# INVESTIGATION INTO ENERGY-EFFICIENT AND APPROXIMATE MULTIPLIER DESIGN

Issa Hani Qiqieh

A Thesis Submitted for the Degree of

Doctor of Philosophy at Newcastle University

School of Engineering

September 2018

# DECLARATION

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

*Newcastle upon Tyne, September 2018*

Issa Hani Qiqieh

## CERTIFICATE OF APPROVAL

---

We confirm that, to the best of our knowledge, this thesis is from the student's own work and effort, and all other sources of information used have been acknowledged. This thesis has been submitted with my approval.

---

Prof. Alex Yakovlev

Dr. Rishad Shafik

Dr. Danil Sokolov

*To the soul of my father who had given me dreams to look forward to*

*To my wonderful mother*

*To my lovely Yara, Hani, Morad and Bassel*

— Issa

# ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisors Prof. Alex Yakovlev, Dr. Rishad Shafik and Dr. Danil Sokolov for their wisdom and guidance through my PhD journey. They have always been a source of motivation and my inspirational model as a researcher.

I am grateful to the Ministry of Higher Education and Scientific Research in Jordan and Al-Balqa' Applied University for funding my PhD study and for their support.

I would like also to express my gratefulness and appreciation to my colleagues and friends in the School of Engineering, especially those in MicroSystems Research Group. We have worked together and discussed many topics over the years, and from them all, I have learned many things. I hope they continue to be successful with their research and future careers. Also, I would like to offer my special regards to all the staff of the School of Engineering at Newcastle University.

Thank you, my mother, my mother-in-law and my sisters, without your unconditional support, this thesis never would have come to be. Finally, and most importantly, I am thankful to my lovely family, Yara my wonderful wife and my lovely sons Hani, Morad and Bassel for all of their love, support, motivation and patience throughout my PhD. They were there to help me at difficult times, and to share in good times. Their encouragement has helped me with the research and the writing of this thesis, and I am very grateful.

## ABSTRACT

There is a persistent demand for higher computational performance at low energy cost for emerging compute-intensive applications. Multipliers constitute a major component of these applications with complex logic design and a large gate count compared to other arithmetic units. As such, there is significant interest in designing new approaches to low-complexity multipliers. Approximate multiplier is a promising paradigm, which is particularly suitable for inherently imprecision-tolerant applications, such as image processing, pattern recognition and machine learning. The basic premise is to relax the precision requirements in favour of lower complexity, thereby achieving reduced circuit delay and energy consumption.

This thesis presents an investigation into novel approximate multiplier design and implementation approaches. In the first approach, a multiplier design using significance-driven logic compression (SDLC) is proposed. Fundamental to this approach is a configurable lossy compression of the partial product rows. The compression is carried out by progressively replacing the exclusive-OR logic gates by low-complexity OR gates based on their bit significance. The compression is followed by commutative remapping of the resulting product terms to reduce the number of product rows. This accounts for substantially reduced number of logic cell counts and lengths of critical paths at the cost of errors in lower significant bits.

In the second approach, a novel multiplier design is presented by combining a Wallace-tree accumulation method together with the SDLC. The logic compression performed by SDLC approach works for reducing the number of product rows using progressive bit significance, thereby decreasing the number of reduction stages and logic counts in accumulation. The errors introduced by lossy logic compression are minimised through a novel error compensation method (ECM). The core of this method is a parallel error detec-

tion logic used to generate error compensation bit-matrix. This matrix is then compressed using OR gates to generate an error compensation vector. To mitigate the impact of error, this vector is either considered as an additional row in the accumulation tree or used to modify an existing row.

To validate the effectiveness of these approaches, a number of multipliers with different compression levels are designed and synthesized showing substantial savings in energy consumption, and reductions in critical delay and silicon area, compared to an accurate equivalent and other existing approaches. These gains are achieved at the cost of errors introduced in the circuit, which are extensively analysed.

The configurable multiplier designs in the first and second approaches exhibit energy/quality trade-offs at different degrees of compression. These trade-offs can be effectively used to implement multipliers in applications, where energy can be opportunistically minimised within the envelope of quality requirements. As such, in the third study, two implementation methods are demonstrated. In the first method, a Gaussian blur filter was designed, demonstrating energy reduction with a minor loss in image quality. In the second method, the energy/quality trade-offs are leveraged in a perceptron-based machine learning application, showing energy reduction for different SDLC configurations.

The proposed logic compression approach and its prototype implementations in various configurations can be suitably used for energy-efficient multiplier designs, where quality requirements can be relaxed.

# PUBLICATIONS AND CONTRIBUTIONS

The publications that were produced as a part of research reported in this thesis are listed as follows:

*Journal publications:*

- **Issa Qiqieh**; Rishad Shafik; Ghaith Tarawneh; Danil Sokolov; Shidhartha Das; Alex Yakovlev, *Significance-Driven Logic Compression for Energy-Efficient Multiplier Design*, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), September 2018, vol. 8, no. 3, pp. 417-430.

*Conference publications:*

- **Issa Qiqieh**; Rishad Shafik; Danil Sokolov; Alex Yakovlev, *Energy-Efficient Approximate Multiplier Design using Bit Significance-Driven Logic Compression*, Design, Automation Test in Europe Conference Exhibition (DATE), March 2017, pp 7-12.
- **Issa Qiqieh**; Rishad Shafik; Ghaith Tarawneh; Danil Sokolov; Shidhartha Das; Alex Yakovlev, *Energy-Efficient Approximate Wallace-Tree Multiplier using Significance-Driven Logic Compression*, IEEE International Workshop on Signal Processing Systems (SiPS), October 2017, pp 1-6.
- Dave Burke; Dainius Jenkus; **Issa Qiqieh**; Rishad Shafik; Shidhartha Das; Alex Yakovlev, *Special Session Paper: Significance-Driven Adaptive Approximate Computing for Energy-Efficient Image Processing Applications*, CODES+ISSS, October 2017, pp 1-2.

CONTENTS


I Thesis Chapters 1

1 INTRODUCTION 2

1.1 Motivation . . . . . 2

1.1.1 Energy-Efficient Computing . . . . . 3

1.1.2 Approximate Multiplier Design . . . . . 4

1.2 Thesis Scope and Contributions . . . . . 6

1.3 Thesis Overview . . . . . 8

2 BACKGROUND AND LITERATURE SURVEY 11

2.1 Introduction . . . . . 11

2.2 Approximate Circuit Design . . . . . 12

2.2.1 Imprecise Hardware Design . . . . . 12

2.2.2 Taxonomy of Approximate Circuits . . . . . 20

2.3 Approximate Multiplier Design . . . . . 23

2.3.1 Taxonomy and Survey . . . . . 24

2.3.2 Error Evaluation and Challenges . . . . . 33

2.4 Concluding Remarks and Discussions . . . . . 37

3 LOGIC COMPRESSION IN MULTIPLIER DESIGN 39

3.1 Introduction . . . . . 39

3.2 Significance-Driven Logic Compression Approach . 40

3.2.1 Logic Clustering . . . . . 41

3.2.2 Logic Compression . . . . . 43

3.2.3 Progressive Cluster Sizing . . . . . 44

3.2.4 Commutative Remapping . . . . . 45

3.2.5 Example of Utilizing 2-bit SDLC . . . . . 45
3.3 Variable Logic Cluster and Scalability . . . . . 47
3.3.1 General Space of $d$-bit Logic Cluster . . . . 48
3.3.2 $d$-bit Logic Cluster: Compression Algorithm 50
3.3.3 Scalability for $(N \times N)$ SDLC Multiplier Design 53
3.3.4 Examples of Utilizing $d$-bit SDLC . . . . . 55
3.4 Error Analysis . . . . . 58
3.5 Design Trade-offs . . . . . 67
3.6 Comparative Analysis . . . . . 73
3.7 Signed Multiplication using SDLC . . . . . 76
3.8 Concluding Remarks . . . . . 80

4 ERROR MITIGATION IN LOGIC COMPRESSION 81
4.1 Introduction . . . . . 81
4.2 Proposed Approximate Wallace Multiplier . . . . . 82
4.2.1 Logic Compression using SDLC . . . . . 82
4.2.2 Accumulation with Wallace Method . . . . . 83
4.2.3 Wallace with Variable Logic Compression . 84
4.3 Error Compensation Method (ECM) . . . . . 91
4.3.1 Parallel Error Detection Logic . . . . . 92
4.3.2 Error Compensation Vector . . . . . 94
4.4 Error Analysis . . . . . 96
4.5 Experimental Results and Design Trade-offs . . . . 99
4.6 Concluding Remarks . . . . . 104

5 IMPLEMENTATION AND VALIDATIONS 105
5.1 Introduction . . . . . 105
5.2 Case Study 1: Gaussian Blur Filter . . . . . 106
5.3 Case Study 2: Perceptron Classifier . . . . . 111
5.4 Concluding Remarks . . . . . 115

6 CONCLUSIONS AND FUTURE WORK 117

6.1 Summary and Conclusions . . . . . 117
6.2 Critical Review and Future Work . . . . . 120

II Thesis Bibliography 123

BIBLIOGRAPHY 124

# LIST OF FIGURES


Figure 2.1 Taxonomy of imprecise computation in hardware. . . . . 13

Figure 2.2 Example of a small quantum circuit *3qubitcnot*: (a) fault-free circuit, and (b) faulty circuit by eliminating a single H gate [6]. . . . 14

Figure 2.3 Stochastic encoding: (a) a stochastic bit stream; (b) a stochastic wire bundle. For each bit in the bit stream or a bundle, the probability that it is 1 is $P(X = 1) = x$. . . . . 15

Figure 2.4 Stochastic multiplication using an AND gate. 15

Figure 2.5 Example of a $(2 \times 2)$ multiplier: (a) approximate, and (b) accurate, with the critical paths highlighted [65]. . . . . 17

Figure 2.6 Taxonomy of approximate circuits. . . . . 20

Figure 2.7 Multi-dimensional taxonomy of approximate multiplier designs. . . . . 24

Figure 2.8 An example of $(2 \times 2)$ multiplier: (a) operating within safe voltage range; (b) lowering the supply voltage below its nominal value. . . 25

Figure 2.9 Increasing the level of truncation from; (a) 3; (b) 4; (c) 5; and (d) 6 columns, translates into additional reductions in area and power; however, error is maximized and this method is not effective to reduce the critical column of the accumulation tree (highlighted in dotted rectangles). . . . . 26

Figure 2.10 Using (2 × 2) approximate multiplier blocks to build larger energy-efficient multipliers [65]. 27

Figure 2.11 Different sizes of approximate and accurate multipliers are used to build large recursive multiplier for pipelined architecture [8]; the carry-in logic is used for the approximate partial product computation only ($A_HB_L, A_LB_H, A_LB_L$) and not for the accurate $A_HB_H$. . . . . 28

Figure 2.12 The partial product accumulation tree: (a) of an accurate (8 × 8) multiplier; (b) design parameters are set using multiple EDA tools, and (c) the reduced partial product matrix after applying the partial product perforation with $j$ =2 and $k$ = 3 [134]. . . . . 29

Figure 2.13 The quality constraint circuit proposed by SALSA [125] (Q is a single Boolean value). . 31

Figure 3.1 Process chart showing the difference between the major stages in: (a) conventional multiplication, and (b) the proposed approach to multiplication. . . . . 41

Figure 3.2 Stylized demonstration of SDLC approach [93]: four different sizes of logic clusters used to compress partial products based on their progressive bit-significance in (8 × 8) parallel multiplier architecture. . . . . 42

Figure 3.3 Eight different sizes of 2-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 46

Figure 3.4 Dot diagram showing the impact of increasing the depth of the logic clusters in the case of $(8 \times 8)$ multiplier: (a) clustering a group of bits within 2 successive rows in the partial product bit-matrix after bitwise multiplication; (b) generating a reduced set of product terms after targeting the depth of 2-row logic compression; (c) ordered matrix after applying commutative remapping of the bit sequence resulting from the SDLC approach; (d), (e) and (f) the same process when applying 3-bit logic compression; (g), (h) and (i) the same process when applying 4-bit logic compression. The dotted rectangles at the right indicate the heights of the critical columns which are further reduced compared to the accurate accumulation tree. . . . . 48

Figure 3.5 Dot diagram showing the general space of targeted partial product terms compressed by a $(d \times L)$ logic cluster to produce array of $L$ bits in $r^{th}$ row of the reduced partial product matrix for $(N \times N)$ multiplier using SDLC approach with $d$-bit logic compression. . . . 49

Figure 3.6 Five different sizes of 3-bit logic clusters used to compress partial products based on their progressive bit-significance in $(16 \times 16)$ parallel multiplier architecture. . . . . 55

Figure 3.7 Four different sizes of 4-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 56
Figure 3.8 Three different sizes of 5-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 57
Figure 3.9 Two different sizes of 6-bit combined with 4-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 57
Figure 3.10 Two different sizes of 7-bit combined with 2-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 58
Figure 3.11 Two different sizes of 8-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture. . . . . 58
Figure 3.12 Error percentage distribution for 8-, 12- and 16-bit proposed multiplier after applying 2-bit depth compression. . . . . 62
Figure 3.13 Cumulative probability distribution for the error induced by different logic compression levels of the proposed SDLC approach in the case of (16 × 16) multiplier. . . . . 66

Figure 3.14 Dynamic/leakage power, area, delay and power-delay-product (PDP) trade-offs for different bit-widths of the (2-bit SDLC) proposed multiplier. . . . . 68
Figure 3.15 Dot diagram highlights the impact of increasing depth of logic clusters on the critical path of (8 × 8) multiplier: (a) partial product bit-matrix of the conventional multiplier; (b) after 2-bit SDLC; (c) 3-bit SDLC; and (d) 4-bit SDLC. The dotted polygons indicate the maximum propagation path for summing up the accumulation tree. Higher degrees of compression minimize the propagation delay associated with accumulation tree (such as (b) and (c)), while a further reduction in (d), since a carry propagation adder is just needed to generate the product (no extra delay required for accumulation tree). The curved lines identify the critical paths for each multiplier (from A1 to P14). . . . . 71
Figure 3.16 Dynamic power, leakage power, delay, area and energy trade-offs for different degrees of logic compression of (8 × 8) multiplier. . . . . 72
Figure 3.17 The mean relative error distance (MRED) and PDP trade-offs for different degrees of logic compression of (8 × 8) and (16 × 16) multipliers. 73
Figure 3.18 Area and power trade-offs for various scalable approximate multipliers. . . . . 74

Figure 3.19 Comparative errors in terms of MRED, normalized mean error distance (NMED) and also ER for various scalable (8 × 8) approximate multipliers. . . . . 76
Figure 3.20 Block diagram for one of the options of hardware implementation required to implement the proposed signed multiplier (inspired from [132]). . . . . 77
Figure 3.21 Partial product matrix of (8 × 8) signed multiplier. Complemented partial products are highlighted in blue. . . . . 78
Figure 4.1 Reduction stages and logic cell counts for (16 × 16) accurate Wallace. . . . . 84
Figure 4.2 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 2-bit significance-driven logic compression (SDLC) with Wallace-tree accumulation (2-bit SDLC Wallace). . . . . 85
Figure 4.3 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 3-bit SDLC with Wallace-tree accumulation (3-bit SDLC Wallace). . . . . 86
Figure 4.4 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 4-bit SDLC with Wallace-tree accumulation (4-bit SDLC Wallace). . . . . 86
Figure 4.5 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 5-bit SDLC with Wallace-tree accumulation (5-bit SDLC Wallace). . . . . 87

Figure 4.6 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 6-bit SDLC with Wallace-tree accumulation (6-bit SDLC Wallace). . . . . 87
Figure 4.7 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 7-bit SDLC with Wallace-tree accumulation (7-bit SDLC Wallace). . . . . 87
Figure 4.8 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 8-bit SDLC with Wallace-tree accumulation (8-bit SDLC Wallace). . . . . 87
Figure 4.9 Wallace reduction stages of an (8 × 8) multiplier: (a) accurate Wallace tree; then Wallace method coupled with (b) 2-bit SDLC; (c) 3-bit SDLC and (d) 4-bit SDLC. . . . . 90
Figure 4.10 2-bit OR gate is sufficient to find the sum of two bits. . . . . 92
Figure 4.11 A parallel error-detection logic to generate the error compensation bit-matrix in the case of 2-bit SDLC, and then, array of OR gates to form the error-compensation vector. . . . . 93
Figure 4.12 The error-detection logic circuit parallel with the logic clusters required by error compensation method (ECM) in: (a) 2-bit; (b) 3-bit; (c) 4-bit logic clusters. . . . . 94
Figure 4.13 Improving accuracy by allowing error-compensation vector to modify an existing row in Wallace accumulation tree. . . . . 95

Figure 4.14 Improving accuracy by including error-compensation vector as an additional row in Wallace accumulation tree. . . . . 96
Figure 4.15 Cumulative probability distribution for the error induced by different logic compression levels coupled with the proposed ECM in the case of (8 × 8) proposed multiplier. . . . . 98
Figure 4.16 The impact of the proposed ECM on the (8×8) approximate Wallace multiplier with: (a) 2-bit, (b) 3-bit and (c) 4-bit logic compression levels. . . . . 100
Figure 4.17 The impact of the proposed ECM on the (16 × 16) approximate Wallace multiplier with: (a) 2-bit, (b) 3-bit and (c) 4-bit logic compression levels. . . . . 102
Figure 5.1 Flowchart diagram showing the main steps for evaluating the impact of the proposed multiplier on the final quality of image processed by Gaussian blur filter. . . . . 107
Figure 5.2 Output quality after applying Gaussian blur filtering for different degrees of logic compression of the proposed (8 × 8) multiplier. . 109
Figure 5.3 Output quality after applying Gaussian blur filtering for different degrees of logic compression of of the proposed (16 × 16) multiplier.110
Figure 5.4 Signal-flow graph of the perceptron. . . . . 111
Figure 5.5 Flowchart diagram demonstrating the main steps for evaluating the impact of the proposed multiplier on a perceptron-based Classifier. . . . . 112

Figure 5.6 The test set perceptron classification using; (a) accurate multiplier; (b) 2-bit SDLC proposed multiplier, where the axises show the random inputs between 0 to 65535. (blue and red points represent two classes -1 and +1, black dots for mismatch classification points.) 114

## LIST OF TABLES

Table 2.1 Truth table for the accurate and approximate (2 × 2) multipliers, used to obtain comparative error analysis in Fig. 2.5, with changed entry highlighted. . . . . 18

Table 2.2 Summary of approximate multiplier design approaches. . . . . 32

Table 3.1 Error metrics for varying sizes of proposed multiplier using 2-bit logic cluster. . . . . 62

Table 3.2 Error metrics for different depths of logic compression in the proposed (8 × 8) multiplier. 64

Table 3.3 Error metrics for different depths of logic compression in the proposed (16 × 16) multiplier. . . . . 65

Table 3.4 Design trade-offs for different bit-widths of the accurate multiplier used to obtain comparative analysis in Fig. 3.14. . . . . 69

Table 3.5 Design trade-offs for different bit-widths of the proposed multiplier used to obtain comparative analysis in Fig. 3.14. . . . . 70

Table 3.6 Number of library cells instantiated to form different bit-widths of the (2-bit SDLC) proposed multiplier. . . . . 70
Table 4.1 Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating different levels of logic compression with Wallace-tree accumulation. . . . . 88
Table 4.2 Reduction stages and logic cell counts for (8 × 8) proposed multiplier when incorporating different levels of logic compression with Wallace-tree accumulation. . . . . 90
Table 4.3 ECM drastically reduces the errors across all metrics. . . . . 97
Table 4.4 Design trade-offs for different compression levels of the proposed multiplier used to obtain comparative analysis in Fig. 4.16. . . . 101
Table 4.5 Design trade-offs for different compression levels of the proposed multiplier used to obtain comparative analysis in Fig. 4.17. . . . 103
Table 5.1 Error rate results and energy savings for perceptron classifier . . . . . 115

## LIST OF ALGORITHMS

3.1 Generating the output bits of the logic cluster of the $r^{th}$ row for the proposed ($N \times N$) multiplier using $d$-bit logic clusters. . . . . 51

3.2 Generating a reduced partial product matrix M for $(N \times N)$ multiplier using SDLC approach with $d$-bit logic clusters,$\forall\{d \in \{2,3,...,N\}$. . . . . 54

4.1 $(N \times N)$ Wallace-tree multiplier using SDLC approach with $d$-bit logic clusters. . . . . 91

# ACRONYMS

**CMOS** complementary metal-oxide-semiconductor

**CPA** carry propagating adder

**DSP** digital signal processing

**ECM** error compensation method

**ED** error distance

**EDA** electronic design automation

**EP** error probability

**ER** error rate

**ETM** error-tolerant multiplier

**MBE** modified Booth-encoding

**MED** mean error distance

**MRED** mean relative error distance

**MSE** mean squared error

**NMED** normalized mean error distance

**NMRED** normalized mean relative error distance

**NMSE** normalized mean squared error

**PDP** power-delay-product

**PEQ** performance-energy-quality

**PPM** partial product matrix

**PSNR** peak signal-to-noise ratio

**QoR** quality of result

**RED** relative error distance

**RTL** register-transfer level

**SDLC** significance-driven logic compression

**SoC** system-on-chip

**VOS** voltage over-scaling

# Part I

# Thesis Chapters

# 1

# INTRODUCTION

A promising design paradigm—*approximate computing*—has recently emerged to harness imprecision resilience, in a broad spectrum of computing applications, to achieve additional optimizations. This chapter presents the motivation and defines the key concepts and terms in context of the research reported in this thesis. It highlights the necessity of approximate computing as a way for improving energy and performance efficiency in the field of arithmetic multiplier design. Then, the main contributions of this research together with thesis organisation are discussed.

## 1.1 MOTIVATION

Over the past decades, the continuing advances in technology scaling have shown an increasingly difficult design challenge to deliver high performance without dramatically increasing energy consumption [53, 26, 14, 7, 129]. Therefore, developing new solutions to tackle this challenge is considered as an imperative need for a number of reasons, such as improving hardware functionality for battery-powered computing devices, or increasing processing capabilities for those devices where energy is harvested from environmental sources. The motivation of this work is introduced as follows.

### 1.1.1 *Energy-Efficient Computing*

The computational performance demands driven by a massive volume of global data and a vast number of connected users, have exceeded the current processing capacity of the ever-evolving digital world [41]. It is expected that emerging applications will create 163 zettabytes of data by 2025 (ten times as much as was made in 2017) [98]. Furthermore, the global internet users have now passed the 4 billion mark (as of January, 2018) [61]. The vast majority of them were accessing their chosen platforms via mobile devices [61]. Moreover, the evolution of worldwide electricity demand of data centres [4] is related to the exponential growth in global digital-data creation [98]. This opens the way for finding alternative computing systems that use relatively less energy, while providing the processing performance they need.

For the last three decades, the need for higher computational performance has been supported by the exponential scaling of integrated circuits and many-core system-on-chip (SoC) technologies [24]. However, the rapid growth in these technologies is synchronized with the decline of *Moore's law* [81]. This means that, as the downscaling of the complementary metal-oxide-semiconductor (CMOS) is completely stretched to limits, technology scaling will less likely be a driver for computing in the near future [110, 105, 103, 62]. Moreover, the per-transistor performance power efficiency is not keeping pace with known power-reduction techniques at various abstraction levels [88, 34, 31]. This eventually results in the so-called *Dark Silicon* era [108], which means it may only be possible to power-on a fraction of on-chip computing resources in order to stay within the power density and safe thermal limits.

Indeed, there is a genuine need to explore new computing paradigms to deliver more energy efficiency and also, to squeeze more functionality out of computing platforms across the spectrum, from mobile and deeply-embedded devices to servers and data centres. Approximate computing paradigm is a promising approach to this end [33, 48, 78, 109, 107]. In the following sub section we introduce approximate computing and multiplier design in details.

### 1.1.2 *Approximate Multiplier Design*

Approximate computing is an emerging design paradigm that leverages the presence of inherent-resilience in a broad spectrum of hardware and software implementations by relaxing the need for completely precise or totally deterministic operations. It can offer substantial reductions in circuit complexity, delay and energy consumption by relaxing accuracy requirements [78]. Approximate computing has the potential to benefit a wide range of modern applications, such as media processing (image, audio, and video), digital signal processing (DSP), machine learning (recognition and data mining), wireless communication, web search, and data analytics [109]. It exploits the inherent-resilience code regions in such applications and also perceptual limitations of users to intelligently trade off outcomes accuracy for performance and energy gains [107].

The basic premise of approximate computing, in the hardware domain, is to build hardware blocks whose implementation does not exactly match the specification, either due to the impact of functional approximation (*e.g.*, implement a slightly different Boolean function that has a faster or more power-efficient implementation), or due to timing approximations (*e.g.*, voltage over-scaling

(VOS) and over-clocking). Therefore, approximate computing trades off output quality for achieving much lower power consumption, shorter run times, and often smaller silicon area at the cost of imprecision introduced to the processed data [17].

Over the years, a number of approximate computing approaches have been proposed (see Chapter 2). These approaches aim to reduce the complexity of the circuits and systems in terms of their computation latency and energy consumption [49]. Approximations can be introduced at all design levels, starting from the circuit [46] via the logic [92] and the architecture [2, 37] to programming language [101] and algorithms [57, 32].

Approximate arithmetic, such as approximate adders and multipliers, has gained significant interest in various group of imprecision-resilient applications [59, 49, 29, 120]. Approximate arithmetic is based on replacing traditional complex and energy-wasteful data processing blocks by low-complexity ones with reduced logic counts, and therefore, improving the computational performance and energy efficiency of the computing systems.

Multipliers are crucial arithmetic units in modern applications, for two major factors. Firstly, they are characterized by complex logic design, being one of the most energy-demanding data processing units in modern microprocessors. Secondly, compute-intensive applications typically exercise a large number of multiplication operations to compute outcomes. Thus, any improvement made in the power/speed of a multiplier is expected to substantially impact on overall system power/performance trade-offs [75, 58].

A typical $(N \times N)$ accurate multiplier generates $N^2$ product terms, which are then accumulated as a final product of size $2N$. The accuracy of this product depends largely on the significance of bits. Hence, preserving higher-significance bits is likely to gen-

erate an outcome close to the exact product than that of lower-significance bits. This can be exploited to progressively compress higher order combinatorial terms systematically and to achieve substantial energy savings at low loss of accuracy.

Existing multiplier design approaches consist of redesign efforts (such as reducing the circuit complexity (*i.e.*, critical path)) and parametric variations (such as supply voltage/frequency over-scaling). The basic principle in these designs is to improve energy/performance-efficiency at cost of functional or timing-induced errors. While these design approaches have made piecemeal advances in different directions, they leave rooms for further improvements (see Chapter 2). For example, *how can multiplier critical path be progressively reduced using different levels of approximations*.

## 1.2 THESIS SCOPE AND CONTRIBUTIONS

This thesis attempts to address the above fundamental question by exploring promising research directions in the development of energy-efficient multiplier design. It presents novel design approaches for improving energy and performance efficiency of approximate multiplier with variable-accuracy implementations. From the perspective of levels of abstraction, the research work in this thesis focuses on modifying the behavioural description of the multiplier design from gate level and register-transfer level (RTL). The proposed designs targets parallel multiplication, particularly for integer input operands. The aim is to develop design-time configurable prototypes to trade off output accuracy for various performance and energy gains. We apply the design approach of approximate multiplier to standard scheme of multiplication (*e.g.*

Wallace-tree). Additionally, we demonstrate a methodology to mitigate the impact of errors at various approximation levels, and study their impact on different imprecision-resilient applications. The main contributions of this thesis are listed as follows:

- a novel energy-efficient approach is proposed for approximate multiplier design using significance-driven logic compression (SDLC). The basic principle of the proposed approach is to cut the carry chain needed by accumulation tree, in a way that the significant bits of the product are slightly affected. To achieve this goal, the number of product terms is reduced by using low-complexity logic gates (*i.e.*, using OR gates to sum up these terms instead of expensive XOR gates) —we denote this method as *logic compression*. The reduced number of product terms is then followed by remapping, based on their commutative properties, to reduce the resulting number of rows in the partial product matrix (PPM). At the core of SDLC approach is a design-time configurable logic compression of product terms aims at achieving different performance-energy-quality (PEQ) trade-offs. Several multiplier prototypes (extending from 4-bit to 128-bit) are synthesized using electronic design automation (EDA) tools, showing up to an order of magnitude energy savings with various reductions in critical path delay and silicon area. Furthermore, extensive error analysis and comparison are placed to evaluate the efficiency of the proposed multiplier with respect to state-of-the-art multiplier designs [93, 94, 92].

- a novel energy-efficient approximate multiplier design by incorporating SDLC approach together with already existing column-compression methods, such as Wallace and Dadda-tree, to shorten the number of reduction stages. As such, the hardware complexity of the multiplier implementation is drastically re-

duced. Moreover, to minimize the impact of error associated with lossy compression, an error detection technique using parallel logic array along with error compensation method (ECM) is added at low overhead cost. The aim is to generate error compensation vector to either combine it as additional row in the accumulation tree or to replace one of the existing rows. A robust error analysis is included to evaluate the effectiveness of ECM. The post-synthesis experiments shows substantial saving in all design trade-offs, when comparing to accurate equivalent [92],

- the PEQ trade-offs achieved by the proposed multiplier designs are extensively investigated into two real-application case studies demonstrating comparative advantages of SDLC approach. First, a Gaussian blur filter with a meagre loss of image quality. Second, machine learning application using perceptron classifier with negligible error rate. For both case studies, substantial energy reductions are achieved and analysed [93, 94, 92].

## 1.3 THESIS OVERVIEW

This thesis is organized into six chapters. The major contributions of this thesis are summarized as follows:

Chapter 2 provides a coherent overview of extensively adopted and recently reported approaches for energy-efficient imprecise hardware. It defines the objectives of approximate computing, and also discusses the design approaches for approximate circuits. This is followed by a survey of approximate multiplier design to date. Various design techniques are discussed and classified based on multi-dimensional taxonomy to emphasize their similarities and differences. This chapter can inspire more efforts to address the

challenges facing quantifying errors derived from approximate arithmetic circuits, in particular, multipliers.

Chapter 3 proposes the SDLC approach for energy-efficient approximate multiplier design. It demonstrates how logic compression and commutative remapping are used to reduce the number of product rows. Error analysis and post-synthesis experiments are discussed, showing the PEQ trade-offs for different input bit-widths and compression levels. This chapter presents an algorithmic and design-time configurable lossy compression of the partial product rows. Additionally, it studies a group of related subjects covering the scalability of the SDLC approach and the feasibility of SDLC in signed multiplication. Comparative results against other existing approaches are also presented.

Chapter 4 studies the multiplier design by combining a Wallace-tree accumulation method together with the SDLC approach. A parallel error-detection logic and also error-compensation method (ECM) are presented, showing different ways to reduce the impact of error associated with SDLC. This chapter provides evaluation of the impact of ECM on variable logic compression, with extensive error analysis supported by a number of post-synthesis experiments.

Chapter 5 demonstrates the effectiveness of the PEQ trade-offs achieved by the proposed multiplier designs in applications. This chapter exhibits two case studies; image processing application using Gaussian blur filter and machine learning application using perceptron classifier. It also discusses the energy savings and quality loss arising from the implementations.

Chapter 6 summarizes the contributions and key highlights of this thesis, showing critical review of this research together with the potential future directions.

Overall, this thesis shows promising design and implementation approaches for approximate multiplier design, with different configuration. It can inspire designers to take advantage of the PEQ trade-offs, achieved by the SDLC approach, to achieve more energy and performance efficiency for ubiquitous computing systems.

# 2

# BACKGROUND AND LITERATURE SURVEY

## 2.1 INTRODUCTION

In last decade, there has been an increasing interest in approximate computing as one of the most promising energy-efficient computing paradigms. Approximate computing exploits the flexibility provided by inherent application resilience in hardware or software implementations for more performance and energy gains [100]. Approximate computing research combines insights from hardware engineering, architecture, system design, programming languages, and even application domains like image processing [17] and machine learning [84]. However, approximate computing based on software techniques has been deemed out of scope for this thesis.

This chapter highlights the basic concepts behind approximate circuits to understand the motivation and the choices made in the context of this work. First, we introduce and discuss several related imprecise computing in hardware to let the reader distinguish the efforts in approximate circuits from other related imprecise designs in the area. Then, a brief review is proposed for the recent developments in approximate circuits, showing the major approaches for exploiting the imprecision-resilience of various applications. Since the approximate multiplier design is the primary theme of this thesis, the second part of this chapter is made to emphasize the research efforts in the field of approximate multiplier design found so far with taxonomy provided. In

order to identify a trade-off between approximation and reliability, the existing error metrics are defined and classified to help the reader to specify right metric for a given approximate multiplier. Finally, a discussion is placed for the challenges of quantifying the imprecision in the multiplier design.

## 2.2 APPROXIMATE CIRCUIT DESIGN

In the literature, there are various research efforts devoted to addressing different designs of imprecise computing in hardware. In this section, we denote the hardware that does not produce the exact output at all times by *imprecise hardware*. The reason for this concept is to make it include a variety of existing designs that can lead to error in the output, such as in approximate, probabilistic and non-Boolean circuits. The next subsection is to let the reader understand the scope of approximate circuit design by distinguishing the work on it from related but conceptually distinct efforts in probabilistic and non-Boolean circuits.

### 2.2.1 *Imprecise Hardware Design*

In this section, imprecise hardware, including probabilistic, non-Boolean and approximate designs, is introduced to understand the scope of the approximate circuits from other related efforts in the literature. Fig. 2.1 presents a taxonomy of related terms in this area. Hence, these terms are briefly discussed with some illustrative examples —as follows:

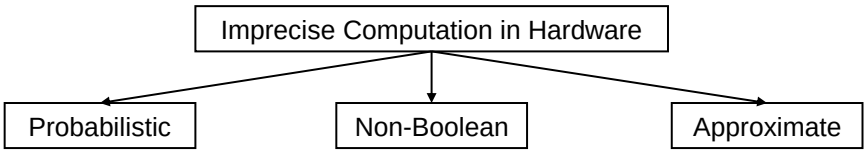


Figure 2.1: Taxonomy of imprecise computation in hardware.

#### 2.2.1.1 *Probabilistic circuits*

Three computing technologies are used by the circuit whose behaviour is inherently probabilistic: probabilistic CMOS (PCMOS), quantum and stochastic circuits. PCMOS and quantum circuits and signals are inherently probabilistic, while stochastic circuits utilize deterministic means for representing and processing probabilistic data [89, 77]. These computing technologies are briefly discussed as follows.

- Probabilistic CMOS is a particular form of CMOS called PCMOS, which is invented with a hope to compete against current energy-efficient CMOS technology. It is motivated by inherent "erroneous" behaviour in hardware as we move to smaller technology nodes. This is due to process variations and noise susceptibility [63, 83, 112]. Considering that CMOS devices have an exponential relationship between the *probability of exactness* ($P$) and the *switching energy* ($E$). To this end, PCMOS devices can harness the noise as a resource to achieve low-power and high-performance computation [16, 56]. However, the computing platforms based on PCMOS devices can compute efficiently at the cost of introducing error with a probability $(1-P)$ [16].

- Quantum circuits show another type of inherently probabilistic computations that perform efficient algorithms for some intractable problems in classical computer science [67, 87], such as fast number factorization [114]. Following the laws of quan-

tum physics, quantum circuits offers enormous processing power through the ability to be in multiple states, and to perform tasks using all possible permutations simultaneously. Quantum computing processes data in the form of *qubits* (quantum bits). A *qubit* $\Psi$ has two parts $\alpha$ and $\beta$ indicating its "zeroness" and "oneness" probabilities, respectively. It can be introduced in several standard forms [87]:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \\ \beta \end{bmatrix} \quad , \tag{2.1}$$

here $\alpha$ and $\beta$ are complex numbers called *probabilistic amplitudes*. When measuring the *qubit*'s value $\Psi$, quantum mechanics requires that the output be 0 or 1 with specific probabilities, implying that $|\alpha|^2 + |\beta|^2 = 1$. Generally, the inherent imprecision behaviour in quantum circuits comes from two reasons. First, the probabilistic results, *i.e.*, a quantum circuit generates different output values when run more than once with the same inputs. Second, when quantum circuit is implemented by a reduced number of quantum gates, *i.e.*, one or more quantum gates is missing, an example is demonstrated in Fig. 2.2.

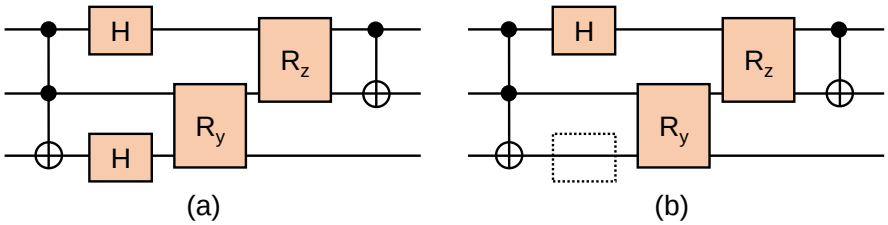


Figure 2.2: Example of a small quantum circuit *3qubitcnot*: (a) fault-free circuit, and (b) faulty circuit by eliminating a single H gate [6].

- Stochastic circuits is a class of computing circuits in which a number is represented by randomized values in serial "streams" or on parallel "bundles" of wires [91]. In the case of serially

streaming, the signals are probabilistic in *time* whereas, in parallel, they are probabilistic in *space*, as demonstrated in Fig. 2.3. Consider an $n$-bit stream $X$ containing $j$ "ones" and $n - j$ "zeros". The ratio $j/n$ can be seen as the probability $x$ of a randomly-observed bit in $X$ being "one".

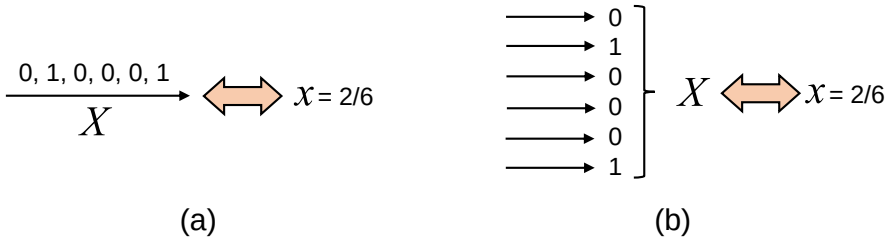


Figure 2.3: Stochastic encoding: (a) a stochastic bit stream; (b) a stochastic wire bundle. For each bit in the bit stream or a bundle, the probability that it is 1 is P($X$ = 1) = $x$.

The main attraction of stochastic hardware is its ability to carry out complex operations with simple circuits. For example, serial multiplication can be implemented by a two-input AND gate, see Fig. 2.4. Stochastic computing consists of operations on the $x_i$'s associated with a set of $X_i$'s of fixed or variable length $n$. In Fig. 2.4, the inputs denoted $x_1 = 9/12 = 0.75$ and $x_2 = 6/12 = 0.5$; the output signal probability is 4/12 = 0.333, which roughly approximates the product by using the input signal probabilities to $x_1 \times x_2 = 0.75 \times 0.5 = 0.375$.

Small errors like a 0 replacing a 1 due to a lost 1-pulse, have little effect on bit-stream signal probabilities, hence stochas-

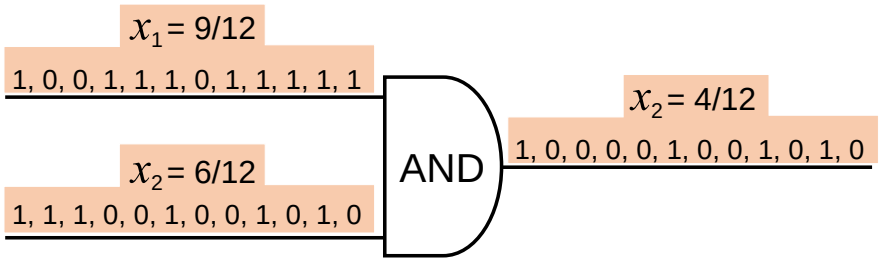


Figure 2.4: Stochastic multiplication using an AND gate.

tic computing is inherently tolerant of bit-flip errors. However, such computing has some drawbacks. For instance, meeting higher precision requirement leads to exponential increase in the computing times of the stochastic circuits, especially when not ensuring sufficiently low correlation among the circuit's signals [3]. To explain, the stochastic multiplier in Fig. 2.4 requires input signals that are pseudo-random and uncorrelated. Meaning that, if the input bit-streams are both identical in bit-for-bit representation $X$ of $x = 0.5$ , *i.e.*, maximally correlated, the output will be the same as $X$ with 0.5 instead of the expected product of $x_1 \times x_2 = 0.5^2 = 0.25$ —which is a huge error.

#### 2.2.1.2 *Non-Boolean circuits*

Non-Boolean computing refers to the class of computing where analog devices are used to store/process data. For example, a single storage bit can correspond to more than 2 values at any instant of time as opposed to combinational logic where a bit can only correspond to truth-values (0 or 1) [111, 68, 22]. Non-Boolean circuits can go beyond these definitions. It has proven to be much more efficient at various computational task, such as simulating biological systems using analog transistors [50], than digital computing. However, since analog computing uses continuous values, processes cannot be reliably repeated with exact equivalence [19]. Moreover, existing analog computers have to be programmed manually, *i.e.*, a complex process that would be very time consuming for large-scale and application-specific simulations [51]. However, the potential for these technologies to perform useful non-Boolean computations remains an opportunity to be explored.

#### 2.2.1.3 *Approximate Circuits*

Approximate circuits refer to a class of computing where deterministic designs are used to produce outputs with some tolerable level of missing accuracy in favour of improving performance and energy-efficiency. Approximate computing does not involve assumptions on the probabilistic nature when implementing circuits. However, it often utilizes statistical properties to trade quality for energy/power reduction [48]. For example, Fig. 2.5 illustrates the accurate and approximate circuits for (2 × 2) multiplier suggested in [65]. The error statistics of the approximate circuit are shown in Table 2.1. The main objective is to introduce error into the multiplier by manipulating its logic function. It can be observed that it is possible to represent the output of (2 × 2) multiplication using just three bits instead of four. As a result, the approximate circuit has an incorrect output out of the sixteen possible inputs (with a magnitude of 9 – 7 = 2 and a probability of 1/16 (assuming a uniform input distribution). However, the approximate multiplier in Fig. 2.5-(a) has nearly half the area of the accurate one, with shorter critical path and less interconnect. Thus, the approximate version offers the potential for significant dynamic power reduction

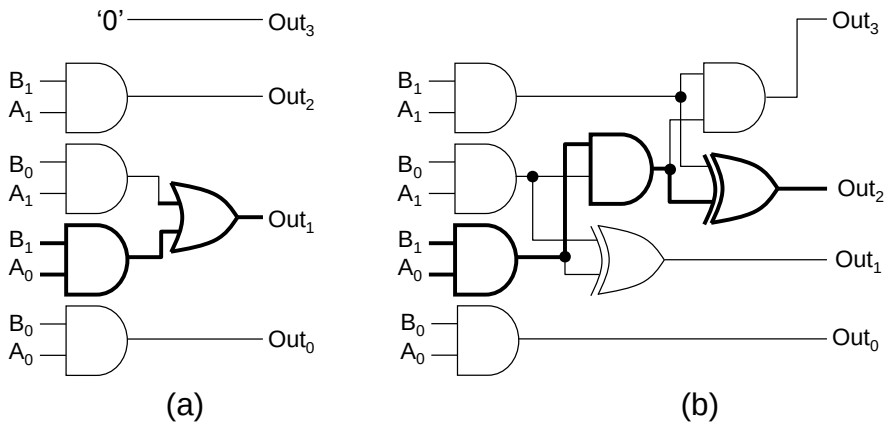


Figure 2.5: Example of a (2 × 2) multiplier: (a) approximate, and (b) accurate, with the critical paths highlighted [65].

Table 2.1: Truth table for the accurate and approximate (2×2) multipliers, used to obtain comparative error analysis in Fig. 2.5, with changed entry highlighted.

| Inputs | | | | Outputs | | Error | |
|---|---|---|---|---|---|---|---|
| B1 | B0 | A1 | A0 | Accurate | Approximate | Free | Distance |
| 0 | 0 | 0 | 0 | 0000 | 0000 | √ | 0 |
| 0 | 0 | 0 | 1 | 0000 | 0000 | √ | 0 |
| 0 | 0 | 1 | 0 | 0000 | 0000 | √ | 0 |
| 0 | 0 | 1 | 1 | 0000 | 0000 | √ | 0 |
| 0 | 1 | 0 | 0 | 0000 | 0000 | √ | 0 |
| 0 | 1 | 0 | 1 | 0100 | 0100 | √ | 0 |
| 0 | 1 | 1 | 0 | 0010 | 0010 | √ | 0 |
| 0 | 1 | 1 | 1 | 0011 | 0011 | √ | 0 |
| 1 | 0 | 0 | 0 | 0000 | 0000 | √ | 0 |
| 1 | 0 | 0 | 1 | 0010 | 0010 | √ | 0 |
| 1 | 0 | 1 | 0 | 0100 | 0100 | √ | 0 |
| 1 | 0 | 1 | 1 | 0110 | 0110 | √ | 0 |
| 1 | 1 | 0 | 0 | 0000 | 0000 | √ | 0 |
| 1 | 1 | 0 | 1 | 0011 | 0011 | √ | 0 |
| 1 | 1 | 1 | 0 | 0110 | 0110 | √ | 0 |
| 1 | 1 | 1 | 1 | 1001 | 0111 | × | 2 |

for the same frequency of operation —due to smaller switching capacitance.

The key principle of the above imprecise hardware designs (*i.e.*, probabilistic, non-Boolean and approximate) is to trade little accuracy loss in the output for improving performance and energy efficiency in emerging computing systems. However, a taxonomy of approximate circuits is further discussed in the following sections, showing their potential to benefit many emerging applications using different design approaches. In the next subsection, we define the intrinsic attribute of imprecision-resilience in a wide range of applications.

#### 2.2.1.4 *Explioting Imprecision-Resilience*

Approximate computing exploits the inherent resilience in a broad spectrum of hardware implementations by relaxing the need for completely precise or totally deterministic operations. Recently, however, ideas have emerged how to make errors help reduce runtime of operations and power consumption. Errors are becoming an intentional feature in the hardware design because of tangible trade-offs and application-based error-resilience. Error-resilience can be defined as the characteristic of a system to produce acceptable results regardless of task computations being achieved imprecisely.

Approximate computing design requires a deeper knowledge of inherent resilience applications [48]. It can improve the performance and energy efficiency by exploiting perceptual limitation (*i.e.*, errors are not recognizable because human perception capabilities, *e.g.*, in multimedia applications). Such applications can adequately operate without the need of functional equivalence of the specification and implementation. These applications are common in the hardware circuits mobile, embedded, and server systems and can be broadly classified into four categories [127, 35, 48]:

- applications with analog inputs that operate on noisy real-world data, such as image processing, sensor data processing, voice recognition,
- applications with analog output prepared for human perception, such as multimedia, image rendering, sound synthesis,
- applications with no golden answer, such as web search and machine learning, and
- iterative and convergent applications that iteratively process large amounts of redundant data and the quality of result (QoR)

depends on the number of iterations. This redundancy often means that an algorithm can be lossy and still be sufficient.

Approximate circuits harness imprecision-resilience in the above applications and thereby offer substantial reduction in delay and energy consumption. This can be done by different explicit design approaches, which are discussed as follows.

### 2.2.2 *Taxonomy of Approximate Circuits*

Applying the concept of approximate computing in hardware consists of two basic design approaches [107, 109, 78]:

- Timing-induced approximation (voltage over-scaling (VOS) and over-clocking), and
- Functional approximation.

Fig. 2.6 shows a taxonomy of design approaches in approximate circuits. The voltage over-scaling [96, 76] and over-clocking [27, 28] approaches use ordinary circuits that work perfectly fine under usual circumstances. The first design approach can reduce power consumption by scaling the supply voltage below nominal value, which can cause the occurrence of occasional violations. Generally, in CMOS technology, the power consumption associated with task

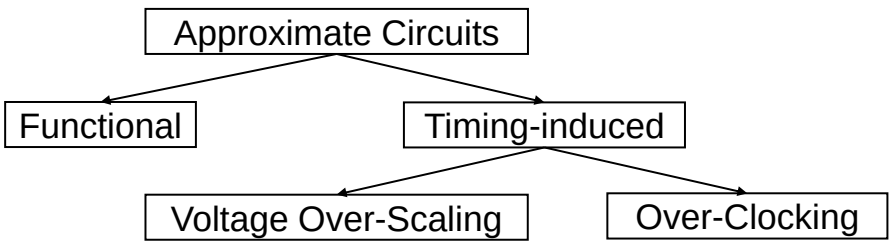


Figure 2.6: Taxonomy of approximate circuits.

is dominated by dynamic power dissipation $P_{dynamic}$, which is given by [95]:

$$P_{dynamic} = C_{eff} \cdot V_{dd}^2 \cdot f \quad , \tag{2.2}$$

where $V_{dd}$ is the supply voltage, $C_{eff}$ is the effective switched capacitance, and $f$ is the clock frequency. Therefore, scaling down the supply voltage leads to an overall quadratic reduction in the energy to complete a task. However, while maintaining a fixed performance, aggressive scaling of supply voltage leads to timing errors. The literature have shown different ways to reduce the impact of such induced errors in the systems, such as adding error detection and correction circuits, which often operate within normal supply voltage [86].

Over-clocking can achieve more performance by improving circuit's working frequency over the maximum frequency. This leads to the occurrence of timing errors, but better overall performance [27, 28]. Generally, for any given voltage, the circuit will have a maximum “stable" speed where they still operate correctly. However, while operating on constant voltage, a designer may trade the manufacturer's safety margin by setting the device to run in the higher end of the margin. For example, a work towards “overclocking friendly”, in online arithmetic implementations, shows that exercising over-clocking technique for serial operations can gain large performance benefits with a graceful degradation of timing violations [113].

Unlike voltage/frequency over-scaling, functional approximation does not use the original circuit but a specially designed one instead. This circuit is redesigned without fully Boolean logic implementation described in the specification. For instance, a common method for implementing functional approximation is to omit the

less significant bits of the result by removing related logic. This is expected to reduce the circuit complexity (*i.e.*, critical path), and therefore, achieving more performance/energy efficiency than the accurate version, with some amount of error in the less significant bits of output. However, the above-mentioned approaches used by approximate circuits are further discussed from the perspective of multiplier design, since it is the main theme of this thesis. The next subsection introduces a brief overview for applying the concept of approximate computing in arithmetic circuits.

#### 2.2.2.1 *Approximate Arithmetic*

Approximate arithmetic, such as approximate adders and multipliers, can be exploited as means of reducing energy requirements, increasing speed, minimizing cost and improving reliability in various computing systems [59]. They have been largely present in computing systems using fixed-point [42] and floating point operations [120]. The main goal has been to obtain exact or close to exact outputs by using error mitigation techniques ranging from increasingly sophisticated rounding to interval arithmetic [29].

Literature has presented different techniques for approximate arithmetic [59], particulary for adders and multipliers. Comparing to multipliers, approximate adders have been attracting most attention [48]. The main design approaches for approximate adders are surveyed into [59, 29, 48]. The next section discusses the related work in the domain approximate multipliers found up to date.

## 2.3 APPROXIMATE MULTIPLIER DESIGN

A multiplier design has a significant impact on the performance and power dissipation of modern applications [58, 29], such as digital signal processing, multi-media, computer vision, robotics, machine learning, pattern recognition and data analytics. This is due to two main reasons:

- Multiplier is one of the most energy-demanding data processing units in arithmetic processors. Compared to other units, such as adders, multipliers have complex logic design in terms of logic cell counts and length of critical path.
- Most of the algorithms exercised by compute-intensive applications use a large number of multiplication operations to compute results.

Thus, as a crucial arithmetic unit, any improvement made in the power/speed of a multiplier are expected to largely impact on the overall system performance/energy trade-offs. In the literature, many approaches have been used to generate various approximate multiplier designs. Not all of these approaches are entirely fall under the concept of approximate computing. For instance, multipliers have been redesigned based on PCMOS technology [43], quantum theory [18] and stochastic concept [115]. Another design approaches target the multiplier for FPGA-based hardware accelerators [123, 122]. However, the next part presents a taxonomy of approximate multiplier designs that is then used to survey and classify the state-of-the-art research in this area.

### 2.3.1 *Taxonomy and Survey*

In this section, we present a taxonomy of six different directions to classify the related research efforts of approximate multiplier design (see Fig. 2.7). From the perspective of approximate circuit approaches, these efforts can be largely categorized as modifications of either timing or functional behaviours.

Firstly, the timing behaviour of multiplier design can be altered using aggressive supply voltage scaling or over-clocking techniques [79, 73, 113]. Operating below nominal voltage allows for reductions in energy consumption at the cost of time-induced errors. The bounds of these errors can be predicted, and so extra error-

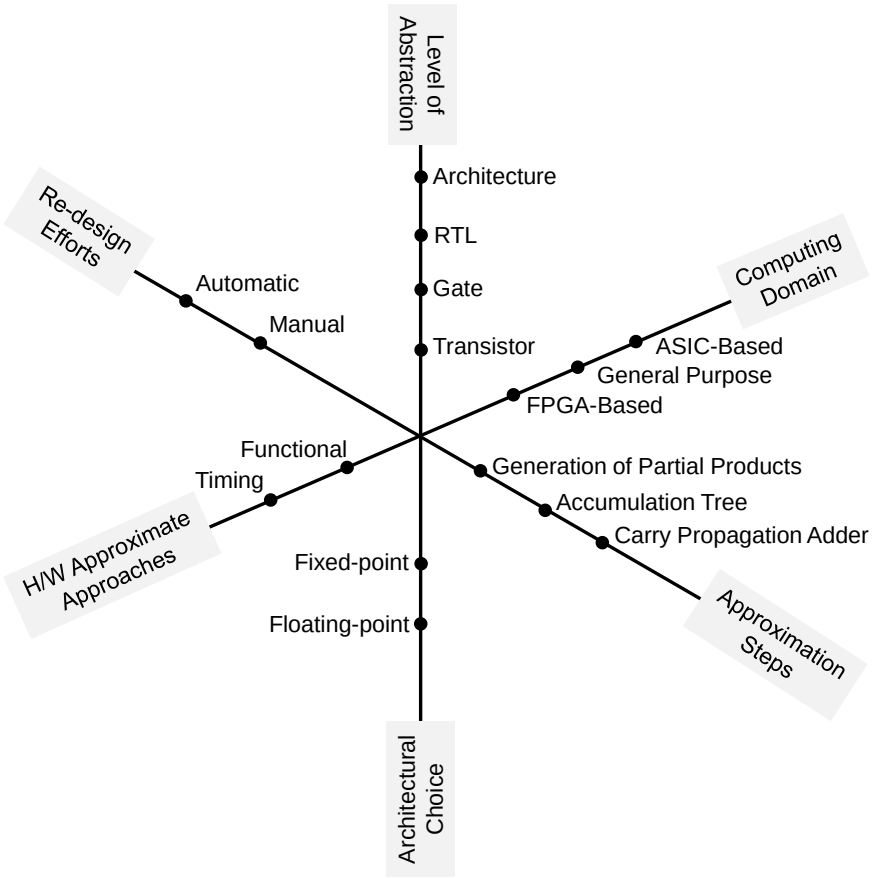


Figure 2.7: Multi-dimensional taxonomy of approximate multiplier designs.

compensation circuits need to be incorporated [30, 44]. Fig. 2.8 demonstrates an example of subjecting VOS to sequential 2-bit multiplier circuit. After scaling the supply voltage below the nominal value, a number of paths fails to meet the delay constraints. However, the timing failures are typically caused by long carry chains, *i.e.*, impact the more significant bits of the final product (seeFig. 2.8-b). It is therefore necessary to quantify the impact of timing violation by modifying the conventional multiplier to allow for graceful degradation [113]. Subjecting VOS or over-clocking to arithmetic circuits without losing the accuracy of the output is still an open challenge (This challenge will be discussed further in Section 2.3.2).

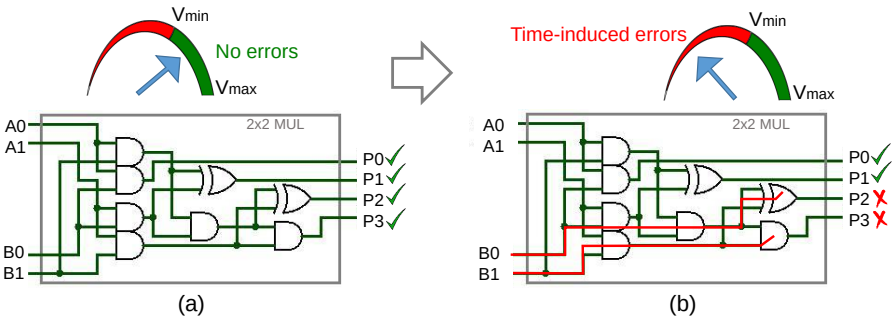


Figure 2.8: An example of (2×2) multiplier: (a) operating within safe voltage range; (b) lowering the supply voltage below its nominal value.

Secondly, functional modifications deal with logic reduction techniques and can be performed by relaxing the need for accurate Boolean equivalence of the specification and implementation. This is leveraged to achieve more energy-efficiency and to accelerate computations.

A common example of functional modifications of approximate multipliers is truncation of the least significant columns, such as when performing fixed-point [25] (*i.e.*, two n-bit input multiplicands producing an n-bit output) and floating-piont multiplication [131]. Truncating multiplier product terms allows for the

elimination of some of the least significant columns in the partial product matrix (PPM). Fig. 2.9 shows different degrees of truncation in the case of (8 × 8) approximate multiplier. Truncated multiplication provides an efficient method for reducing the power dissipation and silicon area [25, 90, 118, 119, 60]. As more columns are eliminated, further energy reduction is achieved; however, errors increase very rapidly. Additionally, as the operand size increases, the relative reduction in energy and cell count also increase [104]. Note that, critical path will not be remarkably shortened unless the columns truncated exceed the critical column–which leads to a huge error (see Fig. 2.9). Therefore, there is an essential need to verifying the truncated multipliers and quantifying associated

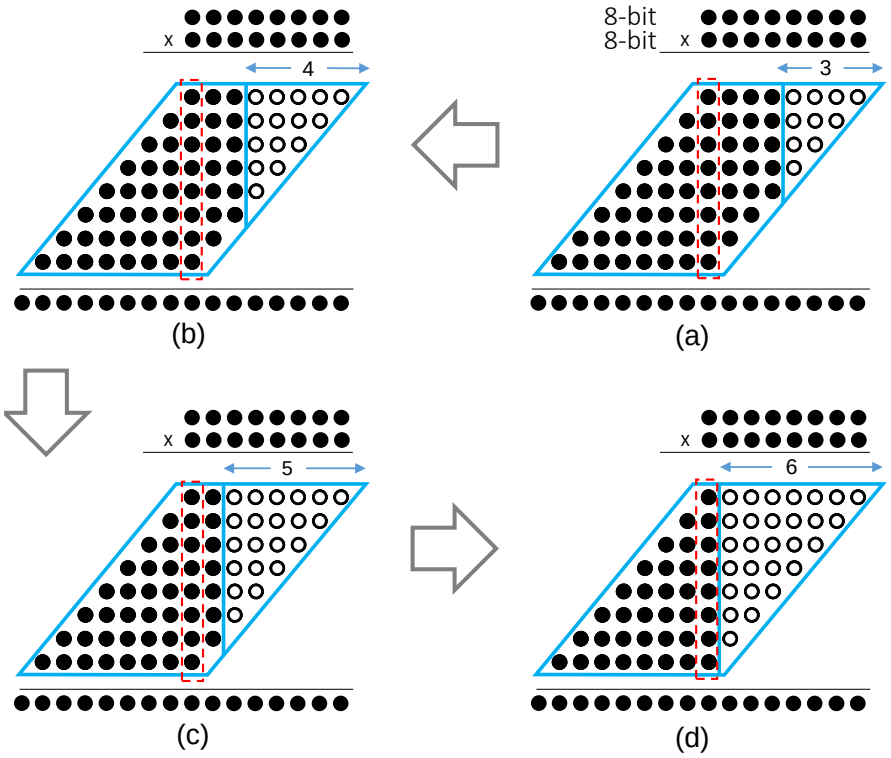


Figure 2.9: Increasing the level of truncation from; (a) 3; (b) 4; (c) 5; and (d) 6 columns, translates into additional reductions in area and power; however, error is maximized and this method is not effective to reduce the critical column of the accumulation tree (highlighted in dotted rectangles).

errors. A number of heuristically inspired schemes exist in the literature and attempt to minimise the impact of error, such as average absolute error or mean square error. Another attempts to create faithfully rounded multipliers based on truncation [64] and analytic error bounds [25].

Modular re-design with low-complexity combinational logic is another effective technique [65, 71]. This allows for building larger energy-efficient multipliers using small approximate ones; however, the hierarchical organization of small approximate blocks will eventually propagate errors, which increase with the multiplier size. Fig. 2.10 describes the process of designing larger multipliers when using $(2 \times 2)$ approximate multiplier as a building blocks. The increased input bit-width of the multiplier translates into higher size of accumulation tree leading to increased height of critical column. However, scalability is not simple and this method may not significantly reduce the critical path, also the hierarchical organization of small approximate blocks will eventually propagate errors which increase with the multiplier size (see Fig. 2.10).

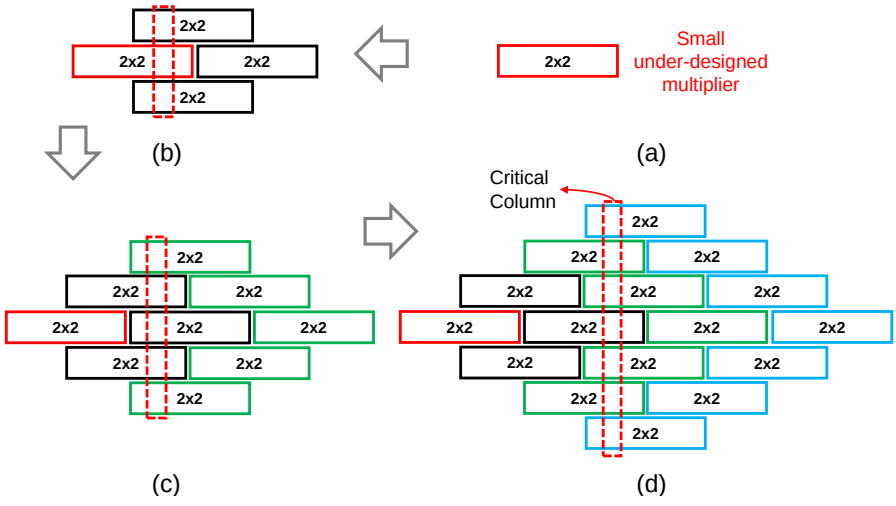


Figure 2.10: Using $(2 \times 2)$ approximate multiplier blocks to build larger energy-efficient multipliers [65].

Fig. 2.11 describes another attempt to design large efficient multiplier, constructed by small approximate and accurate ones, using recursive multiplication scheme [8]. The idea is to use the approximate multiplier blocks to produce the less significant bits of the product while the more significant bits are generated using accurate blocks. To reduce the latency of the accurate multiplier two ways have been applied; (i) applying the recursive scheme once more using smaller multiplier blocks, and (ii) adding carry-in prediction logic to reduce the carry chain through addition stage (see Fig. 2.11).

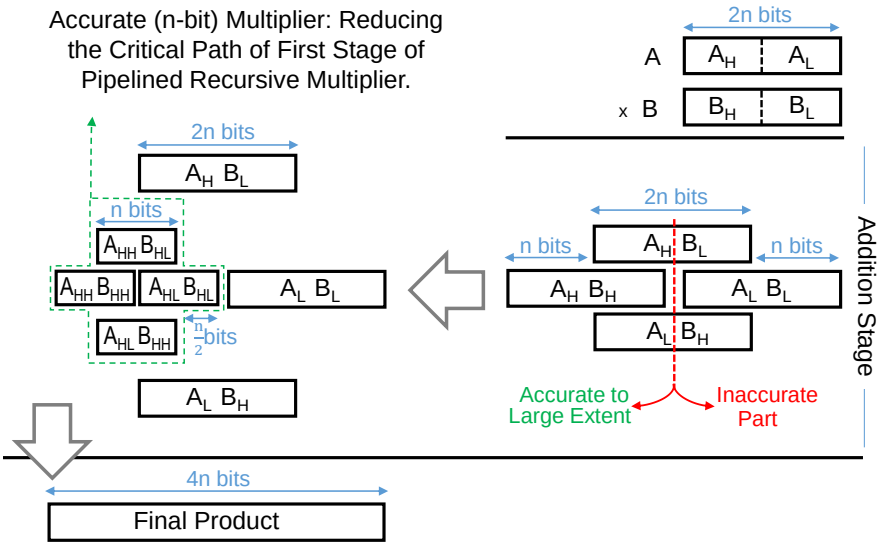


Figure 2.11: Different sizes of approximate and accurate multipliers are used to build large recursive multiplier for pipelined architecture [8]; the carry-in logic is used for the approximate partial product computation only ($A_HB_L, A_LB_H, A_LB_L$) and not for the accurate $A_HB_H$.

A software-based perforation technique has been proposed [134] by obtaining the optimized set of partial product terms based on power-area-accuracy trade-offs. The partial product perforation technique omits the generation of $k$ successive partial products starting from the $j^{th}$ one. Fig. 2.12 depicts an example of applying the partial product perforation method on an 8-bit multiplier with

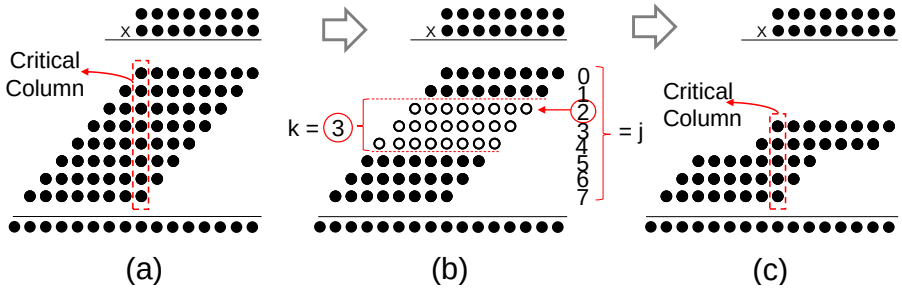


Figure 2.12: The partial product accumulation tree: (a) of an accurate (8× 8) multiplier; (b) design parameters are set using multiple EDA tools, and (c) the reduced partial product matrix after applying the partial product perforation with $j$ =2 and $k$ = 3 [134].

$j$=2 and $k$=3 configuration values. These parameters are controlled by various EDA tools at design-time for optimal approximate multiplier configurations and error constraints. The literature shows a number of power- and area-efficient multiplier redesign approaches, that have been proposed by changing the functional behavior. These changes extend from the architecture [133, 52] to transistor-level [47].

Generally, the structure of a parallel multiplier consists of three main components: the partial product generator, partial product accumulation-tree, and final adder. Thus, the efforts in the domain of the approximate multiplier design can be classified based on the approximation in these components. For example, trucation and software-based perforation [134] are common methods for applying approximation through partial product formation. Such methods jointly consider the deletion, reduction, truncation, and rounding of partial product bits to minimize the hardware complexity associated with the partial product reduction-tree [64].

However, some design techniques begin to generate all product terms, as the accurate multiplier, and then involve approximation to speed up the process of partial product reduction in the accumulation-tree. To achieve this goal, one of the common de-

sign technique is to use approximate compressors [45, 12], such as (4:2) compressors to decrease the number of the reduction stages of a Wallace-tree [70] and Dadda-tree multipliers [80]. Another design technique is to omit some cells of CSA in array multiplier design, leading to a smaller and faster multipliers while introducing approximate results [74], or by reducing the logic counts in a speculative carry-save reduction tree using ($k$:2) counters with $k > 3$ [21]. Recently, algorithmic method for allocating high-order approximate compressors, obtained in a systematic way, aims at improving energy efficiency with minimizing the error probability and the average error [38].

Lastly, the final adder can be approximated by means of cutting the carry-chain required for summing product terms. Thereby, this way is expected to improve the performance efficiency, particularly with column-compression multiplier designs, such as Wallace- and Dadda-tree methods. This is because final adder is considered as dominant component of the multiplier delay. A recent work in [36] presents a realistic MAC architecture in which accurate generation and Wallace compression for the partial-products are followed by a final (approximated) carry propagating adder.

Automated design approaches [125, 127, 85, 126, 97] present design flows for synthesizing approximate circuits using circuit activity profiles and quality bounds. For example, the systematic methodology for automatic logic synthesis of approximate circuits (SALSA) [125] begins with an RT-level description of the accurate circuit and an error constraint. The approach introduces Q-function, which determines whether the quality constraints are satisfied, when comparing both approximate and original outputs. Fig. 2.13 describes the use of Q-function to formulate the problem of approximate synthesis. The idea is to iteratively modify the

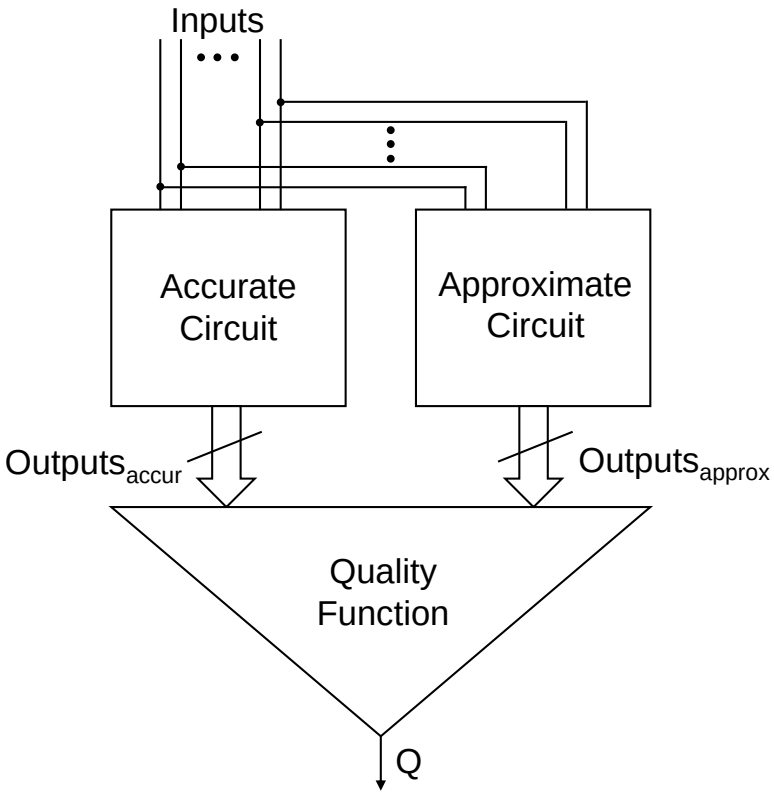


Figure 2.13: The quality constraint circuit proposed by SALSA [125] (Q is a single Boolean value).

behavioural description of approximate circuit with the aim of reducing the complexity of approximate circuit, with keeping the output of the Q-function unchanged.

Recently, evolutionary circuit design shows some evolved implementations of target circuits that can be considered as innovative [124]. However, the evolutionary design approach fails in producing useful implementations for approximate complex circuits, such as 32-bit or 64-bit approximate multiplier. A promising evolutionary design process has been implemented within the ABC tool and extensively evaluated on functional approximation of different sizes of multipliers up to 32-bit [15]. Table 2.2 summarizes the key features and limitations of research efforts to date in the domain of approximate multipliers.

Table 2.2: Summary of approximate multiplier design approaches.

| Approach | Methodology | Features and Limitations |
|---|---|---|
| [79] [73] | Aggressive voltage scaling: lowering the supply voltage below its nominal value. | Unexpected time-induced errors, which normally impact the most significant bits. |
| [90] [118] | Truncation: eliminating partial products from the least significant columns. | As more columns are eliminated, the resulting errors are maximised. |
| [65] [71] | Modular re-design: large efficient multipliers using inaccurate small multiplier blocks. | Scalability is not simple and this method may not significantly reduce the critical path. |
| [134] | S/W-based perforation: approximation of the generation of the partial products. | Decreasing the depth of the accumulation tree by utilizing a tool, and also real-time needs. |
| [125] [127] | Automated and evolutionary re-design: systematically reducing the complexity of circuits. | Greedy approach depending on circuit activity profile and output significance. |
| [9] [47] | Manual re-design: manual alteration of the functional behaviours of the structure. | Different ideas of redesigning the multiplier extend from architecture to transistor level. |

Conflicting performance-energy-accuracy tradeoffs of the above design techniques make it difficult to select the most suitable approximate multiplier for a specific application. Recently, many comparative evaluations of the existing approximate multipliers can be found in the literature [75, 12, 58, 82]. However, while the benefits of approximate multiplier design (*i.e.*, reductions in power, delay and often area) are unquestionable it is necessary to carefully evaluate the impact of errors committed by different design techniques for approximate multiplier design. Hence, the following part discusses the evaluation of the imprecision associated with different approximations of multiplier design.

### 2.3.2 *Error Evaluation and Challenges*

Various approximate multipliers aim at achieving energy/delay/area reduction at the cost of a certain amount of imprecision introduced to the output. Many emerging applications are inherently able to tolerate the presence of a certain amount of such imprecision for many reasons, such as intrinsically embed redundancy and perceptual limitations (see Section 2.2.1.4). Thus, for safety-critical applications, it is very important to identify a trade off between approximation introduced by multiplication and reliability. Generally, the imprecision of approximate multipliers is quantified as an error with respect to the accurate result and measured using a wide variety of error metrics.

#### 2.3.2.1 *Error Metrics*

Different error metrics have been developed for the purpose of examining the impact of imprecision introduced by approximate arithmetic circuits [69, 13, 72, 106]. In this thesis, the error metrics

in the domain of approximate multiplier can be classified into four groups as follows:

- error distance ($ED$): a group of error metrics describes the magnitude of deviation of an approximate product from the corresponding accurate one. The worst-case error is defined as the maximum absolute $ED$ can be obtained for the all possible inputs. Sometimes, a condition of $ED \geq \tau$ is followed to consider the set of inputs that makes the approximate output differs from the accurate one by a threshold $\tau$. Another metrics can be derived from $ED$, such as the average error magnitude (*i.e.*, the average of all $ED$s), which is expressed as the ratio of all absolute $ED$s over the number of input operands,

- error probability ($EP$): which measures the rate of occurrence of imprecise outputs in the approximate multiplier. A careful estimation of $EP$ is beneficial when sequential multipliers decide to spend more clock cycles for error corrections [69, 8]. $EP$ can also be deemed as one of design characteristics, in which a predefined threshold of $EP$ should not be exceeded,

- error significance ($ES$): a group of error metrics quantifies the error severity of the output of the approximate multiplier. $ES$ can thus determine if a approximate multiplier generates a small or a huge error. Many error metrics have been derived from the general definition of the error significance such as relative error distance ($RED$) which defined as the ratio of $ED$ over the accurate output. Another related metric is the mean relative error distance ($MRED$) (*i.e.*, the average of all $RED$s obtained from all possible input combinations), that can be considered as an effective indicator for testing the quality degradation [13], and

- normalized error distance ($NED$): a group of nearly invariant metrics, which is almost independent of the size of an approximate multiplier implementation. Thus, $NED$ is an important metrics for characterizing the reliability of multiple-bit approximate designs [69].

Since approximate multipliers are often used for achieving power/delay/area gains, different error metrics can be connected with these gains to derive reliable metrics. For example, the product of power and $NED$ is further utilized for assessing the trade-offs between power consumption and precision. Although the above metrics have been addressed for multipliers, these metrics are potentially useful in evaluating the different types of approximations in the arithmetic circuit designs. Note that, there is a group of quality metrics used to evaluate the final outcome in various applications, such as peak signal-to-noise ratio (PSNR) [55] for measuring the quality of the image after processing with approximate circuits. However, for this thesis, the error metrics used to evaluate the impact of different approximation is further discussed in Section 3.4.

#### 2.3.2.2 *Error Evaluation Challenges*

The basic premise of the above-mentioned metrics is to evaluate the approximation introduced in multiplier design with respect to the accurate outputs. However, in order to provide a more comprehensive picture of the error evaluation, this thesis briefly discusses some challenges facing the process of quantifying imprecision of approximate multiplier design:

- quantifying time-induce errors introduced by over-scaling and over clocking. One of the suggested methods for this challenge

is to convert timing-induced approximations into the functional domain. An example introduced in [127], requires generation of an equivalent ”untimed circuit", which represents the functional behaviour of the over-scaled or over-clocked circuit. Thus, constructing the ”untimed circuit" means that the error analysis can be quantifiable by comparing its outputs with the accurate circuit,

- large bit-width multiplier poses another challenge when the error analysis requires to go through all input combinations, such as finding exact worst-case error. One of the suggested methods for this challenge is to create analytical model [25]; however, this not straightforward, since it generally depends on the way that multiplier is approximated. Another available solution is statistical analysis using Monte-Carlo simulation to obtain error distribution for a selected ranges of the input operands. Similar challenge takes place when checking errors for approximate multiplier using systematic and evolutionary approaches. For example, checking of a predefined worst-case error for larger sizes of multipliers needs *relaxed equivalence checking* [15], which can be performed based on satisfiability (SAT) solving [127] and based on binary decision diagrams (BDDs) [116]. However, the research area of *relaxed equivalence checking* is rather unexplored as all existing formal approaches have been developed for accurate equivalence checking [54],

- specifying a suitable error metric is highly application dependent [107]. Meaning that, there is a need for a deeper understanding of inherent application resilience across a wide range of applications [17]. Moreover, relying on a single or a group of error metrics (such as error probability and average of errors), is

often not effective for perfectly quality evaluation of final output of a certain application, and

- error accumulation is another challenge for those applications which iteratively exercise an approximate multiplier as a basic unit. For instance, for a given application, feedback circuits move back the product to the input side of sequential multiplier design. Therefore, the quality of final output can be affected and still continue operating in faulty states for many clock cycles. Thus, it is important to build a robust error analysis in which a designer can investigate any likely issue of error accumulation.

## 2.4 CONCLUDING REMARKS AND DISCUSSIONS

This chapter provides background and literature survey of approximate circuits, with focusing in multiplier design. We distinguish the efforts in approximate circuits from other energy-efficient circuits, such as probabilistic (*e.g.*, PCMOS, quantum and stochastic) and non-Boolean. Approximate circuits consists of functional and voltage/frequency over-scaling approaches. These design approaches leverage the intrinsic imprecision-resilience in a wide spectrum of emerging applications to improve the efficiency of computing systems. As the approximate multiplier design is the primary theme of this thesis, a multi-dimensional taxonomy and comprehensive survey of the efforts in the domain of approximate multiplier design, are summarized to date. The key principle of these efforts is to achieve reduced logic complexity for improving energy efficiency at minimal precision loss. Nevertheless, the existing techniques for approximate multiplier face different challenges (see Table 2.2). For instance, truncated multiplier offers significant improvements in area and power; however, while more

columns are eliminated, as means of reducing the hardware complexity of accumulation tree, the resulting errors are maximised. Also, additional overhead cost may be incurred by implementing required error-compensation circuits. Another example when utilizing under-designed multipliers to build large but energy-efficient ones, scalability is not simple and this method is not efficient to reduce the length of critical column of accumulation tree. To this end, in the following chapter, we will show an effective logic design approach that use significance-driven logic compression (SDLC) to mitigate the above challenges.

# 3

# LOGIC COMPRESSION IN MULTIPLIER DESIGN

## 3.1 INTRODUCTION

The previous chapter surveys methods for improving energy efficiency within the domain of approximate circuit design. It provides a taxonomy of various techniques for approximate computing to make it easier to understand the merits of these techniques and also to discuss expected cost of tackling approximate computing across all design levels.

This chapter identifies the main core research contribution of this thesis. It presents a novel approach for approximate multiplier design using significance-driven logic compression (SDLC). Fundamental to this approach is an algorithmic and configurable lossy compression of the partial product rows based on their progressive bit significance. This is followed by the commutative remapping of the resulting product terms to reduce the number of product rows. As such, the complexity of the multiplier in terms of logic cell counts and lengths of critical paths is drastically reduced. A number of multipliers with different bit-widths (4-bit to 128-bit) are designed in SystemVerilog and synthesized using Synopsys Design Compiler. Post-synthesis experiments showed a substantial decrease in run-time, power consumption and even in silicon area, compared to an accurate equivalent. These gains are achieved with low accuracy losses estimated using different error metrics. Additionally, the performance-energy-quality (PEQ)

trade-offs are demonstrated for different degrees of compression, achieved through configurable logic clustering. A comparative analysis is included to evaluate the efficiency of the proposed multiplier against other recent approximate multiplier designs in the literature. Also, the scalability of SDLC approach and the feasibility of performing signed multiplication are discussed.

## 3.2 SIGNIFICANCE-DRIVEN LOGIC COMPRESSION APPROACH

Without loss of generality, let us consider two $N$-bit binary inputs for an $(N \times N)$ multiplier, the multiplicand $(A = a_{N-1}2^{N-1} + \cdots + a_0)$ and the multiplier $(B = b_{N-1}2^{N-1} + \cdots + b_0)$. The product $P$ can be expressed as:

$$P = A \cdot B = p_{2N-1}2^{2N-1} + \cdots + p_0 = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_i b_j 2^{i+j} \tag{3.1}$$

In parallel multiplication design, computation of $P$ is generally divided into three consecutive stages: i) partial product formation, ii) partial product accumulation, and iii) carry propagation adder. Fig. 3.1 shows the difference between the design stages in accurate and the proposed multiplication. First, $N^2$ AND gates are utilized in parallel to generate $N^2$ product terms of partial product matrix (PPM). This matrix is then column-wise summed up by using different accumulation methods, such as *carry-save array*, *Wallace* [128] and *Dadda-tree* [23], followed by carry propagating adder (CPA) to generate the final $2N$-bit product. The performance, hardware complexity and power consumption associated with multiplier design depend largely on the maximum height of the accumulation tree. The proposed approach decreased the number of vertical product

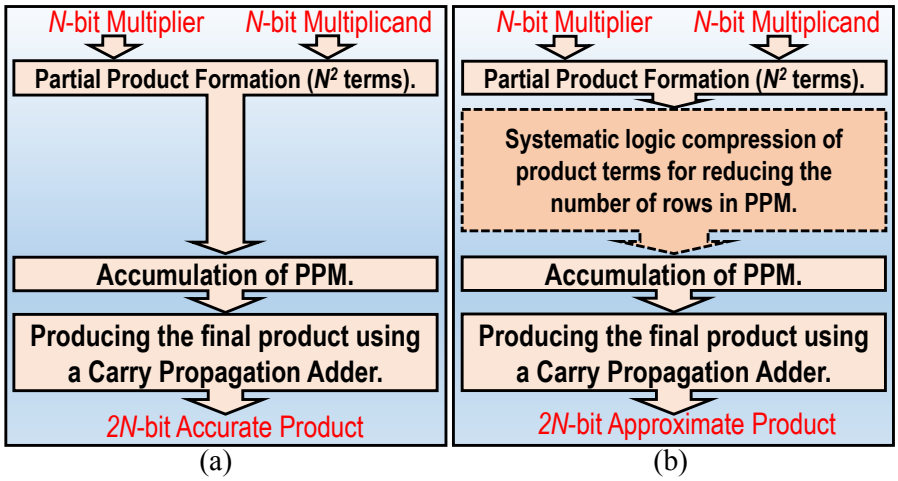


Figure 3.1: Process chart showing the difference between the major stages in: (a) conventional multiplication, and (b) the proposed approach to multiplication.

terms in the PPM with the aim of reducing the height of the critical column in the accumulation tree. This can be achieved by following two major steps demonstrated in Fig. 3.2. First, lossy compression is carried out through logic clustering. Second, the resulting compressed terms are then remapped using their commutative properties. As can be seen in Fig. 3.2, all partial products are generated using $N^2$ AND gates, similar to conventional multiplication. Before proceeding to the accumulation stage, the number of bits in the partial product matrix is reduced by performing lossy logic compression. The aim is to minimize the number of rows in the PPM, thereby achieving low-complexity hardware before proceeding to accumulation. To achieve lossy compression, we follow three key principles as follows.

### 3.2.1 *Logic Clustering*

The proposed multiplier organizes the partial product terms using different sizes of significant-driven logic clusters. Each logic cluster

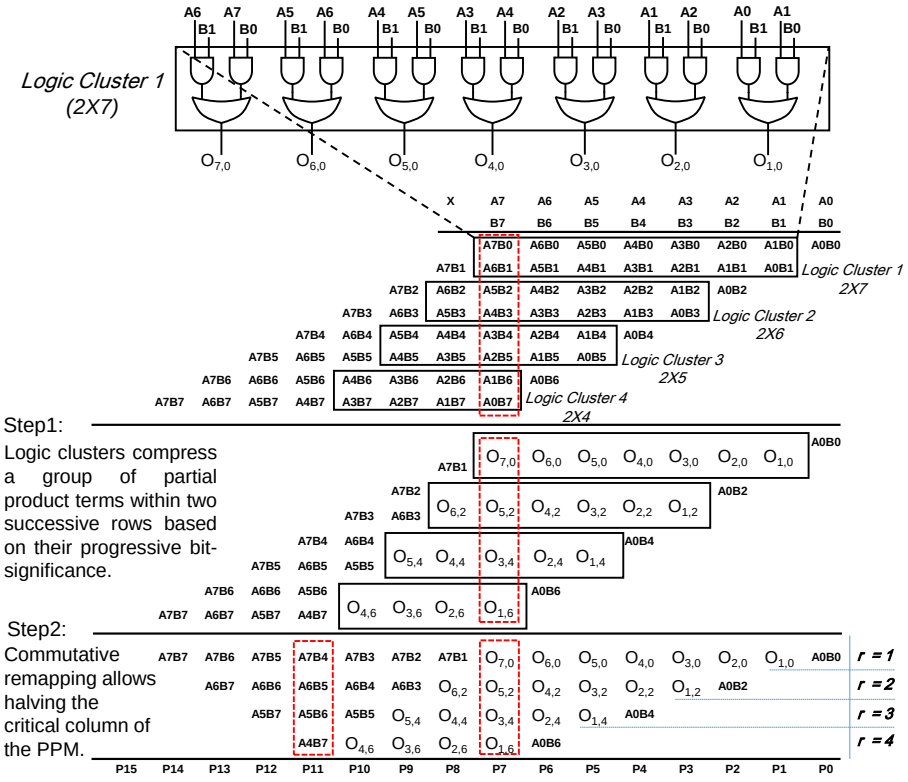


Figure 3.2: Stylized demonstration of SDLC approach [93]: four different sizes of logic clusters used to compress partial products based on their progressive bit-significance in $(8 \times 8)$ parallel multiplier architecture.

targets a group of columns containing two bits starting from the least significant bits in successive partial products.

Two adjacent partial product terms belonging to the same column can be compressed in a single term by using 2-input OR gate (see Fig. 3.2). Let us consider two of vertically aligned bits within two successive partial products $a_i b_j$ and $a_{i-1} b_{j+1}$ of the $(i+j)^{th}$ column in the PPM. $O_{i+j}^{2-bit}$ is an output of a logic cluster, expressed as:

$$O_{i+j}^{2-bit} = a_i b_j \vee a_{i-1} b_{j+1} \quad . \tag{3.2}$$

For purposes of illustration, the logic cluster of size $(2 \times L)$ targets a group of consecutive partial product terms of a length of $L$ columns

within 2 rows. Each $(2 \times L)$ logic cluster is responsible for two operations: i) generating $2L$ partial product bits within two contiguous rows, *i.e.*, $L$ pairs of vertically aligned bits, by utilizing $2L$ AND gates. Then, ii) minimizing these $2L$ bits by half using $L$ OR gates. Fig. 3.2 illustrates the utilization of four sizes of logic clusters in $(8 \times 8)$ parallel multiplier. The first $(2 \times 7)$ logic cluster forms 14 partial products by utilizing 14 AND logic gates and extracts 7-bit value by using an array of 7 OR logic gates. The second $(2 \times 6)$ logic cluster minimizes 12 partial products into 6 bits. In a similar way the third and fourth logic clusters use $(2 \times 5)$ and $(2 \times 4)$ to minimize 10 and 8 partial products into 5 and 4 bits, respectively. By doing so, each logic cluster compresses a group of vertically aligned bits within two successive partial products based on their progressive bit significance.

### 3.2.2 *Logic Compression*

Using an array of OR gates in each logic cluster compresses the partial product terms by half (see Fig. 3.2). A reduced set of pre-processed partial product matrix is thus ready to be accumulated by applying any convenient scheme of multiplication. In theory, a two-input OR gate is sufficient to sum up two bits, *i.e.*, '0'+'1' = '1'+'0' = '0'∨'1' = '1'∨'0' = '1' and also '0'+'0' = '0'∨'0' = '0'. However, the OR gate fails to give an accurate sum if the two inputs are "ones", *i.e.*, '1'+'1' $\neq$ '1'∨'1', the difference value is '1' as the adder returns '10' and OR outputs '1'. So, the arithmetic sum of two successive partial products belonging to the $(i+j)^{th}$ column, can be approximated as:

$$a_i b_j + a_{i-1} b_{j+1} \simeq O_{i+j}^{2-bit} \quad . \tag{3.3}$$

By utilizing a parallel OR compressions through logic clusters, the number of product terms inside the PPM will decreased at the price of an error when the couple of partial product terms $a_i b_j$ and $a_{i-1} b_{j+1}$ are both high, *i.e.*, the error will be when $a_i b_j + a_{i-1} b_{j+1} \neq O_{i+j}^{2-bit}$. Under assumption that the input bits $a_i$ and $b_j$ are uniformly and independently distributed, the probability of having this error is given by: 1/16 (*i.e.*, when all $a_i$, $b_j$, $a_{i-1}$, $b_{j+1}$ equal to '1'). However, the OR compression will not affect the more significant bits of the final product as demonstrated below, moreover error analysis is discussed in Section 3.4.

### 3.2.3 *Progressive Cluster Sizing*

Since the main goal is to design a power-efficient multiplier with negligible loss of accuracy, the length of the logic clusters $L$ is decreased when going down in the PPM. The more significant bits are treated with progressively higher precision, while bits with lower significance are compressed using the SDLC approach. This permits the most significant product terms to be accumulated on a carry-propagation basis as in the conventional multiplier. Thus, the accuracy of the significant bits of the final product is less affected. In general, the length of logic cluster in 2-bit compression used to produce $L$-bit array in the $r^{th}$ row of the compressed PPM, is given by:

$$L_{2-bit}(r) = N - r \quad . \tag{3.4}$$

Despite using the same number of AND gates as the accurate multiplier, this approach will deterministically reduce the hardware complexity of partial product accumulation, i.e. the count of

the compressor cells needed in column compression multiplication for *Wallace* and *Dadda* cases, and also the number of half and full adders in the *carry-save array* will be decreased since the number of bits in the accumulation tree is minimized.

### 3.2.4 *Commutative Remapping*

In the logic compression step (Section 3.2), the number of partial product terms is reduced. This can be leveraged to minimize number of rows in the PPM prior to the accumulation stage. For this purpose, the partial product terms resulted from the logic compression are remapped based on the commutative property of the bits, *i.e.*, bits with the same weight are gathered in the same column.

For example, in the case of $(2 \times L)$ logic cluster, the height of the critical column is reduced by half compared to the accurate accumulation tree. The compressed and remapped bit-matrix after applying commutative remapping of the bit sequence, is shown at the bottom of Fig. 3.2. Due to the reduced number of rows, the critical path delay is drastically shortened (see Section 3.5). The height of the critical columns are further reduced with increased logic compression depth as seen below.

### 3.2.5 *Example of Utilizing 2-bit SDLC*

To illustrate the multiplication process using 2-bit SDLC, an arbitrary example of $(42,617 \times 38,587)$ is executed using $(16 \times 16)$ propose multiplier is as shown in Fig 3.3.

Eight logic clusters of 2-bit depth are used to minimize the number of product terms in the PPM. The length of the logic clusters is decreased from 15 bits for the first one to 8 bits for the eighth. The

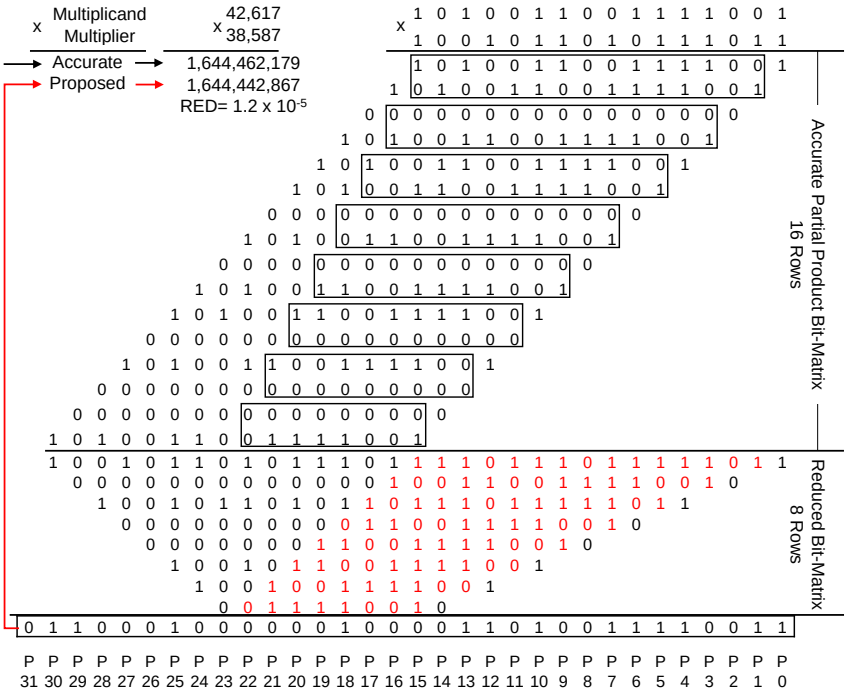


Figure 3.3: Eight different sizes of 2-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

reduction in the size of the logic clusters allows more significant product terms to be accumulated in carry-propagation basis like accurate multiplier (see Section 3.2.3). The logic compression exercised by the OR-gate arrays reduces the number of product terms of the PPM. This is leveraged to minimize the number of the rows in the accumulation tree by applying the commutative remapping discussed (see Section 3.2.4). Thus, 8 rows are resulted after applying the proposed SDLC approach instead of 16 rows in the case (16 × 16) accurate multiplier. From multiplier's design perspective, decreasing the number of rows in the accumulation tree can reduce the hardware complexity and also the critical delay (see Section 3.5). The error distance (ED) between the accurate and the proposed multiplier is (1,644,462,179 – 1,644,442,867 = 19,312). The relative error distance (RED) is the ratio of the error difference over the accurate value (RED = 19,312 / 1,644,462,179 = $1.2 \times 10^{-5}$).

The above values of ED and RED are expected to increase with other binary numbers (*e.g.*, string of 1's on both multiplicands). This is due to the increased likelihood of having two vertically aligned 1's within two successive rows —logic cluster will give error (see Section 3.2.2). For further error analysis, Section 3.4 investigates the impact of error derived from the proposed multiplier, using different error metrics (such as worst-case ED and error probability). Additionally, next chapter introduces an error-compensation circuit to mitigate the impact of error derived from SDLC (see Section 4.3).

## 3.3 VARIABLE LOGIC CLUSTER AND SCALABILITY

The proposed approach is capable of achieving higher degrees of compression by increasing logic cluster depth. Comparing to the work in [93], the space of the product terms compressed by logic cluster has been modified, when the depth of logic compression spans over three or more rows in the PPM. This will provide scalability and improve the quality of the proposed SDLC approach. In this work, the level of logic compression obtained from $(d \times L)$ logic cluster is denoted by $d$-bit logic compression, where $d$ and $L$ indicate the depth and the length of product terms targeted by a logic cluster, respectively.

Fig. 3.4 demonstrates the impact of increasing depth from 2- to 3- and 4-bit logic cluster in the case of $(8 \times 8)$, showing the key steps in logic compression and commutative remapping. As can be seen, with increased depth we can achieve further reduction in the partial product terms, leading to fewer rows for final accumulation and therefore more reductions in the hardware complexity and energy consumption of the multiplier design (see Section 3.5).

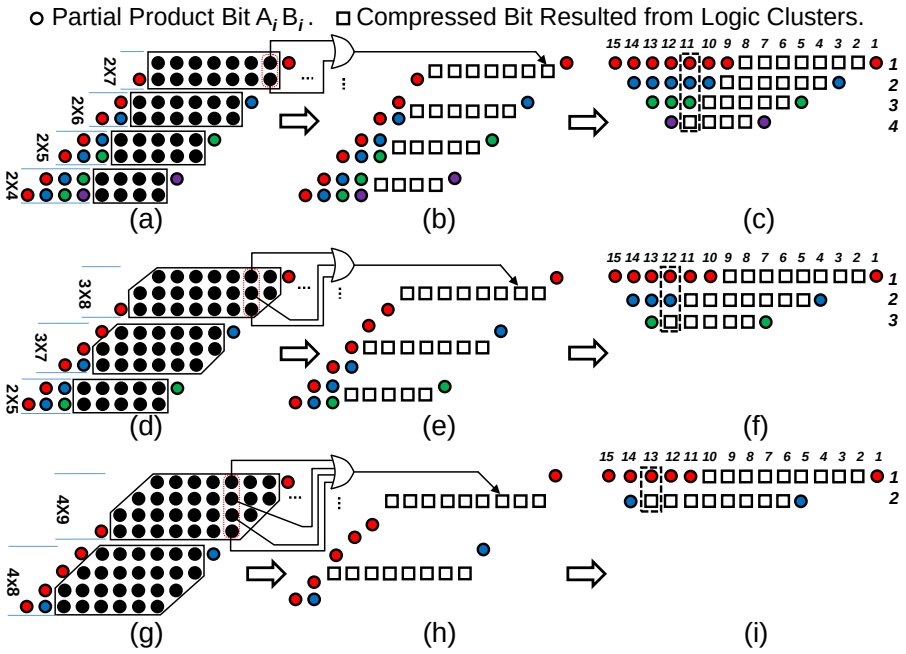


Figure 3.4: Dot diagram showing the impact of increasing the depth of the logic clusters in the case of $(8\times8)$ multiplier: (a) clustering a group of bits within 2 successive rows in the partial product bit-matrix after bitwise multiplication; (b) generating a reduced set of product terms after targeting the depth of 2-row logic compression; (c) ordered matrix after applying commutative remapping of the bit sequence resulting from the SDLC approach; (d), (e) and (f) the same process when applying 3-bit logic compression; (g), (h) and (i) the same process when applying 4-bit logic compression. The dotted rectangles at the right indicate the heights of the critical columns which are further reduced compared to the accurate accumulation tree.

However, this is achieved at the cost of increased error. The impact of increasing the depth of logic clusters is demonstrated in the next section.

### 3.3.1 *General Space of d-bit Logic Cluster*

Fig. 3.5 presents a general $d$-bit logic cluster. In general, $(d\times L)$ logic cluster targets a group of consecutive partial product terms of a length of $L$ columns within $d$ rows in PPM. A $(d\times L)$ logic cluster is used to compress the product terms into a row of $L$

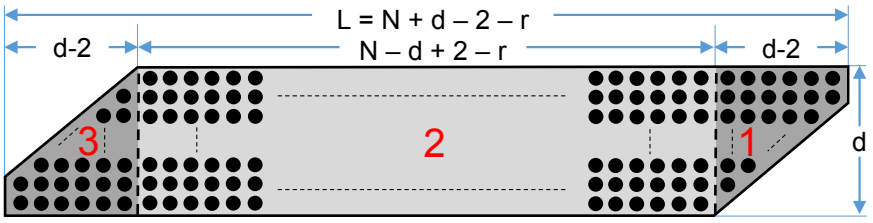


Figure 3.5: Dot diagram showing the general space of targeted partial product terms compressed by a $(d \times L)$ logic cluster to produce array of $L$ bits in $r^{th}$ row of the reduced partial product matrix for $(N \times N)$ multiplier using SDLC approach with $d$-bit logic compression.

terms using an array of $d$-input OR gates. By following the same concept in (3.2), let us consider a depth of $d$ successive product terms belonging to the $(i+j)^{th}$ column in the PPM $a_i b_j$, $a_{i-1} b_{j+1}$, $a_{i-2} b_{j+2}$, $\cdots$, $a_{i-(d-1)} b_{j+(d-1)}$. Thus, $O_{i+j}^{d-bit}$ is an output term of logic cluster returned from logic compression in $(i+j)^{th}$ column and can be expressed as:

$$O_{i+j}^{d-bit} = \bigvee_{k=0}^{d-1} a_{i-k} b_{j+k} \quad . \tag{3.5}$$

The arithmetic sum of $d$ successive partial products belonging to the $(i+j)^{th}$ column, can be approximated as:

$$\sum_{k=0}^{d-1} a_{i-k} b_{j+k} \simeq O_{i+j}^{d-bit} \quad . \tag{3.6}$$

In general, an $N$-row PPM resulted from exact $(N \times N)$ multiplier can be compressed to $\lceil \frac{N}{d} \rceil$ rows, when utilizing SDLC approach with $d$-bit logic cluster. Let us consider $r$ as the row index of the compressed and remapped partial product bit-matrix, in which $r$ is ranging from 1 (the first row in the top), to $\lceil \frac{N}{d} \rceil$ (the last row at the bottom). The length of logic cluster $L$ represents the number of columns targeted by logic compression, and equals to the length of bit array added into $r^{th}$ row. $L$ can be expressed by:

$$L_{d-bit}(r) = \begin{cases} (N+d-2)-r, & 1 \le r < \lceil \frac{N}{d} \rceil \\ (2N-3)-(d+1)(r-1), & r = \lceil \frac{N}{d} \rceil \end{cases} \tag{3.7}$$

The length of last row is separated in the second line of (3.7) to involve the length of the last logic cluster in case of $N \mod d \neq 0$, where the depth of the last cluster equals $N \mod d$. For instance, applying SDLC approach to $(8\times8)$ multiplier with 3-bit compression requires $\lceil \frac{8}{3} \rceil = 3$ logic clusters, the first two logic clusters compress group of product terms within 6 rows, *i.e.*, 3 rows each, leaving only two rows for the last logic cluster spans over 5 columns, see Fig. 3.4 (d). Note that, the most significant part of PPM, where the height of the PPM is already lower or equal to $\lceil \frac{8}{d} \rceil$, logic compression is not required. In general, if $i$ is the column index in the compressed and remapped PPM, (noting that $i$=1 indicates to the least significant column), the last column that can have an OR compression can be obtained as:

$$i_{last} = 2N - \lceil \frac{N}{d} \rceil - 1 \quad . \tag{3.8}$$

The last column affected by OR approximation in the case of $(8 \times 8)$ multiplier is highlighted by dashed-line box in the Fig. 3.4 (c), (f) and (i)).

### 3.3.2 *d-bit Logic Cluster: Compression Algorithm*

Algorithm 3.1 explains the process of forming $L$-bit array resulted from $(d \times L)$ logic cluster for any $N$-bit multiplier. According to (3.7), the length of the logic cluster depends on the row index of the compressed and remapped PPM. For all $r$, except the last

**Algorithm 3.1** Generating the output bits of the logic cluster of the $r^{th}$ row for the proposed $(N \times N)$ multiplier using $d$-bit logic clusters.

**procedure** LOGICCLUSTER($A, B, r$)
  Output:$C[c_1, c_2, \cdots, c_L]$ //Output bits from logic cluster
  Inputs: $A[a_1, a_2, \cdots, a_N]$ //Multiplicand bits
  $B[b_1, b_2, \cdots, b_N]$ //Multiplier bits
  $r$ //Row index of the compressed and remapped PPM

  **if** $\frac{N}{d} = \lceil \frac{N}{d} \rceil$ *or* $r \neq \lceil \frac{N}{d} \rceil$ **then**
//Forming (N+d-r-2)-bit output if (N mod d = 0)
    $\omega \leftarrow 1$ //initialize column index
    $length = N + d - 2 - r$
    **for** $x \leftarrow 1$ *to* $d$-2 **do**
      $C[\omega] \leftarrow (A[x+1] \wedge B[d(r-1)+1]) \vee (A[x] \wedge B[d(r-1)+2])$
      **for** $y \leftarrow 1$ *to* $x-1$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x-y] \wedge B[d(r-1)+2+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**
    **for** $x \leftarrow 1$ *to* $N-d+2-r$ **do**
      $C[\omega] \leftarrow (A[x-d+1] \wedge B[d(r-1)+1]) \vee (A[x+d-2] \wedge B[d(r-1)+2])$
      **for** $y \leftarrow 1$ *to* $d-2$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x+d-2-y] \wedge B[d(r-1)+2+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**
    **for** $x \leftarrow 1$ *to* $d$-2 **do**
      $C[\omega] \leftarrow (A[N-r-1] \wedge B[d(r-1)+1+x]) \vee (A[N-r] \wedge B[d(r-1)+2+x])$
      **for** $y \leftarrow 1$ *to* $d-2-x$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[N-r-y] \wedge B[d(r-1)+2+x+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**

  **else if** $N \mod d = 1$ **then**
//Forming (2N-(d+1)(r-1)-3)-bit output if (N mod d = 1) in last row
    $length = 2N - (d+1)(r-1) - 3$
    **for** $x \leftarrow 1$ *to* $length$ **do**
      $C[x] \leftarrow (A[x+1] \wedge B[N])$
    **end for**

  **else**
//Forming (2N-(d+1)(r-1)-3)-bit output if (N mod d ≠ 1 ≠ 0) in last row
    $\omega \leftarrow 1$
    $length = 2N - (d+1)(r-1) - 3$
    $\acute{d} = N \mod d$
    **for** $x \leftarrow 1$ *to* $\acute{d}-2$ **do**
      $C[\omega] \leftarrow (A[x+1] \wedge B[d(r-1)+1]) \vee (A[x] \wedge B[d(r-1)+2])$
      **for** $y \leftarrow 1$ *to* $x-1$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x-y] \wedge B[d(r-1)+2+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**
    **for** $x \leftarrow 1$ *to* $length - 2(\acute{d}-2)$ **do**
      $C[\omega] \leftarrow (A[x+\acute{d}-1] \wedge B[d(r-1)+1]) \vee (A[x+\acute{d}-2] \wedge B[d(r-1)+2])$
      **for** $y \leftarrow 1$ *to* $\acute{d}-2$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x+\acute{d}-2-y] \wedge B[d(r-1)+2+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**
    **for** $x \leftarrow 1$ *to* $\acute{d}-2$ **do**
      $C[\omega] \leftarrow (A[N-r+1] \wedge B[d(r-1)+1+x]) \vee (A[N-r] \wedge B[d(r-1)+2+x])$
      **for** $y \leftarrow 1$ *to* $\acute{d}-2-x$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[N-r-y] \wedge B[d(r-1)+2+x+y])$
      **end for**
      $\omega \leftarrow \omega + 1$
    **end for**
  **end if**
  **return** $C$
**end procedure**

one, the algorithm begins to generate $(N+d-2-r)$ bits to each row. To explain, the first $(d-2)$ less significant bits as indicated in Line 9 to 15 and highlighted in area 1 (see Fig. 3.5). In this area, the logic cluster compresses two partial product terms in the least significant column and increases the level of compression to $(d-1)$ partial products in the $(d-2)^{th}$ column. This is followed by forming $(N-d+2-r)$ bits of area 2 in Fig. 3.5, as referred in the Lines 16 to 22. Lastly, Lines 23 to 29 indicate generation of the more significant $(d-2)$ bits of area 3. However, when $(N \bmod d) \neq 0$, the logic cluster returns $(2N-3)-(d+1)(r-1)$ bits at the last row. In such a case, if the value of $(N \mod d) = 1$, *i.e.*, logic compression in the last row is not required, the product terms generated by bitwise ANDing as the exact PPM, see Lines 32 to 34, otherwise, when $(N \mod d) = \acute{d}$, where $\acute{d} \neq 1$ and $\acute{d} \neq 0$ (see line 38), the logic compression is required by using $d'$-bit logic cluster using the same steps, see Lines 36 to 62.

However, for purposes of illustration, each row in the compressed and remapped partial product bit-matrix can be subdivided into four parts (for instance, each row illustrated in the Fig. 3.4 (c), (f) and (i))

(i) the less-significant zero bits that represent the number of consecutive shifts at the beginning of each row, given by:

$$l_{zeros}(r) = d(r-1) \quad . \tag{3.9}$$

Then,

(ii) the first product term which is unaffected by the logic compression which can be formed as:

$$A(1) \wedge B(d(r-1)+1) \quad , \tag{3.10}$$

followed by,

(iii) the array of $L$ bits returned from logic cluster given by (3.7) and demonstrated in Algorithm 3.1. Lastly,

(iv) the most significant part of the $r^{th}$, which is unaffected by the logic compression, the length of bits that compose the unaffected most significant part in the $r^{th}$ is obtained as:

$$l_{MSB}(r) = \begin{cases} N - dr + 1, & 1 \leq r < \lceil \frac{N}{d} \rceil. \\ 1, & r = \lceil \frac{N}{d} \rceil. \end{cases} \tag{3.11}$$

### 3.3.3 *Scalability for (N × N) SDLC Multiplier Design*

The proposed SDLC approach using $d$-bit compression is scalable for any $(N \times N)$ multiplier, as shown in Algorithm 3.2. This algorithm generates a compressed and remapped partial product bit-matrix $M$ Line 21, which can then be treated as an accumulation tree by any scheme of multiplication, such as *carry-save*, *Wallace* and *Dadda* accumulation methods. The main loop Lines 5 to 20 is responsible for forming and remapping product terms in $\lceil \frac{N}{d} \rceil$-row approximated matrix, as demonstrated in Fig. 3.4. Lines 7 to 10 indicate forming of the less-significant zero bits that represent the number of consecutive shifts at the beginning of row $r$, managed by (3.9). The first product term unaffected by the logic compression is formed in Line 11. This followed by generating $L$ bits from the logic cluster using (3.7), this is shown in Line 13 by retrieving Algorithm 3.1. Lines 15 to 20 refer to forming the most significant part of $r$, which is unaffected by the logic compression, described by (3.11).

**Algorithm 3.2** Generating a reduced partial product matrix M for $(N \times N)$ multiplier using SDLC approach with $d$-bit logic clusters, $\forall\{d \in \{2,3,...,N\}$.

**procedure** RPPM($M,A,B$)

Output:$M = \begin{bmatrix} m_{1,1} & \cdots & m_{1,2N} \\ \vdots & \vdots & \vdots \\ m_{\lceil \frac{N}{d} \rceil,1} & \cdots & m_{\lceil \frac{N}{d} \rceil,2N} \end{bmatrix}$ //Compressed Matrix

Inputs: $A[a_1,a_2,\cdots,a_N]$ //Multiplicand bits

$B[b_1,b_2,\cdots,b_N]$ //Multiplier bits

//Forming rows of $M$

**for** $r \leftarrow 1$ *to* $\lceil \frac{N}{d} \rceil$ **do**

$\rho \leftarrow 1$ //initialize column index

//Shifting by zeros at the beginning of row $r$

**for** $z \leftarrow 1$ *to* $d(r-1)$ **do**

$M[r][\rho] \leftarrow$ "0"

$\rho \leftarrow \rho + 1$

**end for**

//Forming unaffected first bit in row $r$

$M[r][\rho] \leftarrow A[1] \wedge B[d(r-1)+1]$

$\rho \leftarrow \rho + 1$

//Forming outputs of logic cluster $r$

$M[r][\rho:\rho+length] \leftarrow$ LOGICCLUSTER$(A,B,r)$ //*length* is defined using (3.7)

$\rho \leftarrow \rho + length + 1$

//Forming unaffected MSBs in row $r$

**for** $k \leftarrow 1$ *to* $N-dr$ **do**

$M[r][\rho] \leftarrow A[N-r+1] \wedge B[rd+k-1]$

$\rho \leftarrow \rho + 1$

**end for**

$M[r][\rho] \leftarrow A[N-r+1] \wedge B[N]$

**end for**

**return** $M$ //$M$ is then treated as a reduced accumulation tree

**end procedure**

### 3.3.4 *Examples of Utilizing d-bit SDLC*

In Section 3.2.5, we illustrate an example of the multiplication process using 2-bit SDLC. In this section, the same example of $(42,713 \times 37,548)$ is executed using $(16 \times 16)$ propose multiplier using 3-, 4-, 5-, 6-, 7- and 8-bit depth of logic clusters, as shown in Fig 3.6 to Fig 3.11.

In the case of 3-bit SDLC, five logic clusters are used to minimize the number of product terms in the partial product bit-matrix. The length of the logic clusters are decreased from 16 bits for the first one to 12 bits for the fifth (as described by (3.7)). Compared to 2-bit SDLC, both length and the depth of the logic clusters are increased for all compression levels (from 3- to 8-bit SDLC). This can explain the increasing trend in the error difference from 610032 (RED =

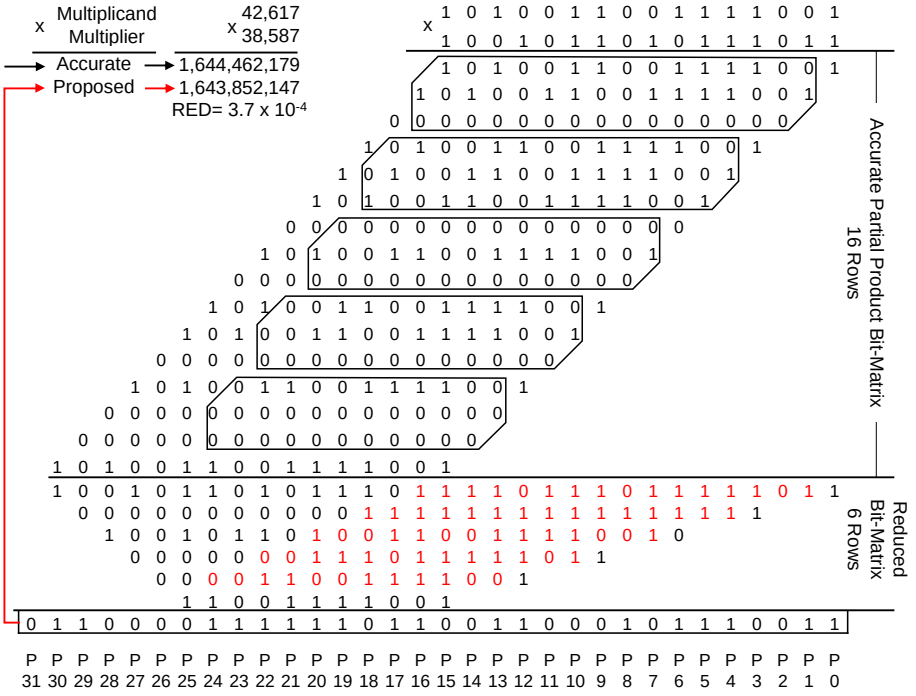


Figure 3.6: Five different sizes of 3-bit logic clusters used to compress partial products based on their progressive bit-significance in $(16 \times 16)$ parallel multiplier architecture.

$3.7 \times 10^{-4}$) in the case of 3-bit SDLC to 44337768 (RED = 0.02315) in 8-bit SDLC.

In the examples shown in Fig 3.6 to Fig 3.11, the logic compression exercised by the OR-gate arrays reduces the number of product terms of the partial product bit-matrix. Thus, instead of having 16-row accumulation tree for (16 × 16) accurate multiplier, applying commutative remapping produces only 6, 4, 4, 3, 3 and 2 rows for 3- to 8-bit SDLC, respectively.

Depending on the concept in (3.5), increased depth of logic cluster can achieve further reduction in the number of product terms, leading to a fewer rows in the accumulation tree. However, the probability of having error from OR compression will increase as will be discussed next.

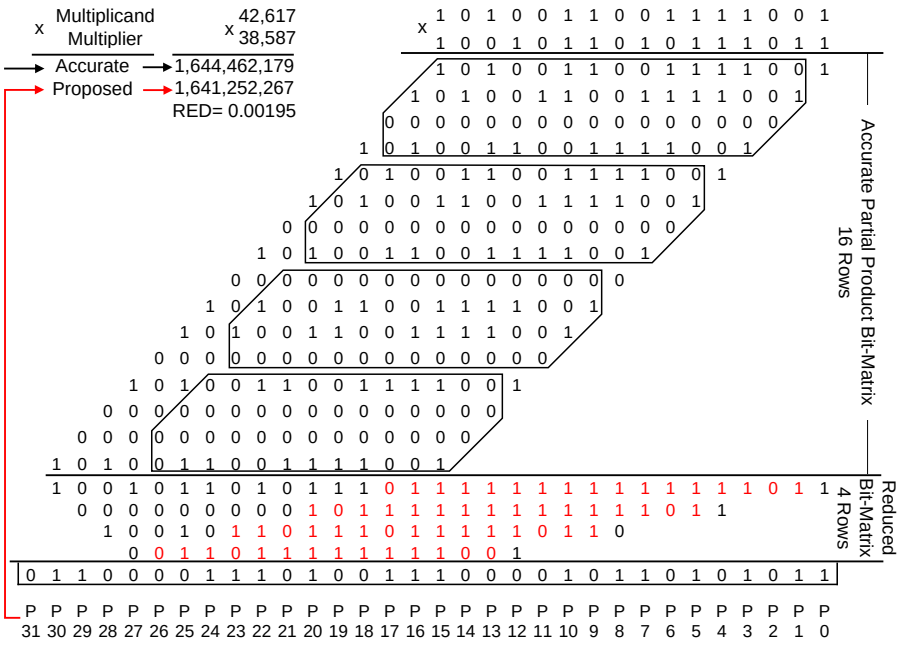


Figure 3.7: Four different sizes of 4-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

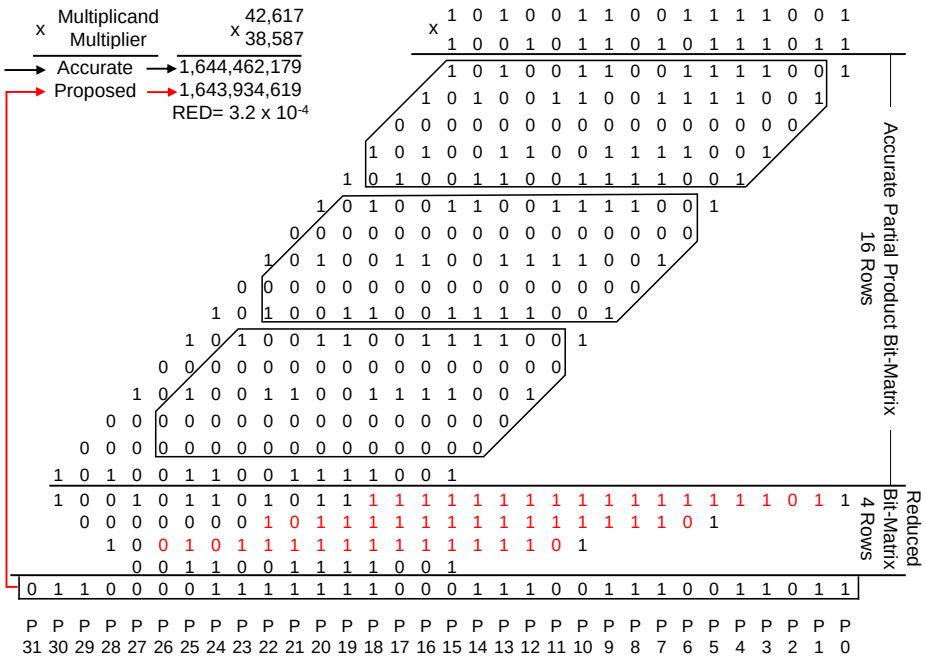


Figure 3.8: Three different sizes of 5-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

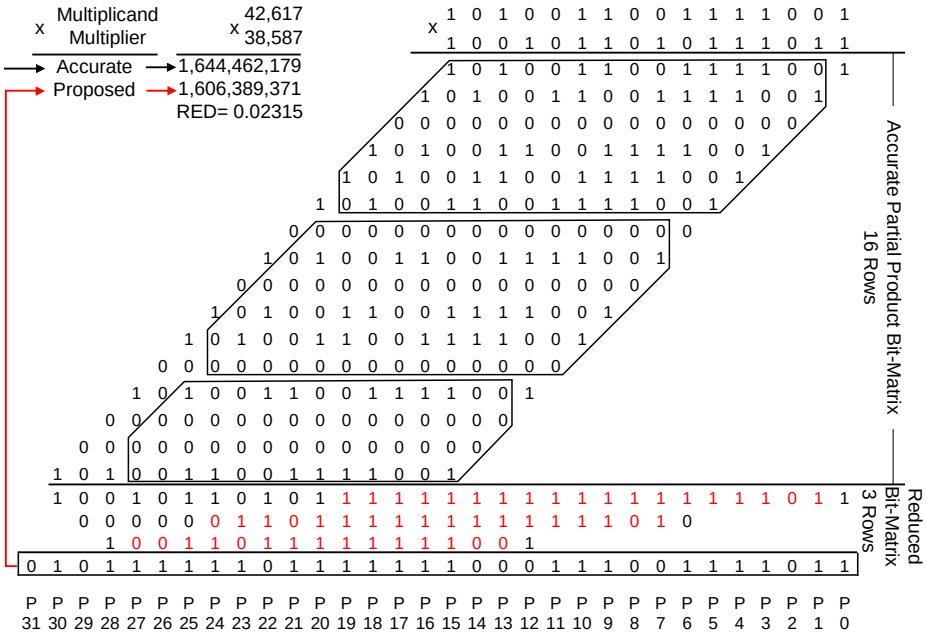


Figure 3.9: Two different sizes of 6-bit combined with 4-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

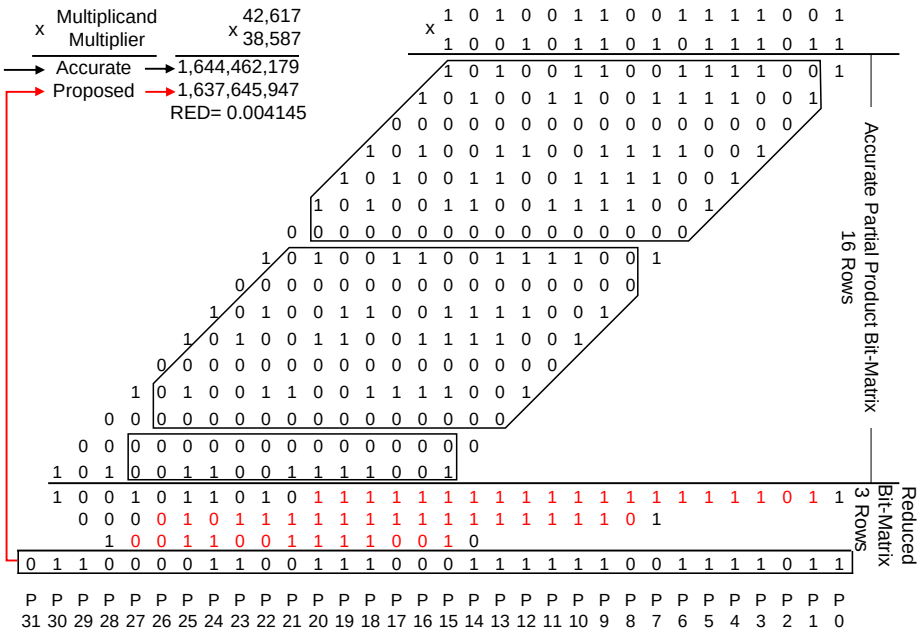


Figure 3.10: Two different sizes of 7-bit combined with 2-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

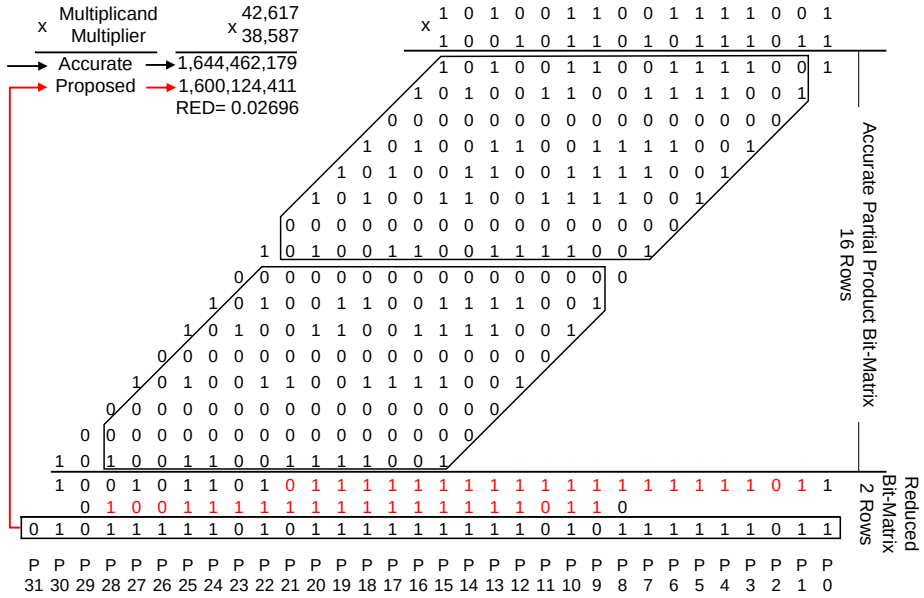


Figure 3.11: Two different sizes of 8-bit logic clusters used to compress partial products based on their progressive bit-significance in (16 × 16) parallel multiplier architecture.

## 3.4 ERROR ANALYSIS

In the the previous sections, for $(N \times N)$ multiplier, generating compressed and remapped PPM using $(d \times L)$ logic clusters has been

discussed. In this section, the impact of the error derived from the proposed SDLC approach, for different sizes of multipliers, is examined. Several error metrics have been discussed in [69] and [72] for evaluating the effectiveness and quantifying errors of approximate adders and multipliers. For any $(N \times N)$ approximate multiplier, the ED is defined as the arithmetic difference between the accurate product $(P)$ and erroneous product $(P')$, *i.e.*, ED $= |P - P'|$. Since the output of the SDLC approach is under-approximated, *i.e.*, the final product derived from the proposed approach is always lower than or equal to the product produced by exact multiplier, ED can be expressed as, $P - P'$. The mean error distance (MED) is defined as the average of the ED values and obtained as:

$$MED = \frac{\sum_{i=0}^{2^{2N}-1} ED_i}{2^{2N}} \quad . \tag{3.12}$$

Also, the mean squared error (MSE) is defined as the average of the squared ED values, which given by:

$$MSE = \frac{\sum_{i=0}^{2^{2N}-1} ED_i^2}{2^{2N}} \quad . \tag{3.13}$$

The relative RED is another useful error metric defined as the ratio of ED over the accurate output, *i.e.*, RED $= \frac{\text{ED}}{P} = \frac{P-P'}{P}$. The error probability (EP) is defined as the ratio of incorrect outputs with respect to the total number of outputs. For any $(N \times N)$ approximate multiplier, the mean relative error distance (MRED), *i.e.*, mean RED, is defined as [69]:

$$MRED = \frac{\sum_{i=0}^{2^{2N}-1} RED_i}{2^{2N}} \quad . \tag{3.14}$$

For comparing multipliers of different sizes, normalized mean relative error distance (NMRED), *i.e.*, normalized MRED, is included in the error analysis of this thesis, and given, using (3.14), by:

$$NMRED = \frac{MRED}{P_{max}} \quad , \tag{3.15}$$

where $P_{max}$ is the maximum product that can be obtained from an $(N \times N)$ accurate multiplier, *i.e.*, $P_{max} = (2^N - 1)^2$. The normalized mean error distance (NMED), *i.e.*, normalized MED, is another metric used to evaluate the impact of errors for different sizes of approximate multipliers, expressed, using (3.12), as [69]:

$$NMED = \frac{MED}{P_{max}} \quad , \tag{3.16}$$

In this work, we include normalized mean squared error (NMSE), *i.e.*, normalized MSE, defined, using (3.13), as:

$$NMSE = \frac{MSE}{(P_{max})(P'_{max})} \simeq \frac{MSE}{P^2_{max}} \quad , \tag{3.17}$$

where $P'_{max}$ is the maximum product that can be obtained from a proposed $(N \times N)$ approximate multiplier. Using NMSE metric for analysing error derived from approximate multipliers has the following advantages: i) avoids bias towards different approximate multipliers that under-approximate and over-approximate, and ii) helps to explain the impact of using approximate multipliers for a set of applications, when the user-experience metrics depends on MSE, such as the peak signal-to-noise ratio (PSNR) (see Section 5.2).

The simulations are performed in Matlab by implementing a functional model of the SDLC approach. The response of all approximate multipliers is evaluated considering all possible combinations of operands if $N < 16$. Monte Carlo approach is used to simulate the functional model of the proposed SDLC if the size of operands

are 16-bit or more, since the simulations run very slow. Using Monte Carlo approach provides different distribution functions to represent all input variables, *i.e.*, multiplicand and multiplier combinations involved in the simulations. For simplicity, the inputs for the multiplier discussed in Algorithm 3.2 are obtained as follows. First, the size of the multiplier $N$ and the depth of logic cluster $d$ are selected, then both operands of multiplicand and multiplier are both assumed to be a random variables with uniform distributions for the all values between 0 and $2^N - 1$. The simulations are repeated for $2^{20}$ input vectors in the case of $N \geq 16$. Note that, the equations (3.12) to (3.14) are applicable when evaluating the proposed SDLC approach for all possible combinations of operands. However, for Monte Carlo simulation, the term $2^{2N}$ in the denominator is replaced by the number of multiplication times performed by each simulation, which is equivalent to $2^{20}$ in this work.

Table 3.1 shows four error metrics using varying sizes of the proposed multiplier in the case of 2-bit logic compression. It can be seen that MRED and NMED fall drastically as the size of the multiplier is increased from 4 to 16-bit. The increasing trend in the error probability (EP) is expected due to the increased bit-width of the multiplier. This is because the error occurrence increases as well due to the growing likelihood of finding a pair of vertically aligned “ones" through two successive rows. In such cases, the corresponding OR gate will return an error, as detailed in Section 3.2.2.

The error probabilities in Table 3.1 can be misleading, as the eventual impact of error is reflected in error distance metrics, such as MRED and NMED [20, 17]. Also, the readings of MAX(RED) would not denote severe degradation of the final output because the occurrence of these errors is rare. This can be seen in Fig. 3.12, which

Table 3.1: Error metrics for varying sizes of proposed multiplier using 2-bit logic cluster.

| Error Metrics | Multiplier Bit-Width (2-bit SDLC) | | | | |
|---|---|---|---|---|---|
| | 4×4 | 6×6 | 8×8 | 12×12 | 16×16 |
| EP(%) | 19.53 | 34.96 | 49.11 | 70.68 | 83.86* |
| Max_ED | 38 | 406 | 3670 | 255318 | 16356728* |
| Max_ED Case | 15×15 | 63×63 | 255×255 | 4095×4095 | 65535×65535 |
| Max_RED | 0.3111 | 0.328 | 0.332 | 0.3332 | 0.3333* |
| Max_RED Case | 15×3 | 63×3 | 255×3 | 4095×3 | 65535×3 |
| MRED | 0.0277 | 0.0266 | 0.0199 | 0.0082 | 0.0029* |
| NMRED | 1.2E-4 | 6.7E-6 | 3.1E-7 | 4.9E-10 | 6.7E-13* |
| MED | 2.375 | 25.375 | 229.375 | 15957.375 | 1041749.375 |
| NMED | 0.0106 | 0.0064 | 0.0035 | 0.001 | 0.0002* |
| MSE | 42.25 | 3543.75 | 2.5E+5 | 1.1E+9 | 4.6E+12* |
| NMSE | 8.3E-4 | 2.2E-4 | 6E-5 | 3.9E-6 | 2.5E-7* |
| SUM _EDs | 608 | 103936 | 1.5E+7 | 2.7E+11 | 1.1E+12* |
| SUM _REDs | 710.02 | 10890.6 | 1.3E+5 | 1.4E+7 | 3026.8* |
| SUM _MSEs | 10816 | 1.5E+7 | 1.6E+10 | 1.8E+16 | 4.8E+18* |

* For $2^{20}$ input vectors.

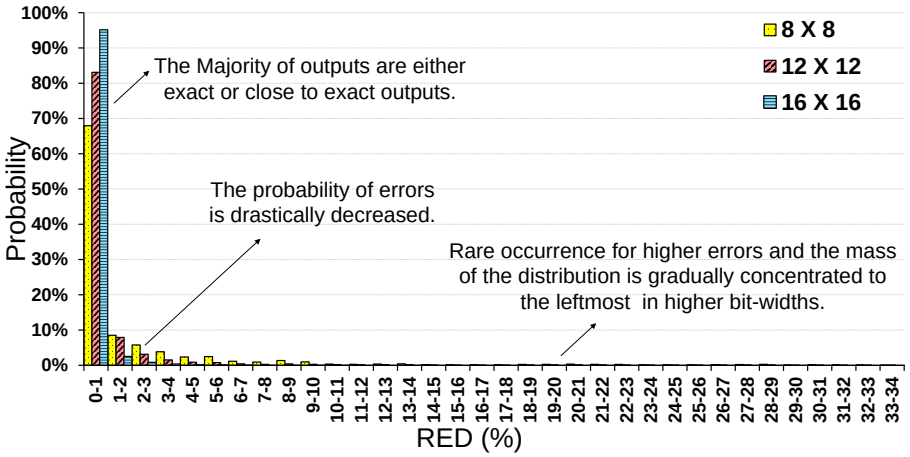


Figure 3.12: Error percentage distribution for 8-, 12- and 16-bit proposed multiplier after applying 2-bit depth compression.

demonstrates the probability distribution for all relative errors resulting from three different sizes of multipliers using the SDLC approach. The probability distribution shows that the proposed approach tends to produce exact or close to exact results. This is seen in the sharp decline of the probability of errors with higher REDs, *e.g.*, the MAX(RED) listed in Table 3.1. Furthermore, as the bit-width of the multiplier is increased, the mass of the distribution is gradually concentrated at a lower error distance. This is because the proposed approach does not sacrifice the precision of the more significant bits when using significance-driven logic compression.

Tables 3.2 and 3.3 depicts the error trade-off with increased degree of compression achieved through higher depths of logic clusters in (8×8) and (16×16) multipliers, respectively. As expected, increased depth of logic compression leads to higher error rates. The error readings of the MRED metric are only marginally higher when increased the depth of logic compression (such as with 8-bit SDLC in (16 × 16) multiplier, and (from 4- to 8-bit) SDLC in (8 × 8) multiplier). This is because the errors associated with reduced number of rows are increased (*e.g.*, the number of rows for 8-bit SDLC in (16 × 16) multiplier is just 2 compared to 16 rows in the accurate PPM). Similar observations can be made in the case of the NMED metric.

For the worst case errors (such as, Max_RED), these errors would not lead to severe degradation in the multiplier output because of their rare occurrence, especially with lower depths of logic compression. Fig. 3.13 shows the cumulative probability distribution for all possible relative errors resulting from (16×16) multiplier for different sizes of logic clusters. The proposed multiplier does not sacrifice the precision of the more significant bits. This is observed in the sharp rise of cumulative probability of errors towards 1,

Table 3.2: Error metrics for different depths of logic compression in the proposed (8 × 8) multiplier.

| SDLC Depth | Error Metrics | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EP (%) | Max_ ED | Max_ ED Case | Max_ RED | Max_ RED Case | MRED | NMRED | MED | NMED | MSE | NMSE | SUM_ EDs | SUM_ REDs | SUM_ MSEs |
| 2-bit | 49.11 | 3670 | 255×255 | 0.3320 | 255×3 | 0.0199 | 3.1E-7 | 229.375 | 0.00353 | 2.5E+5 | 6.0E-5 | 1.5E+7 | 1.3E+3 | 1.6E+10 |
| 3-bit | 64.65 | 7754 | 255×255 | 0.4269 | 255×7 | 0.0410 | 6.3E-7 | 542.94 | 0.00835 | 1.2E+6 | 2.9E-4 | 3.6E+7 | 2.7E+3 | 8.1E+10 |
| 4-bit | 76.28 | 15890 | 255×255 | 0.4648 | 255×15 | 0.0819 | 1.3E-6 | 1383.78 | 0.02128 | 7.2E+6 | 1.7E-3 | 9.1E+7 | 5.4E+3 | 4.7E+11 |
| 5-bit | 77.77 | 15906 | 255×255 | 0.4820 | 255×31 | 0.0781 | 1.2E-6 | 1295.3 | 0.01992 | 6.5E+6 | 1.5E-3 | 8.5E+7 | 5.1E+3 | 4.2E+11 |
| 6-bit | 80.47 | 15938 | 255×255 | 0.4901 | 255×63 | 0.0804 | 1.2E-6 | 1257.76 | 0.01934 | 5.7E+6 | 1.4E-3 | 8.2E+7 | 5.3E+3 | 3.7E+11 |
| 7-bit | 83.67 | 16002 | 255×255 | 0.4941 | 255×127 | 0.0994 | 1.5E-6 | 1557.53 | 0.02395 | 8.3E+6 | 2.0E-3 | 1.0E+8 | 6.5E+3 | 5.4E+11 |
| 8-bit | 87.57 | 32258 | 255×255 | 0.4961 | 255×255 | 0.1537 | 2.4E-6 | 3166.49 | 0.04870 | 3.4E+7 | 7.9E-3 | 2.1E+8 | 1.0E+4 | 2.2E+12 |

Table 3.3: Error metrics for different depths of logic compression in the proposed (16 × 16) multiplier.

| SDLC Depth | Error Metrics | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EP (%) | Max_ ED | Max_ ED Case | Max_ RED | Max_ RED Case | MRED | NMRED | MED | NMED | MSE | NMSE | SUM_ EDs | SUM_ REDs | SUM_ MSEs |
| 2-bit | 83.86 | 1.6E+7 | 49148×65523 | 0.3326 | 16366×60 | 0.0029 | 6.7E-13 | 1.0E+6 | 0.00024 | 4.6E+12 | 2.5E-07 | 1.1E+12 | 3.0E+03 | 4.8E+18 |
| 3-bit | 94.39 | 6.6E+7 | 65533×32744 | 0.4283 | 4095×32309 | 0.0121 | 2.8E-12 | 5.2E+6 | 0.00122 | 1.1E+14 | 5.8E-06 | 5.5E+12 | 1.3E+04 | 1.1E+20 |
| 4-bit | 97.83 | 2.7E+8 | 65512×65494 | 0.4663 | 8190×65488 | 0.0347 | 8.1E-12 | 2.4E+7 | 0.00548 | 2.0E+15 | 1.1E-04 | 2.5E+13 | 3.6E+04 | 2.1E+21 |
| 5-bit | 98.57 | 2.7E+8 | 65520×32765 | 0.4838 | 65533×31 | 0.0412 | 9.6E-12 | 2.5E+7 | 0.00582 | 2.2E+15 | 1.2E-04 | 2.6E+13 | 4.3E+04 | 2.3E+21 |
| 6-bit | 99.08 | 5.3E+8 | 32740×65499 | 0.4902 | 4091×4085 | 0.0583 | 1.4E-11 | 4.8E+7 | 0.01108 | 8.1E+15 | 4.4E-04 | 5.0E+13 | 6.1E+04 | 8.5E+21 |
| 7-bit | 99.31 | 5.3E+8 | 32758×65415 | 0.4955 | 16381×16304 | 0.0579 | 1.3E-11 | 4.3E+7 | 0.01 | 6.4E+15 | 3.4E-04 | 4.5E+13 | 6.1E+04 | 6.7E+21 |
| 8-bit | 99.62 | 1.1E+9 | 65462×65503 | 0.4967 | 8171×65341 | 0.1079 | 2.5E-11 | 1.1E+8 | 0.0247 | 3.7E+16 | 2.0E-03 | 1.1E+14 | 1.1E+05 | 3.9E+22 |

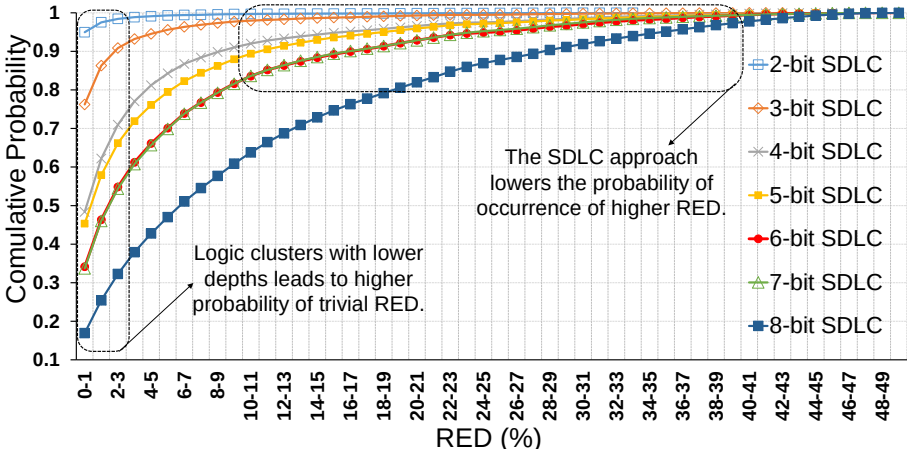


Figure 3.13: Cumulative probability distribution for the error induced by different logic compression levels of the proposed SDLC approach in the case of (16 × 16) multiplier.

especially for lower depth of logic compression, such as 2-, 3- or 4-bit SDLC. The SDLC approach tends to produce results that are closer to the exact outputs when using lower depth of logic clusters. This is because the error occurrence increases as well due to the growing of likelihood of errors returned by higher depth logic cluster. According to (3.6), the probability of having inequality between the output of logic cluster and the accurate arithmetic addition is increased with higher depth logic clusters. This can be observed when the cumulative probability distributions reach to 1 faster than SDLC approach with higher depth of logic clusters, such as 6- to 8-bit SDLC. For example, in the cases of 2-, 3- and 4-bit SDLC, the probability of having errors with less than 1% RED, *i.e.*, RED of 0%-1%, is 0.95, 0.76 to 0.49, respectively. The cumulative probability exceeds 0.9 when the RED is lower than 16% for all logic cluster depths, except the case of 8-bit SDLC, where the likelihood of same RED range is just 75%. Furthermore, the cumulative probability in the cases of 6- and 7-bit logic clusters are almost identical. This can be explained using (3.8), as the number of the last column that can have an OR compression is the same in both cases, leading

to have the same number of more significant bits unaffected by logic compression. The impact of increased degree of compression in the SDLC approach is further investigated in the application case-study in Chapter 5.

## 3.5 DESIGN TRADE-OFFS

To demonstrate the proposed approach, we applied it on eight different sizes of widely known multipliers ranging from 4-bit to 128-bit. Accurate ripple adders were used in both accurate and approximate multipliers to accumulate the partial product rows within the accumulation stage (see Fig. 3.2). We used the first ripple adder to add the first two rows of the PPM, after that the resulted sum is added with the third row using the second ripple adder. By doing so, rows in accumulation tree are sequentially summed up to generate the final product.

A generic SystemVerilog code was used to generate synthesizable modules for all accurate and approximate versions. These modules have been parametrized and configured differently at design time according to the bit-width of multiplier. Run-time reconfigurability of logic cluster using cost vs. degree of approximation (*i.e.* logic compression depth) trade-off is being considered for future work (see Section 6.2). The generated codes were implemented and synthesized using two different off-the-shelf tools: Mentor Graphics Questa Sim was used to compile the SystemVerilog codes and run the associated test benches; and Synopsys Design Compiler was utilized for synthesizing all sizes of accurate and proposed multipliers when mapping the circuits to the Faraday's 90nm technology library [1] and evaluating for power, area, delay and energy in terms of power-delay-product (PDP). For the synthesis process,

1V was set as nominal supply voltage for the given technology library. All designs are synthesized as combinational logic blocks with no clock constraints. This is done to allow the synthesis tool to optimize power and area freely in favour of delay. Note that introducing clock constraints can skew the synthesis tools' optimization algorithm (*e.g.* produce more delay savings at the cost of power/area optimization). However, this does not introduce overall changes between the proposed approximate multipliers and baseline designs.

Fig. 3.14 depicts a comparison of dynamic/leakage power, area, delay and energy trade-offs for all eight sizes of 2-bit SDLC multipliers, when compared with conventional accurate multiplier. As seen, there are significant improvements in all design trade-offs. This is basically because SDLC approach reduces the complexity of multiplier implementation by minimizing the number of rows in the accumulation tree (see Section 3.2.4). This reduction in hardware complexity leads to low switching capacitance and leakage readings, as well as shortened critical paths.

The experiments show reductions in terms of power consumption, run-time and also silicon area used. For dynamic and leakage

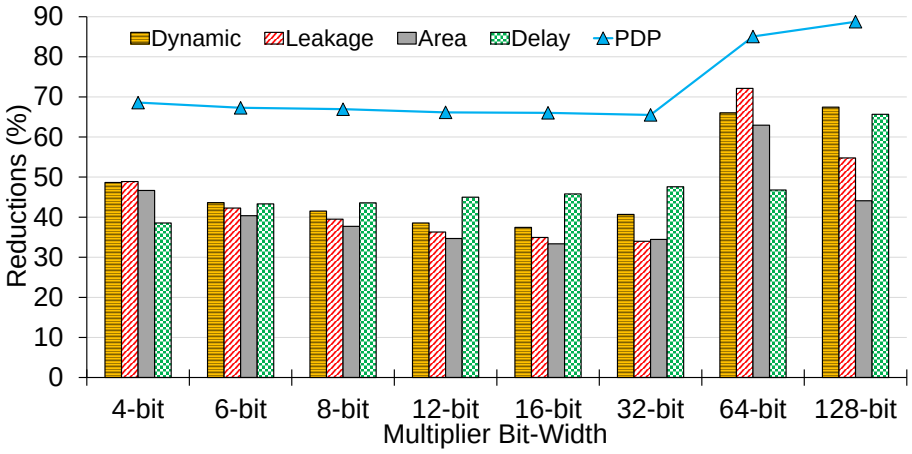


Figure 3.14: Dynamic/leakage power, area, delay and PDP trade-offs for different bit-widths of the (2-bit SDLC) proposed multiplier.

power, the reductions obtained from applying the SDLC approach range from 37.5%-67.4% and 34%-72.1%, respectively, for multiplier bit-widths from 4-bit to 128-bit. The savings in the latency for the same sizes of the proposed multiplier is from 38.5%-65.6%. The reduction in complexity also leads to silicon area to be minimized to 33.4%-62.9%, and energy consumed is substantially reduced in terms of PDP by 65.5%-88.7%. For all bit-widths of the proposed multiplier, the absolute readings of the synthesized designs can be obtained by combining the reduction percentages in Fig. 3.14 with the absolute readings of the accurate multiplier in Table 3.4 and 3.5. For instance, in the case of 32-bit proposed multiplier, the PDP readings of 5.9 pJ is derived using (dynamic, leakage) power consumptions of (1093.2 uW, 38.3 uW), respectively and propagation delay of 5.19 ns, while the silicon area is 16949.3 $um^2$. The non-linear trend of the bars in some cases is attributed to the inconsistency of the ratio of the array of additions in the accumulation tree between the approximate and the accurate multiplier.

Table 3.4: Design trade-offs for different bit-widths of the accurate multiplier used to obtain comparative analysis in Fig. 3.14.

| Bit-Width | Power | | Area ($um^2$) | Delay (ns) | PDP (pJ) |
|---|---|---|---|---|---|
| | Dynamic (uW) | Leakage (uW) | | | |
| 4×4 | 6.36 | 0.65 | 292.43 | 0.96 | 0.007 |
| 6×6 | 21.06 | 1.67 | 740.10 | 1.57 | 0.036 |
| 8×8 | 46.97 | 3.16 | 1388.46 | 2.18 | 0.109 |
| 12×12 | 140.73 | 7.53 | 3287.31 | 3.4 | 0.504 |
| 16×16 | 299.97 | 13.77 | 5988.98 | 4.63 | 1.453 |
| 32×32 | 1843.00 | 57.97 | 25856.32 | 9.9 | 18.820 |
| 64×64 | 17563.60 | 566.91 | 194194.45 | 18.6 | 337.228 |
| 128×128 | 146159.00 | 2684.90 | 832734.22 | 62.35 | 9280.417 |

Table 3.5: Design trade-offs for different bit-widths of the proposed multiplier used to obtain comparative analysis in Fig. 3.14.

| Bit-Width | Power | | Area ($um^2$) | Delay (ns) | PDP (pJ) |
|---|---|---|---|---|---|
| | Dynamic (uW) | Leakage (uW) | | | |
| 4×4 | 3.27 | 0.33 | 156.02 | 0.59 | 0.002 |
| 6×6 | 11.87 | 0.97 | 441.39 | 0.89 | 0.011 |
| 8×8 | 27.46 | 1.91 | 864.75 | 1.23 | 0.036 |
| 12×12 | 86.49 | 4.79 | 2147.38 | 1.87 | 0.171 |
| 16×16 | 187.61 | 8.95 | 3991.34 | 2.51 | 0.493 |
| 32×32 | 1093.20 | 38.27 | 16949.30 | 5.19 | 5.872 |
| 64×64 | 5968.90 | 158.11 | 71985.31 | 9.9 | 60.657 |
| 128×128 | 47587.80 | 1214.50 | 465525.09 | 21.42 | 1045.345 |

Table 3.6 lists the number of the logic cells utilized to build all sizes of the conventional and proposed multiplier designs. As expected, the SDLC approach can be employed to minimize number of cells required to implement all sizes of the proposed multiplier.

Table 3.6: Number of library cells instantiated to form different bit-widths of the (2-bit SDLC) proposed multiplier.

| | Multiplier Bit-Width | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Number of Cells | 4 × 4 | 6 × 6 | 8 × 8 | 12 × 12 | 16 × 16 | 32 × 32 | 64 × 64 | 128 × 128 |
| Accurate | 56 | 132 | 240 | 552 | 992 | 4258 | 27261 | 102205 |
| Proposed (2-bit SDLC) | 30 | 76 | 142 | 342 | 626 | 2605 | 10704 | 76884 |
| Reduction (%) | 46.4 | 42.4 | 40.8 | 38.0 | 36.9 | 38.8 | 60.7 | 24.8 |

The proposed multiplier utilizes different sizes of logic clusters. Each logic cluster contains an array of OR gates used to compress a set of product terms. The logic clusters have no carry propagation and can also work in parallel. Hence, the delay required to generate the compressed partial product matrix is: 2-input AND gate delay (for parallel forming $N^2$ product terms) + $d$-input OR gate delay (for parallel logic compression). Then, the compressed partial product matrix is ready to be accumulated by applying any convenient scheme of multiplication, as shown in Fig. 3.15. Generally, the critical delay of the proposed approach depends on: (i) the size of multiplier (*i.e.* higher bit-widths increase the number of product terms and therefore the propagation delay associated with accumulation tree is also increased), and (ii) the level of logic

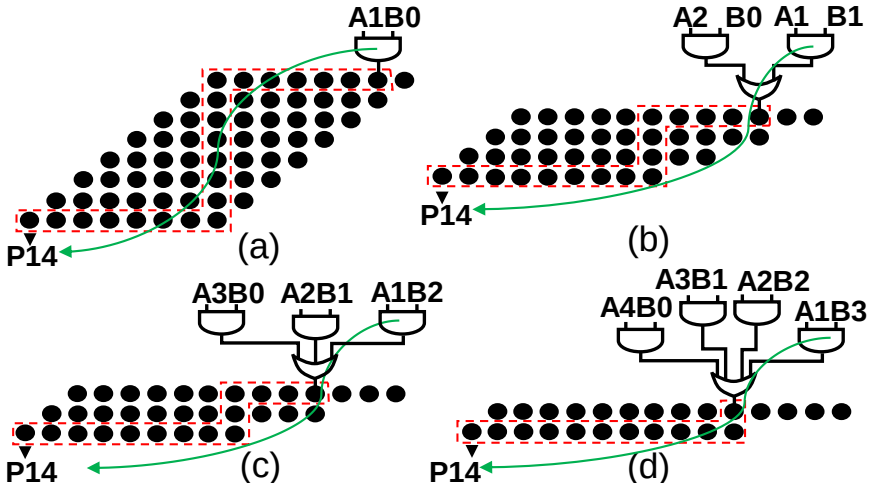


Figure 3.15: Dot diagram highlights the impact of increasing depth of logic clusters on the critical path of $(8 \times 8)$ multiplier: (a) partial product bit-matrix of the conventional multiplier; (b) after 2-bit SDLC; (c) 3-bit SDLC; and (d) 4-bit SDLC. The dotted polygons indicate the maximum propagation path for summing up the accumulation tree. Higher degrees of compression minimize the propagation delay associated with accumulation tree (such as (b) and (c)), while a further reduction in (d), since a carry propagation adder is just needed to generate the product (no extra delay required for accumulation tree). The curved lines identify the critical paths for each multiplier (from A1 to P14).

compression (*i.e.* increased depth of the logic clusters leads to lower number of rows in accumulation tree and also lower height of the critical column). Note that, the critical path of the entire design depends also on the method of summing up the accumulation tree returned from the SDLC approach, such as carry-save array or Wallace method [92]. According to the experimental work described at the beginning of this section, the critical delay of $(N \times N)$ proposed multiplier is identified when any change in the input A(1) resulting in a change in the output of P($2N-2$) of the final product. See Fig. 3.15 as an example illustration of critical path in the case of $(8 \times 8)$ multiplier.

Fig. 3.16 illustrates the dynamic/leakage power, delay, area and energy trade-offs with increased degree of logic compression. Higher depth of clustering achieves considerable savings in all design trade-offs since by increasing the depth of logic clusters, the hardware complexity associated with lower numbers of product rows is also decreased (see Section 3.3).

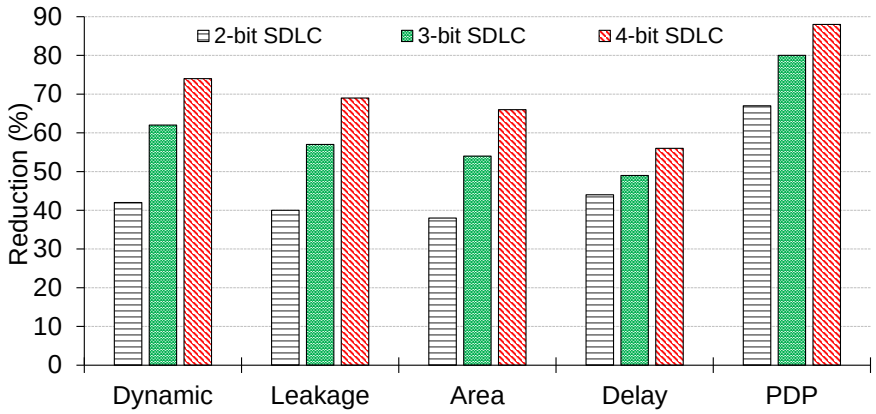


Figure 3.16: Dynamic power, leakage power, delay, area and energy trade-offs for different degrees of logic compression of $(8 \times 8)$ multiplier.

To demonstrate the energy-quality trade-offs for different sizes of proposed multiplier, Fig. 3.17 shows the relation between the MRED and PDP for different levels of logic compression in the case

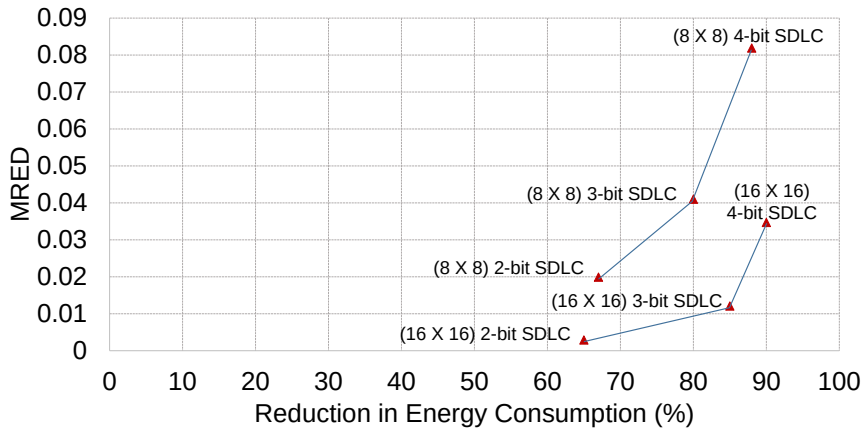


Figure 3.17: The MRED and PDP trade-offs for different degrees of logic compression of (8 × 8) and (16 × 16) multipliers.

of (8 × 8) and (16 × 16) multipliers. Additional reduction in energy consumption can be achieved with increased level of logic compression, which is done at the cost of growing errors measured by MRED metric. For instance, the reduction in energy consumption increases from 65%, 85% up to 90%, with increased logic cluster depth of (16 × 16) multiplier from 2-, 3- to 4-bit, respectively. The absolute MRED readings can be obtained from Tables 3.2 and 3.3. The increased bit-width of proposed multiplier can mitigate the impact of the lossy compression done by logic clusters. For example, applying 4-bit logic cluster can translate into a substantial reduction in PDP readings (about 90% for both (8 × 8) and (16 × 16) multipliers), while the MRED is reduced to the half in the case of (16 × 16) multiplier.

## 3.6 COMPARATIVE ANALYSIS

Fig. 3.18 shows comparative area, power, delay and energy advantages of our approach (with 2-bit SDLC) for different bit-widths. The comparisons are carried out with the following two existing approaches: Kulkarni [65] and error-tolerant multiplier (ETM) [66],

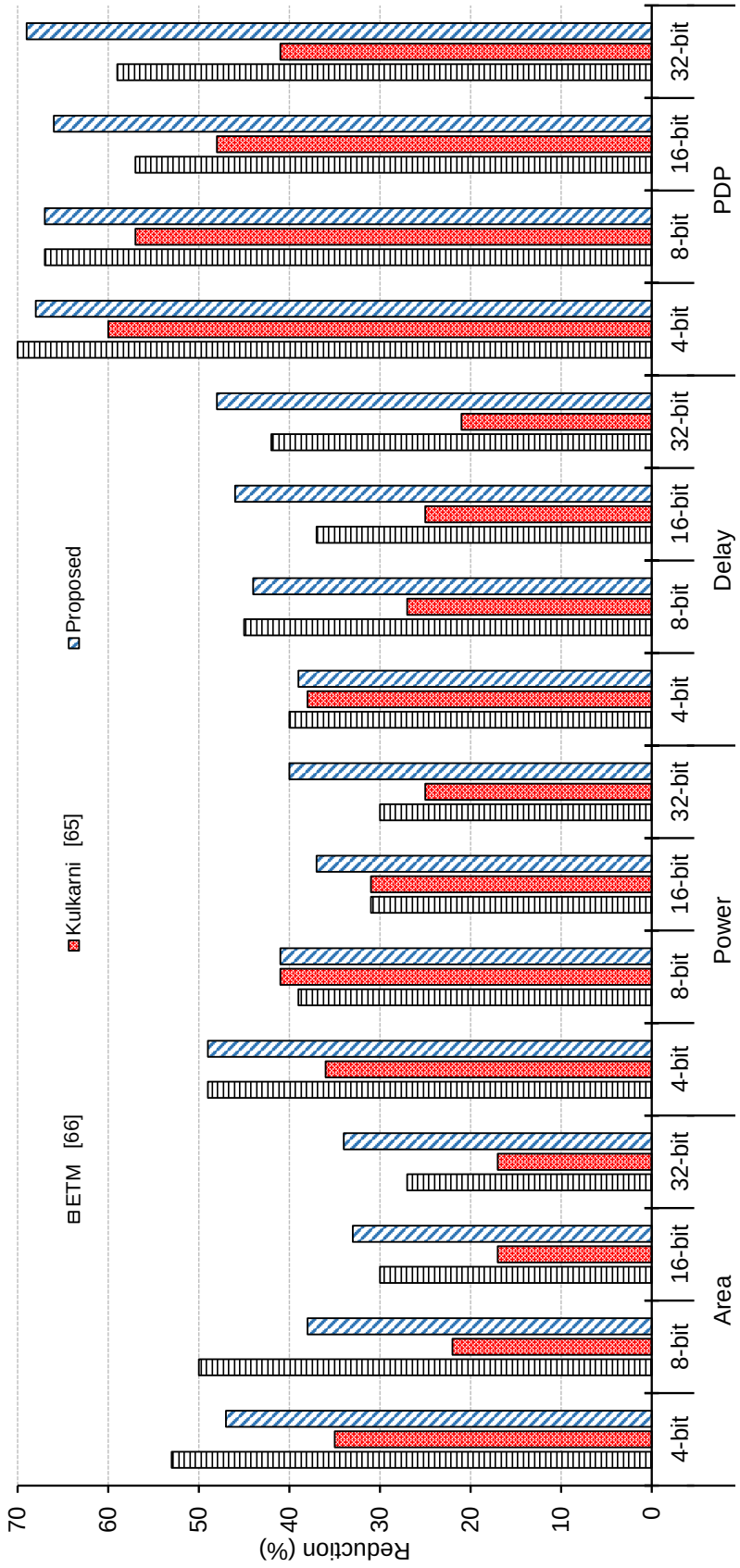


Figure 3.18: Area and power trade-offs for various scalable approximate multipliers.

chosen for their direct relevance to our work. In the Kulkarni [65] approach, a large multiplier is produced using small $(2 \times 2)$ approximate units as building blocks. The design approach of $(N \times N)$ ETM [66] follows truncation principles by dividing the multiplier into accurate and approximate parts. Similar to $N$-bit fixed-width multiplier, only the most significant $N$ bits of the final product are generated using accurate multiplications, whereas in the approximate part, a probabilistic bit manipulation is used to generate the least significant $N$ bits of the product terms.

The proposed approach produces better results as the bit-width of the multiplier is increased. This is seen with the 16- and 32-bit multiplier. For example, in the case of 32-bit approximate multiplier, the (area and power) savings obtained from both ETM [66] and Kulkarni [65] are (27.2% and 30.1%) and (17.1% and 25.2%), respectively, the area and power trade-offs of the proposed multiplier is (34.5% and 40.5%), while the (delay and PDP) savings are (41.6% and 59.2%), (21.2% and 41.1%) and (47.6% and 68.8%) for ETM, Kulkarni and proposed, respectively. In such a case, the proposed approach outperforms both approaches in terms of power, area, delay and PDP savings. This is expected as the number of product rows is halved (with 2-bit clustering) and commutative remapping is used to reduce the parallel accumulation complexity. Noting that, applying higher depths of logic cluster will further increase the design gains of the propose multiplier, thereby making the proposed multiplier outperform even for lower bit-widths, such as $(8 \times 8)$ multiplier (see Fig. 3.16).

The corresponding error comparisons of these approaches are shown Fig. 3.19, demonstrating comparative errors (in terms of MRED, NMED and also ER) using the proposed $(8 \times 8)$ multiplier (with 2-bit SDLC). As expected, our approach outperforms both

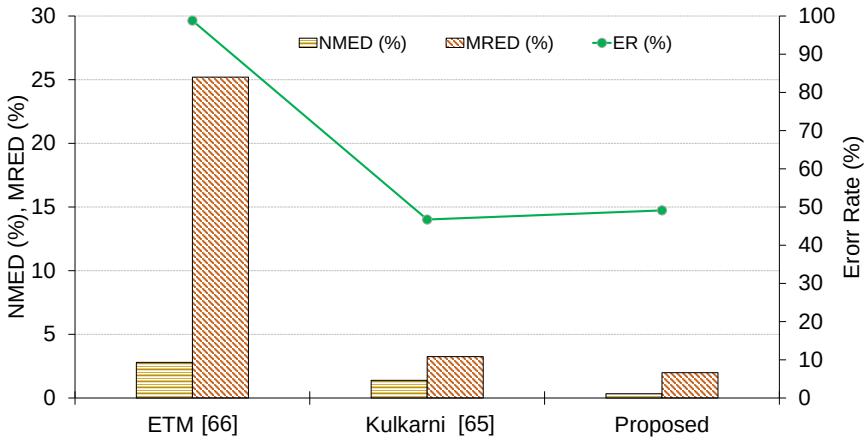


Figure 3.19: Comparative errors in terms of MRED, NMED and also ER for various scalable (8 × 8) approximate multipliers.

approaches in terms of MRED, NMED due to its bit significance-driven logic compression. For example, the MRED equals 0.252 and 0.139 for the cases of the (8 × 8) multiplier proposed by ETM [66] and Kulkarni [65], respectively, whereas, the MRED of the proposed multiplier is just 0.0199. As the proposed approach progressively preserves the high-order bits, it is expected to exhibit significantly lower errors for multipliers with higher bit-widths.

## 3.7 SIGNED MULTIPLICATION USING SDLC

The SDLC approach minimizes the product terms in favour of reducing the number of rows in partial product matrix. Different degrees of logic compression have been exploited to provide comparative design advantages at low loss of accuracy. In this section, we discuss the design and implementation requirements of the proposed approach when performing signed multiplication. The proposed approach is feasible for the case of signed operands; however, some modifications are needed. These modifications can be described in the following two ways:

(i) To exploit the proposed approach for signed multipliers, extra interface circuitry is required to support sign-magnitude complement representation. In this context, only absolute values of the input operands can be considered in the multiplication. The magnitude of the final product will then be computed using ($N$-1)×($N$-1) proposed multiplier to generate (2$N$-2)-bit result. This result is considered as the absolute value of the final product, while the sign of the final product will be generated using single 2-input XOR gate. However, finding the exact absolute value of the input operands may degrade the speed of calculation; hence, we can produce it in an efficient way similar to the method proposed in [132]. Fig. 3.20 illustrates the hardware implementation related to proposed signed multiplier. The sign of the input operands are determined first. Thus, for each operand with negative value, the absolute value is produced. Then, the proposed SDLC approach can be performed to multiply the unsigned operands as discussed in Sections 3.3 and 3.2. Noting that, in the case of using $N \times N$ proposed multiplier, the most significant bit of the absolute value of an $N$-bit number is 0. Also, the $(2N-1)^{th}$ and $(2N-2)^{th}$ bits of the 2$N$-bit final product are affected by the signal generated from the sign detector.

(ii) In the case of 2's complement multiplication, alterations to the proposed design should be considered to support sign exten-

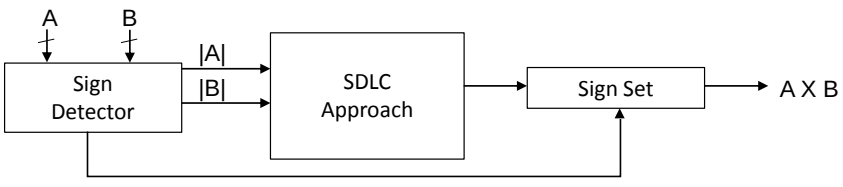


Figure 3.20: Block diagram for one of the options of hardware implementation required to implement the proposed signed multiplier (inspired from [132]).

sion, such as conventional 2's complement array multiplication, in which a circuit unit should be added to ensure correct sign of the final product. An alternative approach is to create reorganized partial-product array, such as in Baugh-Wooley multiplication [5]. Fig. 3.21 shows the partial product matrix in the case of $(8 \times 8)$ signed multiplier. Some of the product terms indicated as $q_{i,j}$ (highlighted in blue), are obtained as the NAND of the operands bits $A_i$ and $B_j$. Assuming that the input operands of the multiplier are uniformly and independently distributed, the probability that a complemented partial product is high is given by:

$$P\left(q_{i,j}\right) = P\left(A_i \uparrow B_j\right) = \frac{3}{4} \qquad >> \qquad P\left(A_i \wedge B_j\right) = \frac{1}{4} \tag{3.18}$$

Equation (3.18) shows that using NAND instead of AND gate to generate complemented partial products increases the probability of having 1's in the partial product matrix. Therefore, the error probability associated with logic cluster having a complemented product terms, can also increase as described in (3.3). In such case, the likelihood of having two 1s or more at the inputs of the OR gate will lead to an error, *i.e.*, when the output of OR gate is not equal the arithmetic sum of the input bits. However, the increase in the error probability will result in slight degradation in the error magnitude (*e.g.*, in terms of the mean error). To explain, the

| | | | | | | | X | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| | | | | | | | 1 | A7B0 | A6B0 | A5B0 | A4B0 | A3B0 | A2B0 | A1B0 | A0B0 |
| | | | | | | | A7B1 | A6B1 | A5B1 | A4B1 | A3B1 | A2B1 | A1B1 | A0B1 | |
| | | | | | | A7B2 | A6B2 | A5B2 | A4B2 | A3B2 | A2B2 | A1B2 | A0B2 | | |
| | | | | | A7B3 | A6B3 | A5B3 | A4B3 | A3B3 | A2B3 | A1B3 | A0B3 | | | |
| | | | | A7B4 | A6B4 | A5B4 | A4B4 | A3B4 | A2B4 | A1B4 | A0B4 | | | | |
| | | | A7B5 | A6B5 | A5B5 | A4B5 | A3B5 | A2B5 | A1B5 | A0B5 | | | | | |
| | | A7B6 | A6B6 | A5B6 | A4B6 | A3B6 | A2B6 | A1B6 | A0B6 | | | | | | |
| 1 | A7B7 | A6B7 | A5B7 | A4B7 | A3B7 | A2B7 | A1B7 | A0B7 | | | | | | | |

Figure 3.21: Partial product matrix of $(8 \times 8)$ signed multiplier. Complemented partial products are highlighted in blue.

complemented bits are concentrated at the most significant part of the partial product terms. As such, the proposed SDLC approach progressively preserves the high-order bits of the partial products. This allows for the complemented bits to be accumulated on a carry-propagation basis as in accurate multiplier.

For the above-mentioned ways, the use of the signed multiplication does not reveal any significant degradation of the results obtained in the previous sections. Note that, exploiting Booth's algorithm [11] may also improve the performance by generating lower number of partial products using parallel Booth encoders. Generally, utilizing conventional Booth's encoding in parallel multiplier implementations can skip addition and perform just shift, when the partial product of '0' occurs. When combining Booth with proposed multiplier design, one may execute the Booth encoders in parallel prior to applying the logic compression. However, conventional Booth-encoding is inconvenient when designing a parallel multiplier, such as the proposed design, for two reasons [10]; the first is due to variable number of add/subtract operations and the second is because of long delays with isolated 1's, *e.g.* bits 001010101(0) recoded as 011111111, requiring 8 instead of 4 partial products. To address this limitation and also to improve the performance, modified Booth-encoding (MBE) scheme, such as Radix-4 [130], can be used to generate a fixed and reduced number of partial products using shifting and complement operations.
Note that, the advantage of implementing higher radix Booth encoders to generate a reduced number of partial products comes at the expense of increased hardware complexity [10]. However, the proposed multiplier can do the same task (*i.e.* reducing the number of partial products at low loss of accuracy) without using Booth schemes. Moreover, the double compression caused by

combining higher radix Booth encoders with existing logic cluster can lead to a major impact on the accuracy. Thus, compared to the above-mentioned ways, exploiting Booth-encoding along with the proposed SDLC approach may not be as feasible.

## 3.8 CONCLUDING REMARKS

In this chapter, a novel approximate multiplier design approach is proposed using significance-driven logic compression (SDLC). This design approach utilizes an algorithmic and configurable lossy compression based on bit significance to form a reduced set of partial product terms. This is then reorganized and accumulated using various schemes of parallel multiplication. On a statistical basis, the results of such as, NMED and MRED metrics show how the impact of error is mitigated when the size of the multiplier is increased. Additionally, the mass of the error distributions are gradually concentrated at a lower error distance, indicating that the proposed multiplier gives close to exact products for most inputs with positive bias. The results obtained after synthesis have shown a substantial decrease in run-time, power consumption and even in silicon area. Also, the possibility to perform SDLC approach in signed multiplication is addressed. Performance-energy-quality trade-offs are demonstrated for different levels of approximations achieved through configurable logic clustering.

# 4

# ERROR MITIGATION IN LOGIC COMPRESSION

## 4.1 INTRODUCTION

Chapter 3 showed a novel SDLC approach for approximate multiplier design. The SDLC approach can offer various degrees of freedom to increase performance and efficiency. This is done by generating different sizes of reduced accumulation tree depending on the depth of logic clusters and bit significance. The hardware complexity of standard multiplier schemes is dominated by the number of product terms in accumulation tree. Thus, the proposed SDLC approach has the potential to benefit various schemes of multiplier designs, such as *carry-save array*, *Wallace* [128] and *Dadda-tree* [23], when relaxing the accuracy requirements.

In this chapter, we introduce a novel multiplier design of combining a Wallace-tree accumulation method together with the SDLC approach. The lossy logic compression performed by SDLC approach works for reducing the number of product rows using progressive bit significance, and thereby decreasing the number of reduction stages in Wallace-tree accumulation. This accounts for substantially lower number of logic counts and lengths of the critical paths at the cost of errors in lower significant bits. These errors are minimised through a parallel error detection logic and compensation vector. To evaluate the effectiveness of our approach, multiple 8- and 16-bit multipliers are designed and synthesized using Syn-

opses Design Compiler with different logic compression levels. Also, extensive error analysis is provided.

## 4.2 PROPOSED APPROXIMATE WALLACE MULTIPLIER

In this section, we introduce an approximate multiplier design by incorporating a Wallace-tree accumulation with hardware-based logic compression techniques. The proposed approach consists of two major steps. First, systematic lossy compression is carried out using SDLC approach to generate a reduced number of partial product rows (discussed in Chapter 3). Second, Wallace method is applied to reduce the number of these rows to the height of two to be then combined using a carry propagating adder. These steps together with different logic compression levels, are described below.

### 4.2.1 *Logic Compression using SDLC*

The proposed SDLC approach generates all partial products in an $(N \times N)$ multiplier using $N^2$ AND gates, similar to accurate multiplier. Then various sizes of logic clusters are utilized to compress a group of vertically-aligned bits within a group of successive partial products based on their progressive bit significance. This results in a reduced number of product terms in the partial product bit-matrix (see Section 3.3). After that a commutative remapping technique is used to reduce the number of rows in the bit-matrix (see Section 3.2.4). This can be leveraged to decrease the number of reduction stages in the Wallace accumulation tree.

### 4.2.2 *Accumulation with Wallace Method*

One of the well-known fast multipliers is the column compression multiplier presented by Wallace [128]. This multiplier consists of three consecutive phases: partial product formation, accumulation, and CPA. In general, ($N \times N$) traditional Wallace multiplier begins to group $N$ rows together in sets of three rows each. Any additional rows that are not a member of a group of three are transferred to the next level without modification. Within each group of three rows, (3, 2) counters (full adders) are applied to the columns containing three bits and (2, 2) counters (half adders) are applied to the columns containing two bits. Columns containing only a single bit are transferred to the next level unchanged. By doing so, the partial product bit-matrix is then column-wise accumulated to a height of two. These two rows are combined in the last phase using a CPA.

Figure 4.1 shows these phases in the case of an accurate ($16 \times 16$) Wallace multiplier. As can be seen, the multiplier begins generating 256 product terms organized in 16-row Wallace-tree. Six reduction stages are required to compress these rows to the height of 2 rows. In stage 1, the number of rows in the accumulation tree reduces from form 16 rows to 11 rows, then stage 2 from 11 rows to 8 rows and after that from 8 to 6 and from 6 to 4 rows in stages 3 and 4 and finally, from 4 to 3 and from 3 to 2 rows in stages 5 and 6. This can be achieved by using total of 200 (3, 2) counters and 52 (2, 2) counters. Then, the resulting 2 rows can be summed up using a 24-bit CPA.

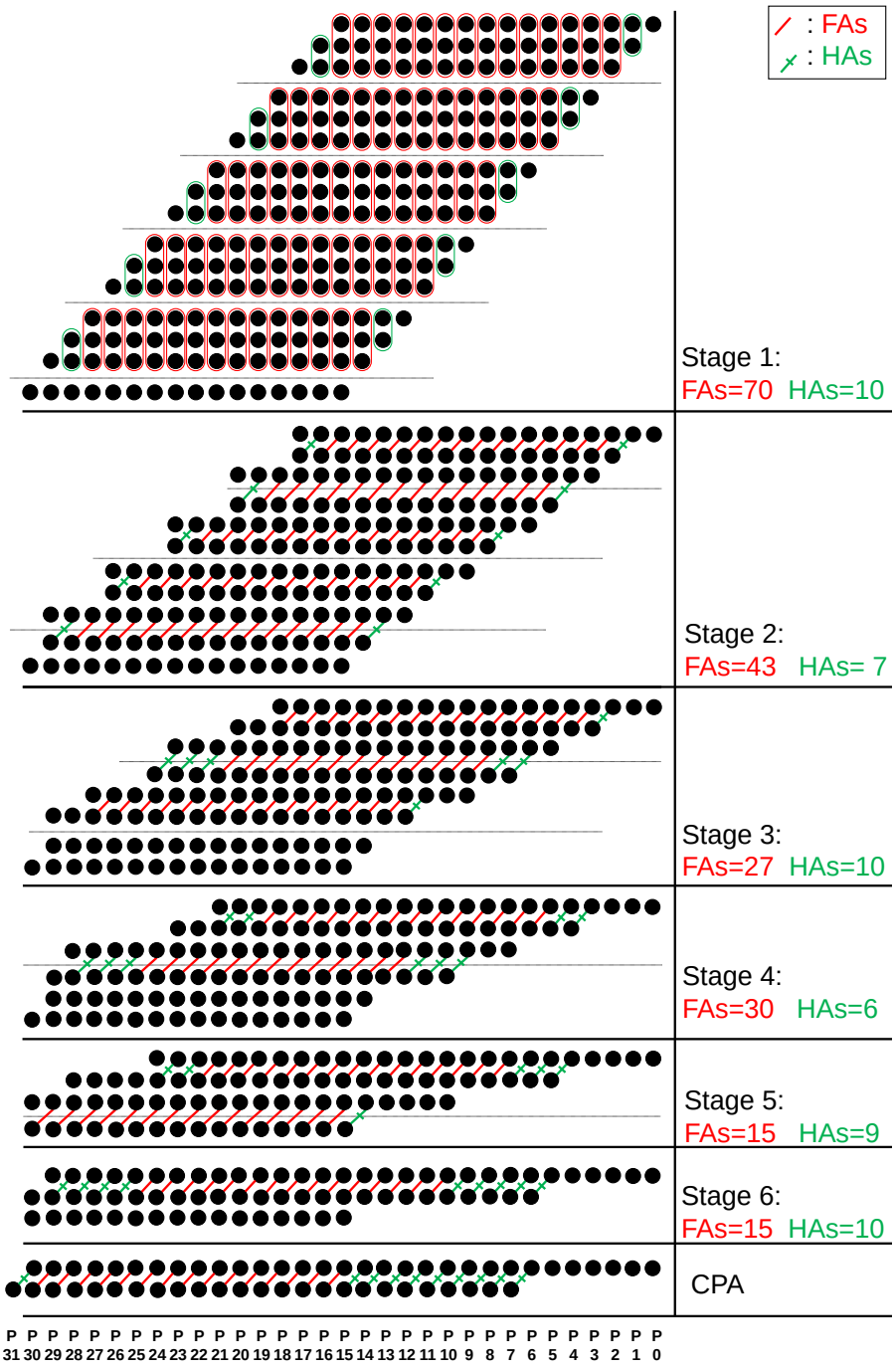


Figure 4.1: Reduction stages and logic cell counts for (16 × 16) accurate Wallace.

### 4.2.3 *Wallace with Variable Logic Compression*

Since the hardware complexity of Wallace multiplier depends on the number of partial product rows in accumulation tree [121],

SDLC approach can be utilized to reduce the height of the rows in partial product matrix. As such, the number of the reduction stages in Wallace accumulation method is deterministically reduced. Figures 4.2 to 4.8 illustrate the impact of using different degrees of logic compression on the number of reduction stages required by Wallace accumulation method. The number of reduction stages is reduced with increased level of logic compression. Thus, the hardware complexity in terms of (3, 2) and (2, 2) counters is substantially decreased. For instance, instead of six reduction stages in the case of accurate (16 × 16) multiplier, the reduced number of rows associated with 2-bit SDLC needs just four reduction stages. Figure 4.2 shows the matrix heights of 8, 6, 4, 3 rows, this requires total number of 107 (3, 2) counters and 28 (2, 2) counters. These numbers are further decreased to only 69 (3, 2) counters and 23 (2, 2) counters for 3-bit logic clusters with only three reduction stages

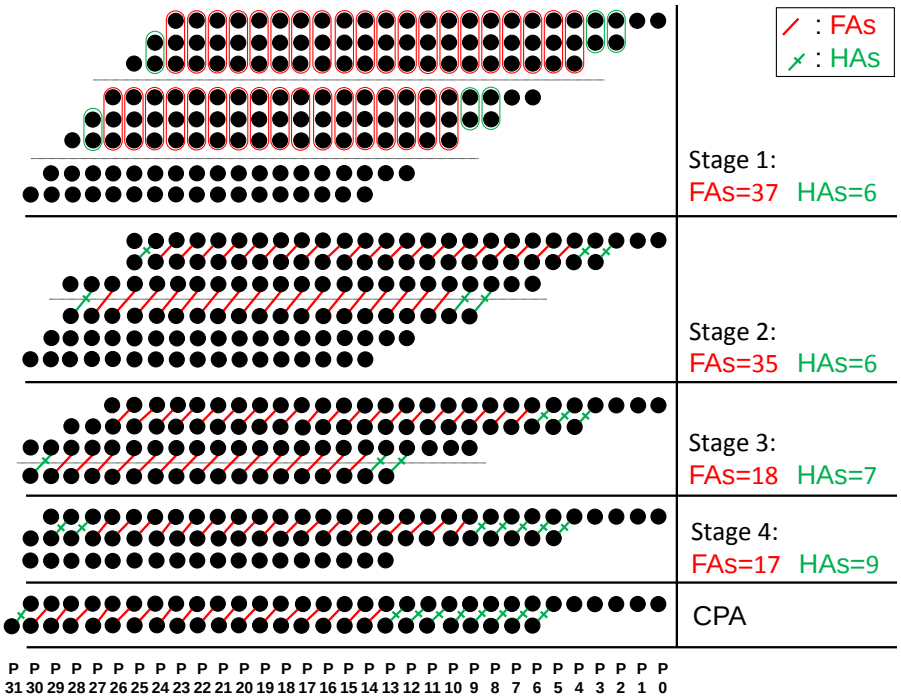


Figure 4.2: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 2-bit SDLC with Wallace-tree accumulation (2-bit SDLC Wallace).

(see Figure 4.3). The height of Wallace tree is minimized from 4 to 3 and from 3 to 2 within two reduction stages when utilizing 4- and 5-bit SDLC (as shown in Figures 4.4 and 4.5). It is only one reduction stage when it comes to accumulate the 3-row partial product matrix resulted from 6- and 7-bit SDLC approach(see Figures 4.6 and 4.7). No reduction stages are needed in the case of 8-bit SDLC, since the height of the accumulation tree is just 2 and the final product is ready to be computed using CPA (see Figure 4.8).

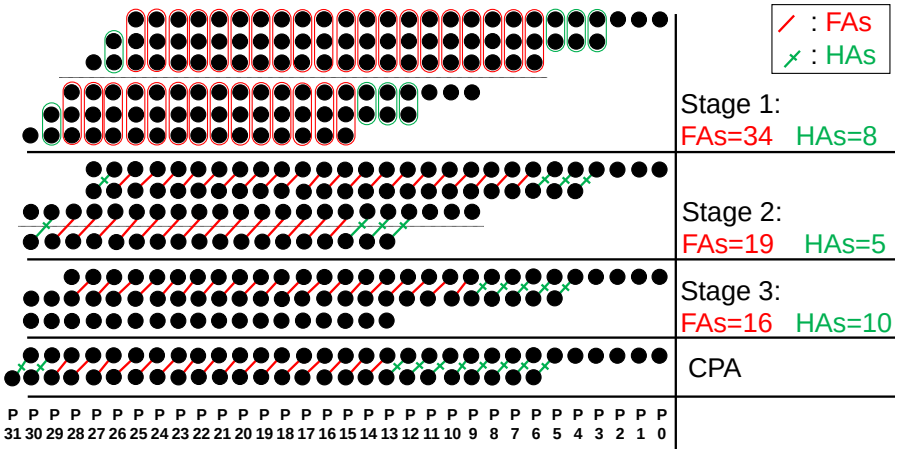


Figure 4.3: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 3-bit SDLC with Wallace-tree accumulation (3-bit SDLC Wallace).

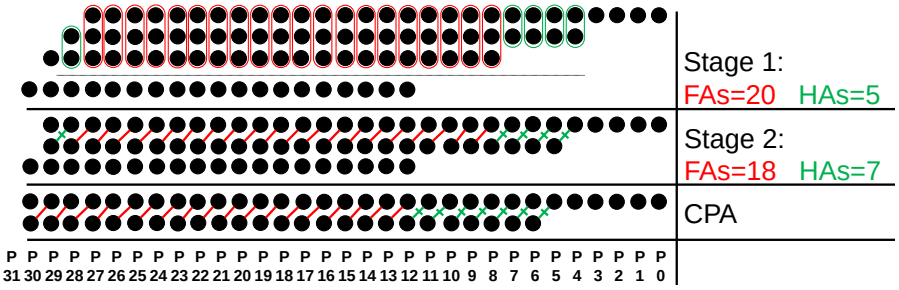


Figure 4.4: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 4-bit SDLC with Wallace-tree accumulation (4-bit SDLC Wallace).

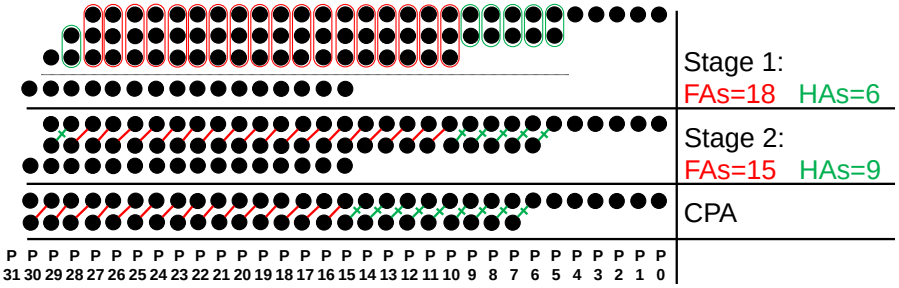


Figure 4.5: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 5-bit SDLC with Wallace-tree accumulation (5-bit SDLC Wallace).

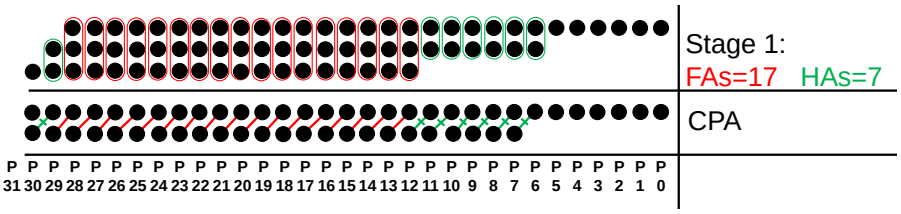


Figure 4.6: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 6-bit SDLC with Wallace-tree accumulation (6-bit SDLC Wallace).

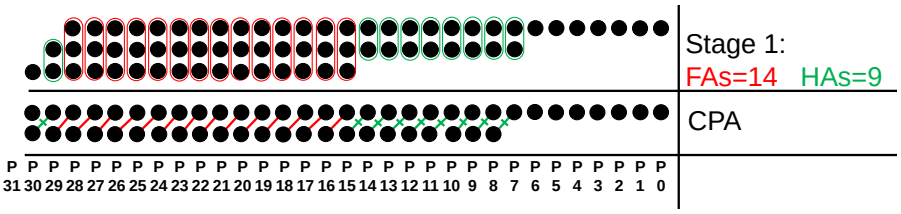


Figure 4.7: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 7-bit SDLC with Wallace-tree accumulation (7-bit SDLC Wallace).

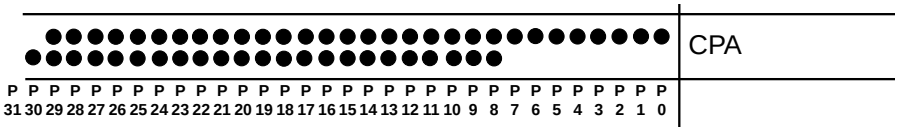


Figure 4.8: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating 8-bit SDLC with Wallace-tree accumulation (8-bit SDLC Wallace).

For an $(N \times N)$ Wallace-tree multiplier, the height of the matrix in the $k^{th}$ reduction stage, $\alpha_k$ is given by the following recursive equations:

$$\alpha_0 = \begin{cases} N, & \text{for traditional Wallace-tree multiplier} \\ \lceil \frac{N}{d} \rceil, & \text{for SDLC with } d\text{-bit compression level} \end{cases}$$

$$\alpha_{k+1} = 2 \cdot \left\lfloor \frac{\alpha_k}{3} \right\rfloor + \alpha_k \ mod\ 3\ . \tag{4.1}$$

The reduction in hardware complexity achieved by SDLC leads to low switching capacitance and leakage reading as well as shortened critical paths (see Section 4.5). Furthermore, different depths of logic compression can support the Wallace-tree multiplier with different energy-accuracy trade-offs (see Section 4.4).

Table 4.1 summarizes the impact of combining the SDLC approach with Wallace method for different degrees of logic compression in the case of (16 × 16) multiplier. The number of reduction stages and also compressor units count are decreased with

Table 4.1: Reduction stages and logic cell counts for (16 × 16) proposed multiplier when incorporating different levels of logic compression with Wallace-tree accumulation.

| 16 × 16 Wallace Multiplier | Phase 1: Partial Product Formation | | Phase 2: Partial Product Accumulation | | | Phase 3: Carry Propagation Adder |
|---|---|---|---|---|---|---|
| | No. of Product Terms | No. of Rows | No. of Reduction Stages | No. of Counters | | Length of Carry Chain |
| | | | | (3, 2) | (2, 2) | |
| Accurate | 256 | 16 | 6 | 200 | 52 | 24-bit |
| 2-bit SDLC | 164 | 8 | 4 | 107 | 28 | 25-bit |
| 3-bit SDLC | 126 | 6 | 3 | 69 | 23 | 25-bit |
| 4-bit SDLC | 94 | 4 | 2 | 38 | 12 | 25-bit |
| 5-bit SDLC | 88 | 4 | 2 | 33 | 15 | 24-bit |
| 6-bit SDLC | 72 | 3 | 1 | 17 | 7 | 24-bit |
| 7-bit SDLC | 68 | 3 | 1 | 14 | 9 | 23-bit |
| 8-bit SDLC | 53 | 2 | – | – | – | 23-bit |

higher depth of logic cluster. This is because the proposed SDLC approach reduces the number of product terms in the partial product bit-matrix. For instance, 2-bit SDLC is capable of minimising the number of product term to 164 compared to 256 in the case of accurate multiplier (about 36% reduction in the number of product terms). The commutative remapping approach of the reduced number of product terms translates into lower number rows in the Wallace accumulation-tree (*e.g.*, only quarter number of rows in the case of 4- and 5-bit SDLC). This accounts for substantially lower number of logic cell counts needed by Wallace accumulation (*e.g.*, only 107 (3, 2) counters and 28 (2, 2) counters are required to accumulate the resulting product terms in the case of 2-bit SDLC). The number of these counters is further reduced with increased the level of logic compression and therefore leads to shortened critical path of the multiplier design at the cost of error. Similarly, Figure 4.9 and Table 4.2 illustrate the impact of increased level of logic compression using SDLC approach coupled with Wallace method in the case of $(8 \times 8)$ multiplier. As seen, a steady length of carry chain in the CPA phase required for the approximate and accurate designs, where as a lower number of (3, 2) and (2, 2) counters with increased the depth of logic cluster. It is therefore expected to have a considerable savings in all design trade-offs as discussed in Section 4.5. Compared to $(8 \times 8)$ multiplier, $(16 \times 16)$ multiplier can provide the multiplier design with wider group of approximate versions. This can allow different applications to benefit from performance-energy-quality (PEQ) trade-offs of SDLC approach. The impact of increased degree of compression is further investigated in implementation case studies in Chapter 5.

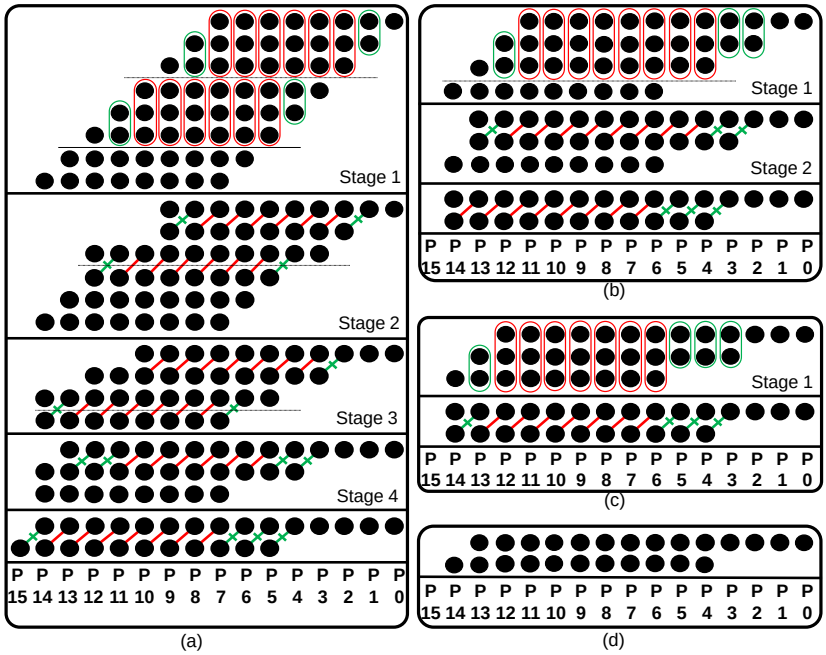


Figure 4.9: Wallace reduction stages of an (8 × 8) multiplier: (a) accurate Wallace tree; then Wallace method coupled with (b) 2-bit SDLC; (c) 3-bit SDLC and (d) 4-bit SDLC.

Table 4.2: Reduction stages and logic cell counts for (8 × 8) proposed multiplier when incorporating different levels of logic compression with Wallace-tree accumulation.

| 8 × 8 Wallace Multiplier | Phase 1: Partial Product Formation | | Phase 2: Partial Product Accumulation | | | Phase 3: Carry Propagation Adder |
|---|---|---|---|---|---|---|
| | No. of Product Terms | No. of Rows | No. of Reduction Stages | No. of Counters | | Length of Carry Chain |
| | | | | (3, 2) | (2, 2) | |
| Accurate | 64 | 8 | 4 | 38 | 15 | 11-bit |
| 2-bit SDLC | 42 | 4 | 2 | 16 | 6 | 11-bit |
| 3-bit SDLC | 33 | 3 | 1 | 7 | 4 | 11-bit |
| 4-bit SDLC | 25 | 2 | – | – | – | 11-bit |

*b. Scalability of the proposed Wallace multiplier design*

The proposed approach is scalable for any ($N \times N$) multiplier, as shown in Algorithm 4.1. This algorithm forms all partial product

terms and apply SDLC approach to generate reduced and ordered partial product bit-matrix $M$ as indicated in Line 7, which can then be treated as an accumulation tree using Wallace method as indicated in Line 8. The two rows resulting from Wallace reduction stages are combined using carry propagating adder Line 9. The algorithm associated with the SDLC approach for any $(N \times N)$ multiplier is detailed in Section 3.3.

**Algorithm 4.1** $(N \times N)$ Wallace-tree multiplier using SDLC approach with $d$-bit logic clusters.

1: **procedure** APPROXIMATE-WALLACE($P, A, B$)
2: Output: $P[1,2,...,2N]$ ▷ Final Product bits
3: Inputs: $A[1,2,...,N]$ ▷ Multiplicand bits
4: $B[1,2,...,N]$ ▷ Multiplier bits
5: Initialize:$M[1,2,...,\left\lceil \frac{N}{2} \right\rceil][1,2,...,2N-1]$ ▷ Reduced Matrix by SDLC
6: $R[1,2][1,2,...,2N-1]$ ▷ Two rows combined by CPA
7: $M \leftarrow SDLC(A,B,d)$ ▷ SDLC with $d$-bit logic compression (Algorithm 3.2)
8: $R \leftarrow WallaceReduction(M)$ ▷ Reducing $M$ to a height of two
9: $P \leftarrow CarryPropagatingAdder(R)$ ▷ Final product is generated
10: **end procedure**

## 4.3 ERROR COMPENSATION METHOD (ECM)

The lossy compressions exercised by the logic clusters introduce error in the final product. This error is originated from utilizing OR logic gate instead of expensive XOR gate as in accurate adders (discussed in Section 3.2.2). The OR gate fails to give an accurate sum if the two inputs are “ones", *i.e.*, ‘1’+‘1’ $\neq$ ‘1’$\vee$‘1’, in such case, the error value is ‘1’ as the adder returns ‘10’ and OR outputs ‘1’.

This section proposes a systematic error compensation method (ECM) to reduce the impact of the error associated with the SDLC approach. This method consists of two main steps: (i) a parallel error detection logic used to generate error-compensation bit-matrix, and then, this matrix is compressed using OR gates to generate an error compensation vector; (ii) this vector is then combined as

an additional row in the accumulation tree or used to modify an existing row. These steps together with the variable logic clusters, are described below.

### 4.3.1 *Parallel Error Detection Logic*

The error associated with OR compression can be detected using AND logic gate. For instance, 2-input AND gate is capable to detect the error when OR gate fails to find the sum of two inputs. This is shown in Figure 4.10. The output of 2-input AND gate is only set to one when both of its inputs are ones. Therefore, the output of the AND gate can be used as error signal when an error occurs.

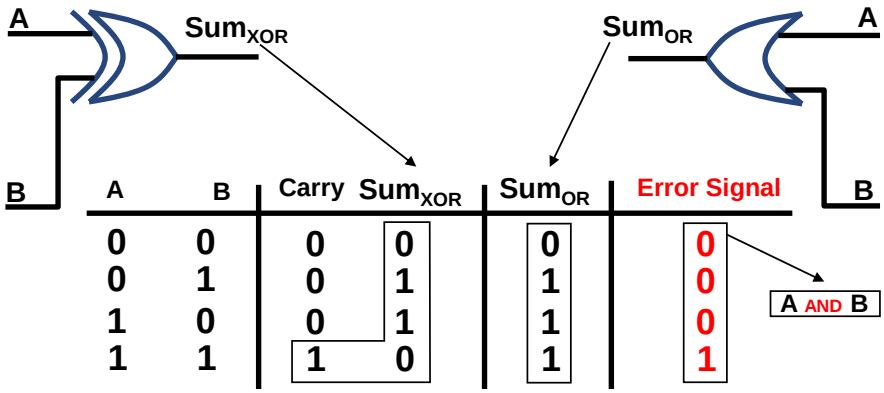


Figure 4.10: 2-bit OR gate is sufficient to find the sum of two bits.

Figure 4.11 depicts the first step in the proposed error-compensation method. This method utilizes array of AND gates to generate error signals. Each error signal indicates an error committed by the corresponding OR gate, in such case, the AND gate returns “one". The error signals are used for forming the error-compensation matrix. This matrix is generated along with the logic-OR compression in logic clusters. Thus, no overhead costs are required due to the delay of generating both of the reduced partial product and error-compensation matrices (see Figure 4.11). Then,

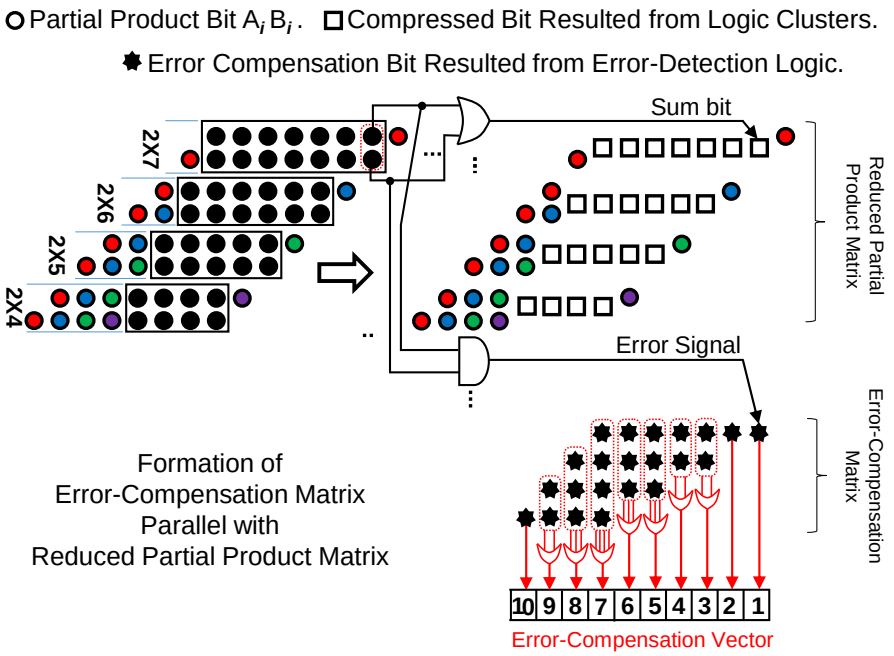


Figure 4.11: A parallel error-detection logic to generate the error compensation bit-matrix in the case of 2-bit SDLC, and then, array of OR gates to form the error-compensation vector.

the error-compensation matrix is minimized to one row, *i.e.*, error-compensation vector. This can be achieved by compressing the bits with same weights together using multiple-input OR gates. Figure 4.11 shows the process of forming 10-bit error-compensation vector in the case of $(8 \times 8)$ proposed multiplier with 2-bit logic clusters. The idea is to collect the scattered “ones" in the error-compensation bit-matrix (*i.e.*, the cases that lead to error) in a single row using low-complexity OR gates. The error-compensation vector is generated based on their bit significance to be then used for improving the accuracy of the final product.

Figure 4.12 illustrates various logic structures used to detect such errors for 2-, 3- and 4-bit depths of logic compression. These logic structures have been designed to run along with the logic clusters to generate the error signals. The error detection logic are designed in a way that the propagation delay for generating error signals does not exceed the delay required by logic compression.

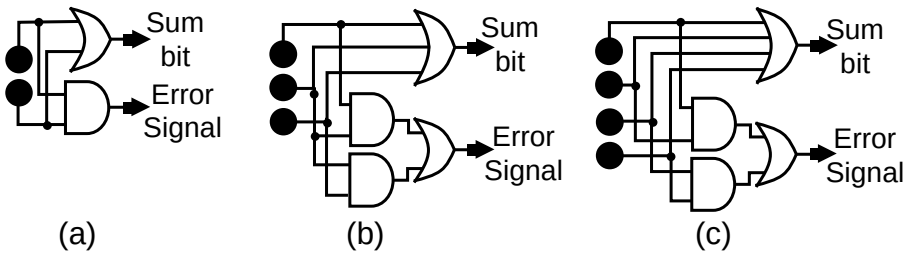


Figure 4.12: The error-detection logic circuit parallel with the logic clusters required by ECM in: (a) 2-bit; (b) 3-bit; (c) 4-bit logic clusters.

For this reason, the error detection circuit of particularly 3- and 4-bit logic clusters, is not designed to detect all possible errors, since the main aim is to improve the accuracy with minimum decreases in the performance gains. As shown in Figure 4.12, in the cases of 3- and 4-bit logic clusters, a structure of 2-input OR gate followed by a pair of 2-input AND gates is sufficient to detect most of the cases that lead to error, without causing any additional cost of delay.

### 4.3.2 *Error Compensation Vector*

The second step in the proposed error-compensation method is shown in Figure 4.13. In theory, each reduction stage in Wallace method is responsible for accumulate a group of product terms within three consecutive rows. The time delay taken by a reduction stage is equivalent to a (3, 2) counter delay, *i.e.*, a full-adder delay. For illustration purposes, the reduced partial product matrix is defined as an input requirement for Wallace accumulation, and the error compensation matrix is defined as an input requirement to generate the error compensation vector. Both of these input matrices are produced together just after one logic-gate delay

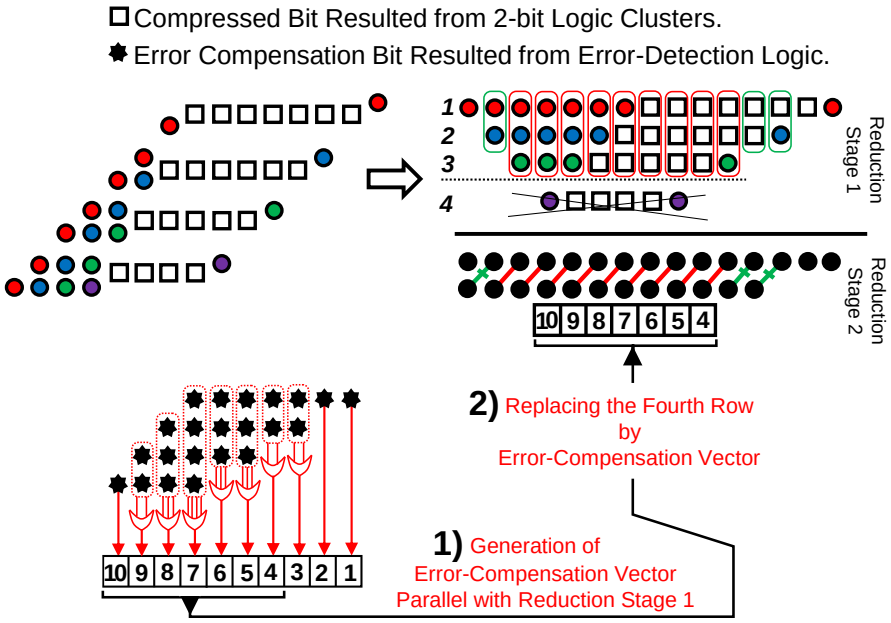


Figure 4.13: Improving accuracy by allowing error-compensation vector to modify an existing row in Wallace accumulation tree.

(assuming that OR and AND gate requires the same propagation delay, as illustrated in Figure 4.11). By allowing for maximum parallelization, the error compensation vector can be generated during the time period of the first reduction stage (see Figure 4.13). As such, the the critical delay of multiplier design less affected by the proposed ECM.

Improving accuracy of the final product depends on the way of utilizing the error compensation vector. The idea is to permit the error compensation vector to be accumulated on a carry-basis in Wallace tree. In this study, two ways are suggested. The first is to replace an existing row in Wallace-tree by the error compensation vector. In the example of 2-bit SDLC shown in Figure 4.13, after completion of the first reduction stage, the forth row of the Wallace tree is transferred to the second stage without modification. Then, this row is replaced by the error compensation vector. The second way is to consider error compensation vector as an additional row in the Wallace tree. Figure 4.14 exhibits an example of including

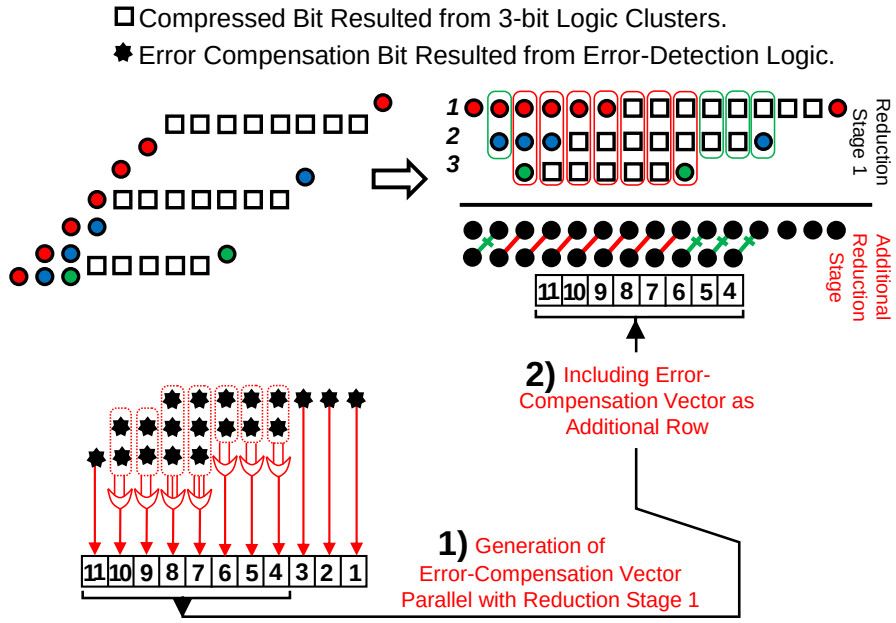


Figure 4.14: Improving accuracy by including error-compensation vector as an additional row in Wallace accumulation tree.

such a vector in the accumulation tree as an additional row in the case of 3-bit SDLC. Note that, for both ways of the error compensation, only a part of successive significant bits in the error compensation vector is included. This is because, the aim is not to increase the carry chain of the CPA phase. Thereby, the overhead costs associated with the proposed ECM is at a minimum. Compared to replacement of existing row in accumulation tree, including the error compensation vector as additional row leads to increase the hardware complexity and also the critical path delay (see Section 4.5). However, the effectiveness of the proposed ECM is examined and analysed in the next section.

## 4.4 ERROR ANALYSIS

A number of simulations are carried out to examine the impact of proposed ECM on for different degrees of logic compression. Several

error metrics have been discussed in Section 3.4 for evaluating the effectiveness and quantifying errors of proposed SDLC approach.

Further simulations are performed in Matlab by incorporating a functional model of the SDLC approach with (8 × 8) Wallace-tree accumulation. The response of approximate multipliers are evaluated for all possible combinations of operands. Table 4.3 shows five error metrics using different depths of logic compression with (8 × 8) Wallace accumulation. It can be seen that the proposed ECM improves the accuracy for all depths of logic compression. The MRED is improved more than 45% for 2-and 4-bit logic clusters and (up to 75%) for 3-bit logic clusters. Similar observation for NMED improvements (up to 76%) for 3-bit logic clusters. The increasing trend in the error rate is expected due to the increased bit-depth of logic cluster of the multiplier. This is because the growing likelihood of finding a pair of vertically aligned “ones" through two successive rows. In such cases, the corresponding OR gate will return an error (as detailed in Section 4.3). However, such proba-

Table 4.3: ECM drastically reduces the errors across all metrics.

| (8x8) Wallace Multiplier | EP (%) | MED | MSE | MRED (%) | NMED (%) |
|---|---|---|---|---|---|
| 2-bit SDLC | 49.11 | 229.38 | 251733.8 | 1.9883 | 0.3527 |
| 2-bit SDLC_ Modified_ECM | 36.75 | 167.18 | 204883.93 | 1.0762 | 0.2571 |
| **Improvements(%)** | **25.17** | **27.12** | **18.61** | **45.87** | **27.11** |
| 3-bit SDLC | 65.73 | 654.94 | 1590278.5 | 4.6847 | 1.0072 |
| 3-bit SDLC_ECM | 43.19 | 162.52 | 187754.3 | 1.1725 | 0.2499 |
| **Improvements(%)** | **34.29** | **75.19** | **88.19** | **74.97** | **75.19** |
| 4-bit SDLC | 77.57 | 2127.78 | 15309286 | 10.5835 | 3.2723 |
| 4-bit SDLC_ECM | 69.45 | 1111.45 | 4743255 | 5.5382 | 1.7093 |
| **Improvements(%)** | **10.47** | **47.76** | **69.02** | **47.67** | **47.76** |

bility of error can be misleading, as the eventual impact of error is reflected in error distance metrics, *i.e.*, MRED and NMED [20].

The majority of these errors would not denote severe degradation of the final output because the occurrence of the higher errors is regarded as very rare. This can be seen in Figure 4.15 which

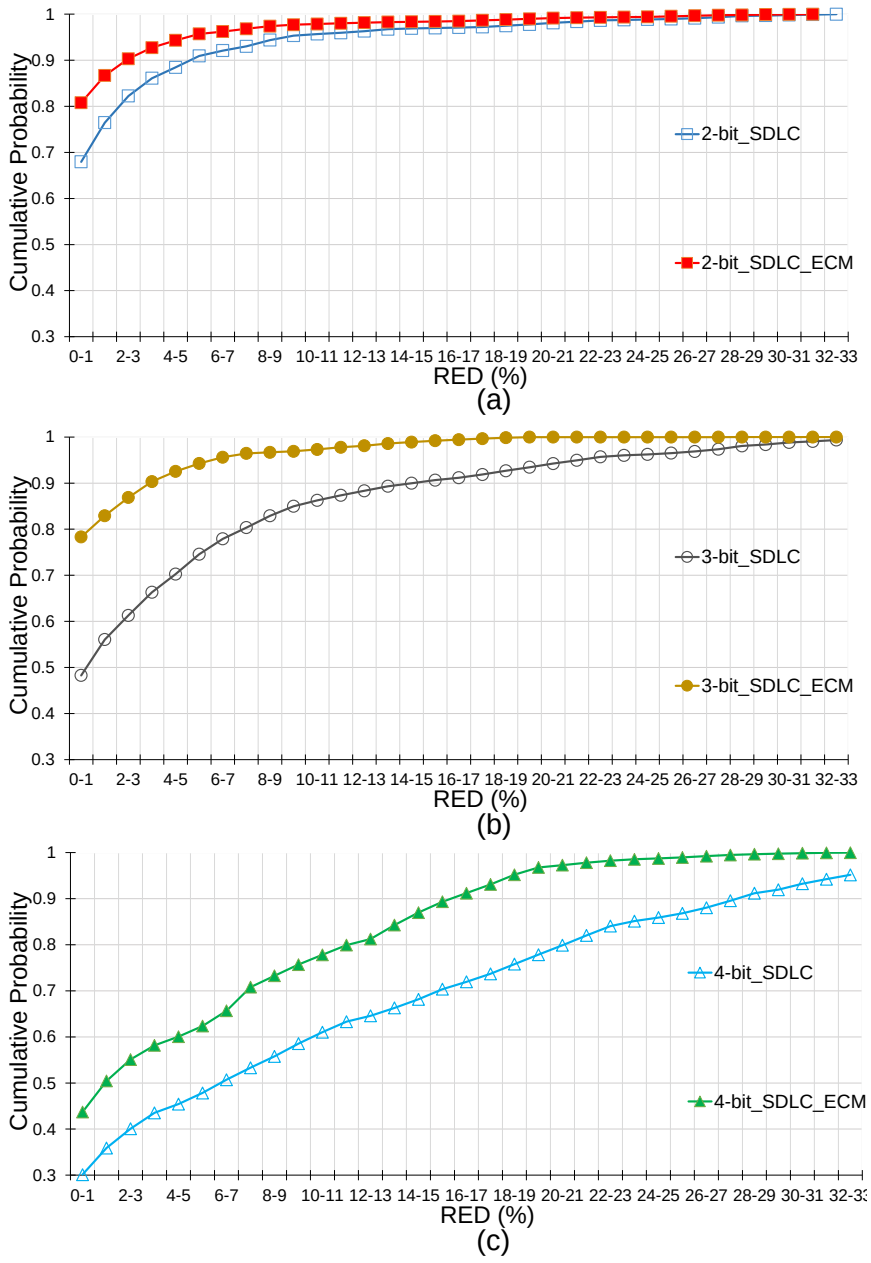


Figure 4.15: Cumulative probability distribution for the error induced by different logic compression levels coupled with the proposed ECM in the case of $(8 \times 8)$ proposed multiplier.

demonstrates the cumulative probability distribution for the relative errors resulting from Wallace-tree multiplier for different sizes of logic clusters coupled with ECM. The proposed multiplier does not sacrifice the precision of the more significant bits when using SDLC approach. This can be observed in the sharp rise of the cumulative probability of errors towards 1, especially for lower depth of logic compression, such as 2- or 3-bit SDLC. Furthermore, incorporating ECM together with SDLC approach tends to produce results that are closer to the accurate outputs. This is seen when the cumulative probability distributions reach to 1 faster than SDLC approach without ECM. The proposed ECM increases the probability of trivial errors; however, it lowers the probability of occurrence of higher RED. For example, in the case of 2-bit SDLC, the probability of having errors with less than 1% RED, *i.e.*, RED of 0%-1%, is increased from 0.68 to 0.81 when applying ECM, while the likelihood of RED of the range 9%-10% is decreased form 0.02 to 0.002 for the same case. Similar observations can be made in the case of 3 and 4-bit logic clusters. The impact of increased degree of compression coupled with ECM is further investigated in the application case-study in Section 4.5.

## 4.5 EXPERIMENTAL RESULTS AND DESIGN TRADE-OFFS

To demonstrate the proposed approach, we applied it on different $(8 \times 8)$ parallel multiplier designs. A SystemVerilog code was used to generate synthesizable modules for Wallace-tree accumulation structure coupled with 2-bit, 3-bit and 4-bit logic clusters. Accurate ripple adders were used in the last phase for adding the resulting two rows after Wallace accumulation phase. The generated codes were implemented and synthesised using two different off-the-

shelf tools: Mentor Graphics Questa Sim was used to compile the SystemVerilog codes and run the associated test benches; and Synopsys Design Compiler was utilized for synthesising all sizes of accurate and proposed multipliers when mapping the circuits to the Faraday's 90nm technology library and evaluating for power, delay and area.

Figure 4.16 illustrates the impact of proposed ECM in terms of dynamic/leakage power, delay, area and PDP savings with increased degree of logic compression. The related absolute readings of the proposed multipliers are listed in Table 4.4. As seen, there are significant improvements in all design trade-offs. This is basically

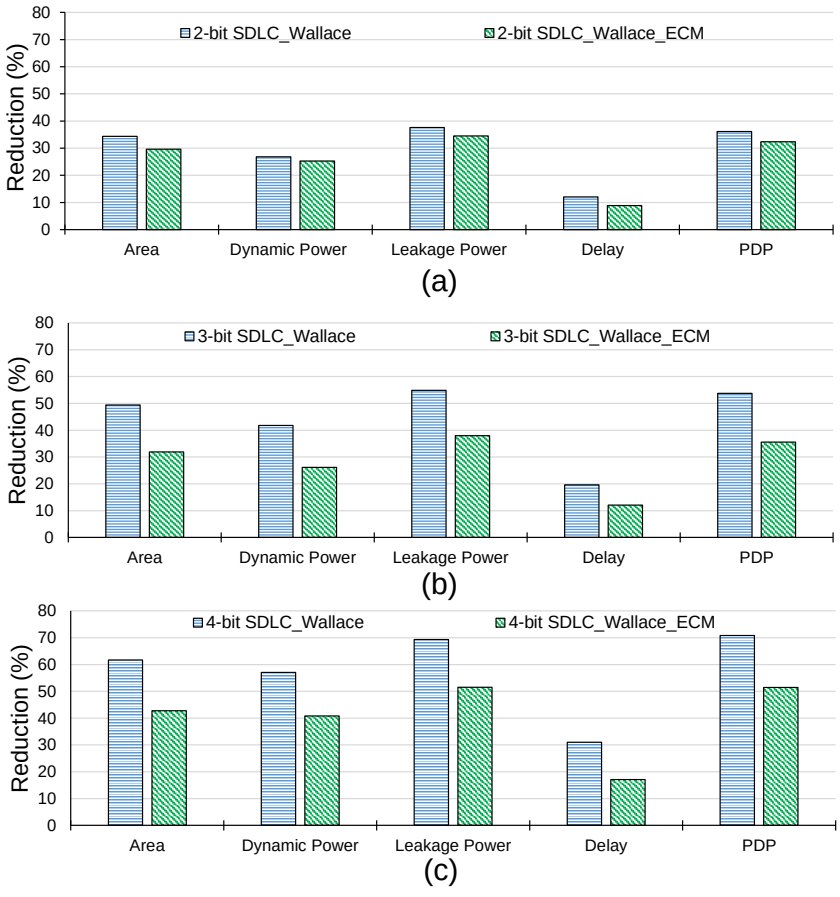


Figure 4.16: The impact of the proposed ECM on the $(8 \times 8)$ approximate Wallace multiplier with: (a) 2-bit, (b) 3-bit and (c) 4-bit logic compression levels.

Table 4.4: Design trade-offs for different compression levels of the proposed multiplier used to obtain comparative analysis in Fig. 4.16.

| | (8×8) Wallace-tree Multiplier | | | | | | |
|---|---|---|---|---|---|---|---|
| | Accurate | 2-bit SDLC | 2-bit SDLC _ECM | 3-bit SDLC | 3-bit SDLC _ECM | 4-bit SDLC | 4-bit SDLC _ECM |
| Area ($um^2$) | 1726.4 | 1133.7 | 1214.4 | 873.4 | 1175.2 | 661.7 | 988.6 |
| Dynamic (uW) | 68.46 | 50.09 | 51.14 | 39.88 | 50.56 | 29.42 | 40.51 |
| Leakage (uW) | 3.83 | 2.39 | 2.51 | 1.73 | 2.38 | 1.18 | 1.86 |
| Delay (ns) | 1.58 | 1.39 | 1.44 | 1.27 | 1.39 | 1.09 | 1.31 |
| PDP (pJ) | 114.2 | 72.94 | 77.26 | 52.84 | 73.59 | 33.35 | 55.51 |

because SDLC approach decreases the number of reduction stages in Wallace accumulation phase (see Section 4.2.3). Furthermore, this reduction in hardware complexity leads to low switching capacitance and leakage reading as well as shortened critical paths. In the case of 2-bit logic clusters, slight decreasing of critical delay and power consumption comparing to 3-bit and 4-bit logic compressions. This is because the error compensation vector is utilized by replacing the fourth existing row without increasing the number of reduction stages (see Section 4.3).

Combining ECM method with the SDLC translates into additional cost of area, delay and power. For example, while the area was divided by 2 with the 3-bit SDLC compared to the traditional Wallace multiplier, it is only reduced by 30% with the 3-bit SDLC-ECM (first item of Figure 4.16-b). This is due to increasing the number of reduction stages when including the error compensation vector

to the accumulation tree as additional row for the cases 3- and 4-bit SDLC approach (see Section 4.3). For dynamic and leakage power, the reductions obtained from applying error compensation method range from 25.3%-40.8% and 34.5%-51.5% respectively. Furthermore, the range of savings in the operating delay for the proposed multiplier is from 8.9%-17.1%. The reduction in hardware complexity also leads to silicon area to be reduced by 29.7%-42.7%, and energy is reduced as a PDP by 32.4%-51.4%. Figure 4.17 shows the impact of proposed ECM in terms of power, delay, area and PDP savings with increased degree of logic compression (2- to 4-bit SDLC) for (16 × 16) Wallace multiplier. The related absolute read-

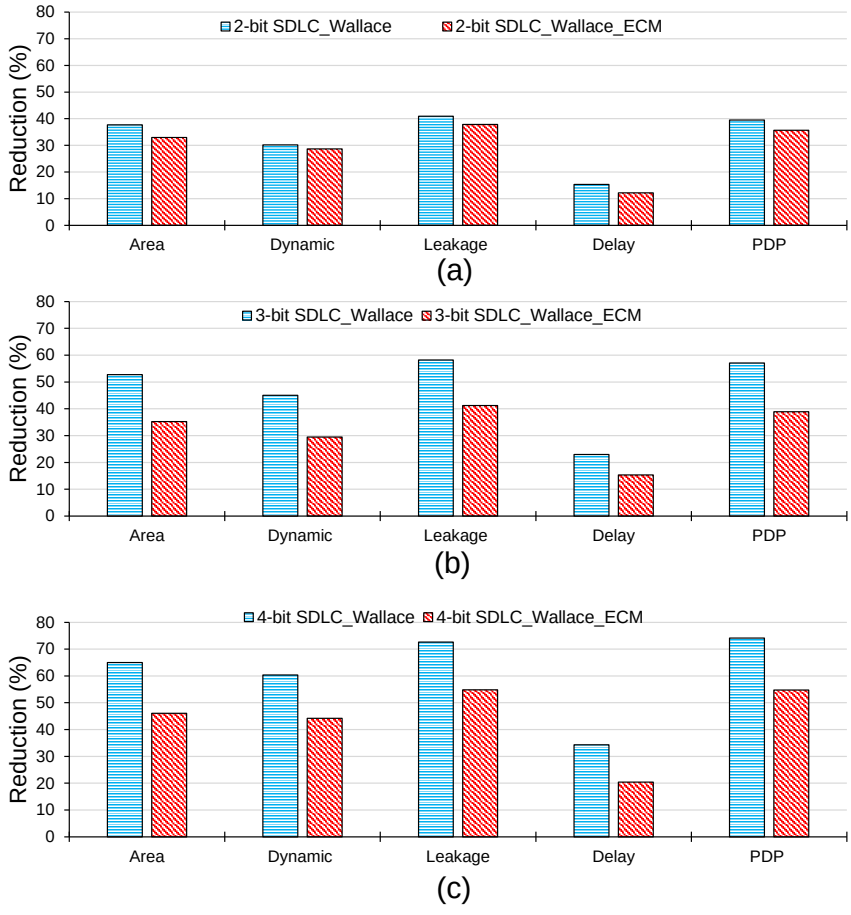


Figure 4.17: The impact of the proposed ECM on the (16×16) approximate Wallace multiplier with: (a) 2-bit, (b) 3-bit and (c) 4-bit logic compression levels.

ings of the proposed multipliers are listed in Table 4.5. Similar to proposed (8 × 8) multiplier, there are significant improvements in all design trade-offs. As can be observed, the overhead cost associated with replacing existing row in PPM by error-compensation vector (Figure 4.17-a), is less than the case when combining this vector as additional row (Figure 4.17-b and -c). For example, in Figure 4.17-a, the area, (dynamic/leakage) power, delay and PDP savings are slightly decreased by 4.7%, (1.5% 3.1%), 3.2% and 3.8%, respectively, compared to approximate Wallace multiplier without applying ECM approach. While in Figure 4.17-b, the overhead cost with ECM are increased by 17.5%, (15.6% 17%), 7.6% and 18.2% for the same order. This is because the proposed ECM offers different techniques to compensate the error introduced by SDLC, with variable overhead costs. Therefore, the SDLC approach with its potential to provide different levels of logic compression

Table 4.5: Design trade-offs for different compression levels of the proposed multiplier used to obtain comparative analysis in Fig. 4.17.

| | (16×16) Wallace-tree Multiplier | | | | | | |
|---|---|---|---|---|---|---|---|
| | Accurate | 2-bit SDLC | 2-bit SDLC _ECM | 3-bit SDLC | 3-bit SDLC _ECM | 4-bit SDLC | 4-bit SDLC _ECM |
| Area ($um^2$) | 7036.40 | 4386.49 | 4715.80 | 3325.40 | 4555.37 | 2462.74 | 3795.43 |
| Dynamic (uW) | 324.29 | 226.45 | 231.44 | 178.10 | 228.72 | 128.55 | 181.08 |
| Leakage (uW) | 16.16 | 9.55 | 10.05 | 6.75 | 9.50 | 4.43 | 7.30 |
| Delay (ns) | 3.05 | 2.58 | 2.68 | 2.35 | 2.58 | 2.00 | 2.43 |
| PDP (pJ) | 22.45 | 13.59 | 14.44 | 9.64 | 13.71 | 5.81 | 10.16 |

together with the proposed ECM can allow for more diverse in PEQ trade-offs.

## 4.6 CONCLUDING REMARKS

This chapter studied two novel design approaches. First is an approximate multiplier design combines the SDLC approach with Wallace-tree accumulation method. The reduced number of rows resulted from configurable lossy compression, led to decreased number of reduction stages and logic units. The results obtained after synthesis have shown remarkable decrease in latency, power consumption and area, compared to accurate Wallace-tree multiplier. This is achieved at the cost of introduced error in the less significant bits of multiplier output.

Second is a systematic ECM approach, which consists of a parallel error detection logic used to generate error compensation bit-matrix. This matrix is then compressed vertically using OR gates to generate an error compensation vector. In order to mitigate the impact of error introduced by SDLC, this vector is either considered as an additional row or used to modify an existing one. Multiple 8- and 16-bit multipliers were designed and synthesized to evaluate the effectiveness of ECM approach. We showed that the proposed error compensation method effectively reduces the resulted error from variable sizes of logic clusters at variable overhead costs. These trade-offs are further substantiated by a case study of convolution filter used in image processing in next chapter.

# 5

# IMPLEMENTATION AND VALIDATIONS

## 5.1 INTRODUCTION

Chapter 3 and 4 demonstrated the performance-energy quality (PEQ) trade-offs for different approximate multiplier configurations. The SDLC approach is aimed at reducing the number of product rows using progressive bit significance, and thereby decreasing the number of reduction stages in Wallace-tree accumulation. This accounts for substantially lower number of logic counts and lengths of the critical paths at the cost of errors in lower significant bits. These errors are minimised through a parallel error detection logic and compensation vector approach.

This chapter demonstrates the effectiveness of the energy-quality trade-offs achieved by the proposed approaches. These trade-offs can be used to implement multipliers in applications. As such, two case studies are set up. First, image processing application where a Gaussian blur filter was designed, which demonstrated up to 80% energy reduction with a negligible loss of image quality. Second, we evaluate our approach in machine learning application using perceptron classifier, showed up to 74% energy reduction with negligible error rate. Note that, for this chapter, the investigation into enabling energy-efficiency is done by replacing the standard existing multiplier units across applications with proposed approximate multipliers.

## 5.2 CASE STUDY 1: GAUSSIAN BLUR FILTER

We evaluate the efficiency of the proposed approach using a Gaussian blur filter application. The application consists of additions and multiplications using key multipliers as building blocks. Our analysis considers the Gaussian blur filter [117] since it is widely used to reduce image noise and detail by acting as a low-pass filter. This filter involves the convolution through a "kernel", described by a Gaussian function, with the pixels of the image. The pixel values in the input image are multiplied by the corresponding entry of the kernel (*i.e.*, the pixels that overlap with the a given kernel). Then, all the obtained multiplications are added and become a new pixel in the output image.

Fig. 5.1 demonstrates a test platform to examine the effectiveness of the proposed multipliers on the quality of final output image processed by Gaussian blur filter. Different versions of 8-bit and 16-bit multipliers together with the Gaussian blur algorithm are combined. All modules are implemented in Matlab covering 2-, 3- and 4-bit depth clustering, in the case of $(8 \times 8)$ multiplication, and from 2- to 8-bit depth clustering for $(16 \times 16)$ multiplication. The Gaussian kernel is $(3 \times 3)$ with a 1.5 standard deviation value and it uses 8- and 16-bit fixed point arithmetic. The Gaussian blur filter is applied to 8-bit and 16-bit gray-scale input images of size of $(500 \times 500)$ pixels. We approximate Gaussian blur by replacing the standard multiplication in the Gaussian filter with the aforementioned approximate multipliers.

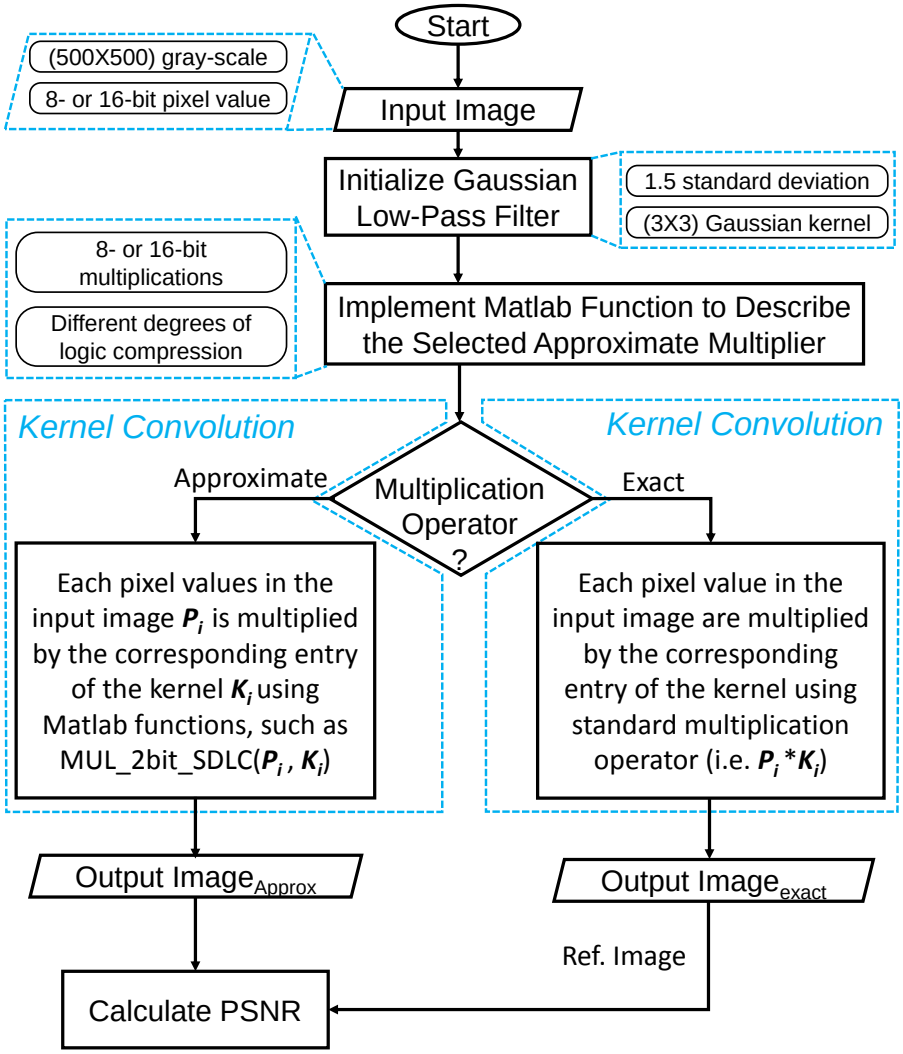


Figure 5.1: Flowchart diagram showing the main steps for evaluating the impact of the proposed multiplier on the final quality of image processed by Gaussian blur filter.

The peak signal-to-noise ratio (PSNR) is a fidelity metric used to measure the quality of the output images. PSNR is expressed as [55]:

$$PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right) \quad , \tag{5.1}$$

where MSE is the mean squared-error measured with respect to the reference pixel. To calculate the consumed energy in the multiplier unit required to process the input image, we follow this equation:

$$Energy = Power * Delay * N \quad , \tag{5.2}$$

where *Power* and *Delay* are obtained for one multiplier design from the synthesis tool. $N$ is the number of multiplications necessary to treat the input image by Gaussian filter. The energy savings are then calculated compared to the accurate Wallace multiplier.

Fig. 5.2 and Fig. 5.3 demonstrate the impact of different bit-depth clustering on the image quality after applying the Gaussian blur filter. The standard multiplier and number of different levels of approximation for the proposed $(8 \times 8)$ and $(16 \times 16)$ multipliers are used.

As can be seen, the use of the SDLC approach can yield fruitful results. The PSNR for the case of 2-, 3- and 4-bit logic clustering for $(8 \times 8)$ SDLC are 50.2, 39 and 30 dB respectively, whereas the PSNR values are 70.2, 61.3, 51.4, 42.8, 39.3, 83.2 and 30.2 dB, when treating the images using 2- to 8-bit logic clustering for $(16 \times 16)$ SDLC. The values of PSNR are computed compared to the image resulting after applying Gaussian blur filtering with the case of accurate multiplication. Thus, the proposed approach can provide a significant dynamic energy saving up to 80.1% with acceptable quality of output image, especially when using smaller bit depth clusters such as 2- and 3-bit for $(8 \times 8)$ SDLC and 2- to 6-bit for $(16 \times 16)$ SDLC. As can be observed from Fig. 5.3, the proposed $(16 \times 16)$ multiplier allows for more levels of energy/quality trade-offs, comparing to the output quality of $(8 \times 8)$ multiplier in Fig. 5.2.

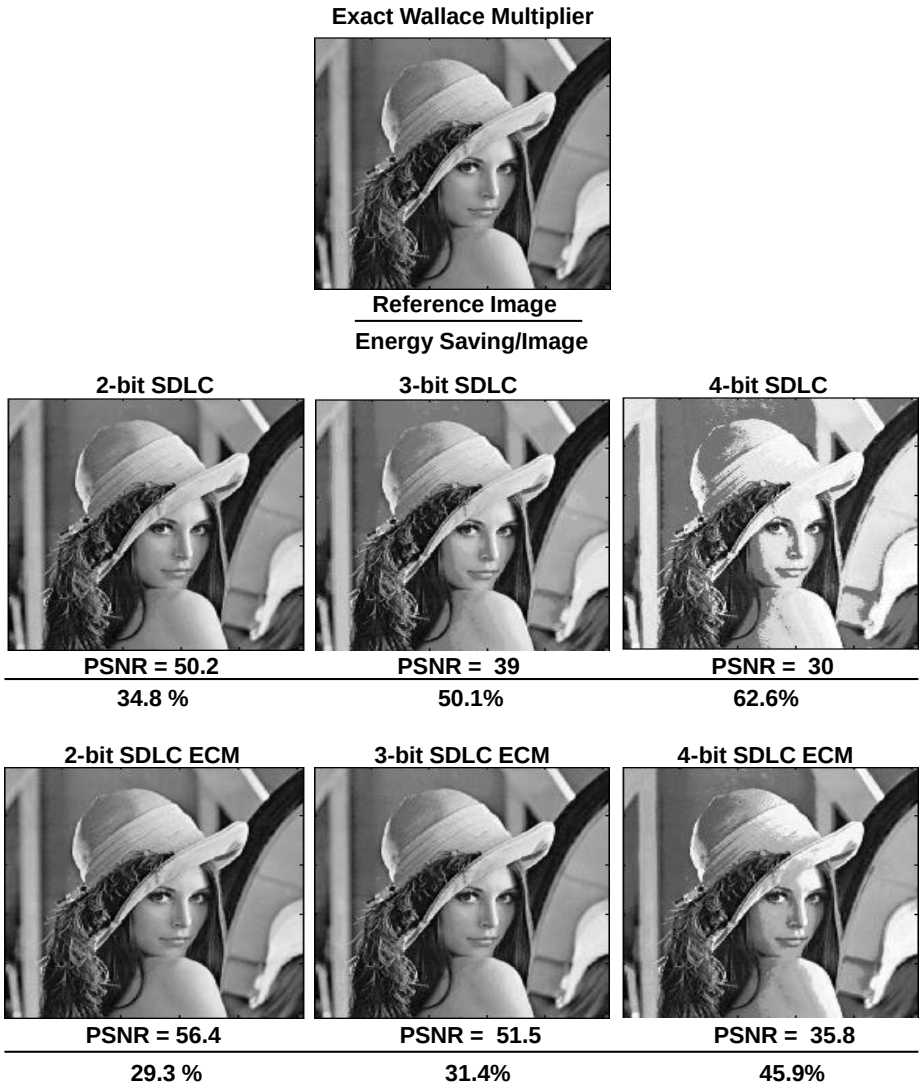


Figure 5.2: Output quality after applying Gaussian blur filtering for different degrees of logic compression of the proposed ($8 \times 8$) multiplier.

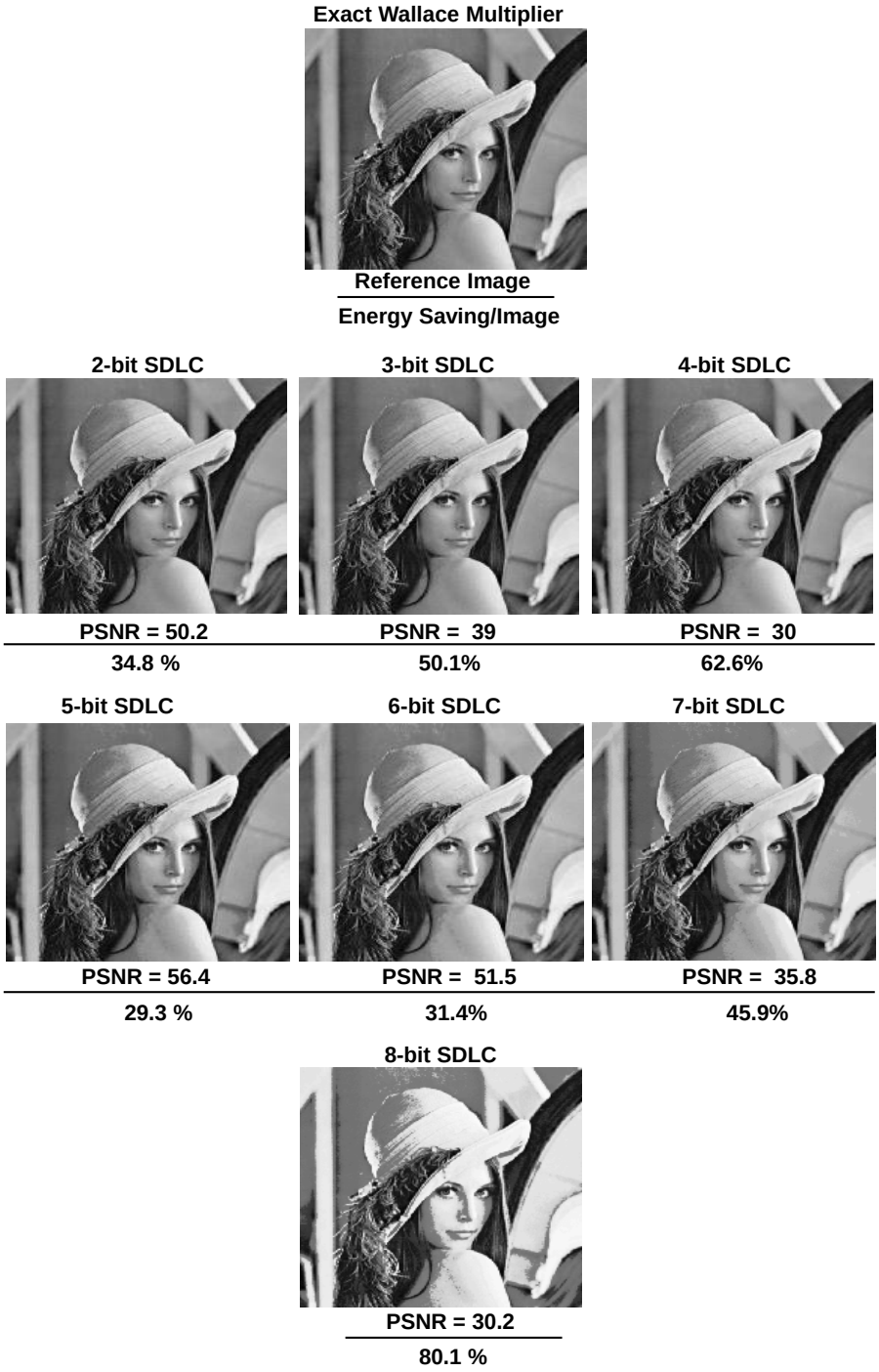


Figure 5.3: Output quality after applying Gaussian blur filtering for different degrees of logic compression of of the proposed (16 × 16) multiplier.

## 5.3 CASE STUDY 2: PERCEPTRON CLASSIFIER

Neural networks are flexible model functions that are built up from the single or cascade of several layers, each of which is a collection of perceptron functions. A perceptron is a binary linear classifier that divides space into parts using linear functions [39]. We implement the proposed multiplier in a single layer feed-forward neural perceptron, as shown in Fig. 5.4. We exercise perceptron for learning a binary classifier, *i.e.* a function which takes the inputs $x_1, x_2, ..., x_m$ and produces an output value y. The output y is a single binary value, expressed as:

$$y = \begin{cases} +1, & w \cdot x + b > 0, \\ -1, & \text{otherwise}, \end{cases} \tag{5.3}$$

where $w$ is a vector of real-valued weights, which vary over runtime depending on the number of training input samples and the training rate, $w \cdot x$ is the dot product, *i.e.* $\sum_{i=1}^{m} w_i \cdot x_i$, $m$ is the number of inputs to the perceptron and b is the bias (0 used in our example). We used (5.3) to classify patterns that are linearly separable [40].

Fig. 5.5 shows a test platform to evaluate the effectiveness of proposed multipliers on perceptron-based machine learning appli-

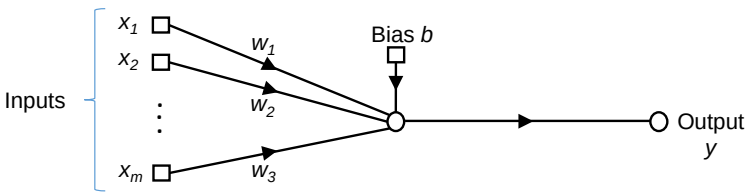


Figure 5.4: Signal-flow graph of the perceptron.

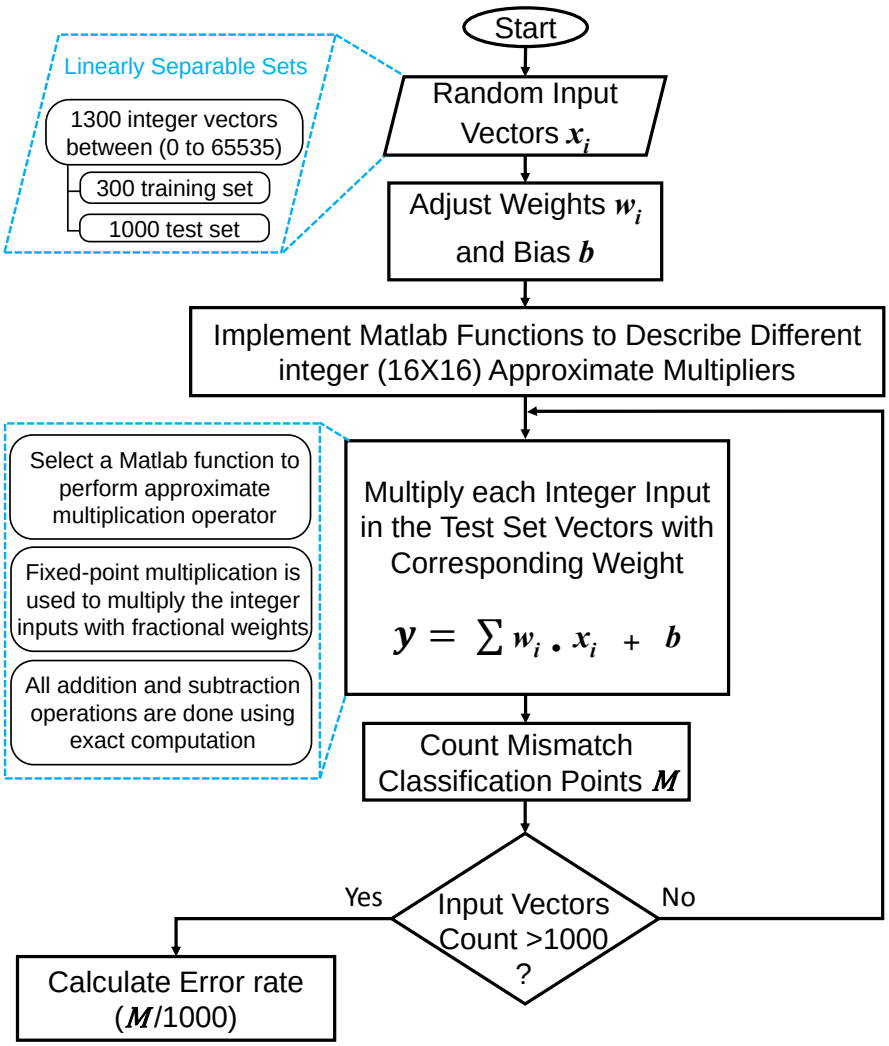


Figure 5.5: Flowchart diagram demonstrating the main steps for evaluating the impact of the proposed multiplier on a perceptron-based Classifier.

cation. For perceptron learning algorithm, a training set is used to train the perceptron to classify inputs correctly. This is accomplished by adjusting the connecting weights and the bias to properly handle linearly separable sets. All input sets are randomly generated and independently distributed to generate integer numbers between (0 to 65535) to allow for using 16-bit multiplication. Then, we evaluate the classifier against test set of 1000 two-dimensional points that belong to two classes [-1,+1], see (5.3). The approxi-

mate multipliers are used to multiply the perceptron inputs by the weights vectors.

However, for all experiments, fractional numbers are computed by using fixed-point representation. The idea is to exploit low-cost integer multipliers to perform multiplication of each integer input in the test set with the corresponding fractional weight $w$. To perform fixed-point multiplication, the binary point is ignored by scaling up the weights by a constant factor and then, determine the position of the binary point for the result [84]. For instance, if the $w_i = (0001.100110110001)_2 = (0001100110110001)_2 \times (2^{-12})_{10}$ and $x_i = (1100100100100101)_2$, then the $w_i \times x_i = (0001100110110001)_2 \times (1100100100100101)_2 \times (2^{-12})_{10}$. This means that the binary point is put to the left of the $12^{th}$ bit of the final product to obtain the desired multiplication result. The addition and subtraction operations used to generate the output value $y$ are done by performing exact computation. This allows only multipliers to impact the quality of classification done by the perceptron.

The error rate (ER) is the ratio of mismatch between classified class and the actual output. Fig. 5.6 demonstrates the comparison of the classification problem with accurate ($16 \times 16$) multiplier and the proposed 2-bit SDLC design. Compared to the accurate multiplier, the proposed SDLC multiplier classifies six points, from the 1000 points in the testing set, as class 1 by mistake. Note that, even the design that uses the accurate multiplier cannot classify all points correctly (three mismatched points).

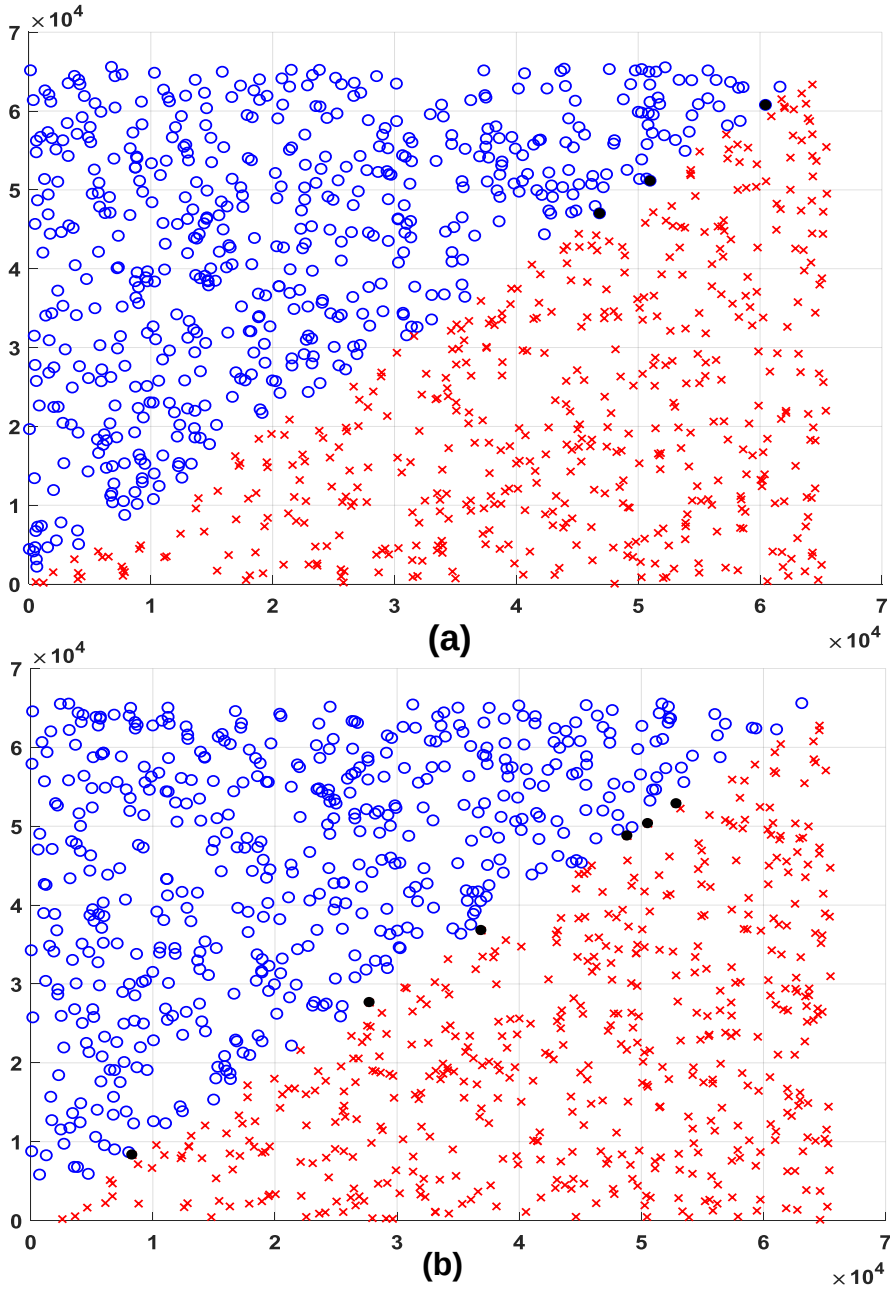


Figure 5.6: The test set perceptron classification using; (a) accurate multiplier; (b) 2-bit SDLC proposed multiplier, where the axises show the random inputs between 0 to 65535. (blue and red points represent two classes -1 and +1, black dots for mismatch classification points.)

Table 5.1 shows the comparative error rates and energy advantages of our approach and various (16 × 16) multipliers. The energy savings is calculated by (5.2). The proposed approach outperforms the other designs and can provide energy saving of up to 74.2% with acceptable error rates, especially when utilizing lower bit depth clusters such as 2- and 3-bit SDLC. However, for 4-bit SDLC, the increased ER in Table 5.1, is expected. This is because higher depth of logic clusters affects the accuracy when multiplying the inputs with their associated weights. When compared with existing approaches, such as ETM [66] and Kulkarni *et al.* [65], the 4-bit SDLC produces a solution with comparable ER.

Table 5.1: Error rate results and energy savings for perceptron classifier

| (16× 16) Multiplier | ER (%) | Energy savings (%) |
|---|---|---|
| Accurate | 0.3 | – |
| Proposed (2-bit SDLC) | 0.6 | 40.2 |
| Proposed (3-bit SDLC) | 2.2 | 68.7 |
| Proposed (4-bit SDLC) | 12.7 | 74.2 |
| ETM [66] | 12.1 | 38.2 |
| Kulkarni [65] | 6.7 | 27.4 |

## 5.4 CONCLUDING REMARKS

This chapter showed an investigation into enabling energy-efficiency by implementing the SDLC design approaches in imprecision-resilient applications. This investigation involves a various group of (8×8) and (16×16) multipliers with different levels of approximations achieved through configurable logic clustering. The multiplier designs were implemented in two real-application case studies demonstrating comparative advantages of the proposed approaches.

The first case study shows energy-quality trade-offs of SDLC multiplier applied in an image processing application. A Gaussian blur filter was designed to investigate into these trade-offs, demonstrating up to 80% energy reduction with a minor loss in image quality. A group of simulations was carried out to examine the effectiveness of ECM, showing an increase in the image quality with comparative energy overhead costs.

The second case study evaluates the SDLC approach in machine learning application. The energy-quality trade-offs are leveraged in a perceptron-based classification. Using different configurations of SDLC multiplier led to substantial reductions in the energy consumed by the perceptron (up to 74%), with negligible error rate. The literature has not shown many research works to exploit approximate multiplier design in machine learning applications. Very recently, a promising efforts for using approximate multiplier for extracting patterns and detecting trends in neural computing paradigm can be found [102].

# 6

# CONCLUSIONS AND FUTURE WORK

## 6.1 SUMMARY AND CONCLUSIONS

Approximate computing has recently gained a lot of traction as a viable alternative to exact computing in many of imprecision-resilient applications. It offers various design techniques for building highly performance- and energy-efficient on-chip systems at different abstraction levels. As one of the de facto sub-area of approximate circuits, approximate arithmetic (such as adder and multiplier) has received more attention in the literature. This thesis proposed an investigation into approximate multiplier design as a promising basis for tackling the performance/energy efficiency challenges in the electronics and ubiquitous computing industry. This section summarises the main conclusions drawn from this thesis.

Multipliers, with complex logic design, have been considered as a real challenge in modern applications. This is either because they are the most energy-demanding data processing units or due to the large number of multiplications required to compute outcomes. The literature shows different research efforts to use approximate multiplier to improve the computational and energy efficiency with various degrees of accuracy loss. The key design principle of these efforts is either to apply functional (such as reducing logic complexity) or timing (such as VOS and over-clocking) approximations, at different abstraction levels. However, the state-of-the-art tech-

niques of approximate multipliers face different challenges, which are discussed in Chapter 2.

To mitigate the impact of challenges, a novel energy-efficient approximate multiplier design using significance-driven logic compression (SDLC) approach, have been proposed. The SDLC approach has the ability to reduce the number of product terms by exercising variable sizes of logic clusters, whose main component is low-complexity structure of OR logic gates. Then the commutative remapping method is used to reduce the number of product rows. As such, the complexity of the multiplier in terms of logic cell counts and lengths of critical paths is drastically reduced. The results obtained after synthesis have shown substantial decrease in run-time, power consumption and even in silicon area. On a statistical basis, various error metrics, such as NMED and MRED, show how the impact of error is alleviated when the size of the multiplier is increased. Additionally, the error distributions show high right-skewness for error probabilities, indicating that the proposed multiplier gives close to exact products for most inputs. This is because the SDLC approach preserves higher-significance bits of the final product, to a large extent. We demonstrate the performance-energy-quality (PEQ) trade-offs for variable levels of approximations achieved through configurable logic clustering, showing that higher depth of clustering achieves considerable savings in all design trade-offs. Additionally, we show that the proposed multiplier is scalable for any $(N \times N)$ size with $d$-bit compression. Also, a comparative approach is proposed to examine the SDLC approach against different state-of-the-art approximate multipliers. We show that the proposed multiplier outperforms in terms area, power, delay and PDP, especially with higher bit-widths, such as 16- and 32-bit multipliers. The implementation

requirements of proposed approach when performing signed multiplication is also described.

The advantages achieved by SDLC approach can be harnessed to benefit different schemes of standard multiplication. In this work, we combine the SDLC approach together with a Wallace-tree accumulation method to shorten the number of reduction stages. The SDLC approach aims to decrease the number of product rows, while Wallace method is applied to reduce these rows to the height of two before the final product is generated by carry propagating adder (CPA). As such, the hardware complexity of the multiplier implementation is drastically reduced. The results obtained after synthesis have revealed remarkable improvements in latency, area and power consumption. These savings have been achieved at variable costs of error, depending on the level of approximation performed by variable depths of logic clusters. To mitigate the impact of such error, a parallel error-detection-compensation method (ECM) is proposed using low-complexity logic structure. This method aims at generating error compensation vector to either combine it as additional row in the accumulation tree or to replace one of the existing rows. We have examined the effectiveness of the ECM, based on the results after synthesis and the error analysis. We demonstrate that using SDLC along with standard schemes of multiplication can extract manifold improvements with a minimal loss in output quality. Furthermore, we establish that the skewed error behaviour for different combinatorial pattern of inputs can be mitigated by using ECM at low overhead cost.

The PEQ trade-offs achieved by the proposed multiplier designs are investigated into two real-application case studies demonstrating comparative advantages of SDLC approach. First, Gaussian blur filter is designed, demonstrating remarkable energy reduction

with a meagre loss of image quality. Second, perceptron classifier is used to evaluate our approach in machine learning application, showing that the proposed multiplier design can do its job in classifying inputs with negligible error rate. These case studies show that exploiting advantages of the proposed design is highly conducive to significant energy improvements with an almost imperceptible loss of application quality.

## 6.2 CRITICAL REVIEW AND FUTURE WORK

The objectives of this thesis include opening a new research horizons in future approximate multiplier designs. Therefore, many research directions can be drawn and motivated from this thesis to achieve more performance and energy efficiency. The limitations of this work and directions for future research are discussed as follows:

•*Dynamic Reconfigurability*: The performance-energy-quality (PEQ) derived from the SDLC approach depends on the configuration parameters, such as the size of the multiplier and depth of logic compression (see Section 3.3). The SDLC approach provides design-time configurable logic clustering of product terms, which is suitably chosen for a given energy-accuracy trade-off. However, as the quality requirements of applications may vary significantly at runtime, accuracy-configurable approximate multiplier designs are preferable. This can be done by implementing systematic models that evaluate different depth logic compression for different application quality constraints. Similarly, the proposed error-detection logic and error-compensation method (ECM) could configure runtime accuracy by controlling the length/value of error-compensation vector.

•*Dynamic Voltage Frequency Scaling* (DVFS): Due to the reduced number of rows in the accumulation tree, the critical path delay in the proposed multiplier is drastically shortened (see Section 3.5). This can be leveraged to improve energy/performance efficiency by allowing the voltage/frequency to be set for reduction in energy consumption or increasing the throughput of multiplication, without introducing additional timing errors. For instance, for power-adaptive computing purposes, a design may use various implementations of approximate and exact multipliers to execute the workload. This can be explioted by slack reclamation approach [99], which utilizes the available slack time of the tasks executed by approximate multipliers to deliberately slow down the execution (*e.g.* scaling down the operational clock frequency). The aim is to reduce power/energy, while meeting performance deadlines.

•*Systematic Analysis and Verification*: In this work we analyse the impact of the error derived from various levels of approximations by performing statistical techniques (Monte-Carlo simulations). These techniques are very time consuming (*e.g.* 32-bit and 64-bit multipliers), and not flexible enough to support the design of the multiplier where formal error bounds can be given as a part of the input. Conventional Boolean analysis techniques (*i.e.*, verification tools such as Boolean satisfiability (SAT) solvers and Binary Decision Diagrams (BDDs)) can be used to ensure that the selected approximations satisfy a given quality constraints; however, these techniques have some limitations, such as for cases where the BDDs cannot be constructed [127] (e.g., larger bit-width multipliers), or where time consuming loops are required for *fitness function* [15]. As such, the use of the SDLC approach in industrial practice could be limited by the lack of verification. Finding closed-form expression for faithful logic compression in the SDLC

approach is very effective choice to this end; however, it requires good understanding of a range of aspects, such as parametrization and analytic error bounds, which are both not simple and highly application dependent. As a consequence, similar to other existing approximate circuits, the systematic analysis and verification of the proposed multiplier is also being considered for future research.

We believe that the research outcomes generated by this thesis will be useful for circuit design community, and continue inspire further research and development in the above-mentioned directions.

# Part II

# Thesis Bibliography

# BIBLIOGRAPHY


[1] Faraday technology corporation. http://www.faraday-tech.com. [Online; accessed 19-July-2018].

[2] K. Al-Maaitah, G. Tarawneh, A. Soltan, I. Qiqieh, and A. Yakovlev. Approximate adder segmentation technique and significance-driven error correction. In *PATMOS*, pages 1–6, 2017. doi: 10.1109/PATMOS.2017.8106986.

[3] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu. A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks. *IEEE Transactions on Circuits and Systems*, 38(1):109–123, Jan 1991. ISSN 0098-4094.

[4] A. S. G. Andrae and T. Edler. On global electricity usage of communication technology: Trends to 2030. *Challenges*, 6(1): 117–157, 2015. ISSN 2078-1547. doi: 10.3390/challe6010117. URL http://www.mdpi.com/2078-1547/6/1/117. [Online; accessed 19-July-2018].

[5] C. R. Baugh and B. A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22(12):1045–1047, 1973. ISSN 0018-9340. doi: 10.1109/T-C.1973.223648.

[6] D. Bera, S. Maitra, S. Roychowdhury, and S. Chakraborty. Diagnosis of single faults in quantum circuits. *arXiv preprint arXiv:1512.05051*, 2015.

[7] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis. Energy-efficient cloud computing. *The computer journal*, 53(7):1045–1051, 2010.

[8] K. Bhardwaj and P. S. Mane. ACMA: Accuracy-configurable multiplier architecture for error-resilient System-on-Chip. In *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–6, July 2013. doi: 10.1109/ReCoSoC.2013.6581532.

[9] K. Bhardwaj, P. S. Mane, and J. Henkel. Power- and area-efficient approximate wallace tree multiplier for error-resilient systems. In *ISQED*, pages 263–269, 2014. doi: 10.1109/ISQED.2014.6783335.

[10] P. Bonatto and V. G. Oklobdzija. Evaluation of Booth's algorithm for implementation in parallel multipliers. In *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 608–610 vol.1, Oct 1995. doi: 10.1109/ACSSC.1995.540620.

[11] A. D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951. doi: 10.1093/qjmam/4.2.236.

[12] S. Boroumand, H. P. Afshar, P. Brisk, and S. Mohammadi. Exploration of approximate multipliers design space using carry propagation free compressors. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 611–616, Jan 2018. doi: 10.1109/ASPDAC.2018.8297390.

[13] M. A. Breuer. Intelligible test techniques to support error-tolerance. In *13th Asian Test Symposium*, pages 386–393, Nov 2004. doi: 10.1109/ATS.2004.51.

[14] D. J. Brown and C. Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3):50–58, 2010.

[15] M. Ceska, J. Matyas, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *ICCAD*, pages 416–423, 2017. doi: 10.1109/ICCAD.2017.8203807.

[16] L. N. Chakrapani, B. E. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee. Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology. In *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, volume 1, pages 1–6. IEEE, 2006.

[17] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, page 113, 2013.

[18] H. Cho and E. E. J. Swartzlander. Serial parallel multiplier design in quantum-dot cellular automata. In *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, pages 7–15, June 2007. doi: 10.1109/ARITH.2007.32.

[19] V. Choi. Systems, devices, and methods for analog processing, 2012. US Patent 8,190,548.

[20] I. Chong, H. Y. Cheong, and A. Ortega. New quality metric for multimedia compression using faulty hardware. In *VPQM for Consumer Electronics*, pages 267–272, 2006.

[21] A. Cilardo, D. De Caro, N. Petra, F. Caserta, N. Mazzocca, E. Napoli, and A. G. M. Strollo. High speed speculative multipliers based on speculative carry-save tree. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(12):3426–3435, Dec 2014. ISSN 1549-8328. doi: 10.1109/TCSI.2014.2337231.

[22] G. Csaba and W. Porod. Computational study of spin-torque oscillator interactions for non-Boolean computing applications. *IEEE Transactions on Magnetics*, 49(7):4447–4451, July 2013. ISSN 0018-9464. doi: 10.1109/TMAG.2013.2244202.

[23] L. Dadda. Some schemes for parallel multipliers. *Alta frequenza*, 34(5):349–356, 1965.

[24] V. De. Energy-efficient computing in nanoscale CMOS. *IEEE Design Test*, 33(2):68–75, 2016. ISSN 2168-2356. doi: 10.1109/MDAT.2015.2513400.

[25] T. Drane, T. Rose, and G. A. Constantinides. On the systematic creation of faithfully rounded truncated multipliers and arrays. *IEEE Transactions on Computers*, (1):2513–2525, 2013.

[26] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, Feb 2010. ISSN 0018-9219. doi: 10.1109/JPROC.2009.2034764.

[27] R. P. Duarte and C. S. Bouganis. A unified framework for over-clocking linear projections on FPGAs under PVT varia-

tion. In *International Symposium on Applied Reconfigurable Computing*, pages 49–60. Springer, 2014.

[28] R. P. Duarte and C. S. Bouganis. Zero-latency datapath error correction framework for over-clocking DSP applications on FPGAs. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pages 1–7, Dec 2014. doi: 10.1109/ReConFig.2014.7032566.

[29] M. D. Ercegovac. On approximate arithmetic. In *Asilomar Conference on Signals, Systems and Computers*, pages 126–130, 2013. doi: 10.1109/ACSSC.2013.6810243.

[30] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24 (6):10–20, 2004. ISSN 0272-1732. doi: 10.1109/MM.2004.85.

[31] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.

[32] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. *SIGPLAN Not.*, 47(4):301–312, 2012. ISSN 0362-1340. doi: 10.1145/2248487.2151008.

[33] H. Esmaeilzadeh, A. Sampson, M. Ringenburg, L. Ceze, D. Grossman, and D. Burger. Addressing dark silicon challenges with disciplined approximate computing. In *Proc. ISCA*, pages 1–4, 2012.

[34] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger. Power challenges may end the multicore

era. *Commun. ACM*, 56(2):93–102, February 2013. ISSN 0001-0782.

[35] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. *Commun. ACM*, 58(1):105–115, 2014. ISSN 0001-0782.

[36] D. Esposito, D. De Caro, E. Napoli, N. Petra, and A. G. M. Strollo. On the use of approximate adders in carry-save multiplier-accumulators. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. doi: 10.1109/ISCAS.2017.8050437.

[37] D. Esposito, A. G. M. Strollo, and M. Alioto. Low-power approximate MAC unit. In *PRIME*, pages 81–84, 2017. doi: 10.1109/PRIME.2017.7974112.

[38] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra. Approximate multipliers based on new approximate compressors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–14, 2018. ISSN 1549-8328. doi: 10.1109/TCSI.2018.2839266.

[39] F. Fleuret. Deep learning – linear classifiers, perceptron. 2018. URL https://documents.epfl.ch/users/f/fl/fleuret/www/dlc/dlc-slides-3a-linear.pdf. [Online; accessed 19-July-2018].

[40] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3): 277–296, 1999.

[41] J. Gantz and D. Reinsel. Extracting value from chaos, 2011. URL http://

www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf. [Online; accessed 19-July-2018].

[42] M. Gao, Q. Wang, A. S. K. Nagendra, and G. Qu. A novel data format for approximate arithmetic computing. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 390–395, Jan 2017. doi: 10.1109/ASPDAC.2017.7858354.

[43] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 158–168, New York, NY, USA, 2006. ACM. ISBN 1-59593-543-6.

[44] S. Ghosh, S. Bhunia, and K. Roy. CRISTA: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation. *IEEE TCAD/ICAS*, 26 (11):1947–1956, 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2007.896305.

[45] A. Gorantla and D. P. Design of approximate compressors for multiplication. *J. Emerg. Technol. Comput. Syst.*, 13(3): 44:1–44:17, April 2017. ISSN 1550-4832.

[46] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathanand, and K. Roy. IMPACT: Imprecise adders for low-power approximate computing. In *ISLPED*, pages 409–414, 2011. ISBN 978-1-61284-660-6.

[47] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders.

*IEEE TCAD/ICAS*, 32:124–137, 2013. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2217962.

[48] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Test Symposium (ETS), 2013 18th IEEE European*, pages 1–6. IEEE, 2013.

[49] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *ETS*, pages 1–6, 2013. doi: 10.1109/ETS.2013.6569370.

[50] L. Hardesty. *Mimicking cells with transistors*, 2011. URL http://news.mit.edu/2011/analog-systems-biology-0928. [Online; accessed 19-July-2018].

[51] L. Hardesty. *Analog computing returns*, 2016. URL http://news.mit.edu/2016/analog-computing-organs-organisms-0620. [Online; accessed 19-July-2018].

[52] S. Hashemi, R. I. Bahar, and S. Reda. DRUM: A dynamic range unbiased multiplier for approximate applications. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 418–425, Nov 2015. doi: 10.1109/ICCAD.2015.7372600.

[53] T. Higgs. Energy efficient computing. In *Electronics & the Environment, Proceedings of the 2007 IEEE International Symposium on*, pages 210–215. IEEE, 2007.

[54] L. Holik, O. Lengal, A. Rogalewicz, L. Sekanina, Z. Vasicek, and pages=1–6 year=2016 T. Vojnar, booktitle=2nd Workshop on Approximate Computing (WAPCO 2016) HiPEAC.

Towards formal relaxed equivalence checking in approximate computing methodology.

[55] A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 2366–2369. IEEE, 2010.

[56] C. Y. Huang, Z. S. Yu, Y. C. Hu, T. C. Tsou, C. Y. Wang, and Y. C. Chen. Correctness analysis and power optimization for probabilistic Boolean circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(4):615–628, 2015.

[57] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *JACM*, 48 (2):274–296, 2001.

[58] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han. A comparative evaluation of approximate multipliers. In *International Symposium on NANOARCH*, pages 191–196, 2016. doi: 10.1145/2950067.2950068.

[59] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han. A review, classification, and comparative evaluation of approximate arithmetic circuits. *J. Emerg. Technol. Comput. Syst.*, 13(4): 60:1–60:34, August 2017. ISSN 1550-4832.

[60] J. M. Jou, S. R. Kuang, and R. D. Chen. Design of low-error fixed-width multipliers for DSP applications. *IEEE TCAS-II: Analog and Digital Signal Processing*, 46(6):836–842, 1999. ISSN 1057-7130. doi: 10.1109/82.769795.

[61] S. Kemp. 2018 global digital reports, 2018. URL https://wearesocial.com/blog/2018/01/

global-digital-report-2018. [Online; accessed 19-July-2018].

[62] L. B. Kish. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3-4):144–149, 2002.

[63] L. B. Kish. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3-4):144–149, 2002.

[64] H. J. Ko and S. F. Hsiao. Design and application of faithfully rounded and truncated multipliers with combined deletion, reduction, truncation, and rounding. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(5):304–308, May 2011. ISSN 1549-7747. doi: 10.1109/TCSII.2011.2148970.

[65] P. Kulkarni, P. Gupta, and M. D. Ercegovac. Trading accuracy for power in a multiplier architecture. volume 7, pages 490–501. American Scientific Publishers, 2011.

[66] K. Y. Kyaw, W. L. Goh, and K. S. Yeo. Low-power high-speed multiplier for error-tolerant application. In *EDSSC*, pages 1–4, 2010. doi: 10.1109/EDSSC.2010.5713751.

[67] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. Quantum computers. *Nature*, 464 (7285):45, 2010.

[68] C. Li, Y. Li, H. Jiang, W. Song, P. Lin, Z. Wang, J. J. Yang, Q. Xia, M. Hu, E. Montgomery, J. Zhang, N. Davila, C. E. Graves, Z. Li, J. P. Strachan, R. S. Williams, N. Ge, M. Barnell, and Q. Wu. Large memristor crossbars for analog computing. In *2018 IEEE International Symposium on*

*Circuits and Systems (ISCAS)*, pages 1–4, May 2018. doi: 10.1109/ISCAS.2018.8351877.

[69] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013. ISSN 0018-9340. doi: 10.1109/TC.2012.146.

[70] C. H. Lin and I. C. Lin. High accuracy approximate multiplier with error correction. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 33–38, Oct 2013. doi: 10.1109/ICCD.2013.6657022.

[71] C. H. Lin and I. C. Lin. High accuracy approximate multiplier with error correction. In *ICCD*, pages 33–38, 2013. doi: 10.1109/ICCD.2013.6657022.

[72] C. Liu, J. Han, and F. Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *DATE*, pages 1–4, 2014. doi: 10.7873/DATE.2014.108.

[73] Y. Liu, T. Zhang, and K. K. Parhi. Computation error analysis in digital signal processing systems with overscaled supply voltage. *IEEE Transactions on VLSI Systems*, 18(4): 517–526, 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2009.2012863.

[74] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57 (4):850–862, April 2010. ISSN 1549-8328. doi: 10.1109/TCSI.2009.2027626.

[75] M. Masadeh, O. Hasan, and S. Tahar. Comparative study of approximate multipliers. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 415–418. ACM, 2018.

[76] D. May and W. Stechele. Voltage over-scaling in sequential circuits for approximate computing. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, April 2016. doi: 10.1109/DTIS.2016.7483887.

[77] A. K. Mishra, R. Barik, and S. Paul. iACT: A software-hardware framework for understanding the scope of approximate computing. In *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[78] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):1–33, 2016. ISSN 0360-0300. doi: 10.1145/2893356.

[79] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *DATE*, pages 1–6, 2011. doi: 10.1109/DATE.2011.5763154.

[80] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, 64(4):984–994, April 2015. ISSN 0018-9340. doi: 10.1109/TC.2014.2308214.

[81] G. E. Moore. Cramming more components onto integrated circuits. Electronics 38 (8): 114–117, 1965.

[82] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. Evoapprox8b: Library of approximate adders and multipliers for

circuit design and benchmarking of approximation methods. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 258–261. European Design and Automation Association, 2017.

[83] K. Natori and N. Sano. Scaling limit of digital circuits due to thermal noise. *Journal of applied physics*, 83(10):5019–5024, 1998.

[84] M. Nazemi and M. Pedram. Deploying customized data representation and approximate computing in machine learning applications. *arXiv preprint arXiv:1806.00875*, 2018.

[85] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE*, pages 361:1–361:6, 2014. ISBN 978-3-9815370-2-4.

[86] M. Nicolaidis. Double-sampling design paradigm–A compendium of architectures. *IEEE Transactions on Device and Materials Reliability*, 15(1):10–23, March 2015. ISSN 1530-4388. doi: 10.1109/TDMR.2014.2388358.

[87] M. A. Nielsen and I. L. Chuang. Quantum computation and quantum information, 2002.

[88] K. Palem and A. Lingamneni. What to do about the end of Moore's law, probably! In *DAC Design Automation Conference 2012*, pages 924–929, June 2012. doi: 10.1145/2228360.2228525.

[89] A. Paler, A. Alaghi, I. Polian, and J. P. Hayes. Tomographic testing and validation of probabilistic circuits. In *2011 Sixteenth IEEE European Test Symposium*, pages 63–68, May 2011. doi: 10.1109/ETS.2011.43.

[90] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo. Truncated binary multipliers with variable correction and minimum mean square error. *IEEE TCAS-I: Regular Papers*, 57(6):1312–1325, 2010. ISSN 1549-8328. doi: 10.1109/TCSI.2009.2033536.

[91] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[92] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, and A. Yakovlev. Energy-efficient approximate Wallace-tree multiplier using significance-driven logic compression. In *IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 1–6, 2017. doi: 10.1109/SiPS.2017.8109990.

[93] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev. Energy-efficient approximate multiplier design using bit significance-driven logic compression. In *DATE*, pages 7–12, 2017. doi: 10.23919/DATE.2017.7926950.

[94] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, and A. Yakovlev. Significance-driven logic compression for energy-efficient multiplier design. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pages 1–1, 2018. ISSN 2156-3357. doi: 10.1109/JETCAS.2018.2846410.

[95] J. Rabaey. *Low power design essentials*. Springer Science & Business Media, 2009.

[96] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys. Pushing the limits of voltage over-scaling for error-resilient applications. In *Design, Automation Test in Europe Conference*

*Exhibition (DATE), 2017*, pages 476–481, March 2017. doi: 10.23919/DATE.2017.7927036.

[97] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. ASLAN: Synthesis of approximate sequential circuits. In *DATE*, pages 1–6, 2014. doi: 10.7873/DATE.2014.377.

[98] D. Reinsel, J. Gantz, and J. Rydning. Data Age 2025: The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on Data That's Big. *IDC, Seagate, April*, 2017. URL https://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf. [Online; accessed 19-July-2018].

[99] N. B. Rizvandi, J. Taheri, and A. y. Zomaya. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *Journal of Parallel and Distributed Computing*, 71(8):1154–1164, 2011.

[100] A. Sampson. *Hardware and software for approximate computing*. PhD thesis, 2015. URL https://www.cs.cornell.edu/~asampson/media/dissertation.pdf. [Online; accessed 19-July-2018].

[101] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI*, pages 164–174, 2011. ISBN 978-1-4503-0663-8. doi: 10.1145/1993498.1993518.

[102] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy. Energy-efficient neural computing with approxi-

mate multipliers. *J. Emerg. Technol. Comput. Syst.*, 14(2): 16:1–16:23, July 2018. ISSN 1550-4832.

[103] R. R. Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[104] 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*, pages 61–69. IEEE, 1999.

[105] M. Schulz. The end of the road for silicon? *Nature*, 399 (6738):729, 1999.

[106] L. Sekanina. Introduction to approximate computing: Embedded tutorial. In *DDECS*, pages 1–6, 2016. doi: 10.1109/DDECS.2016.7482460.

[107] L. Sekanina. Introduction to approximate computing: Embedded tutorial. In *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 1–6, April 2016. doi: 10.1109/DDECS.2016.7482460.

[108] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The EDA challenges in the Dark Silicon era: Temperature, reliability, and variability perspectives. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 185:1–185:6, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2730-5.

[109] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. Cross-layer approximate computing: From logic to architectures. In *DAC*, pages 1–6, 2016.

[110] J. M. Shalf and R. Leland. Computing beyond Moore's law. *Computer*, 48(12):14–23, Dec 2015. ISSN 0018-9162. doi: 10.1109/MC.2015.374.

[111] M. Sharad, D. Fan, K. Aitken, and K. Roy. Energy-efficient non-Boolean computing with spin neurons and resistive memory. *IEEE Transactions on Nanotechnology*, 13(1):23–34, Jan 2014. ISSN 1536-125X. doi: 10.1109/TNANO.2013.2286424.

[112] K. L. Shepard and V. Narayanan. Conquering noise in deep-submicron digital ICs. *IEEE Design & Test of Computers*, (1):51–62, 1998.

[113] K. Shi, D. Boland, E. Stott, S. Bayliss, and G. A. Constantinides. Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. In *DAC*, pages 1–6, 2014. doi: 10.1145/2593069.2593118.

[114] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[115] H. Sim and J. Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. doi: 10.1145/3061639.3062290.

[116] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler. BDD minimization for approximate computing. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 474–479. IEEE, 2016.

[117] C. Solomon and T. Breckon. Fundamentals of digital image processing : a practical approach with examples in Matlab. chapter 4, pages 95–96. Wiley-Blackwell, 2011. ISBN 978-0-470-84472-4.

[118] J. E. Stine and O. M. Duverne. Variations on truncated multiplication. In *DSD*, pages 112–119, 2003. doi: 10.1109/DSD.2003.1231908.

[119] E. E. Swartzlander. Truncated multiplication with approximate rounding. In *Proc. of 33rd Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1480–1483, 1999. doi: 10.1109/ACSSC.1999.831996.

[120] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In *DATE*, pages 1051–1056, 2018. doi: 10.23919/DATE.2018.8342167.

[121] W. J. Townsend, E.E. Swartzlander, and J. A. Abraham. A comparison of Dadda and Wallace multiplier delays. In *Advanced signal processing algorithms, architectures, and implementations XIII*, volume 5205, pages 552–561. International Society for Optics and Photonics, 2003.

[122] S. Ullah, S. S. Murthy, and A. Kumar. Smapproxlib: library of fpga-based approximate multipliers. In *Proceedings of the 55th Annual Design Automation Conference*, page 157. ACM, 2018.

[123] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar. Area-optimized low-latency approximate multipliers for FPGA-based hardware

accelerators. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 159:1–159:6, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5700-5.

[124] Z. Vasicek and L. Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *2014 IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 135–140. IEEE, 2014.

[125] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: Systematic logic synthesis of approximate circuits. In *DAC*, pages 796–801, 2012. doi: 10.1145/2228360.2228504.

[126] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *DATE*, pages 1367–1372, 2013. ISBN 978-1-4503-2153-2.

[127] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *ICCAD*, pages 667–673, 2011. doi: 10.1109/ICCAD.2011.6105401.

[128] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, 1964. ISSN 0367-7508. doi: 10.1109/PGEC.1964.263830.

[129] D. H. Woo and H. H. S. Lee. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41(12):24–31, Dec 2008. ISSN 0018-9162. doi: 10.1109/MC.2008.494.

[130] W. C. Yeh and C. W. Jen. High-speed Booth encoded parallel multiplier design. *IEEE Transactions on Computers*, 49(7): 692–701, Jul 2000. ISSN 0018-9340. doi: 10.1109/12.863039.

[131] P. Yin, C. Wang, W. Liu, E. E. Swartzlander, and F. Lombardi. Designs of approximate floating-point multipliers with variable accuracy for error-tolerant applications. *Journal of Signal Processing Systems*, 90(4):641–654, 2018.

[132] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram. RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Transactions on VLSI Systems*, 25(2): 393–401, 2017. ISSN 1063-8210. doi: 10.1109/TVLSI.2016.2587696.

[133] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram. RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):393–401, Feb 2017. ISSN 1063-8210. doi: 10.1109/TVLSI.2016.2587696.

[134] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi. Design-efficient approximate multiplication circuits through partial product perforation. *IEEE Transactions VLSI Systems*, 24(10):3105–3117, 2016. ISSN 1063-8210. doi: 10.1109/TVLSI.2016.2535398.