present in the left three columns leading to $D_H$ $(d_{h3}d_{h2}d_{h1}d_{h0})$.

Work in [47] modifies the architecture in [44] by adding the contributions in a BCD fashion. This design partitions or splits the binary input into two sub-parts, three MSBs and four LSBs. It calculates the contributions for the two BCD digits and adds them in a BCD fashion to get the final result.

Work presented in [48] proposes two schemes, 'three-four split' and 'four-three split' binary to BCD converters. The 'three-four split' algorithms have optimized $D_L$ and $D_H$ generator blocks resulting in better performance in terms of area, delay and power. An illustration of the 'three-four' split algorithm is provided in Fig.2.25.

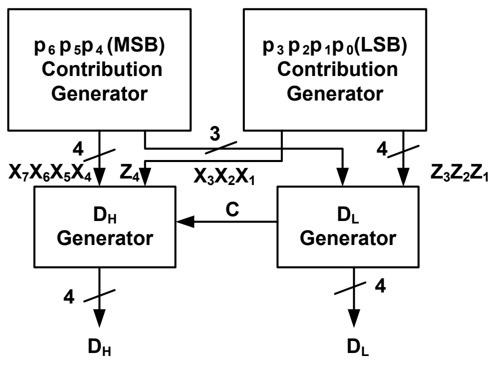

p₆p₅p₄(MSB) Contribution Generator
p₃p₂p₁p₀(LSB) Contribution Generator
4
3
4
X₇X₆X₅X₄
Z₄
X₃X₂X₁
Z₃Z₂Z₁
C
D_H Generator
D_L Generator
4
4
D_H
D_L

Figure 2.25: Block diagram of 'three-four split' binary to BCD converter

The 'four-three split' design partitions the 7-bit binary input into four MSBs and three

LSBs. Since the LSBs do not contribute to the higher BCD digit the LSB contribution generator is removed resulting in area savings. However, this comes at the cost of increased complexity of MSB contribution generator as shown in Fig.2.26. The ‘three-four split’ is faster than the ‘four- three split’ whereas the ‘four-three split’ results in a more area efficient design.

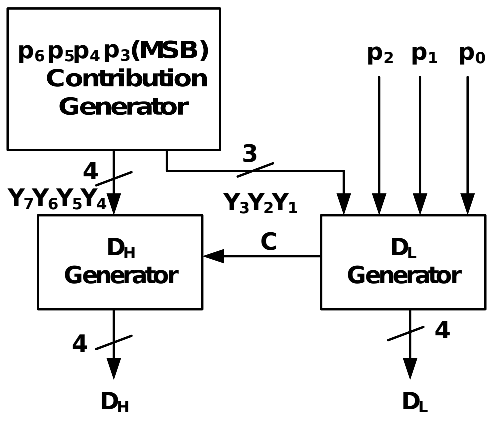


Figure 2.26: Block diagram of ‘four-three split’ binary to BCD converter

Work published in [49] adapted a binary-to-BCD conversion cell proposed by Nicoud [50]. Although, it was Dadda [51] who first showed that an iterative array of Nicoud’s cells can be used to design multi-operand BD converters at PPR level mentioned in later Section. This idea was used in [49] however at PPG level to design PPBD converter.

Certain shortcomings, however, have been recognized in these methods such as (i) redundant contribution blocks [47, 48] and (ii) large area consumption [49]. To alleviate these, two partial product BD converters, namely ‘high performance’ PPBD (HPPPBD) and ‘low area’ PPBD (LAPPBD) converters, are proposed in this work (Section 6.2).

#### 2.8.2.2 Partial Product Reduction

Partial product reduction in the first stage of ‘digit-by-digit’ multiplier is achieved using a binary CSA structure [52]. The binary result of each partial product column is subsequently converted to decimal (BCD) using a multi-operand BD converter consisting of iterative connection of Nicoud cells [50] suggested by Dadda [51]. A typical Nicoud (ND) cell would accept a 4-bit binary input $(b_j)$, multiplies it by two, and then adds it to $b_i$ as depicted in Fig. 2.27(a). Thus the computation of BCD (decimal) outputs, $b_0$ (higher digit) and $D_0$ (lower digit) is carried out using the relation $\{b_0, D_0\} = 2.b_j + b_i$ where the maximum values of $b_0$ and $D_0$

are $(1)_{10}$ and $(9)_{10}$ respectively.

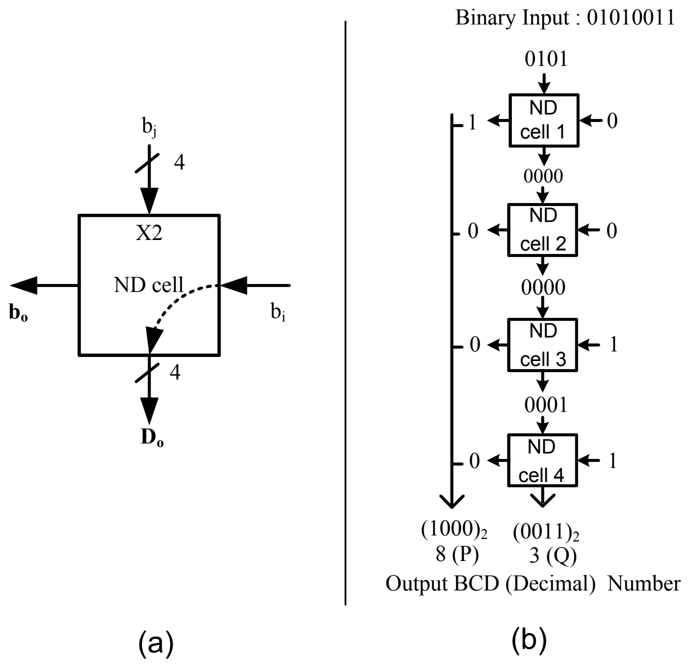


Figure 2.27: (a) Compact notation of Nicoud cell (b) Linear array of Nicoud cells to form Dadda multi-operand BD converter

An example to convert a binary number to two BCD digits ($b_0$ and $D_0$) is illustrated in Fig.2.27(b). Since the binary number $(1010011)_2$ to be converted here is larger than $(19)_{10}$, four Nicoud cells are required to realize the converter. As illustrated in Fig.2.27(b), the input to the Nicoud cell is restricted to $(1001)_2$. Hence the 3 MSBs of binary input along with '0' prepended $(0101)_2$ is accepted as $b_j$ and the next significant binary input '0' as $b_i$ resulting in the outputs $(1)_2$ and $(0000)_2$. The 4-bit output $(0000)_2$ of cell 1 along with the next significant binary input '0' form input to cell 2, resulting in $(0)_2$ and $(0000)_2$. Similarly, the 4-bit output of each subsequent cell along with residual 1-bit binary input feeds the decimal input of the following cell resulting in higher (P) digit $(1000)_2$ and lower (Q) digit $(0011)_2$. In general, binary number of any operand width can be converted to decimal by a linear arrangement of Nicoud cells.

The limitation of Nicoud cells however is their latency and thus the delay of multi-operand BD converter increases with the size of the binary number. To mitigate this, a hybrid multi-operand BD converter is proposed in this work. A detailed discussion of the partial product reduction scheme is presented in Section 6.3 .

## 2.9 Conclusions

In this chapter, the necessary background material about the multipliers based on different number systems is presented. This knowledge is required to understand the subsequent chapters included in the thesis. The objective of this chapter was to provide a quick introduction to various architectures such as fixed width binary multipliers, logarithmic and BCD multipliers. The multipliers based on binary and logarithmic number offers a low power alternative solution in error resilience applications. On the other hand decimal arithmetic has been increasing used in the financial applications where precision is very important. Finally, the overall research approach followed in this thesis is presented.

# Chapter 3

# An Efficient Reconfigurable Binary Multiplier with 2-Dimensional bypassing

## 3.1 Introduction

Recent developments in digital signal processing (DSP) have necessitated development of reconfigurable binary multiplier architectures that can dynamically adapt to varying application needs [53]. For example, a typical digital system may need to switch between one application that requires 4-bit accuracy to another application that needs 8-bit accuracy. This could partially be compensated by having two multipliers, each of precise bit-width, and having smallest bit-width multiplier that is adequate for current multiplication. Though, this approach optimizes the multiplier in terms of delay, it results in area and power overhead in view of multiple multiplier instances. Thus, one of the objectives in this chapter is to design a data path component that can be configured to perform either one $N$ or two independent $N/2$ multiplication operations.

A wide variety of low-power array multiplier architectures exist in literature that are listed in [10, 11, 54]. A simple and straightforward method to save power in these multipliers is to use bypassing technique [10] which reduces the switching activity by avoiding unnecessary computations. The second objective of this chapter is to design novel two-dimensional bypassing computational cells and incorporate them into the reconfigurable multiplier mentioned above to reduce the switching activity further. Also, a reconfigurable Ladner-Fisher prefix adder that simplifies the final product computation is included in the multiplier.

The proposed multiplier architecture is described in section 3.2 while synthesis results of the performance of various multipliers are compared in section 3.3.

## 3.2 Proposed Reconfigurable Binary Multiplier Architecture

In this section, a bit-width aware reconfigurable multiplier architecture with two-dimensional bypassing is proposed. In normal operation mode, this reconfigurable multiplier performs 8-bit multiplication while for applications where accuracy can be relaxed, it can perform 4-bit multiplication with only a fraction of the energy of 8-bit multiplication being expended. Also, when performing two 4-bit parallel multiplication within a 8-bit multiplier, only one half of the logic is used.

Further, to reduce the dynamic power, a new 2-dimensional bypassing technique is incorporated into the multiplier architecture. The bypass technique uses selective disabling of computation cells when the column and/or row partial products are zero. This is achieved by incorporating the new bypassing computational cells that improve the power efficiency and also facilitate reconfigurability of the multiplier.

A block diagram of the proposed reconfigurable multiplier with bypass cells is shown in Fig.3.1 below. As can be seen, it has two inputs each of m-bit width. The partial product (PP) generation is accomplished using AND gates while the PP matrix reduction is achieved using novel 2-dimensional bypass adder cells. The bypass adder logic, while reducing the PPs into two rows, also helps in minimizing the latency and power dissipation by disabling the unnecessary PPs. In addition, capability to reconfigure is built into the bypass computation cells. To further improve the speed of operation, two rows are reduced to a final product using a scalable Ladner-Fisher prefix adder.

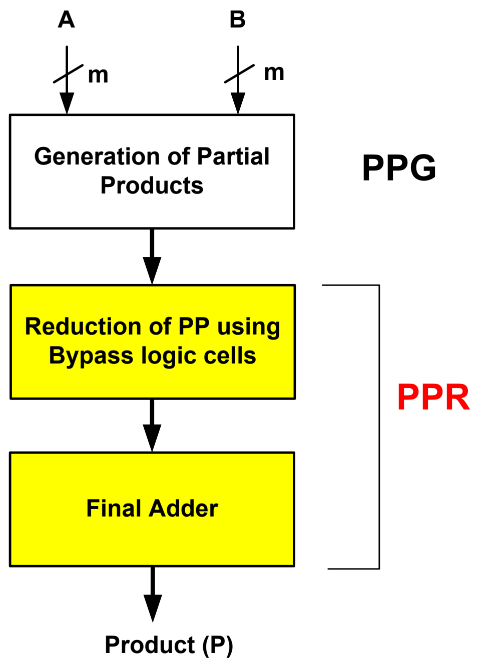

A
B
m
m
Generation of Partial Products
PPG
Reduction of PP using Bypass logic cells
Final Adder
PPR
Product (P)

Figure 3.1: Block diagram of proposed reconfigurable binary multiplier

### 3.2.1 Partial Product Arrangement

Figure.3.2 illustrates the partial product arrangement in a 8*8 array multiplier while their reduction is shown in Section 3.2.2. Based on the configuration mode given in Table 3.1, the partial product matrix in Fig.3.2 can be configured to perform either as one $8*8$ multiplier or two independent $4*4$ multipliers.

Table 3.1: Proposed configuration modes of an array multiplier

| Configuration Mode, CM | Function Description |
|---|---|
| $CM1$ | one 8*8 full-width multiplier |
| $CM0$ | two independent 4 * 4 full-width multipliers |

#### 3.2.1.1 Configuration Mode 1 (CM1) :

In configuration mode 1, one $8*8$ multiplication can be performed as illustrated in Fig.3.2. The PPG operation generates a total of 64 PPs, which are arranged in a matrix form as shown in Fig.3.2. To eliminate the redundant switching activity, all the PPs are reduced using bypass computation cells as discussed in section 3.2.2.

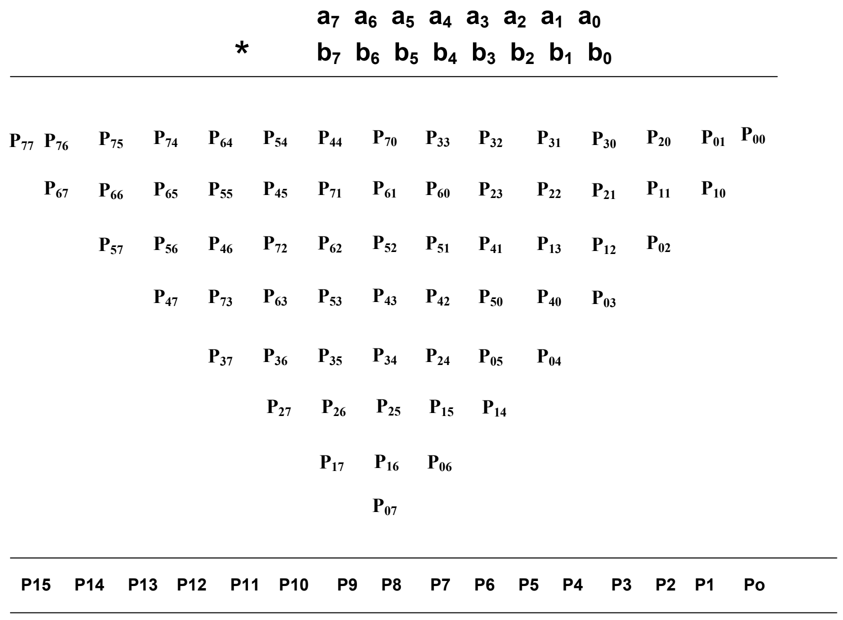

a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P77 P76 P75 P74 P64 P54 P44 P70 P33 P32 P31 P30 P20 P01 P00
P67 P66 P65 P55 P45 P71 P61 P60 P23 P22 P21 P11 P10
P57 P56 P46 P72 P62 P52 P51 P41 P13 P12 P02
P47 P73 P63 P53 P43 P42 P50 P40 P03
P37 P36 P35 P34 P24 P05 P04
P27 P26 P25 P15 P14
P17 P16 P06
P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po

Figure 3.2: Partial product matrix in configuration mode 1

#### 3.2.1.2 Configuration Mode 0 (CM0) :

In configuration mode 0, two parallel $4*4$ multiplications can be performed as illustrated in Fig.3.3. The partial products (colored in gray and black) that contribute to independent $4*4$ multiplications are given in the upper half of the PP matrix while those (enclosed in dotted box in Fig.3.3) that do not contribute to the product are shown in the lower half of the PP matrix. The computation cells corresponding to these PPs are turned-off, thus leading to power saving. This has been achieved using new bypass computation cells discussed below.

### 3.2.2 Partial Product Reduction using 2-Dimensional Bypass Cells

Figure 3.4 illustrates the proposed bypassing architecture for an 8*8 array multiplier. The reduction of partial products generated (shown in Fig.3.3) is carried out using the new adder bypass logic cells besides the existing cells. The bypassing cells incorporated into the multiplier skip the redundant computations whenever the row (horizontal) or the column (vertical) partial product is zero. Two adder cells namely, TDBA and MRBA with bypassing capability, are adapted from previous design [15] for this purpose and used. The TDBA cell has the capability to bypass when either row or column element is zero while MRBA bypasses when only the row element is zero.

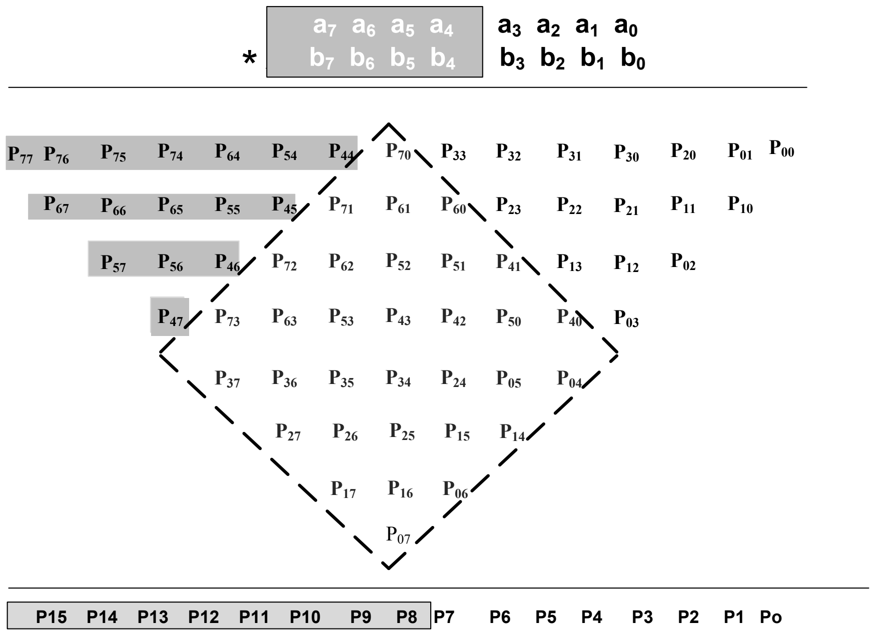

a7 a6 a5 a4 a3 a2 a1 a0
* b7 b6 b5 b4 b3 b2 b1 b0
P77 P76 P75 P74 P64 P54 P44 P70 P33 P32 P31 P30 P20 P01 P00
P67 P66 P65 P55 P45 P71 P61 P60 P23 P22 P21 P11 P10
P57 P56 P46 P72 P62 P52 P51 P41 P13 P12 P02
P47 P73 P63 P53 P43 P42 P50 P40 P03
P37 P36 P35 P34 P24 P05 P04
P27 P26 P25 P15 P14
P17 P16 P06
P07
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 Po

Figure 3.3: Partial product matrix in configuration mode 0

In mode 0 (CM 0), the bypass cells are intended to provide reconfigurability to the multiplier besides minimizing the redundant switching activity. Therefore, two new adder cells, reconfigurable two-dimensional bypass cell (RTDBC) and reconfigurable row-bypass cell (RRBC) are proposed in this work. These reconfigurable computation cells can be configured according to the mode of operation using configuration mode bit, *CM*. The RTDBC cell is deactivated when the corresponding row or column partial product is zero. Conversely, RRBC skips the computation unit when the corresponding row partial product is zero. A detailed implementation of these cells is provided in Section 3.2.2.1 & 3.2.2.2. With the proposed RTDBC and RRBC, the power saving of the multiplier is higher than the conventional bypassing architectures as will be demonstrated later.

The operation of the multiplier that depends on the configuration mode can be described as follows: When *CM* = ‘1’, one 8*8 full-width multiplication is carried out. Since in this mode all the PPs contribute to the final result $(P0-P15)$ it can be said that RTDBC and RRBC work as normal bypass logic cells. The reconfigurable cells along with TDBA and MRBA are turned off that depends on the nullity of partial products.

Conversely, when *CM* = ‘0’, two independent 4*4 multiplications can be performed. In this mode, the partial products enclosed in dotted box in Fig.3.3 do not contribute to the final result.

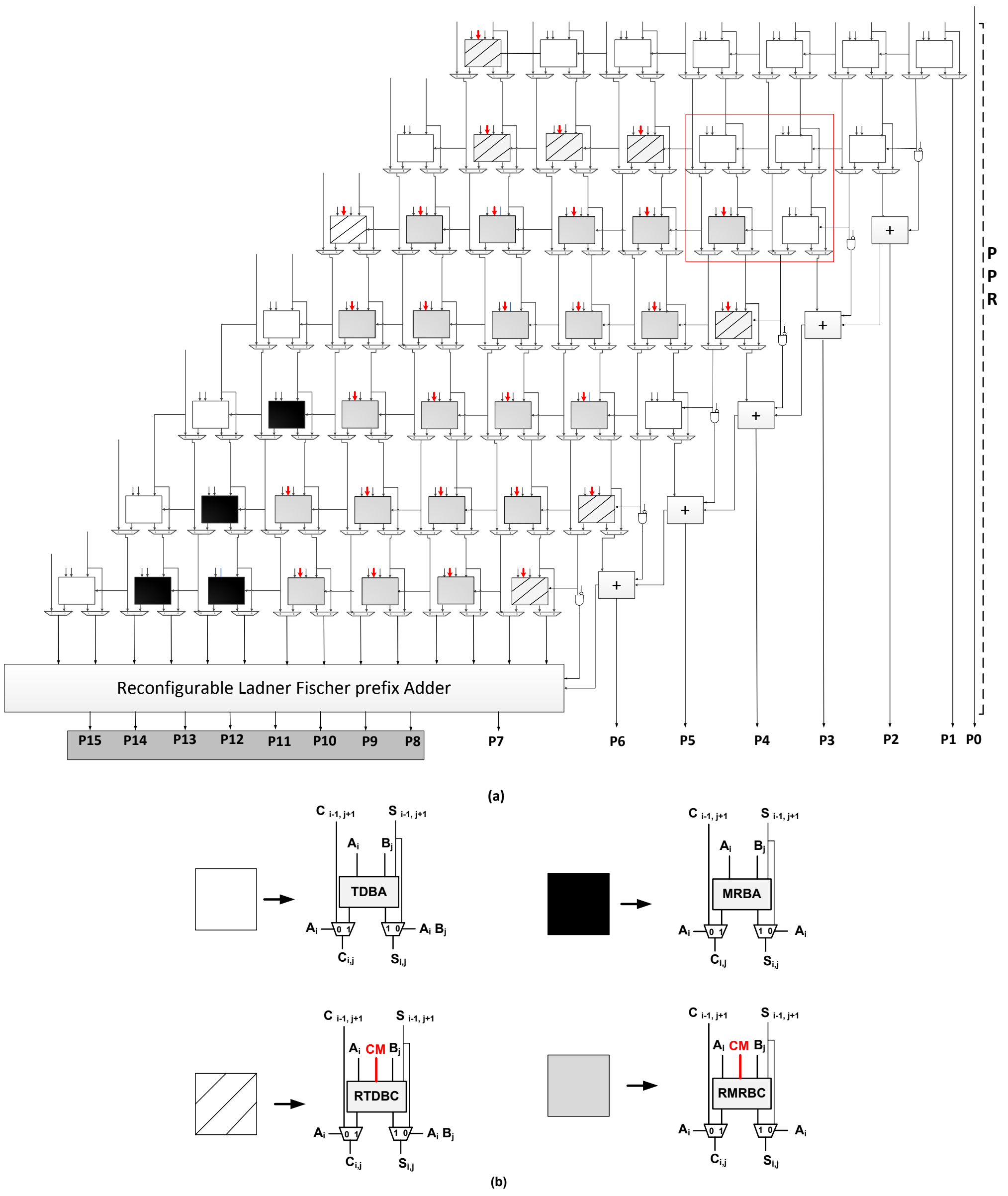

PPR
Reconfigurable Ladner Fischer prefix Adder
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
(a)
TDBA
MRBA
RTDBC
RMRBC
CM
(b)

Figure 3.4: (a) Proposed reconfigurable multiplier architecture with bypass computation cells (b) Reconfigurable row and column bypass cells with mode bit, CM

Therefore, RTDBC and RRBC computation cells corresponding to these PP bits are disabled in Fig.3.4. This helps in minimizing the unnecessary transition activity in the multiplier.

Finally, the PPs out of the bypass cells are accumulated to two rows regardless of the mode of operation. These rows are eventually reduced to final product using a scalable Ladner-Fisher prefix adder discussed in section 3.2.3.

The reason for implementing the proposed multiplier with different cells (two dimensional and row bypassing) is to avoid carry problem that occurs when both row and column-bypassing are applied simultaneously. For example, in Fig.3.5 (a section of proposed multiplier high-

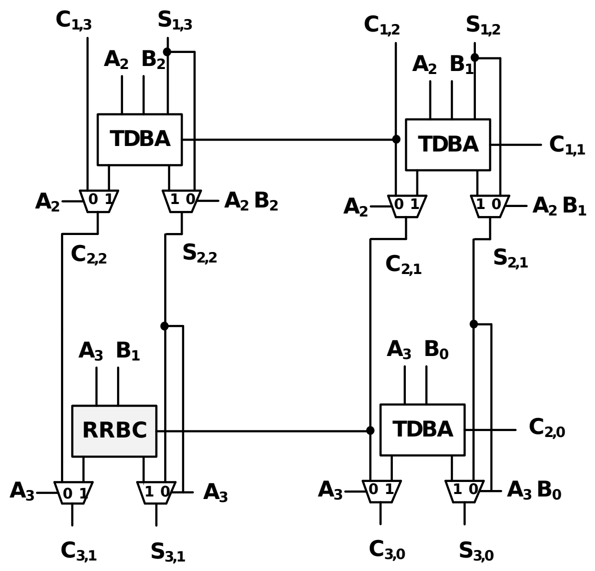


Figure 3.5: A section of the proposed reconfigurable multiplier

lighted in dotted box in Fig.3.4), assume both column input $B_1$ and row input $A_2$ are zero and the carry-input $C_{1,2}$ is '1'. In such a case, the respective computation cell will be disabled and carry output $C_{2,1}$ is '1' due to bypassing. If $A_3$ is '1', $C_{2,1}$ is lost due to column-bypassing. This introduces errors in the multiplication. To overcome this problem, the proposed multiplier has 2-dimensional cells (TDBC / RTDBC) only in the first two rows and the first and the last columns in which the carry problem does not occur, as illustrated in Fig.3.4. The remaining portion of the multiplier should have different types of logic cells which are different from RT-DBCs. The carry problem is solved using row-bypassing cells (MRBA/RRBC) built with less number of logic gates.

### 3.2.2.1 Reconfigurable Two-Dimensional Bypass Cell (RTDBC)

The bypassing scheme in existing two-dimensional bypass approaches [12,54] consists of a full adder and additional logic circuitry. The reconfigurable two-dimensional bypass cell (RTDBC) in this work however is designed as per the Table 3.2. Clearly, the truth table of the RTDBC cell is simple and can be implemented only with a few logic gates. In RTDBC, the unnecessary logic computations are disabled using internal tri-state buffers (ITBs). The EX-OR used in this cell is a 4-transistor type with cascaded inverter for driving the output [55]. As a result, the area and power overhead are reduced.

Figure 3.6 shows a schematic of the reconfigurable two-dimensional bypass cell. As illus-

Table 3.2: Truth Table of reconfigurable two-dimensional bypass adder cell (RTDBC)

| $CM$ | A | B | $C_{out}$ | $S_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | $C$ | $S_{in}$ |
|  | 0 | 1 | $C$ | $S_{in}$ |
|  | 1 | 0 | $C$ | $S_{in}$ |
|  | 1 | 1 | $C$ | $S_{in}$ |
| 1 | 0 | 0 | $C$ | $S_{in}$ |
|  | 0 | 1 | $C$ | $S_{in}$ |
|  | 1 | 0 | 0 | $S_{in}$ |
|  | 1 | 1 | $C_{in}+S_{in}$ | $C_{in} \odot S_{in}$ |

trated in Fig. 3.6, ITBs that are augmented with 2-input NAND gate and inverters form the input to the MUXs. When the input $CM$ is ‘0’, both the NAND and inverter logic are disabled and the MUX logic passes $C$ and $S_{in}$ to the outputs $C_{out}$ and $S_{out}$ respectively. For the case when $CM$ and the row input A is ‘1’, the output $C_{out}$ depends on the column input $B$ as shown in Fig. 3.6 . However, when row input $A$ is ‘0’, the inverter logic gates are disabled and the output $C_{out}$ depends on input $C$ while $S_{out}$ follows the input $S_{in}$. Hence, using RTDBC, power consumption can be reduced.

#### 3.2.2.2 Reconfigurable Row Bypass Cell (RRBC)

The reconfigurable row bypass cell (RRBC) has the capability to bypass when the row element is zero. The truth table of the RRBC illustrated in Table 3.3 is simple and can be implemented with a few logic gates instead of using a full adder and additional logic. Further, RRBC does not have the carry problem which is unlike RTDBC. To reduce the power consumption in RRBC, ITBs are used to disable computation in cases where it is unnecessary. Figure 3.7 illustrates the schematic of RRBC.

Table 3.3: Truth Table of reconfigurable row-bypass cell (RRBC)

| $CM$ | $A$ | $B$ | $C_{out}$ | $S_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | $C$ | $S_{in}$ |
| 0 | 0 | 1 | $C$ | $S_{in}$ |
| 0 | 1 | 0 | $C$ | $S_{in}$ |
| 0 | 1 | 1 | $C$ | $S_{in}$ |
| 1 | 0 | 0 | $C$ | $S_{in}$ |
| 1 | 0 | 1 | $C$ | $S_{in}$ |
| 1 | 1 | 0 | $C_{in}.S_{in}$ | $C_{in} \oplus S_{in}$ |
| 1 | 1 | 1 | $C_{in}+S_{in}$ | $C_{in} \odot S_{in}$ |

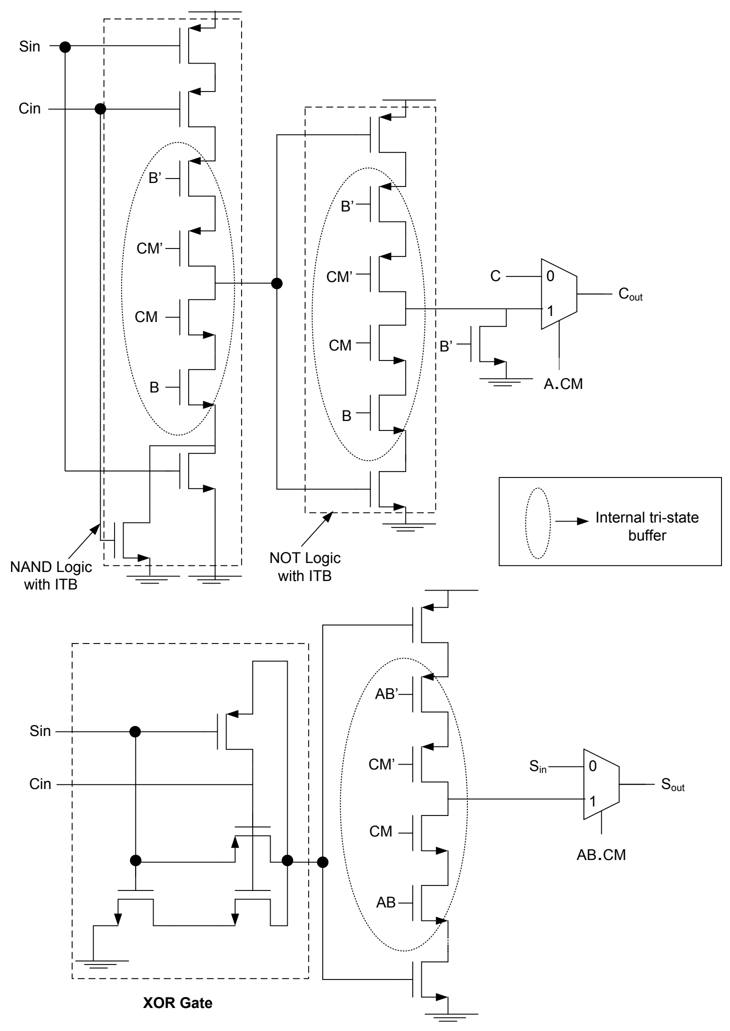


Figure 3.6: Logic schematic of RTDBC

A typical RRBC consists of ITBs, 2-input NOR gate, 2-input NAND gate, and inverters in front of MUXs. The EX-OR and EX-NOR gates used in this cell are 4-transistor type with cascaded inverter for driving the output [55]. When the input *CM* is '0', the ITB disables the logic gates and the MUX logic passes *C* and $S_{in}$ to the outputs $C_{out}$ and $S_{out}$ respectively. For the case when *CM* is '1' and the row input *A*, the output $C_{out}$ depend on the column input *B* as shown in Fig.3.7. However, when row input *A* is '0', the inverter logic gates are disabled and the output $C_{out}$ depends on input *C* while $S_{out}$ follows the input $S_{in}$.

### 3.2.3 Reconfigurable Ladner-Fisher Prefix Adder

The partial products are reduced to two rows using bypass computation cells at PPR level and are converted to a product using a final adder. This adder implementation is based on Ladner-

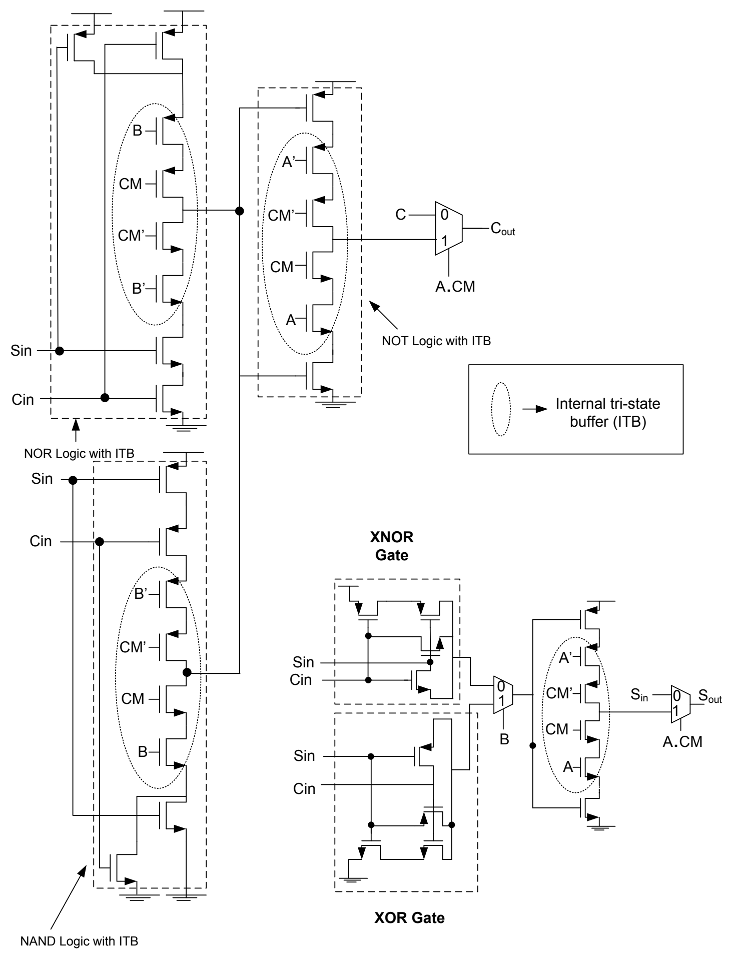


Figure 3.7: Logic schematic of RRBC

Fischer [9] adder architecture that is known to have a good trade-off between area and the performance .

The Ladner-Fischer architecture illustrated in Fig.3.8 is modified to support either a one 8-bit addition or two 4-bit additions. Since the focus in this work is reconfigurability, an AND gate is inserted in the carry propagation path and one of its inputs is connected to configuration mode $(CM)$. If the control bit $CM$ is '1', the adder will operate as a one 8-bit adder while if it is '0', the carry propagation is broken and the adder operates as two independent 4-bit adders. This adder facilitates the proposed multiplier to carry out either one 8-bit multiplication or two 4-bit multiplications.

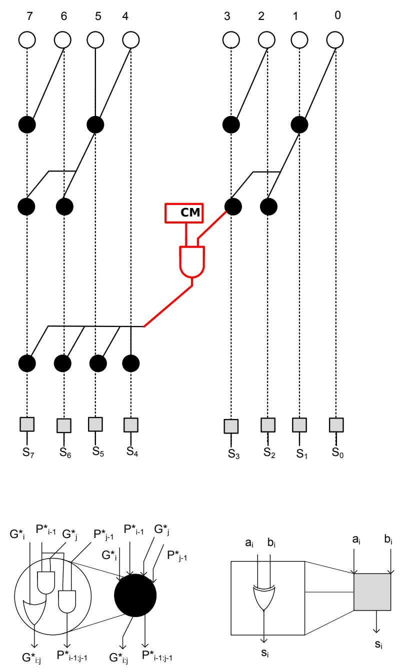


Figure 3.8: A reconfigurable 8-bit Ladner-Fisher prefix adder

## 3.3 Simulation and Synthesis

Detailed simulations of the proposed multiplier have been carried out and a comparison with similar designs existing in the literature has been made. For a fair comparison, existing row-bypass [10] and column-bypass [11] designs are independently combined with twin-precision [14] to form a multiplier. For example, twin row-bypass multiplier is obtained by incorporating row-bypass logic into twin-precision multiplier. Similarly, twin column-bypass is formed using column bypass logic with twin-precision multiplier.

All the multiplier designs with bypass schemes given in Table 3.4 including the proposed one have been described structurally using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1. These multipliers have been mapped on to TSMC 180 nm technology slow-normal library (operating conditions 1.8 V, 25°C) using cadence RTL compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating dynamic power.

Table 3.4 presents performance metrics such as area, delay, power, and power-delay product for the 8-bit reconfigurable multiplier with different bypass schemes including the proposed one. Figures 3.9-3.12 provide a graphical comparison of the same.

Table 3.4: Area, delay and power of various multipliers with various bypassing schemes

| Mode | Multiplier | Area ($\mu m^2$) | % change | Delay (ps) | % change | Power (nW) | % change | Power -Delay product ($10^6$)J | % change |
|---|---|---|---|---|---|---|---|---|---|
| CM1 | Twin precision [14] | 586 | 94% | 2094 | 134% | 13454 | 105% | 28.1 | 140% |
| | Twin Row [10] | 660 | 105% | 2013 | 127% | 13254 | 103% | 26.6 | 132% |
| | Twin Column [11] | 630 | 101% | 1960 | 123% | 13142 | 102% | 25.7 | 128% |
| | Proposed | 625 | 100% | 1560 | 100% | 12838 | 100% | 20 | 100% |
| CM0 | Twin precision [14] | 586 | 94% | 2094 | 140% | 13454 | 129% | 28.1 | 181% |
| | Twin Row [10] | 660 | 105% | 1905 | 127% | 11073 | 106% | 21 | 136% |
| | Twin Column [11] | 630 | 101% | 1844 | 123% | 10867 | 104% | 20 | 129% |
| | Proposed | 625 | 100% | 1492 | 100% | 10379 | 100% | 15.48 | 100% |

It is clear from the Table 3.4 and Fig.3.9 that the proposed multiplier in mode 1 (CM1) achieves 25 to 34% lesser delay compared to the existing designs.

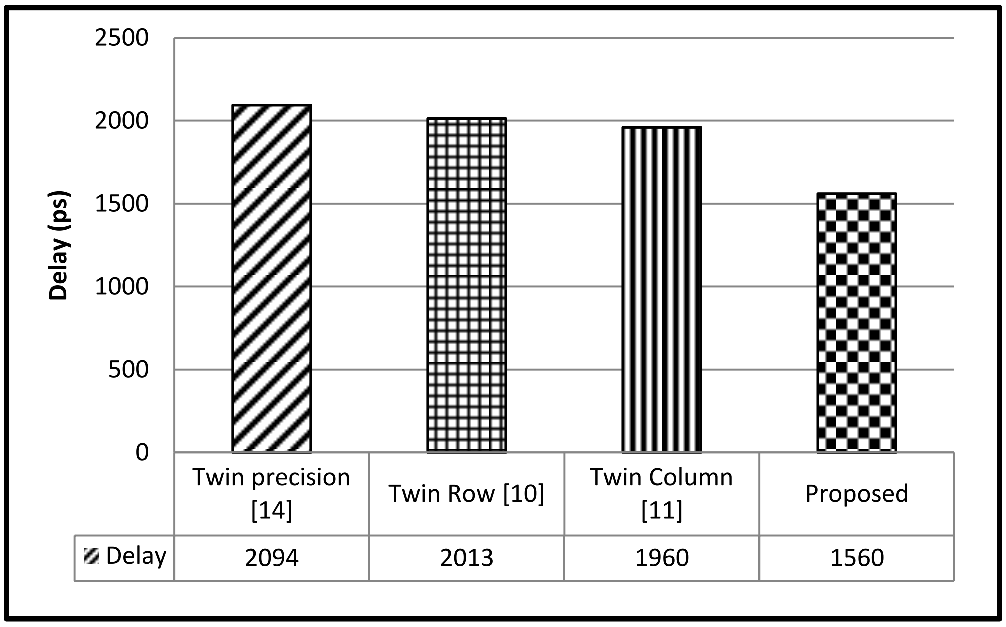
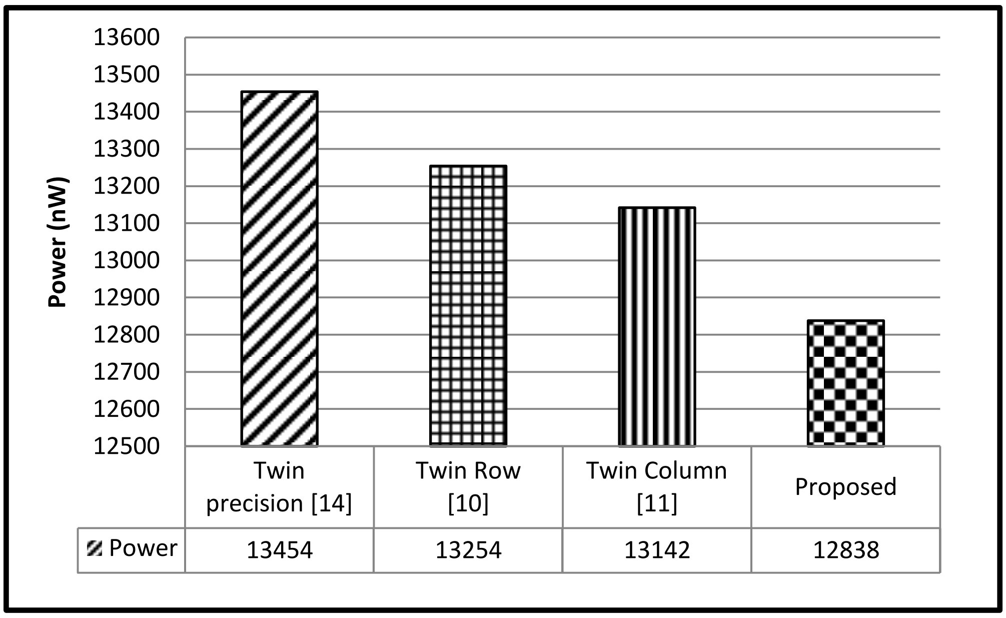


Figure 3.9: Latency of various bypass multiplier schemes

It can also be seen from the Table 3.4 (CM1) and Fig.3.10 that the proposed design consumes 2 to 5% lesser power than the existing ones. This reduction in power dissipation is due to the reduction in switching activity arising out of disabling the computational units.

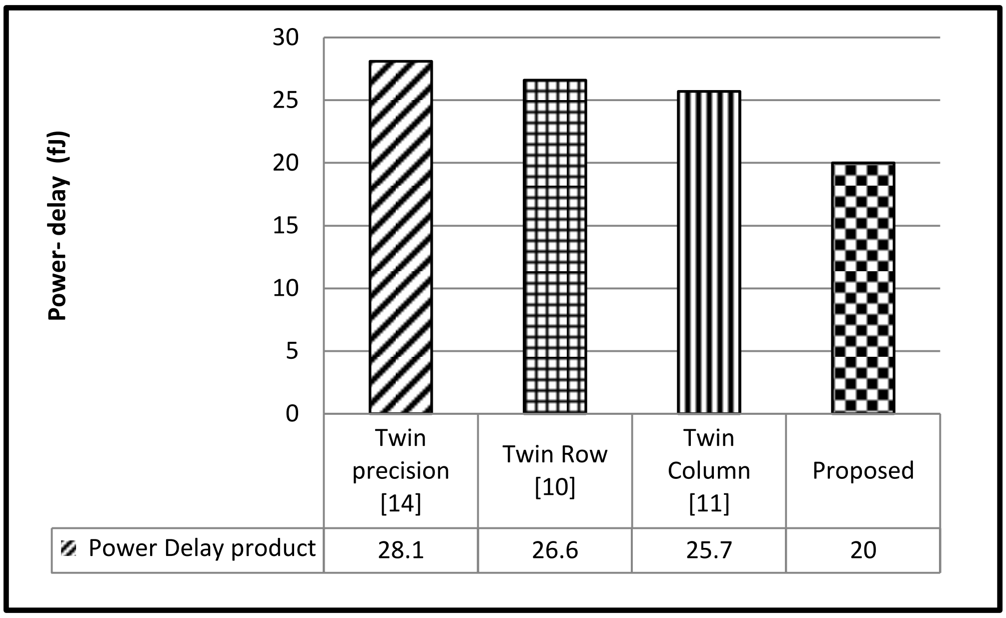
From Table 3.4 (CM1) and Fig.3.11, it can be observed that there is also an improvement of 28 to 40% in power-delay product. This is due to the saving achieved in both power and delay



| | Twin precision [14] | Twin Row [10] | Twin Column [11] | Proposed |
|---|---|---|---|---|
| ▨ Power | 13454 | 13254 | 13142 | 12838 |

Figure 3.10: Comparison of various bypass multiplier schemes in terms of power

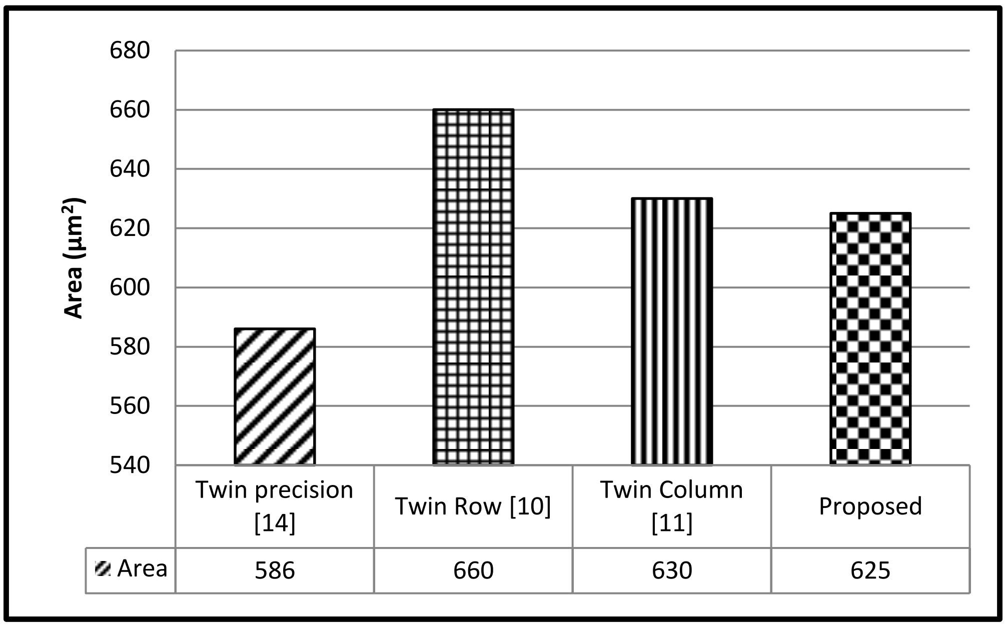
mentioned above.Thus the proposed multiplier with a new 2-dimensional bypass scheme is found to be better in terms of power, delay and power-delay product when compared to earlier designs.



| | Twin precision [14] | Twin Row [10] | Twin Column [11] | Proposed |
|---|---|---|---|---|
| ▨ Power Delay product | 28.1 | 26.6 | 25.7 | 20 |

Figure 3.11: Power-delay product of various bypass multiplier schemes

It is evident from the Table 3.4 (CM1) and Fig.3.12 that the area overhead of row bypass scheme alone is 9% while this increase is only 6% in the proposed design even after incorporating both row and column bypassing logic. This is because of the usage of efficient adder bypass cells which take up less area and are also fast.



| | Twin precision [14] | Twin Row [10] | Twin Column [11] | Proposed |
|---|---|---|---|---|
| ▨ Area | 586 | 660 | 630 | 625 |

Figure 3.12: Comparison of various bypass multiplier schemes in terms of area

It can also be seen from the Table 3.4 (CM0) that the proposed design achieves 23-40% and 4-29% lesser delay and power respectively compared to the existing designs in mode 0 (CM0). Also, an improvement of 29 to 80% in power-delay product is achieved. This reduction of delay is because of the new bypass adder cells that have shorter critical path compared to the conventional bypass cells. Another reason for the improvement in delay is due to bypassing of the adder cells whenever the partial products in the PP matrix are zero. Accordingly, the more the number of zeroes are in the PP matrix, the faster will be the design. For this reason, the power improvement achieved in mode 0 is much larger compared to mode 1.

## 3.4 Conclusions

In this chapter, a reconfigurable multiplier with two dimensional bypassing scheme has been proposed. In normal operation mode, the reconfigurable multiplier performs 8-bit multiplication. For applications where accuracy can be relaxed, the multiplier can perform 4-bit multiplication while expending only a fraction of the energy of a conventional 8-bit array multiplier. Also, when performing two 4-bit parallel multiplications within an 8-bit multiplier only one half of the logic is used.

Further, to reduce the dynamic power, a new 2-dimensional bypassing technique is incorporated into the multiplier architecture. The bypass technique uses selective disabling of computation cells when the column and/or row partial products are zero. This is achieved by incorporating the new bypassing computational cells ( RTDBC and RMRBC) that improve the power efficiency and also facilitate reconfigurability of the multiplier.

The proposed multiplier with the new 2-dimensional bypass scheme performs better than other designs in terms of delay (a reduction of 34%) and power (a reduction of 5% ) resulting in a overall reduction of 40% in power-delay product. The delay advantage in the proposed design is due to the RTDBC and RMRBC cells that are simple and take up less logic unlike in the existing designs.

# Chapter 4

# An Improved Fixed-Width Recursive Binary Multiplier

## 4.1 Introduction

Most of the signal and image processing applications possess an inherent quality of error resilience and hence can tolerate error up to a certain limit in computations [56]. In such applications, savings in power are achieved by pruning the data path units [57] such as truncating a multiplier. Truncation however may lead to errors in computing and therefore it‘s always a challenge between the amount of error that can be tolerated in an application and the advantage that can be obtained in terms of it‘s implementation as reflected by the parameters such as area, latency and power. Thus, the focus of this chapter is to implement and validate a fixed-width multiplier with improved efficiency for error resilient applications.

A large variety of fixed-width (or truncated) multiplier designs has been proposed in the literature and found to be a potential solution for efficient implementation. In these designs, most of which are either array or recursive-based [17,18], the least significant bits (LSBs) of the partial product matrix are removed and an error compensation function is added in their place [21–23]. Multiplier designs based on recursive technique are faster and have the regularity of array multipliers making them the most suitable candidate for VLSI implementation [19].

In this work, a recursive binary multiplier architecture based on Karatsuba algorithm [16]

is chosen. This architecture possesses an inherent hierarchical structure that consists of several sub-multipliers making it suitable for fixed-width applications [18]. Thus, rather than modifying the entire multiplier structure, the sub-multiplier in the least significant position can be removed and replaced with a data-dependent correction term. Following this, a new correction scheme is developed in this work for fixed-width recursive multipliers. This scheme enables tuning of the accuracy through a new error compensating function achieved in a systematic manner. The proposed and the existing truncation schemes have been applied on an image sharpening algorithm and compared in the context of certain well-known image processing benchmarks such as Lena, Cameraman and Pirate for performance.

The rest of the chapter is organized as follows. The proposed fixed-width recursive multiplier architecture is described in Section 4.2. Performance analysis of both the proposed and the existing multiplier architectures is carried out and compared in Section 4.3. They are used in an image sharpening algorithm to quantify their performance, results of which are provided in Section 4.4.

## 4.2 A New Approach to Error Correction in Fixed-Width Recursive Multipliers

The proposed architecture is based on the fact that the error after removing the least significant sub-multiplier $A_LB_L$ (which contributes minimum to the final result) is always positive and hence the multiplication product obtained is less than the exact value. Error due to this approximation is further reduced by using a data-dependent correction function that is based on the least significant partial product columns within $A_LB_L$ that are not formed.

A typical recursive multiplication of two inputs $A$ and $B$, each of $2n$-bit width, results in four sub-multipliers as illustrated in Fig.4.1. The PPs generated in these sub-multipliers are reduced using the partial product reduction structure [5] as mentioned earlier.

The product $(P_{exact})$ after multiplication is given by,

$$P_{exact} = P_{H1} + P_{H2} + P_{H3} + P_L \tag{4.1}$$

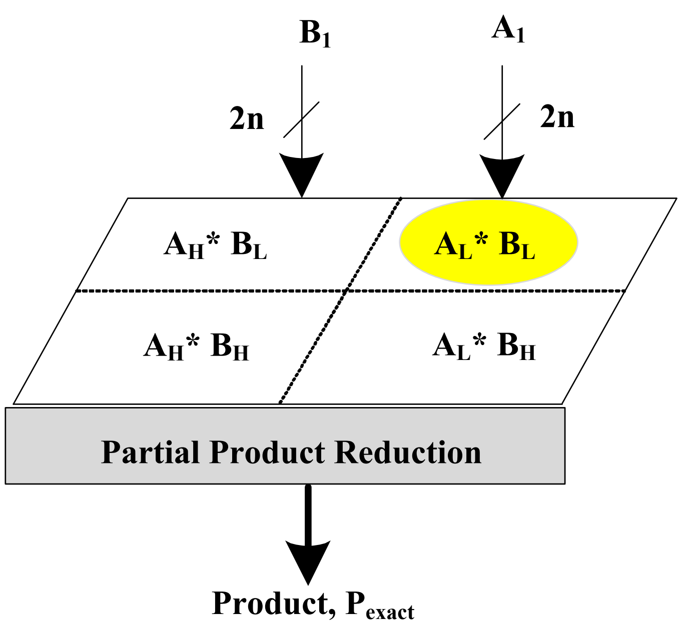


Figure 4.1: A 2n*2n recursive multiplication structure illustrating sub-multipliers

Where $P_{H1} = A_H B_H$, $P_{H2} = A_H B_L$, $P_{H3} = A_L B_H$ and $P_L = A_L B_L$

In the proposed scheme, as in the other scheme, the sub-multiplier $(A_L B_L)$ that contributes minimum to the final result is removed. Further, error due to this approximation is reduced using a data-dependent correction function $(CF_0)$ which is based on the least significant partial product columns within $A_L B_L$ that are not formed.

The expression for approximate product $(P_{approx})$ after removing $A_L B_L$ and subsequent correction is as follows:

$$P_{approx} = P_{H1} + P_{H2} + P_{H3} + CF_0 \tag{4.2}$$

Since the correction function $(CF_0)$ does not include all the PP columns of $A_L B_L$, it leads to an error. Further, the error magnitude and the hardware complexity depend on the function selected.

The error due to replacement of $A_L B_L$ using the correction function is determined as,

$$Error, e_0 = A_L B_L - CF_0 \tag{4.3}$$

Where

$$A_L B_L = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} a_i b_j 2^{i+j} \tag{4.4}$$

The operation of the proposed approach is demonstrated on a 8*8 recursive multiplier and error analysis carried out. Figure.4.2 illustrates four sub-multipliers and their corresponding partial products in a 8*8 recursive multiplier. As mentioned earlier, the sub-multiplier $A_L B_L$ is removed and replaced with the correction function $(CF_0)$.

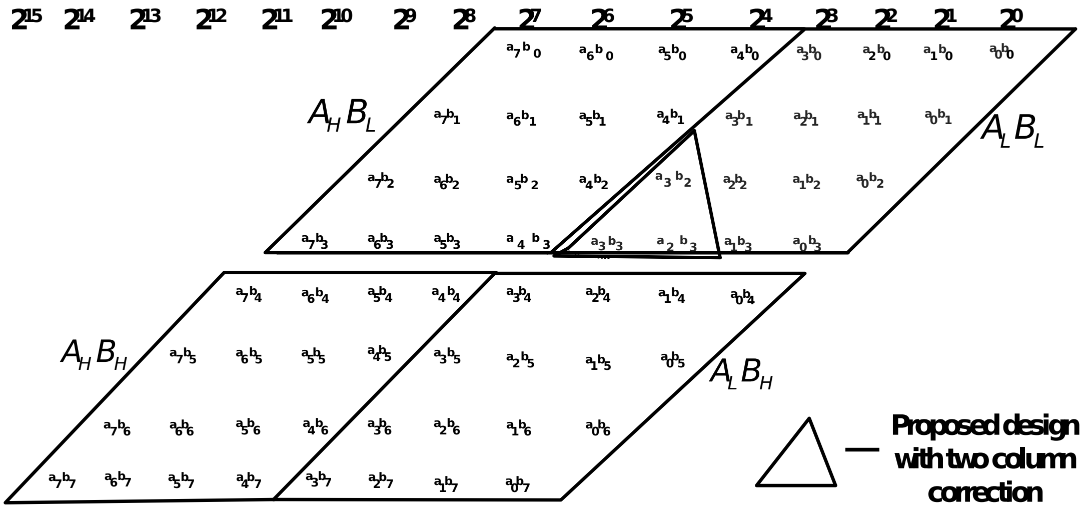


Figure 4.2: Partial product matrix of a fixed-width recursive multiplier with two most significant columns considered for correction

The proposed approach considers the most significant partial product columns of $A_L B_L$ for the correction function. Based on the number of columns considered, the accuracy and hardware requirements vary. The variation of average error with number of columns (considered for correction) is carried out on a 8*8 recursive binary multiplier mentioned above, as illustrated in Fig.4.3. The plot shows the variation of average error as a function of the number of most significant partial product columns of $A_L B_L$. It can be observed from the figure that the average error reduces with the increase in number of partial product columns, though with an area overhead. Further, it can be noted that the average error drops very sharply with only two most significant columns considered. Beyond this point, even with the inclusion of more partial product columns the rate at which the average error decreases is less.

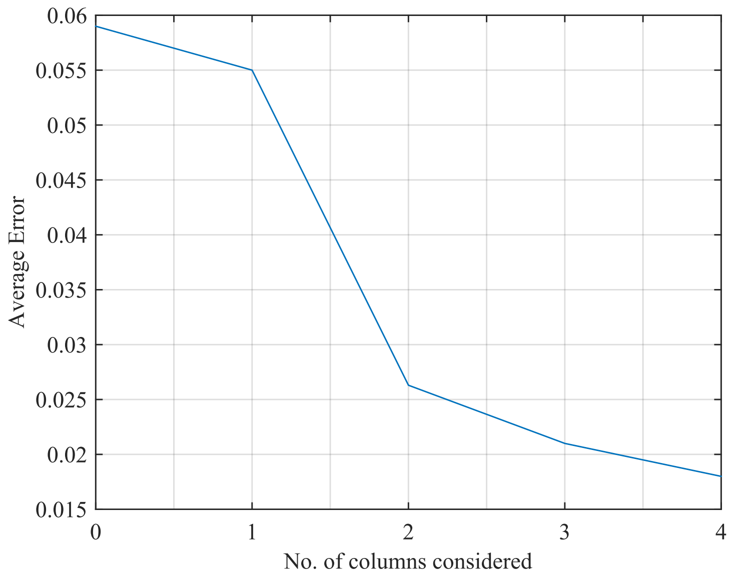


Figure 4.3: Average error Vs Number of most significant partial products columns considered for correction in a 8 * 8 fixed-width recursive multiplier

### 4.2.1 Error Analysis for the proposed correction function

For the sake of understanding, error analysis when two most significant columns of $A_LB_L$ (highlighted in a triangle in Fig.4.2) are considered for correction, is given below. The advantage of the proposed method is that the partial product columns selected for correction have higher weight when compared to those in the existing schemes. The correction function proposed in this work thus tends to achieve better accuracy. The same is proved in Section 4.3.

A step-by-step implementation of unsigned multiplication using existing and the proposed correction (where two most significant partial product columns are considered) schemes is given in the numerical example 4.1 below .

**Example 4.1.**

$A = (25)_{10} = (00011001)_2$

$B = (20)_{10} = (00010100)_2$

Actual Product, $P_{exact} = A * B = (500)_{10} = (0000000111110100)_2$

Product using recursive multiplier with average value correction terms, Scheme 2 [18] :

$P_{approx2} = (475)_{10} = (0000000111011011)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx2})}{P_{exact}} = \frac{(500-475)}{500} * 100 = 5$

Product using recursive multiplier with 1-bit correction, Scheme 3 [18] :

$P_{approx1} = (464)_{10} = (0000000111010000)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx1})}{P_{exact}} = \frac{(500-464)}{500} * 100 = 7.2$

Product using recursive multiplier with proposed correction (two most significant columns considered for correction) scheme :

$P_{approx} = (0000000111110000)_2 = (496)_{10}$

Average error (%) = $\frac{(P_{actual} - P_{approx})}{P_{exact2}} = \frac{(500-496)}{500} * 100 = 0.8$

It can be observed from the above analysis that the proposed correction with two most significant columns considered for correction achieves better accuracy when compared to the existing schemes.

#### 4.2.1.1 Hardware Implementation of the Proposed Fixed-Width Recursive Multiplier

The hardware block diagram of a $8*8$ fixed-width recursive multiplier is illustrated in Fig.4.4. The sub-multipliers $A_HB_H, A_LB_H, A_HB_L$ are implemented using 4*4 binary array multipliers [3] while the compensation function essentially includes the two most significant partial product columns highlighted in triangle in Fig.4.2. The PPs corresponding to sub-multipliers along with the correction function are reduced using a tree carry save adders and final adder. A detailed PP reduction mechanism is explained below.

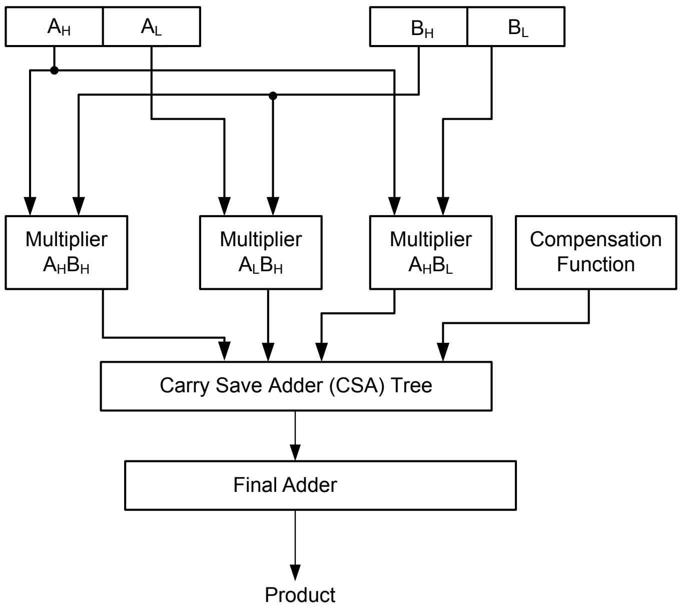


Figure 4.4: A new fixed-width recursive multiplier hardware

# Chapter 4

# An Improved Fixed-Width Recursive Binary Multiplier

## 4.1 Introduction

Most of the signal and image processing applications possess an inherent quality of error resilience and hence can tolerate error up to a certain limit in computations [56]. In such applications, savings in power are achieved by pruning the data path units [57] such as truncating a multiplier. Truncation however may lead to errors in computing and therefore it‘s always a challenge between the amount of error that can be tolerated in an application and the advantage that can be obtained in terms of it‘s implementation as reflected by the parameters such as area, latency and power. Thus, the focus of this chapter is to implement and validate a fixed-width multiplier with improved efficiency for error resilient applications.

A large variety of fixed-width (or truncated) multiplier designs has been proposed in the literature and found to be a potential solution for efficient implementation. In these designs, most of which are either array or recursive-based [17,18], the least significant bits (LSBs) of the partial product matrix are removed and an error compensation function is added in their place [21–23]. Multiplier designs based on recursive technique are faster and have the regularity of array multipliers making them the most suitable candidate for VLSI implementation [19].

In this work, a recursive binary multiplier architecture based on Karatsuba algorithm [16]

is chosen. This architecture possesses an inherent hierarchical structure that consists of several sub-multipliers making it suitable for fixed-width applications [18]. Thus, rather than modifying the entire multiplier structure, the sub-multiplier in the least significant position can be removed and replaced with a data-dependent correction term. Following this, a new correction scheme is developed in this work for fixed-width recursive multipliers. This scheme enables tuning of the accuracy through a new error compensating function achieved in a systematic manner. The proposed and the existing truncation schemes have been applied on an image sharpening algorithm and compared in the context of certain well-known image processing benchmarks such as Lena, Cameraman and Pirate for performance.

The rest of the chapter is organized as follows. The proposed fixed-width recursive multiplier architecture is described in Section 4.2. Performance analysis of both the proposed and the existing multiplier architectures is carried out and compared in Section 4.3. They are used in an image sharpening algorithm to quantify their performance, results of which are provided in Section 4.4.

## 4.2 A New Approach to Error Correction in Fixed-Width Recursive Multipliers

The proposed architecture is based on the fact that the error after removing the least significant sub-multiplier $A_LB_L$ (which contributes minimum to the final result) is always positive and hence the multiplication product obtained is less than the exact value. Error due to this approximation is further reduced by using a data-dependent correction function that is based on the least significant partial product columns within $A_LB_L$ that are not formed.

A typical recursive multiplication of two inputs $A$ and $B$, each of $2n$-bit width, results in four sub-multipliers as illustrated in Fig.4.1. The PPs generated in these sub-multipliers are reduced using the partial product reduction structure [5] as mentioned earlier.

The product ($P_{exact}$) after multiplication is given by,

$$P_{exact} = P_{H1} + P_{H2} + P_{H3} + P_L \tag{4.1}$$

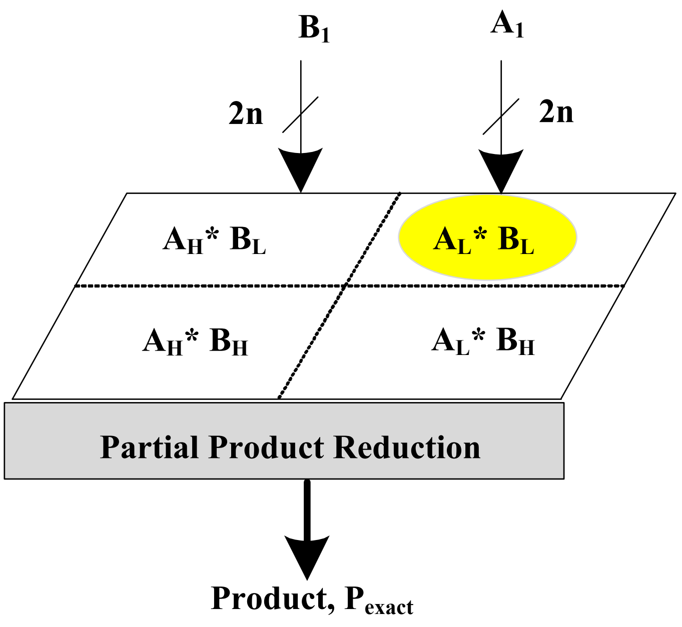


Figure 4.1: A 2n*2n recursive multiplication structure illustrating sub-multipliers

Where $P_{H1} = A_H B_H$, $P_{H2} = A_H B_L$, $P_{H3} = A_L B_H$ and $P_L = A_L B_L$

In the proposed scheme, as in the other scheme, the sub-multiplier $(A_L B_L)$ that contributes minimum to the final result is removed. Further, error due to this approximation is reduced using a data-dependent correction function $(CF_0)$ which is based on the least significant partial product columns within $A_L B_L$ that are not formed.

The expression for approximate product $(P_{approx})$ after removing $A_L B_L$ and subsequent correction is as follows:

$$P_{approx} = P_{H1} + P_{H2} + P_{H3} + CF_0 \tag{4.2}$$

Since the correction function $(CF_0)$ does not include all the PP columns of $A_L B_L$, it leads to an error. Further, the error magnitude and the hardware complexity depend on the function selected.

The error due to replacement of $A_L B_L$ using the correction function is determined as,

$$Error, e_0 = A_L B_L - CF_0 \tag{4.3}$$

Where

$$A_L B_L = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} a_i b_j 2^{i+j} \tag{4.4}$$

The operation of the proposed approach is demonstrated on a 8*8 recursive multiplier and error analysis carried out. Figure.4.2 illustrates four sub-multipliers and their corresponding partial products in a 8*8 recursive multiplier. As mentioned earlier, the sub-multiplier $A_L B_L$ is removed and replaced with the correction function $(CF_0)$.

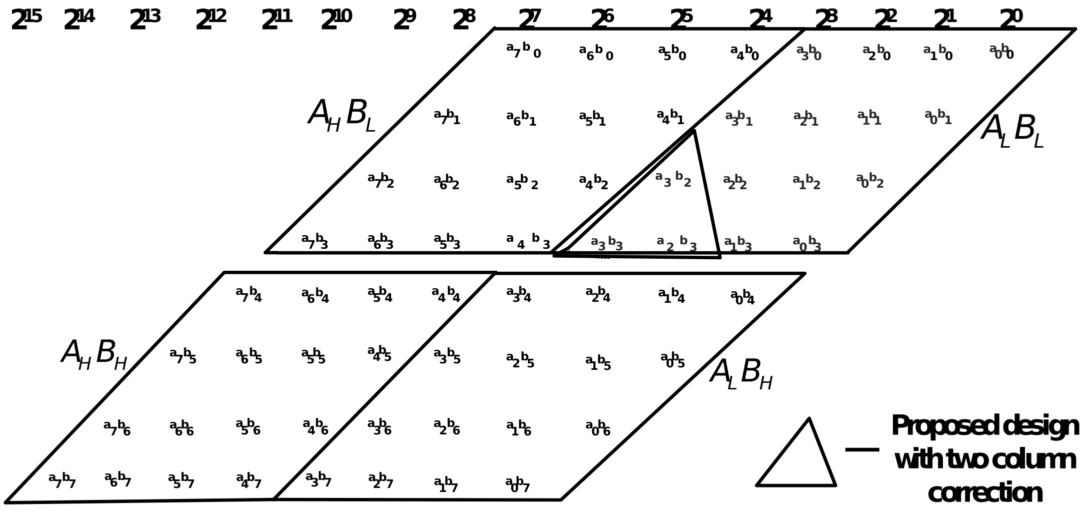


Figure 4.2: Partial product matrix of a fixed-width recursive multiplier with two most significant columns considered for correction

The proposed approach considers the most significant partial product columns of $A_L B_L$ for the correction function. Based on the number of columns considered, the accuracy and hardware requirements vary. The variation of average error with number of columns (considered for correction) is carried out on a 8*8 recursive binary multiplier mentioned above, as illustrated in Fig.4.3. The plot shows the variation of average error as a function of the number of most significant partial product columns of $A_L B_L$. It can be observed from the figure that the average error reduces with the increase in number of partial product columns, though with an area overhead. Further, it can be noted that the average error drops very sharply with only two most significant columns considered. Beyond this point, even with the inclusion of more partial product columns the rate at which the average error decreases is less.

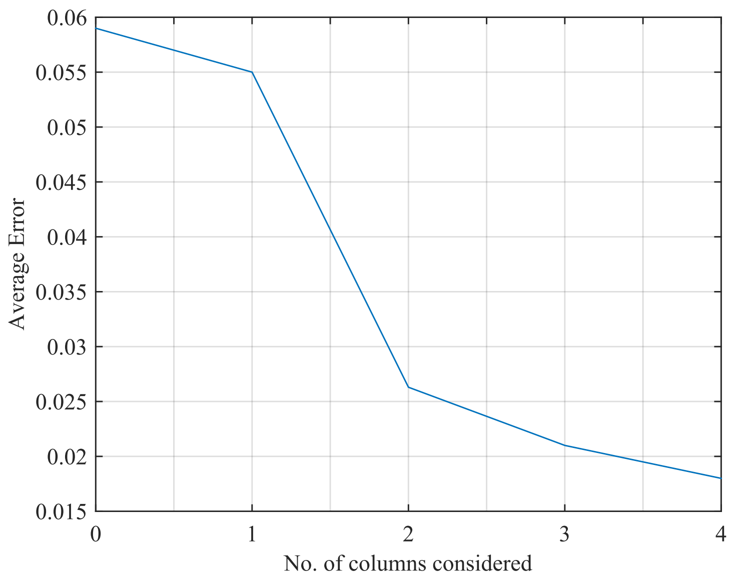


Figure 4.3: Average error Vs Number of most significant partial products columns considered for correction in a 8 * 8 fixed-width recursive multiplier

### 4.2.1 Error Analysis for the proposed correction function

For the sake of understanding, error analysis when two most significant columns of $A_L B_L$ (highlighted in a triangle in Fig.4.2) are considered for correction, is given below. The advantage of the proposed method is that the partial product columns selected for correction have higher weight when compared to those in the existing schemes. The correction function proposed in this work thus tends to achieve better accuracy. The same is proved in Section 4.3.

A step-by-step implementation of unsigned multiplication using existing and the proposed correction (where two most significant partial product columns are considered) schemes is given in the numerical example 4.1 below .

**Example 4.1.**

$A = (25)_{10} = (00011001)_2$

$B = (20)_{10} = (00010100)_2$

Actual Product, $P_{exact} = A * B = (500)_{10} = (0000000111110100)_2$

Product using recursive multiplier with average value correction terms, Scheme 2 [18] :

$P_{approx2} = (475)_{10} = (0000000111011011)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx2})}{P_{exact}} = \frac{(500-475)}{500} * 100 = 5$

Product using recursive multiplier with 1-bit correction, Scheme 3 [18] :

$P_{approx1} = (464)_{10} = (0000000111010000)_2$

Average error (%) = $\frac{(P_{exact} - P_{approx1})}{P_{exact}} = \frac{(500-464)}{500} * 100 = 7.2$

Product using recursive multiplier with proposed correction (two most significant columns considered for correction) scheme :

$P_{approx} = (0000000111110000)_2 = (496)_{10}$

Average error (%) = $\frac{(P_{actual} - P_{approx})}{P_{exact2}} = \frac{(500-496)}{500} * 100 = 0.8$

It can be observed from the above analysis that the proposed correction with two most significant columns considered for correction achieves better accuracy when compared to the existing schemes.

#### 4.2.1.1 Hardware Implementation of the Proposed Fixed-Width Recursive Multiplier

The hardware block diagram of a $8*8$ fixed-width recursive multiplier is illustrated in Fig.4.4. The sub-multipliers $A_HB_H, A_LB_H, A_HB_L$ are implemented using 4*4 binary array multipliers [3] while the compensation function essentially includes the two most significant partial product columns highlighted in triangle in Fig.4.2. The PPs corresponding to sub-multipliers along with the correction function are reduced using a tree carry save adders and final adder. A detailed PP reduction mechanism is explained below.

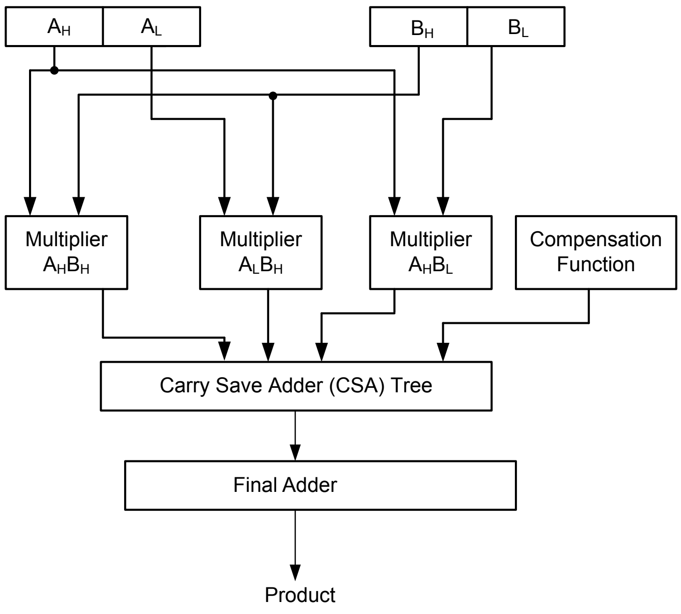


Figure 4.4: A new fixed-width recursive multiplier hardware

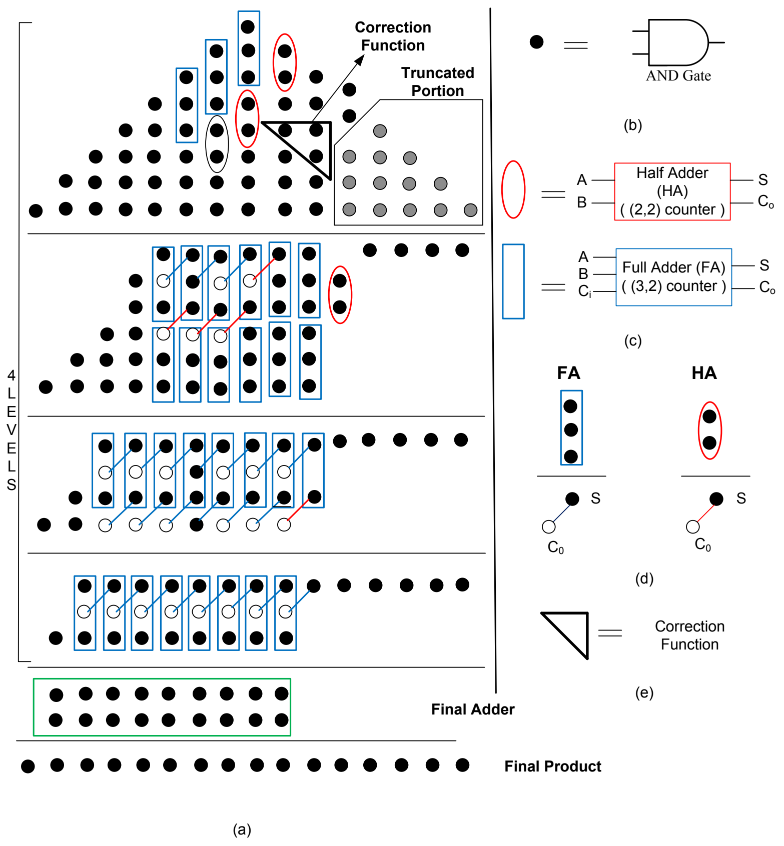


Figure 4.5: (a) Partial product reduction structure of a fixed-width recursive multiplier with two most significant columns considered for correction (b) Partial product computed using an AND gate (c) Representation of 3:2 and 2:2 counters (d) Computation of Sum and Carry-out using dot notation in a full and half adder circuits (e) Notation for correction function with two most significant columns considered

Figure 4.5 illustrates the reduction of PPs generated from individual sub-multipliers and compensation function using a CSA tree structure. The partial products shown in grey color corresponding to $A_L B_L$ are discarded and a correction function highlighted in a triangle is added. As mentioned earlier, the CSA structure is formed using full adder (3:2 counter) and half adder (2:2 counter) circuits as shown in Fig.4.5(c). The sum output $(S)$, from a full or half-adder at one stage places a dot in the same column at the next stage. A carry output $(C_o)$ from a full or half-adder at one stage places a dot one order of magnitude higher i.e., in the column to its left, at the next stage. As shown in Fig.4.5(d), three dots joined by a solid diagonal line indicates that these PPs are outputs of (3,2) counter while two dots joined by diagonal line indicates that these PPs are outputs of (2,2) counter. Consequently, the PP matrix is accumulated to a height of two in four levels using a carry save adder (CSA) tree structure, formed using full adder and half adder as shown in Fig.4.5(a), that are eventually reduced to a product using a final ripple carry adder [6] .

## 4.3 Results

In order to compare the proposed fixed-width multiplication scheme with the existing ones, error analysis has been carried out using MATLAB to check the improvement in precision. This is followed by unit gate level modeling and synthesis based analysis to understand the hardware savings achieved.

### 4.3.1 Error Analysis

MATLAB program has been used to simulate various multiplier architectures including the proposed one. For the simulation purpose, we randomly selected 10,000 inputs from all possible input patterns (i.e., 0–65 535). Error analysis has been carried out to compute the average and maximum error in the proposed and the existing schemes [18, 21–23]. An example of calculating the average error has already been illustrated in example 4.1 given earlier. A comparison of the errors is provided in Table 4.1. It may be noted that the proposed multiplier with two most significant columns is taken as the reference (ratio of '1'). For example, constant correction [23] scheme has 15.2X error in comparison with the reference design. Also, lesser ratio implies better precision.

For a fair comparison, the following have been taken into consideration. The existing schemes have correction function which is fixed and hence the maximum and average error shown in Table 4.1 do not mention the number of most significant columns considered for correction unlike the proposed multiplier.

The fixed-width recursive multiplier schemes [18] including the proposed one perform better in terms of precision compared to the fixed-width array multipliers [21, 22]. This is due to the fact that in all the recursive architectures the sub-multiplier $A_L B_L$ that contributes marginally to final result is discarded.

It can be seen that in the proposed scheme when more number of columns is considered, the average error reduces and hence the accuracy of multiplier improves. Results in Table 4.1

Table 4.1: An error comparison of various multiplier architectures

| Multiplier scheme | Most significant columns considered | Average error | Ratio | Maximum error | Ratio |
|---|---|---|---|---|---|
| Constant correction [23] | - | 0.4 | 15.2 | 4 | 4.77 |
| Constant correction [21] | - | 0.06 | 2.28 | 3 | 3.58 |
| Variable correction [22] | - | 0.06 | 2.28 | 1.4 | 1.67 |
| Direct truncated recursive multiplier [58] (Without correction) | - | 0.059 | 2.1 | 1 | 1.19 |
| Fixed-width recursive multiplier with average value correction (Scheme 2) [18] | - | 0.037 | 1.4 | 0.875 | 1.04 |
| Fixed-width recursive multiplier with 1-bit correction (Scheme 3) [18] | - | 0.055 | 2.24 | 1.25 | 1.49 |
| Proposed recursive fixed-width recursive multiplier | 1 | 0.055 | 2.24 | 1.25 | 1.49 |
| | 2 | 0.0263 | 1 | 0.837 | 1 |
| | 3 | 0.0229 | 0.87 | 0.810 | 0.96 |
| | 4 | 0.0189 | 0.71 | 0.782 | 0.93 |

show that the new recursive scheme with two most significant columns as correction has an improvement in average error of 1.4 to 15.2X compared to the existing designs. Similarly, an improvement of 1.04 to 4.77X in maximum error is achieved. It can also be seen that an improvement of 1.5 to 15.3X and 1.7 to 15.5X in average error is achieved, if column sizes of ‘3’ and ‘4’ are included for correction respectively. It can be observed that significant reduction in error is obtained when only two most significant columns are considered for correction. Although, there is an increase in the accuracy when the number of columns considered is more than two, it is only nominal which comes at the cost however of increased hardware, as shown in the next subsection.

### 4.3.2 Area and Delay Comparison of Various Multipliers using Unit Gate Analysis

Different correction schemes of various multipliers have been analyzed using unit gate modeling as this approach provides a decent model for computing the real cost of each component. Further, it does not depend on any synthesis tool [59]. Design metrics such as area (A) and delay (D), have been considered and compared for all the designs. Assumptions made while calculating the area of components are shown in Table 4.2.

Table 4.2: Assumptions made for unit gate modeling

| Gates | Count |
|---|---|
| 2- INPUT AND, OR, NAND, NOR | 1 |
| M- INPUT AND, OR, NAND, NOR | M-1 |
| 2- INPUT XOR, XNOR, MUX | 2 |

Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for area [59]. Moreover, an m-input gate is assumed to be composed of a tree of 2-input gates and the effects of wiring, buffering and inverting costs are neglected.

Table 4.3: Unit gate modeling analysis of various multiplier architectures

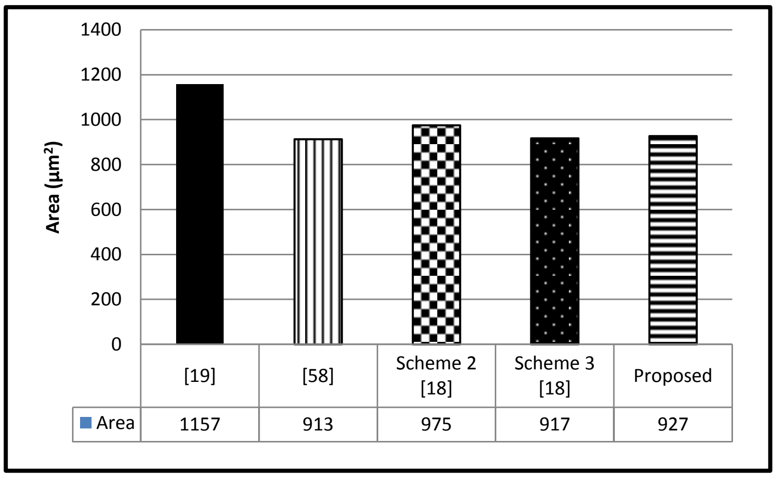
| Multiplier scheme | Most significant columns considered | Area | Percentage |
|---|---|---|---|
| Accurate recursive multiplier [19] | - | 470 | 124% |
| Direct truncated recursive multiplier (Without correction) [58] | - | 364 | 96% |
| Fixed-width recursive with average value correction (Scheme 2) [18] | - | 396 | 104% |
| Fixed-width recursive with 1-bit correction (Scheme 3) [18] | - | 367 | 97% |
| Proposed recursive fixed-width recursive multiplier | 1 | 367 | 97% |
|  | 2 | 379 | 100% |
|  | 3 | 401 | 106% |
|  | 4 | 421 | 111% |

The results of unit gate modeling of various 8-bit multipliers, including the proposed multiplier, (with various number of columns considered) have been compared and given in Table

4.3. The proposed multiplier with two most significant columns as correction is taken as the reference. It can be seen that the hardware of the proposed design increases with the number of columns considered for correction. It may be observed that there is an area overhead of 4% of the proposed multiplier compared to the best performing recursive multiplier when only two columns are considered for correction while it increases to 10% and 15% for column sizes of three and four respectively. Thus, taking into account the precision improvement and hardware required, it can be concluded that including two most significant columns for correction results in a good trade-off between the precision and area of the hardware.

### 4.3.3 Hardware Synthesis Results

For a fair comparison, all the multiplier designs of 8-bit width have been modeled with Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1 and mapped on to TSMC 180nm technology, slow-normal library using cadence RTL compiler v7.1. Hardware synthesis has been carried out to compare important implementation metrics such as area, delay and power.



| | [19] | [58] | Scheme 2 [18] | Scheme 3 [18] | Proposed |
|---|---|---|---|---|---|
| ■ Area | 1157 | 913 | 975 | 917 | 927 |

Figure 4.6: A Comparison of the proposed fixed-width recursive multiplier with various existing multipliers in terms of area

Table 4.4 presents performance metrics such as area, delay and power for the 8-bit multiplier with different correction schemes including the proposed one. Also included in the Table is performance in percentage of various designs in comparison with the proposed designs. Further, Figs.4.6-4.8 provide a graphical comparison of area, delay and power of various

Table 4.4: Area, delay and power of various multipliers with correction schemes

| Multiplier scheme | Area ($\mu m^2$) | Percentage (%) | Delay ($ps$) | Percentage (%) | Power ($nW$) | Percentage (%) |
|---|---|---|---|---|---|---|
| Accurate recursive multiplier [19] | 1157 | 124% | 2120 | 115% | 40834 | 114% |
| Direct truncated recursive multiplier (Without correction) [58] | 913 | 98% | 1715 | 93% | 35196 | 98% |
| Fixed-width recursive with average value correction (Scheme 2) [18] | 975 | 105% | 1952 | 106% | 36405 | 102% |
| Fixed-width recursive with 1-bit correction (Scheme 3) [18] | 917 | 99% | 1795 | 97% | 35245 | 98% |
| Proposed fixed-width recursive multiplier with two column correction scheme | 927 | 100% | 1843 | 100% | 35849 | 100% |

multipliers including the proposed one.

It is evident from Table 4.4 and Figs.4.6 & 4.7, that an improvement of up to 5% and 6% in area and delay respectively is achieved by the proposed scheme compared to scheme 2 [18] which comes closest in terms of the precision. It should however be remembered that scheme 2 has a precision that is 40% less than that of the proposed one. While other schemes may marginally perform better in terms of area and delay, they are way off in terms of the precision compared to the existing one. Thus, it can be concluded that the proposed fixed-width multiplier scheme outperforms all other existing schemes in the literature.

## 4.4 Benchmarking Various Multiplication Schemes-Application to Image processing

Image sharpening is an important image enhancement technique employed in image processing applications. The computational process of sharpening an image involves a number of fixed

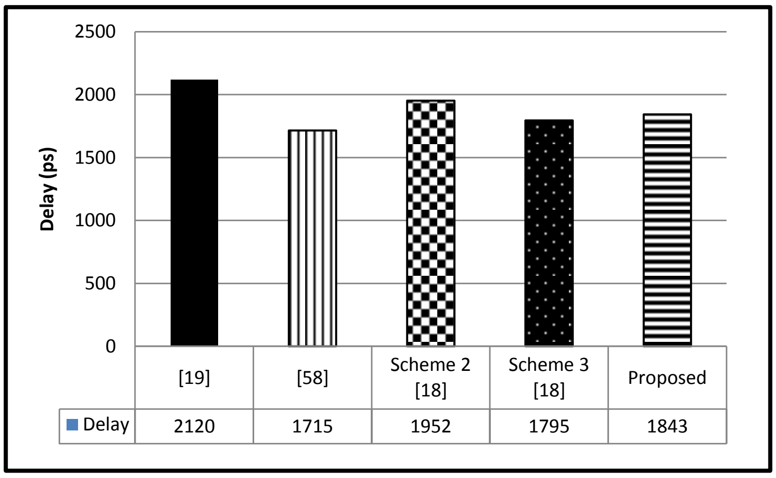


Figure 4.7: A Comparison of proposed fixed-width recursive multiplier scheme with various existing multipliers in terms of delay

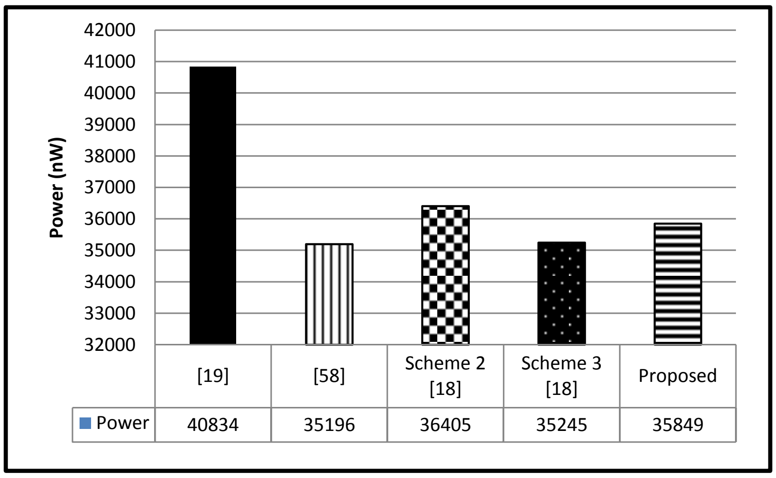


Figure 4.8: A Comparison of proposed fixed-width recursive multiplier scheme with various existing multipliers in terms of power

point multiplications. It is therefore a good application to prove the efficacy of the proposed fixed-width recursive multiplier.

### 4.4.1 Image Sharpening Algorithm

Human perception is highly sensitive to edges and fine details of an image. Since images essentially consist of high-frequency components their visual quality is corrupted if these high frequencies are removed. Conversely, increasing the high-frequency components of an image improves the image quality. Image sharpening algorithm described in [60] is one such enhancement technique which highlights the edges and fine details in an image.

This algorithm described below, accepts an image, processes it, and produces an image of high quality. Suppose $I$ is the original image, the processed image $S$ is described using the expression

$$S(x,y) = 2I(x,y) - M \tag{4.5}$$

where $M = \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} H(i+3, j+3) I(x-i, y-j)$

and $H$ is a matrix defined as

$$H = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Since this expression involves a number of multiplications, an exact multiplier such as an array multiplier can perform these operations accurately thereby producing an image of high quality. On the other hand, using an approximate multiplier would result in an image of certain quality which is quantified using established metrics such as mean square error (MSE) and peak signal to noise ratio (PSNR).

The MSE represents the loss of information in the image and is expressed as,

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i, j) - K(i, j)]^2 \tag{4.6}$$

where $MAX_I$ represents the maximum possible pixel value of the image.

The peak signal to noise ratio (PSNR) in dB is expressed using $MSE$ as follows:

$$PSNR \text{ in dB} = 10.log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{4.7}$$

While the use of approximate multiplier may affect the image quality, it has the advantage of savings in terms of area and delay as compared to an accurate multiplier. In what follows, the performance of the existing recursive multipliers (Schemes 2 and 3) [18] and the proposed multiplier with two column correction is studied and compared with reference to the image sharpening algorithm. The algorithm is applied to blocks of 5*5 pixels on a set of standard images ( Cameraman, Lena and pirate). The exact multiplications are replaced by approximate multiplications using existing multipliers and the proposed multiplier, while addition, subtraction and division operations are carried out using accurate techniques. The metric MSE is computed by finding the mean of squares of difference in pixel values between original image and the processed image using approximate multipliers and these values are substituted in equation (4.7) to calculate PSNR values.

Table 4.5 provides a comparison of these metrics on a set of standard images ( Cameraman, Lena and pirate). As is well known, the quality of image is decided by the magnitude of MSE and PSNR values. It is evident from Table 4.5 that images processed with the proposed multiplier have better MSE and PSNR compared to the images processed using the schemes mentioned in [18].

Table 4.5: A comparison of values of MSE and PSNR for benchmark images using various multiplier schemes

| Image | Metric | Scheme 3 [18] | Scheme 2 [18] | Proposed |
|---|---|---|---|---|
| Cameraman | PSNR (dB) | 43.2 | 43.6 | 44.1 |
| Lena | | 39.2 | 39.3 | 39.7 |
| Pirate | | 42.7 | 43.2 | 43.8 |
| Cameraman | MSE | 3.1 | 2.8 | 2.5 |
| Lena | | 7.8 | 7.5 | 7 |
| Pirate | | 3.5 | 3.1 | 2.7 |

The results are only expected since the proposed scheme has less average and maximum error compared to schemes mentioned in [18] as illustrated in Figs.4.9 & 4.10.

Further, the performance of the proposed multiplier, in terms of both PSNR and MSE, may be understood by observing the images processed by it. Figure.4.11 illustrates the images that are processed using exact (array) and the proposed multiplier. It can be observed that the images of Cameraman, Lena and Pirate processed with the proposed multiplier look very similar to the original ones. Thus, it can be concluded that the proposed fixed-width recursive multiplier has a better performance compared to all other similar and existing multipliers.

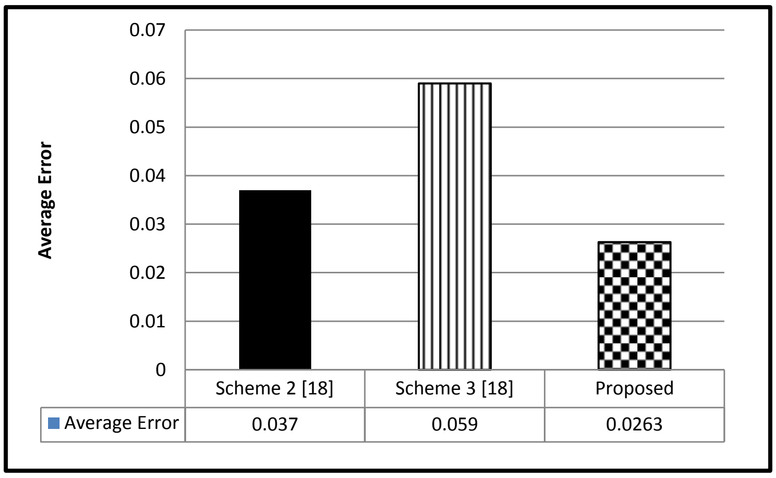


Figure 4.9: Average errors of various multipliers

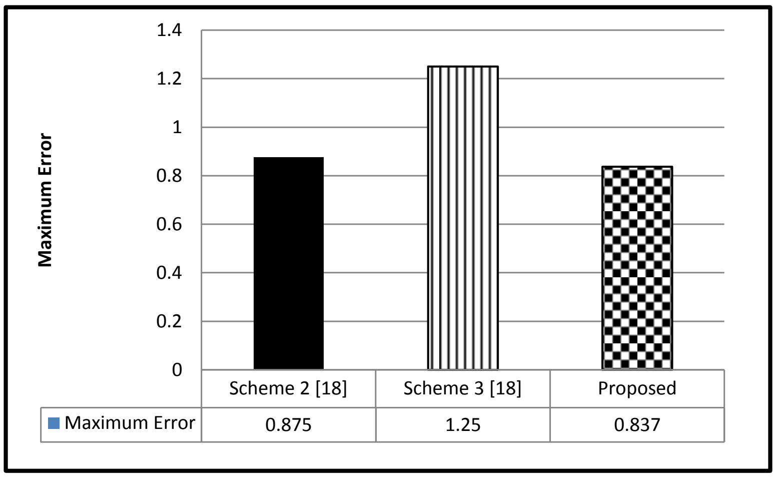


Figure 4.10: Maximum errors of various multipliers

## 4.5 Conclusions

In this chapter, a reconfigurable multiplier design with a new truncation technique targeted for error resilience applications has been proposed. In this scheme, the sub-multiplier is replaced $A_L B_L$ with a data-dependent correction term. The heuristic correction function retains the portion of partial product matrix that is more relevant and thus helps to achieve higher accuracy compared to the earlier works. Also, the hardware overhead due to correction scheme is nominal.

Exhaustive analysis was carried out to compute the average and maximum error bound in the proposed and existing recursive multiplier schemes. Results show 15.2X times and 3.7X times improvement in average error and maximum error respectively. The error analysis

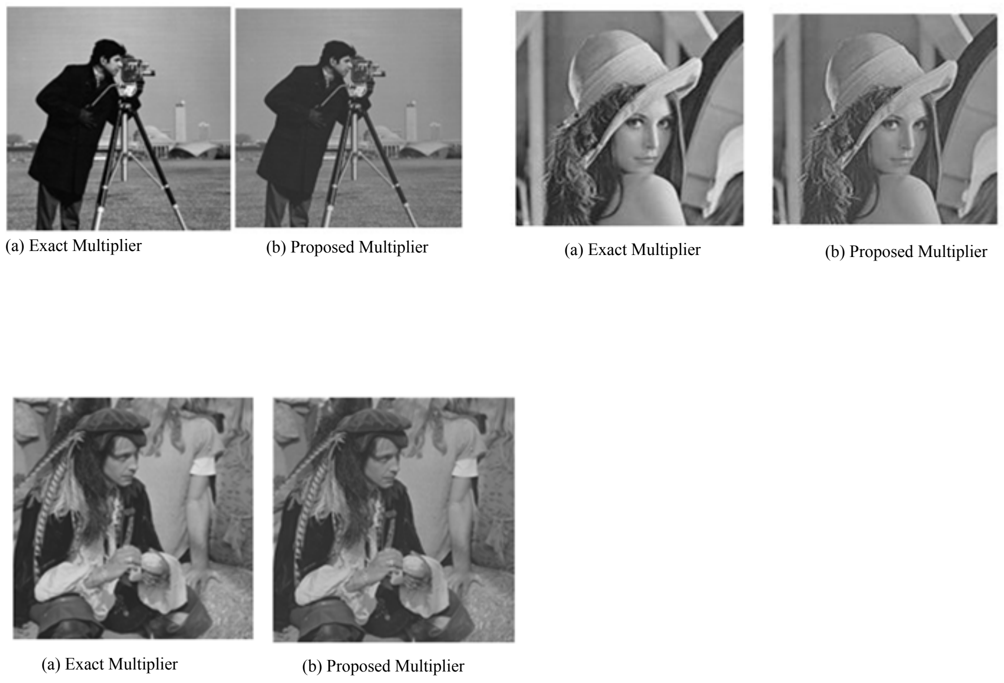


Figure 4.11: Cameraman, Lena and pirate Images obtained using exact and the proposed multiplier

show that the proposed multiplier scheme represent, in most cases, a better trade-off between accuracy and complexity.

Further, the proposed architecture efficiently performs N-bit fixed-width multiplications. For applications with high demands on precision, the multiplier is capable of performing two independent N/2-bit full precision multiplications in parallel.

# Chapter 5

# An Iterative Logarithmic Multiplier with Improved Precision

## 5.1 Introduction

In recent years, logarithmic number system (LNS) has been increasingly used as an alternative to the binary number system as it converts multiplication to addition resulting in simplified hardware [61]. However, they suffer from inherent error and any efforts in improving their accuracy would help find their increased usage in arithmetic computations with efficient hardware. In this chapter, a novel binary logarithmic multiplier with improved precision is designed and demonstrated to perform better than the existing designs. Also, the multiplier has been synthesized and it‘s performance metrics such as area, delay and power have been shown to be better than those of the existing designs.

Throughout this chapter, the symbols ‘ ~ ’ and ‘ ’ ’ are used to denote 1‘s and 2‘s complement while the symbols ‘ & ’ and ‘ | ’ denote logic AND and logic OR operations respectively. Rest of the chapter is organized as follows: Mathematical modeling and hardware architecture of the proposed scheme are presented in Section 5.2. Detailed error analysis and hardware synthesis results of various iterative logarithmic multipliers including the proposed one are presented and compared in Section 5.3. Proposed and the existing schemes have been applied on an image sharpening algorithm to quantify their performance, results of which are provided in

Section 5.4.

## 5.2 Proposed Approach

The contributions of this work include speculation of carry which results in better precision when compared to BIM and TEC approaches. Further, savings in hardware is achieved using the fractional predictor and adapted truncation scheme proposed in this work. This section presents the mathematical modeling while the hardware implementation details are discussed in the subsequent sections.

### 5.2.1 Mathematical Analysis of the Proposed Scheme

Similar to Babic's approach, equation (2.26) is expressed when carry = 1 ($x_1 + x_2 \geq 1$ ) as

$$\begin{aligned} N_1 * N_2 &= 2^{k_1+k_2+1}(x_1+x_2) + 2^{k_1+k_2}(x_1^{'} * x_2^{'}) \\ &= 2^{k_2+1} * f_1 + 2^{k_1+1} * f_2 + f_1^{'} * f_2^{'} \\ &= A^1 + f_1^{'} * f_2^{'} \end{aligned} \tag{5.1}$$

$$\text{Where approximate product, } A^1 = f_1 * 2^{k_2+1} + f_2 * 2^{k_1+1} \tag{5.2}$$

Combining the equations 2.28 and 5.1, the final product terms based on carry information can be obtained as,

$$N_1 * N_2 = A^0 + (f_1 * f_2), \quad x_1 + x_2 < 1 \tag{5.3}$$

$$N_1 * N_2 = A^1 + (f_1^{'} * f_2^{'}), \quad x_1 + x_2 \geq 1 \tag{5.4}$$

where $A^0$ and $A^1$ are the approximate product terms, $f_1 * f_2$ and $f_1^{'} * f_2^{'}$ are the correction terms that form the input to the next iteration based on carry condition. Since the carry information is available prior to the completion of an iteration, error correction can be performed concurrently in hardware.

While this approach leads to improved precision, it also results in area overhead. Thus, a new truncation scheme for logarithmic multiplication is proposed which when combined with the above approach results in area savings.

### 5.2.2 A New Approach to the Approximation of Logarithmic Multiplier

The proposed approximation of logarithmic multiplier is based on the fact that the error due to Mitchell approach is always positive [59] and hence the multiplication product obtained is less than the exact value. Further, the least significant bits of these fractional portions ($f_1 * 2^{k_2}$, $f_2 * 2^{k_1}$, $f_1 * 2^{k_2+1}$ and $f_2 * 2^{k_1+1}$) are inaccurate. Thus, rounding off and manipulating these inexact terms aid in achieving substantial hardware savings without compromising on precision. On the other hand, the term $2^{k_1+k_2}$ (leading one) contributes to integer portion which is an exact value and hence need not be approximated.

The product terms in equations 5.3 and 5.4 after truncation ($T_t$) of fractional portions are written as follows:

$$N_1 * N_2 = 2^{k_1+k_2} + T_t(f_1 * 2^{k_2}) + T_t(f_2 * 2^{k_1}) + f_1 * f_2, \quad x_1 + x_2 < 1 \tag{5.5}$$

$$N_1 * N_2 = T_t(f_1 * 2^{k_2+1}) + T_t(f_2 * 2^{k_1+1}) + f_1^{'} * f_2^{'}, \quad x_1 + x_2 \geq 1 \tag{5.6}$$

where approximate product terms without and with carry are,

$A_t^0 = 2^{k_1+k_2} + T_t(f_1 * 2^{k_2}) + T_t(f_2 * 2^{k_1})$ and

$A_t^1 = T_t(f_1 * 2^{k_2+1}) + T_t(f_2 * 2^{k_1+1})$ respectively. Here $t$ denotes the truncation width.

A typical logarithmic multiplication of two inputs $N_1$ and $N_2$, each of $n$-bit width, results in $2n$ product bits. In the proposed truncated approach however, from these $2n$ fractional bits, only $t$ bits are retained. The remaining $(2n - t)$ bits are replaced either by ‘1’ (rounding up) or ‘0‘ (rounding down), denoted as $T_{t,1}$ and $T_{t,0}$ respectively. In order to arrive at an optimal value of

$t$, error analysis of an 8*8 logarithmic multiplier is carried out for one iteration as illustrated in Fig.5.1. The plot shows the variation of average error as a function of the number of fractional bits. The horizontal line gives the error without truncation $(T_{wt})$, while the curves $T_{t,1}$ and $T_{t,0}$ provide the error due to truncation.

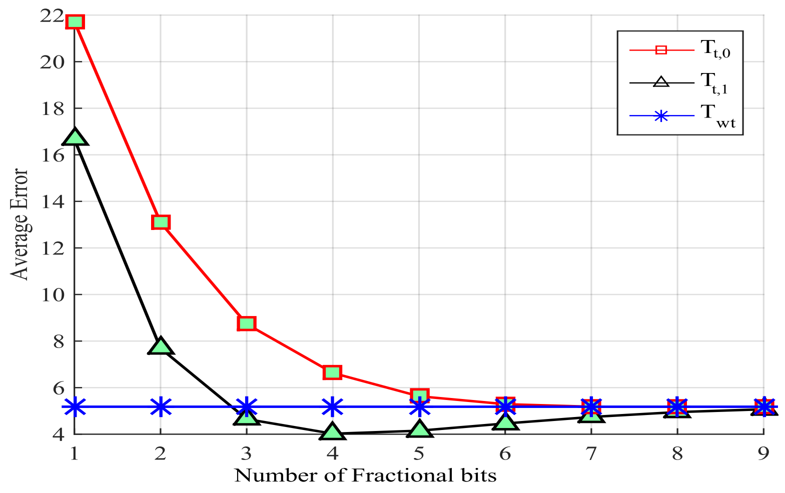


Figure 5.1: Average error Vs Number of fractional bits in 8 * 8 multiplication for one iteration

It can be seen from the figure that the average error without truncation $(T_{wt})$ (considering 2n fractional bits) remains constant and is independent of truncation width. Also, it is evident from $T_{t,0}$ plot that with the increase in truncation width$(t)$, the average error decreases rapidly and from $t = 6$ onwards, attains a value almost equal to that without truncation. Therefore, it can be inferred that the first 6 MSB fractional bits alone are contributing to the precision of the multiplication product in most cases. Similarly, from $T_{t,1}$ plot it can be inferred that for $t = 3$ error is almost same as that without truncation with precision being the best for $t = 4$. Therefore, it can be concluded that the average error obtained is minimum for $T_{4,1}$ scheme compared to all other cases.

Since the error in multiplication due to approximation should decrease with each iteration, the viability of $T_{4,1}$ truncation scheme alone for multiple iterations has been investigated. It is however found that the result obtained using $T_{4,1}$ method for first iteration is more than the exact value and further with each successive iteration there is a probability for the error to increase. Thus, utilizing $T_{4,1}$ scheme alone for multiple iterations may not be desirable. Based on the error simulations, it is observed that truncating with $T_{6,0}$ in initial iterations and $T_{4,1}$ in the final iteration results in improved error and the same is proved in the example given below.

Further, it should be noted that error plots obtained for 16* 16 and 32*32 multipliers will have characteristics similar to that in Fig.5.1.

The step-by-step procedure of the proposed truncation approach for two iterations is illustrated in example 5.1. Initially, for both 1st and 2nd iterations, truncation method $T_{6,0}$ is applied. Next, $T_{6,0}$ is applied in the 1st iteration with $T_{4,1}$ in the 2nd iteration.

**Example 5.1.** Let the two binary numbers $N_1$ and $N_2$ be

$N_1 = (00110011)_2 = (51)_{10}; N_2 = (01110111)_2 = (119)_{10}$

Exact product, $A_{exact}$ is given by $N_1 * N_2 = (6069)_{10}$

**Case 1**: $T_{6,0}$ is considered in both iterations

**Iteration 1:**

1. Initialization:

$(N_1) = (110011)_2$ ; $(N_2) = (1110111)_2$

$(f_1) = (10011)_2$ ; $(f_2) = (110111)_2$

$(k_1) = (101)_2 = (5)_{10}$ ; $(k_2) = (110)_2 = (6)_{10}$

The addition of $f_1$ and $f_2$ generates a carry satisfying the condition $x_1 + x_2 \geq 1$ and hence equation (5.6) is considered.

2. Left shift the fractional portions,

$f_1 * 2^{k_2+1} = (2432)_{10}$ ; $f_2 * 2^{k_1+1} = (3520)_{10}$

3. Truncate the fractional portion and compute the approximate product :

$T_{6,0}(f_1 * 2^{k_2+1}) = (2432)_{10}; T_{6,0}(f_2 * 2^{k_1+1}) = (3520)_{10}$

which after the 1st iteration is,

$$A^1 = (5952)_{10} \tag{5.7}$$

4. Percentage average error accumulated after 1st iteration:

$$\text{Average Error} = \left(\frac{A_{exact} - A^1}{A_{exact}}\right)$$

$$= 1.92\%$$

**Iteration 2:**

Since $x_1 + x_2 \geq 1$ in the 1st iteration,

the inputs $(N_1 \text{ and } N_2)$ to this iteration are $f_1^{'}$ and $f_2^{'}$ of 1st iteration

1. Initialization:

$(N_1) = (01101)_2$ ; $(N_2) = (001001)_2$

$(f_1) = (101)_2 = (5)_{10}$ ; $(f_2) = (001)_2 = (1)_{10}$

$(k_1) = (11)_2 = (3)_{10}$ ; $(k_2) = (11)_2 = (3)_{10}$

For the proposed design with truncation ($T_{6,0}$), condition $x_1 + x_2 < 1$ is satisfied as addition of $f_1$ and $f_2$ does not generate a carry and hence equation (5.5) is considered.

2. Left shift the fractional portions:

$f_1 * 2^{k_2} = (00101000)_2 = (40)_{10}$; $f_2 * 2^{k_1} = (1000)_2 = (8)_{10}$

3. Truncate the fractional portion and compute $\left(A^0\right)$:

$T_{6,0}\left(f_1 * 2^{k_2}\right) = (00101000)_2 = (40)_{10}$,

$T_{6,0}\left(f_2 * 2^{k_1}\right) = (1000)_2 = (8)_{10}$; $2^{k_1+k_2} = (64)_{10}$

Approximate product after the 2nd iteration is,

$A^0 = (112)_{10}$

4. Percentage average error accumulated after two iterations:

$A_{total} = A^1 + A^0 = (6064)_{10}$

$\text{Average Error} = \left(\frac{A_{exact} - A_{total}}{P_{exact}}\right)$

$\text{Average Error} = 0.082\%$

**Case 2:**

The product in the first iteration is computed using the $T_{6,0}$ scheme while in the second iteration $T_{4,1}$ scheme is adopted

**Iteration 1 :**

From equation (5.7) we obtain the approximate product in the 1st iteration as,

$\text{Approximate Product}\left(A^1\right) = (5952)_{10}$

**Iteration 2 :**

$T_{4,1}(f_1 * 2^{k_2}) = (00101011)_2 = (43)_{10},$

$T_{4,1}(f_2 * 2^{k_1}) = (1000)_2 = (8)_{10};\ \ 2^{k_1+k_2} = (64)_{10}$

$A^0 = (115)_{10}$

Combining the products obtained in the first and second iteration with $T_{6,0}$ and $T_{4,1}$ schemes respectively, we get the product:

$A_{total} = A^1 + A^0 = (6067)_{10}$

The corresponding average error is given by,

$\text{Average Error} = 0.032\%$

Thus, it is observed that lower error is achieved if $T_{6,0}$ is used for initial iterations followed by $T_{4,1}$ in the final iteration.

### 5.2.3 Truncated Iterative Multiplier (TIM) Hardware Implementation

The iterative multiplier scheme illustrated in Fig.5.2 is essentially an implementation of equations 5.5 and 5.6. It comprises of truncated basic logarithm blocks (TBLBs), decoder logic, adder blocks and mask logic. Based on the binary numbers ($N_1$ and $N_2$), the TBLBs generate the characteristics and truncated fractional portions. The addition of characteristics $k_1$ and $k_2$ obtained from the respective TBLBs is accomplished using the Adder 2 block. Likewise, the sum of inexact fractional portions $(f_1 * 2^{k_2})_T$ and $(f_2 * 2^{k_1})_T$ obtained from the respective TBLBs is found using the Adder 1 module. Based on carry information, the control signal (m_fp) is activated by the modified fractional predictor (MFP). The truncated BLB and MFP are presented later in Sections. 5.2.3.1 & 5.2.3.2.

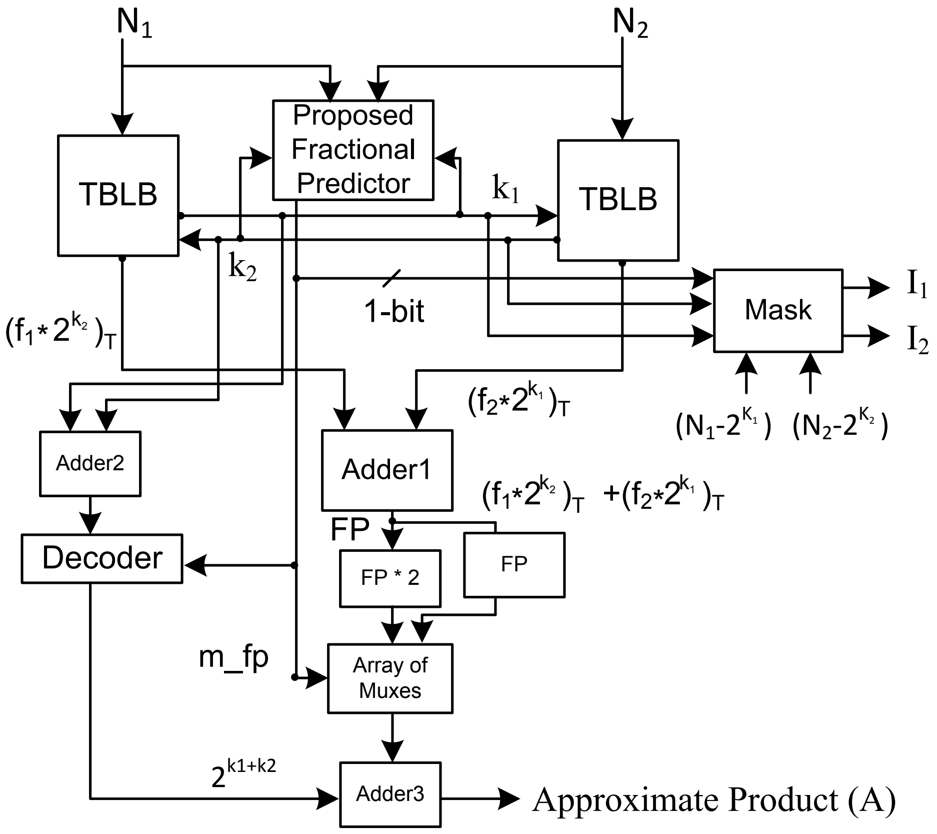


Figure 5.2: Block diagram of the proposed truncated iterative multiplier (TIM)

When the control signal m_fp is '0', meaning no carry generation, the leading one $\left(2^{k_1+k_2}\right)$ obtained from the Decoder and the truncated fractional portions are added using Adder 3 to form the approximate product $(A_t^0)$ according to equation (5.5). On the other hand, if m_fp is '1', the Decoder is disabled and the fractional portions are left-shifted by 1-bit as per equation

(5.6) to form the approximate product $\left(A_t^1\right)$. The MUX structure based on the carry signal facilitates the selection of appropriate truncated fractional portions.

#### 5.2.3.1 Truncated Basic Logarithmic Block (TBLB)

The Truncated BLB (TBLB) shown in Fig.5.3 used in this work is similar to the BLB presented in [26] except for the truncated logarithmic shifter.

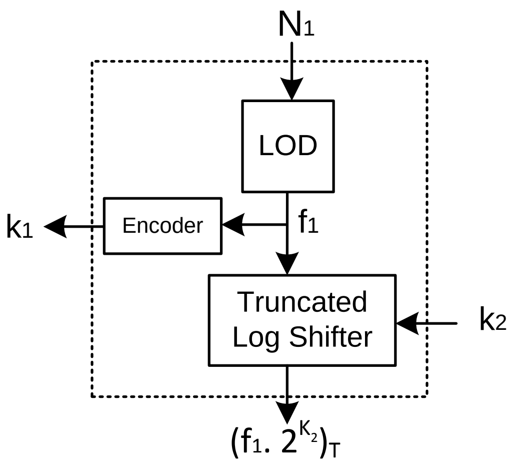


Figure 5.3: Block diagram of the truncated basic logarithmic block (TBLB)

The fact that the fractional portion $\left(f * 2^k\right)$ computed by the shifter is always a positive approximate term provides the possibility for truncating it without a significant loss of precision. Based on this idea, a method proposed in [59] implements a truncated logarithmic shifter which is used in this work. The operation of the shifter is such that it retains only those most significant bits as dictated by the truncation width. The truncated LSB section is manipulated with either ‘1’ or ‘0’ that depends on rounding up or down.

#### 5.2.3.2 Modified Fractional Predictor (MFP)

The TEC scheme [27] involves prohibitively large hardware circuitry in the form of fractional predictor, shared logic and multi-operand adder to speculate the carry out of the fractional portion. In order to overcome this, a simple and flexible carry prediction logic namely, modified fraction predictor (MFP) is proposed in this work. The output of MFP logic is 1-bit (‘0’ or ‘1’), irrespective of the number of input bits unlike in TEC approach.

In this work, MFP is preceded by a number that refers to the number of fractional bits considered for prediction. For instance, if number of fractional bits is one then it is termed as 1-bit MFP and so on. A 1-bit MFP is explained in detail using Example 5.2.

**Example 5.2.** Suppose two input binary numbers $N_1$ and $N_2$ are given by,

$N_1 = (00110011)_2 = (51)_{10}$; $N_2 = (01110111)_2 = (119)_{10}$

The characteristics ($k_1$ and $k_2$) and their 1's complement ($S_1$ and $S_2$) are :

$k_1 = (101)_2 \ = (5)_{10}$ ; $k_2 = (110)_2 \ = (6)_{10}$

$S_1 = \sim k_1 = (010)_2$ ; $S_2 = \sim k_2 = (001)_2$

The leading one of these two numbers is found by shifting $N_1$ and $N_2$ by an amount of $S_1$ and $S_2$ respectively resulting in $A$ and $B$ as depicted in Fig.5.4. .

$A = (11001100)_2$ ; $B = (11101110)_2$

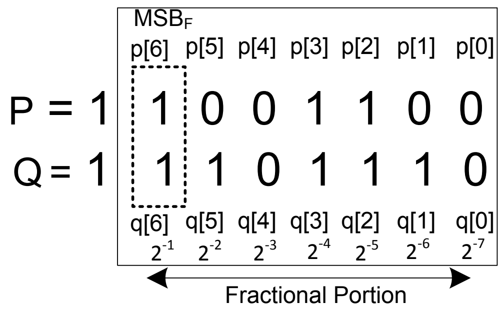


Figure 5.4: Proposed method to detect leading one in fractional portion

The $MSB_F$ in A and B has a weight of $2^{-1}$ or 0.5 and carry will be generated to next stage when $a[6]$ and $b[6]$ is '1'. Conversely, if either $a[6]$ or $b[6]$ is logic '1', then generation of carry depends on the LSB portion of $A[5:0]$ and $B[5:0]$. Thus, carry detection is performed without waiting for actual addition to happen leading to implementation of error correction parallely.

Similarly, the logic for 2-bit and 3-bit MFP is deduced. The carry prediction logic corresponding to 1-bit, 2-bit and 3-bit MFP are deduced as specified in equations (5.8-5.10) as

$$sel1 = a[6] \,\&\, b[6] \tag{5.8}$$

$$sel2 = sel1\,|\,((a[6]|b[6])\&(a[5]\&b[5])) \tag{5.9}$$

$$sel3 = sel2\,|sel \tag{5.10}$$

Where $sel = ((a[6]|b[6])\&(a[5]|b[5])\&(a[4]\&b[4]))$

To analyze the variation of average error with the number of fractional input bits, a graph is plotted for one iteration as illustrated in Fig.5.5. It is evident from the graph that the error is minimum for bit size of 4 but thereafter remains almost constant. Hence, MFP with four input fractional bits is sufficient for accurate carry prediction.

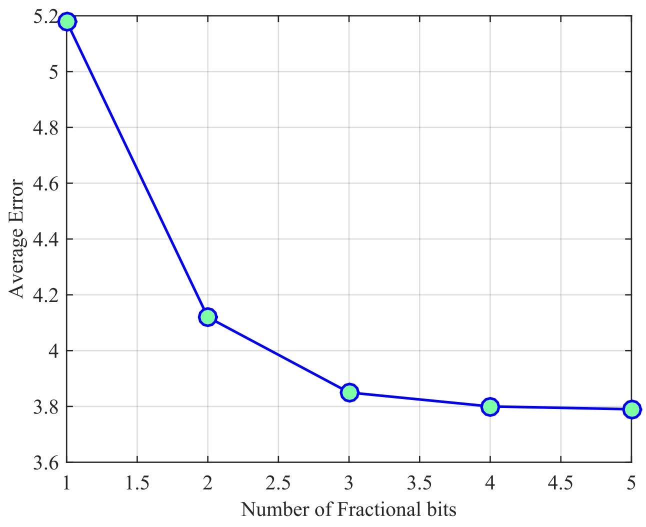


Figure 5.5: Variation of average error based on number of fractional bits

## 5.2.4 TIM Hardware for two Iterations

Figure.5.6 illustrates the cascaded TIM multipliers. While the first one computes the approximate product (A), the second one calculates the correction term. The input to the second TIM is determined by the mask circuit depending on m_fp. If m_fp is '1', the inputs to next iteration $I_1$ and $I_2$ are $f_1^{'}$ and $f_2^{'}$ otherwise $f_1$ and $f_2$ respectively as per equations 5.5 and 5.6. The second TIM computes the correction term (C) based on $I_1$ and $I_2$. Similarly, in case of three iterations there will be an approximate product (A) and two correction terms. Evidently, with each successive iteration the error gets reduced.

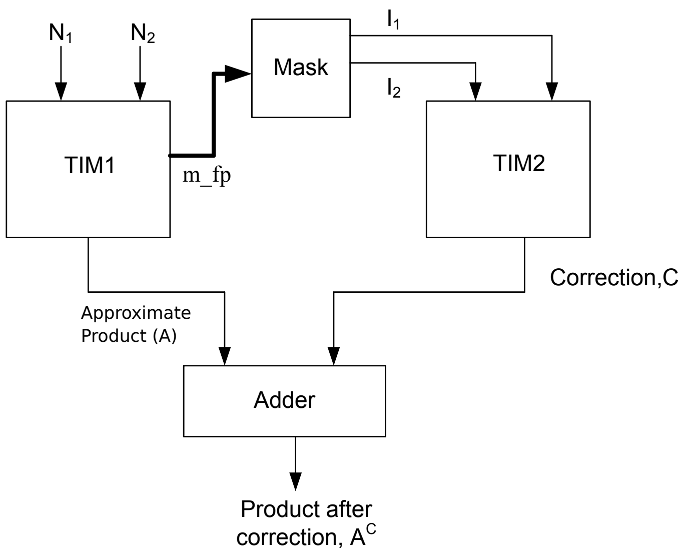


Figure 5.6: Truncated iterative multiplier implementation for two iterations

## 5.3 Error Analysis

Exhaustive error analysis has been carried out using MATLAB to compare the accuracy of the proposed multiplier with the existing iterative multiplier [26, 27].

The comparison of maximum and average error of existing designs with $T_{6,0}$ and $T_{4,1}$ schemes for three iterations is illustrated in Figs.5.7 & 5.8 respectively. Clearly, it can be observed from both the graphs that except for the first iteration, truncated TIM has least maximum and average error. A more elaborate analysis corresponding to maximum and average error for three iterations is presented in Table.5.1.

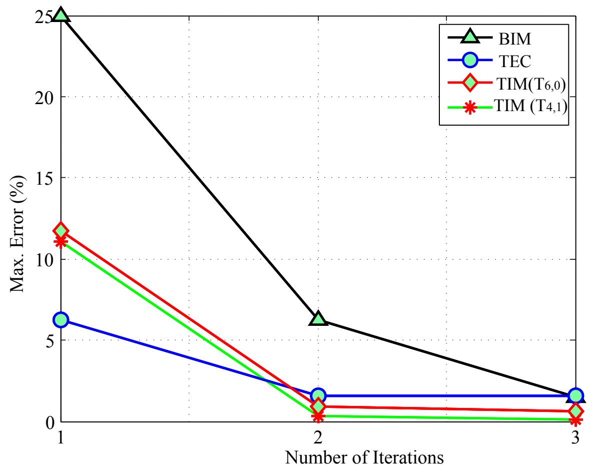


Figure 5.7: A comparison of maximum error of existing iterative multiplier designs with TIM ($T_{6,0}$ and $T_{4,1}$) approach

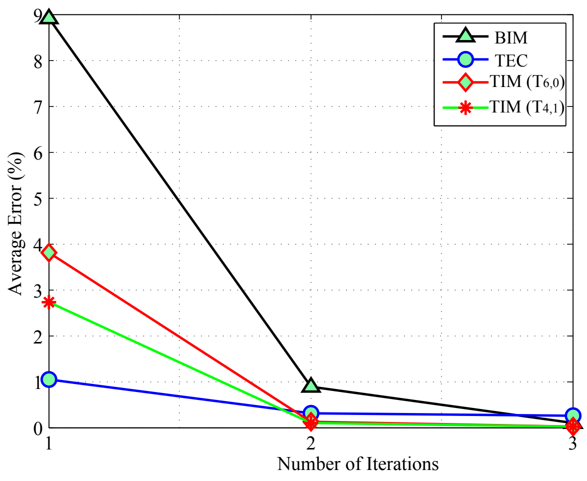


Figure 5.8: A comparison of average error of existing iterative multiplier designs with TIM ($T_{6,0}$ and $T_{4,1}$) approach

For a fair comparison, the following have been taken into consideration. The Babic iterative multiplier (BIM) [26] is devoid of error correction logic and hence the maximum and average error shown in Table.5.1 do not mention the number of mantissa fractional bits. The error analysis carried out in TEC [27] corresponds to 3-bit and 4-bit MFP hence for proper comparison the proposed TIM scheme with 3-bit and 4-bit MFP respectively is considered. Moreover, analysis for 1-bit and 2-bit MFP proposed in this work has been carried out to prove the improvement in precision achieved compared to higher bit TEC. Error metrics (maximum and average error) have been considered and compared with that of existing designs for each iteration.

A comparison of the proposed TIM scheme ($T_{6,0}$) (as per the equations 5.5 and 5.6 ) with BIM and TEC for maximum and average error for three iterations is shown in Table.5.1. It can be observed that BIM multiplier is taken as a reference (ratio of 1) corresponding to each iteration. The TIM ($T_{6,0}$) performs as well as BIM with no MFP while it performs better even with 1-bit MFP. Although, TIM approach achieves less precision compared to TEC [27] in the first iteration, the same gets better with further iterations. Maximum error with 1-bit and 2-bit MFP design has an improvement of at least 64% compared to both TEC of 3-bit MFP and BIM in second iteration. Also, the TIM design with 3-bit and 4-bit MFP has 5X and 1.7X better precision compared to TEC (3-bit and 4-bit) respectively in second iteration. In third iteration, the maximum error of the proposed scheme with 3-bit and 4-bit MFP is 9X and 2.5X better than TEC (3-bit and 4-bit) MFP respectively.

Table 5.1: A comparison of maximum and average error (%) in the proposed TIM ($T_{6,0}$) and existing schemes for 3 iterations

| | | Maximum Error (%) | | | | | | Average Error (%) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme | No. of MFP bits | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio |
| BIM [26] | - | 25 | 1 | 6.25 | 1 | 1.5 | 1 | 8.9 | 1 | 0.89 | 1 | 0.083 | 1 |
| TIM | 0-bit | 25 | 1 | 6.25 | 1 | 1.5 | 1 | 8.9 | 1 | 0.89 | 1 | 0.083 | 1 |
| TIM | 1-bit | 16 | 0.64 | 2.25 | 0.36 | 1.11 | 0.74 | 5.18 | 0.58 | 0.24 | 0.27 | 0.011 | 0.13 |
| TIM | 2-bit | 14 | 0.56 | 1.75 | 0.28 | 0.82 | 0.55 | 4.12 | 0.46 | 0.16 | 0.18 | 0.006 | 0.07 |
| TIM | | 12.5 | 0.5 | 1.19 | 0.19 | 0.67 | 0.45 | 3.85 | 0.43 | 0.13 | 0.14 | 0.005 | 0.06 |
| TEC [27] | 3-bit | 6.25 | 0.25 | 6.25 | 1 | 6.25 | 4.16 | 1.36 | 0.15 | 0.77 | 0.86 | 0.76 | 9.15 |
| TIM | | 11.75 | 0.47 | 0.92 | 0.14 | 0.6 | 0.4 | 3.8 | 0.42 | 0.12 | 0.13 | 0.004 | 0.05 |
| TEC [27] | 4-bit | 6.25 | 0.25 | 1.56 | 0.25 | 1.56 | 1.04 | 1.05 | 0.11 | 0.304 | 0.34 | 0.26 | 3.13 |

Further, Table.5.1 provides a comparison of the proposed TIM scheme ($T_{6,0}$) (as per the equations 5.5 and 5.6 ) with BIM and TEC for average error. TIM scheme with 1-bit and 2-bit MFP design has at least 54% improvement in error compared to both TEC of 3-bit MFP and BIM in second iteration. Moreover, TIM design with 3-bit and 4-bit MFP has 6X and 2.5X better precision compared to TEC, 3-bit and 4-bit, respectively in second iteration. In third iteration, the average error of the proposed scheme with 3-bit and 4-bit MFP is 152X and 65X better compared to TEC, 3-bit and 4-bit, MFP respectively.

Having established the superiority of the $T_{6,0}$ technique over BIM and TEC, the following Table 5.2 provides a comparison of the same with TIM ($T_{4,1}$). An improvement in maximum and average error in truncated multiplier ($T_{4,1}$) compared to $T_{6,0}$ is evident from this Table. Finally, as mentioned earlier in Section 5.2.2, irrespective of the number of iterations, $T_{6,0}$ is used for initial iterations while the final iteration is carried out using $T_{4,1}$. For instance, if total number of iterations is three, the initial two iterations are performed using $T_{6,0}$ while the third iteration is carried out using $T_{4,1}$.

## 5.3.1 Area and Delay Comparison of Various Multipliers using Unit Gate Level Modeling

All the logarithmic multipliers have been analyzed using unit gate modeling as this approach provides a decent model for estimating the real cost of each component and does not depend strongly on any synthesis tool. Design metrics, area ($A$) and delay ($D$), have been considered

Table 5.2: A comparison of maximum and average error (%) in the proposed TIM ($T_{4,1}$) for 3 iterations

| | No. of MFP | Maximum Error (%) | | | | | | Average Error (%) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme | bits | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio | Iteration 1 | Ratio | Iteration 2 | Ratio | Iteration 3 | Ratio |
| TIM ($T_{6,0}$) | | 16 | 1 | 2.25 | 1 | 1.11 | 1 | 5.18 | 1 | 0.24 | 1 | 0.011 | 1 |
| TIM ($T_{4,1}$) | 1-bit | 15.91 | 0.99 | 2.2 | 0.97 | 1.01 | 0.9 | 4.02 | 0.77 | 0.2 | 0.83 | 0.0107 | 0.97 |
| TIM ($T_{6,0}$) | | 14 | 1 | 1.75 | 1 | 0.82 | 1 | 4.12 | 1 | 0.16 | 1 | 0.006 | 1 |
| TIM ($T_{4,1}$) | 2-bit | 13.46 | 0.96 | 1.52 | 0.86 | 0.5 | 0.6 | 3 | 0.72 | 0.12 | 0.75 | 0.005 | 0.83 |
| TIM ($T_{6,0}$) | | 12.5 | 1 | 1.19 | 1 | 0.67 | 1 | 3.85 | 1 | 0.13 | 1 | 0.005 | 1 |
| TIM ($T_{4,1}$) | 3-bit | 11.95 | 0.95 | 0.8 | 0.67 | 0.34 | 0.5 | 2.81 | 0.7 | 0.09 | 0.69 | 0.004 | 0.8 |
| TIM ($T_{6,0}$) | | 11.75 | 1 | 0.92 | 1 | 0.6 | 1 | 3.8 | 1 | 0.12 | 1 | 0.004 | 1 |
| TIM ($T_{4,1}$) | 4-bit | 11.11 | 0.94 | 0.56 | 0.68 | 0.3 | 0.5 | 2.72 | 0.71 | 0.08 | 0.67 | 0.002 | 0.5 |

and compared for all the designs. Assumptions made while calculating them are shown in Table 5.3. Each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates, for both area and delay. Moreover, a m-input gate is assumed to be composed of a tree of 2 input gates and the effects of wiring, buffering and inverting costs are neglected [59]. First, the unit gate modeling of 8-bit, 16-bit and 32-bit truncated logarithmic shifter in comparison with the logarithmic shifter is carried out and later it is extended to multiplier designs.

Table 5.3: Area and Delay metrics of basic design components

| Design Component (2-input) | Area (A) | Delay (D) |
|---|---|---|
| AND, OR, NAND, NOR Gates | 1 | 1 |
| EX-OR, EX-NOR, MUXes and Half Adder | 2 | 2 |

The area and delay of leading one detector (LOD), encoder, decoder and adder are computed as mentioned in [59] which are constant in the design. Similarly, the area $(A_{LS})$ and delay $(D_{LS})$ of the logarithmic shifter (LS) without truncation is computed based on equations 5.11 and 5.12 shown below. The area of truncated logarithmic shifter with t = 4 is calculated and shown in Table.5.4.

$$A_{LS}(n) = A_{mux}\left(n + \sum_{0}^{t} 2^{T}\right) \tag{5.11}$$

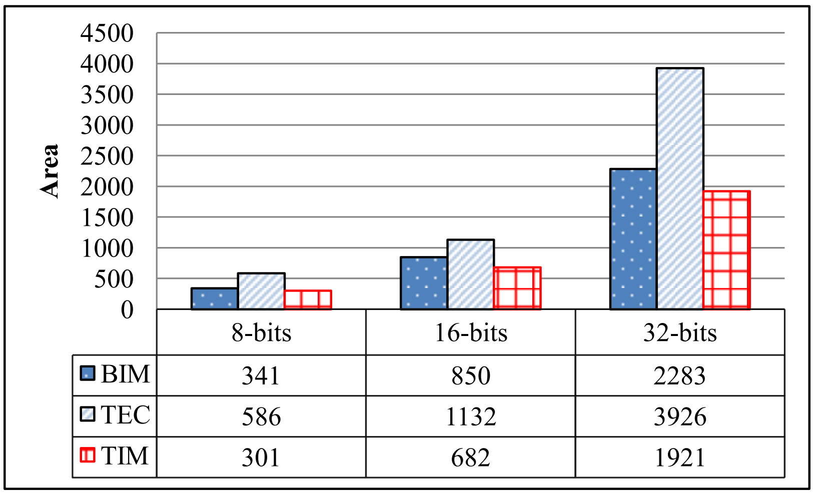
$$D_{LS}(n) = D_{mux} * log_2(n) \tag{5.12}$$

From Table.5.4, it is clear that the truncated logarithmic shifter is more area efficient compared to the logarithmic shifter. For example, the 16-bit truncated shifter occupies 35% less area compared to the logarithmic shifter. However, the delay of the truncated shifter is same as that of normal shifter since the number of logic levels of computation does not change.

Table 5.4: Area of logarithmic and truncated logarithmic shifter computed using unit gate analysis

| Logarithmic Shifter | No. of bits | Area | Percentage |
|---|---|---|---|
| logarithmic Shifter [29] | 8-bit | 84 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 60 | 71% |
| Logarithmic Shifter [29] | 16-bit | 240 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 156 | 65% |
| Logarithmic Shifter [29] | 32-bit | 420 | 100% |
| Truncated logarithmic shifter (truncation width =4) | | 348 | 83% |

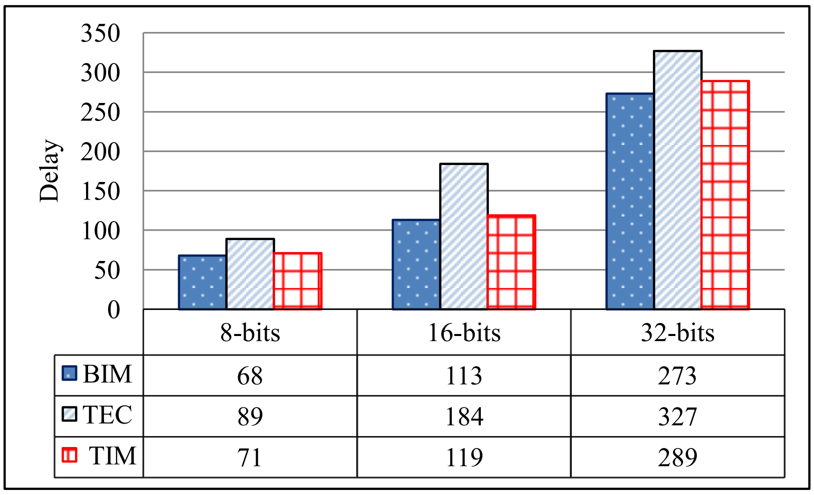
The unit gate area and delay analysis is carried out for various multiplier schemes in [26,27], and the proposed TIM designs for 8-bit, 16-bit and 32-bit. Results have been compiled and show in graph in Fig.5.9, that highlight the area savings achieved by TIM design compared to BIM. For example, the 16-bit TIM scheme occupies 20% less area compared to the BIM of same bit-width. The reason behind the area savings are attributed to the new truncation scheme proposed in this work. Similarly, it can be concluded from Fig.5.10 that the 16-bit TIM is faster compared to TEC however having a delay overhead of 5% compared to BIM of 16-bit width.



| | 8-bits | 16-bits | 32-bits |
|---|---|---|---|
| ■ BIM | 341 | 850 | 2283 |
| ▨ TEC | 586 | 1132 | 3926 |
| ▦ TIM | 301 | 682 | 1921 |

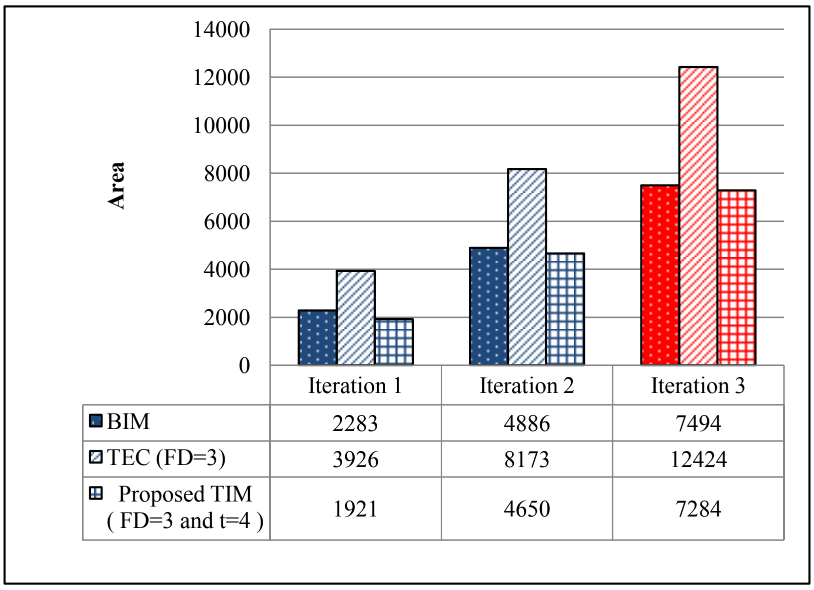
Figure 5.9: Area comparison of various multipliers for different bit-widths

Figures 5.11 & 5.12 present the area and delay improvement achieved by different 32-bit iterative multiplier schemes for three iterations. It is observed from Fig.5.11 that the area occupied by TIM is less compared to BIM and TEC in all iterations. For example, 32-bit TIM scheme in the first iteration occupies at least 19% less area compared to BIM and TEC designs.



| | 8-bits | 16-bits | 32-bits |
|---|---|---|---|
| ▨ BIM | 68 | 113 | 273 |
| ▨ TEC | 89 | 184 | 327 |
| ▦ TIM | 71 | 119 | 289 |

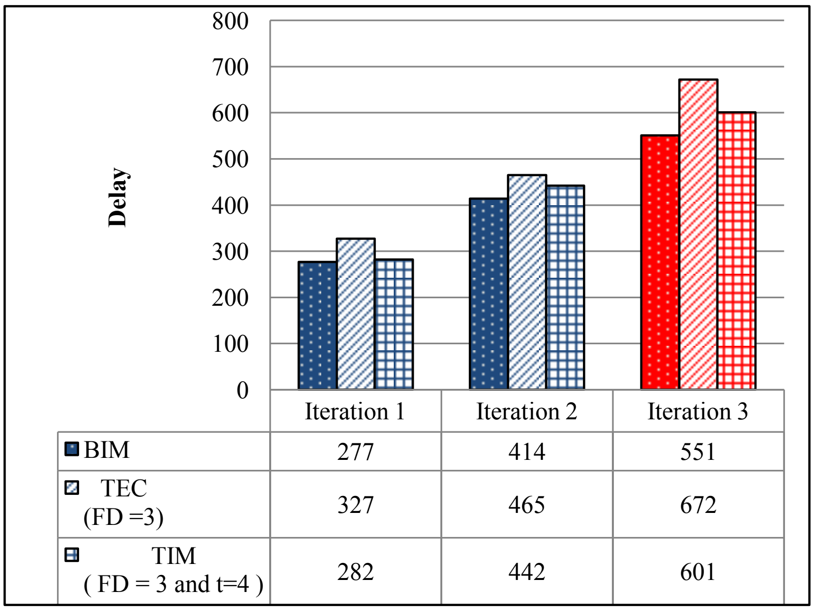
Figure 5.10: Delay comparison of various multipliers for different bit-widths



| | Iteration 1 | Iteration 2 | Iteration 3 |
|---|---|---|---|
| ▨ BIM | 2283 | 4886 | 7494 |
| ▨ TEC (FD=3) | 3926 | 8173 | 12424 |
| ▦ Proposed TIM ( FD=3 and t=4 ) | 1921 | 4650 | 7284 |

Figure 5.11: Area comparison of various multipliers for three iterations

The delay comparison of various multipliers for three iterations is presented in Fig.5.12. It is observed that TIM method performs better than TEC in terms of delay for same precision. However, it appears to have more delay compared to BIM which is not surprising because the precision provided by BIM is much less compared to TIM and thus needs more iterations leading to more delay if a precision similar to that of TIM is to be achieved. The delay overhead in TEC is due to carry speculation. The proposed design overcomes this problem by successfully replacing the complex fractional predictor design with a simple one (MFP) as discussed in Section 5.2.3.2.



| | Iteration 1 | Iteration 2 | Iteration 3 |
|---|---|---|---|
| ■ BIM | 277 | 414 | 551 |
| ▨ TEC (FD =3) | 327 | 465 | 672 |
| ⊞ TIM ( FD = 3 and t=4 ) | 282 | 442 | 601 |

Figure 5.12: Delay comparison of various multipliers for three iterations

### 5.3.2 Synthesis Results of Various Multipliers

For a fair comparison, TEC and BIM multipliers and the TIM have been modeled using Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1. They are mapped on to TSMC 180nm technology slow-normal library using cadence RTL compiler v7.1.

Table 5.5: Synthesis results of various 32*32 multipliers for one iteration

| 32 * 32 Multiplier | Area ($\mu m^2$) | % change | Delay($ps$) | % change | Area-Delay Product ($*10^5$) ($\mu m^2 - ps$) | % change |
|---|---|---|---|---|---|---|
| BIM [26] | 3675 | 100% | 15454 | 100% | 568 | 100% |
| TEC [27] | 5696 | 155% | 18235 | 117% | 1038 | 182% |
| Proposed TIM ($T_{6,0}$) | 3285 | 90% | 16088 | 104% | 528 | 93% |

Hardware synthesis has been carried out to compare the important metrics such as area and delay. Table.5.5 provides a performance comparison of 32*32 multiplier designed with above three schemes for one iteration. As seen from the Table, the area consumed by the proposed TIM scheme is 10% less compared to BIM and 65% less compared to TEC. While TIM has a delay overhead of 4% compared to BIM, it has a 64% better precision than BIM. Although, BIM performs better in delay with more iterations, TIM achieves a much better precision compared to BIM with only a marginal increase in delay. The reason for increase in delay in TIM scheme is due to the inclusion of mask from second iteration onwards. Nevertheless, TIM performs much better in terms of precision, area and delay compared to TEC. It is also evident from Table.5.5 that TIM scheme has improved area-delay product (7 to 89%) compared to TEC and BIM respectively. Overall, TIM scheme outperforms all other similar designs that exist in the literature.

## 5.4 Benchmarking Various Multiplication Schemes-Application to Image processing

Image sharpening is an important image enhancement technique employed in image processing applications. The computational process of sharpening an image involves a number of fixed point multiplications. It is therefore a good application to prove the efficacy of the proposed truncated iterative multiplier.

### 5.4.1 Image Sharpening Algorithm

Human perception is highly sensitive to edges and fine details of an image. Since images essentially consist of high-frequency components their visual quality is corrupted if these high frequencies are removed. Conversely, increasing the high-frequency components of an image improves the image quality. Image sharpening algorithm described in [60] is one such enhancement technique which highlights the edges and fine details in an image.

This algorithm described below, accepts an image, processes it, and produces an image of high quality. Suppose $I$ is the original image, the processed image $S$ is described using the expression

$$S(x,y) = 2I(x,y) - M \tag{5.13}$$

where $M = \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} H(i+3, j+3)\, I(x-i, y-j)$

and $H$ is a matrix defined as

$$H = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Since this expression involves a number of multiplications, an exact multiplier such as an array multiplier can perform these operations accurately thereby producing an image of high quality. On the other hand, using an approximate multiplier would result in an image of certain quality which is quantified using established metrics such as mean square error (MSE) and peak signal to noise ratio (PSNR).

The MSE represents the loss of information in the image and is expressed as,

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{5.14}$$

The peak signal to noise ratio (PSNR) in dB is expressed using $MSE$ as follows:

$$PSNR \text{ in dB} = 10 log_{10} \left( \frac{MAX_I^2}{MSE} \right) \tag{5.15}$$

where $MAX_I$ represents the maximum possible pixel value of the image.

While the use of approximate multiplier affects the image quality, it has the advantage of savings in terms of area and delay as compared to an accurate multiplier. In what follows, the performance of the existing multipliers such as BIM and TEC and the proposed TIM is studied and compared with reference to the image sharpening algorithm. The algorithm is applied to blocks of 5*5 pixels on a set of standard images ( Cameraman and Lena). The exact multiplications are replaced by approximate multiplications using BIM, TEC and TIM, while addition, subtraction and division operations are carried out using accurate techniques. The metric MSE is computed by finding the mean of squares of difference in pixel values between original image and the processed image using approximate multipliers and these values are substituted in equation (5.15) to calculate PSNR values.

Table 5.6 provides a comparison of these metrics on a set of standard images (Lena and

Cameraman). While PSNR of the proposed TIM is less compared to that of TEC in the first iteration, the same gets better with higher number of iterations. This is due to the increase in number of iterations, the accuracy of TIM improves over that of TEC. It can also been seen from Fig.5.13 where maximum and average error for all the techniques is compared.

Table 5.6: A comparison of values of MSE and PSNR for benchmark images using various multiplier schemes

| | | 1st Iteration | | | 2nd Iteration | | | 3rd Iteration | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Image | Metric | BIM | TEC | TIM | BIM | TEC | TIM | BIM | TEC | TIM |
| Cameraman | | 43.2 | 43.4 | 43.4 | 43.46 | 43.6 | 43.8 | 43.7 | 44.2 | 44.45 |
| Lena | | 39 | 39.4 | 39.2 | 39.2 | 39.5 | 39.53 | 39.4 | 39.5 | 39.7 |
| Pirate | PSNR ( dB) | 42.8 | 43 | 42.8 | 43.1 | 43.2 | 43.4 | 43.2 | 43.4 | 43.7 |
| Cameraman | | 3.1 | 3 | 3 | 2.95 | 2.8 | 2.7 | 2.8 | 2.5 | 2.35 |
| Lena | | 7.9 | 7.5 | 7.7 | 7.75 | 7.35 | 7.3 | 7.5 | 7.2 | 7.05 |
| Pirate | MSE | 3.42 | 3.3 | 3.39 | 3.2 | 3.15 | 3 | 3.1 | 3 | 2.8 |

Further, BIM has the highest error for all iterations while between TEC and TIM, the latter has relatively larger error in the first iteration which however falls sharply from the second iteration onwards. While the difference between TEC and TIM for higher number of iterations is not apparent from the graph, it is clear from Table 5.1 that TIM technique has a maximum error and an average error that is less than that of TEC.

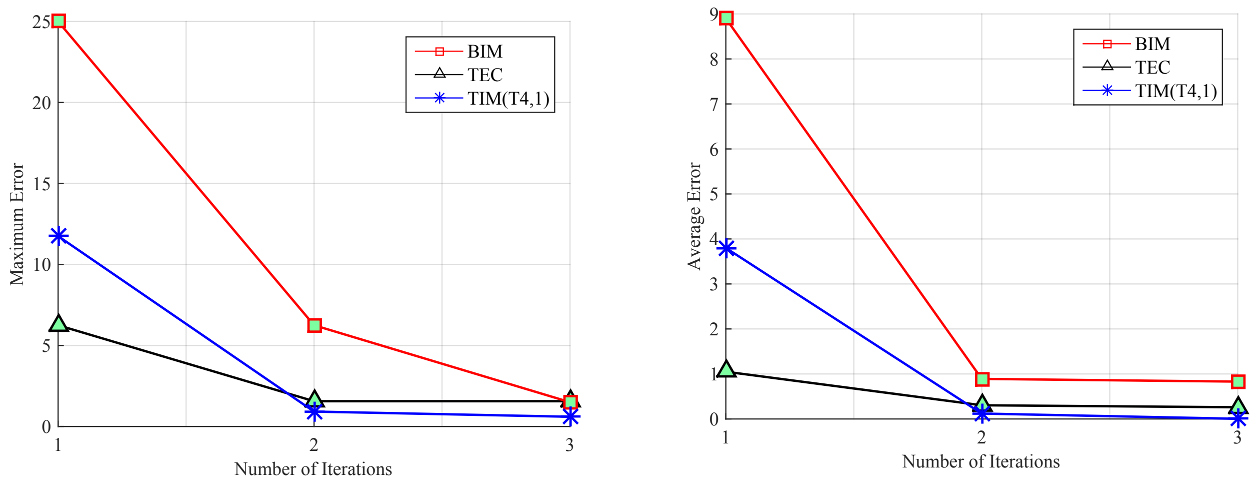


Figure 5.13: Maximum and Average errors of various multipliers for three iterations

In what follows, BIM, TEC, and TIM are compared for precision while keeping in mind the area and delay performance of the hardware. Figures.5.14 and 5.15 provide a comparison of area and delay performance of various multiplier schemes. It is evident from these figures that

TIM performs better than TEC. Although, as stated earlier, TEC has better precision for first iteration, TIM performs better from the second iteration onwards. Also, TIM performs better in terms of both area and delay for any number of iterations.

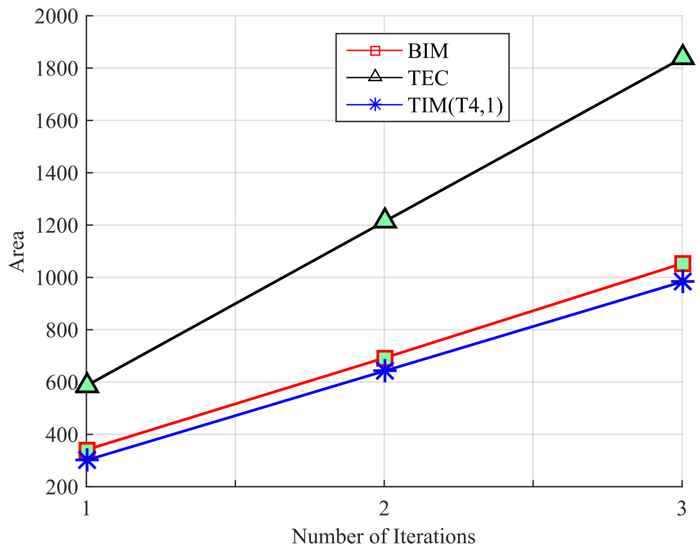


Figure 5.14: Unit gate area statistics of various multiplier for three iterations

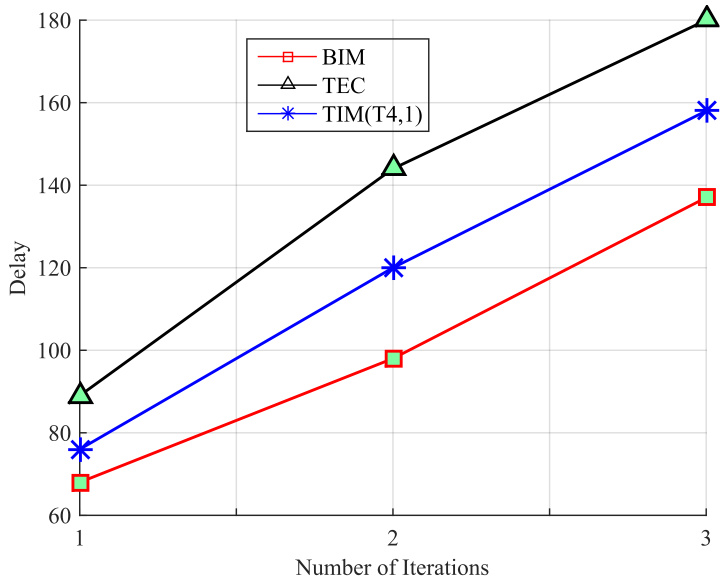


Figure 5.15: Unit gate delay statistics of various multiplier for three iterations

Further, while TIM has a slightly better performance than BIM in terms of area, BIM does well in terms of delay (Fig.5.15). However, this is not considered significant in the current context since TIM has better precision as well as better PSNR and MSE for the benchmark images considered, compared to BIM.

Also, the performance of the proposed multiplier, in terms of both PSNR and MSE, may be understood by observing the images processed by it. Figure.5.16 illustrates the images that are processed using exact (array) and the TIM. It can be observed that the images of Lena and Cameraman processed with the proposed multiplier look very similar to the original ones.

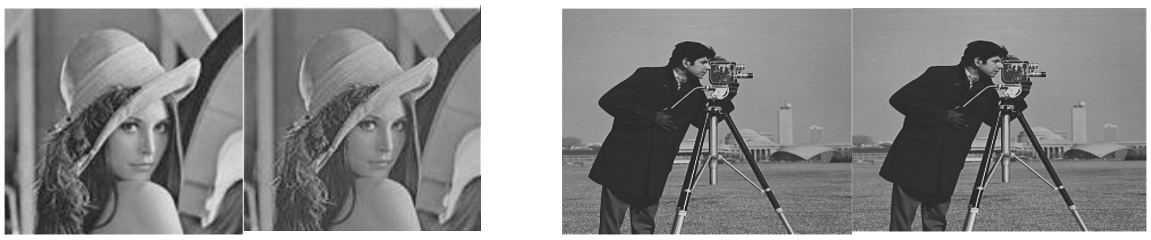


Figure 5.16: Lena and Cameraman images obtained using exact and the proposed multiplier

## 5.5 Conclusions

In this work, an improved iterative multiplier has been proposed based on Mitchel algorithm with enhanced precision. A new fractional detector scheme and a modified truncation method presented significantly reduce the area of the related hardware. The fractional detector logic and its efficient precomputation contribute to improved overall accuracy due to fewer number of iterations required compared to the existing ones. Further, the precision of the multiplier improves as the number of iterations increases. Performance improvement has been achieved through the use of truncated logarithmic shifter and a fractional predictor.

Extensive analysis of the existing (TEC and BIM) and the proposed (TIM) schemes has been carried out using unit gate modeling and compared with that obtained using synthesis tool. Results of accuracy and hardware performance prove the superiority of the TIM technique over TEC and BIM. The same has also been validated using image processing benchmarks, Lena, cameraman, and pirate.

# Chapter 6

# An Improved 'Digit-by-Digit' Decimal Multiplier

## 6.1 Introduction

The previous chapters presented improved designs of binary and logarithmic multipliers. However, it may be noted that decimal arithmetic is preferred in applications such as financial, scientific and commercial etc. owing to their higher precision compared to binary arithmetic. However, these computations are generally sluggish (slow) and tend to occupy more silicon area [48]. This has led to efforts in improving decimal architectures to enable high performance and compact arithmetic circuits. Like in binary arithmetic, one of the most vital and common operations in decimal arithmetic, is multiplication. While a large body of literature on decimal arithmetic covers serial multiplication [62, 63], parallel ('word-by-digit') [40–43] and ('digit-by-digit') [44, 45] multiplication has also been reported recently. Decimal (BCD) 'digit-by-digit' multipliers are appropriate for pipelined computations and result in improved regularity of the circuits. This regularity, in conjunction with shorter interconnects, results in improvement in the multiplier performance [46]. In this work, we focus on developing efficient architectures for decimal 'digit-by-digit' multiplication.

The partial products in 'digit-by-digit' multiplication scheme are generated using BCD digit-multiplier (BDM) and their reduction is accomplished using carry-free binary adders, multi-operand binary to decimal (BD) converters and decimal adder. Since BDM is an important component in partial product generation, we focus on new and improved designs for BDM cells in this chapter. Besides, novel designs of multi-operand BD converters are proposed to convert the column binary sum to decimal in partial product reduction. Further, a hybrid multi-operand BD converter algorithm is proposed and analyzed for its performance. It is expected that these improvisations would result in significant savings in terms of area and latency.

Throughout this chapter, upper and lower case letters are used to signify decimal digits and binary bits respectively, where a digit represents a 4-bit BCD number. The symbols ‘ . ’ and ‘ + ’ are used to denote AND and OR gates while the symbols ‘$\oplus$’ and ‘$\odot$’ denote XOR and XNOR operations respectively. Further, the term binary coded decimal (BCD) is used interchangeably with decimal.

Rest of the chapter is organized as follows: An outline of the proposed partial product generation and reduction schemes in 16*16 ‘digit-by-digit’ multiplier is provided and discussed in Sections 6.2 and 6.3. In addition, design of hybrid multi-operand BD converters is described in Section 6.3.2. A detailed performance analysis of 16*16 ‘digit-by-digit’ multiplier is carried out and compared in Section 6.4.

## 6.2 A New Partial Product Generation Scheme in ‘Digit-by-Digit’ Multiplier

This section presents two new partial product generation (PPG) schemes for improved area and performance of BDM cell in ‘digit-by-digit’ multiplication. This is achieved through novel designs of PPBD converter which forms a part of the BDM cell.

### 6.2.1 High Performance Partial Product Binary to Decimal (PPBD) Converter

The first of the proposed two PPBD converters, the ‘high performance’ PPBD converter, is designed using the fast BD (FBD) converter cells. A typical FBD cell, would accept a 4-

bit binary input $(b_j)$, multiplies it by four, and then adds it to $b_i$ of 2-bits as illustrated in Fig.6.1(a). Thus the computation of BCD (decimal) outputs, $H_0$ (higher digit) and $L_0$ (lower digit) is carried out by using the relation $\{H_0, L_0\} = 4\,.\,b_j + b_i$ where the maximum values of $H_0$ and $L_0$ are $(3)_{10}$ and $(9)_{10}$ respectively. Therefore, the output of FBD cell is limited at most to $(39)_{10}$ unlike $(19)_{10}$ in Nicoud cell [50]. The binary inputs, $b_j$ and $b_i$ comprise of $\{b_3, b_2, b_1, b_0\}$ and $\{b_{i1}, b_{i0}\}$ bits respectively.

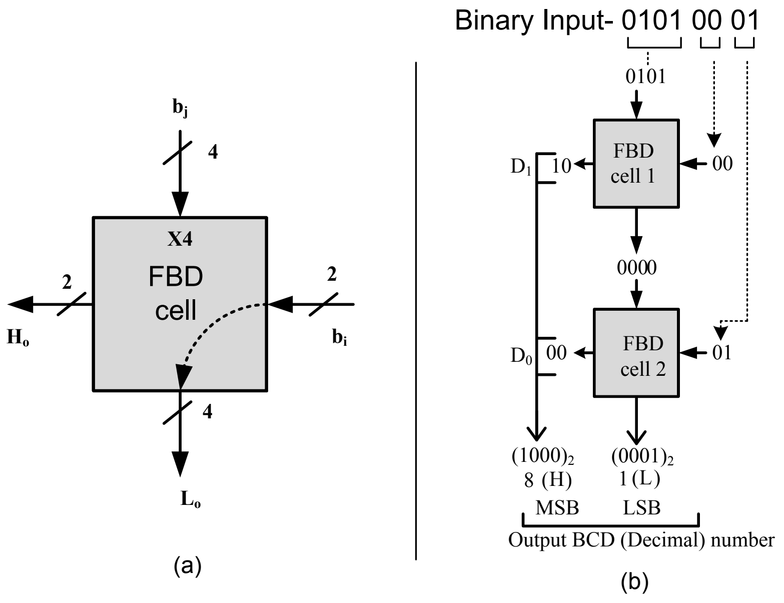


Figure 6.1: (a) Compact notation of FBD cell (b) Linear array of FBD cells to form PPBD converter

The FBD cell is shown in Fig.6.1(a) and is described by the following equations derived using truth tables :

$$\begin{cases} H_0\,[1] = b_3 + b_2.b_1'.b_0' + b_2'.b_1.b_0 + b_2'.b_1.b_{i0} \\ H_0\,[0] = b_3 + b_2.(b_1 + b_0) \\ L_0\,[3] = b_{i0}.b_3.b_0 + b_1.(b_{i0}'(b_2 \odot b_0) + b_1'.(b_2 b_{i0} b_0') \\ L_0\,[2] = b_3.b_0'.b_{i0} + b_2.b_0'.(b_1 + b_{i0}') + b_2'.b_0.(b_1' + b_3'.b_{i0}) \\ L_0\,[1] = b_2'.(b_1'.(b_3 \oplus b_{i0}) + b_2.(b_0'.(b_1 \odot b_{i0}) + \emptyset \\ L_0\,[0] = b_{i1} \end{cases} \qquad (6.1)$$

where $\emptyset = b_0(b_2'.b_1.b_{i0}' + b_2.b_1'.b_{i0})$

An example to convert a 7-bit binary number to two BCD digits (H and L) is illustrated in Fig.6.1(b). Since the binary number $(01010001)_2$ to be converted is larger than $(39)_{10}$, two FBD cells are required to realize the converter. As illustrated in Fig.6.1(b), the input to the FBD cell 1 is restricted to $(1001)_2$. Hence the 3 MSBs of binary input along with a ‘0’ prepended $(0101)_2$ are accepted as $b_j$ and the next significant two binary input bits “00” as $b_i$ results in the outputs $(10)_2$ and $(0000)_2$. The 4-bit output $(0000)_2$ of cell 1 along with residual binary input “01” form inputs to cell 2, resulting in higher (H) digit $(1000)_2$ formed by $\{D_1, D_0\}$ and lower (L) digit $(0001)_2$. In general, binary number of any operand width can be converted to BCD (decimal) by a linear arrangement of FBD cells.

Since a BCD digit can take values only between $(0)_{10}$ and $(9)_{10}$, the output of the 4*4 binary multiplier in BDM illustrated in Fig.2.24 is restricted to $(81)_{10}$, that is equivalent to $(1010001)_2$. Thus, the input binary number to a PPBD converter is limited to $(1010001)_2$. This gives a possibility for optimization of FBD cell 1 shown in Fig.6.1(b), which eventually results in simplified PPBD converter. As a consequence, the input to the FBD cell 1 is restricted to $(0101)_2$ as depicted in Fig.6.1(b). In view of this, the FBD cell 1 in Fig.6.1(a) gets simplified resulting in ‘low area’ BD (LABD) cell as shown in Fig.6.2(a). The binary inputs $\{b_2, b_1, b_0\}$ and $\{b_{i1}, b_{i0}\}$ are applied to this cell which converts them into equivalent decimal number $H_0\,[1:0]$ and $L_0[3:0]$.

The resulting simplified equations for LABD cell which can be obtained from truth table can be written as :

$$\begin{cases} H_1\,[1] = b_2.b_0^{'} + b_1.(b_0 + b_{i0}) \\ H_0\,[0] = b_2.b_0 \\ L_0\,[3] = b_1.b_0^{'}.b_{i0}^{'} + b_2.b_{i0} \\ L_0\,[2] = b_0.(b_2^{'}.b_1 + b_{i0}) + b_2.b_0^{'}.b_{i0}^{'} \\ L_0\,[1] = b_{i0}^{'}.(b_2.b_0^{'} + b_1.b_0) + b_2^{'}.b_0^{'}.b_{i0} \\ L_0\,[0] = b_{i1} \end{cases} \qquad (6.2)$$

Figure.6.2(b) depicts the 'high performance' PPBD (HPPPBD) converter scheme achieved by a linear arrangement of LABD and FBD cells. This converter has improved performance in terms of delay as illustrated later.

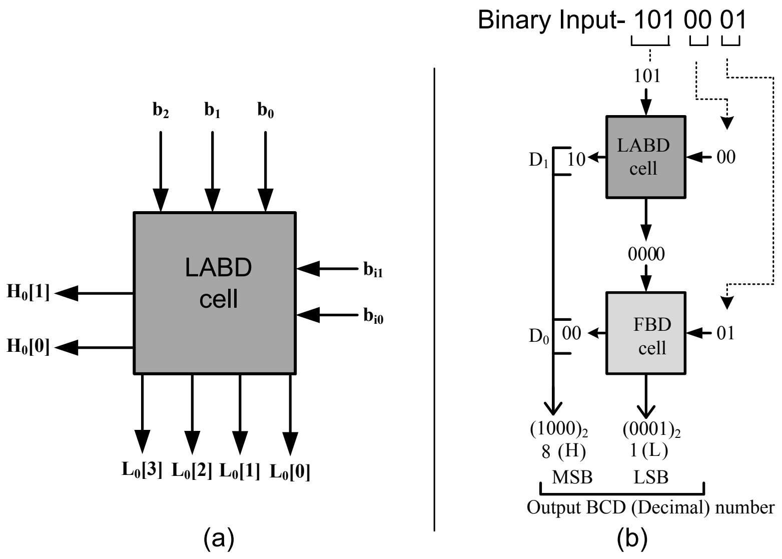


Figure 6.2: (a) Compact notation of LABD cell (b) Linear array of LABD and FBD cells to form 'high performance' PPBD (HPPPBD) converter

### 6.2.2 Low Area Partial Product Binary to Decimal Converter (LAPPBD)

Work in [51] presents partial product reduction using a multi-operand BD converter consisting of Nicoud cells as illustrated in Fig.6.3(a). We propose a design of PPBD converter using Nicoud cells and FBD cells which however are used for partial product generation to achieve area reduction as well as improve the performance. This design comprises of a linear array of LABD cells (shown earlier in Fig.6.2(a)) and Nicoud cells as illustrated in Fig.6.3(b). Since Nicoud cells have more latency, achieving a PPBD converter with these cells alone results in a slower design. An alternative method is to use area optimized and faster LABD cell so as to improve the performance of PPBD converter.

Referring to Fig.6.3(a), Nicoud cells (circled with dotted lines) are replaced with LABD cell resulting in a PPBD converter shown in Fig.6.3(b) which has the advantage of lower area as compared to all Nicoud cell converter.

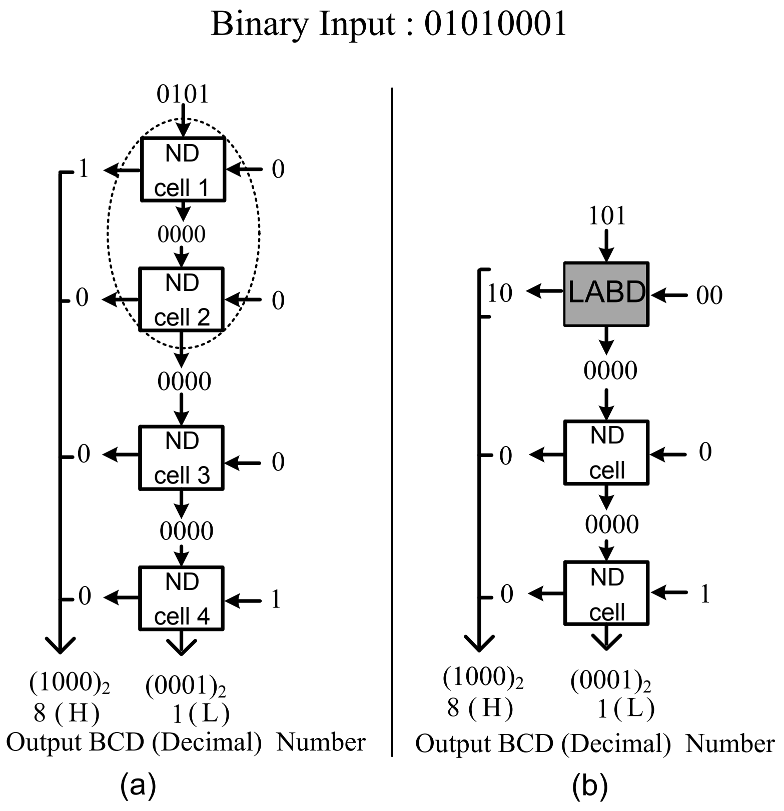


Figure 6.3: (a) Iterative array of Nicoud cells to form BDM (b) Linear array of Nicoud and LABD cells to form LAPPBD

## 6.3 Partial Product Reduction (PPR) in 'Digit-by-Digit' Multiplier

This section discusses the decimal (BCD) partial product reduction scheme in 'digit-by-digit' multiplier using the proposed multi-operand BD converters. An example of this approach is illustrated in Fig.6.4. Referring to decimal partial product of column length C=6 with six partial products, each partial product consists of 4-bits, denoted with solid dots. The columns c0, c1, c2 and c3 are reduced to two rows using tree of 3:2 binary Carry Save Adders (CSA) which requires 2 levels to compress all the columns into two rows. These two rows are reduced to further obtain the final binary result using the carry propagation adder represented with straight line in grey color as shown in Fig.6.4.

A numerical example illustrates the addition of six BCD digits, each of 4-bits $[(9)_{10} + (9)_{10} + (9)_{10} + (9)_{10} + (9)_{10} + (9)_{10}]$, as mentioned in Fig.6.5 (a). The maximum value of each digit is $(9)_{10}$. Addition of these digits is performed by a tree of CSAs and a carry propaga-

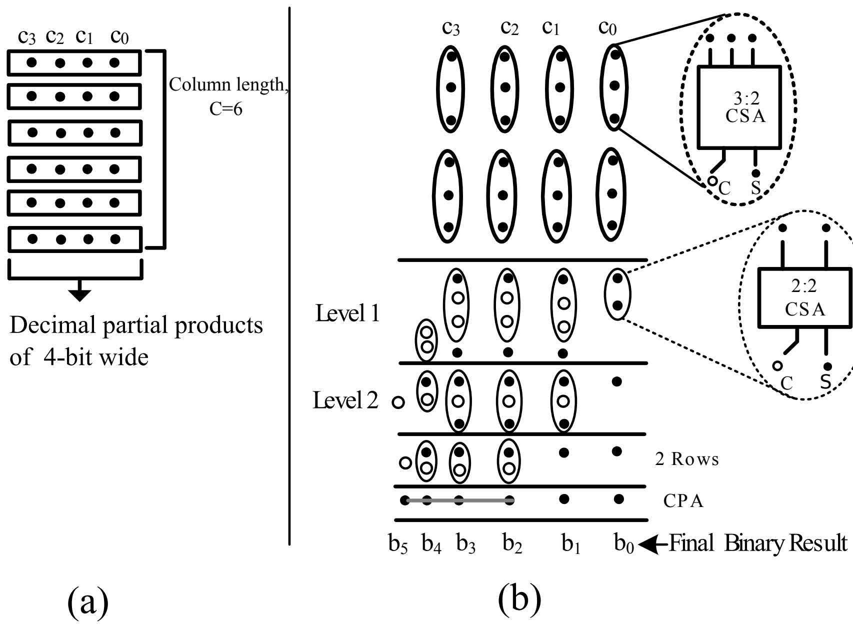


Figure 6.4: (a) Decimal partial products of six columns each of 4-bit (b) Example of partial product, denoted using dot, reduction of column size C= 6

tion adder resulting in final binary result $(110110)_2$. The conversion of this binary number to decimal (BCD) is accomplished by using the proposed multi-operand binary to decimal converter. The two FBD cells are connected in a linear array to convert binary number $(110110)_2$ to decimal $(54)_{10}$ as illustrated in Fig.6.5(b).

The implementation details of partial product reduction in 16*16 'digit-by-digit' multiplier is presented below.

### 6.3.1 Implementation of 16*16 'Digit-by-Digit' Multiplier

In a typical $N*N$ 'digit-by-digit' multiplication, $2N^2$ decimal partial products with a maximum column size of $2N-1$ are generated using BDM cells. As mentioned in Section 2.8.1, multi-operand BD (MBD) converters are used at the partial product reduction stage. The reduction scheme using these converters for $16*16$ 'digit-by-digit' multiplier is illustrated in Fig.6.6. It can be seen here that the layout of decimal partial products ($H$ and $L$) is in columns of varied length as illustrated in Fig.6.6(a). The reduction of decimal partial products generated happens in two steps: First, all the partial product columns are compressed in parallel using CSAs . Next, the binary number obtained from respective columns is converted in to a decimal number using the multi-operand binary to decimal (MBD) converters.

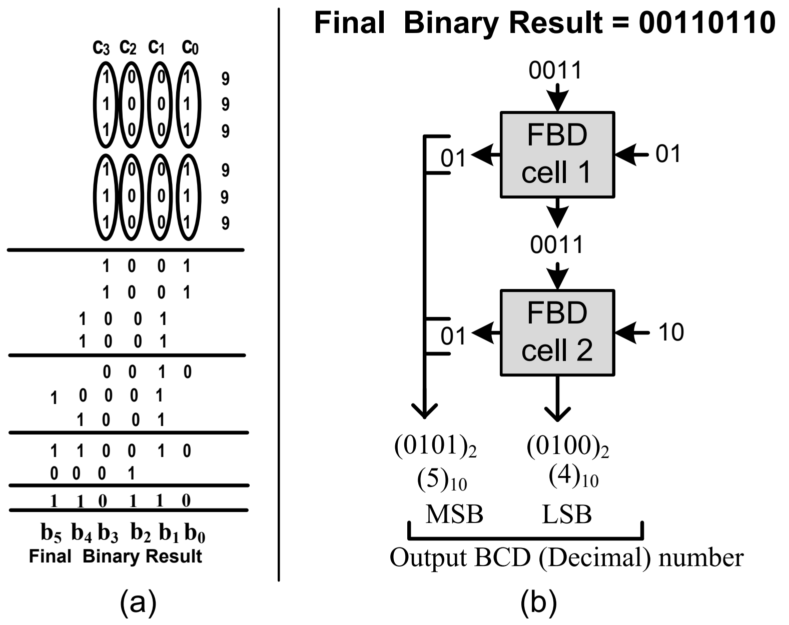
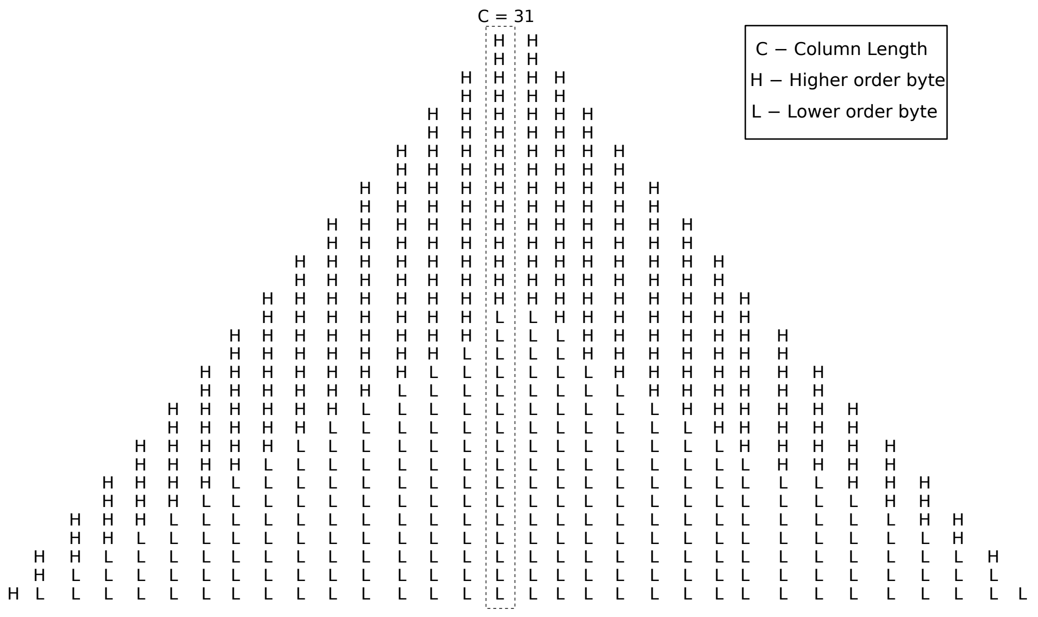


Figure 6.5: (a) Numerical example illustrating the reduction of decimal (BCD) partial products using CSAs (b) MBD converter formed using linear array of FBD cells to convert binary number to decimal (BCD)

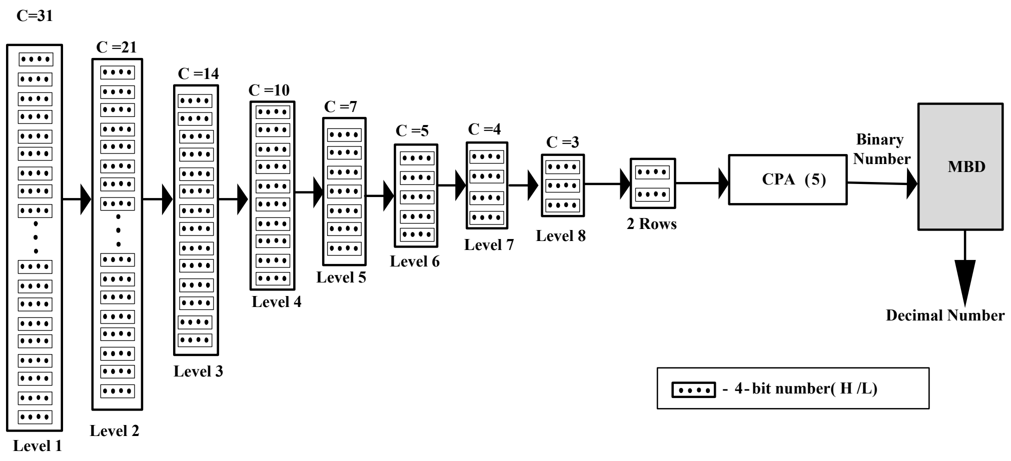
In this work, the existing MBD converter [51] is modified by replacing Nicoud cell with FBD cell resulting in improved performance. The operation of the proposed MBD converter for the largest column, $C = 31$ (highlighted by using dotted line in the Fig.6.6 (a)), is illustrated in Fig.6.6(b) as an example. The binary addition of 31-digits column comprising of $H$ and $L$ (whose maximum values, mentioned earlier, are considered) is performed using a tree of 3:2 binary CSAs [52]. It requires eight reduction levels to compress this column into 2 rows, which are eventually reduced to a binary number using the carry propagation adder (CPA). For instance, considering the case when output of the column size (C=31) results in a binary number $(0100001000)_2$, it is converted into decimal in two stages using a linear connection of FBD cells (MBD converter) as shown in Fig.6.7.

In a similar manner, binary result from respective columns are compressed in parallel to obtain decimal numbers which are then aligned according to their decimal weight and reduced to a product using the decimal adder [64].

While this approach improves the performance by reducing the latency of the MBD converter, it also results in area overhead. Further, decimal conversion of a column using Nicoud cells [51] alone, while resulting in smaller area as illustrated in Fig.6.8(a) it increases the converter latency significantly. To address these competing requirements, a hybrid multi-operand



(a)



(b)

Figure 6.6: (a) Partial product matrix in a 16*16 ‘digit-by-digit’ multiplier (b) Partial product reduction of column of largest size C=31 using CSA structure

BD converter using a mix of FBD and Nicoud cells is proposed that results in small area but high speed of operation. For instance, Fig.6.8(b) shows a hybrid MBD converter consisting of fast FBD cells in the critical path (stage1) while stage 2 comprises of (small area) Nicoud cells resulting in a hybrid MBD converter that consumes less area without compromising on speed when compared to the converter shown in Fig.6.7. Clearly, the hybrid converter is faster with only a marginal increase in area when compared with that in Fig.6.8(a). As the column size increases, selection of optimal number of FBD and Nicoud cells becomes tedious. Hence, a hybrid multi-operand BD algorithm which helps in selecting the best combination of FBD and

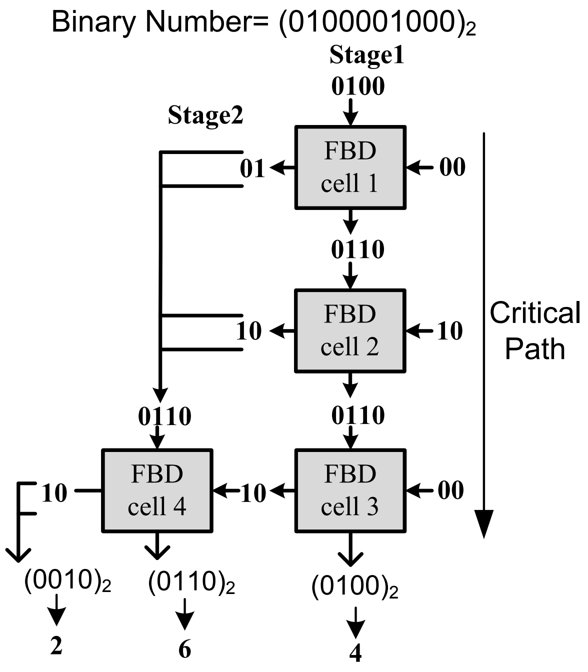


Figure 6.7: MBD converter for column size, C=31 in 16*16 ‘digit-by-digit’ multiplier

Nicoud cells is also proposed in this work and is discussed below.

### 6.3.2 Algorithm for Hybrid Multi-operand Binary to Decimal Converter

The algorithm for the design of hybrid multi-operand BD converter uses number of bits (*nob*) to compute all possible combinations of Nicoud and FBD cells. The pseudo code of the algorithm (given below) accepts the column size (C) and converts it into ‘*nob*’ using the relation $nob = ceil(log_2(C * P))$, where P is the maximum decimal weight of partial product $((9)_{10})$. For instance, given a value of $C = 31$ and $P = 9$, the value of ‘*nob*’ turns out to be 9. Depending on the number of bits, the algorithm recursively computes all possible combinations of Nicoud (ND) and fast binary to decimal (FBD) cells and their respective area and delay. The selection of appropriate multi-operand BD converter itself is based on the area and delay requirements of the design on hand. The area of individual FBD and ND cells is denoted by $A_1$ and $A_2$ respectively.

The algorithm, which calls three sub-functions Max_ND_Cell, Max_FBD_Cell and CAL is explained as follows :

1. The input to the algorithm is the binary number obtained by reducing the partial product column of length C.

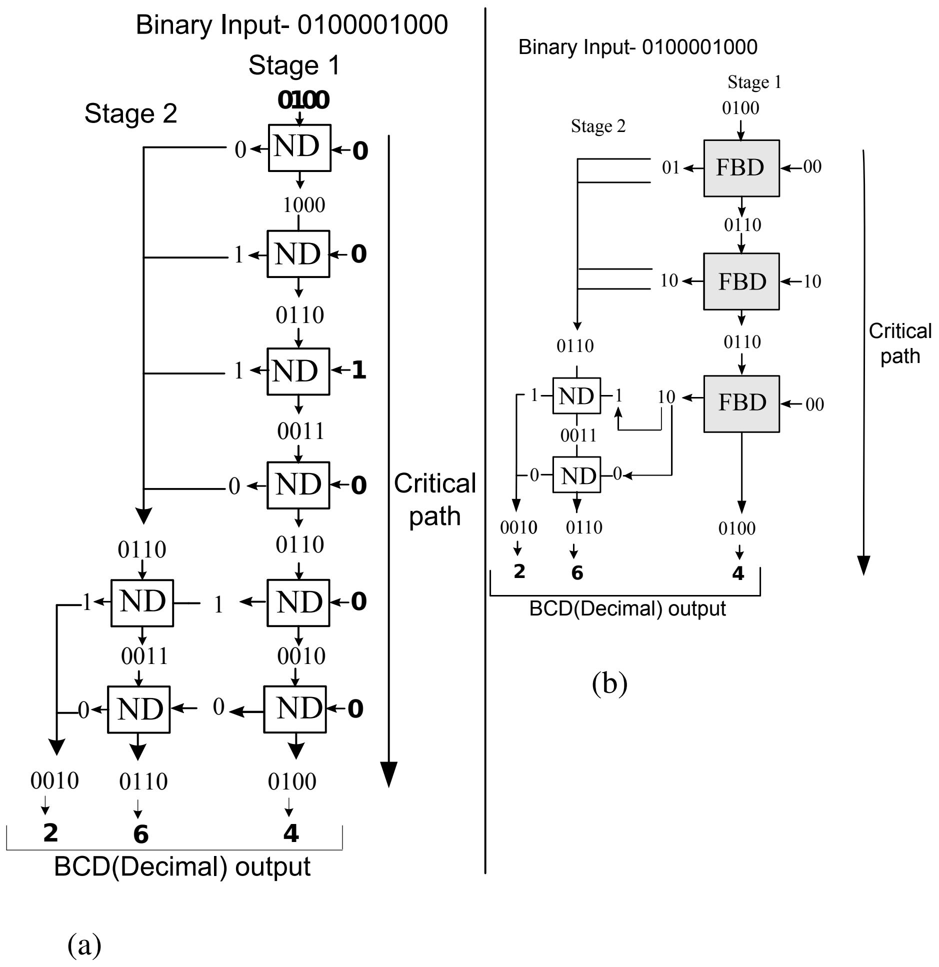


Figure 6.8: MBD converter for column size, C=31 in 16*16 'digit-by-digit' multiplier (a) Using Nicoud cells (b) Hybrid converter using Nicoud and FBD cells

2. The sub-function Max_ND_Cell computes the maximum number of ND cells (mndc) required to design the converter.
3. Similarly, the sub-function Max_FBD_Cell determines the maximum number of FBD cells (mfbdc) necessary to design the converter.
4. Higher priority is given for selection of FBD cells if the objective is to obtain a low latency design.
5. The sub-function CAL calculates the number of unused FBD $(m1)$ and ND $(m2)$ cells for various valid combinations.
6. For a given set of values $m_1$ and $m_2$, the algorithm computes the total number of utilized FBD $(n_1)$ and ND $(n_2)$ cells and their area as well as the critical path delay for that combination.

7. The area and delay for all valid combinations are calculated and stored in a list as a tuple $<n_1, n_2, delay, area>$.

8. The above steps are repeated till all valid combinations are exhausted.

**Algorithm 6.1** Pseudo code of hybrid multi-operand converter

**Input**: *nob*

**Output**: utilized FBD cells $(n_1)$, utilized ND cells $(n_2)$, area, delay

```
1.  function Hybrid_MBD
2.   mndc = Max_ND_Cell
3.   mfbdc = Max_FBD_Cell
4.    for i = 0; i < mndc + 1; i++
5.     for j = mfbdc + 1; j > -1; j–
6.      $m_1$, $m_2$= CAL
7.       if $delay != 0$
8.        if $m_1 >= 0$ and $m_2 >= 0$
9.         set $area$ to $((|i-m_1|)*A_1)+((|j-m_2|)*A_2)$
10.         if $[|i-m_1|, |j-m_2|, delay, area]$ not in list
11.          add $([|i-m_1|, |j-m_2|, delay, area])$ to list
12.         end if
13.        end if
14.       end if
15.     end for
16.    end for
17. return $(delay, area, n_1, n_2)$
```

It can be noted from Fig.6.8(b) that the critical path of the MBD converter is dictated by the first stage wherein the delay of individual cells chosen gets added up. However, in the subsequent stages the delay of final cell alone is added [51]. Therefore, to achieve a faster hybrid converter, delay per bit in the first stage has to be reduced by exclusively using FBD cells.

The algorithm terminates after storing all the values in a data set (list). No need to mention, area efficient converter can be designed using more number of ND cells. Conversely, if high speed converter is the objective, then more number of FBD cells can be used. Thus, the hybrid algorithm provides flexibility in choosing appropriate multi-operand converter.

## 6.4 Results and Discussion

In order to compare the proposed multiplier with existing designs, initially unit gate level modeling is carried out and later synthesis based analysis is performed.

### 6.4.1 Area and Delay Comparison using Unit Gate based Modeling

All the designs under consideration have been modeled using unit gate approach [59] to obtain a synthesis-independent estimate of area $(A)$ and delay $(D)$. Further, each two-input gate (AND, OR, NAND, NOR) is counted as one gate while EX-OR and EX-NOR are counted as two gates for both area and delay [59]. Moreover, an m-input gate is assumed to be composed of a tree of m-1 input gates while the effects of wiring, buffering and inverting costs (area and delay) are neglected.

As discussed earlier, a BDM typically consists of a binary multiplier and a partial product binary to decimal (PPBD) converter. In this work, the binary multiplier proposed by Jaberipur 2 [44], being the most efficient design in literature, is adopted for all the designs, shown in Table below, together with respective PPBD schemes to form the BDM. For instance, the HPBDM comprises of a Jaberipur binary multiplier and a HPPPBD.

Table 6.1 compares the area and delay performance of various BDM converters including the two proposed schemes (HPBDM and LABDM) mentioned earlier. Besides each value of area and delay, the percentage mentioned signifies how the proposed (HPBDM and LABDM) designs perform compared to the existing designs. It can be observed that an improvement of 5% in delay is achieved in proposed HPBDM compared to the best performing Split_4_3 BDM. Further, it can be seen that area-delay product of proposed LABDM design is better than every other design. Also, the area occupied by LABDM and HPBDM is less in comparison with

Table 6.1: Comparison of area and delay of various BDMs

| Scheme | Area | % change | Delay | % change | Area-Delay Product | % change |
|---|---|---|---|---|---|---|
| **HPBDM** | **130** | **107%** | **20** | **95%** | **2600** | **102%** |
| **LABDM** | **121** | **100%** | **21** | **100%** | **2541** | **100%** |
| N2BDM [65] | 119 | 98% | 22 | 105% | 2618 | 103% |
| Split_4_3 BDM [48] | 173 | 143% | 21 | 100% | 3633 | 143% |
| Bhatt BDM [47] | 175 | 144% | 22 | 105% | 3850 | 151% |
| Sree BDM [49] | 187 | 154% | 37 | 176% | 6919 | 272% |

other designs except N2BDM scheme.

As mentioned earlier, the decimal partial products obtained from BDMs are compressed using binary CSA tree, multi-operand converters and decimal adders. The main focus at PPR stage is the efficient design of multi-operand binary to decimal (MBD) converter. The PPR stage of present work is compared with that of Dadda [51] which provides an efficient implementation in ‘digit-by-digit’ multipliers. The main difference between the Dadda and proposed hybrid PPR schemes lies in MBD converters, which were realized using ND cells in Dadda scheme, while hybrid (ND and FBD) cells are used in the present scheme. Hence, to prove the efficacy of proposed hybrid MBD design, exhaustive comparisons are done at the multi-operand binary to decimal converter stage.

The unit gate area and delay statistics of Dadda, FBD and proposed hybrid MBD converters for various column (C) sizes in a 16* 16 multiplier illustrated in Fig.6.6 are shown in Table 6.2.

It can be noted that the number of cells (Dadda and/or FBD) required in each of the MBD designs depends on the number of bits as well as the binary number. Hence for the same ‘*nob*’ the number of cells required varies. From the results obtained it can be concluded that MBD converters formed using FBD cells alone have the speed advantage with, however, area overhead while employing Nicoud (ND) cells alone results in area efficient design with increased latency. On the other hand, MBD converters obtained from the hybrid converter algorithm (discussed in Section 6.3.2) have lesser delay compared to Dadda converters with marginal increase in area. Further, hybrid MBD converters are more area efficient compared to FBD based, that

Table 6.2: Performance comparison of different multi-operand converter designs using Dadda and proposed cells for different column size

| | | Dadda [51] | | | FBD | | | Hybrid | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Column Size (C) | No. of bits (nob) | No. of ND Cells [50] | Area | Delay | No. of FBD Cells | Area | Delay | No. of ND Cells [50] | No. of FBD Cells | Area | Delay |
| 31 | 9 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 29 | 9 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 27 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 25 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 23 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 21 | 8 | 7 | 105 | 28 | 4 | 184 | 24 | 3 | 2 | 137 | 24 |
| 19 | 8 | 6 | 90 | 24 | 3 | 138 | 18 | 2 | 2 | 122 | 20 |
| 17 | 8 | 6 | 90 | 24 | 3 | 138 | 18 | 2 | 2 | 122 | 20 |
| 15 | 8 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 13 | 7 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 11 | 7 | 5 | 75 | 20 | 3 | 138 | 18 | 1 | 2 | 107 | 16 |
| 9 | 7 | 4 | 60 | 16 | 2 | 92 | 12 | 0 | 2 | 92 | 12 |
| 7 | 6 | 3 | 45 | 12 | 2 | 92 | 12 | 1 | 1 | 61 | 10 |
| 5 | 6 | 3 | 45 | 12 | 2 | 92 | 12 | 1 | 1 | 61 | 10 |
| 3 | 5 | 2 | 30 | 8 | 1 | 46 | 6 | 2 | 0 | 30 | 8 |

too without any compromise on latency.

The total area and delay statistics of hybrid and Dadda MBD schemes for a 16*16 digit by digit multiplier are shown in Table 6.3. The total area of MBD converter is decided by the number of partial product columns while the critical path is decided by the largest column size. It can be observed from Table 6.3 that hybrid MBD design is 17% faster compared to Dadda however with 5% overhead in area.

Table 6.3: Performance comparison of Dadda and Hybrid multi-operand converter in 16*16 multiplier

| Design | Area | % change | Delay | % change |
|---|---|---|---|---|
| **Proposed Hybrid MBD Converter** | **14138** | **100%** | **24** | **100%** |
| Dadda MBD Converter [51] | 13490 | 95% | 28 | 117% |

## 6.4.2 Synthesis based Comparison

To obtain more exact comparisons, Verilog-HDL models of various multiplier designs are synthesized across different technology nodes. For a fair comparison, all the existing and proposed multiplier schemes based on ‘digit-by-digit’ algorithm are extended to perform 16-digit parallel decimal multiplication, modeled using Verilog data flow modeling and simulated using cadence incisive unified simulator (IUS) v6.1. The performance evaluation of each design has been carried out at 180nm,130nm, 90nm, 65nm and 28nm process technology nodes using TSMC library. Low power standard libraries of 13 track for 180nm/130nm, 11 track for 90nm/65nm and 9 track for 28nm provided by foundry are adopted. Synopsys 'Prime time' tool is used for delay calculation for all the BDM topologies on gate level net list with back annotated RC values.

In this section, initially the synthesized area and delay statistics of various multiplier schemes at partial product generation (PPG) stage are presented. Next, the area-delay figures of hybrid converters and partial product reduction (PPR) stage are detailed. Towards the end, a comprehensive performance analysis of $16 * 16$ 'digit-by-digit' multiplier based approaches is presented and discussed.

#### 6.4.2.1 Partial Product Generation (PPG)

An area comparison of six BCD digit multipliers (BDMs) in 'digit-by-digit' scheme, including that of two new schemes (depicted using solid lines), is provided in Fig.6.9. The BDM cells considered are HPBDM, LABDM, N2BDM, split_4_3 BDM, Bhatt BDM and Sree BDM. It can be observed from Fig.6.9 that the proposed LABDM designs performs well at all technology nodes compared to the existing schemes and also conform to the gate level analysis carried out in Section 6.4.1. The area consumed by LABDM is least in comparison to designs considered across various technology nodes, which is due to the LABD converter cells designed as a part of this work except N2BDM reported in [65].

From the graphs shown in Fig.6.10, it can be observed that the proposed HPBDM is the fastest among all the BDM cells compared. The HPBDM achieves a 15% reduction in delay across various technology nodes compared to other implementations. Further, it can be noted that proposed LABDM, though optimized for area, performs better compared to all existing BDM cells.

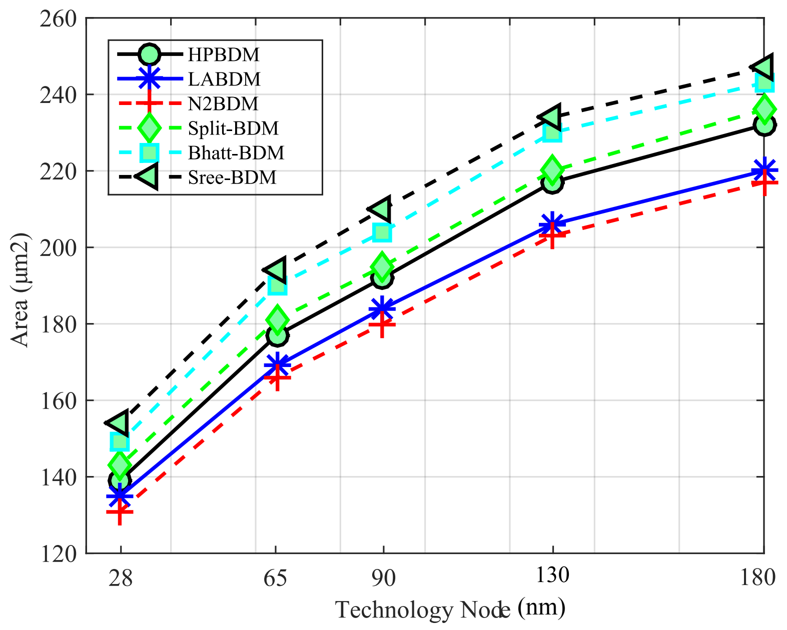


Figure 6.9: A Comparison of area consumption by various BDM at different technology nodes

#### 6.4.2.2 Partial Product Reduction (PPR)

The area and delay statistics across various technology nodes in case of Dadda [51] and proposed hybrid MBD converters are shown in Table 6.4. It can be noted that hybrid MBD converters perform better in terms of delay compared to Dadda MBD converters with only a marginal increase in area.

Table 6.4: A Comparison of Dadda and proposed hybrid MBD converters for area and delay performance across various technology nodes

| Technology (nm) | Dadda MBD Converter | | | | Hybrid MBD Converter | | | |
|---|---|---|---|---|---|---|---|---|
| | Area ($\mu m^2$) | Percent | Delay (ps) | Percent | Area ($\mu m^2$) | Percent | Delay (ps) | Percent |
| 180 | 1268 | 98% | 175 | 113% | 1295 | 100% | 155 | 100% |
| 130 | 1214 | 97.70% | 161 | 116% | 1242 | 100% | 138 | 100% |
| 90 | 1092 | 93% | 143 | 116% | 1174 | 100% | 123 | 100% |
| 65 | 1024 | 92.70% | 127 | 117% | 1104 | 100% | 108 | 100% |
| 28 | 804 | 92% | 98 | 119% | 871 | 100% | 82 | 100% |

As pointed out earlier, PPR stage in Dadda and hybrid PPR schemes consists of binary CSA [51], MBD converters and decimal adders [64]. While, Dadda PPR scheme uses the MBD converters based on Nicoud cells, hybrid MBD converters are used in the proposed PPR scheme. Figures 6.11 and 6.12 provide a comparison of the area and latencies of these two approaches respectively at various technology nodes. It can be seen from these figures that the

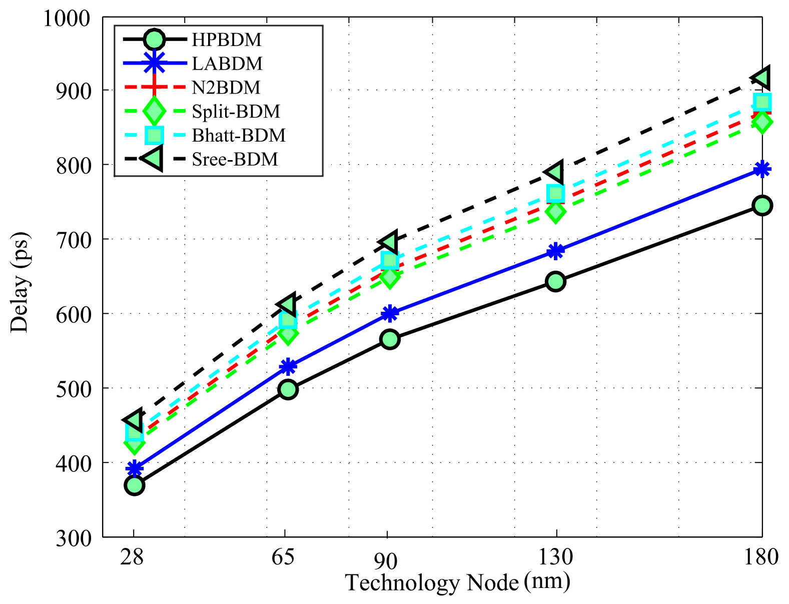


Figure 6.10: A Comparison of delay in various BDM at different technology nodes

proposed scheme results in a 11% improvement in delay with only a small increase in area. Similar pattern of behavior can be observed across all technology nodes.

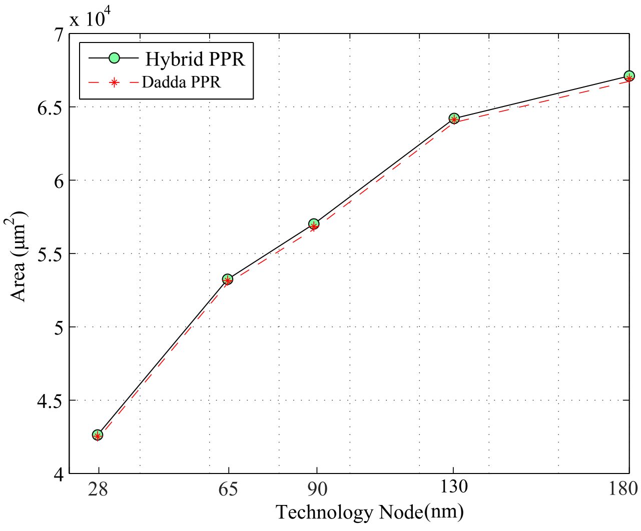


Figure 6.11: A Comparison of area consumption at PPR level in 16*16 ‘digit-by-digit’ multiplier at different technology nodes

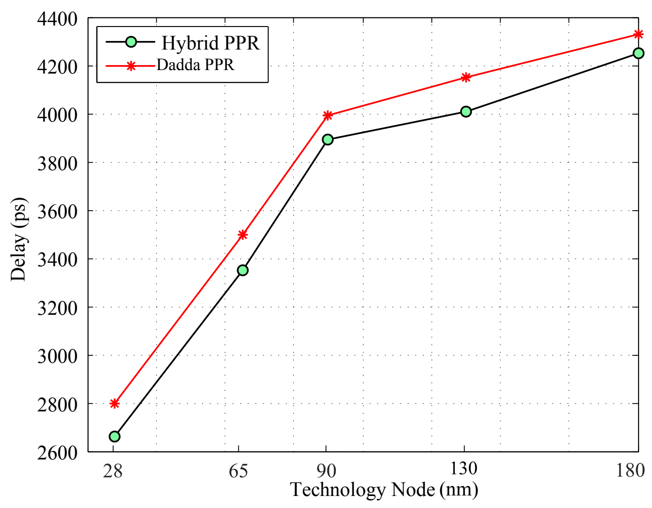


Figure 6.12: A Comparison of delay at PPR level in 16*16 ‘digit-by-digit’ multiplier at different technology nodes

#### 6.4.2.3 Synthesis Results of 16*16 ‘Digit-by-Digit’ Multiplier

The area and delay of different multiplier schemes based on ‘digit-by-digit’multiplication algorithm have been investigated at various technology nodes 180nm,130nm, 90nm, 65nm and 28nm using TSMC library and are presented in Table 6.5. The PPG is carried out using HPBDM, LABDM, N2BDM, split_4_3 BDM, Bhatt BDM and Sree BDM schemes. In PPR stage, while HPBDM and LABDM schemes use the proposed hybrid PPR technique, rest of the BDM schemes use Dadda PPR [51] approach. The proposed schemes have been found to be functional across all the technology nodes. A detailed comparison of these multipliers in terms of area, delay and area-delay product is provided in Table 6.5 and Figs.6.13-6.15. It is evident that the HPBDM with Hybrid PPR performs better in terms of delay (10 to 29%) and area-delay product (4 to 38%) while LABDM with Hybrid PPR achieves an improvement of 9 to 28% in delay and 5 to 39% in area-delay product across various technology nodes. The improvements in delay and area-delay product are bound to increase further in multipliers involving larger operand width. This is due to the efficient PPBD converters and multi-operand converter realized using hybrid converter algorithm.

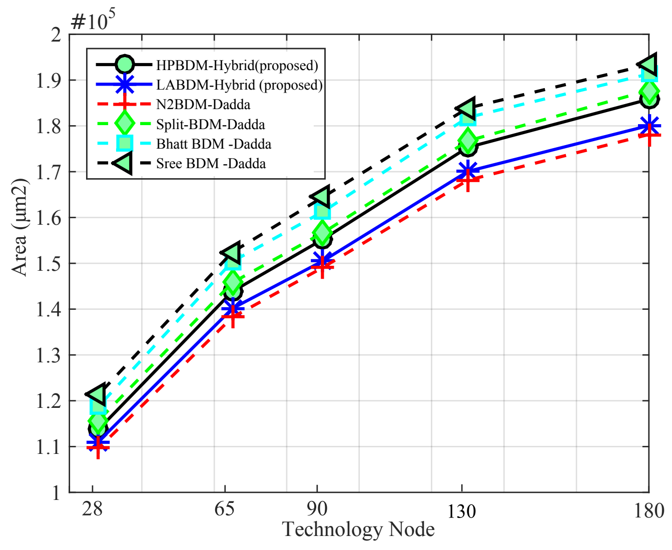


Figure 6.13: A Comparison of area consumption by various multiplier at different technology nodes

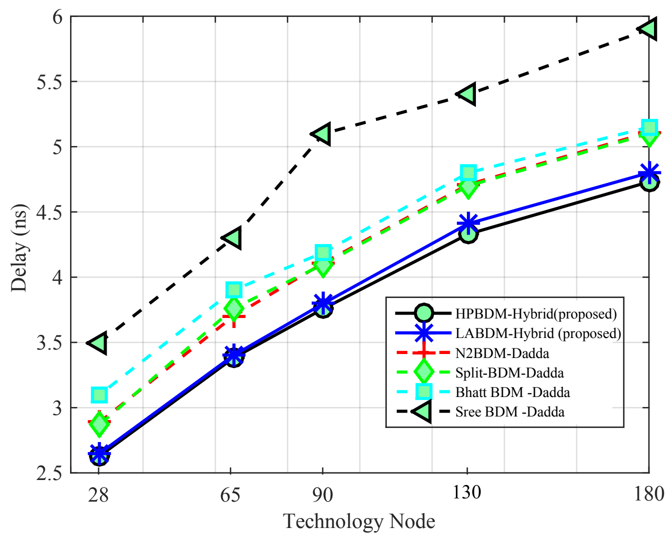


Figure 6.14: A Comparison of delay of various multiplier at different technology nodes

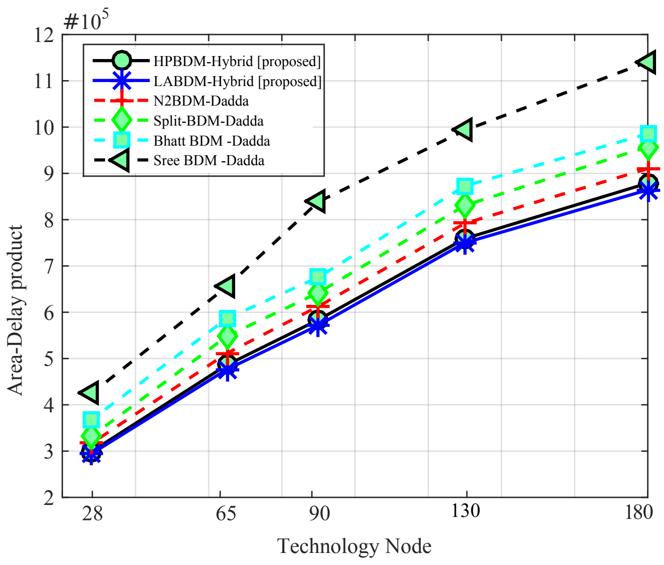


Figure 6.15: A Comparison of area-delay product of various multiplier at different technology nodes

## 6.5 Conclusions

Digital multipliers (binary/decimal) have partial product generation and reduction stages and any performance improvement in these stages will contribute to overall performance of the multiplier. In this work, new schemes have been proposed to improve the partial product generation and reduction stages in decimal multipliers. Decimal partial products are generated in parallel using fast and area efficient BCD digit multipliers and their reduction is achieved using hybrid multi-operand binary to decimal converters. Also, a new hybrid algorithm to design a multi-operand binary to decimal converter based on area and delay requirements has been proposed. In contrast to most of the previous implementations, which propose changes either in partial product generation or reduction, this work proposes modifications at both partial product generation and reduction stages resulting in an improved performance. Results obtained for a 16*16 'digit-by-digit' multiplier clearly show that a performance improvement, in terms of delay of upto 8 to 24%, and an area-delay product of upto 4 to 32%.

Table 6.5: Area and Delay comparison in 16*16 ‘Digit-by-Digit’ multipliers

| Technology Node (nm) | Scheme | Area ($\mu m^2$) | % | Delay(ns) | % | Product (µm2- ns) | % |
|---|---|---|---|---|---|---|---|
| 180 | **HPBDM-Hybrid** | 185806 | 103% | 4.73 | 98% | 878862 | 101% |
| | **LABDM-Hybrid** | 180065 | 100% | 4.8 | 100% | 864312 | 100% |
| | N2 BDM - Dadda | 178126 | 99% | 5.11 | 106% | 910224 | 105% |
| | Split_4_3BDM - Dadda | 187745 | 104% | 5.1 | 106% | 957500 | 110% |
| | Bhatt BDM - Dadda | 191329 | 106% | 5.15 | 107% | 985344 | 114% |
| | Sree BDM - Dadda | 193377 | 107% | 5.9 | 122% | 1140924 | 132% |
| 130 | **HPBDM-Hybrid** | 175306 | 103% | 4.33 | 98% | 759075 | 101% |
| | **LABDM-Hybrid** | 170066 | 100% | 4.41 | 100% | 749991 | 100% |
| | N2 BDM - Dadda | 168138 | 99% | 4.7 | 106% | 790249 | 105% |
| | Split_4_3BDM - Dadda | 176722 | 104% | 4.73 | 107% | 835895 | 111% |
| | Bhatt BDM - Dadda | 181842 | 107% | 4.8 | 108% | 872842 | 116% |
| | Sree BDM - Dadda | 183890 | 108% | 5.4 | 122% | 993006 | 132% |
| 90 | **HPBDM-Hybrid** | 155240 | 103% | 3.76 | 98% | 583702 | 102% |
| | **LABDM-Hybrid** | 150515 | 100% | 3.8 | 100% | 571957 | 100% |
| | N2 BDM - Dadda | 149096 | 99% | 4.11 | 108% | 612785 | 107% |
| | Split_4_3BDM - Dadda | 156659 | 104% | 4.1 | 108% | 642302 | 112% |
| | Bhatt BDM - Dadda | 161267 | 107% | 4.19 | 110% | 675709 | 118% |
| | Sree BDM - Dadda | 166456 | 110% | 4.7 | 123% | 782343 | 136% |
| 65 | **HPBDM-Hybrid** | 143151 | 102% | 3.38 | 99% | 483850 | 102% |
| | **LABDM-Hybrid** | 140157 | 100% | 3.4 | 100% | 476534 | 100% |
| | N2 BDM - Dadda | 136219 | 97% | 3.7 | 109% | 504010 | 106% |
| | Split_4_3BDM - Dadda | 145789 | 104% | 3.76 | 110% | 548167 | 115% |
| | Bhatt BDM - Dadda | 150397 | 107% | 3.9 | 114% | 586548 | 123% |
| | Sree BDM - Dadda | 152445 | 109% | 4.3 | 126% | 655514 | 137% |
| 28 | **HPBDM-Hybrid** | 113718 | 102% | 2.63 | 99% | 299078 | 101% |
| | **LABDM-Hybrid** | 111067 | 100% | 2.65 | 100% | 294328 | 100% |
| | N2 BDM - Dadda | 107922 | 97% | 2.89 | 109% | 311895 | 105% |
| | Split_4_3BDM - Dadda | 115675 | 104% | 2.87 | 108% | 331987 | 112% |
| | Bhatt BDM - Dadda | 118747 | 106% | 3.1 | 117% | 368116 | 125% |
| | Sree BDM - Dadda | 121307 | 109% | 3.4 | 128% | 412444 | 139% |

# Chapter 7

# Conclusions and Future Work

This thesis focused on designing new multiplier architectures in binary, logarithmic and BCD number systems customized for different requirements (accuracy, speed, area and power) to meet the diverse needs of practical applications. The architectures included that of a two-dimensional binary bypass multiplier, a fixed-width binary multiplier, an iterative logarithmic multiplier, and a 'digit-by-digit' BCD multiplier. Various schemes developed to improve these architectures have been detailed, discussed and superiority of their performance has been demonstrated.

The first contribution of this thesis is the development of a reconfigurable two-dimensional bypass multiplier based on dynamic bypassing of partial products. The bypass elements incorporated into the multiplier reduce the power consumption by eliminating redundant signal transitions. Further, the reconfigurable multiplier offers a good trade-off between area, delay and power dissipation by using the same multiplier for performing one $N$ or two $N/2$ multiplications.

The second contribution of this thesis is the development of a novel fixed-width binary multiplier with a target to deploy in error resilient applications where the focus is less on accuracy and more in terms of improved hardware performance. Such units result in area savings while also resulting in reduced power consumption. Further, to quantify the benefits achieved, the performance of this multiplier has been validated using the image sharpening algorithm applied on image processing benchmarks such as Lena, Cameraman, and Pirate.

The third contribution of this thesis is the development of a novel binary logarithmic multiplier with improved precision. The hardware implementation of this multiplier shows that it also has an improved performance in terms of parameters area and delay compared to other designs existing in the literature.

The fourth contribution of this thesis is the development of a generalized design approach and architectural framework for decimal multiplication. Decimal partial products have been generated in parallel using fast and area efficient BCD digit multipliers and their reduction is achieved using new hybrid multi-operand binary to decimal converters. The resulting multiplier has been shown to perform better than the existing BCD multipliers in terms of delay and area-delay product.

## 7.1 Future Work

The binary multiplier units designed and implemented in this work targeted at 180 nm technology and hence dynamic power dominates the overall power consumption. As process technologies shrink, leakage power dominates the overall power dissipation. Hence, it would be interesting to understand the performance of the binary designs proposed in this work at lower technology nodes.

Approximate computing offers potential benefits in terms of area, power and performance. However, its impact on applications is difficult to measure. Researchers and practitioners alike need tools to automate the process of carrying out approximate computing. Hence, developing new approaches that are capable of generating and synthesizing circuits with reasonable error tolerance and significantly less area consumption and power dissipation is an immediate need.

# Bibliography

[1] B. Parhami, *Computer arithmetic*, vol. 20. Oxford university press, 1999.

[2] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[3] E. L. Braun, *Digital computer design: logic, circuitry, and synthesis*. Academic Press, 2014.

[4] P. Behrooz, "Computer arithmetic: Algorithms and hardware designs," *Oxford University Press*, vol. 19, pp. 512583–512585, 2000.

[5] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.

[6] R. Zimmermann, *Binary adder architectures for cell-based VLSI and their synthesis*. Citeseer, 1998.

[7] W. N. HE *et al.*, *Cmos Vlsi Design: A Circuits And Systems Perspective, 3/E*. Pearson Education India, 2006.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE transactions on computers*, vol. 100, no. 8, pp. 786–793, 1973.

[9] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 831–838, 1980.

[10] J.-n. Ohban, V. G. Moshnyaga, and K. Inoue, "Multiplier energy reduction through bypassing of partial products," in *Circuits and Systems, 2002. APCCAS'02. 2002 Asia-Pacific Conference on*, vol. 2, pp. 13–17, IEEE, 2002.

[11] M.-C. Wen, S.-J. Wang, and Y.-N. Lin, "Low power parallel multiplier with column bypassing," in *2005 IEEE International Symposium on Circuits and Systems*, pp. 1638–1641, IEEE, 2005.

[12] G.-N. Sung, Y.-J. Ciou, and C.-C. Wang, "A power-aware 2-dimensional bypassing multiplier using cell-based design flow," in *2008 IEEE International Symposium on Circuits and Systems*, pp. 3338–3341, IEEE, 2008.

[13] M. Själander, M. Draždžiulis, P. Larsson-Edefors, and H. Eriksson, "A low-leakage twin-precision multiplier using reconfigurable power gating," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 1654–1657, IEEE, 2005.

[14] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin-precision multiplier," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 30–33, IEEE, 2004.

[15] S. Hong, T. Roh, and H.-J. Yoo, "A 145$\mu$w 8$\times$ 8 parallel multiplier based on optimized bypassing architecture," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 1175–1178, IEEE, 2011.

[16] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, p. 595, 1963.

[17] M. J. Schulte, J. E. Stine, and J. G. Jansen, "Reduced power dissipation through truncated multiplication," in *Low-Power Design, 1999. Proceedings. IEEE Alessandro Volta Memorial Workshop on*, pp. 61–69, IEEE, 1999.

[18] K. Biswas, P. Mokrian, H. Wu, and M. Ahmadi, "Truncation schemes for recursive multipliers," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, pp. 1177–1180, IEEE, 2005.

[19] A. N. Danysh and E. E. Swartzlander Jr, “A recursive fast multiplier,” in *Signals, Systems &amp; Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference on*, vol. 1, pp. 197–201, IEEE, 1998.

[20] K. Biswas, H. Wu, and M. Ahmadi, “Fixed-width multi-level recursive multipliers,” in *Signals, Systems and Computers, 2006. ACSSC’06. Fortieth Asilomar Conference on*, pp. 935–938, IEEE, 2006.

[21] E. E. Swartzlander Jr, “Truncated multiplication with approximate rounding,” in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, vol. 2, pp. 1480–1483, IEEE, 1999.

[22] E. J. King and E. E. Swartzlander Jr, “Data-dependent truncation scheme for parallel multipliers,” in *Signals, Systems &amp; Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 2, pp. 1178–1182, IEEE, 1997.

[23] Y. Lim, “Single-precision multiplier with reduced circuit complexity for signal processing applications,” *Computers, IEEE Transactions on*, vol. 41, no. 10, pp. 1333–1336, 1992.

[24] S. S. Kidambi, F. El-Guibaly, and A. Antoniou, “Area-efficient multipliers for digital signal processing applications,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 2, pp. 90–95, 1996.

[25] J. M. Jou, S. R. Kuang, and R. Der Chen, “Design of low-error fixed-width multipliers for dsp applications,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 6, pp. 836–842, 1999.

[26] Z. Babić, A. Avramović, and P. Bulić, “An iterative logarithmic multiplier,” *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[27] M. Sullivan and E. Swartzlander, “Truncated error correction for flexible approximate multiplication,” in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, pp. 355–359, Nov 2012.

[28] K. Abed and R. Siferd, “Cmos vlsi implementation of a low-power logarithmic converter,” *Computers, IEEE Transactions on*, vol. 52, pp. 1421–1433, Nov 2003.

[29] K. Abed and R. Siferd, “Vlsi implementation of a low-power antilogarithmic converter,” *Computers, IEEE Transactions on*, vol. 52, pp. 1221–1228, Sept 2003.

[30] M. Combet, H. Van Zonneveld, and L. Verbeek, “Computation of the base two logarithm of binary numbers,” *Electronic Computers, IEEE Transactions on*, vol. EC-14, pp. 863–867, Dec 1965.

[31] E. L. Hall, D. Lynch, and I. Dwyer, S.J., “Generation of products and quotients using approximate binary logarithms for digital filtering applications,” *Computers, IEEE Transactions on*, vol. C-19, pp. 97–105, Feb 1970.

[32] S. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, “A fast, low-power logarithm approximation with cmos vlsi implementation,” in *Circuits and Systems, 1999. 42nd Midwest Symposium on*, vol. 1, pp. 388–391 vol. 1, 1999.

[33] V. Mahalingam and N. Ranganathan, “Improving accuracy in mitchell’s logarithmic multiplication using operand decomposition,” *Computers, IEEE Transactions on*, vol. 55, pp. 1523–1535, Dec 2006.

[34] T. Brubaker and J. Becker, “Multiplication using logarithms implemented with read-only memory,” *Computers, IEEE Transactions on*, vol. C-24, pp. 761–765, Aug 1975.

[35] D. Mclaren, “Improved mitchell-based logarithmic multiplier for low-power dsp applications,” in *SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip]*, pp. 53–56, Sept 2003.

[36] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *Electronic Computers, IRE Transactions on*, vol. EC-11, pp. 512–517, Aug 1962.

[37] M. Cowlishaw, “Decimal floating-point: algorism for computers,” in *Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on*, pp. 104–111, June 2003.

[38] S. Shankland, “IbmŠs power6 gets help with math, multimedia.,” 2006.

[39] C. Webb, “Ibm z10: The next-generation mainframe microprocessor,” *Micro, IEEE*, vol. 28, pp. 19–29, March 2008.

[40] L. Dadda and A. Nannarelli, “A variant of a radix-10 combinational multiplier,” in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 3370–3373, May 2008.

[41] L. Han and S.-B. Ko, “High-speed parallel decimal multiplication with redundant internal encodings,” *Computers, IEEE Transactions on*, vol. 62, pp. 956–968, May 2013.

[42] G. Jaberipur and A. Kaivani, “Improving the speed of parallel decimal multiplication,” *Computers, IEEE Transactions on*, vol. 58, pp. 1539–1552, Nov 2009.

[43] A. Vazquez, E. Antelo, and P. Montuschi, “Improved design of high-performance parallel decimal multipliers,” *Computers, IEEE Transactions on*, vol. 59, pp. 679–693, May 2010.

[44] G. Jaberipur and A. Kaivani, “Binary-coded decimal digit multipliers,” *Computers Digital Techniques, IET*, vol. 1, pp. 377–381, July 2007.

[45] R. James, T. Shahana, K. Jacob, and S. Sasi, “Decimal multiplication using compact bcd multiplier,” in *Electronic Design, 2008. ICED 2008. International Conference on*, pp. 1–6, Dec 2008.

[46] S. Gorgin, G. Jaberipur, and B. Parhami, “Design and evaluation of decimal array multipliers,” in *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pp. 1782–1786, Nov 2009.

[47] J. Bhattacharya, A. Gupta, and A. Singh, “A high performance binary to bcd converter for decimal multiplication,” in *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pp. 315–318, April 2010.

[48] O. Al-Khaleel, Z. Al-QudahJ, M. Al-Khaleel, C. A. Papachristou, and F. Wolff, “Fast and compact binary-to-bcd conversion circuits for decimal multiplication,” in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pp. 226–231, Oct 2011.

[49] S. Veeramachaneni and M. Srinivas, "Novel high-speed architecture for 32-bit binary coded decimal (bcd) multiplier," in *Communications and Information Technologies, 2008. ISCIT 2008. International Symposium on*, pp. 543–546, Oct 2008.

[50] J.-D. Nicoud, "Iterative arrays ror radix conversion," *Computers, IEEE Transactions on*, vol. C-20, pp. 1479–1489, Dec 1971.

[51] L. Dadda, "Multioperand parallel decimal adder: A mixed binary and bcd approach," *Computers, IEEE Transactions on*, vol. 56, pp. 1320–1328, Oct 2007.

[52] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, 1965.

[53] M. Sjalander and P. Larsson-Edefors, "Multiplication acceleration through twin precision," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 9, pp. 1233–1246, 2009.

[54] C.-C. Wang and G.-N. Sung, "Low-power multiplier design using a bypassing technique," *Journal of Signal Processing Systems*, vol. 57, no. 3, pp. 331–338, 2009.

[55] J.-M. Wang, S.-C. Fang, and W.-S. Feng, "New efficient designs for xor and xnor functions on the transistor level," *IEEE Journal of solid-state Circuits*, vol. 29, no. 7, pp. 780–786, 1994.

[56] K. V. Palem, "Energy aware computing through probabilistic switching: A study of limits," *IEEE Trans. Computers*, vol. 54, no. 9, pp. 1123–1137, 2005.

[57] M. Hasan, T. Arslan, and J. S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 fft processor for wireless lan applications," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 1, pp. 128–134, 2003.

[58] J.-H. Tu and L.-D. Van, "Power-efficient pipelined reconfigurable fixed-width baugh-wooley multipliers," *IEEE transactions on computers*, vol. 58, no. 10, pp. 1346–1355, 2009.

[59] M. Sullivan and E. Swartzlander, “Truncated logarithmic approximation,” in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, pp. 191–198, April 2013.

[60] M. S. Lau, K.-V. Ling, and Y.-C. Chu, “Energy-aware probabilistic multiplier: Design and analysis,” in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES ’09, (New York, NY, USA), pp. 281–290, ACM, 2009.

[61] V. Paliouras and T. Stouraitis, “Low-power properties of the logarithmic number system,” in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, pp. 229–236, 2001.

[62] M. Erle and M. Schulte, “Decimal multiplication via carry-save addition,” in *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, pp. 348–358, June 2003.

[63] M. Erle, E. Schwarz, and M. Schulte, “Decimal multiplication with efficient partial product generation,” in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pp. 21–28, June 2005.

[64] M. S. Schmookler and A. Weinberger, “High speed decimal addition,” *IEEE Transactions on Computers*, vol. 20, no. 8, pp. 862–866, 1971.

[65] S. Gorgin, G. Jaberipur, and R. Hashemi Asl, “Efficient asic and fpga implementation of binary-coded decimal digit multipliers,” *Circuits, Systems, and Signal Processing*, vol. 33, no. 12, pp. 3883–3899, 2014.

# High Performance Binary, Logarithmic, and BCD Multiplier Architectures

**THESIS**

Submitted in partial fulfillment

of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

by

**Syed Ershad Ahmed**

**ID No. 2009PHXF448H**

Under the Supervision of

**Prof. M. B. Srinivas**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI**

**December, 2017**

# Chapter 7

# Conclusions and Future Work

This thesis focused on designing new multiplier architectures in binary, logarithmic and BCD number systems customized for different requirements (accuracy, speed, area and power) to meet the diverse needs of practical applications. The architectures included that of a two-dimensional binary bypass multiplier, a fixed-width binary multiplier, an iterative logarithmic multiplier, and a 'digit-by-digit' BCD multiplier. Various schemes developed to improve these architectures have been detailed, discussed and superiority of their performance has been demonstrated.

The first contribution of this thesis is the development of a reconfigurable two-dimensional bypass multiplier based on dynamic bypassing of partial products. The bypass elements incorporated into the multiplier reduce the power consumption by eliminating redundant signal transitions. Further, the reconfigurable multiplier offers a good trade-off between area, delay and power dissipation by using the same multiplier for performing one $N$ or two $N/2$ multiplications.

The second contribution of this thesis is the development of a novel fixed-width binary multiplier with a target to deploy in error resilient applications where the focus is less on accuracy and more in terms of improved hardware performance. Such units result in area savings while also resulting in reduced power consumption. Further, to quantify the benefits achieved, the performance of this multiplier has been validated using the image sharpening algorithm applied on image processing benchmarks such as Lena, Cameraman, and Pirate.

The third contribution of this thesis is the development of a novel binary logarithmic multiplier with improved precision. The hardware implementation of this multiplier shows that it also has an improved performance in terms of parameters area and delay compared to other designs existing in the literature.

The fourth contribution of this thesis is the development of a generalized design approach and architectural framework for decimal multiplication. Decimal partial products have been generated in parallel using fast and area efficient BCD digit multipliers and their reduction is achieved using new hybrid multi-operand binary to decimal converters. The resulting multiplier has been shown to perform better than the existing BCD multipliers in terms of delay and area-delay product.

## 7.1 Future Work

The binary multiplier units designed and implemented in this work targeted at 180 nm technology and hence dynamic power dominates the overall power consumption. As process technologies shrink, leakage power dominates the overall power dissipation. Hence, it would be interesting to understand the performance of the binary designs proposed in this work at lower technology nodes.

Approximate computing offers potential benefits in terms of area, power and performance. However, its impact on applications is difficult to measure. Researchers and practitioners alike need tools to automate the process of carrying out approximate computing. Hence, developing new approaches that are capable of generating and synthesizing circuits with reasonable error tolerance and significantly less area consumption and power dissipation is an immediate need.